Self-Organizing Feature Maps for Exploratory Data Analysis and Data Mining: A Practical Perspective

by

Willem S. van Heerden

Submitted in partial fulfillment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

April 2017

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Publication data:

Willem S. van Heerden. Self-Organizing Feature Maps for Exploratory Data Analysis and Data Mining: A Practical Perspective. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, April 2017.

An electronic, hyperlinked version of this dissertation is available as an Adobe® PDF document:

http://repository.up.ac.za/

Copyright © 2017 University of Pretoria. All rights reserved. The copyright in this work vests in the University of Pretoria. No part of this work may be reproduced or transmitted in any form or by any means, without the prior written permission of the University of Pretoria.

Self-Organizing Feature Maps for Exploratory Data Analysis and Data Mining: A Practical Perspective

by

Willem S. van Heerden
E-mail: wvheerden@cs.up.ac.za

Abstract

The self-organizing feature map (SOM) is an unsupervised machine learning approach that offers extremely useful data modeling and visualization abilities. The approach has been successfully employed in order to solve a wide variety of problems. Exploratory data analysis (EDA) and data mining (DM) are both fields that attempt to algorithmically extract insightful knowledge from a data set, with a greater or lesser level of human assistance. This work overviews the SOM algorithm, with a focus on EDA and DM applications. Supervised and unsupervised neuron labeling methods for SOMs, which are an integral part of most SOM-based EDA and DM exercises, are investigated. Existing SOM-based EDA and DM approaches are also critically compared. A novel unsupervised neuron labeling approach, namely unsupervised weight-based cluster labeling, is introduced. This research also proposes HybridSOM, a novel hybrid framework for DM-oriented rule extraction, which combines a SOM with an arbitrary rule extraction algorithm. Two empirical investigations are reported. Firstly, existing supervised neuron labeling techniques are experimentally compared. Secondly, the HybridSOM framework is empirically compared to several existing DM rule extractors.

Keywords: Artificial intelligence, artificial neural networks, computational intelligence, data mining, data visualization, emergent systems, exploratory data analysis, hybrid models, rule extraction, self-organizing feature maps.

Supervisor : Professor Andries P. Engelbrecht
Department : Department of Computer Science
Degree : Master of Science

"This is indeed a mystery," I remarked. "What do you imagine that it means?"

"I have no data yet. It is a capital mistake to theorise before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts . . . "

A Scandal in Bohemia, by Sir Arthur Conan Doyle (1892)

"My dear Watson, try a little analysis yourself," said he, with a touch of impatience. "You know my methods. Apply them, and it will be instructive to compare results."

The Sign of Four, by Sir Arthur Conan Doyle (1890)

Acknowledgements

I would like to express my sincere gratitude to the following people and groups for their assistance, direct or indirect, during the production of this research work:

- Professor Andries P. Engelbrecht, my supervisor, for his insight and motivation;
- My mother, late father, sister and brother, for their support and motivation;
- My examiners, Professor Barbara Hammer and Doctor Mardé Helbig, whose comments have greatly contributed towards the quality of the final manuscript;
- The colleagues I have shared a department and research group with, many of whom offered advice or support in one form or another;
- My many friends, for helping to keep me motivated and in good spirits through the frustrating and demotivating times that I experienced;
- The TechTeam of the Computer Science Department at the University of Pretoria, for maintaining the computer infrastructure used during this research;
- The staff of the Department of Library Services for maintaining the library and electronic reference infrastructure used during the literature survey;
- The developers, maintainers, and user communities of the various open-source and freeware tools that were used during the production of this dissertation;
- The National Research Foundation (NRF) and the Department of Labour for the financial assistance, in the form of a Scarce Skills Scholarship, that they have provided during the course of this research.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

> *Worldwide, trillions of information transactions take place every day. Banks move their clients' funds around. Retail outlets buy, sell and re-sell thousands of items. Hospitals receive pathologists' reports on many hundreds of patients. The figures of the stock exchange fluctuate on an hourly basis, reflecting the rising and falling fortunes of hundreds of companies. Genetic research produces countless gene sequences and experimental results in a search for effective therapies. This "information age" has given us vast stores of data that record reality, and hide patterns and interconnections more complex than anyone can imagine. What mysteries might these vast stores of data hold? What great rewards might we glean from exploring them?*

The field of artificial intelligence (AI) [45, 177] has become quite active in recent years, and attempts to model "intelligent" behavior algorithmically. More specifically, computational intelligence (CI) [68], a sub-discipline of AI, attempts to develop such systems that are capable of adaptive behavior within changing environments [60, 64]. The most notable algorithms have been based on a variety of natural systems, ranging from the human neurological and immune systems, to ant colonies and bird flocks.

While many AI and CI approaches have become highly refined, they have traditionally been used to solve theoretical engineering and scientific problems. Most of these problems have been strictly laboratory-based, with relatively few practical benefits. As such, AI has developed a reputation for being a somewhat abstract discipline.

In the past, computer science research focused on the effective storage and processing of data. In recent years, however, the emphasis has largely shifted towards the useful *exploitation* of these stored resources. The fields of exploratory data analysis (EDA) and data mining (DM) attempt to derive useful meaning from data stores, and encouraging results have been shown [8, 9, 29]. More recently, EDA and DM have achieved their aim by adapting and applying AI and CI paradigms within a *practical realm*. These advances promise to offer great value to businesses and other organizations.

The rest of this introduction is organized as follows: Section 1.1 elaborates on the motivation for this work. The objectives that this work set out to achieve are listed under Section 1.2, while Section 1.3 outlines the broad methodology used in order to realize these objectives. Section 1.4 lists the novel contributions made by this dissertation. Finally, an outline for the remainder of the report is given in Section 1.5.

1.1 Motivation

The self-organizing feature map (SOM) [142, 146] is a CI approach developed by Tuevo Kohonen at the Helsinki University of Technology (now part of Aalto University). Due to an extensive body of work, SOMs are now well understood. An extensive bibliography of SOM research between 1981 and 2005 has been published [130, 178, 184][1].

SOMs offer a number of very useful advantages to data clustering and analysis, particularly in terms of their powerful visualization abilities. Practical applications have been demonstrated in areas as diverse as engineering [150], medicine [234], and financial analysis [52]. Several SOM-based tools, such as Viscovery SOMine [51], SOM_PAK [148] and SOM Toolbox [147, 248, 254, 258], have also been developed and put to effective use. These factors all point towards the rise of the SOM as a mature technology.

In light of the above discussion, it is clear that SOMs may be very usefully employed for EDA or DM purposes. While a great deal of work has been done in the field of SOM-based EDA, no summarizing survey of existing EDA techniques has been published. In addition, the literature has focused very little attention on DM based on SOMs.

[1] A consolidated BibTeX database of this bibliography is available from the Adaptive Informatics Research Centre within Aalto University, at `http://www.cis.hut.fi/research/som-bibl/`.

Furthermore, neuron labeling methods are often used during SOM-based EDA, and are required by all SOM-based DM techniques. Several labeling methods have been proposed, but no critical discussions or relative performance analyses exist.

The primary motivation behind this dissertation is thus to provide an overview of existing SOM-based EDA techniques, and to investigate SOM-based DM in greater detail. The secondary motivation is to investigate SOM-based neuron labeling more thoroughly.

1.2 Objectives

The research objectives of this dissertation are summarized as follows:

- To contextualize the use of the SOM algorithm within the realm of EDA and DM, in a generic practical setting within the real world.
- To describe the most common types of SOM-based data visualization techniques, and the main classes of EDA that are possible using SOMs.
- To investigate techniques for labeling SOM neurons, because neuron labeling is an important component of many SOM-based EDA and DM methods.
- To investigate SOM-based DM approaches that have been described in the literature, and investigate novel approaches to DM using the SOM as basis.

1.3 Methodology

The methodology used by this dissertation incorporates the following components:

- A detailed literature survey related to SOMs, EDA and DM techniques that are based on SOMs, and SOM-based neuron labeling.
- Prototypes and algorithms of the novel SOM-based neuron labeling and DM techniques that are proposed within this dissertation.
- Experimental investigations into the performance characteristics associated with the existing and proposed SOM-based neuron labeling and DM techniques.

1.4 Contributions

This research makes several novel contributions to knowledge in the SOM and DM domains. These contributions are summarized below:

- A broad taxonomy, which describes the general types of EDA that are possible when using a trained SOM as a basis for the analysis task, is outlined.
- A survey of existing neuron labeling techniques for trained SOMs is presented. The approaches are critically discussed and compared to one another.
- A novel neuron labeling algorithm, named unsupervised weight-based cluster labeling, is described and discussed critically in relation to existing labeling approaches.
- A detailed survey, which describes DM methodologies that are based upon SOMs and have previously been described in the literature, is presented.
- A novel framework, named HybridSOM, is proposed for SOM-based rule extraction in a DM context. The framework is described and critically discussed.
- The performance characteristics of several existing supervised neuron labeling methods are experimentally compared to one another.
- Several general conclusions and recommendations, related to the advantages and drawbacks linked to each supervised and unsupervised neuron labeling method, and how appropriate each approach is within a practical setting, are presented.
- The performance of HybridSOM is empirically compared to several classical DM methods, as well as one established SOM-based DM algorithm known as SIG*.
- Conclusions that elucidate the advantages and drawbacks that are associated with the various SOM-based DM approaches are presented.
- A number of observations, related to the overall feasibility of SOM-based rule extraction approaches, are presented and discussed. These observations lead to conclusions on the types of settings within which SOM-based DM should be used, and conversely, when alternatives to SOM-based methods should be favored.

1.5 Dissertation Outline

The list below presents the organization of the remaining chapters making up this dissertation, as well as a brief description of the topics that each deals with:

- **Chapter 2** focuses on the definition and delineation of EDA and DM. Several important concepts directly related to these fields are also defined and discussed.
- **Chapter 3** covers the theoretical background relating to the primary focus of this work, namely the SOM. The general operation of the learning algorithm, specific implementation details, and variations on the algorithm are all discussed.
- **Chapter 4** broadly outlines the most commonly used SOM-based visualization techniques, which indirectly visualize the underlying data. The broad categories of SOM-based EDA, which use such map visualizations, are also taxonomized.
- **Chapter 5** describes a variety of techniques that exist for discovering structures of emergent neuron clusters within trained SOMs. The described techniques are critically discussed in the context of practical EDA and DM applications.
- **Chapter 6** discusses methods for labeling map neurons, which is often an important component for both SOM-based EDA and DM. Techniques are taxonomized, novel approaches are suggested, and the methods are critically discussed.
- **Chapter 7** focuses on DM techniques. Existing techniques are identified, and a new hybrid approach is described. All these approaches are described and theoretically compared, in order to highlight potential flaws in their respective designs.
- **Chapter 8** presents the results and discusses the conclusions of two experimental comparisons. Firstly, the performance characteristics of unsupervised neuron labeling methods are analyzed. Secondly, the performance characteristics of SOM-based DM algorithms and several classical DM algorithms are compared.
- **Chapter 9** gives a summary of the general conclusions that this dissertation work has arrived at. The chapter also provides details relating to several potential future research avenues emanating from the work presented in this dissertation.

All references cited within this dissertation are listed in a bibliography on pages 309 to 345. References are ordered alphabetically, according to the family names of authors. Publication information is as complete as possible at the time of writing.

A digital object identifier (DOI)[2] is provided whenever one is available, both for scholarly publications[3] and archives of digitally published experimental data sets[4]. The identifiers in the PDF version of this document link to the digital resources.

Publications that are maintained only in the ArXiv repository[5] at the time of publication are presented with all available information (this is typically limited to the relevant author information, title of the document, and date of publication). For each such reference, the ArXiv identifier and category[6] for the document are provided.

A reference to a non-permanent uniform resource locator (URL) is only provided when a document's primary publication method is via that URL (for example, in the case of an electronic journal, or a manual published online). The full details of a physical publication of a source are also provided, should such a document be available. The URL text in the PDF version of this dissertation also links to the source in question.

In a few cases, the seminal publications in which concepts were introduced are unavailable. In such instances the original reference is provided for the sake of completeness, with a note citing a reliable source that is available and quotes the original publication.

---

[2] A DOI is a unique alphanumeric string, which provides a permanent link to a digital object, even if the location of the object, such as a URL, changes. A DOI takes the form of `doi:10.1234/5678`, which is resolved manually at `http://www.doi.org/`, or directly via `http://doi.org/10.1234/5678`. Additional information that is relevant to the DOI standard is available at `http://www.doi.org/`.

[3] The CrossRef association provides DOIs for the electronic documents of participating publishers and organizations. Further information on CrossRef is available at `http://www.crossref.org/`.

[4] The DataCite organization allows participating organizations to assign DOIs to electronic data sets available via the Internet. Further information on DataCite is provided at `http://www.datacite.org/`.

[5] ArXiv is an electronic repository maintained by the Cornell University Library. The repository makes electronic preprints of academic papers available for free access (these documents are referred to as e-prints). Additional information related to the ArXiv service is available at `http://arxiv.org/`.

[6] An ArXiv document reference usually takes the form of `1206.12345v1 [math.FA]`, where the identifier of the document is `1206.12345v1` and the category of the document is `math.FA`. Identifiers are resolved manually at `http://arxiv.org/`, or directly via `https://arxiv.org/abs/1212.12345v1`.

The following appendices are included after the bibliography. They contain discussions of background details related to the main text, as well as quick reference lists:

- **Appendix A** describes a rule induction algorithm, namely Clark and Niblett's CN2. This algorithm was used in the experiments reported in Chapter 8.
- **Appendix B** outlines the C4.5 algorithm, by J. Ross Quinlan, which extracts rules by building decision trees. Like CN2, the C4.5 algorithm was used experimentally.
- **Appendix C** lists the publications derived from this work, including full bibliographic information and a short description of the contributions made by each.
- **Appendix D** lists and defines the mathematical symbols used in this work, categorized according to the relevant chapter in which they first appear.
- **Appendix E** provides a list of important acronyms and associated definitions, both for existing terms and new concepts defined through the course of this work.

Chapter 2

Foundations of Exploratory Data Analysis and Data Mining

This dissertation focuses on the application of SOMs to the fields of EDA and DM. Before attention can be given to the various concrete applications, the scope of these fields must be defined. Both EDA and DM are defined in terms of several concepts.

Firstly, the concepts of data and knowledge are defined under Section 2.1. Knowledge extraction concepts are discussed in Section 2.2. Knowledge discovery in databases is outlined in Section 2.3. Section 2.4 covers data pre-processing operations important for SOMs. Section 2.5 differentiates EDA and DM. A summary is given in Section 2.6.

For the purpose of illustration, a single example is used throughout this discussion: a store of recorded statistics relating to the characteristics and the business performance of various companies. Of course, the concepts discussed in this chapter are broadly applicable to any other field, whether the focus is commercial, scientific, or otherwise.

2.1 Data versus Knowledge

The terms *data* and *knowledge* are fairly poorly defined and thus often misunderstood. To make matters worse, several conflicting definitions and perspectives have been proposed by various authors. Sections 2.1.1 and 2.1.2 present the viewpoint on the nature of data and knowledge, respectively, that has been taken by this dissertation.

2.1.1 Data

An *environment* denotes a system (i.e., the real world, or an aspect of the real world). At a moment in time, t, an environment's *state* is denoted by $\hat{S}_t$. State encapsulates an environment's *objects* and their associated *properties* [112]. Objects are observations that may represent actual concrete entities, or entail more abstract concepts.

Data is thus defined, in this work's context, as: *a representation of the state of an environment*. The exact nature of the representation remains unspecified at this point.

2.1.1.1 Categories of Data

Data must take some form. However, the term is used to refer to any of a fairly broad range of artifacts. Nevertheless, two general categories of data may be defined [92]:

- *Unstructured* and *semi-structured data*, which may take the form of raw text, image, audio, or video in any physical or machine-readable storage format. Such data is contained within documents, with either no structure at all (unstructured) or an arbitrary structure that varies between documents (semi-structured).
- *Structured data*, which is data that has been organized according to a data model (e.g., relational or object-oriented) so as to give it a well-defined structure. Such data encoded into a machine-readable form is called a *data set* (interchangeably, a *database*). In some other contexts, structured data is termed *information*.

In the current context, only structured, text-based data will be considered. The inherent complexities involved in the processing of unstructured data and multimedia elements are beyond the scope of the research presented in this dissertation.

2.1.1.2 Structured Data Representation

A wide variety of different storage and representation paradigms for structured data have become available. These range from simple flat text file representations to more complex database management systems (DBMSs) such as object-oriented DBMSs, object-relational DBMSs, or even data warehouses. The focus of this research is not on these systems, but the reader is referred to the literature [67, 199, 221, 236] for more details.

Most commercially available DBMSs, however, are based on a *relational data model*, as described by Elmasri [67]. A *relational database* is represented by several inter-related tables consisting of rows and columns. DM and EDA applications generally use a simplification of the relational model, called a *table-based data model* [255], which is assumed for the remainder of this dissertation. In this model, the database consists of a single table, thus providing a more readily usable (but possibly less efficient) interface. A table-based model can be applied to all database schemata and queries, as long as *null* values are allowed [236] (*null* values are discussed in more detail below). Figure 2.1 shows a hypothetical example of a relational DBMS using the table-based model.

Both the relational and table-based data models are based on the concept of a *relation schema*, which is denoted by $H(A_1, A_2, \ldots, A_{\tilde{n}})$. A relation schema consists of the *name* of the relation, denoted H, and a set of *attributes*, denoted $\mathcal{A} = A_1, A_2, \ldots, A_{\tilde{n}}$. The *degree* of the relation is denoted by $\tilde{n}$. Each attribute A_l also has a name and is associated with a *domain*, denoted $dom(A_l)$. A domain describes a set of valid atomic values for an attribute in terms of the *data type* and the *format* that the data may take. In summary, a relation (consisting of attributes and associated domains) describes a template for the form that objects with certain properties can take within an environment.

The data type of a domain is either *continuous* or *nominal*. Continuous attribute values fall within a continuous range, and are usually represented as floating-point numeric values. Nominal values are chosen from a discrete set of values. Integers are also nominal values, and integer encoding is often used to represent nominal values. Boolean attributes have a special nominal data type, with only a `true` or `false` value.

Associated with a relational schema is a *relation*, which consists of a set of $\tilde{m}$ *tuples*, denoted $\{tup_1, tup_2, \ldots, tup_{\tilde{m}}\}$. Each tuple, denoted $tup_{\bar{t}}$, is an ordered list of $\tilde{n}$ *values*, such that $tup_{\bar{t}} = \langle f_1, f_2, \ldots, f_{\tilde{n}} \rangle$. Each value, f_l, is an element that must either be selected from $dom(A_l)$, or have a *null* value that indicates the value is unknown (or a *missing value*). A tuple is therefore effectively a representation of a single object, obj, within an environment, while the tuple's values represent the properties of obj.

In Figure 2.1, `company` is the name of the relation, while the column headings (i.e., `name`, `turnover`, `avg_remun`, `ISO9000`, `size` and `status`) represent the relation's attributes. The degree of the relation is 6 (thus, $\tilde{n} = 6$). The domains of `turnover` and

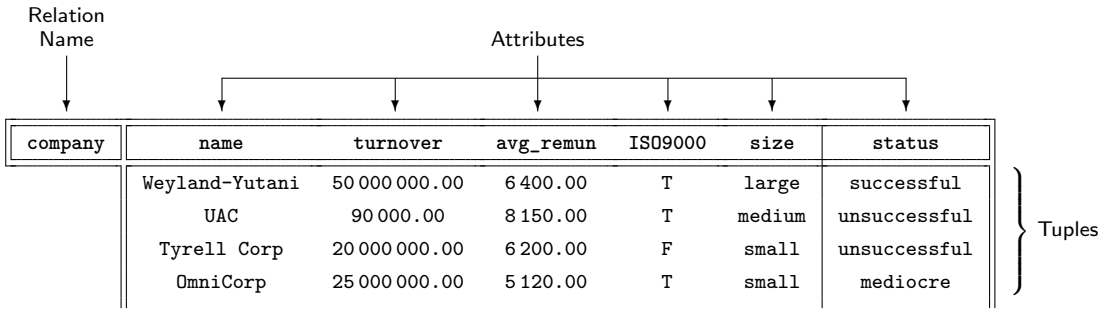


Figure 2.1: A hypothetical relational database representing company details.

`avg_remun` are continuous values between 0 and some maximum value defined for each domain. The domain of `size` is the nominal value set $\{\texttt{small}, \texttt{medium}, \texttt{large}\}$, while for `ISO9000` the domain is the Boolean set $\{\texttt{T}, \texttt{F}\}$. The example also highlights four tuples. Each row below the column headings is a single tuple, made up of values associated with the attribute of the column in which they appear. Finally, the example illustrates that the values are chosen from the domains associated with each attribute.

2.1.2 Knowledge

In light of the above discussion, this dissertation defines knowledge as: *some form of insight into an environment that is represented by a set of data*. Typically, when given only a raw data set, this insight is not obvious to the casual human observer.

2.1.2.1 Types of Knowledge

While the insight that knowledge represents can take many forms, and often has many representations, the following two broad divisions are identifiable:

- *Logical consequences* of a data set. These are based on deductions from the contents of the data set through the application of formal logic.
- *Generalizations* that exploit identifiable *regularities* within a data set. These types of generalizations are represented by means of a *model*.

This dissertation focuses on the latter type of knowledge. In fact, many basic logical consequences arguably represent a weaker form of insight than generalizations do.

2.1.2.2 Models

A model, in the context of this dissertation, is a *generalization of a data-represented environment*. The discussion presented within this section is primarily based on the work of Holsheimer *et al* [112]. The relevant concepts are illustrated by Figure 2.2, which shows how a model of an environment relates to the environment itself.

The definition of data implies that a dynamic environment, with a time-variant state, is possible. Thus an environment can be viewed as a state transition system, where $\mathcal{V}$ is the set of all possible states. Given any state, $\hat{S}_t$, a *state transition function*, $\mathcal{F} : \mathcal{V} \rightarrow \mathcal{V}$, defines the following state $\hat{S}_{t+1}$. State $\hat{S}_{t+1}$ could represent a different number of objects, each with different properties and interrelationships, than those of $\hat{S}_t$.

A model contains a set of *classes*, denoted $\mathcal{C}$, which are aggregations of objects. Each class, C_m, has a *class condition*, denoted B_m, that defines object membership to a class. In the context of models, objects are often referred to as *examples*. The class condition separates *positive examples* of C_m (objects that satisfy class condition B_m, and are thus members of class C_m) from *negative examples* of C_m (objects that do not satisfy B_m). The nature of the class condition is discussed in relation to rules, below.

Essentially, a model is characterized by two functions, namely a *classification function* and a *model transition function*. A classification function, denoted $\mathcal{P} : \hat{S}_t \rightarrow \mathcal{C}$, maps each object, *obj*, in state $\hat{S}_t$ to a corresponding class, $C_m \in \mathcal{C}$, only if *obj* satisfies an appropriate class description, D_m. The form of D_m is discussed in terms of rules, below. The classification function often maps several objects to one class. A model transition function, $\mathcal{F}'$, models $\mathcal{F}$ in dynamic environments, by representing how the classes of an environmental state, $\hat{S}_t$, transition to classes of the subsequent state, $\hat{S}_{t+1}$.

A model is considered correct if, at any time t, the representation of the next state is identical to the representation predicted by the model. This is expressed as:

$$\mathcal{P}\big(\mathcal{F}(\hat{S}_t)\big) = \mathcal{F}'\big(\mathcal{P}(\hat{S}_t)\big)$$

This equation illustrates that if a model of an environment is correct, the representation that is shown in Figure 2.2 must also be commutative. While SOMs have been used for temporal data analysis [32, 263, 96, 98, 140, 144, 152, 230], this work focuses on static environments and only briefly considers state and model transitions in Section 4.5.5.

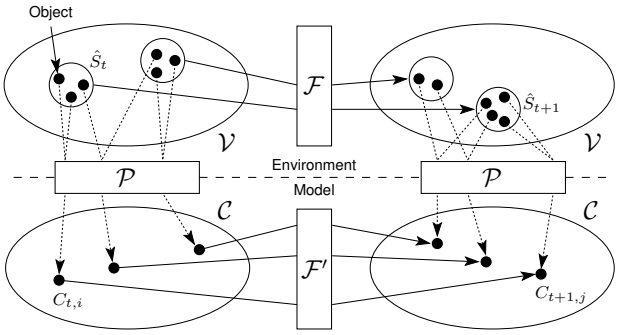


Figure 2.2: The relationship between an environment and a model that represents this environment. Figure adapted from the work of Holsheimer *et al* [112].

2.1.2.3 Rules

The term *symbolic knowledge* refers to knowledge that is represented in a formal way, such that an interpreter with the necessary competence is able to utilize it [237]. In this dissertation's context, the knowledge being represented is embodied in a model. For symbolic knowledge to exist, a knowledge representation scheme is required. In the most general sense, any such scheme takes the form of a set of *rules* [54, 112].

To define a rule, a data set is assumed, with a set, $\mathcal{A}$, of associated attributes. A subset of attributes, here denoted $\mathcal{A}_{cls}$, is called the *classification attribute* set. All class conditions are expressed in terms of value ranges for one or more classification attributes, using equalities (that is, $=$) or inequalities ($\neq$, $>$, $\geq$, $<$, and $\leq$). In Figure 2.1, {`status`} is the set of classification attributes, and `status` = `solvent` is a class condition.

The remaining attributes of $\mathcal{A}$, that are not in $\mathcal{A}_{cls}$, are in the set of *descriptive attributes*, denoted $\mathcal{A}_{des}$. The class description, D_m, is expressed as conditions on the values for one or more of these attributes, again using equalities or inequalities. In Figure 2.1, the set of descriptive attributes is {`company`, `turnover`, `avg_remun`, `ISO9000`, `size`}. Examples of conditions on these are `turnover` $> 50,000$ and `size` = `large`.

A rule relates a class description to the class associated with a class condition. If an example from the data set satisfies the description of a rule, it is said to be *covered* by that rule. This means that the rule predicts that the example belongs to the class denoted by the class condition. Thus a rule predicts class membership (and consequently the values of the classification attributes) based on the values of descriptive attributes.

2.1.2.4 Knowledge Representation Techniques

The rules discussed above can be expressed in a variety of ways. The simplest approach is to use informal natural language to describe either class characteristics, or correlations between the values of descriptive attributes and classification attributes.

Propositional logic structures provide a formal representation. An overview of propositional logic is given by Huth and Ryan [118]. Using propositional logic, class descriptions are represented as formulas that relate attribute value conditions to one another using the standard logical connectives $\wedge$ and $\vee$. A rule is then an implication between a class description and a class symbol that is associated with a class condition.

Research has identified a number of other approaches, collectively known as *knowledge representation techniques* [36, 205]. While exotic methods such as semantic networks and frames exist, this dissertation only focuses on the following two approaches:

- *Production rules* represent rule sets of if-then rules, each with an *antecedent* and a *consequent*. The antecedent is usually a conjunctive normal form (CNF) propositional expression, which is a single term or a conjunction of terms, and each term is an attribute value condition or a disjunction of conditions. The consequent is a class condition. The antecedent and consequent are related as follows:

  IF *antecedent* **THEN** *consequent*

  New examples are compared to each rule in order, receiving the classification of the first matching antecedent's consequent. Usually, one default rule exists, with a consequent that classifies any examples matching none of the previous rules.

- *Decision trees*, which are tree-like representations of rule sets. Each non-leaf node represents an attribute test checking equality or inequality. Node branches correspond to the possible outcomes of a test. Leaf nodes represent class conditions. New examples are classified by starting at the root node and performing node tests in sequence, following the appropriate outcome branch for each test, and stopping once a leaf node provides a specific classification for the example.

Figure 2.3 shows a production rule set, while Figure 2.4 illustrates a decision tree, both of which represent one possible model of the example data set shown in Figure 2.1.

```
IF size = small AND ISO9000 = true THEN status = acceptable
IF size = medium AND turnover >= 100 000 THEN status = successful
IF size = large THEN status = successful
DEFAULT status = unsuccessful
```

Figure 2.3: An example set of production rules, based on the data set of Figure 2.1. The rule antecedents are all in CNF, and a default rule is included at the end of the rule set.

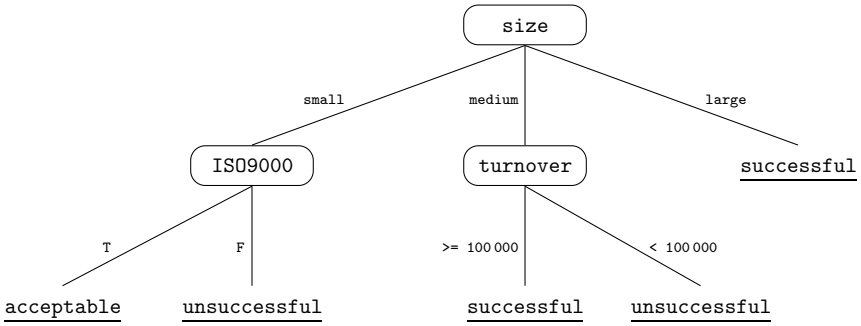


Figure 2.4: An example of a decision tree, equivalent to the production rules in Figure 2.3. Rounded rectangles are attribute tests, and underlined labels are data example classifications.

2.2 Knowledge Extraction

The nature of knowledge has been thoroughly discussed in the previous section. The process by which knowledge is created has, however, been ignored until this point. This section focuses on the broader aspects of this knowledge extraction process.

In a general sense, *learning* is simply the process by which knowledge is derived [56]. Within the context of DM and EDA tasks, the learning process must extract knowledge from a source of data. Section 2.2.1 discusses general learning approaches, while Section 2.2.2 describes learning that is specifically machine-based in nature.

2.2.1 Learning Approaches

Two types of knowledge were described in Section 2.1.2.1, namely logical consequences and generalizations. Two corresponding categories of learning approaches are conse-

quently identifiable. Each of the general learning approaches is defined according to the class of knowledge that is produced during the knowledge creation process, as follows:

- *Deductive learning* derives knowledge based on logical consequences. This uses a variety of technologies, such as Structured Query Language (SQL) [67, 199, 221], Data Mining Query Language (DMQL) [102, 103], deductive databases [46], online analytical processing (OLAP) [103, 220] or data cubes [95, 103].

- *Inductive learning* [111, 112] attempts to identify generalizations applicable to an environment. This is done by building a model based on environmental observations, and translating it into symbolic knowledge using an appropriate knowledge representation technique, as discussed in the previous section.

Because the focus of this research is on knowledge embodied in the form of generalizations, inductive learning is assumed for the remainder of this dissertation. Inductive learning aims to build a model by performing two general tasks [112]:

- Firstly, an appropriate internal representation for the data set must be constructed. This representation takes the form of classes and descriptions for these classes.

- Secondly, in dynamic environments, an appropriate state transition function must be represented. This is achieved by building a model transition function.

As previously mentioned, this research generally considers only static environments. As a consequence, the first task of inductive learning is primarily focused upon, while the second task is largely ignored for the remainder of this dissertation.

2.2.2 Machine Learning

The field of *machine learning* encompasses automated, algorithmic approaches to the inductive learning process [56, 63]. This means that the environmental observations on which the built model is based should be encoded in some sort of suitable machine-readable format. Typically, the machine-readable environmental observations are represented using some sort of database, as described in Section 2.1.1.2. Some form of learning algorithm must also be employed, in order to extract usable knowledge.

Obviously, different means by which this model construction is achieved are possible, depending on the specific learning algorithm being employed. It is possible to divide all current machine learning algorithms into the following general categories [68]:

- *Supervised learning* uses a set of data examples, where each example has a pre-associated class. The learning algorithm builds either mutually distinguishing descriptions of the classes, or a regression model describing continuous attributes.
- *Unsupervised learning* uses a set of examples with no prior classification. The learning algorithm finds patterns in the provided examples, typically by modeling the distribution of the examples, and then builds a description for each pattern.
- *Reinforcement learning* rewards a learning algorithm's correct outcomes, and punishes poor outcomes. The learning algorithm aims to discover which actions maximize the overall reward, without any prior knowledge of which actions to take.

Many machine learning approaches only address the first task of learning (i.e., the construction of an internal representation). Model transition functions are derived using special *temporal learning techniques* [69, 88, 214], which are beyond this work's scope.

2.3 Knowledge Discovery in Databases

The focus now shifts towards the practical means through which it is possible to facilitate learning, based upon a *real-world* set of examples. These examples are usually stored in the form of a database, which is maintained by some sort of organization.

In the following discussion, the nature of the setting within which learning must take place is firstly considered in Section 2.3.1. This leads directly to a justification for the existence of the knowledge discovery in databases (KDD) process, which incorporates either EDA or DM as a component, and is described within Section 2.3.2.

2.3.1 Data-Centric Organizations

Over time, organizations have become progressively more data driven. A modern organization processes and works with more data than was typical in the past, and many are

today considered to be data-centric in nature. Any data-centric organization is affected by a variety of unique factors. Most notable amongst these are the following:

- Modern data capturing tools are used to acquire real-time data relating to transactions as they happen. This means that it is possible to collect a vast quantity of relevant data with almost no disruption to the operation of the organization.
- Modern relational database storage allows for the efficient storage of complex data entities, as well as interrelationships between these entities. Thus it is possible to store a large number of related data objects with ease.
- Modern storage media allow for the compact storage of huge quantities of data for an indefinite time. Thus it is possible to archive the huge quantity of acquired data with little or no degradation, over a very long period of time.

These combined factors result in a situation in which it is possible for even small organizations to be in possession of a huge amount of data. The potential advantage this provides is obvious: the information in the possession of an organization is bound to hide much useful knowledge. The more data that is available, the more potential there is to find useful knowledge. Also, if the information recorded spans a long period, it will be easier to uncover interesting patterns (or trends) that are dependent on time.

There are, however, also drawbacks to this proliferation of data. Firstly, the sheer volumes involved present an overwhelming obstacle. For knowledge to accurately reflect reality, it should encompass all available data. Clearly, the search for knowledge becomes increasingly computationally expensive as more records need to be examined.

Secondly, a typical relation in a database often consists of many different attributes, where the domains of these attributes have a variety of data types. In simple environments it would be possible for a human expert to relatively easily analyze a database consisting entirely of numeric attribute values, but when string-based nominal values and Boolean values are added, sensible analysis becomes more difficult.

In the third place, dimensionality begins to present a problem. Within a typical database relation, consisting of a number of attributes, a generalization is often described using a combination of attribute values. For example, the status of a company could be dependent on the turnover, *in conjunction with* the size of the company. This means that

the knowledge extraction task may be thought of as a form of search process through an $\tilde{n}$-dimensional space, where $\tilde{n}$ represents the number of attributes making up a record. It should be obvious that, in the presence of a large number of records (as is usually the case in practice), this search becomes exponentially more time complex.

2.3.2 The KDD Process

The most obvious approach to extracting knowledge from a data set would be for a human expert to manually inspect the entire data set, devise an appropriate model through the combination of their observations with appropriate domain knowledge, and finally represent the salient details of this model in some way. This is sometimes referred to as *manual data mining* [71]. However, due to the drawbacks mentioned in Section 2.3.1, it quickly becomes apparent that this approach is usually overwhelming. Even a team of several experts would have difficulty coordinating their efforts.

This sets the stage for the introduction of some form of automation to the process of knowledge extraction. The purpose of KDD is to extract new knowledge from databases in which the volume, complexity and dimensionality of the stored data has proved prohibitively large for human analysts [222]. It is clear from the above discussion that machine learning is an ideal candidate for providing this automation.

It is not possible to simply apply machine learning approaches directly to a real-world database. This is because machine learning approaches operate on a largely consistent, machine-encoded environment. Conversely, KDD views its environment through an operational database, which is subject to a wide variety of problems and inconsistencies. These problems often include noisy data, incomplete data, redundant data, and the possibility of multiple heterogeneous data sources. These and other problems have been discussed in great detail elsewhere in the literature [54]. The KDD process must therefore overcome these issues through a series of *pre-processing* phases.

Because the aim of the KDD process is the extraction of useful knowledge, the model produced by a machine learning approach must be converted into a usable form. This becomes the task of several *post-processing* phases of the KDD process.

KDD therefore encompasses the entire process of knowledge extraction, which begins with an unprocessed database, and finally produces a useful representation of the

resulting knowledge. This dissertation divides the KDD process into the following seven distinct steps, which are based on the work of Han and Kamber [103]:

1. *Data cleaning*, which is the process whereby any noise and inconsistencies that are present in the data source are removed or dealt with in some way.

2. *Data integration*, which involves the process of combining various (often heterogeneous) data sources into a single, consolidated store [271, 273].

3. *Data selection*, which is the task of selecting only the parts of the data source that are relevant to the KDD task, and using only this data for knowledge extraction.

4. *Data transformation*, where summary or aggregation tasks are performed to transform data into a form appropriate for the knowledge extraction process.

5. *Knowledge extraction*[1], which represents the application of machine learning approaches in order to extract knowledge from the pre-processed data source.

6. *Pattern evaluation*, which employs some kind of measure to identify and select the truly interesting, non-trivial generalizations and patterns identified.

7. *Knowledge presentation*, where the selected patterns are presented by means of one or more visualization and/or knowledge representation techniques.

The KDD process, broken down into the above sequential steps, is illustrated in Figure 2.5. The figure also shows the end-product generated by each step.

The pre-processing phase of KDD is encompassed in steps 1 to 4, while the post-processing phase includes steps 6 and 7. The focus of this work is on the knowledge extraction of step 5, which is where it is proposed to employ the SOM algorithm. Post-processing is not discussed, but the data cleaning and transformation operations that are most important in the context of SOMs are covered in Section 2.4.

The current trend is to combine data cleaning and integration into a single phase, which stores the output of both these steps in a data warehouse. It is not uncommon

[1] Step 5 is called "data mining" by Han and Kamber [103]. Section 2.5.2 of this work uses a more specific definition for this term, necessitating the use of the term "knowledge extraction" in its place.

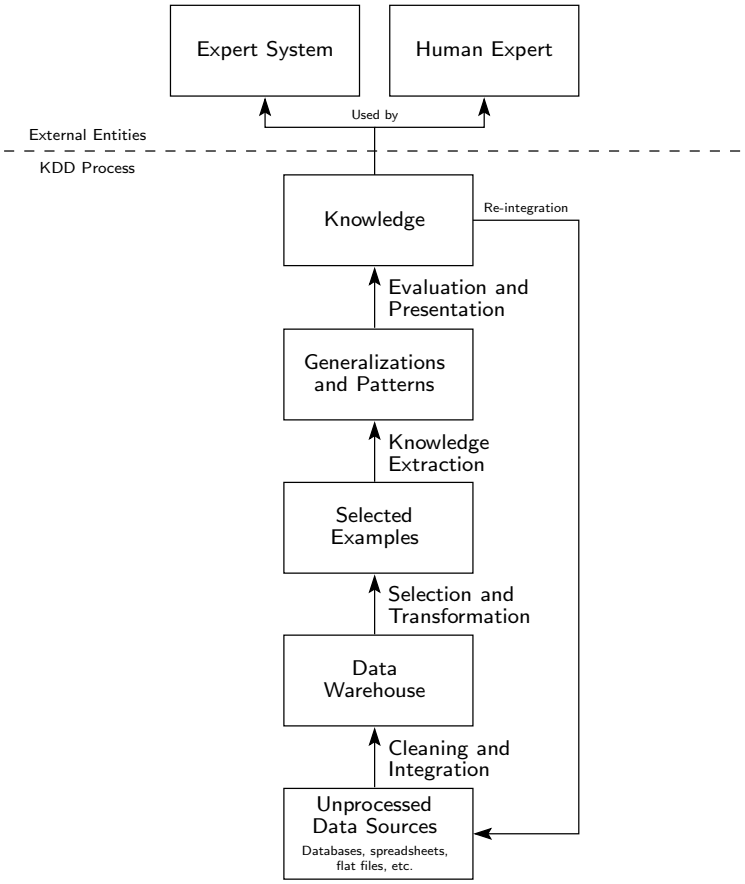


Figure 2.5: An overview of the KDD process and related external entities. Arrows are process phases, with related phases combined for simplicity. Rectangles are inputs, outputs and entities.

to perform data transformation prior to selection, especially when data warehouses are used. It is also possible to generate additional data from the insight gained from the final knowledge output. This data is usable as input for another KDD iteration, and thus has the potential to refine the knowledge generated by the process.

The knowledge that is produced by the KDD process is often used directly by human domain experts for decision-making tasks. Such a context often requires the generation of reports for stakeholders who are not well versed in EDA or DM. This necessarily raises concerns surrounding the clear representation of the extracted knowledge in an easily understandable manner. Alternatively, knowledge integration into an expert system is also possible, in order to automate a recurring decision-making process.

2.4 Basic Data Pre-Processing

As discussed in the previous section, data pre-processing includes data cleaning, integration, selection, and transformation. These are broad topics, thus Sections 2.4.1 and 2.4.2 only briefly touch on some of the more important and commonly performed operations within the phases of data cleaning and data transformation.

2.4.1 Basic Data Cleaning Operations

Data cleaning addresses the fact that real-world data sets are often noisy, incomplete, and contain inconsistencies. The most common data cleaning operations are *missing value replacement*, *noise reduction*, and *inconsistency cleaning*, and are discussed below:

2.4.1.1 Missing Value Replacement

Examples with *null* attribute values interfere with knowledge extraction. While a naïve option simply removes such examples, replacement strategies are preferable. There are many approaches for replacing *null* items with valid values [25, 84, 192]. Commonly used methods include the manual replacement of missing values, replacing missing values with the mean for the attribute in question (computed across all examples, or only over the relevant class), and inferring the most probable value using statistical techniques.

2.4.1.2 Noise Reduction

Noise reduction is necessary when random errors or variances are present in a data set. Again, there are many ways to smooth out noise [103]. The most prevalent approaches for dealing with data set noise include smoothing attribute values within bins of examples, clustering examples to detect and remove outliers, and the use of regression techniques.

2.4.1.3 Inconsistency Cleaning

Inconsistent data usually results from human error amongst data capturers, or a poor data gathering system. Correcting such problems is typically quite manual, although some algorithmic approaches use attribute dependencies to help correct inconsistencies.

2.4.2 Basic Data Transformation Operations

Data transformation is a very important data pre-processing procedure encompassing all the operations that are required to convert data into a form more appropriate to a specific machine learning approach. In the context of this research work, this data transformation conversion must take the SOM algorithm into consideration.

As Section 3.2 describes, the SOM algorithm bases its model on continuous values. All learning algorithms that are based on continuous values require two data transformations: *attribute value normalization* and *binary encoding of nominal attributes*. Of course, all SOM-based DM and EDA methods implicitly require these transformations. Attribute value normalization and binary encoding are both discussed in more detail, below:

2.4.2.1 Attribute Value Normalization

The SOM's input is a relational schema of continuous-valued attributes. In a practical setting, these attributes often have ranges that differ significantly from one another. Because SOM training is based on distances calculated within the space defined by the attribute values, attributes that have significantly larger value ranges (such as `turnover` in the example of Figure 2.1) will outweigh and dominate over attributes with smaller ranges (such as `avg_remun` in the example). This effect biases the learning process, but is addressable with *attribute value normalization* (which is also called scaling).

Min-max, z-score, and decimal scaling normalization are some of the techniques that have been developed to address the above-mentioned problem [103]. Each normalization method has particular advantages and drawbacks under different conditions [189].

Min-max normalization is one of the most widely used data normalization techniques, and is often used in the context of SOMs. Using min-max normalization, each original attribute value, f_l, is adapted to a normalized value, f'_l, as follows:

$$f'_l = \frac{f_l - f_{l,min}}{f_{l,max} - f_{l,min}} \cdot (f'_{l,max} - f'_{l,min}) + f'_{l,min} \tag{2.1}$$

where $f_{l,min}$ and $f_{l,max}$ respectively denote the minimum and maximum possible values that f_l can take; and $f'_{l,min}$ and $f'_{l,max}$ respectively indicate the minimum and maximum possible values that f'_l should be able to take. This normalization method preserves the relationships that are present between attribute values within the original data set.

If a machine learning algorithm is based on data that contains normalized attributes, its model will also be normalized with respect to these attributes. In order for such a model to be interpreted, the appropriate values should be de-normalized. The de-normalization process returns normalized values to their original range, and allows for the model's interpretation in the context of the original data source. The nature of de-normalization depends on the normalization method that was used on the original training data, and is essentially the complement of this original process.

The de-normalization process that is appropriate for min-max normalization, is derived from Equation (2.1). The procedure converts a min-max normalized value, f'_l, to its corresponding de-normalized value, f_l, and is based on the following equation:

$$f_l = \frac{f'_l - f'_{l,min}}{f'_{l,max} - f'_{l,min}} \cdot (f_{l,max} - f_{l,min}) + f_{l,min} \tag{2.2}$$

It is, of course, necessary to use the original $f_{l,min}$ and $f_{l,max}$ values that attribute value f_l could take prior to the data normalization. This naturally requires that these values be stored accurately whenever de-normalization might need to be performed.

2.4.2.2 Binary Encoding of Nominal Attributes

Binary attribute encoding is required when an attribute's domain is of a nominal type. An example of such a nominal attribute in Figure 2.1 is `size`. Typically, an integer encoding is used to represent such values. A possible value encoding for the example's `size` attribute is `0` = `small`, `1` = `medium`, and `2` = `large`. Such a representation is not ideal, because a SOM requires continuous-valued attributes. A SOM will thus interpret each discretely encoded nominal attribute value as a single continuous value. In the example, `size` would be interpreted as a continuous value in the range $[0, 2]$.

Such an attribute should be replaced by a separate attribute for each allowable nominal value. For the example in Figure 2.1, a data analyst might create three attributes in place of the single `size` attribute, called `size_small`, `size_medium` and `size_large`. To indicate the value of the original attribute, a continuous value of `1.0` is given for the corresponding new attribute, and `0.0` for the others. Because the tuple for the company name `UAC` in Figure 2.1 has a `size` value of `medium`, the values for `size_small`, `size_medium` and `size_large` for this tuple would be `0.0`, `1.0` and `0.0`, respectively.

2.5 Exploratory Data Analysis versus Data Mining

The fields of EDA (exploratory data analysis) and DM (data mining) are closely related, and must thus be differentiated in the context of this work. Either EDA or DM are substitutable within the knowledge extraction phase of the KDD cycle identified in Section 2.3.2. Section 2.5.1 discusses EDA, while Section 2.5.2 covers DM.

2.5.1 Exploratory Data Analysis

The field of EDA [235, 250] uses various techniques to augment a human analyst's expertise. The type of augmentation differs, depending on the level of assistance offered to the analyst. Some EDA tools simply represent the most important aspects of the examined data set in a particularly clear and concise way. It is also possible for EDA tools to perform very complex analysis on relationships within the data, and present a set of results in a form that can be easily understood and assimilated by a user.

Common to most EDA approaches, however, is some kind of visual representation based on one or more of the many information visualization techniques [109, 110] that have been developed. Such visualizations represent the data set in an understandable form, while still preserving key aspects of the data's essential meaning [131].

It is important to note, however, that the aim of EDA is not to provide some kind of interpretation to the data, but simply to *aid* the task of a human expert during knowledge extraction. EDA systems are thus typically interactive, and the actual "intelligence" of the knowledge extraction task still lies in human hands. As such, tools for EDA need not necessarily utilize any AI or CI paradigms (although this dissertation focuses only on applications that do). In fact, in many practical situations, only the application of comparatively simple statistical or OLAP systems is sufficient. Figure 2.6 presents an overview of the component interactions within a generic EDA system.

EDA can be appropriately used in two contexts. Firstly, EDA can be used when an analyst has no clear idea of what knowledge is being searched for. In this case, EDA is used as a tool to allow the analyst to explore the data in a roughly trial-and-error manner, searching for any knowledge that seems to be of interest. Alternatively, EDA is also appropriate if an analyst has a clear idea of what knowledge is being searched

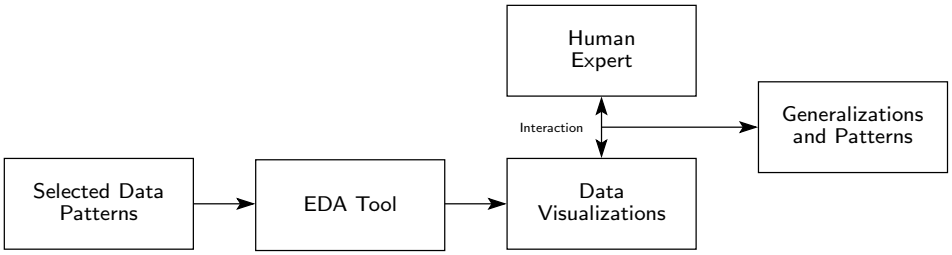


Figure 2.6: An overview of the EDA process.

for, what questions need to be answered, or what hypotheses must be validated by the exploratory exercise. This focus can quickly guide the exploration process to an outcome.

2.5.2 Data Mining

In contrast to EDA, the field of DM attempts to fully automate the extraction of knowledge from data, using machine learning methods. Typically a DM application simply generates a rule set, given an input data set. There is no human intervention, except for the evaluation and presentation of the rule sets after knowledge extraction, and the use of the extracted knowledge or its integration into an expert system.

Many DM algorithms have been developed. While many older techniques, for example CN2 [23, 37, 38] and C4.5 [193, 194, 195, 197], use statistical and information theoretic measures to build rule sets, the current trend is towards using CI paradigms.

Amongst many CI-based techniques, feedforward neural networks [213], evolutionary algorithms [18, 83], ant colony optimizers [179], support vector machines [265], and fuzzy systems [74] have been used as an algorithmic basis for rule extraction in a DM setting. Such CI-based rule extraction is the field that is focused upon within this dissertation. Figure 2.7 provides a broad schematic overview of the DM process.

In contrast to EDA approaches, DM is typically employed when the nature of the knowledge that is desired from the mining process is not clearly defined, or cannot be defined. It is usual for there to be no fixed questions that have to be answered, and no set hypotheses that require validation. DM is appropriate in this type of context because the DM tool simply generates a representation of the knowledge that can be extracted from the data, without the need for direct guidance from a human analyst.

Figure 2.7: An overview of the DM process.

Furthermore, DM is often appropriate when extracted knowledge must be integrated into an expert system. This type of integration is facilitated by the fact that the knowledge generated by a DM approach is already in a machine-readable format that can usually be directly integrated into an automated system. If this is not the case, it is usually simple to modify the extracted knowledge so that it can be integrated.

2.6 Summary

This chapter focused on the delineation of the fields of EDA (exploratory data analysis) and DM (data mining). Section 2.1 defined data as a representation of the state of an environment, and knowledge as some form of insight into the environment as represented by a set of data. These concepts were theoretically expanded upon. Section 2.2 defined learning as the process by which knowledge can be derived, and considered deductive and inductive learning, and also defined machine learning as the algorithmic automation of inductive learning. Section 2.3 defined the field of KDD (knowledge discovery in databases) as the entire process of extracting useful, presentable knowledge from a real-world data set. Section 2.4 briefly outlined a few important data pre-processing operations. In particular, the need for attribute value normalization and binary encoding were justified. Finally, Section 2.5 defined the fields of EDA and DM as steps in the KDD process, both of which utilize machine learning approaches. KDD was defined as the use of machine learning to augment the expertise of a human expert. DM was defined as the complete automation of the knowledge extraction process using data mining.

The following chapter considers the more important aspects relating to the architecture, training, parameter settings and evaluation of the CI approach under consideration, namely the SOM. In addition, the concepts surrounding emergent behavior in SOMs are discussed. Finally, variations on the basic theme of the SOM are considered.

Chapter 3

Self-Organizing Feature Maps

The previous chapter discussed the fields of EDA and DM, in which the SOM algorithm has been applied. This chapter provides an overview of SOM theory as well as recent research on SOMs. The basic algorithm was developed in 1982 by Teuvo Kohonen [142]. Most of this chapter is based on Kohonen's summarizing work on the topic [146].

Section 3.1 gives an overview of the approach, while Section 3.2 discusses the map architecture. Section 3.3 describes the classical training algorithm. Section 3.4 discusses emergence in SOMs, Section 3.5 covers the parameters affecting a SOM's training, and Section 3.6 considers factors affecting SOM model accuracy. Variations on the basic SOM are discussed under Section 3.7. Finally, Section 3.8 summarizes the chapter.

3.1 Overview of the Approach

Like many CI approaches, the SOM algorithm is based on a system that exists in nature. This biological basis is discussed in Section 3.1.1. Following this background, Section 3.1.2 briefly outlines how this basis is modeled by the SOM approach.

3.1.1 Physiological Basis

The physiological underpinnings of the algorithm are the self-organizing nature of associative memory and the human cerebral cortex (particularly the motor, somatosensory, visual and auditory cortices) [142, 145]. These cortices form in such a way as to represent

similar sensory inputs within localized cortex areas. This structuring allows, for example, images of trees to be differentiated from images of faces, and previously unseen trees to be recognized as trees, because the visual similarity of trees causes neural responses in similar cortex areas. The structure of these regions of the brain can be represented by topological feature maps. It is this representation that the SOM models algorithmically.

3.1.2 Algorithmic Overview

Broadly speaking, the SOM is a type of artificial neural network (ANN). ANNs constitute a class of machine learning approaches inspired by the structures of biological nervous systems, and build a model by using a set of provided data examples [68, 122].

Supervised learning ANNs, given input examples, iteratively adjust their produced outputs to desired outputs. Unsupervised learning ANNs model input data, thus finding patterns without using desired classifications. The SOM is an unsupervised ANN.

A SOM performs a multidimensional scaling of a set of I-dimensional input examples to a discrete output space, called the map structure, which consists of neurons (also known as map units). In general, the map structure is less complex than the set of input examples for two reasons. Firstly, the map's neurons are usually arranged in a structure with a lower dimensionality than that of the input examples [133]. Secondly, the map is typically made up of fewer neurons than the number of input examples [68].

The SOM thus aggregates (or compresses) the input space using neurons to form an approximate model of the input data, where the model has two important characteristics:

- It approximates the probability density function of the input space by means of neurons that cluster similar and frequently occurring input examples together. This means that neurons tend to model dense areas in the input space.

- It maintains the local topological structure (or *local order*) of the input space. This means that if two input examples are close to one another in the I-dimensional input space, they will be close to one another in the SOM's output space.

This form of mapping differs from the models created by other unsupervised approaches (such as the learning vector quantizer, or LVQ [143]), which typically take the approach of only sub-dividing the data into clusters, with no topological arrangement.

3.2 SOM Architecture

Figure 3.1 illustrates a SOM's architecture. SOM learning is based on a *training set*, $\mathcal{D}_T$, which is a subset of a master data set, $\mathcal{D}$. The training set consists of P_T object examples, called *training examples*, each represented by an I-dimensional *training vector*, $\vec{z}_s$, such that $\mathcal{D}_T = \{\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_{P_T}\}$. Each training vector consists of I continuous-valued components, each denoted $z_{s\hat{w}}$, called *input parameters*, such that $\vec{z}_s = (z_{s1}, z_{s2}, \ldots, z_{sI})$. Within an EDA or DM context, $\mathcal{D}$ is usually a structured database, as defined in Section 2.1.1.2. In this case, each $\vec{z}_s$ is a tuple, and every $z_{s\hat{w}}$ is an attribute value.

The map normally takes the form of a two-dimensional, rectangular grid of $Y \times X$ neurons, where Y is the total number of rows, and X is the total number of columns in the map. Neurons are arranged in a *lattice structure*, which defines the neighborhood structure around each neuron. The lattice topology therefore determines the spatial relationships of neurons to one other, within the grid's map space.

Two general lattice topologies, namely *rectangular* and *hexagonal* lattices [148], are the most common within the SOM literature. A rectangular lattice considers neighboring neurons to be those located adjacently above, below, and to either side of one another. Consequently, this structure arranges neurons in regular rows and columns, as shown in Figure 3.2 (a). A hexagonal lattice defines a neighborhood of six neurons, requiring a tessellated arrangement, which is shown in Figure 3.2 (b). From Figure 3.2, it is also clear that both lattices result in smaller neighborhoods for neurons at the corners and sides of the map grid. In the literature, the most widely-used topology is the hexagonal lattice, which is assumed for the remainder of the dissertation.

It should, however, be noted that arbitrary lattice structures are also possible, such as hypercubical [15], irregular [4, 7, 20, 85, 87, 100, 127, 167, 169] and cyclic [159, 207, 215] lattices. Such lattices introduce complexities, and are beyond this work's scope.

The training procedure of the SOM is based on the positions of neurons relative to one another in map space. The lattice structure is thus used during training to define the spatial positions of neurons in map space, and allows for the calculation of distances between the neurons. However, the structure of the lattice typically only becomes apparent when the SOM's map grid is visually represented. This concept is investigated in Chapter 4, which deals with SOM visualization in some detail.

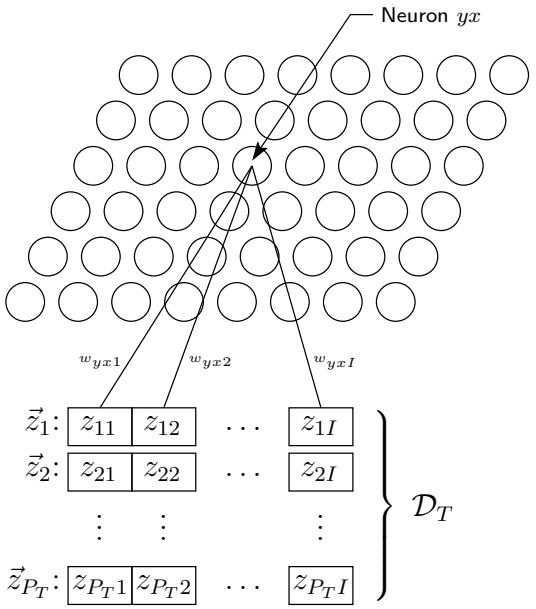


Figure 3.1: The basic architecture of a typical SOM. Circles represent neurons arranged in a rectangular map structure. The lines connected to neuron yx are weight vector components. Weights are connected to all the other neurons in the same way, but are not shown. Rectangles represent input parameters, which are adjacently connected into training vectors.

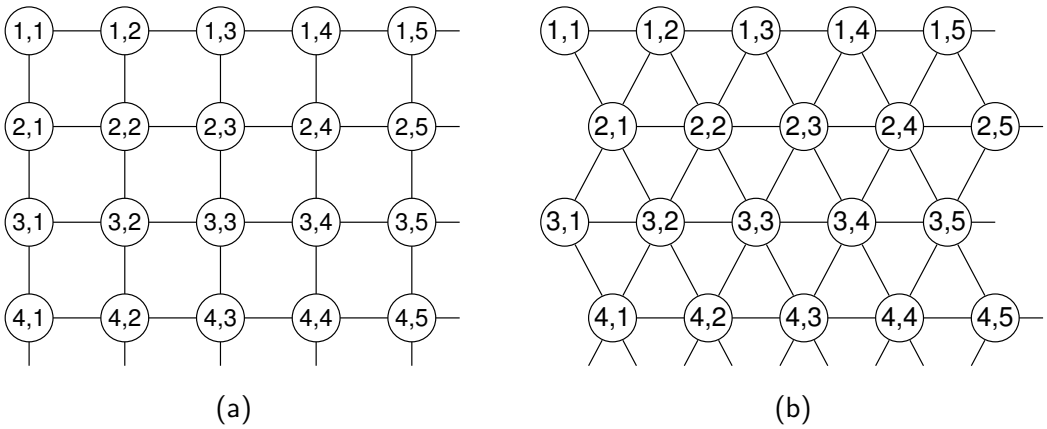


Figure 3.2: The two most common map lattice structures: (a) shows the simple rectangular lattice; (b) shows the more commonly used hexagonal lattice. In both diagrams, circles represent neurons, while lines link neighboring neurons. The first integer within a circle denotes the row coordinate of the neuron in question, while the second signifies the column coordinate.

Each neuron, n_{yx}, on the map is connected to a weight vector (or codebook vector), denoted $\vec{w}_{yx}$, where y is the row coordinate, and x is the column coordinate of the neuron. Row and column coordinates both start at one. Each weight vector represents the centroid of a cluster associated with its neuron. Each $\vec{w}_{yx}$ is also I-dimensional (where I corresponds to the dimensionality of the map's training vectors), and consists of continuous-valued weight values, each denoted w_{yxv}, such that $\vec{w}_{yx} = (w_{yx1}, w_{yx2}, \ldots, w_{yxI})$.

3.3 Stochastic SOM Training

Map *training* is the process whereby the weight vector values of every neuron on the map are adjusted, so that the weights eventually approximate the distribution of the input examples. The training process is iterative in nature, updating the weight vector components that make up the map structure by means of a series of adjustments.

The standard SOM training algorithm performs a weight adjustment for each training example. A single example presentation is called a *training iteration*. The current iteration, t, takes on an initial value of 0, and increases by one after each training vector triggers a weight update. Usually, training requires several passes through the entire training set (called *epochs*). To avoid any bias based on example order, the set is randomly shuffled before each epoch, making the algorithm stochastic in nature.

The process can be roughly categorized into three general tasks, namely initialization, weight adjustment and the evaluation of stopping criteria. Each of these tasks is elaborated upon under Sections 3.3.1, 3.3.2, and 3.3.3, respectively. Section 3.3.4 briefly discusses the possible approaches for handling data set classification attributes. Algorithm 3.1 outlines the general training procedure of a SOM, in the form of pseudocode.

3.3.1 Initialization

Several training parameters, all of which are discussed in greater detail under Section 3.5, affect the operation of the training process that is followed by the SOM. Therefore, as a preparatory step performed before training commences, initialization of the SOM first requires user-selected values to be assigned to each of these training parameters.

Initialize a map of $Y \times X$ neurons, and initialize the algorithmic training parameters
Set the current training iteration, $t = 0$
Begin the first training epoch, and randomly shuffle $\mathcal{D}_T$
repeat:
    Select the next training vector, $\vec{z}_s \in \mathcal{D}_T$, where $\vec{z}_s$ is unselected in the current epoch
    Find the best matching unit (BMU) for $\vec{z}_s$, according to Equation (3.2)
    for all *weight vectors,* $\vec{w}_{yx}$*, in the map* **do**
        Update $\vec{w}_{yx}$ according to Equation (3.3)
    end for
    Update the current training iteration, $t = t + 1$
    Update all parameter values that are dependent on t
    If all $\vec{z}_s \in \mathcal{D}_T$ have been selected during the current epoch, randomly shuffle $\mathcal{D}_T$
until *at least one stopping criterion is met*

Algorithm 3.1: Pseudocode of the standard stochastic SOM training algorithm.

Next, each weight vector's constituent weights must be given initial values. There are various possible methods for selecting these weight values, which include:

- The simplest approach gives each weight a uniform random value [146], constrained to the corresponding attribute range [68]. Such weights do not resemble the training data's distribution, and require many adjustments, thus causing slow training.
- Another simple method sets weight vectors to uniformly randomly selected training example values. Training is biased towards these data examples, and often converge prematurely. Weights should thus be perturbed with small random values [68].
- It is possible to initialize weight vectors in an orderly fashion, relative to the principal components that account for most of the training data variance [11, 146].
- Su *et al* devised an initialization technique that defines a weight hypercube large enough to uniformly cover most of the training examples in the training set [231].

Algorithm 3.2 shows hypercube initialization, which was used in the experiments of Chapter 8. Examples of hypercube and random initialization are compared in Figure 3.3.

Set $\vec{w}_{Y1} = \vec{z}_{s1}$ and $\vec{w}_{1X} = \vec{z}_{s2}$, where $\|\vec{z}_{s1} - \vec{z}_{s2}\|_2$ is maximal over $\mathcal{D}_T$
Set $\vec{w}_{11} = \vec{z}_{s3}$, where $\|\vec{z}_{s1} - \vec{z}_{s3}\|_2 + \|\vec{z}_{s2} - \vec{z}_{s3}\|_2$ is maximal over $\mathcal{D}_T$
Set $\vec{w}_{YX} = \vec{z}_{s4}$, where $\|\vec{z}_{s1} - \vec{z}_{s4}\|_2 + \|\vec{z}_{s2} - \vec{z}_{s4}\|_2 + \|\vec{z}_{s3} - \vec{z}_{s4}\|_2$ is maximal over $\mathcal{D}_T$
for all *column indices* $x \in \{2, 3, \ldots, X-1\}$ *in the map* **do**
    Set $\vec{w}_{1x} = \frac{\vec{w}_{1X} - \vec{w}_{11}}{X-1} \cdot (x-1) + \vec{w}_{11}$ and $\vec{w}_{Yx} = \frac{\vec{w}_{YX} - \vec{w}_{Y1}}{X-1} \cdot (x-1) + \vec{w}_{Y1}$
end for
for all *row indices* $y \in \{2, 3, \ldots, Y-1\}$ *in the map* **do**
    Set $\vec{w}_{y1} = \frac{\vec{w}_{Y1} - \vec{w}_{11}}{Y-1} \cdot (y-1) + \vec{w}_{11}$ and $\vec{w}_{yX} = \frac{\vec{w}_{YX} - \vec{w}_{1X}}{Y-1} \cdot (y-1) + \vec{w}_{1X}$
    for all *column indices* $x \in \{2, 3, \ldots, X-1\}$ *in the map* **do**
        Set $\vec{w}_{yx} = \frac{\vec{w}_{yX} - \vec{w}_{y1}}{X-1} \cdot (x-1) + \vec{w}_{y1}$
    end for
end for

Algorithm 3.2: Pseudocode of the hypercube weight initialization algorithm.

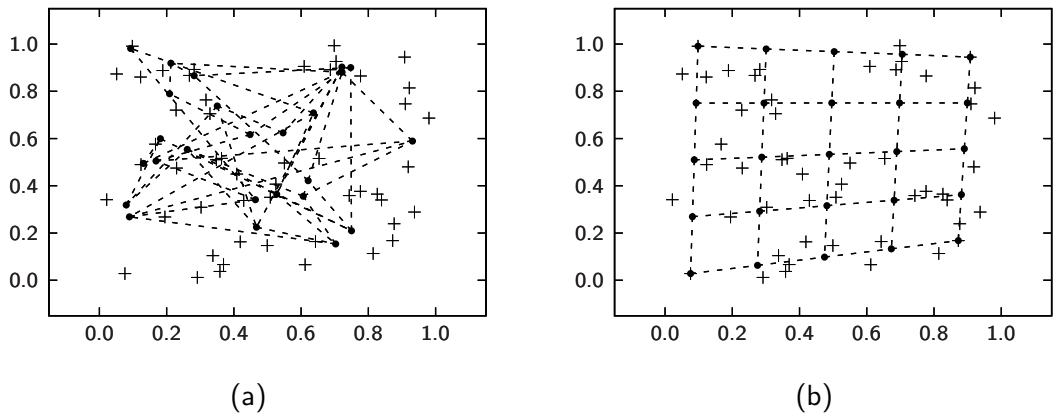


Figure 3.3: Weight initialization for two-dimensional data: (a) shows random initialization; (b) shows hypercube initialization. Crosses show data, and dashes link adjacent weight vectors.

Both initializations are performed for the same randomly generated two-dimensional data set. Figure 3.3 (a) shows random weight initialization, which clearly produces an unordered initial map that will take longer to optimize. Figure 3.3 (b) shows the result of the hypercube initialization, which is ordered and uniformly covers most of the data.

3.3.2 Weight Adjustments

Each training example generates a series of adjustments to the weight vector components associated with the neurons making up the map structure. The process followed during each series of weight adjustments consists of two general steps, as follows:

Step 1: Determine the Best Matching Unit

Many of the operations related to a SOM, including training, are heavily based on a distance measure. Although any feasible measure may be utilized, the *Euclidean distance* is the most commonly used in practice, and is assumed throughout this dissertation. The notation $\|\cdot\|_2$ denotes the *Euclidean norm*. The Euclidean distance between two arbitrary V-dimensional vectors is defined, using Euclidean norm notation, as follows:

$$\|\vec{q_1} - \vec{q_2}\|_2 = \sqrt{\sum_{\hat{p}=1}^{V}(q_{1\hat{p}} - q_{2\hat{p}})^2} \tag{3.1}$$

where $\vec{q_1}$ and $\vec{q_2}$ are the two vectors, $q_{1\hat{p}}$ and $q_{2\hat{p}}$ are respectively the components at position $\hat{p}$ in vectors $\vec{q_1}$ and $\vec{q_2}$, and both $\vec{q_1}$ and $\vec{q_2}$ must contain V components.

The best matching unit (BMU) represents the neuron whose weight vector most closely matches the current training vector in terms of Euclidean distance, as follows:

$$\|\vec{z}_s - \vec{w}_{ba}\|_2 = \min_{\forall yx}\{\|\vec{z}_s - \vec{w}_{yx}\|_2\} \tag{3.2}$$

where $\vec{z}_s$ denotes the training vector, $\vec{w}_{ba}$ denotes the weight vector of the BMU at row b and column a, and $\vec{w}_{yx}$ denotes an arbitrary weight vector at row y and column x. In the presence of missing attribute values, the distances are calculated using only the available vector components in $\vec{z}_s$, and their corresponding weights in $\vec{w}_{yx}$.

Step 2: Update Weight Vectors

Once a BMU has been established for the current training example, a series of weight adjustments must take place across the *entire map*. The intended effect of a single weight adjustment is to move the weight vectors of the BMU *and its neighboring neurons in lattice space* closer to the current training example, as illustrated in Figure 3.4. The

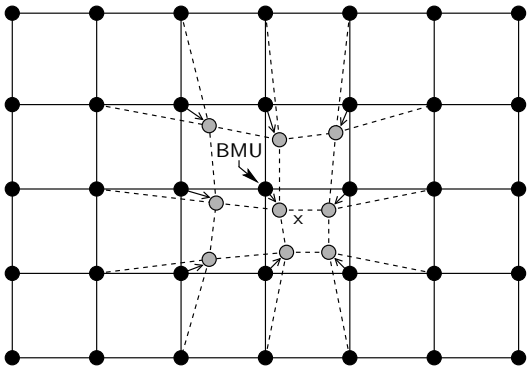


Figure 3.4: The effect of a weight update on the best matching unit (BMU) and its neighbors given a training example, denoted by x. Black and grey dots show weight vector positions before and after the update, respectively. Figure adapted from the work of Simula *et al* [222].

weight adjustments are relative to the BMU, with the largest adjustment applied to the BMU itself. Progressively smaller modifications take place the further neurons are from the BMU on the map lattice, until the adjustments eventually become negligible.

Two important processes are observable during an iterative sequence of updates to the weight vectors, which are performed during the training of a SOM:

- The area-based neuron updates create a smoothing effect on the map's weight vectors. This helps establish a locally ordered topological structure for the mapping, where neighboring neurons adapt to have similar weight vector values.

- In a locally ordered topology, neighboring neurons have similar weight vectors. These similar weights tend to result in the selection of neighboring BMUs for training examples that are close to one another in the input space, resulting in numerous similar updates in the same map area. These repeated updates further aid the formation of local topological order, and will draw weight vectors into dense groupings that mimic the probability density of the input space.

These two processes interact with one another, in order to preserve the two crucial characteristics of the model built by a SOM, which were discussed in Section 3.1.2. The overall effect is that of an "elastic net" of weight vectors that is stretched to fit over, and approximately model, the data space [68]. It has been shown that, when allowed sufficient training cycles, a SOM's map structure will become locally ordered [146].

It should be noted that the local ordering effect does not necessarily imply the existence of only single areas of homogeneous weight vectors within a map structure. It is possible for several areas that contain similar BMUs to be formed during training.

In a *globally ordered* (or continuous) map, groups of similar weight vectors are localized within only single areas of the map. Global order can be expected, given sufficient training iterations and a globally ordered initial map (the hypercubical and principal eigenvector methods mentioned in Section 3.3.1 produce such maps). Conversely, poor initialization or premature training termination may produce an unordered map.

The magnitude of every update, at training iteration t, to the weight vectors associated with each neuron across the entire map, is expressed by the following:

$$\vec{w}_{yx}(t+1) = \vec{w}_{yx}(t) + \Delta\vec{w}_{yx}(t) \tag{3.3}$$

where $\vec{w}_{yx}(t)$ denotes an arbitrary weight vector that is located at row y and column x of the map at training iteration t, and $\Delta\vec{w}_{yx}(t)$ is the change applied to this weight vector at iteration t. The change to weight vector $\vec{w}_{yx}$ is, in turn, defined as:

$$\Delta\vec{w}_{yx}(t) = \big(\Delta w_{yx1}(t), \Delta w_{yx2}(t), \ldots, \Delta w_{yxI}(t)\big) \tag{3.4}$$

where $\Delta w_{yxv}(t)$ denotes the update that is applied at training iteration t to w_{yxv}, which is weight component v of weight vector $\vec{w}_{yx}$. Finally, the change that is calculated for an individual weight component is defined according to the equation:

$$\Delta w_{yxv}(t) = h_{ba,yx}(t) \cdot \big(z_{s\hat{w}} - w_{yxv}(t)\big) \tag{3.5}$$

where $z_{s\hat{w}}$ denotes the corresponding input parameter $\hat{w}$ of $\vec{z}_s$ (the training vector currently under consideration), and $h_{ba,yx}$ is the so-called *neighborhood function*. The neighborhood function determines the magnitude of the adjustment applied to each w_{yxv} that makes up $\vec{w}_{yx}$, and is relative to the BMU at row b and column a of the map.

In order to achieve the local topological weight vector grouping that Figure 3.4 illustrates, $h_{ba,yx}$ must be a function of the Euclidean distance, $\|c_{ba} - c_{yx}\|_2$, between the two-dimensional lattice coordinates of the BMU and the weight vector being updated (respectively, c_{ba} and c_{yx}). As the magnitude of $\|c_{ba} - c_{yx}\|_2$ increases to a maximum for the map dimensions (in other words, $\vec{w}_{yx}$ is further from the BMU), $h_{ba,yx} \to 0$.

Any form of the neighborhood function is possible, as long as the above-mentioned condition is maintained. Examples include the square or hexagonal "bubble" neighborhoods [148]. Most often, however, a smooth Gaussian kernel is used, defined as:

$$h_{ba,yx}(t) = \eta(t) \cdot \exp\left(-\frac{\|c_{ba} - c_{yx}\|_2^2}{2 \cdot \big(\sigma(t)\big)^2}\right) \tag{3.6}$$

where $\eta(t)$ denotes the so-called *learning rate factor* and $\sigma(t)$ denotes the *kernel width*, both at training iteration t. The nature of $\eta(t)$ and $\sigma(t)$ are discussed in detail under Sections 3.5.2 and 3.5.3, respectively. Figure 3.5 visualizes a Gaussian kernel.

Weights can still be adapted in the presence of missing attribute values [210, 246], if the values of all Δw_{yxv} that correspond to any missing $z_{s\hat{w}}$ values are assumed to be 0.0. Naturally, training becomes less accurate in the presence of many missing values.

Several attempts have been made to describe the dynamics of SOM training mathematically [42, 44, 81], highlighting that SOMs pose many challenges to theoretical analysis. SOM dynamics is not a focus of this dissertation, and is not discussed further.

3.3.3 Stopping Criteria

A condition is required to indicate that SOM training is to terminate. *Convergence* is the point where the learning process stabilizes, and the map's structure ceases to improve. Because SOM performance is dictated by the map structure, performance also stabilizes at this point. Stopping criteria should thus indicate when a SOM has converged.

The stopping criteria for stochastic training that are identified by this dissertation are broadly categorized as being based on either the *number of training iterations*, the *network training error*, the *network weight changes*, or the *network's topological error*. Each category is elaborated upon under a separate heading within this section.

While every SOM must have at least one stopping criterion, implementations typically combine several criteria using Boolean connectives. Such combinations allow, for example, training to stop upon the satisfaction of only one of several criteria, thus ensuring timely termination if any criterion has been specified too strictly. In a similar fashion, it is possible to terminate training only when multiple criteria are satisfied, thus ensuring that a map has a combination of several characteristics upon the termination of training (for example, both an optimal training and topological error can be enforced).

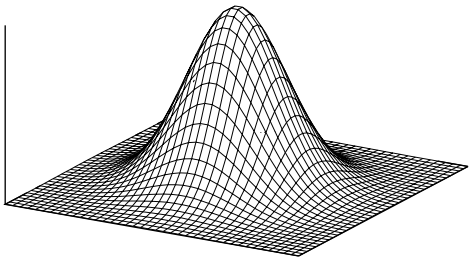


Figure 3.5: A three-dimensional visualization of a smooth Gaussian kernel. The height of the graph denotes the magnitude of the weight adjustment at varying distances from the BMU. The highest point, at the center of the graph, is the location of the BMU.

3.3.3.1 Training Iteration Limit

Training may cease after a *maximum number of training iterations* has been exceeded. However, convergence occurs at varying rates for different problems, meaning that this criterion is not normally sufficient on its own. Therefore, an iteration limit is typically used in combination with other criteria. Such a limit ensures algorithm termination in the event that the SOM does not converge in a reasonable number of iterations.

3.3.3.2 Training Error

It is possible to use a measure of a map's training accuracy to decide when to terminate the algorithm. Usually *average training quantization error* is used (although a weighted variant [148] is less frequently employed). The average training quantization error is defined in terms of the Euclidean distances between each training vector and the BMU that is associated with that training vector, and is calculated as follows:

$$Q_T(t) = \frac{1}{P_T} \cdot \sum_{p=1}^{P_T} \|\vec{z}_s - \vec{w}_{ba}(t)\|_2 \tag{3.7}$$

where $\vec{w}_{ba}$ is the weight vector of the BMU calculated for the training vector $\vec{z}_s$, and t is the training iteration for which the training quantization error is calculated.

The average quantization error typically starts at a relatively high level, and asymptotically approaches a problem-dependent convergence level as training continues. Figure 3.6 (a) shows an example of training error variations over several training iterations.

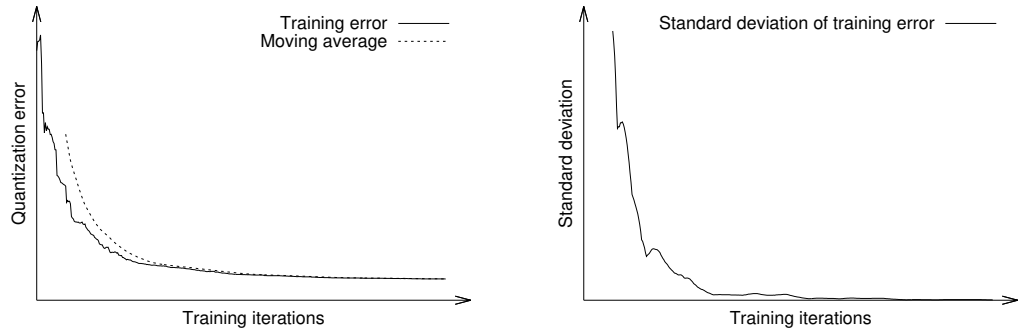


Figure 3.6: A typical change in training quantization error measures as training iterations progress: (a) compares the raw training error and its moving average over a 30-iteration window; (b) shows the training error's sample standard deviation, also over a 30-iteration window.

Three ways of using the training quantization error as a stopping criterion exist: the first using the level of the training error itself, the second using the decrease in training error, and the third utilizing the sample standard deviation of the training error.

A SOM achieving a *minimal quantization error* indicates that all the weight vectors represent optimal centroids for their respective clusters of training examples. Training can stop once Q_T is sufficiently small. Unfortunately, because the error becomes asymptotic at varying levels depending on the nature of the modeled data, it is very difficult (if not impossible) to specify a good generic threshold value for such a condition.

A minimum *decrease in quantization error* from the previous training iteration to the current one indicates that training is not improving the network appreciably. However, training error fluctuations (with temporary plateaus or increases) may prematurely satisfy this criterion. To solve this problem, a *training error moving average* at iteration t, over a sliding window of the last W measurements, smooths these fluctuations:

$$\overline{Q}_T(t) = \frac{1}{W} \cdot \sum_{\hat{t}=0}^{W-1} Q_T(\, t - \hat{t}\,) \qquad \text{if } t \geq W - 1 \tag{3.8}$$

where $\hat{t}$ is an offset from the end of the sliding window and $Q_T(\, t - \hat{t}\,)$ is the average training quantization error at iteration $t - \hat{t}$. At least W training iterations must be completed before calculating the moving average. Once $\overline{Q}_T(t-1) - \overline{Q}_T(t) \approx 0$, training stops. The

fluctuation level of Q_T determines the appropriate W value, where less severe fluctuations use a smaller W, and more drastic fluctuations need a larger W. Figure 3.6 (a) illustrates the smoothing effect produced by the moving average of the training error.

The *sample standard deviation of the training quantization error*, which is also calculated over a sliding window of the previous W training iterations, indicates the degree of fluctuation in training error over time. As the learning process begins to stagnate, and the network's structure begins to change less, this fluctuation decreases. The sample standard deviation of the training quantization error, at iteration t, is calculated as:

$$d_T(t) = \sqrt{\frac{1}{W-1} \cdot \sum_{\hat{t}=0}^{W-1} \Big(Q_T(\, t - \hat{t}\,) - \overline{Q}_T(t) \Big)^2} \qquad \text{if } t \geq W-1 \tag{3.9}$$

where $Q_T(\, t - \hat{t}\,)$ is the average training quantization error at iteration $t - \hat{t}$, and $\overline{Q}_T(t)$ is the moving average of Q_T. The moving average is calculated over the same sliding window size that was selected for the standard deviation, using Equation (3.8). As is the case for the training error moving average, at least W training iterations must have passed. The network is considered to have converged when $d_T(t) \approx 0$, meaning that map training should cease. An example of this convergence effect is shown in Figure 3.6 (b).

3.3.3.3 Weight Change

The *average component weight change* indicates to what overall degree the weight vectors across the map have been adapted during a specific training iteration. The measure represents the mean of the change applied during a training iteration to each weight vector component across every neuron making up a map, and is computed as follows:

$$\Delta w_{ave}(t) = \frac{1}{Y \times X \times I} \cdot \sum_{k,j=1}^{K,J} \sum_{i=1}^{I} |\Delta w_{yxv}(t)| \tag{3.10}$$

where $|\Delta w_{yxv}(t)|$ is the absolute value of the update to vector component v of weight vector $\vec{w}_{yx}$, at training iteration t. When small weight modifications occur across the map, it indicates that the map structure is not being changed a great deal, and that training has stagnated. Therefore, when $\Delta w_{ave}(t) \approx 0$, the SOM has converged.

3.3.3.4 Topological Error

Finally, it is possible to use a measure of the error in map topology as an indicator for training termination. Such a measure sets out to quantify how well a map preserves the topological structure of the input space. The definition of topological error is still open to debate [146], and a very large number of topological error measures exist [183]. Topological error is thus not a focus of this work. Due to the proliferation of measures in this category, this section only briefly overviews the most common approaches.

The *topographic product* [14], tests how well the distances between neighboring neurons in map space and weight space correlate. For a neuron, n_{yx}, the ratio of $\|c_{yx} - c'_{yx}\|_2$ to $\|c_{yx} - c''_{yx}\|_2$ is computed, where c'_{yx} is the map coordinate of the $\hat{o}^{\text{th}}$ closest neuron to n_{yx} in map space, and c''_{yx} is the map coordinate of the $\hat{o}^{\text{th}}$ closest neuron to n_{yx} in weight space. The ratio of $\|\vec{w}_{yx} - \vec{w}'_{yx}\|_2$ to $\|\vec{w}_{yx} - \vec{w}''_{yx}\|_2$ is also computed, where $\vec{w}'_{yx}$ is the weight vector of the $\hat{o}^{\text{th}}$ closest neuron to n_{yx} in map space, and $\vec{w}''_{yx}$ is the weight vector of the $\hat{o}^{\text{th}}$ closest neuron to n_{yx} in weight space. The ratios are calculated per neuron for a specified value of $\hat{o}$, and combined into a normalized value that is averaged over all neurons. Values closer to 1.0 indicate better topological structure.

The *geometrical organization measure* [277] considers every pair of neighboring neurons. A neuron's Voronoi region is the portion of weight space that is closer to the neuron's weight vector than to any other weight vector. For a pair of neighboring neurons, an "intruder" is defined as any neuron with a Voronoi region that intersects with a straight line between the pair. The final measure is the total of the number of intruders across all pairs of neighboring neurons on the map, where higher values indicate a poorer preservation of topological structure. This measure can also be normalized according to the size and topology of the SOM's grid, to facilitate comparisons between maps.

The *topographic error* [138] is another measure, which calculates the number of local topographic errors for every training vector, $\vec{z}_s$, in a data set. To identify local topographic errors, the BMU and second best matching neuron are determined for each $\vec{z}_s$. If these two neurons are not adjacent, a local topographic error has occurred. The final topographic error is the total number of local topographic errors over every example in the training set, normalized by the number of training vectors in the set. A higher value denotes poor topology, while a lower value indicates better topographical structure.

Villman *et al* [262] propose the *topographic function*. For each neuron, the measure determines a receptive field, which is the set of training examples that are closer to the neuron's weight vector than to all other weight vectors. For every neuron, n_{yx}, a term is calculated in relation to a specified threshold value. For positive threshold values, the term is the number of neurons that have receptive fields directly adjacent to the receptive field of n_{yx}, while being separated from n_{yx} in map space by more than the threshold. For negative threshold values, the term is the number of neurons that are directly adjacent to n_{yx} in map space, with receptive fields separated by more than the absolute value of the threshold. The value of the topographic function for positive or negative thresholds is the mean value of the terms for all neurons on the map. For a threshold value of 1.0, the topographic function is the sum of the term's value for a threshold of 1.0 and -1.0, respectively. The topographic function is then computed for a range of threshold values. The largest absolute value of a threshold that produces a topographic function value not equal to 0.0 gives the degree of topographical disorganization in the map.

Polani [183] overviews a large number of alternate topological error measures developed before 2002, although the above-mentioned approaches remain the most commonly cited and used. Many measures have been proposed since Polani's work. A small sample includes measures proposed by De Bodt *et al* [49], Vesanto *et al* [261], Hetel *et al* [106], Vegas-Azcárate *et al* [249], Molle and Claussen [173], and Zhang [272]. These measures are not widely cited in the literature and are thus not described in detail.

It should be noted that topological error alone is unsuitable for use as a stopping condition. This is because topological order does not necessarily imply an accurate data model. For instance, an untrained map that has been initialized in an ordered fashion (for example, using the principal component or hypercube initialization approaches discussed in Section 3.3.1) is generally not yet an accurate model of the training data, but has a well ordered structure. As a consequence, a stopping condition based on the previously described topological error measures should always be paired with at least one condition that represents either the map's training error, or weight change stagnation.

Another approach to basing a stopping condition on both topological error and quantization error is to combine the two into a unified measure. One such measure, proposed by Kaski and Lagus [132], iterates through every training vector, $\vec{z}_s$, and computes the

distance of a two-part path. The first part of the path runs from $\vec{z}_s$ directly to the BMU of $\vec{z}_s$. The second part of the path connects the BMU of $\vec{z}_s$ to the weight vector that is the second closest to $\vec{z}_s$, taking the shortest route that only transits through the weight vectors of neighboring neurons. The final value of the error measure is the sum of this distance for all $\vec{z}_s$ in the training set, where lower values indicate either better training error or topological structure. Because it is impossible to determine whether high errors are as a result of training error or topological error, a separate stopping condition for each type of error would be preferred when more detailed feedback is desired.

Unfortunately, it is unclear which topological error measure is the best suited in practice [146]. Each measure will also vary differently as training progresses, which means that stopping conditions based on different measures would have to be customized. Topological error measures are thus less commonly used as SOM stopping conditions.

3.3.4 Handling Classification Attributes

Although the stochastic SOM algorithm described above is unsupervised in nature, the literature identifies three ways in which the algorithm can handle classification attributes:

- A *purely unsupervised SOM* [146] uses no example classifications at all in its structure, or during training. All $\vec{z}_s$ and $\vec{w}_{yx}$ include only descriptive attributes. Classification attributes are only used to add meaning through post-processing (e.g., map visualization or neuron labeling, covered in Chapters 4 and 6, respectively).

- A *supervised SOM* [146] includes classification attributes in each $\vec{z}_s$ and $\vec{w}_{yx}$, and uses these attributes throughout training. This approach is named somewhat misleadingly, because such SOMs only treat example classification as an additional data dimension to be approximated, and are not true supervised ANNs.

- *Semi-supervised SOMs* [139] include classification attributes in every $\vec{z}_s$ and $\vec{w}_{yx}$. However, these attributes are not used in Equation (3.2) to find the BMU, but are used for the weight vector update of Equation (3.4). The SOM thus learns the classification attribute distribution, without biasing the learning process.

The purely unsupervised method is more common within the literature, while the two last-mentioned approaches are less prevalent, and thus not focused on within this work.

3.4 Emergence in SOMs

In situations where sufficient neurons constitute a map, a SOM may exhibit emergent characteristics [241]. The general concept of emergence is discussed in Section 3.4.1, while the requirements for, and characteristics of non-emergent and emergent SOMs are elaborated upon within Sections 3.4.2 and 3.4.3, respectively. Section 3.4.4 discusses the notion of interpolating units, which are closely associated with emergent SOMs.

3.4.1 Emergent Systems

Various complex systems exist, which consist of many elementary processes interacting with one another. While the behavior of these elementary processes is not very complex, new higher level phenomena may become visible. The appearance of such phenomena is termed emergence [126, 232]. Emergent systems may be natural (e.g., cloud formations, bacterial colonies, and even crowds of people) or man-made (e.g., lasers and masers).

3.4.2 Non-Emergent Feature Maps

Because each neuron's weight vector is the centroid for a group of input examples, there is a lower bound on the number of map neurons. This bound is the number of linearly independent training set examples, which is an expression of the number of clusters in the data set. SOMs with this minimum number of neurons are called *non-emergent feature maps*, and their clustering behavior is similar to that of k-means clustering [80, 240].

3.4.3 Emergent Feature Maps

SOMs that display emergent behavior are referred to as *emergent feature maps* [80, 241], and require many more neurons than the number of expected clusters (the implied lower bound on the number of neurons). An implied upper bound on the number of neurons is P_T, the number of training examples. A greater number guarantees neurons that cluster no examples, and negates some of the complexity reduction discussed in Section 3.1.2. This particularly hampers SOM-based DM, which is discussed in Chapter 7. Generally, a problem dependent optimal upper bound, βP_T, exists [68], where $\beta \in (0, 1]$.

In emergent feature maps, dense macro-level groups of neurons (called emergent clusters) may form. Emergent clusters have the following general characteristics:

- Weight vectors within clusters tend to be mutually similar, while weight vectors from different clusters tend to vary. Therefore, inter-vector distances within clusters tend to be smaller, while those between clusters are typically larger [244].
- A high absolute number of data examples from the training data set, or a set with similar characteristics, tend to have BMUs within emergent clusters. Very few examples tend to fall in the boundaries between clusters [274].
- Data examples with similar characteristics, which usually also have similar classifications, tend to have BMUs within the same clusters. Data examples that have BMUs in different clusters tend to have dissimilar characteristics.

Emergent clusters have characteristics that differ from, and are often superior to, classical clustering methods [240]. Advantages offered by emergent SOMs include:

- The shapes of emergent clusters are often relatively complex in contrast to the often convex clusters that are discovered by classical clustering algorithms [222].
- Emergent clusters represent a faithful model of the input data, because no biased domain knowledge (such as the number of desired clusters) is required.
- They tend to be robust in the face of noisy data. Their performance degradation due to such noise is typically more graceful than that of other CI algorithms.
- Typically, outliers (noisy training examples differing greatly from the overall data distribution) affect very few neurons, not changing the map's overall model.
- Emergent SOMs even tend to perform adequately when trained on incomplete data sets, within which attribute values are missing from training set examples.

These characteristics are all very advantageous in the context of EDA and DM exercises, and emergent feature maps are almost exclusively used in practical settings. As a result, emergent feature maps are assumed for the remainder of this dissertation.

Global ordering of a trained SOM was discussed in Section 3.1.2. In emergent feature maps, global ordering is observable when each set of homogeneous weight vectors occupies only a single emergent cluster. Such emergent clusters make map interpretation easier.

3.4.4 Interpolating Units

Another phenomenon that is associated with emergent feature maps is the presence of *interpolating units* [185]. Interpolating units constitute a special category of map neuron, where the associated weight vector of each interpolating unit is positioned far from all the other neurons making up the map grid, including other interpolating units.

Interpolating units usually do not constitute the BMU of any data examples contained within the training data set. This is also generally the case for any other data examples that are selected from the same distribution as the original training data. Interpolating units are typically located between emergent clusters, and are responsible for producing a smooth transition between emergent clusters. As a result of these characteristics, interpolating units also delineate boundaries that surround emergent clusters.

It is also important to note that interpolating units do not model any part of the original training data distribution. In this sense, therefore, interpolating units constitute a type of noise that is introduced into the data model represented by a SOM grid.

3.5 Training Parameters

Given the SOM's structure and training, several parameters affect performance. The relevant parameters can be grouped into those affecting map dimensions, those related to the learning rate, and those that affect the neighborhood radius. All parameters must be optimally set before training, and are elaborated on under Sections 3.5.1 to 3.5.3.

3.5.1 Map Dimensions

The number of rows, Y, and columns, X, of the SOM clearly affect the map shape. For example, a rectangular map is often preferable for visual interpretation [146]. More importantly, however, these parameters change the number of neurons in the map.

The number of neurons has implications for SOM training [68, 200]. Too few neurons produce poorly performing maps, with high inter-neuron variance within emergent clusters. Maps with too many neurons are highly time complex, and have many redundant neurons that do not model the training data, and to which no data examples map.

Section 3.4.3 introduced theoretical upper and lower bounds on the number of map neurons, and thus Y and X. However, optimal values are normally found by means of trial and error or an exhaustive search. It is also possible to use one of the growing SOM techniques discussed in Section 3.7, in order to adaptively find a good balance.

3.5.2 Learning Rate

Assuming that the neighborhood function is a Gaussian kernel, the learning rate, $\eta(t)$, adjusts the height of the kernel, and thus the size of the weight adjustments across the map. If a SOM is thought of as performing an optimization within an error space as it trains, $\eta(t)$ is proportional to the size of "jumps" taken through the error space.

A large value for $\eta(t)$ results in a higher kernel, and thus larger weight adjustments per iteration. Conversely, a lower $\eta(t)$ value produces a shallower kernel, and a smaller weight adjustment each iteration. This is shown comparatively in Figure 3.7 (a) and (b). It should be noted that the area of the map affected by an update is not changed.

A high $\eta(t)$ value causes rapid learning as weight vectors are quickly pulled into a crude approximation of input data's form, but finer map structure adjustments are not made, and an optimally performing map may be "overshot". A lower $\eta(t)$ causes less drastic adjustments, thus fine-tuning the map structure. Training will take many iterations, however, because the size of each weight adjustment is small.

For the SOM to converge, it is necessary that $h_{ba,yx} \to 0$ as $t \to \infty$ [146]. It is clear that $\eta(t)$ affects the magnitude of $h_{ba,yx}$. This means that $\eta(t)$ should be a monotonically decreasing function of t, resulting in a map that trains quickly initially, but is refined once an initial structure is found. An exponential decay function [211] may be used:

$$\eta(t) = \eta(0) \cdot e^{-t/\tau_1} \tag{3.11}$$

where $\eta(0)$ denotes an initial learning rate which a large value, t denotes the current training iteration of the weight optimization procedure, and τ_1 is a positive constant.

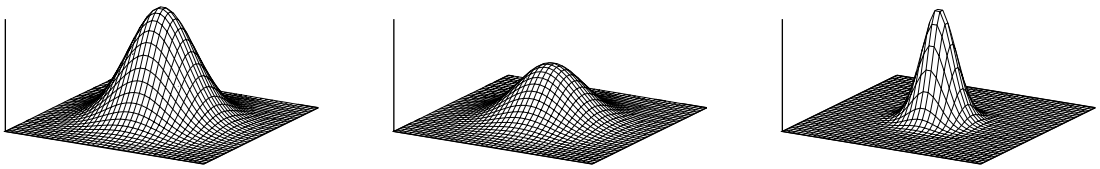


Figure 3.7: The general effect of parameter values on the shape of the smooth Gaussian kernel: (a) shows the original kernel; (b) shows the original kernel with a reduced learning rate; (c) shows the original kernel with a reduced kernel width.

The value taken on by τ_1 constitutes yet another parameter value that must be set before training commences. A larger value for τ_1 results in a slower rate of decay for the learning rate, while a smaller value conversely causes a faster decay rate. The form of this decay function, and the effect of a modification to τ_1, are shown in Figure 3.8 (a).

3.5.3 Neighborhood Radius

When a smooth Gaussian kernel is used as a neighborhood function, the value of the kernel width, $\sigma(t)$, modifies the radius of the neighborhood. The kernel width affects the area around the BMU over which a weight vector update is effective, and consequently the number of weight vectors that are modified with each training step.

A higher value for $\sigma(t)$ results in a wider kernel. This causes a larger number of weight vectors to be affected by a weight update step. Conversely, a lower $\sigma(t)$ value results in a narrower kernel, and a tighter area of effect on fewer map weight vectors. Figure 3.7 (a) and (c) illustrate the comparative effect this value has. It is important to note that the extent of the map area affected by an update is not adapted.

A larger $\sigma(t)$ value causes many weight vectors to be pulled into a rough cluster grouping. As a result, clustering takes place quickly as training progresses, but forms groupings that are relatively unrefined in structure. A smaller value for $\sigma(t)$ conversely causes the effect of a weight update to be localized to a very small part of a larger cluster, therefore modifying the finer structure of the map. These finer-grained updates are not ideal for the formation of initial clusters, however, because the overall cluster structure would appear only after a large number of weight adjustments have taken place.

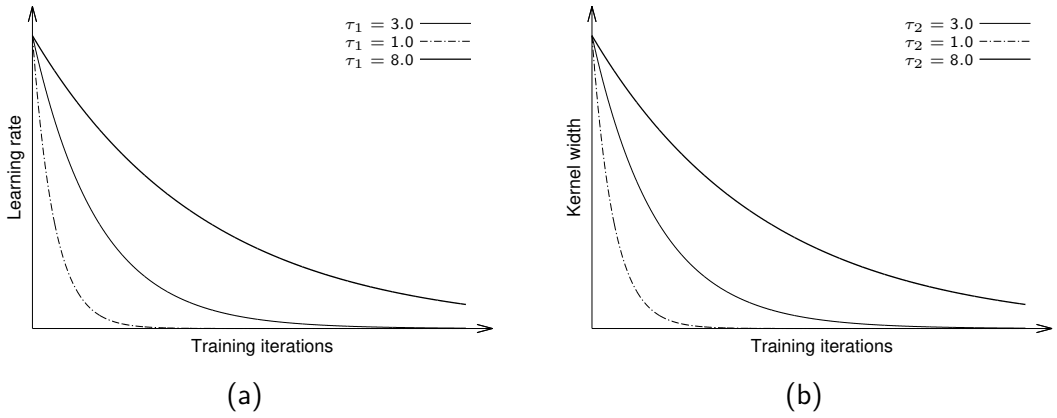


Figure 3.8: Exponential training parameter decay functions: (a) shows learning rate decay, with varying τ_1 values; (b) shows kernel width decay, with varying τ_2 values.

As was the case for the learning rate discussed in the previous section, $\sigma(t)$ must also be a monotonically decreasing function of t, in order to ensure convergence of the map. This results in the quick formation of rough clusters, before their finer structure is refined over time. It is possible to use the following decay function [211]:

$$\sigma(t) = \sigma(0) \cdot e^{-t/\tau_2} \tag{3.12}$$

where $\sigma(0)$ is a large initial kernel width, t is the current training iteration, and τ_2 is a positive constant. The value of τ_2 is also a parameter that is specified prior to training, where a larger τ_2 value gives a slower decay. Equation (3.12) is an exponential decay function, with the same general form as Equation (3.11). The kernel width decay function's shape, and the effect of a τ_2 modification, are shown in Figure 3.8 (b).

3.6 Factors Affecting SOM Model Accuracy

The data model embodied in a trained SOM is considered accurate if the weight vectors making up the map structure closely mimic the characteristics of the underlying training data. It is possible to measure aspects of model accuracy in two ways: firstly by means of the training quantization error defined in Equation (3.7), and secondly using one of the topological error measures discussed in Section 3.3.3.4. This section focuses on several factors that have an influence on the final accuracy of a SOM-based data model.

The first factor affecting SOM model accuracy is the configuration of the parameters outlined in the previous section, namely the map dimensions, the initial learning rate and neighborhood radius, and the decay constants for the learning rate and neighborhood radius. The optimal settings for these parameters are problem dependent, and must therefore be optimized prior to using the SOM model in any way. Several approaches to parameter optimization are possible, all of which require repeated simulations on either the full training data set or a representative subset thereof. Section 8.3.4 outlines the parameter optimization procedure that was followed during the reported experimental work, and which involved all the previously discussed SOM parameters.

The map size has a particularly important effect on SOM model accuracy. This is because a map that is too small for the training data set has fewer neurons, which requires each weight vector to serve as a centroid for a larger number of data examples. Because these data example sets will be less homogeneous, they will be less accurately represented by their BMUs, and map quantization error will increase.

Finally, the "curse of dimensionality" [16, 19, 216] poses several problems for high-dimensional data. For a fixed training set size, the predictive power of a learned model drops as data dimensions increase [115]. The characteristics of low-dimensional and very high-dimensional data spaces also often differ [59]. Many distance measures (particularly Euclidean distance) behave unexpectedly when used in a variety of very high-dimensional data distributions [2]. In such cases, the ratio of the distances between a point and its nearest and furthest neighbors approaches unity, and the differences between inter-point distances becomes less meaningful. Algorithms based on inter-point distance measures (such as the SOM) suffer when distance measures become impaired in this way.

3.7 Variations on the Stochastic SOM

While this dissertation focuses on the basic stochastic SOM, a number of variations and improvements to the algorithm have been proposed. This section elaborates on the most important of these variations. Each of these approaches is based on the premise and architecture of the stochastic SOM, but modifies or extends an aspect of the approach in some way. Section 3.7.1 discusses the neural gas algorithm, while Section 3.7.2 describes

batch training of maps. Growing map implementations are covered in Section 3.7.3, while Section 3.7.4 outlines clipping of the neighborhood function. Section 3.7.5 investigates optimized BMU searches, and Section 3.7.6 describes competitive learning using multiple SOMs. Finally, Section 3.7.7 overviews hardware-based SOM implementations.

3.7.1 Neural Gas

The *neural gas* method [165] is closely related to the stochastic SOM, but training is not constrained by the lattice topology. The weight update equations are very similar to those in Section 3.3. However, while the standard neighborhood function is computed using the distance between c_{yx} and c_{ba} in map space, the neural gas neighborhood is a function of the number of weight vectors closer to $\vec{z}_s$ than $\vec{w}_{yx}$. The map topology also develops during training, with neighborhoods of adjacent neurons adaptively forming and breaking down. This is achieved by connecting the two weight vectors closest to the current $\vec{z}_s$, and progressively weakening the connections that are not renewed.

3.7.2 Batch Training

A *batch map* [146] differs from a stochastic map in that weight values are not updated after every learning iteration, but are instead updated only after all training examples have been presented. This results in much faster training, because the cost of complex weight updates at the end of each iteration is reduced to a single update every epoch. Additionally, it has been observed that it is possible for a stochastic SOM to repeatedly un-learn and re-learn useful data set characteristics. These counter-productive weight vector adjustments are subsumed into a single update step by the batch map.

3.7.3 Growing Maps

One of the major problems when training a SOM is that the optimal size of the map, described in terms of the Y and X parameters, is not known *a priori*. A growing map implementation solves this problem by using an initially small map, and incrementally increasing the map size. The simplest approach adds entire rows or columns of neurons to a rectangular map, thus maintaining the regularity of the map structure. More complex

approaches add individual neurons, which lead to irregular map topologies. Finally, it is also possible for maps to grow in complex hierarchical structures, where parts of maps are expanded into sub-maps. In the last-mentioned case, it is naturally possible for the sub-maps to take on either regular or irregular topological structures.

The most common growing approach that maintains a regular grid structure is the growing grid [86]. The technique maintains a count per neuron, which represents the number of times the neuron has been a BMU. After a number of training iterations proportional to the number of neurons in the map grid, the neuron with the highest count is chosen. The neighboring neuron with a weight vector furthest from the weight vector of the neuron with the highest count is chosen, and a row or column is inserted between the two neurons. The newly inserted row or column weight vector values are interpolated from their neighboring row or column weight vectors, respectively. The map grid continues to grow until either a maximum number of neurons is reached, or the number of training examples per neuron drops below a certain threshold.

The growing hierarchical SOM [203] maintains a regular lattice structure, but builds a hierarchical structure of connected sub-maps. A top-level neuron, initialized to the mean of the training data set, is the root of the recursively defined hierarchy of sub-maps. A new sub-map of 2×2 neurons is created under the root neuron. Normal SOM training proceeds in cycles of a fixed number of iterations on all sub-maps in which the mean quantization error over all BMU neurons is at least a chosen fraction of the quantization error of the sub-map's parent neuron. After a training cycle, the neuron with the highest quantization error in relation to its mapped training examples is selected, and a new row or column (with weights interpolated from neighboring rows or columns, respectively) is grown between the selected neuron and its most distant neighbor in weight space. Once training and growing has ceased for a sub-map, new sub-maps are created as children of any neurons for which the quantization error is at least a pre-defined fraction of the quantization error of the root neuron. Sub-maps are initialized relative to the neighbors of the parent neuron, and train on the data examples mapped to the parent.

Several more complex approaches exist for growing irregular two-dimensional maps through the addition of individual neurons to map lattices. Such maps are still relatively straightforward to represent and understand, but the connections in the lattices tend to

directly represent emergent clusters in the training data space. The most commonly used techniques that fall within this category include the following:

- Incremental grid growing [20] begins with four neurons connected in a rectangular lattice. Each neuron has an error value that is updated whenever the neuron is a BMU, by the square of the distance between the BMU weight vector and the training example under consideration. After a fixed number of training iterations, the border neuron with the highest error value grows neurons in all unoccupied neighboring lattice positions. Connections are created between non-neighboring neurons that are sufficiently close to one another in weight space. Neurons that are connected, but are far from one another in weight space, are disconnected.

- The growing SOM [4] begins with a four-neuron square grid. The neighborhood is clipped to only the BMU and directly adjacent neurons, and the learning rate is linked to the number of neurons in the map. An error value is associated with each neuron, to which the difference between the current training example and the neuron weight vector is added each time the neuron is a BMU. When a neuron error value exceeds a growth threshold and the neuron is on the grid boundary, neurons are added in every available neighboring position, otherwise the neuron weights are distributed to the neighbors of the neuron. New neuron weight vectors are initialized to maintain the local order of the weight vectors around their parent neuron. Periodically during map growing, the learning rate must be reset and redundant neurons must be pruned. After a period of map growing, a smoothing phase is entered, during which no additional neurons are added and the map stabilizes.

- The high-dimensional growing self-organizing map [7] extends the growing SOM method. Growth is preceded by a calibration phase of several epochs, in which only BMUs are trained. Training continues during the map growing phase, with the neighborhood including only the BMU and BMU neighbors (as in the growing SOM). A modified growth threshold, which decreases over time to the highest neuron error value recorded in the calibration phase, avoids growth inhibitions or map structure anomalies intrinsic to the growing SOM. Two smoothing phases are used, the first with larger learning rate and kernel width values than the second.

Adaptive hierarchical incremental grid growing [169] is similar to the previously discussed growing hierarchical SOM technique, but uses incremental grid growing to train irregular sub-map lattices. Sub-map weight vectors are random perturbations of the parent neuron weight vector. The neighborhood function uses connectivity between neurons, rather than simple map space Euclidean distance. Furthermore, there is a fine-tuning phase of training after map grid growing has ceased and before sub-trees are generated. During the fine-tuning phase the neighborhood only includes the BMU.

It is also possible to grow irregular lattices that are multi-dimensional. Such maps are intrinsically difficult to represent and interpret, and are therefore less commonly used (particularly for EDA purposes). Techniques in this category include the following:

- The growing cell structures method [85] initializes a simplex of neurons in a chosen dimensionality. Neurons have continuous valued counters, indicating either how many times each neuron has been a BMU during training, or the cumulative quantization error of BMU matches. Counters decay after each training iteration, and are normalized in relation to all counters across the map. Periodic neuron insertions ensure that the network topology still consists of only simplices with the initially chosen dimensionality. An insertion occurs between the neuron with the highest normalized counter and that neuron's furthest neighbor in weight space. The counters of the three neurons involved in the insertion are then re-computed. Local estimates of the data space probability density function around weight vectors are made at intervals, and neurons with low estimates are pruned.

- The hypercubical growing SOM [15] starts with two neighboring neurons, and periodically either adds a neuron in one of the existing lattice dimensions, or adds neurons to form a new lattice dimension. For each neuron, the reconstruction error of the receptive field is computed, given the neuron's weight vector, and the error is decomposed along the possible dimensions of growth. Growth then takes place in the dimension with the largest average normalized error coefficient over all training examples. Growing ends once the map contains a specified number of neurons.

Finally, it is also possible to apply similar concepts of network growing to the neural gas algorithm that was discussed in Section 3.7.1. The growing neural gas technique [87]

extends the neural gas algorithm, by allowing the number of neurons in the network to increases as training progresses. The growing neural gas and growing hierarchical SOM approaches have also been combined to form the dynamic adaptive self-organizing hybrid model [117], which allows hierarchical growing of neural gas networks. This model was developed for text clustering, but is general enough for broader application.

3.7.4 Neighborhood Clipping

Another optimization that can be applied to the SOM training procedure is based on *clipping the Gaussian neighborhood* at a certain deviation from the mean of the neighborhood's kernel [68]. This optimization is possible because, while weight vector updates occur across the *entire* map, only those in the vicinity of the BMU have an important effect. Conversely, weight vector adjustments that occur very far from the BMU tend to be minor, and have a negligible effect. Neighborhood clipping will result in no weight adjustments beyond a certain threshold distance from each BMU, and improves performance because of the elimination of unimportant updates to weight vectors.

3.7.5 Optimized BMU Searches

Variations on the search for the BMU are also possible. *Shortcut BMU search* approaches aim to reduce the computational complexity of the SOM training algorithm by eliminating a number of the distance comparisons that are used to find a BMU.

Kohonen *et al* [149] propose limiting the search to the neighborhood of the previous BMU for the training example in question. This approach affects the weight updates only slightly, because a SOM maintains the topological structure of the input space, thus increasing the likelihood that the current BMU of a training example will be in the general vicinity of the previous BMU for the same training data example.

Another BMU search optimization, developed by Kaski [129], divides the map neurons into nested subsets. A tree-like search is performed when a BMU must be determined, starting with the largest sets and focusing on successively smaller contained sets. This approach is still guaranteed to find the correct BMU, but eliminates a large number of distance comparisons that are likely to be fruitless, thus speeding the search up.

3.7.6 Competitive Learning

It is possible to use multiple SOMs in conjunction with one another [33]. In such a case the learning procedure becomes a competitive process between the SOM networks, and each map approximates a subset of the training data. As a result of the complex interactions between such networks, interpretation becomes more difficult.

3.7.7 Hardware Implementations

In addition to the above basic variations, the reader's attention should be drawn to a number of hardware-based implementations of the SOM. Figure 3.9 shows an example of such a hardware-based SOM implementation, namely the NBISOM_25 chip [208]. Hardware-based solutions are usually intended for application in very high-performance, real-time environments, which are not typical within an EDA or DM setting. Such implementations are thus of little interest in the context of this research, although a good overview of related work is given in Kohonen's summarizing text [146].

3.8 Summary

This chapter examined a variety of important aspects relating to the SOM (or self-organizing feature map) algorithm. Section 3.1 gave a brief overview of the conceptual underpinnings of the algorithm, namely the self-organizing structures of the human brain. Section 3.2 laid out the data structure architecture that is required by the algorithm. Section 3.3 discussed the phases of the stochastic training algorithm, namely initialization, weight adjustment and training termination. Section 3.4 outlined the basic concept underlying emergent systems, and described the characteristics of emergent feature maps, as well as the requirements for the formation of emergent feature maps. Section 3.5 looked at the parameters of the stochastic algorithm, which include the dimensions of the map structure, the learning rate, the neighborhood radius, and decay constants for the learning rate and neighborhood radius. Section 3.6 investigated factors that affect the final accuracy of a trained SOM. Finally, Section 3.7 discussed some of the most important variations on the standard stochastic SOM algorithm and architecture.

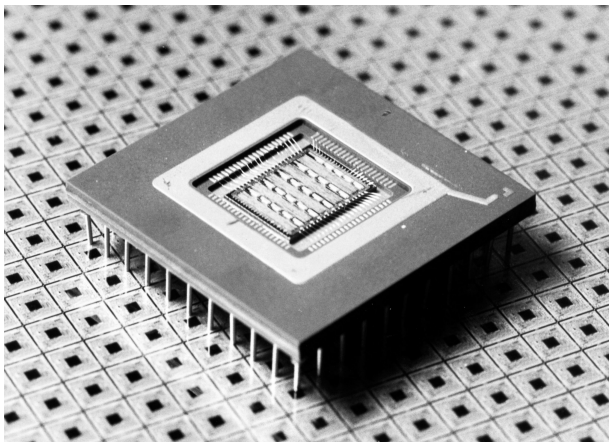


Figure 3.9: The NBISOM_25 chip, an example of a hardware-based implementation of the SOM algorithm. Image and permission for reproduction provided by Dr. Mario Porrmann.

The following chapter considers the most important classes of techniques that are available for visualizing a SOM's map structure. These methods visualize the data model represented by a trained SOM and, by implication, also represent the training data set that the visualized SOM is based on. Examples of specific techniques are also discussed. In addition, an outline is provided of the general types of SOM-based EDA.

Chapter 4

SOM-Based Visualization and Exploratory Data Analysis

The previous chapter discussed the SOM algorithm in detail. This chapter focuses on methods for visualizing SOM structure. Because EDA focus on visual representation and interpretation, the main categories of SOM-based EDA are also outlined.

Because SOMs are widely used for visual data exploration, there are a vast number of SOM-based visualization and EDA methods. This chapter therefore covers only very common techniques and those related to the discussions in Chapters 5, 6, and 7.

Section 4.1 outlines traditional data visualization. Section 4.2 introduces the philosophy underlying SOM-based visualization. Sections 4.3 and 4.4 respectively discuss grid-based and irregular representations for SOM maps. Section 4.5 briefly considers SOM-based EDA. Finally, Section 4.6 provides a summary of the chapter.

4.1 Traditional Data Visualization

Data visualization aims to provide mechanisms for the accurate visual representation of data sets. Due to unstructured data's complexity, data visualization usually assumes table-based structured data sets, which are described in Section 2.1.1.2.

Traditionally, the field of data visualization uses *information visualization techniques*, each of which is a computer-based method of visually representing abstract data [27]. In

general, these methods represent each tuple as a *mark* on a visual display. Each mark, in turn, has several *visual dimensions*. A visual dimension is any visual feature that can be mapped to an attribute value's numeric magnitude. Possible visual features include textual labels, color or grayscale values, point size, and shape. Some common traditional information visualizations are scatter plots, line charts, and bar graphs.

As database volume and dimensionality has increased, information visualization has focused more on managing data set complexity. Data sets with huge numbers of tuples or tuple dimensions cause several problems for traditional visualization methods:

- The computational complexity of the visualization can become problematic. Some complex techniques may even become unusable if they exceed the available computational resources, such as primary memory.

- Visualizations of many tuples can become too dense to interpret easily, due to the resultant proliferation of visual marks that can overlap and obscure one another. This is particularly problematic if the visualization's space is limited.

- High dimensional tuples can be problematic, because most traditional data visualizations can represent only a limited number of dimensions. Multiple visualizations may therefore be needed to represent an entire data set.

Some contemporary information visualizations, such as RadVis [110], attempt to solve one or more of these problems in various ways. However, a more general solution is to reduce the *overall complexity* of the represented data. Visualizations using nonlinear dimensionality reduction methods, including SOM-based visualizations, achieve this goal. While alternate nonlinear dimensionality reduction visualizations are not the focus of this research, Gisbrecht and Hammer provide a thorough survey of these methods [94].

4.2 The Philosophy of SOM-Based Visualization

The remainder of this dissertation illustrates examples using a purely unsupervised SOM with a hexagonal lattice, trained on the Iris flowers data set [3]. Section 8.2.1 describes the data set and its pre-processing in detail, and only the most pertinent aspects are

mentioned here. The data set describes plants and has four continuous attributes, namely `sepal_length` in the range $[4.3, 7.9]$, `sepal_width` in the range $[2.0, 4.4]$, `petal_length` in the range $[1.0, 6.9]$, and `petal_width` in the range $[0.1, 2.5]$. Nominal data example classifications are either `Iris_Setosa`, `Iris_Versicolor`, or `Iris_Virginica`.

As Section 3.1.2 described, a SOM models a high-dimensional data set using a grid with fewer dimensions. A SOM's training data is approximated by a map structure that is essentially characterized by a set of weight vectors. However, as Figure 4.1 illustrates using the Iris data set SOM, raw weights are usually difficult to interpret.

Because a SOM models its training data, the map structure of an adequately trained SOM (that is locally and globally ordered) should have overall characteristics that are similar to the characteristics of the training data. Therefore, *a visualization of a trained map should provide insight into the modeled data.* Because a SOM is defined by its constituent weight vectors, most SOM visualizations are based on these vectors.

A general discussion on SOM-based visualization is difficult, due to the sheer number of applicable techniques. Therefore, in order to organize this discussion, this research work proposes a two-level approach to understanding visualization techniques:

- Firstly, visualizations are based on *map representation frameworks*, of which there are two broad categories. The most common map representations are grid-based, and are discussed in Section 4.3. A variety of less commonly used representations are characterized as irregular, and are investigated in Section 4.4. As is the case for all information visualizations, map representations consist of visual marks. This taxonomy of map representation frameworks is illustrated in Figure 4.2.

- Secondly, *visual augmentations* can be added to a map representation. An augmentation links a map characteristic to a visual dimension of all marks on the map representation. To reduce complexity, thresholds may be represented (e.g., "high" or "low" values). Several augmentations may be combined by linking each to a different visual dimension [99]. While all augmentations are not sensible in every map representation's context, Figure 4.3 shows a general taxonomy of augmentations.

Visual augmentations are an almost essential addition to grid-based map representations. Consequently, Sections 4.3.1 to 4.3.3 discuss augmentations in the specific context of

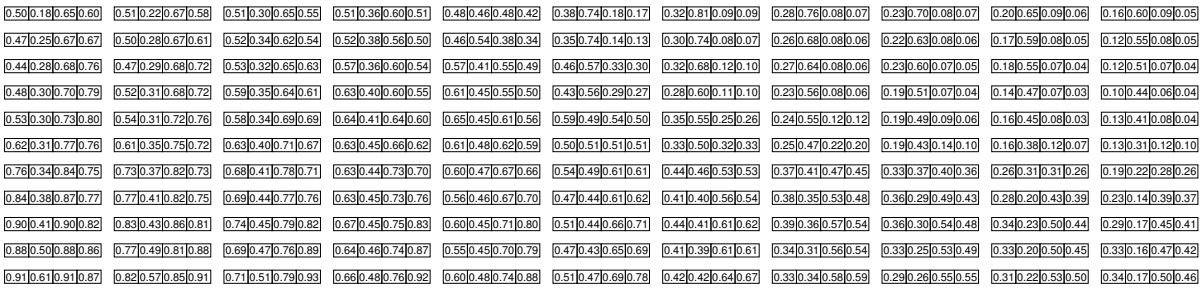


Figure 4.1: The raw weight vectors making up an 11×11 neuron grid of a purely unsupervised SOM trained on the Iris data set [3]. While not shown, the map's lattice topology is hexagonal.

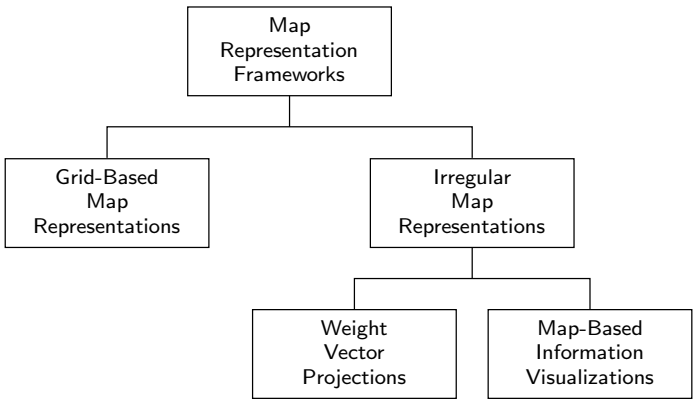


Figure 4.2: A taxonomy of the general map representation frameworks that are available. A representation framework is used as the basis for a SOM-based visualization technique.

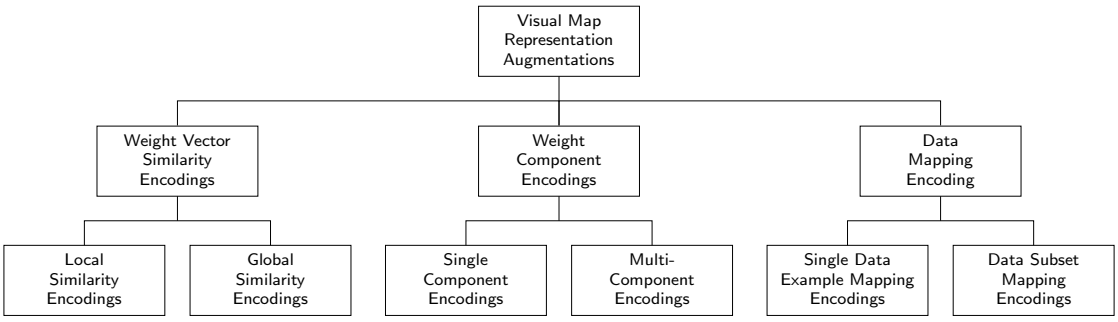


Figure 4.3: A taxonomy of the more prevalent visual map representation augmentations. It is possible to add these augmentations to a chosen map representation framework.

grid-based representations. However, the majority of these visual augmentations are also applicable to the irregular map representations that are described in Section 4.4.

4.3 Grid-Based Map Representations

Grid-based map representations simply denote each neuron as a cell within a regular grid, which has the same number of rows and columns as the SOM's map structure does. It is also possible for a grid-based representation to include intermediary cells between the cells that directly represent neurons. A grid-based representation thus depicts an actual entity, which classifies these representations as *scientific visualizations* [27].

Figure 4.4 shows two possible grid representations of the example SOM of Figure 4.1. The first representation shows no topology information, simply representing the grid with uniform rows and columns. The second grid depicts the map's hexagonal lattice structure, where the neighbors of each neuron are more clearly identifiable.

The following advantages are inherent to grid-based map representations, and contribute to making such representations fairly versatile and commonly used in practice:

- Grid-based representations are general-purpose visualizations. Irregular representations are often designed for a special purpose, which may limit their wider use.
- Grids are compact, unlike many irregular map representations. Grid-based representations are thus easy to display and scale according to an analyst's needs.
- Grids always have a sensible layout. Non-deterministic irregular visualizations, like Sammon's mapping, do not always generate a good structure [128].
- Because their layouts are consistent, different grid visualizations of the same map can be compared easily [131]. This is often not the case for irregular visualizations.
- The uniform grid layout allows arbitrarily complex visual augmentations to be added to the neuron cells, without the risk of overlap between the augmentations.
- It is computationally negligible to display a grid. Irregular representations (particularly weight projections) are often computationally expensive [128].

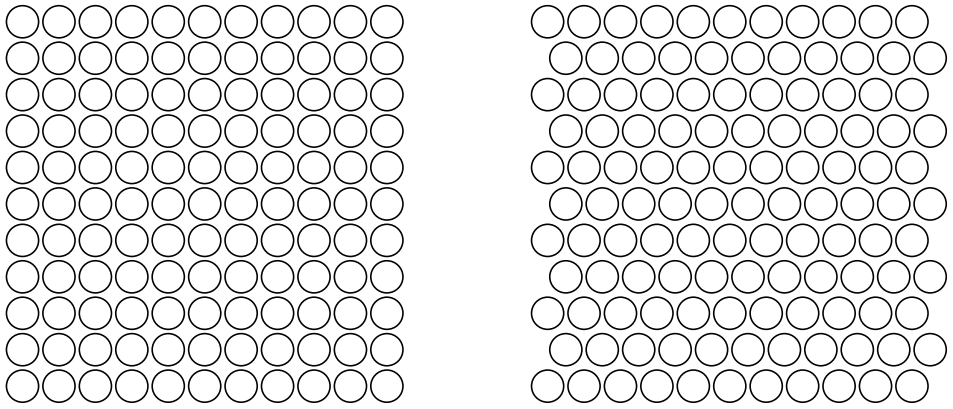


Figure 4.4: Examples of non-augmented grid-based map representations with no intermediary cells, based on the example SOM of Figure 4.1: (a) shows a grid without topology information; (b) shows the map's hexagonal topology. In both representations, circles represent neurons.

Of course, despite the numerous previously mentioned advantages, grid-based representations are not without several associated disadvantages. The most important drawbacks that are intrinsic to grid-based visualizations are each elaborated upon, below:

- The regularity of the grid does not intrinsically represent any information about the map, other than its width and height. Thus, for these representations to be useful, further information must be added via one or more visual augmentations.

- Due to their regular structure, grid-based map representations cannot display a subset of the map's neurons. Because irregular map representations have no constrained structure, they are free to depict any number of a map's neurons.

- No clear interpretation can be attached to either the axes of a map [253], or the relative sizes of map areas. However, due to the regular proportions of a map representation, analysts may be tempted to make such interpretations

For the sake of consistency, the visual arrangement of neurons in a grid-based representation should mimic the lattice topology used during training (discussed in Section 3.2). Figure 4.4 shows two grid-based representations, the first with a rectangular lattice and the second with a hexagonal lattice. These visualizations also reiterate that

the only useful information a non-augmented grid shows is the map's dimensions. This chapter's remaining grid-based visualization examples assume a hexagonal lattice.

Some of the more widely used visual augmentations are discussed in more detail in the following sections: Section 4.3.1 covers weight vector similarity encodings, Section 4.3.2 deals with weight vector encodings, and Section 4.3.3 introduces data mapping encodings. Figure 4.3 depicts these augmentations in the form of a taxonomy.

4.3.1 Weight Vector Similarity Encoded Augmentations

Visual augmentations that show a weight vector similarity encoding attempt to visually represent areas that have similar weight vectors. Such areas are, as noted in Section 3.4.3, considered to be emergent clusters. Traditionally, these types of augmentations use color to denote similarity [120], although any other visual dimension may also be used [99].

4.3.1.1 Local Similarity Encoding

According to Section 3.4.3, emergent clusters may be characterized by inter-weight-vector distances. Inter-vector distances within emergent clusters are usually small, while between emergent clusters inter-vector distances are typically large. This characteristic of emergent maps is exploited by local weight vector similarity encodings, which represent the distances between topologically adjacent neuron weight vectors.

While local similarity encodings may use any distance measure, it is typical to use the same distance measure that was used during training (see Section 3.3.2). Consequently, the majority of local similarity encodings use Euclidean distance, which is used by most SOMs during training. The remainder of this dissertation therefore assumes Euclidean distance as a basis for all local similarity encoded visual augmentations.

Every local similarity encoding augmentation requires a *local similarity matrix*, which stores distance values computed from the map grid's weight vectors. A local similarity matrix differs in form, depending on the exact nature of the augmentation, but always stores at least one value for each neuron. In addition, a local similarity matrix may also contain several intermediary distance values that fall between the neuron values.

After computing a local similarity matrix, its constituent values must be linked to any appropriate visual dimension [99]. It is, however, typical to map each local similarity

value to a color continuum, with most augmentations using grayscale [120]. Grayscale encodings are easy to understand and less prone to ambiguity of interpretation.

It is also possible to produce a summarizing view on local distance similarities, by augmenting visualizations using distance thresholds [170], rather than raw distance values. Such augmentations have less detail, but may be easier for a human to interpret.

The two most common local similarity encoding augmentations are considered below. Both are illustrated in Figure 4.5, using examples based on the SOM of Figure 4.1:

- An *aggregate distance matrix* uses a local similarity matrix to represent only one distance value per neuron. A neuron's distance value is an aggregate of all the inter-vector distances between the neuron and its direct neighbors (as defined by the map's lattice topology). A neuron's aggregate is usually the mean [246], median, maximum [153], or minimum of the neuron's inter-neighbor distances. The advantages of these augmentations are that they are simple to implement and compactly visualize only a single piece of information per neuron. However, two major drawbacks are that much detail is lost (because no inter-neighbor distances are represented), and that the aggregate similarity values can be sensitive to extremely large or small distances (especially near emergent cluster boundaries).

- The *unified distance matrix* (U-matrix) [244] technique is the most prevalent local similarity encoding scheme, and extends the idea underlying aggregate distance matrices. As in an aggregate distance matrix, aggregate values are usually stored for each neuron cell. Intermediate cells are also placed between neighboring pairs of neuron cells, according to the map's lattice topology. Each intermediate cell stores the inter-weight-vector distance between the neurons that it separates. An advantage of U-matrices is that they hide no local similarity detail (aggregate values are not the focus of the augmentation, and mainly serve to smooth the visualization). Minor drawbacks are that U-matrices are more complex to implement, and more visually complex than aggregate distance matrices due to the intermediate cells.

Both the aggregate distance matrix and U-matrix in Figure 4.5 use grayscale encodings, where darker shades denote larger distances and lighter shades are smaller distances. Both maps show two areas characterized by small inter-vector distances, which consti-

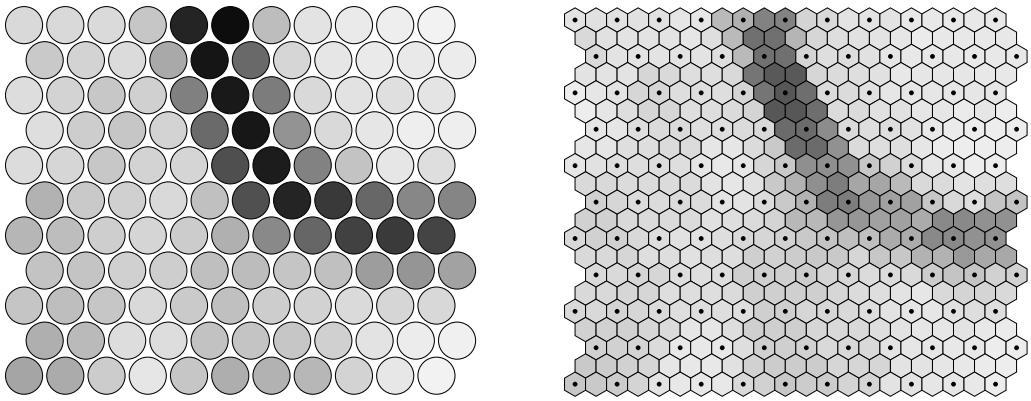


Figure 4.5: Two grid-based map representations of the example SOM in Figure 4.1, with local similarity encodings: (a) shows an aggregate distance matrix, representing mean distances to neighboring neurons; (b) shows a U-matrix, where cells containing dots represent neurons encoded with the mean of the distances to direct neighbors. In both cases, darker shades of gray denote larger distances, and lighter shades indicate smaller distances.

tute emergent clusters (one in the upper right of the map, and another to the left and bottom). The clusters are separated by a band of large inter-vector distances.

Because both aggregate distance matrices and U-matrices compute the same neighborhood structure to calculate their similarity values, neither approach is to be significantly preferred in terms computational complexity. However, due to the above-mentioned major drawbacks associated with aggregate distance matrices, this research recommends that U-matrices should be preferred for local similarity encoding purposes.

4.3.1.2 Global Similarity Encoding

Whereas local similarity encoded maps represent the distances between topologically adjacent neuron weight vectors, global similarity representations uniformly encode sets of neurons that are deemed to be similar to one another in some way.

The most common global similarity encoding approach visualizes neuron labels produced by any of the neuron labeling methodologies discussed in Chapter 6. Within the context of this chapter, it is simply sufficient to understand that neuron labels are textual characterizations that somehow describe the characteristics of each neuron.

For visualization, all map neurons with equivalent labels must be augmented in the same way. An easy approach simply augments each neuron cell with a textual description that is equivalent to the neuron label. It may, however, ease human interpretation of a map if each set of equivalently-labeled neurons instead receives a unique color.

A relatively simple neuron labeling method, discussed in more detail within Section 6.3.2, algorithmically discovers emergent cluster groupings of neurons, and then uniquely labels neurons belonging to each cluster. Commonly used techniques for discovering emergent SOM map clusters include the k-means [161] and SOM-Ward [176] clustering algorithms. It is fairly common for SOM-based EDA tools (such as Viscovery SOMine [51]) to map a visual dimension to these types of cluster-based labels.

Figure 4.6 shows two visualizations of the SOM represented in Figure 4.1, each with color-encoded emergent neuron clusters. Each example used the SOM-Ward algorithm to discover different numbers of emergent clusters. Figure 4.6 (a) shows two clusters, correlating to the structure suggested by the local similarity visualizations shown in Figure 4.5. Figure 4.6 (b) illustrates four cluster groupings, which reveal sub-structures within the larger map cluster. In addition, the two visualizations illustrate that it is possible for different label assignments to produce very different visual representations.

4.3.2 Weight Vector Encoded Augmentations

While the weight vector similarity encoded augmentations of the previous section represent the relationships between the weight vectors of a map, weight vector encodings depict the actual weight components of each neuron's weight vector. In other words, weight vector encodings represent the actual underlying structure of the map.

4.3.2.1 Single Weight Encoding

It is possible to provide useful insight into the data model of a SOM by visualizing the distribution of a single weight value across the entirety of a map structure. The most common single weight encoding is the *component plane* [131], which uses the same weight component within each weight vector to visually encode the neuron cell associated with the weight. Therefore, in essence, a component plane provides a human analyst with a "sliced" view of a SOM's structure, showing one attribute's probability distribution.

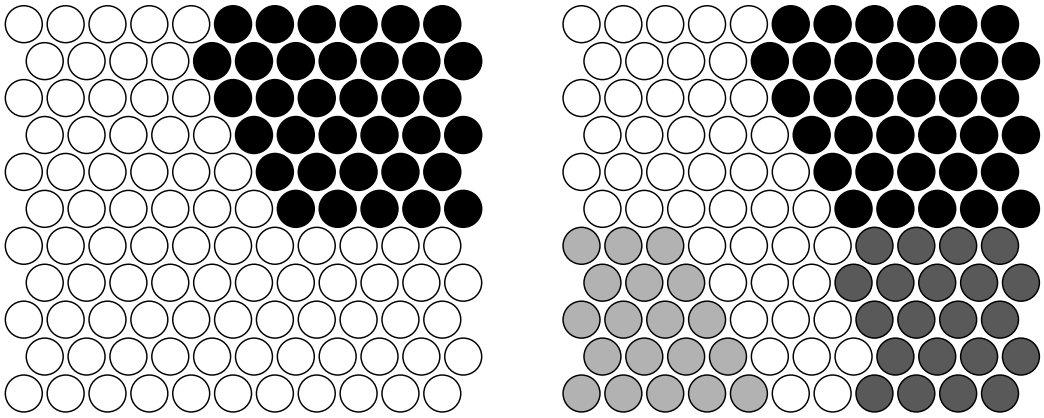


Figure 4.6: Two grid-based map representations of the example SOM in Figure 4.1, with global similarity encodings using grayscale to represent clusters found using the SOM-Ward algorithm of Section 5.3.1: (a) visualizes two clusters; (b) visualizes four clusters.

Figure 4.7 illustrates four grayscale-encoded component planes derived from the example SOM shown in Figure 4.1. One component plane is shown for each of the attributes that define examples from the Iris data set. In the visualizations, higher component values are encoded as lighter shades of gray, while darker shades denote lower values. There are a variety of characteristics, which are illustrated by these visualizations:

- Firstly, each display shows a component's overall distribution across the SOM's data model (e.g., the `sepal_length` and `sepal_width` components have relatively uniform distributions, given the relatively uniform coloring of these planes).

- Secondly, correlations between data components are easily discernible (e.g., the component planes for `petal_length` and `petal_width` are similar, suggesting that these components have similar distributions, and are thus strongly correlated).

- Finally, correlations between components and the map structure can be highlighted (e.g., both `petal_length` and `petal_width` have very low values in the vicinity of the smaller cluster in the upper right corner of Figures 4.5 and 4.6).

As the above interpretations suggest, a drawback of single weight encodings is that they must typically be compared to other visualizations to be useful. Very large numbers

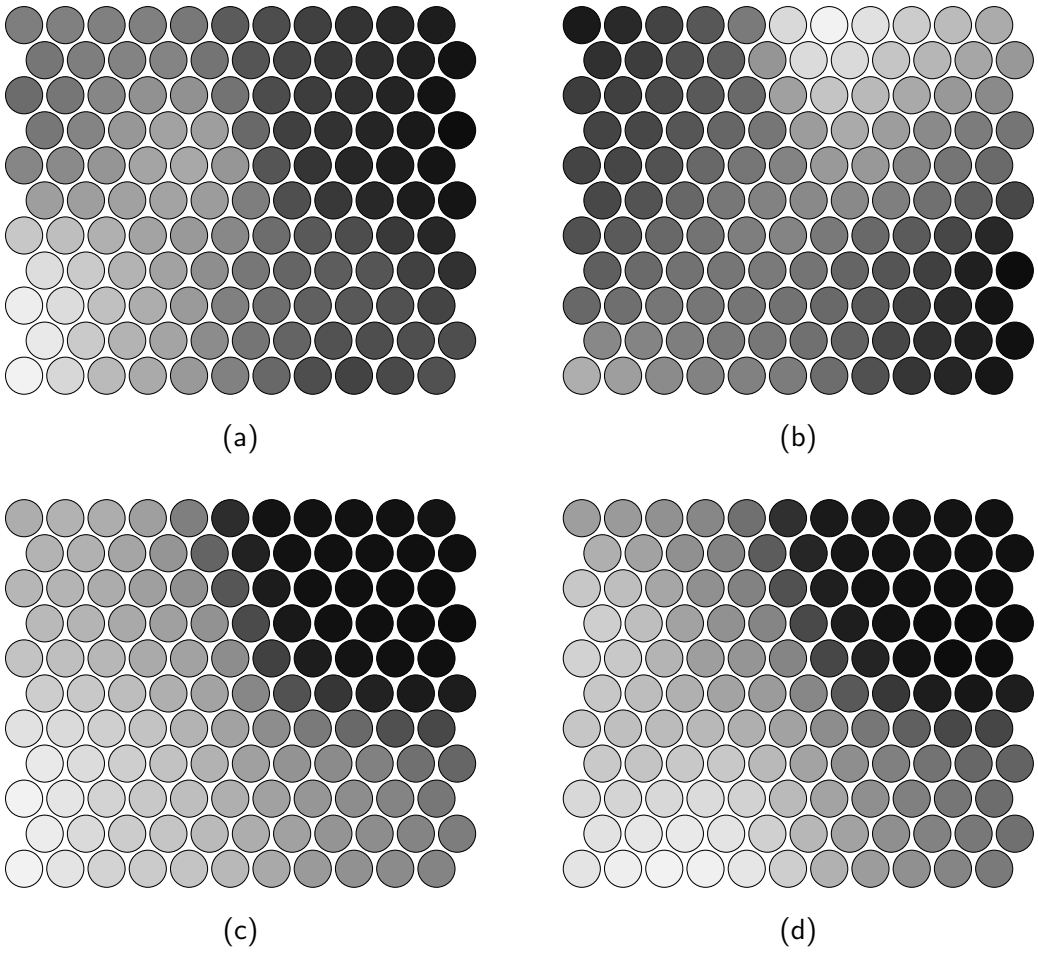


Figure 4.7: Four grid-based map representations of the example SOM in Figure 4.1, each augmented with one weight, where lighter gray shades denote higher values, and darker shades are lower values: (a) shows the `sepal_length` component; (b) shows the `sepal_width` component; (c) shows the `petal_length` component; (d) shows the `petal_width` component.

of attributes produce many component planes, which tend to become unmanageable. Another drawback is that a map's structure depends on each neuron's entire weight vector, meaning that component planes are fairly limited forms of representation.

Of course, there are also a number of advantages that are associated with component plane visualizations. The technique is relatively straightforward and simple to implement, the visualizations are very computationally inexpensive to generate, and the planes are typically fairly easy to interpret when visually analyzed independently.

4.3.2.2 Multi-Weight Encoding

While a visualization of the distribution for a single component is often of interest, such a representation usually provides a fairly limited view on the map's data model. As a consequence, it frequently proves very useful to instead represent several components for every map neuron. This is the aim of multi-weight encoding augmentations.

The most intuitive and widely utilized method for visually encoding multiple weights is referred to as the *glyph map* [99], which augments each neuron cell with a weight vector glyph. In general, glyphs are very compact information visualizations, one glyph representing each data point, which form parts of a larger information visualization [110]. Glyph maps simply employ the same type of glyph, in order to represent a uniform set of weights within each neuron's weight vector in exactly the same manner.

Figure 4.8 illustrates examples of two common glyph maps: Figure 4.8 (a) uses bar graphs [256] to augment each neuron cell of the example SOM from Figure 4.1, while Figure 4.8 (b) employs star plots [31] for the same purpose. In the former case, each bar represents a weight value, where bar height shows a value's magnitude. In the latter case, each ray indicates a weight's value, and ray length indicates value magnitude. Both glyph maps show the overall characteristics of the map's model. It is clear that the weight vectors to the right of the map (particularly those in the upper right area of the map) differ somewhat in structure from those making up the remainder of the map.

Beyond simply bar graphs and star plots, many more exotic glyphs also exist. Each possible glyph has varying associated advantages and drawbacks [110]. For instance, Chernoff faces [34] depict simple human faces, where weight values are mapped to facial feature characteristics (e.g., nose width and length). Chernoff faces exploit the well-known natural human ability to quickly identify and compare faces, but do not facilitate intuitively and exact comparisons of features and relative feature importance.

When the weight vector dimensionality is not too large, it is possible for each glyph to simply represent the entire weight vector of the glyph's associated neuron. If too many weights are present, which will result in the representation of the entire weight vector being difficult or unclear, it becomes necessary to represent only a subset of weights. Of course, deciding which specific weights to include in this type of subset becomes an added complexity for a human analyst to deal with, often requiring additional analysis.

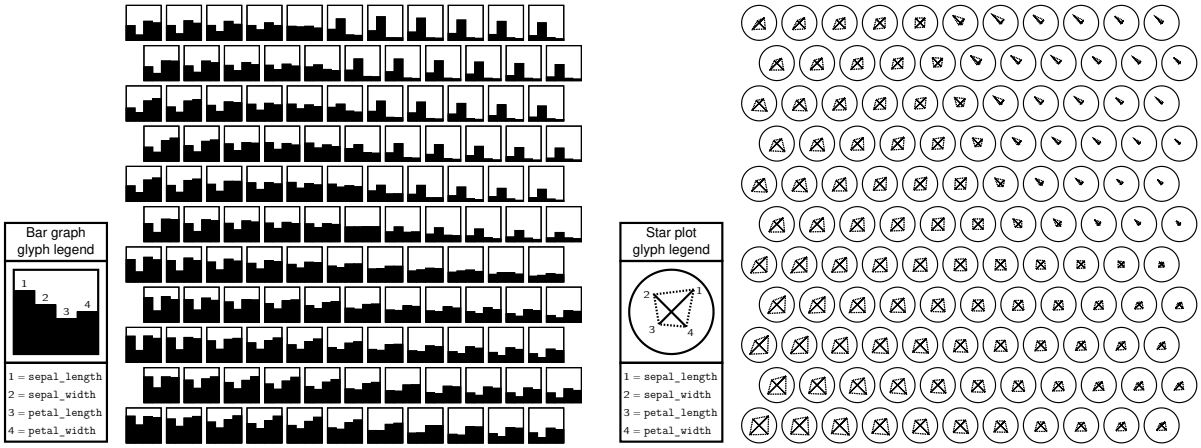


Figure 4.8: Two grid-based map representations of the example SOM in Figure 4.1, each augmented with weight vector glyphs: (a) uses bar graph glyphs; (b) uses star plot glyphs.

Glyph maps have the principal advantage of presenting an easy-to grasp overview of the differences between weight vectors across the map. Of course, due mostly to additional complexity, glyph maps also have some disadvantages, which include the following:

- Glyph maps are relatively complex visualizations to implement in comparison to many of the other grid-based representations for maps.
- Some glyph maps are computationally complex to build. This is the case if the glyphs themselves are complex, or many weights must be represented.
- If the individual glyphs are too visually complex, a compact and understandable overall representation of the map is often difficult to achieve.
- Because glyph maps are dependent on the selected glyphs, the augmentation is subject to any drawbacks associated with the chosen glyph representation.
- For complex glyphs or very high-dimensional weight vectors, it is difficult to discern which specific visual dimension of a glyph maps to a particular weight.
- Because glyph maps are intended for overview map representations, it is often difficult to get an idea of the actual weight value magnitudes that are represented.

The above drawbacks suggest that analysts should be judicious when choosing glyph representations, and should err on the side of simplicity. In general, legends should also be provided, to clearly link each glyph's visual dimension to a weight.

4.3.3 Data Mapping Encoded Augmentations

While the visualization augmentations discussed in Sections 4.3.1 and 4.3.2 represent an aspect of the map's weight vector structure, data mapping encodings visualize the ways in which I-dimensional data examples, denoted $\vec{z}_s$, from a data set relate to the map grid. This data set must have the same structure as the training data.

It is possible for the data vectors to be either data examples drawn from the training set, or data examples that were not presented to the SOM during training. In the latter case, it is typical for an analyst to be interested in the underlying characteristics of new data in relation to the training data that has been modeled by the SOM. This relationship is then determined by studying the characteristics of the map neurons that produce the strongest response for the data example under examination.

Every data mapping encoding augmentation is based on the membership of a data set example to one or more map neurons. This membership is a continuous magnitude usually referred to as the *response* of the neuron to the example, and is typically measured as a distance-based quantity, $res(\vec{z}_s, \vec{w}_{yx})$, calculated as follows:

$$res(\vec{z}_s, \vec{w}_{yx}) = \|\vec{z}_s - \vec{w}_{yx}\|_2 \tag{4.1}$$

where $\vec{w}_{yx}$ is the weight vector of the neuron for which the response is calculated. A smaller $res(\vec{z}_s, \vec{w}_{yx})$ is interpreted as a stronger response, and a larger value indicates a weaker response. The distance measure is usually the same one used during training, thus Euclidean distance is typically used, and the response is called the *Euclidean response*.

4.3.3.1 Single Data Example Mapping

Single data example mapping augmentations depict the neuron response (or responses) generated within a map by a single data example. The data example may be drawn from the SOM's training set or a data set not presented to the SOM during training.

Two main approaches can be used for the representation of single data example mappings to the SOM's map structure. Examples of each are illustrated in Figure 4.9:

- A *BMU mapping* [146] marks the neuron with the strongest Euclidean response (i.e., the example's BMU). These visualizations are easy to interpret, but are often simplistic, as several neurons may have strong responses to an example [217].
- *Response surfaces* [217, 259] visualize multiple responses, by encoding *every* map neuron's response to the example. Response surfaces can identify examples with the characteristics of several emergent clusters, but may be difficult to interpret.

Figure 4.9 (a) shows the mapping of an example to the SOM of Figure 4.1. Figure 4.9 (b) shows the response surface for the same example and map that Figure 4.9 (a) represents.

4.3.3.2 Data Subset Mapping

Data subset mapping visualizations simply extend single data example mapping visualizations, by representing how multiple data examples map to the SOM's neuron grid structure. Once again, the mapped examples may be drawn from the training data set, or a set of examples that was not previously presented during the training process.

Visualizations in this category are based on a data subset mapping matrix, which must be computed prior to visualization. This matrix stores the unified mapping of every example in the data subset to every neuron on the map surface. The simplest and most common approach stores a matrix value for each neuron, which represents the total number of data examples that produced the strongest Euclidean response in the neuron, according to Equation (4.1) of the previous subsection [274]. In other words, each neuron stores the total number of examples sharing that neuron as a BMU.

It is a trivial matter to represent a data subset mapping matrix by means of a visual augmentation. This is achieved by linking any appropriate visual dimension (such as the color attached to a cell) to the unified mapping value computed for each neuron.

A visual encoding of the SOM in Figure 4.1, using varying shades of gray to show the total number of examples that map to each neuron, is shown in Figure 4.10 (a). This type of visual representation is often called a *data histogram*. This sub-figure also illustrates one of the main drawbacks associated with visualizations of data subset mappings: if the

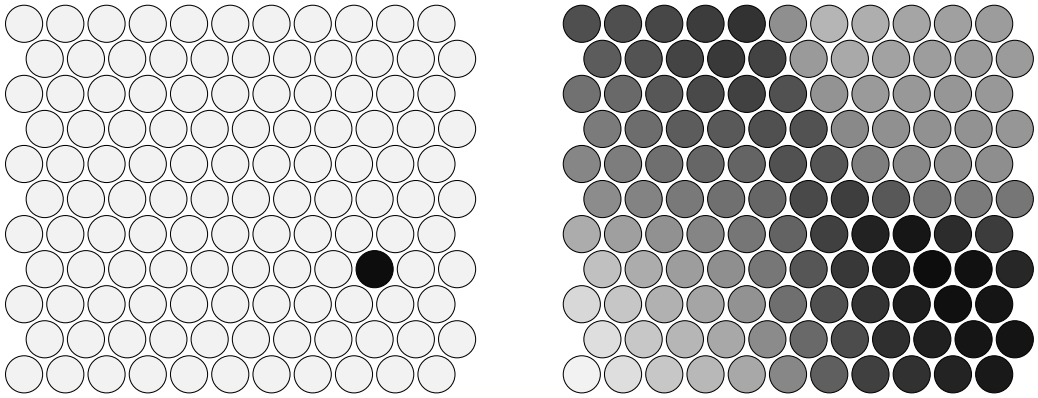


Figure 4.9: Two grid-based map representations of the example SOM in Figure 4.1, each with single data example mapping encodings for an example of the `Iris_Versicolor` class: (a) shows the BMU as a black cell; (b) shows a Euclidean response surface in grayscale, where darker shades indicate stronger mappings, and lighter shades denote weaker mappings.

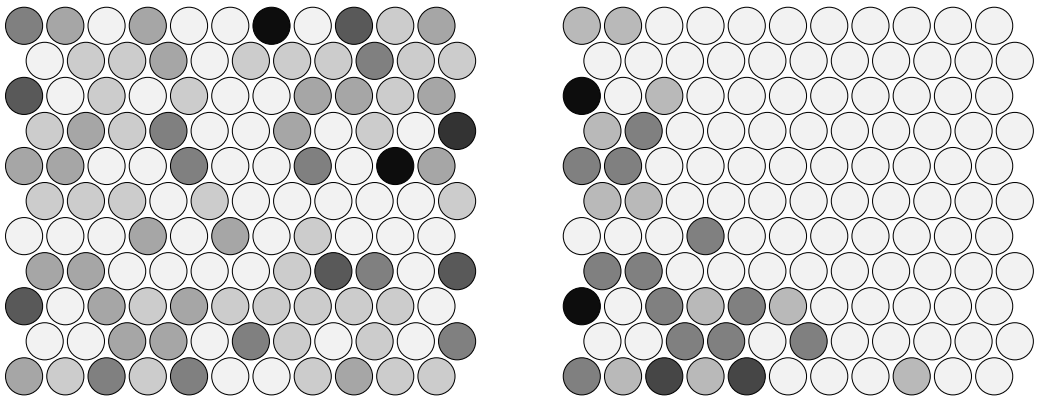


Figure 4.10: Two grid-based map representations of the example SOM in Figure 4.1, with data subset mapping encodings: (a) shows mappings for the entire training data set; (b) shows mappings for training examples classified as `Iris_Virginica`. Darker shades of gray indicate a higher number of mappings, while lighter shades indicate a lower number of mappings.

mapped data examples are sparse in relation to the size of the map, the visualization may highlight no useful examples within the distribution of examples across the map surface. This problem will typically not occur if there are substantially more data examples than the total number of neurons that make up the map's grid structure.

Naturally, it is also possible to generate a data subset mapping visualization with examples of only a particular type with common characteristics. For instance, if an analyst has access to a classified data set, a visualization can be augmented with only the mappings of examples belonging to a particular class. This approach is shown in Figure 4.10 (b), where only examples of the `Iris_Virginica` class are mapped. This example also illustrates that data subset mapping visualizations can be used to show any localization that the mapped data examples have to a particular map area.

4.4 Irregular Map Representations

An irregular map representation is simply any SOM-based visual representation that does not have a regular structure based on the layout of the map grid. There are two types of irregular map representations: weight vector projections and map-based information visualizations, respectively discussed in Sections 4.4.1 and 4.4.2 below.

4.4.1 Weight Vector Projections

Weight vector projections are somewhat more complex to generate and interpret than their grid-based counterparts. SOM-based projections use a lower-dimensional projection space to represent the map's I-dimensional weight space. Each neuron's I-dimensional weight vector, $\vec{w}_{yx}$, is represented by a single two- or three-dimensional projection vector, $\vec{r}_{yx}$. Importantly, the projection vectors are positioned to *visually represent* the relative weight-space distances between all of the corresponding weight vectors.

A projection algorithm post-processes a trained SOM, and arranges the projection vectors for all neurons in projection space. Such a projection is sometimes referred to as a vector quantization-projection (VQ-P) [253]. While many appropriate linear and nonlinear projection methods exist [153], an approach known as *Sammon's mapping* [212] is one of the most commonly used in conjunction with trained SOM grids [148].

The algorithm used by Sammon's mapping is represented, in the form of pseudocode, in Algorithm 4.1. In essence, the algorithm iteratively updates all the vectors making up the projection, with the objective of optimizing an error function. The error function, which is denoted $E(t)$, quantifies how well the projection models the weight vector distribution of the SOM at iteration t of the projection optimization process.

The set of all weight weight vectors contained in the map being represented is denoted *weights*. The error function for Sammon's mapping depends on the set $\mathcal{Y}$, which contains all the subsets of unique weight vector pairs, and is defined as follows:

$$\mathcal{Y} = \{\delta \subseteq weights\} \tag{4.2}$$

where δ is the set $\{\vec{w}_{yx}, \vec{w}_{y'x'}\}$ containing the pair of weight vectors $\vec{w}_{yx}$ and $\vec{w}_{y'x'}$, such that $\vec{w}_{yx} \neq \vec{w}_{y'x'}$. The error function computed at iteration t of the projection vector update procedure, which is optimized by Sammon's mapping, is then defined as:

$$E(t) = \left(\sum_{\delta \in \mathcal{Y}} \frac{\left(\|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2 - \|\vec{r}_{yx}(t) - \vec{r}_{y'x'}(t)\|_2 \right)^2}{\|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2} \right) \div \left(\sum_{\delta \in \mathcal{Y}} \|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2 \right) \tag{4.3}$$

where $\vec{r}_{yx}(t)$ and $\vec{r}_{y'x'}(t)$ are distinct projection vectors that respectively correspond to $\vec{w}_{yx}$ and $\vec{w}_{y'x'}$ at iteration t of the optimization. While Euclidean distance is the most commonly used distance metric in practice, any sensible measure is applicable.

The projection vectors of a Sammon's mapping are often initialized by projecting the weight vectors orthogonally onto a projection space spanned by the weight vectors with the largest variance. This initialization ensures that the projection optimization starts from a state that adequately covers the space occupied by the weight vectors.

Sammon's mapping must optimize the $E(t)$ function using an error minimization technique. Although the original publication describes a gradient-based optimization approach, outlined in Algorithm 4.2, a variety of alternative techniques are also feasible. For example, optimization approaches based on particle swarm optimization [136, 219] and evolutionary computation [65] are feasible for this error minimization.

Sammon's mapping needs one or more stopping criteria in order to cease optimization. The most common criteria include a limit on the number of optimization iterations, and a minimum error threshold. These conditions are usually user-specified, and trade off the projection's accuracy against the time required to generate the visualization.

Train a SOM grid, which will form a basis for the projection
Initialize a low-dimensional projection vector, $\vec{r}_{yx}$, for each weight vector $\vec{w}_{yx}$
Set the current optimization iteration, $t = 0$
repeat:
  for all *weight vectors, $\vec{w}_{yx}$, making up the SOM grid* **do**
    Compute the error function, $E(t)$, defined in Equation (4.3)
    call function UpdateProjectionVector($\vec{r}_{yx}$) to optimize $E(t)$
    Update the current optimization iteration, $t = t + 1$
  end for
until *stopping criteria are met*

Algorithm 4.1: Pseudocode of the Sammon's mapping projection algorithm.

begin function UpdateProjectionVector($\vec{r}_{yx}$) **:**

  for all *projection components $r_{yx\hat{e}} \in \vec{r}_{yx}$* **do**

$$\frac{\partial E(t)}{\partial\, r_{yx\hat{e}}(t)} = \frac{-2}{\sum_{\delta\in\mathcal{Y}} \|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2} \times \sum_{\vec{w}_{y'x'}\neq\vec{w}_{yx}} \left(\frac{\|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2 - \|\vec{r}_{yx} - \vec{r}_{y'x'}\|_2}{\|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2 \cdot \|\vec{r}_{yx} - \vec{r}_{y'x'}\|_2} \right) \times (r_{yx\hat{e}} - r_{y'x'\hat{e}})$$

$$\frac{\partial^2 E(t)}{\partial\, r_{yx\hat{e}}(t)^2} = \frac{-2}{\sum_{\delta\in\mathcal{Y}} \|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2} \times \sum_{\vec{w}_{y'x'}\neq\vec{w}_{yx}} \left(\frac{1}{\|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2 \cdot \|\vec{r}_{yx} - \vec{r}_{y'x'}\|_2} \right)$$
$$\times \left[\left(\|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2 - \|\vec{r}_{yx} - \vec{r}_{y'x'}\|_2 \right) - \frac{(r_{yx\hat{e}} - p_{y'x'\hat{e}})^2}{\|\vec{r}_{yx} - \vec{r}_{y'x'}\|_2} \right.$$
$$\left. \times \left(1 + \frac{\|\vec{w}_{yx} - \vec{w}_{y'x'}\|_2 - \|\vec{r}_{yx} - \vec{r}_{y'x'}\|_2}{\|\vec{r}_{yx} - \vec{r}_{y'x'}\|_2} \right) \right]$$

$$\Delta r_{yx\hat{e}}(t) = \frac{\partial E(t)}{\partial\, r_{yx\hat{e}}(t)} \div \left| \frac{\partial^2 E(t)}{\partial\, r_{yx\hat{e}}(t)^2} \right|$$

    Update $r_{yx\hat{e}}(t+1) = r_{yx\hat{e}}(t) - \big(\varphi \cdot \Delta r_{yx\hat{e}}(t)\big)$, where $\varphi \approx 0.3$ or 0.4
  done

end function

Algorithm 4.2: Pseudocode of the original Sammon's mapping error optimization technique, which optimizes the components making up the projection vector $\vec{r}_{yx}$.

Once the projection vector positions have been optimized, the projection space is visualized, with a mark for each projection vector. The result is a map representation that uses spatial positioning to represent the weight-space distribution of neurons. Because the display typically bears no resemblance to the regular map grid, the correlation between the projection and the map grid is usually difficult to see [108]. A common solution is to use line segments to link the projection vectors of neighboring neurons [253].

Projection algorithms were originally intended to be applied directly to a data set. However, for many data examples, such projections become computationally too complex. Because a map's weight vectors approximate a data set's distribution using fewer weight vectors, applying projection methods to a SOM trained on a data set is more computationally feasible than a direct projection [253]. However, SOM training is itself computationally complex, thus often negating this advantage. On the other hand, if maps with fewer neurons than training examples are used, map projections are less visually dense and easier to interpret than those based on raw training data [131].

Figure 4.11 shows two Sammon's mapping projections of SOMs trained on the Iris data set. Figure 4.11 (a) and (b) respectively illustrate the U-matrix and Sammon's projection of the example SOM from Figure 4.1, which is globally ordered. Within the projection, dots represent neurons, where line segments connect the dots representing neurons that are topologically adjacent to one another in the map structure. Both representations of the ordered map show two fairly tightly grouped weight vector clusters: one to the upper right, and another in the lower left, in both visualizations.

Figure 4.11 (c) and (d) show the U-matrix and Sammon's map, respectively, of a globally unordered map trained on the Iris data set. The U-matrix shows three weight vector clusters: one in the map's upper right corner, another occupying the center area, and a third in the lower left. As is the case in Figure 4.11 (b), line segments in the projection connect adjacent neurons. The Sammon's mapping indicates three topologically separate weight vector groups: one in the projection's lower left, and two in its upper right part. However, the two groups in the upper right are spatially close to one another, indicating that these groups are actually constituents of one emergent cluster.

The unordered map example depicted by Figure 4.11 (c) and (d) illustrates that weight vector projections can be used to more accurately visually determine the number

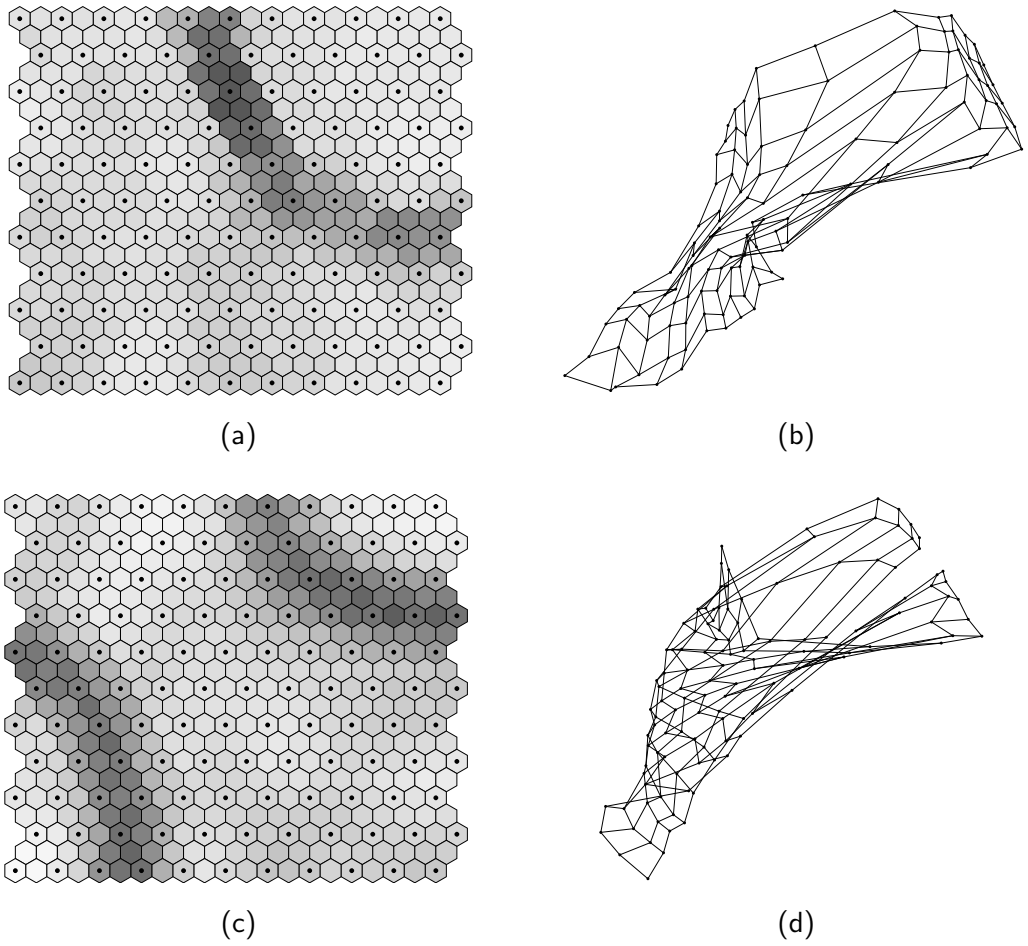


Figure 4.11: Two weight vector projections of SOMs trained on the Iris data set: (a) and (b) respectively show a U-matrix and a Sammon's mapping projection of the globally ordered example SOM in Figure 4.1; (c) and (d) respectively show a U-matrix and a Sammon's mapping projection of a SOM trained on the same data, which lacks global order.

of emergent map clusters than either the local or global weight vector similarity encodings of Section 4.3.1. Projection-based visualizations can therefore be used to help determine whether a map representation is globally ordered or unordered.

Of course, the individual projection vector marks making up a projection-based visualization can be augmented in a similar fashion to a regular grid-based map representation. Any of the techniques described in Sections 4.3.1 to 4.3.3 may be utilized. Once again, any visual dimension (such as the size or color of the vector's mark) may be used

for the encoding. Of course, the weight vector similarity encodings of Section 4.3.1 are less sensible in the context of weight vector projection methods, because this type of similarity is implicitly represented by spatially arranged projection vectors.

4.4.2 Map-Based Information Visualizations

Map-based information visualizations are general in nature, and use any information visualization to either represent the map structure, or meta-information in relation to the map structure. As generic information visualization methods are not designed with SOMs in mind, they are non-uniform and do not depict the map's actual grid form.

The simplest approach based on information visualization involves the representation of sub-aspects of the weight vectors across a subset of a map's neurons. Examples of this approach are shown in Figure 4.12, where scatter plots represent the distribution of weight vector component pairs in relation to one another, for all the neurons of the example SOM from Figure 4.1. Interpretation of the actual modeled attribute values requires that the weight components in an information visualization should be de-normalized, as described in Section 2.4.2.1. In the example, min-max normalization was performed on the training data, thus requiring de-normalization according to Equation (2.2).

The examples of Figure 4.12 illustrate the primary advantage associated with map-based information visualizations: map-based information visualizations are often very good at highlighting specific aspects of interest in the structure of a map, depending on the method used. For example, Figure 4.12 (a) shows no strong correlation between the `sepal_length` and `sepal_width` components, while Figure 4.12 (b) clearly indicates a strong positive correlation between `petal_length` and `petal_width`.

However, the main advantage of map-based information visualizations is also their biggest drawback: their application may be limited if their specific focus is at the expense of other map information. Any limitations that are intrinsic to the chosen information visualization method may also detract from the final representation. For example, the scatter plots of Figure 4.12 can only compare two weight vector components.

In a similar fashion to the projection-based approaches of Section 4.4.1, all information visualizations may be applied directly to a map's training data. As previously noted, the weight vectors of a converged map should model the training data's distribution, us-

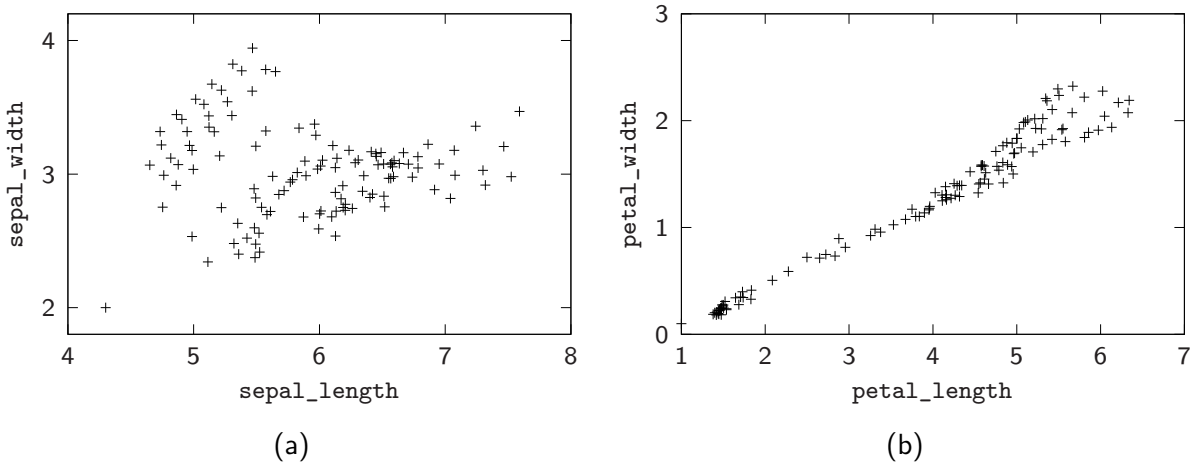


Figure 4.12: Two examples of scatter plot map information visualizations of the example SOM in Figure 4.1: (a) shows a scatter plot comparing the `sepal_length` and `sepal_width` components; (b) shows a scatter plot comparing the `petal_length` and `petal_width` components. The components have been de-normalized to their original value ranges.

ing fewer representational vectors. Consequently, a map-based information visualization should again offer a representation that is less visually dense than a raw data set visualization, and is easier for an analyst to interpret. For some very complex information visualization techniques, or on very large data sets, a SOM-based visualization may also be more computationally feasible than a raw data set representation.

It is also possible for a map-based information visualization to represent meta-information related to the map. The types of meta-information that can be represented on a map-based information visualization are varied, and include groups of aggregated neurons, discovered emergent clusters of neurons (see Chapter 5 for further details on emergent cluster discovery), the raw number of data examples falling within a particular map area or BMU, and the strength of a map response to a data example.

Figure 4.13 illustrates an example of a map meta-information visualization. Figure 4.13 (a) shows three arbitrary map areas, while Figure 4.13 (b) visualizes the raw numbers of training examples from each of the three possible data classifications that fall within each of the map areas. From this information visualization it is clear that map areas B and C respectively contain mostly `Iris_Versicolor` and `Iris_Setosa` examples, while area A holds a mixture of `Iris_Versicolor` and `Iris_Virginica` examples.

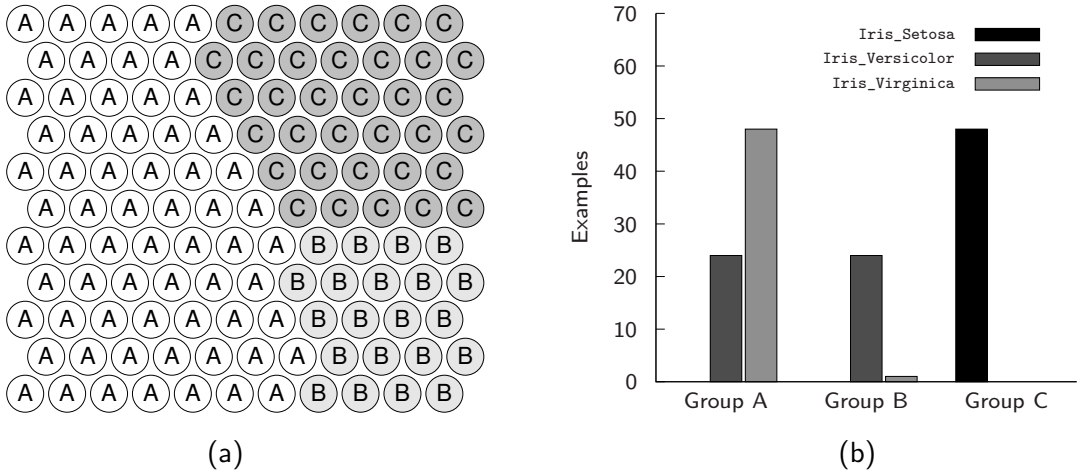


Figure 4.13: An example of a map information visualization that shows map meta-data, based on the example SOM of Figure 4.1: (a) shows a map grid with three groups of neurons respectively designated A, B and C; (b) shows a histogram that breaks down, per example classification, the number of BMUs that fall within map areas A, B and C.

Of course, if data mapping information is represented, it is possible to use either the training data set, or another disjoint set that was not presented during map training. Information visualizations showing map meta-information show information that is specific to the SOM's data model. Consequently, there are no analogous information visualizations that can be generated entirely from the raw training data.

Naturally, any map-based information visualization can be augmented using any of the methods described in Sections 4.3.1 to 4.3.3. For example, in Figure 4.12 each scatter plot point can be encoded with the number of data examples mapping to its corresponding neuron. Of course, not all augmentations are sensible for all information visualizations. For example, the marks in Figure 4.12 (a) should not be encoded with `sepal_width` weight values, as this information is intrinsic to the visualization.

4.5 SOM-Based Exploratory Data Analysis

SOMs have been widely applied to a variety of EDA tasks [52, 131, 150]. In the context of an EDA exercise, a human typically uses a graphical SOM-based tool [51], where multiple visualizations give different views of the modeled data. For optimal utility,

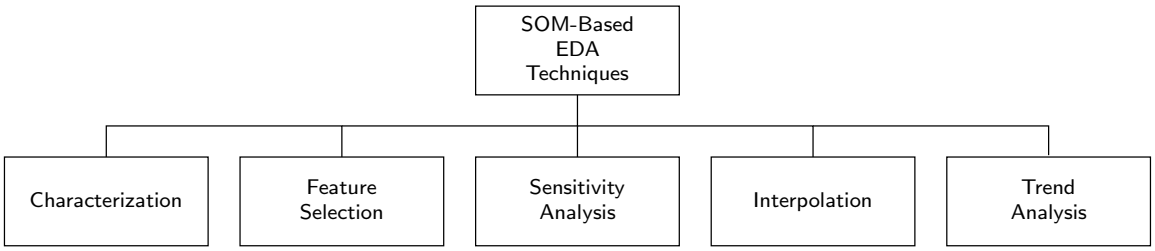


Figure 4.14: A taxonomy of the categories of SOM-based EDA identified by this dissertation.

SOM-based tools should also be interactive. Such tools allow, for example, comparative visualization linking [168], and map views with varying levels of granularity.

This work identifies five categories of SOM-based EDA, illustrated in Figure 4.14. This research does not aim to provide an exhaustive discussion, instead presenting only a broad overview of the general analytical methods that each EDA category encompasses.

4.5.1 Characterization

Visualizations of a trained SOM can provide general characterizing descriptions of a training SOM, or portions thereof. These descriptions include the overall nature of a map's constituent emergent clusters, the attribute value distributions over map areas or emergent clusters, and data example prevalence over map areas or clusters.

It is also possible to determine the characteristics of either training or unseen data examples through the use of a SOM. This is achieved by visualizing data examples in relation to a trained map or the emergent clusters within such a map. Because map areas model different parts of a training data set, the characterizations of map areas or emergent clusters are effectively characterizations for subsets of data examples. It is thus possible to assign a characterization to a data example, which is taken from the map area or emergent cluster within which the BMU of the data example falls.

Yet another use of a SOM involves the characterization of data examples as either accurate or erroneous. Fessant and Midenet [75] propose storing a data example list per neuron, which holds all the training data examples that share the same neuron as a BMU. The mean distance between each data example in the list and the BMU's weight

vector, $\vec{w}_{ba}$, is then found, and denoted $\vartheta(\vec{w}_{ba})$. The following coefficient is computed when an unseen data example, denoted $\vec{z}_s$, is presented to the map:

$$\exp\left(\frac{-\|\vec{z}_s - \vec{w}_{ba}\|_2}{2 \cdot \vartheta(\vec{w}_{ba})}\right)$$

In the event that the value of this coefficient is lower than a user-defined threshold value, the data example represented by $\vec{z}_s$ should be considered to be erroneous.

4.5.2 Feature Selection

It is possible to use visualizations augmented with the weight vector encodings discussed in Section 4.3.2, to find attributes with very specific values for one or more emergent clusters. Such attributes are responsible for emergent cluster formation and data example membership to clusters, and consequently help determine data classification.

Conversely, attributes with values that vary greatly over an emergent cluster are not important to the formation of that cluster. In a case where specific values for an attribute do not correlate to any particular cluster on the map, or when an attribute has a generally uniform value over the entire map, it is an indication that the attribute is unimportant, and can safely be discarded from any further exploratory analysis tasks.

4.5.3 Sensitivity Analysis

The sensitivity of a data example to specific attribute value changes can also be determined using a SOM. During this analysis, an expert observes the change in an example's mapping to a SOM as one or more of the example's attributes are modified.

This approach to analysis answers "what-if" questions by determining which ranges of attribute modifications affect the classification of an example. Conversely, the set of attribute changes that will safely produce negligible effects can also be determined.

4.5.4 Interpolation

It is also possible to use a trained SOM to infer the probable values for missing attributes within a data set example, where the data example in question has usually not been presented during map training. The attribute value replacement is done by

replacing missing data example values with the corresponding component value of the example's BMU [210]. The Euclidean distance between the example and the BMU, possibly weighted by the number of non-missing attribute values, serves as a confidence measure for the missing values that have been replaced.

Fessant and Midenet [75] propose replacing missing values of a data example using the example's BMU, as well as the neurons neighboring the BMU. The BMU and its neighbors are referred to as the *activation group* of the example. Missing values are then replaced with the mean of the corresponding weight value within the activation group. It is also possible to discard any neuron, n_{yx}, from the BMU neighborhood when the value of the following function exceeds a user-selected threshold:

$$\frac{\|\vec{z}_s - \vec{w}_{yx}\|_2 - \|\vec{z}_s - \vec{w}_{ba}\|_2}{\|\vec{z}_s - \vec{w}_{ba}\|_2}$$

where $\vec{z}_s$ is the data example for which missing values are being replaced, and $\vec{w}_{ba}$ is the weight vector of the BMU for $\vec{z}_s$. This removal procedure aims to reduce the neighborhood when the BMU is sufficient for missing value replacement. The lower the selected threshold value, the more aggressive the removal of neurons will be.

Cottrell and Letrémy [43] suggest an approach that determines missing attribute values using all the neurons across a trained SOM. First, the method computes the probabilities with which the example in question is mapped to each neuron. The missing value is predicted to be the mean of the weight value that corresponds to the missing attribute, computed over all the map's neurons. However, each neuron's contribution to this mean is weighted by the previously computed mapping probability of that neuron. The approach also calculates the confidence levels for the replaced values.

Finally, an analyst is also able to interpolate the attribute modifications that are required in order to migrate an example from one area of the map to another (for example, from an "undesirable" emergent cluster to a more "desirable" cluster).

4.5.5 Trend Analysis

Finally, SOMs are capable of modeling time-series and historical data, in order to determine the nature of time-variant changes or the likely course of future changes. A very simple and commonly used method [96, 144] utilizes a SOM trained on a time-series data

set that represents a sequence of measurements or observations at regular intervals. The BMUs of several related time-series observations (typically generated by a single entity) are then marked on any grid-based visualization of the map, and the BMUs are linked in sequence to form a trajectory. Such trajectories are used either to study data changes over time, or to suggest emerging trends in data changes over time.

Other trend analysis approaches exist, but modify the SOM training procedure or the map architecture from the basic stochastic method focused on in this work. These methods are thus not discussed in great detail within this dissertation. The more notable and commonly referenced examples of techniques in this category include the following:

- Trajectory maps [140] train a first-level SOM using observations of entities at different time steps. BMU trajectories of entities on the first-level map are used to train a second-level map, thus modeling the nature of time-dependent changes.
- The temporal Kohonen map [32] incorporates each neuron's response for the previous training iteration into the neuron's response computed for the current iteration (thus affecting BMU determination), but otherwise uses normal stochastic training.
- The recurrent SOM [152] breaks down a neuron's response into a vector of components, one for each weight. Each component incorporates its value for the previous iteration. BMUs and weight updates are then computed using these vectors.
- The recursive SOM [263] uses two maps. The first is updated using training examples, while the second is trained on vectors representing the first map's responses in the previous iteration. BMUs are determined using information from both maps.
- The SOM for structured data [98] is an approach developed for training maps on directed acyclic graph data structures. Because temporal data is a special type of directed acyclic graph, this approach is also appropriate for temporal learning.
- MergeSOM [230] is a technique that allows for the simulation and extraction of finite state automata from SOMs. These finite state automata are able to model time series characteristics that are present within a training data set.

The interested reader is referred to the work of Guimarães *et al* [96, 97], which provides a taxonomy and extensive survey of SOM-based approaches for temporal learning.

4.6 Summary

This chapter gave a critical analysis of the main categories of visualization techniques that can be used to represent a trained SOM grid and data examples that relate to such a grid. Section 4.1 gave a brief overview of the ideas underpinning traditional data visualization approaches. Section 4.2 described the overall philosophy that underlies all the SOM-based visualization methodologies. Sections 4.3 and 4.4 discussed the two map representation categories (namely grid-based and irregular representations), and the visual augmentations that can be used to add meaning to such representations. Finally, Section 4.5 briefly considered the main categories of SOM-based EDA.

Map neuron labeling is an integral step of almost all SOM-based EDA and DM, and is discussed within Chapter 6. The next chapter deals with emergent map cluster discovery, which is a prerequisite for several of the most common neuron labeling approaches. In addition, cluster discovery is also used by some of the SOM-based DM algorithms. Consequently, the discovery of emergent clusters appears as a component in many practical EDA and DM exercises that use the SOM algorithm as a basis.

Chapter 5

Emergent Neuron Cluster Discovery

The previous chapter described methods for SOM visualization, as well as SOM-based EDA. This chapter covers emergent cluster discovery, which is an important component within several of the map neuron labeling methods described in Chapter 6.

Section 5.1 overviews emergent cluster discovery. Section 5.2 covers the evaluation of emergent clusters. This research defines three classes of emergent cluster discovery methods, namely *algorithmic*, *exploratory*, and *hybrid* (respectively discussed in Sections 5.3, 5.4, and 5.5, and illustrated as a taxonomy in Figure 5.1). Section 5.6 touches on cluster stability in the context of SOMs. Finally, Section 5.7 gives a chapter summary.

5.1 An Overview of Emergent Cluster Discovery

Section 3.4.3 described emergent feature maps, and the advantages that emergent maps have over their non-emergent counterparts. However, the following must be noted:

- Emergent feature maps only explicitly represent clusterings of training examples around neuron centroids represented by weight vectors.
- While macro-level clusters of neurons do exist within emergent SOMs, these neurons are not explicitly organized into groups.
- An additional step is therefore required in order to group neurons that are similar to one another into a set of macro-level clusters.

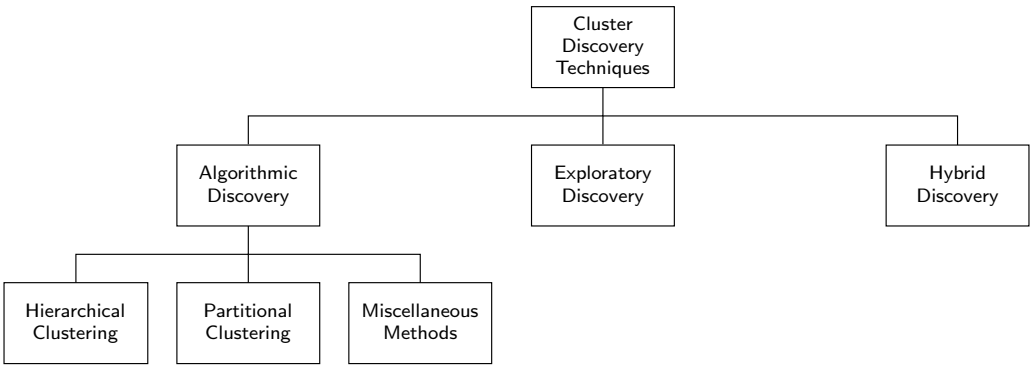


Figure 5.1: A taxonomy of SOM cluster discovery techniques. Note that only the algorithmic discovery techniques that are most commonly used in a SOM-based context are shown.

Discrete emergent cluster discovery addresses the third point, and is concerned with the division of neurons between a set of k clusters, denoted $\mathcal{L} = \{S_1, S_2, \ldots, S_k\}$. In turn, each cluster, S_i, is a discrete set of neurons that are all judged to be mutually similar in some way. Each neuron may belong to only a single cluster. Fuzzy clustering [35, 226] is also possible, but is difficult to interpret, and is thus not considered further.

All cluster discovery methods use an indicator of emergent cluster structure. Based on the emergent feature map characteristics of Section 3.4.3, two indicators are possible:

- *Inter-weight-vector similarity*, where groups of very similar neurons indicate emergent clusters, and neurons that are dissimilar to all other neurons denote borders between clusters. Inter-weight-vector distance is a simple measure of similarity (Euclidean distance is most commonly used, but other measures will also work).

- *Data example concentration*, where higher concentrations of BMUs indicate emergent clusters, and lower concentrations indicate cluster boundaries. If the examples have classification attributes, higher concentrations of similarly classified data examples may provide a more accurate indicator of emergent clusters.

Data example concentration is much less frequently used, and may give inconclusive results when small data sets used on large maps produce sparse mappings. Inter-weight-vector similarity may thus be seen as a primary cluster indicator, and data example concentration as a secondary indicator. The two could conceivably be combined into a more complex indicator, but it seems no such research has yet been conducted.

5.2 Cluster Quality Evaluation

A cluster quality evaluation measure quantitatively scores the relative desirability of neuron clusterings. A variety of cluster quality measures exist [17, 172]. This research considers only one such quality measure, called the *Davies-Bouldin index* [48]. The Davies-Bouldin index is commonly used to evaluate discovered emergent clusters in SOMs [257], and was used in the experiments that are described in Chapter 8.

The Davies-Bouldin index calculates a real-valued result for a group of clusters within which $k \in [2, \infty)$, where lower values indicate more optimal clusterings, as follows:

$$index = \frac{1}{k} \cdot \sum_{i=1}^{k} \max_{j \neq i} \left\{ \frac{intra(S_i) + intra(S_j)}{inter(S_i, S_j)} \right\} \tag{5.1}$$

where k is the number of clusters in the grouping being evaluated, S_i and S_j are arbitrary clusters, $intra(S_i)$ is a measure of the intra-cluster distances between the neuron weight vectors within cluster S_i, and $inter(S_i, S_j)$ is a measure of the inter-cluster distance between the neuron weight vectors within clusters S_i and S_j, respectively.

Very commonly used measures for $intra(S_i)$ and $inter(S_i, S_j)$ are, respectively, the *centroid distance* and *centroid linkage*. The centroid of a cluster is an I-dimensional vector, computed as the average over all neuron weight vectors in the cluster.

Centroid distance is defined as the average Euclidean distance between the weight vectors of cluster S_i and the cluster's centroid vector, calculated as follows:

$$intra(S_i) = \frac{1}{o_i} \cdot \sum_{\vec{w}_{yx} \in S_i} \|\vec{w}_{yx} - \vec{g}_i\|_2 \tag{5.2}$$

where o_i is the number of weight vectors in S_i, $\vec{w}_{yx}$ is an arbitrary weight vector, and $\vec{g}_i$ is the centroid vector of S_i. It is usually assumed that $intra(S_i) = \infty$ for singleton clusters, to prevent *index* from biasing towards clusters of only one neuron.

The centroid linkage measure is calculated as the Euclidean distance between the centroid vectors of two clusters, S_i and S_j, defined as follows:

$$inter(S_i, S_j) = \|\vec{g}_i - \vec{g}_j\|_2 \tag{5.3}$$

where $\vec{g}_i$ and $\vec{g}_j$ are the centroid vectors of clusters S_i and S_j, respectively. Equation (5.1) implies that S_i and S_j will never be the same cluster when calculating *index*.

In addition to the centroid distance of Equation (5.2), there are several other measures for $intra(S_i)$, including average and nearest neighbor distance [256]. Similarly, other than the centroid linkage of Equation (5.3), there are other possibilities for $inter(S_i, S_j)$, including single linkage, complete linkage and average linkage [70]. These methods are uncommon in the context of SOMs, and are thus not considered in more detail.

5.3 Algorithmic Cluster Discovery

Algorithmic cluster discovery uses *hard clustering algorithms* to computationally partition generic objects (usually represented as vectors) into roughly homogeneous discrete clusters, where object membership is exclusive to one cluster [123]. In the context of SOMs, clustering algorithms group a trained map's neurons [247].

Jain *et al* [123] taxonomize a variety of clustering algorithms, any of which can be used for algorithmic cluster discovery. However, *hierarchical* and *partitional* clustering algorithms are most commonly used with SOMs [255, 257], and are considered in turn, below. In the third place, some notable miscellaneous approaches are discussed.

Clustering algorithms are often biased towards certain cluster shapes (for example, elongated or spherical clusters) [70]. It is therefore advisable to use a hybrid cluster discovery approach discussed in Section 5.5 to visually inspect clusters for feasibility, allowing inappropriately discovered clusters to be adjusted if necessary.

This work's literature survey found no examples of SOM-based clustering algorithms that use the data example concentration cluster indicator. The remainder of this subsection thus assumes only inter-weight-vector similarity (i.e., the algorithms group together weight vectors separated by smaller distances). It is, however, conceivable that existing clustering algorithms could be modified to use data example concentration instead.

Because a SOM's weight vectors approximate the distribution of the original training data set, clustering the weight vectors should find clusters with characteristics that are approximately equivalent to those found if the training data itself is clustered [257]. Therefore, in general, either weight vectors or training examples can be clustered. However, the fundamental nature of the clusters differ in each case: in the former, components of a data model are grouped; in the latter, actual data members are clustered.

Due to the approximating nature of a SOM's mapping, it is likely that SOM-based clusters will be less accurate than clusters found in the map's training data [257]. However, SOM-based algorithmic cluster discovery does offer the following advantages:

- A SOM manifests *unsupervised*, and *emergent* clusters. Most standard clustering algorithms are supervised because they need contextual information, and are thus biased. The standard algorithms also do not have emergent characteristics.
- Clustering algorithms are often time complex, becoming impractical on large data sets. A SOM typically models many training examples using substantially fewer weight vectors [247], which are more time-efficient to cluster [257].

5.3.1 Hierarchical Clustering Algorithms

Hierarchical approaches derive their name from the hierarchy of different sized clusters that they discover. The structure of clusters can be visualized as a *dendrogram*, which shows how larger high-level clusters are composed of smaller low-level clusters. To produce a specific number of clusters, it is possible to cut dendrograms in various ways. Figure 5.2 shows a dendrogram with two cuts that both produce four clusters.

Most hierarchical methods are iterative and greedy (i.e., they do not backtrack to modify the groups formed by previous iterations), making the clustering only approximately optimal. Hierarchical methods may be either agglomerative, or divisive:

- *Agglomerative algorithms* use a bottom-up approach to successively select similar lower-level clusters, and merge them into larger higher-level groups.
- *Divisive algorithms* follow a top-down approach. Larger, higher-level clusters are iteratively split into smaller, lower-level ones that are more cohesive.

Pseudocode outlines of both these classes are given in Algorithm 5.1. In each case, the cluster selection is based on an optimality criterion. The exact specification of this detail is what differentiates specific hierarchical clustering algorithms from one another.

A drawback which is associated with all hierarchical clustering algorithms is that dendrograms produced for a large number of map neurons are computationally expensive to build and require a large amount of storage space to represent [123].

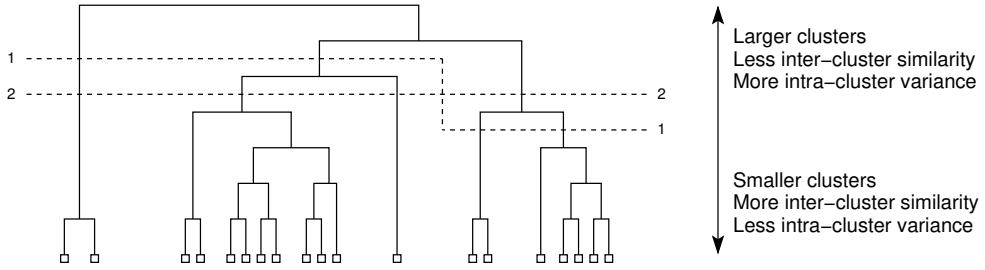


Figure 5.2: A hierarchical dendrogram, where squares denote neurons and lines link composite clusters. Dashed lines show two ways to cut the dendrogram, both producing four clusters.

(a) Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-dimensional training set, denoted $\mathcal{D}_T$, until convergence

Define a set of clusters, $\mathcal{L} = \{S_1, S_2, \ldots, S_{Y \times X}\}$, where each cluster contains a single unique neuron from the complete set of $Y \times X$ neurons in *map*
repeat:
Select $S_i \in \mathcal{L}$ and $S_j \in \mathcal{L}$ using an optimality criterion (e.g., Ward distance [269])
Create a new cluster S_q that contains both S_i and S_j as sub-clusters
Remove S_i and S_j from $\mathcal{L}$, and add S_q to $\mathcal{L}$
until $\mathcal{L}$ *contains a single cluster of all map neurons*

(b) Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-dimensional training set, denoted $\mathcal{D}_T$, until convergence

Define a set containing a single cluster, $\mathcal{L} = \{S_1\}$, where S_1 contains all the unique neurons within the complete set of $Y \times X$ neurons in *map*
repeat:
Select $S_q \in \mathcal{L}$ using an optimality criterion (e.g., min-max cut [57])
Use a splitting method (e.g., Mcut [57]) to break S_q into sub-clusters S_i and S_j
Remove S_q from $\mathcal{L}$, and add S_i and S_j to $\mathcal{L}$
until $\mathcal{L}$ *contains* $Y \times X$ *clusters, each of a single unique map neuron*

Algorithm 5.1: Pseudocode of the two generic classes of greedy hierarchical clustering algorithms: (a) shows the agglomerative technique; (b) shows the divisive technique.

This research only considers agglomerative approaches in more detail, because divisive methods are far less common in the clustering literature [134]. Divisive algorithms are also usually more complex, because these methods have to find an optimal division along which to split clusters. This work only investigates agglomerative *Ward clustering* and a SOM-specific variant thereof, because of their prevalence in the SOM literature.

Classical Ward clustering [266] defines a cost function over all clusters. The cost function's value always increases with each merge that the algorithm performs. At each iteration, the algorithm selects and merges the two clusters that result in the smallest cost increase. Classical Ward clustering is a greedy algorithm. The sum-of-squares criterion [70] is often used as a cost function. However, the classical approach is fairly inefficient, especially when large maps require the clustering of many neurons.

A more efficient method, shown to be equivalent to Ward clustering using the sum-of-squares criterion [269], simply merges cluster pairs with the smallest *Ward distance*, with no backtracking. The Ward distance between clusters S_i and S_j is:

$$ward\_dist(S_i, S_j) = \frac{o_i \cdot o_j}{o_i + o_j} \cdot \|\vec{g}_i - \vec{g}_j\|_2^2$$

where o_i and o_j denote the number of neurons that constitute the two clusters S_i and S_j respectively, $\vec{g}_i$ and $\vec{g}_j$ are respectively the centroids of these two clusters, and $\|\vec{g}_i - \vec{g}_j\|_2$ is the Euclidean distance between the two cluster centroids $\vec{g}_i$ and $\vec{g}_j$.

As previously defined, a centroid is an I-dimensional vector average of a cluster's weight vector constituents. Before any cluster merging steps take place, each centroid is simply equivalent to the weight vector of the single neuron that belongs to the centroid's cluster. After the Ward clustering algorithm merges two lower-level clusters S_i and S_j to form a new higher-level cluster S_q, the centroid of S_q is defined as:

$$\vec{g}_q = \frac{1}{o_i + o_j} \cdot (o_i \vec{g}_i + o_j \vec{g}_j)$$

The number of neurons in the newly formed cluster is referred to as o_q, and is simply the sum of the number of neurons in S_i and S_j. In other words, $o_q = o_i + o_j$.

If a SOM lacks global order (see Section 3.4.3), it is possible for Ward clustering to merge clusters that are not topologically adjacent on the map. Such merging will result in discovered clusters that are split up over the map's surface, which will complicate

the interpretation of an exploratory analysis task. In a SOM-based context, therefore, Ward clustering implementations should only allow adjacent clusters to merge with one another [176]. This approach is sometimes referred to as *SOM-Ward clustering*.

It is possible to use a cluster evaluation measure to find an optimal level of hierarchical clustering. Firstly, a hierarchical clustering algorithm is executed until termination (i.e., an agglomerative algorithm merges clusters until a single cluster is produced, while a divisive algorithm splits clusters until only singleton clusters remain). At each algorithmic step, the chosen cluster evaluation measure is recorded for the clustering produced by the merge or split. Finally, the dendrogram is cut at the level that produces the best evaluation, thus producing an approximately optimal clustering of map neurons.

5.3.2 Partitional Clustering Algorithms

Partitional clustering algorithms (which are also called non-hierarchical or k-clustering algorithms) divide all neurons between k disjoint clusters $S_1, S_2, \ldots, S_k$. The value of k is a user-specified algorithmic parameter. The aim of a partitional algorithm is usually to maximize intra-cluster cohesion, while minimizing inter-cluster coupling.

This dissertation only focuses on the *k-means* method [161], also often used with SOMs. First, k clusters are defined, each with a randomly selected neuron as initial member, and a centroid equivalent to the neuron's weight vector. Each remaining neuron is then added to the cluster possessing the centroid closest to the neuron's weight vector. With each neuron addition, the updated cluster's centroid shifts to represent the new mean of the cluster's member vectors. Algorithm 5.2 outlines the technique.

The sizes, shapes and optimality of clusters may vary dramatically for different values of k, making the selection of this parameter's value a very important choice. Methods like k-means are also sensitive to the initial centroid initializations [123].

It is again possible to use a cluster evaluation measure to find an optimal partitional clustering. A partitional algorithm is executed for a range of potential k values, chosen so that $k \in [2, Y \times X]$. For each neuron clustering that is produced by an algorithm execution, the related evaluation measure is recorded. The final clustering is the one with the most optimal associated evaluation measure. This approach is generally very time complex, because it necessitates multiple executions of the clustering algorithm.

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-dimensional training set, denoted $\mathcal{D}_T$, until convergence

Define a cluster count k, and a set of k empty clusters $\mathcal{L} = \{S_1, S_2, \ldots, S_k\}$

for all *clusters* $S_i \in \mathcal{L}$ **do**
    Select a random neuron n_{yx} that is unassigned to any $S_i \in \mathcal{L}$, and add it to S_i
    Set centroid $\vec{g}_i$ of cluster S_i equivalent to the weight vector $\vec{w}_{yx}$ of neuron n_{yx}
end for

while *neurons exist that remain unassigned to any* $S_i \in \mathcal{L}$ **do**
    Select a random neuron n_{yx} (and associated $\vec{w}_{yx}$) that is unassigned to any $S_i \in \mathcal{L}$
    Find cluster $S_i \in \mathcal{L}$ with centroid $\vec{g}_i$, such that $\|\vec{w}_{yx} - \vec{g}_i\|_2 = \min_{\forall i'}\{\|\vec{w}_{yx} - \vec{g}_{i'}\|_2\}$
    Add n_{yx} to S_i and adjust $\vec{g}_i$ to be the mean over all neuron weight vectors in S_i
end while

Algorithm 5.2: Pseudocode of the k-means partitional clustering algorithm.

5.3.3 Miscellaneous Clustering Algorithms

The focus of this research is not an exhaustive review of the huge variety of less-used clustering algorithms that are available. The miscellaneous techniques discussed in this subsection are applied specifically within the domain of SOMs, and are therefore of interest within the domain of SOM-based EDA and DM. A detailed analysis of the benefits and drawbacks associated with each of the described miscellaneous approaches is beyond the scope of this dissertation, and is therefore left to future work.

This research work identifies three main SOM-specific miscellaneous clustering algorithms, namely partitional methods with centroids based on map-specific information, hierarchical SOM systems, and algorithms focused on cluster boundary discovery. Each of these three broad approaches is discussed under a separate heading, below:

5.3.3.1 Partitional Centroids Based on Map-Specific Information

The first type of miscellaneous clustering algorithm simply extends standard partitional clustering algorithms in order to allow the exploitation of information derived specifically from a trained SOM's map structure. Approaches in this category use map-specific

information to initialize cluster centroids using the map structure's own constituent neuron weight vectors. Two variations on this general approach exist:

- The centroids can be chosen as the neurons that have minimal values in the map's local similarity matrix [251]. Each remaining neuron on the map is then clustered with the closest centroid in weight space, in terms of Euclidean distance. One possible additional requirement is that clusters must remain contiguous [256]. This technique is based on the inter-weight-vector similarity cluster indicator.

- Lewis *et al* [158] propose choosing the neuron with the highest value in the map's data subset mapping matrix as an initial centroid. Adjacent neurons are added to the centroid's cluster, while the addition does not increase the cluster's evaluation beyond a threshold. A possible evaluation of cluster S_i is simply $intra(S_i)$[1]. Once a cluster cannot be grown further, the process is repeated on the unclustered neurons. This method uses the data example concentration cluster indicator.

5.3.3.2 Hierarchical SOMs

Another miscellaneous approach uses a hierarchical SOM system to discover emergent neuron clusters. A hierarchical SOM simply trains a secondary SOM on an aspect of the map that is being analyzed. Two general variants on this theme are possible:

- It is possible to train the secondary SOM on the weight vectors of the analyzed SOM [164]. The secondary SOM models the analyzed SOM's weight vector distribution, and thus uses the inter-weight-vector similarity cluster indicator.

- Another alternative trains the secondary SOM on the two-dimensional map coordinates of neurons that are data example BMUs [156]. This method models the map's BMU distribution, and focuses on the data example concentration indicator.

In both cases a non-emergent secondary SOM is equivalent to using k-means clustering on the analyzed SOM. The k-means algorithm is less time complex than the SOM algorithm,

[1] In the original work of Lewis *et al*, the cluster evaluation mechanism is based on the problem-specific characteristics of the training examples mapped into each cluster. However, the previously discussed intra-cluster cohesion measure, $intra(S_i)$, is an applicable general cluster evaluation measure.

and should thus be preferred in practice. Emergent secondary SOMs, however, are able to find emergent clusters in the analyzed maps, but are more complex to interpret.

5.3.3.3 Cluster Boundary Discovery Algorithms

Finally, it is possible for algorithms to focus on discovering boundaries for emergent map clusters. This research identifies two alternative strategies in this category:

- The rule extractor by Malone *et al* (see Section 7.2) proposes the boundary difference value (BDV) [162] statistic for identifying boundaries of widely separated weight vectors. Such boundaries are based on inter-weight-vector similarity.
- Another approach is to mark neurons that are not BMUs for any labeling examples, and designate these neurons as cluster boundaries [256]. This technique uses data example concentration as an indicator of emergent cluster structure.

Finally, both boundary discovery methods merge the neurons within boundary divisions into clusters. These approaches suffer from clustering errors when boundary gaps occur, and have complex implementations in contrast to traditional clustering methods.

5.4 Exploratory Cluster Discovery

All exploratory clustering methods are SOM-specific, due to a reliance on SOM visualization techniques. As Figure 5.3 shows, map visualizations aid a human in identifying emergent clusters [257, 274], or boundaries between clusters [50]. Exploratory clustering thus has much in common with EDA methods, but is also applicable to DM tasks.

Because the grid-based representations of Section 4.3 closely resemble the neurons' actual organization on the map, grid visualizations are very commonly used for exploratory cluster discovery. While also usable, the irregular map representations discussed in Section 4.4 are generally more difficult to interpret in an exploratory context.

Using an exploratory approach, it is up to the human analyst to subjectively decide whether a particular region of interest is significant enough to constitute a cluster. The analyst then typically makes use of an interactive tool to select single neurons or groups of neurons, and to designate these neurons into cohesive, disjoint clusters [257].

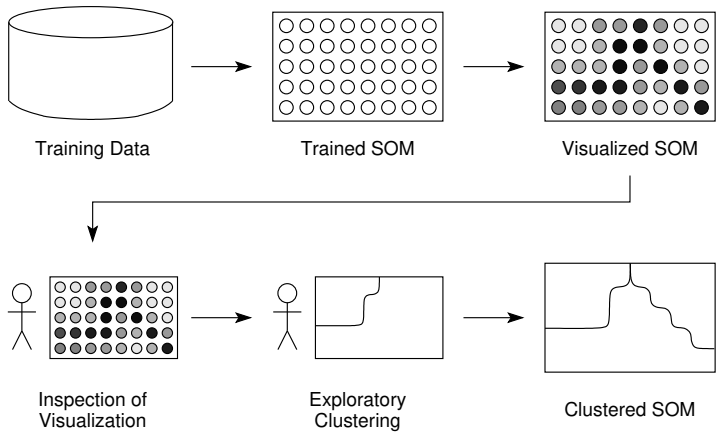


Figure 5.3: The overall procedure for the exploratory clustering process.

As is the case for the algorithmic clustering methods, exploratory cluster discovery may use either of the two emergent cluster structure indicators described in Section 5.1. The cluster indicator that is exploited depends on the map visualization used.

Using the inter-weight-vector similarity indicator, visually similar weight vectors indicate clusters, with dissimilar neighboring neurons forming cluster boundaries [257]:

- It is possible for an analyst to use either a grid-based representation augmented with similarity information (see Section 4.3.1), or an irregular map representation that intrinsically represents similarity information (see Section 4.4). However, the use of visualizations that directly represent algorithmically discovered clusters will clearly be senseless in a practical setting, due to the fact that such visualizations already represent the results of a cluster discovery process.

- Alternatively, visualizations that directly represent each neuron's entire weight vector (see Section 4.3.2.2) may highlight groups of similar weight vectors. Visualizing only a subset of the weight components will be insufficient in most practical situations, unless feature selection techniques are used to determine and display only the most informative weights. It should also be noted that complex representations and high-dimensional weight vectors may present interpretation difficulties.

Exploratory cluster discovery can also focus on the data example concentration cluster indicator [274]. In this case, neurons that are BMUs for many examples should

be identified as cluster members. Neurons that no (or very few) examples map to are considered to fall on the border between clusters. It is possible to use the data subset mapping visualizations that are mentioned in Section 4.3.3.2, provided that the number of data examples used is sufficient to adequately delineate cluster structure.

It is possible to use cluster evaluation measures to help guide a human analyst during an exploratory cluster discovery task. Exploratory clustering tools can display a cluster evaluation measure that is updated as the analyst assigns neurons to different clusters. The evaluation measure serves as a constantly updated numerical guide to the quality of the clustering that an analyst is in the process of creating. Evaluation measures also allow the comparison of several analyst's clusterings of the same map, allowing for the selection of the best discovered clusters for further EDA or DM analysis.

There are some very important drawbacks that are inherent to the use of any of the above-mentioned exploratory cluster discovery methodologies. These factors should always be carefully considered before such an exploratory approach is used in a practical setting, and may mean that an algorithmic approach could be more appropriate:

- The subjectivity of the human analyst's interpretation may become a concern [257]. Analysts may bias their interpretations towards examples that they expect to find, thus reducing the possibility of discovering interesting and unexpected clusters.

- The clustering process results are not reproducible. This is because every individual human analyst may cluster one map in a variety of different ways. Even a single analyst's results may vary over time due to factors such as mood and fatigue.

- The clustering process may become extremely tedious and time-consuming [257]. This is especially problematic when the clustering process involves SOMs that are very large in terms of map structure size or weight vector dimensionality.

However, despite the above concerns, an exploratory approach may allow for the discovery of clusters that algorithmic approaches cannot detect. For instance, an intrinsic algorithmic bias or an inherent uncertainty in a map's cluster structure might result in certain clusters being overlooked by a clustering algorithm. Additionally, a human analyst may denote certain clusters as irrelevant [104] (for example, due to a lack of interesting characteristics), which a purely algorithmic approach is incapable of.

5.5 Hybrid Cluster Discovery

Hybrid cluster discovery techniques simply combine a machine-based algorithmic method (i.e., an approach from Section 5.3) and a human-oriented exploratory technique (i.e., a method from Section 5.4). Hybrid cluster discovery methods use a clustering algorithm's output as a base, and a human analyst's insight to improve this output.

Figure 5.4 illustrates one hybrid cluster discovery approach, which uses interactive tools that allow an analyst to manually affect the operation of a clustering algorithm. Two simple examples, related to the methods described in Section 5.3, are:

- A hierarchical dendrogram can be interactively cut at varying levels on different branches, thus producing different clusterings [24]. This is shown in Figure 5.2.
- The output of partitional clustering algorithms can be changed through manual selection of varying k values, as well as the initial centroid values.

Another possible hybrid cluster discovery method, shown in Figure 5.5, requires manual post-modification of the cluster output produced by an algorithmic cluster discovery method. This cluster modification must be performed in an exploratory fashion, where a human expert re-assigns neurons to either existing or new emergent clusters.

Cluster quality evaluation measures may again be used in several ways to aid a hybrid cluster discovery exercise. Of course, quality evaluations are separately applicable to both the algorithmic and the exploratory components of a hybrid cluster discovery approach, as previously described. In addition, the following approaches are also possible:

- For both hierarchical and partitional methods, clusterings with quality values above or below (depending on the measure) a certain threshold may be added to a subset of interesting groupings that are worthy of further investigation [182, 257].
- Hierarchical clustering dendrograms may be pruned using such measures. Groupings that are indicated as uninteresting may be discarded from the tree, thus reducing the tree's size, and limiting the complexity of the discovery process.

Ultimately, the intention is that the use of hybrid cluster discovery approaches will allow for the exploitation of the advantages of both algorithmic and exploratory techniques, while mitigating the drawbacks associated with each methodology.

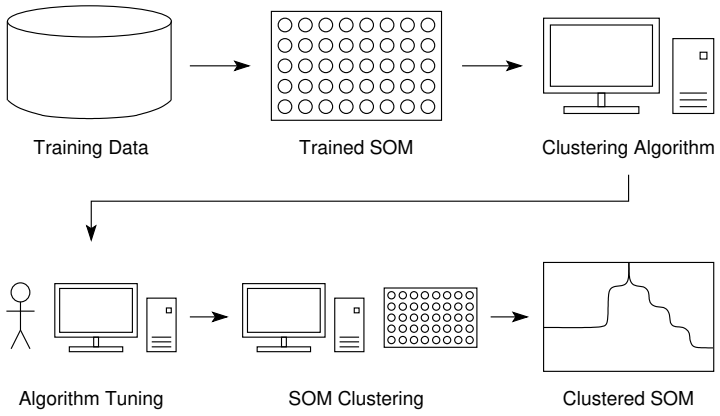


Figure 5.4: The overall procedure for a hybrid cluster discovery procedure, wherein a human data analyst performs a manual modification on the operation of a clustering algorithm.

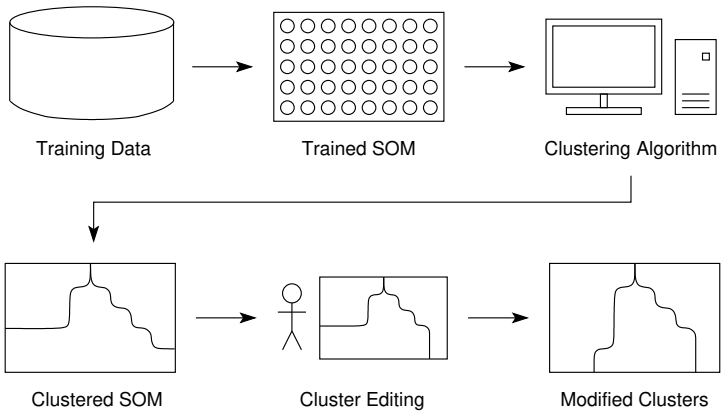


Figure 5.5: The overall procedure for a hybrid cluster discovery procedure, wherein a human data analyst performs a manual modification to the clusters produced by a clustering algorithm.

5.6 Cluster Stability and SOMs

Cluster stability [1, 13, 28, 264] encompasses the study of how closely the result of cluster discovery approximates the actual cluster structure present in the underlying data. For emergent SOMs, the map weight vectors constitute the clustered data.

Cluster stability studies usually theoretically analyze the stability (or lack of stability) that is intrinsic to a particular clustering approach. It is also possible to determine the

stability of a clustering, given a particular configuration of the clustering algorithm (including, for example, a particular value of k for partitional algorithms) [264]. Studies also focus on the nature of data that produces stable clusters [1]. The assumption in all cases is that a stable clustering should also represent clusters of good quality.

While there is no single definition for the concept of cluster stability, the following three general perspectives on stability have been identified in the literature [252]:

- Sets of perturbed weight vectors are generated. The clustering technique is applied to each perturbed set, producing a clustering per set. Finally, the generated clusterings are compared to one another. The smaller the differences between the clusterings, the more stable the discovered clusters are judged to be [264].

- Assuming that an inter-weight-vector distance measure is used as a cluster indicator, the second approach computes the size of the perturbation that can be applied to the distance measure, while still preserving the structure of the discovered clusters. The larger the perturbation, the more stable the clusters are [1].

- If all near-optimal clusterings on a data set (in terms of a clustering algorithm cost function) are assumed to also be close to the true cluster structure, finding a clustering that is approximately close to the true cluster structure is achievable in polynomial time, even if finding a near-optimal cost value is NP-hard [13].

While several analyses have been performed to investigate the intrinsic stability characteristics and shortcomings of the more established hierarchical [28] and partitional [1, 13, 264] clustering algorithms, the stability of emergent clusters in SOMs has not been the focus of much research. A literature review identified no publications that scrutinize the innate stability characteristics of SOM-based emergent clusters.

Most studies that do consider emergent SOM cluster stability report specific results in very narrow application areas, and thus the generality of these findings is unclear. Examples of such studies include work focused on microarray data [91, 218], gene expression data [41, 137], and three-dimensional body scans for medical applications [160].

Some SOM-based cluster stability results are presented only in the context of simple empirical comparisons to newly proposed algorithms, where the focus is the performance of the new approaches rather than that of the SOM. For example, Kim and Lee compare

SOMs and several other clustering approaches to a newly proposed ensemble clustering method [137], while Cabanes and Bennani [26] compare two proposed SOM-inspired algorithms (the S2L-SOM and DS2L-SOM approaches) and standard SOMs clustered using two techniques (k-means and single linkage agglomerative clustering).

Finally, some publications assess the stability of SOMs that differ from the standard stochastic approach discussed in this research. For example, one-dimensional SOMs [41], and two-dimensional non-emergent SOMs [91] have been studied. These findings are not general enough to be relevant for the methods covered in this dissertation's work.

5.7 Summary

This chapter gave an overview of techniques appropriate for the discovery of emergent neuron clusters. Emergent neuron cluster discovery is a requirement of several neuron labeling methodologies that are discussed in the following chapter. Section 5.1 introduced the overarching objective of emergent cluster discovery, and the main concepts underlying all cluster discovery techniques. Section 5.2 discussed quality evaluation techniques for discovered emergent clusters. The next three sections in this chapter focused on each of the main categories of cluster discovery approaches within SOMs: Algorithmic cluster discovery techniques were discussed in Section 5.3, Section 5.4 described exploratory cluster discovery, and Section 5.5 dealt with hybrid cluster discovery. Finally, Section 5.6 touched upon cluster stability in the context of emergent clusters in SOMs.

The following chapter focuses on the variety of neuron labeling techniques that are available for self-organizing maps. Such labeling techniques are integral to a large number of data mining and exploratory data analysis techniques that are based on SOMs.

Chapter 6

Map Neuron Labeling

The previous chapter described methods for discovering emergent clusters of neurons within a SOM's map structure. The global similarity encoded visualizations of Section 4.3.1.2, and most of the SOM-based DM methods in Chapter 7, rely on map neuron labels. This chapter discusses methods for applying labels to neurons.

Section 6.1 discusses the viewpoint on labeling that is taken by this research. Section 6.2 provides more detail on supervised labeling techniques, while Section 6.3 investigates unsupervised labeling methods. Section 6.4 focuses on the actual application of neuron labeling techniques to trained SOMs. Section 6.5 discusses neuron labeling with high-dimensional data sets. Finally, a chapter summary is provided in Section 6.6.

6.1 An Overview of Neuron Labeling

As was noted in Section 3.1.2, a SOM models the distribution of a training data set. This model, however, is purely defined by the distribution of weight vector components across the map structure, and is not characterized in any meaningful way.

Neuron labeling characterizes a SOM by associating textual labels with a subset of the map's neurons, while possibly leaving some neurons unlabeled. A neuron's label should describe the characteristics of the part of the data distribution modeled by the neuron's weight vector. Fuzzy labels are also possible [227], but are often difficult to interpret and verify. Fuzzy labeling is thus not considered further in this research.

This dissertation differentiates between two broad classes of neuron labeling techniques, based on whether supervision is provided by means of data classifications:

- *Supervised labeling methods* derive labels from the classification attribute values of examples. Labels are usually based on training examples, but examples from a separate labeling set can be used instead. If class information is unavailable, a human expert must manually provide classification attribute values.

- *Unsupervised labeling methods* use no intrinsic data classifications, and can thus function in the absence of class information. Labels are derived either from a human analyst's observations, directly from the SOM's weight vectors, or from data examples in conjunction with the map's weights.

Figure 6.1 illustrates the differences between supervised and unsupervised labeling, as well as the three main approaches that fall under the banner of unsupervised labeling. Supervised labeling is both commonly used and empirically verifiable, and is therefore the main focus of this study. In contrast to supervised approaches, the results of unsupervised labeling techniques are subjective, and are thus difficult to verify empirically. In addition, unsupervised labeling methods are much less prevalent in the literature.

The presence or lack of supervision within a labeling technique is unrelated to the SOM's level of training supervision. Supervised and unsupervised labeling techniques can both be applied to either unsupervised, supervised or semi-supervised SOMs. Section 6.4 discusses neuron labeling for SOMs with different levels of supervision.

EDA and DM exercises very often focus on neuron labels that are nominal, and thus discrete in nature. Such discrete labels are this chapter's main concern. This chapter also investigates continuous-valued map neuron labels [12]. However, it should be noted that continuous labels are often difficult to interpret and verify for accuracy.

6.2 Supervised Neuron Labeling

This research identifies three supervised neuron labeling techniques, namely *example-centric neuron labeling*, *example-centric cluster labeling*, and *weight-centric neuron labeling*. These methods are investigated in Sections 6.2.1 to 6.2.3. Section 6.2.4 describes

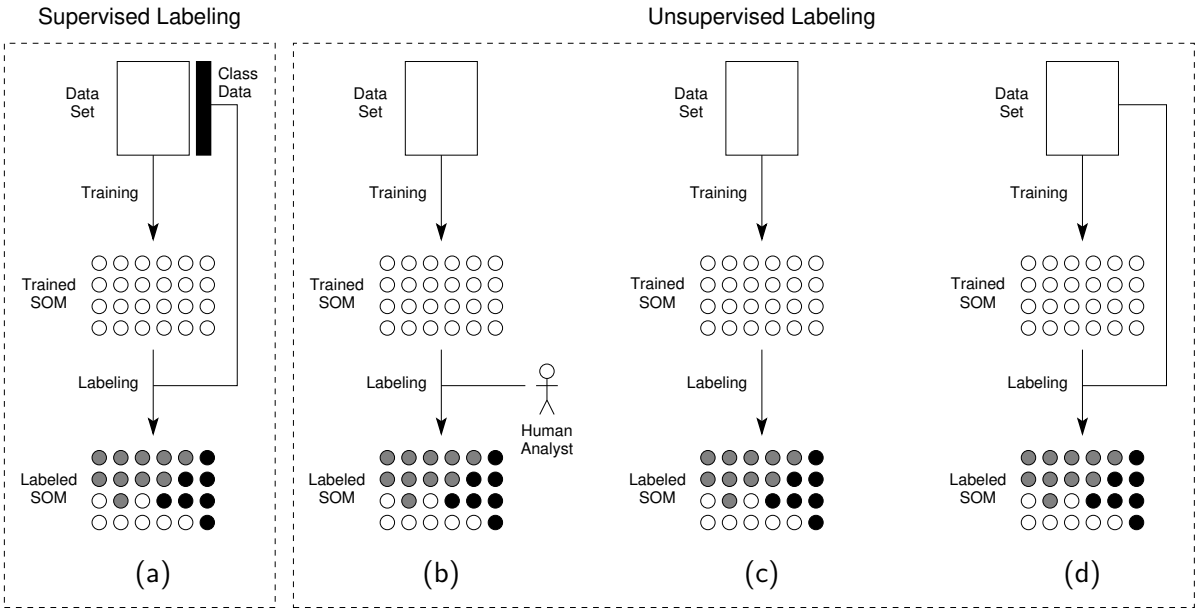


Figure 6.1: An overview of the main approaches to supervised and unsupervised neuron labeling: (a) shows supervised labeling; (b) shows unsupervised labeling that is guided by a human analyst; (c) shows unsupervised labeling based entirely on the map's weights; (d) shows unsupervised labeling based on the map's weights and an unclassified data set.

some esoteric variations on the three approaches, based on how multiple label mappings are handled. A taxonomy of these labeling methods is illustrated within Figure 6.2.

6.2.1 Example-Centric Neuron Labeling

Algorithm 6.1 shows example-centric neuron labeling [145]. Each neuron, n_{yx}, keeps a mapped example set, M_{yx}. A BMU is found for each labeling example, and the example is added to the BMU's mapped example set. Each neuron is then labeled with the majority classification in the associated mapped example set, with ties settled arbitrarily.

Example-centric neuron labeling has two advantages. Firstly, the approach is simple to implement. The method is also efficient, because the labeling data is iterated through only once, and each example's search is limited to a relatively small set of neurons.

However, a drawback of example-centric neuron labeling is that all non-BMU neurons remain unlabeled. A less serious concern is that the mapped example sets associated

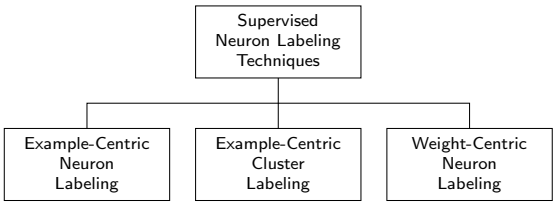


Figure 6.2: A taxonomy of SOM-based supervised neuron labeling techniques.

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence

for all *neurons* n_{yx} *in map* **do**
    Define an empty mapped example set, denoted M_{yx}
    Associate M_{yx} with n_{yx}
end for

for all *labeling example vectors* $\vec{z}_s$ **do**
    Determine the BMU for $\vec{z}_s$, denoted n_{yx}, over all neurons in *map*
    Add labeling example $\vec{z}_s$ to M_{yx}
end for

for all *neurons* n_{yx} *in map* **do**
    Determine the value of $\mathcal{A}_{cls}$ that is most common in M_{yx}
    Label n_{yx} with the $\mathcal{A}_{cls}$ value that was determined
end for

Algorithm 6.1: Pseudocode of the example-centric neuron labeling algorithm.

with every neuron on a map occupy memory resources, which will be problematic when very large labeling example sets require many examples to be stored.

The problem of unlabeled neurons is particularly acute when very large maps are labeled using a limited number of labeling examples. This is because the small number of examples will have relatively few BMUs, leading to fewer labeled neurons. This research therefore recommends that example-centric neuron labeling should be avoided during EDA, where a partially labeled map will hinder easy visual analysis.

The drawback of unlabeled neurons can be addressed by techniques that propagate labels from labeled neurons to adjacent unlabeled neurons [105]. Such methods have thus far not been investigated in relation to practical EDA or DM exercises, and are consequently not discussed in further detail within this dissertation.

This research work identifies three conditions that decrease the accuracy of individual labels that are assigned to neurons by means of example-centric neuron labeling:

- Labeling data examples containing many missing values have less accurate BMU matches, which will tend to result in mislabelings. It is consequently advisable to ensure that the majority of labeling examples are as complete as possible.

- Label accuracy is compromised for neurons with very few mapped data examples, particularly when a label is based on only one data example. Such a neuron's label is based on a less representative sample of data examples, and the label has a higher probability of being assigned due to chance mappings. This research thus suggests analyzing an example-centric neuron labeling in conjunction with a data subset mapping visualization, such as the data histogram described in Section 4.3.3.2.

- A label is of questionable quality when several classes are almost equally prevalent within the label's mapped examples (e.g., out of ten mapped examples, five belong to class A, four to class B, and one to class C). For such a neuron, there is no clear distinction between the most prevalent classifications (A and B in the example), and the neuron effectively represents multiple classes. It is possible to use a map-based information visualization (these are discussed in Section 4.4.2) that shows mapped examples for individual neurons, in order to identify such labels.

Figure 6.3 (a) illustrates a data histogram visualization of the examples used to train the example SOM presented in Figure 4.1. In the histogram, a darker shade indicates a larger number of mapped examples, and the lightest shade denotes that no examples have been mapped to a neuron. The histogram should be compared to the example-centric neuron labeling, which is shown in Figure 6.3 (b), and was performed using the full training data set. The labels SET, VER and VIR respectively correlate to the `Iris_Setosa`, `Iris_Versicolor` and `Iris_Virginica` classes. Clearly, the neurons with no mapped examples receive no labels, giving the labeling a fragmented appearance that is relatively

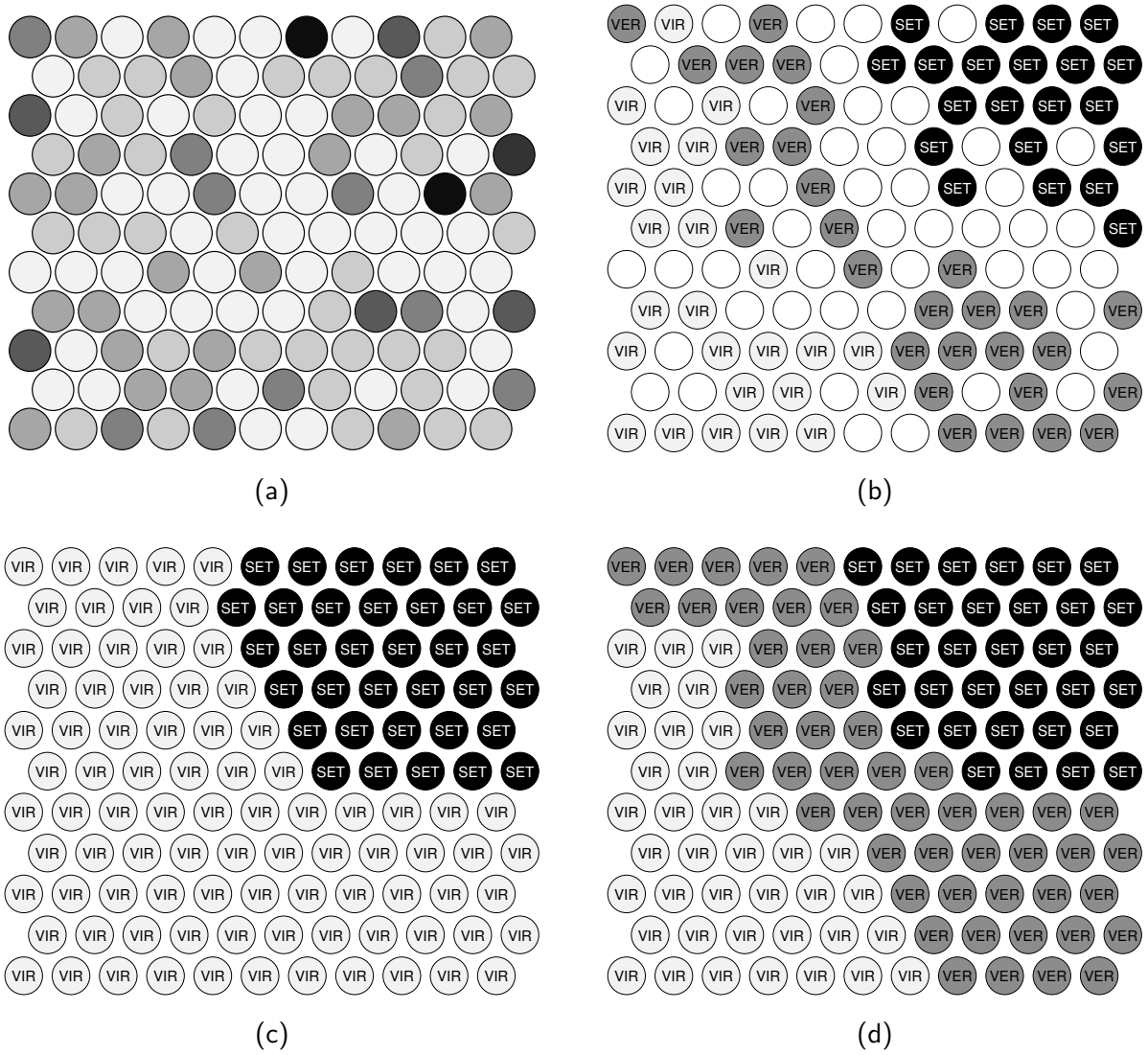


Figure 6.3: Supervised labelings for the Iris data set SOM of Figure 4.1: (a) shows a data histogram; (b) shows example-centric neuron labeling; (c) shows example-centric cluster labeling, using SOM-Ward clustering; (d) shows weight-centric neuron labeling. The `Iris_Setosa`, `Iris_Versicolor` and `Iris_Virginica` classes are respectively labeled SET, VER and VIR.

difficult to visually interpret. Even so, it is possible to discern that `Iris_Setosa` labels tend to be grouped in the upper right of the map, while `Iris_Virginica` labels can be found in the lower left portion of the map, and `Iris_Versicolor` labels generally occupy the space between the `Iris_Setosa` and `Iris_Virginica` groups.

Kayacık and Zincir-Heywood [135] propose an approach that extends example-centric neuron labeling, where labels are not based on unique mappings between data examples and BMUs. Instead, every neuron has a continuous-valued counter for each classification in the training data set. Each labeling example is then presented to the map, and the five neuron weight vectors that are closest to the example are found. These neurons are then ranked according to distance from the labeling example (with a rank of one for the BMU). Finally, for each of the selected neurons, the counter for the classification of the labeling example is incremented by a value that is inversely proportional to the neuron's rank. This approach does not seem to be used beyond the original paper. Furthermore, no analysis in relation to standard example-centric neuron labeling has been done, and the rationale behind the number of neurons considered for each mapping is not clear. Investigations into the effectiveness of this method are thus left to future work.

6.2.2 Example-Centric Cluster Labeling

Example-centric cluster labeling [211] is outlined as pseudocode in Algorithm 6.2. As an initial step, example-centric cluster labeling requires the discovery of emergent neuron clusters within a trained map, as described in Chapter 5. Each discovered cluster is labeled as a whole. While example-centric neuron labeling keeps a set of examples for each neuron, example-centric cluster labeling links a mapped example set, N_i, with each cluster, S_i. Labeling examples are added to the mapped example set of the cluster in which the example's BMU falls. The majority classification in a cluster's mapped example set then labels all the cluster's neurons, with ties settled arbitrarily.

Within an example-centric cluster labeling system, clusters that contain no BMUs remain unlabeled. This phenomenon of unlabeled neurons is similar to the main drawback of example-centric neuron labeling. However, the aggregating effect of emergent clusters minimizes the number of unlabeled neurons by grouping and labeling large sets of map neurons together. It is therefore likely that an example-centric cluster labeling will leave fewer unlabeled neurons than an example-centric neuron labeling.

As for example-centric neuron labeling, one drawback of example-centric cluster labeling is the potentially large memory requirements of the mapped example sets, particularly when large numbers of examples are used for labeling. This drawback is somewhat

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence
Derive a discrete set of emergent clusters, $\mathcal{L} = \{S_1, S_2, \ldots, S_k\}$, of all $\vec{w}_{yx}$ in the map

for all *clusters* $S_i \in \mathcal{L}$ **do**
    Define an empty mapped example set, denoted N_i
    Associate N_i with S_i
end for

for all *labeling example vectors* $\vec{z}_s$ **do**
    Determine the BMU for $\vec{z}_s$, denoted n_{yx}, over all neurons in *map*
    Determine emergent cluster S_i, such that the BMU of $\vec{z}_s$ is in S_i, and add $\vec{z}_s$ to N_i
end for

for all *clusters* $S_i \in \mathcal{L}$ **do**
    Determine the value of $\mathcal{A}_{cls}$ that is most common in N_i
    Label all $n_{yx} \in S_i$ with the $\mathcal{A}_{cls}$ value that was determined
end for

Algorithm 6.2: Pseudocode of the example-centric cluster labeling algorithm.

mitigated by the fact that fewer mapped example sets need to be maintained, in comparison to example-centric neuron labeling. Another drawback of example-centric cluster labeling is that the cluster discovery step adds time complexity to the method.

A more subtle drawback associated with example-centric cluster labeling arises when at least one heterogeneous cluster exists, such that two or more inseparable example classes are modeled within a single emergent cluster. Example-centric cluster labeling applies a uniform labeling to such a heterogeneous cluster. This type of uniform labeling eliminates any distinction between the classes of examples that map to the cluster.

The same situations that adversely affect example-centric neuron label accuracy also apply to example-centric cluster labeling. Using labeling examples with many missing values impacts label accuracy, but this is ameliorated by the large numbers of mapped examples per cluster, which average out small numbers of mis-mapped examples. Clusters with very few mapped labeling examples also have suspicious labels. Furthermore, label

quality should be questioned when multiple classes have close to the maximum number of mapped labeling examples for a cluster. The last two situations are highlighted by map-based information visualizations that show map meta-data for the emergent clusters, similar to the histogram visualization that was shown in Figure 4.13 (b).

Figure 6.3 (c) illustrates a pathological example of a heterogeneous cluster, with reference to the Iris data set SOM example of Figure 4.1. SOM-Ward clustering discovered two emergent clusters of neurons: one in the upper right of the map, containing mostly `Iris_Setosa` examples, and one heterogeneous cluster to the lower left of the map, containing predominantly `Iris_Virginica` and `Iris_Versicolor` examples. Example-centric cluster labeling uniformly labels all neurons that model the two inseparable classes. However, when compared to the example-centric neuron labeling in Figure 6.3 (a), the example-centric cluster labeling leaves no neurons unlabeled, which results in an unfragmented map appearance that is more suitable for visual EDA.

6.2.3 Weight-Centric Neuron Labeling

Weight-centric neuron labeling [145] is based on each neuron's best matching example (BME). The BME of a neuron is the training or labeling data example that is closest to the neuron's weight vector, usually in terms of Euclidean distance, and is defined as:

$$\|\vec{w}_{yx} - \vec{z}_e\|_2 = \min_{\forall p}\big\{\|\vec{w}_{yx} - \vec{z}_s\|_2\big\} \tag{6.1}$$

where $\vec{w}_{yx}$ denotes the weight vector of the neuron for which the BME is calculated, $\vec{z}_e$ denotes the training or labeling example vector of the BME, and $\vec{z}_s$ denotes an arbitrary training or labeling example vector. Using a similar approach as the BMU calculation procedure, missing attribute values can be easily dealt with by simply calculating distances using only the vector components that are available within each $\vec{z}_s$.

It should, however, be noted that BMEs will clearly be less accurate in the presence of a large number of missing values. Consequently, it is sensible to require that all training or labeling examples used to calculate a BME should have no (or at least very few) missing attribute values. The remainder of this dissertation assumes that there are no missing attribute values present in any examples that are used to calculate a BME.

The weight-centric neuron labeling approach is outlined in terms of pseudocode within Algorithm 6.3. Weight-centric labeling essentially reverses the strategy that is used by both example-centric neuron labeling and example-centric cluster labeling. Instead of mapping labeling examples to map neurons or emergent clusters, weight-centric labeling determines a BME for each neuron on the map. Finally, the class of each BME simply becomes the label that is associated with the neuron that the BME maps to.

Like example-centric neuron labeling, weight-centric neuron labeling is efficient, because the map neurons are iterated through only once, and a limited BME search takes place per neuron. Because no mapped example sets must be stored and searched, weight-centric neuron labeling has an additional performance and memory advantage over the example-centric methods. As Algorithms 6.1, 6.2, and 6.3 illustrate, the weight-centric approach is also easier to implement than both example-centric methods. Another advantage associated with weight-centric labeling is that a label is guaranteed for every neuron, which will be of assistance during visual exploratory analysis of a map.

Figure 6.3 (d) illustrates a weight-centric neuron labeling applied to the example Iris data set SOM of Figure 4.1. When this label assignment is compared to the example-centric neuron labeling shown in Figure 6.3 (b), it is clear that both approaches can distinguish between the three class regions on the map. However, the weight-centric technique labels every neuron, producing a less fragmented visualization.

In contrast to example-centric cluster labeling, weight-centric neuron labeling has the advantage of being able to function well in the presence of heterogeneous emergent clusters. This is because weight-centric labeling does not rely on the aggregating effect of neuron clusters. A comparison of Figure 6.3 (c) and (d) shows how weight-centric labeling distinguishes between neurons representing the inseparable `Iris_Virginica` and `Iris_Versicolor` classes, even though both classes fall in the same cluster.

Unfortunately, weight-centric labeling provides an inaccurate neuron label if a neuron's BME is a very poor match (i.e., the neuron's weight vector and the BME are far apart). Neurons on the boundaries between emergent clusters often have poor BME matches, as no data examples map to such neurons. Weight-centric labels on cluster boundaries should thus be analyzed before being used for EDA or DM. An appropriate map visualization, such as a U-matrix, can help identify boundary neurons.

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence

for all *neurons* n_{yx} *in map* **do**
    Determine the BME for n_{yx}, denoted $\vec{z}_e$, according to Equation (6.1)
    Determine the value of $\mathcal{A}_{cls}$ for BME $\vec{z}_e$, and label n_{yx} with this value
end for

Algorithm 6.3: Pseudocode of the weight-centric neuron labeling algorithm.

The above discussion suggests that example-centric cluster labeling and weight-centric neuron labeling should be preferred to example-centric neuron labeling during EDA, because example-centric neuron labeling often leaves a map only partially labeled. Furthermore, example-centric cluster labeling cannot accurately characterize real-world data containing inseparable classes, which weight-centric neuron labeling can handle. This research thus recommends weight-centric neuron labeling for EDA. Chapter 8 investigates how well the labeling approaches perform within a DM context.

Valero *et al* [245] propose performing a weight-centric neuron labeling on a SOM, and using each neuron's weight vector and label to train a Support Vector Machine (SVM) [40], which learns the SOM's labeling. The weight vectors of unlabeled SOMs are later presented to the SVM, which predicts a label for each weight vector's neuron. The SVM is thus used to label these SOMs without requiring further labeled data examples. This approach has not been applied beyond the field of sound event recognition. The method also assumes that the labeled SOM models the same data space as any unlabeled SOMs to which the SVM is applied, which is not necessarily the case. In a broader sense, the efficacy of applying a supervised learning model, such as an SVM, to SOM neuron labeling has not been demonstrated nor adequately analyzed in relation to other labeling approaches. This technique is therefore ignored for the remainder of this dissertation.

6.2.4 Supervised Labeling using Multiple Label Mappings

Example-centric neuron and cluster labeling map multiple data examples to, respectively, individual neurons and emergent clusters. Weight-centric neuron labeling maps single

BMEs to neurons, but can be modified to choose each label as the most common class amongst a neuron's $\bar{q}$ nearest examples [12]. This adapted weight-centric method thus maps multiple examples, in a similar fashion to the example-centric techniques.

Variations on neuron label selection are also possible [12]. The following variations are applicable to any of the above-mentioned supervised labeling methods that are based on multiple mappings of labeling data set examples to map neurons or clusters[1]:

- The most basic label selection variation simply selects a random class label from the set of mapped examples. The random nature of this approach is likely to result in inaccurate labels, and this approach should therefore be avoided.

- Each set of mapped examples is ranked from nearest to furthest from a neuron weight vector or cluster centroid. Examples are given weights that are inversely proportional to the associated rank. Finally, the classification with the highest weighted sum of appearances amongst the mapped examples become neuron or cluster labels. Better matching classifications thus have greater contributions.

- A set of sub-labels can be associated with each labeled neuron or cluster. The sub-label set contains *every* unique classification that occurs within the set of labeling examples that map to the neuron or cluster being labeled. Crude labels are produced when examples of many different classes map to the same labeling location on the map (e.g., when the set of labeling examples is noisy).

- Again, neurons or clusters can be labeled with a set of sub-labels. However, each sub-label set contains only those classifications that appear at least a user-defined *percentage of times* amongst the labeling examples mapping to the labeled neuron or cluster. Again, noisy labeling sets give crude labels. Also, the management of the percentage threshold parameter complicates the labeling process.

The two last-mentioned approach variations usually introduce multiple neuron sub-labels. A large set of sub-labels is often more difficult to interpret than single labels, and may therefore be considered inappropriate within a practical EDA or DM setting.

[1] The original literature [12] describes these variations only in the context of the modified weight-centric method. However, all three variations are also valid for both example-centric labeling approaches.

It should be noted that the variations based on multiple example mappings are not widely used in practice, nor have the performance characteristics of these methods been studied in detail. The investigation of these extensions is thus left to future work.

6.3 Unsupervised Neuron Labeling

This work defines four unsupervised labeling method classes, shown in Figure 6.4: *exploratory labeling*, which is discussed in Section 6.3.1, *unique cluster labeling*, which Section 6.3.2 describes, *unsupervised weight-based labeling*, which is covered in Section 6.3.3, and *unsupervised example-based labeling*, which Section 6.3.4 investigates.

Unique cluster labeling, and unsupervised weight- and example-based labeling are all algorithmic techniques that perform automatic neuron label assignment. These algorithmic unsupervised labeling techniques should be used when non-subjective labeling is required in a short amount of time, or when many maps must be labeled.

This dissertation generalizes several similar techniques into the unified categories of unsupervised weight-based and example-based labeling. Approaches in both categories build labels from the components of a SOM's model, which is non-trivial, and means that these labeling methods are the most complex covered by this research.

6.3.1 Exploratory Labeling

Figure 6.5 illustrates exploratory labeling [39], which is driven by a human expert's analysis of a trained SOM. The approach is similar to the exploratory cluster discovery that Section 5.4 describes. An expert's analysis is guided by the map visualizations described in Chapter 4, after which neurons or neuron groups must be assigned labels manually. An interactive tool is usually used during exploratory labeling.

Exploratory cluster labeling has the same drawbacks as exploratory cluster discovery. Firstly, the human analyst's results are subjective and possibly biased. Secondly, exploratory labels are typically non-reproducible, as different analysts' assessments are likely to vary [227]. Finally, the exploratory procedure is often very time-consuming [227]. Despite these drawbacks, it is often possible for the human insight of an exploratory approach to produce results superior to those of algorithmic labeling methods.

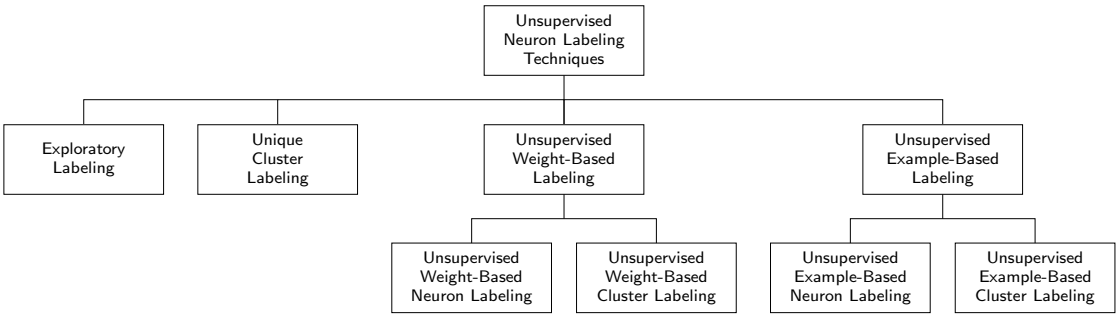


Figure 6.4: A taxonomy of SOM-based unsupervised neuron labeling techniques.

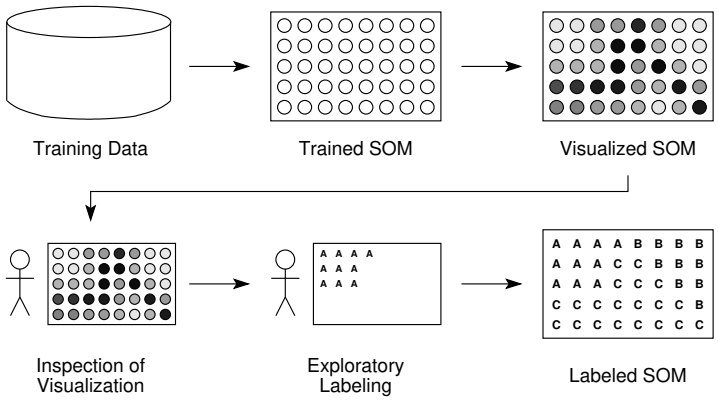


Figure 6.5: The overall procedure for the exploratory neuron labeling process.

6.3.2 Unique Cluster Labeling

Algorithm 6.4 illustrates unique cluster labeling [51]. The technique uses a set of discovered emergent clusters as a basis for labeling. An algorithm simply assigns unique machine-generated labels, such as `cluster_1` and `cluster_2`, to each individual emergent cluster on a map. Finally, all the neurons that make up an emergent cluster are given the same label as the cluster. This process is repeated for every cluster.

Clearly, unique cluster labels simply differentiate emergent clusters from one another. The labels are in no way descriptive of the underlying characteristics of clusters or neurons. The intention of unique cluster labeling is therefore typically to provide only basic assistance in the analysis of the structure of unique emergent map clusters. Deeper analysis requires the use of visualizations, which must be compared to the labeled areas.

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence
Derive a discrete set of emergent clusters, $\mathcal{L} = \{S_1, S_2, \ldots, S_k\}$, of all $\vec{w}_{yx}$ in the map

for all *clusters* $S_i \in \mathcal{L}$ **do**
    Generate a unique cluster label, unassigned to any clusters in $\mathcal{L}$, denoted L_i
    Label all $n_{yx} \in S_i$ with cluster label L_i
end for

Algorithm 6.4: Pseudocode of the unique cluster labeling algorithm.

6.3.3 Unsupervised Weight-Based Labeling

Unsupervised weight-based labeling methods define neuron sub-labels, which are based on a subset of the map's attributes. Chosen attributes are judged to be informative according to a significance statistic that is based on the attribute's weight value.

Weights must usually be normalized, so that components are fairly compared. Such weights are produced by normalizing training data attribute values before training. Alternatively, weights must be normalized after training. Because attributes with large domains dominate during training, this work recommends the former method.

The complexity of multiple sub-labels is often undesirable for EDA or DM. Even if appropriate, too many sub-labels reduce interpretability. Analysts must thus subjectively limit the number of sub-labels, often using a maximum sub-label count per neuron. Several labelings can also be produced, from which an appropriate one is chosen.

This research identifies two classes of unsupervised weight-based labeling methods: *unsupervised weight-based neuron labeling*, which is discussed within Section 6.3.3.1, and *unsupervised weight-based cluster labeling*, which Section 6.3.3.2 describes. The advantages and drawbacks of the two approaches are contrasted in Section 6.3.3.3.

6.3.3.1 Unsupervised Weight-Based Neuron Labeling

Algorithm 6.5 illustrates unsupervised weight-based neuron labeling [155, 217]. The approach performs three steps, each of which is elaborated upon separately. Finally, several examples of unsupervised weight-based neuron labeling are presented and discussed.

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence

for all *neurons* n_{yx} *in map* **do**
    for all *attributes* A_l *represented by a weight in* $\vec{w}_{yx}$ **do**
        Use weight w_{yxl} associated with A_l to compute $sig(A_l, n_{yx})$
        Associate the new significance value with A_l in n_{yx}
    end for
end for

for all *neurons* n_{yx} *in map* **do**
    for all *sufficiently significant attributes* A_l *with corresponding* w_{yxl} **do**
        Build a sub-label using the name of A_l and value of w_{yxl}
        Add the new sub-label to the label for n_{yx}
    end for
end for

Algorithm 6.5: Pseudocode of the unsupervised weight-based neuron labeling algorithm.

Step 1: Compute Attribute Significance Values for Neurons

The operation of unsupervised weight-based neuron labeling is primarily based on a significance statistic, which is denoted $sig(A_l, n_{yx})$. This significance statistic computes a value representing the significance of attribute A_l within neuron n_{yx}. The statistic must be based on the continuous weight value w_{yxl}. Attributes with values indicating higher significance are chosen to become neuron sub-labels in the following step.

Many $sig(A_l, n_{yx})$ statistics are possible. Serrano-Cinca's absolute weight value significance [217] considers very high and very low weight values to be dominant. If weights are normalized to a $[-1.0, 1.0]$ range, absolute weight value significance is defined as:

$$sig(A_l, n_{yx}) = |w_{yxl}| \tag{6.2}$$

The absolute value is thus in the range $[0.0, 1.0]$, with higher values denoting greater significance. Lagus and Kaski [155] propose a keyword-based significance measure specifically for SOMs trained on text documents, which has not been applied more generally.

Step 2: Select Informative Attributes for Neurons

During this step of the labeling procedure, a chosen $sig(A_l, n_{yx})$ measure assigns a significance value to each attribute associated with every neuron on the map. Next, an attribute selection mechanism is used in order to identify one or more sufficiently significant attributes for inclusion in each neuron's sub-labels. The attribute selection mechanisms appropriate for unsupervised weight-based neuron labeling include the following:

- The simplest mechanism is to select only the single most significant attribute for each neuron [217]. However, single-attribute labels often lose too much descriptive detail, because neurons are usually characterized by several attributes.
- Another simple method chooses a user-defined number of the most significant attributes per neuron. Unfortunately, this technique is crude because the required number of informative attributes is often unclear. In addition, adequate descriptions of different neurons typically require varying numbers of attributes.
- A more sensible method is to select all attributes with at least a user-specified significance level. In addition, to guarantee a label for every neuron, each neuron label can be constrained to include at least the single most significant attribute, regardless of what the attribute's actual associated significance value may be.
- Another approach selects attributes with highly characteristic significance values within each neuron. Such components can be identified for absolute weight value significance, as the attributes with significance values greater than one positive standard deviation from the mean significance over all the neuron's attributes.
- It is possible to select attributes with significance values that are distinctive across the whole map. Using absolute weight value significance, for example, such components are identifiable by significance values that differ by more than one positive standard deviation from the attribute's mean significance value across the map.

Step 3: Associate Values with Selected Attributes

Neuron sub-labels that consist only of selected attribute names are possible, but usually not sufficiently detailed. It is therefore prudent to combine each sub-label's attribute

name with a value. Sub-label values should reveal details such as whether a selected attribute has a very high or a very low value. The most basic sub-label values are simply the weights corresponding to the selected sub-label attributes for each neuron.

Unfortunately, sub-label values based on raw weights are typically fairly difficult for a human data analyst to interpret, particularly if the neuron weight values are normalized. Consequently, in order to improve the readability of sub-label values for unsupervised weight-based neuron labeling, two simple post-processing operations are available:

- If data normalization was performed either on a map's training data or weight vectors, it is recommended that sub-label values be appropriately de-normalized to the original training data component value ranges. This post-processing step is generally simple and efficient to perform. De-normalized sub-label values have more meaningful interpretations, because the label values are more directly related to the map's original training data. As an example, Equation (2.2) is used to de-normalize map weights that are normalized using min-max normalization.

- Thresholds can be used to aggregate raw weight values into broad categories [217]. A simple scheme respectively labels the lower, upper, and middle third of an attribute value range as high, medium, and low. While appropriate threshold selection is not this work's focus, different thresholds generally result in different label interpretations (a similar problem arises when binning attribute values [103]). Thresholds are easier to understand than raw values, allow for quicker analysis of large map areas, and facilitate more general deductions about a map's characteristics.

Examples: Unsupervised Weight-Based Neuron Labeling

Figure 6.6 shows examples of the Iris data set SOM from Figure 4.1, labeled with unsupervised weight-based neuron labeling using absolute weight value significance. Min-max normalization was used on the SOM's training data. The sub-labels sl, sw, pl and pw respectively denote the components `sepal_length`, `sepal_width`, `petal_length`, and `petal_width`. Figure 6.6 (a) shows the map's emergent clusters, using a U-matrix.

Figure 6.6 (b) illustrates an example of a labeling using only the most significant attribute and associated weight value, where all label values have been de-normalized to

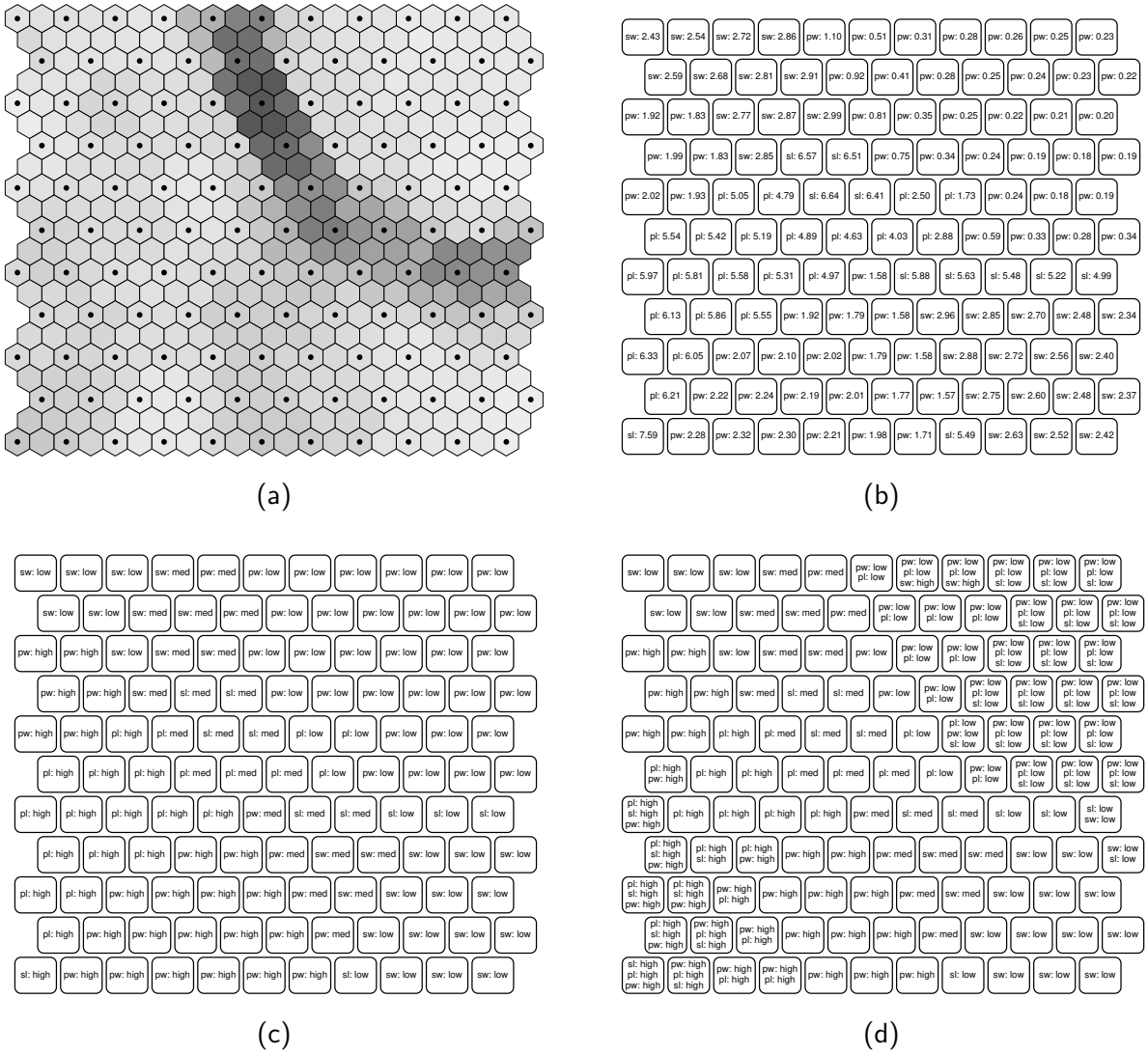


Figure 6.6: The Iris data set SOM of Figure 4.1, with unsupervised weight-based neuron labels using the absolute weight value significance of Equation (6.2). The components `sepal_length`, `sepal_width`, `petal_length`, and `petal_width` are respectively denoted sl, sw, pl, and pw: (a) shows the map's U-matrix; (b) shows neurons labeled with only the most significant weight value, de-normalized to the original attribute value range; (c) shows neurons labeled with only the most significant component value threshold (the low, med, and high labels respectively denote values in the lower, middle, and upper third of each attribute range); (d) shows map neurons labeled using all the components with a significance above 50% (where at least one label is required for each neuron), using the low, med, and high value thresholds.

the appropriate original attribute ranges. While raw weight values provide some insight into the map structure, it is clear that interpretation by humans is fairly difficult.

Figure 6.6 (c) shows the same labeling as Figure 6.6 (b), but uses threshold-based labels instead of raw weight values. The thresholds used are low, med and high. These labels respectively denote the lower, middle and upper third of each weight's range, which was computed across the entire map. In comparison to Figure 6.6 (b), general emergent map structures are clearly far easier to visually discern within Figure 6.6 (c).

Finally, Figure 6.6 (d) shows an example of a labeling scheme where multiple sub-labels are assigned per neuron, using the same threshold-based labels as in Figure 6.6 (c), where all attributes with significances above a 50% threshold are selected. To guarantee labels for all neurons, at least one attribute is chosen for each neuron, regardless of the attribute's significance. The labeling in Figure 6.6 (d) gives a more detailed description for neurons that are characterized by several highly-informative attributes.

6.3.3.2 Unsupervised Weight-Based Cluster Labeling

Unsupervised weight-based cluster labeling is a novel proposal of this research, and is summarized in Algorithm 6.6. In contrast to unsupervised weight-based neuron labeling, unsupervised weight-based cluster labeling derives sub-labels from the weight vectors in emergent neuron clusters, rather than from individual neurons. The technique has four steps, which are discussed separately, followed by examples of the labelings produced.

Step 1: Discover Emergent Clusters

As an initial step, emergent clusters of neurons must be discovered. Any of the techniques described in Chapter 5 are appropriate for use during this phase. Naturally, if an entirely automatic labeling process is required, an algorithmic clustering method should be used.

Step 2: Compute Significance Values for Emergent Clusters

In a similar fashion to unsupervised weight-based neuron labeling, a statistical attribute significance measure is required. For unsupervised weight-based cluster labeling, this significance measure is denoted as $sig(A_l, S_i)$, and calculates a numerical significance

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence
Derive a discrete set of emergent clusters, $\mathcal{L} = \{S_1, S_2, \ldots, S_k\}$, of all $\vec{w}_{yx}$ in the map

for all *clusters* $S_i \in \mathcal{L}$ **do**
    for all *attributes* A_l *represented by a weight in* $\vec{w}_{yx}$ **do**
        Use weight w_{yxl} associated with A_l, for all $\vec{w}_{yx} \in S_i$, to compute $sig(A_l, S_i)$
        Associate the new significance value with A_l in S_i
    end for
end for

for all *clusters* $S_i \in \mathcal{L}$ **do**
    for all *sufficiently significant attributes* A_l *with corresponding* w_{yxl} **do**
        Build a sub-label using the name of A_l and value of w_{yxl} over S_i
        Add the new sub-label to the label of each $n_{yx} \in S_i$
    end for
end for

Algorithm 6.6: Pseudocode of the unsupervised weight-based cluster labeling algorithm.

value for attribute A_l within cluster S_i. The $sig(A_l, S_i)$ measure bases the computation of the significance value on all the weight values that correspond to attribute A_l, measured over the entire emergent cluster S_i. Each attribute within every emergent cluster is assigned a significance value, which is computed using the chosen measure.

Several statistical measures [47] are possible for $sig(A_l, S_i)$, although each must be suited to continuous value analysis. This work identifies only three possible significance measures: the simplest is based on absolute weight values, a second focuses on the standard deviation of weights within emergent clusters, and a final measure [5, 6] uses the Kolmogorov-Smirnov (or K-S) statistic [151]. This research focuses neither on a detailed discussion dealing with the operational characteristics of these proposed measures, nor on the advantages and drawbacks associated with each method. A detailed theoretical and empirical analysis of significance measures is thus deferred to future work.

The absolute weight value $sig(A_l, S_i)$ measure simply extends Serrano-Cinca's neuron-based absolute weight value measure [217], by using mean weight values over emergent

clusters, rather than weight values from single neurons. Again, weight values with very high or very low values are assumed to be dominant features, and all weights must be in the range $[-1.0, 1.0]$. Cluster-based absolute weight value significance is calculated as:

$$sig(A_l, S_i) = |mean(w_{yxl}, S_i)| \tag{6.3}$$

where $mean(w_{yxl}, S_i)$ denotes the mean value of weight value w_{yxl} calculated over all the neuron weights making up the discovered emergent cluster S_i, and w_{yxl} is the weight that corresponds to attribute A_l. The mean value for the weight w_{yxl} within emergent neuron cluster S_i is a trivial measure, which is simply computed as follows:

$$mean(w_{yxl}, S_i) = \frac{1}{o_i} \cdot \sum_{\vec{w}_{yx} \in S_i} (w_{yxl}) \tag{6.4}$$

where o_i is the number of neurons in neuron cluster S_i. The absolute weight value significance measure for unsupervised weight-based cluster labeling produces a value in the range $[0.0, 1.0]$, where higher values denote attributes with greater significance.

The second $sig(A_l, S_i)$ measure ascribes higher significance to weights with values that have a low sample standard deviation over a cluster, because such weights have fairly constant characteristic values within the cluster. The measure is defined as:

$$sig(A_l, S_i) = -\sqrt{\frac{1}{o_i - 1} \cdot \sum_{\vec{w}_{yx} \in S_i} \big(w_{yxl} - mean(w_{yxl}, S_i)\big)^2} \tag{6.5}$$

where $mean(w_{yxl}, S_i)$ is calculated according to Equation (6.4). Higher values for this significance measure indicate a greater significance. The measure has a maximum value of 0.0, which denotes a weight that is perfectly constant over a neuron cluster.

Alhoniemi and Simula [5, 6] suggest using the K-S statistic [141] to assess attribute significance. The K-S statistic was originally developed by Kolmogorov [151], and later refined by Smirnov [224]. The $sig(A_l, S_i)$ measure that is based on the K-S statistic compares the cluster S_i to a set of clusters, $out(S_i)$. The $out(S_i)$ set is defined as:

$$out(S_i) = \left(\bigcup_{j=1}^{k} S_j \right) \setminus S_i \tag{6.6}$$

where k is the number of emergent neuron clusters that have been discovered. The set $out(S_i)$ is thus the combination of all the map's emergent clusters, other than S_i.

An algorithmic implementation of the K-S statistic, by Press *et al* [188], is often used in practice. A higher statistic value indicates a weight with a cumulative distribution over S_i that differs greatly from the weight's cumulative distribution over $out(S_i)$, meaning that the corresponding attribute is unusual, and therefore highly informative.

Step 3: Select Informative Attributes for Emergent Clusters

The third phase of unsupervised weight-based cluster labeling uses the significance values computed during the previous step to guide a suitable attribute selection mechanism. Selected attributes must be the ones judged to be more significant, based on $sig(A_l, S_i)$. The selected attributes are then used to build sub-labels for emergent clusters.

The attribute selection mechanisms for unsupervised weight-based cluster labeling are similar to the unsupervised weight-based neuron labeling attribute selection methods, although attributes are selected for clusters instead of for individual neurons:

- Only the most significant attribute in a cluster, or a user-specified number of the most significant attributes per cluster, can be selected to form sub-labels. As is the case for unsupervised weight-based neuron labeling, both these techniques are generally too simplistic to produce accurate and sufficiently detailed sub-labels.

- It is also possible to select attributes for each cluster, such that the significance of each selected attribute reaches at least a user-defined threshold of significance. As for unsupervised weight-based neuron labeling, labels can be ensured for all neurons by selecting at least one attribute for each emergent neuron cluster.

- Another selection method identifies attributes with very characteristic significance values within a cluster [12]. One method identifies such attributes as those in cluster S_i with $sig(A_l, S_i)$ values that are more significant than the mean $sig(A_l, S_i)$ value for all the attributes within cluster S_i, by at least one standard deviation.

- A final selection method chooses, for each cluster, attributes with significance values that differ substantially from the map's other clusters. For example, $sig(A_l, S_i)$ values that are more significant by at least one standard deviation than the mean $sig(A_l, S_i)$ value across all clusters will identify such characteristic attributes.

Step 4: Associate Values with Selected Attributes

Because labeling neurons with only the selected attribute names is typically insufficient, a representative value is usually computed for each selected attribute. One simple representative value is the mean of the weight corresponding to A_l, measured over the cluster. Each selected attribute's name and representative value are combined to form cluster sub-labels, which are also associated with the cluster's member neurons.

When data normalization has been performed prior to SOM training, sub-label values should be de-normalized to the attribute value ranges present in the original training data, to aid sensible interpretation of the labels. Thresholds allow for the aggregation of sub-label values into broad ranges that are often easier to understand [217].

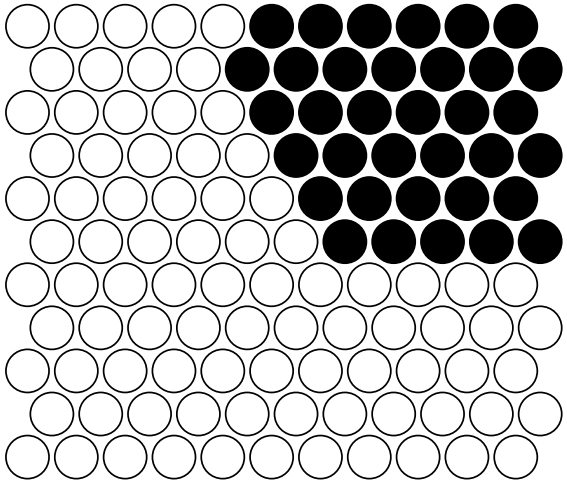
Examples: Unsupervised Weight-Based Cluster Labeling

Figure 6.7 illustrates examples of unsupervised weight-based cluster labeling performed on the SOM of Figure 4.1, which was trained on the Iris data set. The labelings use the same encoding for sub-labels that Figure 6.6 does: sl, sw, pl, and pw correspond to `sepal_length`, `sepal_width`, `petal_length`, and `petal_width`, respectively.

Figure 6.7 (a) shows a grid-based visualization with a global similarity encoding representing the result of a SOM-Ward cluster discovery of emergent clusters. Clearly, two emergent clusters have been discovered: a smaller one in the upper right corner of the map, and a larger cluster occupying the left and lower portions of the grid.

Figure 6.7 (b) presents a table, which summarizes the labeling statistics that are relevant to these examples, for each emergent cluster. For every cluster, the table records the mean and standard deviation of the corresponding weight value for each attribute. The derived value of the standard deviation significance measure of Equation (6.5) is also provided for the attributes within every cluster. The significance values presented in this table were used to produce the labels shown in Figure 6.7 (c) and (d).

Figure 6.7 (c) depicts an unsupervised weight-based cluster labeling of the example map, using standard deviation significance. Components with a significance above -0.0165 are selected (this threshold was established via trial and error), and sub-labels combine attribute names and mean weight values across clusters. The neurons in each emergent cluster receive a uniform labeling, where the upper right cluster is labeled with

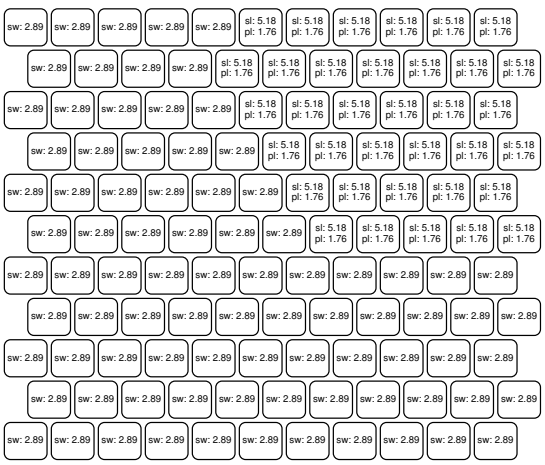


(a)

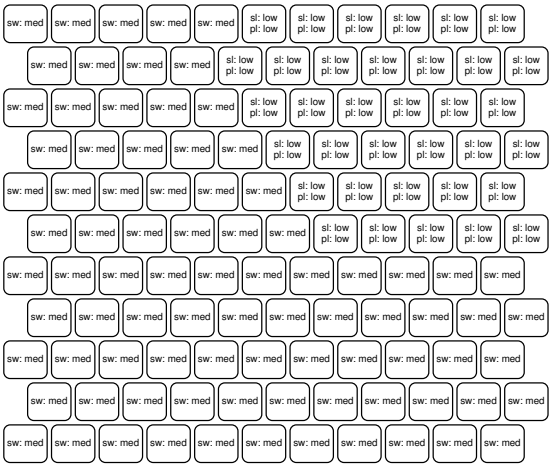
	Cluster 1 (○)		
Attr.	Mean	Std. Dev.	$sig(A_l, S_i)$
sl	0.53	0.0372	-0.0372
sw	0.37	0.0164	-0.0164
pl	0.64	0.0394	-0.0394
pw	0.63	0.041	-0.041

	Cluster 2 (•)		
Attr.	Mean	Std. Dev.	$sig(A_l, S_i)$
sl	0.24	0.0014	-0.0014
sw	0.57	0.0167	-0.0167
pl	0.13	0.015	-0.015
pw	0.11	0.0166	-0.0166

(b)



(c)



(d)

Figure 6.7: The Iris data set SOM of Figure 4.1, with unsupervised weight-based cluster labels using the standard deviation significance of Equation (6.5). The components `sepal_length`, `sepal_width`, `petal_length`, and `petal_width` are respectively denoted sl, sw, pl, and pw: (a) shows a global similarity visualization of two clusters discovered using the SOM-Ward method; (b) shows, for each cluster, weight means and standard deviations, and attribute significance values; (c) shows neurons labeled using a minimum significance threshold of -0.0165, with sub-labels showing mean weight values across clusters; (d) shows the same unsupervised weight-based cluster labeling, with component value thresholds (where low, med, and high respectively denote mean values in the lower, middle, and upper third of each attribute range).

the attributes `sepal_length` and `petal_length`, while the cluster to the left and bottom is only labeled with the `sepal_width` attribute. The uniformly labeled clusters sacrifice detail for a broader, aggregated characterization of the map's larger areas.

Finally, Figure 6.7 (d) shows the same labeling that Figure 6.7 (c) illustrates, with the exception that threshold-based sub-label values are used. Again, as in Figure 6.6, low, med, and high represent the lower, middle, and upper third of each weight's range. As was the case for unsupervised weight-based neuron labeling, it is clear that the threshold-based sub-labels are more interpretable by a human than mean weight values are.

6.3.3.3 A Critical Analysis of Unsupervised Weight-Based Labeling

Both unsupervised weight-based neuron and cluster labeling are capable of producing labels for every neuron on a SOM grid. Unsupervised weight-based neuron labeling, however, produces a much less regular labeling than its cluster-oriented counterpart. Unsupervised weight-based neuron labels typically vary from neuron to neuron, and are therefore more difficult for humans to interpret during EDA investigations.

Unsupervised weight-based cluster labeling should be used when larger emergent map areas require uniform labels. Such uniformly labeled areas are useful when analysts are likely to be overwhelmed by the detail of irregular labels on large maps. This increased uniformity within the labeling is traded off against reduced label detail.

A drawback of unsupervised weight-based cluster labeling is the failure of the approach to successfully deal with heterogeneous clusters, which contain two or more inseparable classes. Section 6.2.2 describes the same problem in the context of supervised example-centric cluster labeling. Unsupervised weight-based neuron labeling does not have this disadvantage, because the aggregating effect of clusters is absent.

Both unsupervised weight-based neuron labeling and unsupervised weight-based cluster labeling require potentially complex significance measure computations. It is possible for these computations to adversely impact the efficiency of the methods. However, the calculations are generally performed over a comparatively small number of neurons, even for fairly large maps, which limits this problem's effect in practice.

In the case of unsupervised weight-based cluster labeling, the emergent cluster discovery step introduces additional computational complexity, which becomes more pro-

nounced as the number of neurons being clustered increases. This effect is obviously not a factor during unsupervised weight-centric neuron labeling. An analysis of unsupervised weight-based cluster labeling's performance characteristics is left to future work.

6.3.4 Unsupervised Example-Based Labeling

Like the unsupervised weight-based labeling methods, unsupervised example-based labeling techniques select statistically informative components to form neuron sub-labels. In contrast to the weight-based methods, however, unsupervised example-based labeling chooses components from data examples that map to a SOM's neuron structure.

The labeling data is chosen from either the training set or a separate labeling set. Unsupervised example-based labeling uses the notion of a BMU, defined in Equation (3.2). Labeling examples must thus be normalized in the same way as the training data.

It should also be noted that the labeling examples need not necessarily consist of exactly the same attributes as the SOM's original training data, although some overlap is necessary. Data analysts should take note that using labeling examples with fewer attributes than the training examples will result in less accurate BMU mappings, thus compromising label accuracy. The use of labeling examples with additional attributes allows labels to be based on attribute data that was not present during training.

As was noted for both the unsupervised weight-based labeling methods, it is possible for multiple sub-labels to introduce an undesirable level of complexity for practical EDA or DM tasks. The optimal number of sub-labels is a subjective parameter. In order to ease label interpretation, data analysts can choose to tune any of the example-based labeling approaches by limiting the number of allowed sub-labels per neuron.

Unsupervised example-based labeling is not based on a SOM's weight structure, and is thus likely to produce a different labeling to unsupervised weight-based labeling. It is therefore possible for an unsupervised example-based labeling, based on a sufficiently large and descriptive labeling example collection, to provide new insights into a SOM's structure that both the unsupervised weight-based labeling methods cannot.

In a similar fashion to the unsupervised weight-based labeling techniques that were discussed in the previous section, this dissertation's research identifies two general classifications for unsupervised example-based neuron labeling approaches: *unsupervised*

example-based neuron labeling, which is described in detail within Section 6.3.4.1, and *unsupervised example-based cluster labeling*, which is discussed in Section 6.3.4.2. Finally, Section 6.3.4.3 critically compares the two approaches to one another.

6.3.4.1 Unsupervised Example-Based Neuron Labeling

Algorithm 6.7 summarizes unsupervised example-based neuron labeling [201, 202]. This method goes through four sequential steps, each of which is separately elaborated upon. Several examples of unsupervised example-based neuron labeling are then presented.

Step 1: Map Labeling Examples to Neurons

Unsupervised example-based neuron labeling techniques require a set of labeling data examples to be associated with each neuron that is a component of the map grid. At the start of the labeling process, an empty mapped data example set, M_{yx}, is associated with each neuron, n_{yx}. Labeling data examples are then mapped to each set, such that each set, M_{yx}, stores all the labeling examples that share neuron n_{yx} as a BMU.

Step 2: Compute Significance Values for Neurons

For each neuron n_{yx} on the map, unsupervised example-based neuron labeling methods use the mapped example set M_{yx} to build sub-labels describing the neuron. In a similar fashion to the unsupervised weight-based labeling techniques, unsupervised example-based neuron labeling methods use a significance measure, which is denoted as $sig(A_l, M_{yx})$. This numeric significance measure is computed for attribute A_l within the mapped set of examples M_{yx}, and is also connected, by extension, with neuron n_{yx}.

A variety of different statistics can feasibly be used for the $sig(A_l, M_{yx})$ significance measure. LabelSOM is one example of an unsupervised example-based neuron labeling method, which was developed by Rauber and Merkl [201, 202]. The LabelSOM approach assumes that an attribute, A_l, is descriptive of a neuron, n_{yx}, if the values of attribute A_l within the set of mapped examples, M_{yx}, are all very similar to the corresponding weight vector component, w_{yxl}, within the neuron. The LabelSOM technique assigns higher significance to the attributes from the mapped example set for neuron n_{yx} that

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence

for all *neurons* n_{yx} *in map* **do**
    Define an empty mapped example set, denoted M_{yx}
    Associate M_{yx} with n_{yx}
end for

for all *labeling example vectors* $\vec{z}_s$ **do**
    Determine the BMU for $\vec{z}_s$, denoted n_{yx}, over all neurons in *map*
    Add labeling example $\vec{z}_s$ to M_{yx}
end for

for all *neurons* n_{yx} *in map* **do**
    for all *attributes* A_l *in* M_{yx} **do**
        Use attribute A_l in M_{yx} to compute $sig(A_l, M_{yx})$
        Associate the new significance value with A_l in M_{yx}
    end for
end for

for all *neurons* n_{yx} *in map* **do**
    for all *sufficiently significant attributes* A_l *in* M_{yx} **do**
        Build a sub-label using the name of A_l and aggregate z_{sl} value in M_{yx}
        Add the new sub-label to the label for n_{yx}
    end for
end for

Algorithm 6.7: Pseudocode of the unsupervised example-based neuron labeling algorithm.

have the lowest cumulative Euclidean quantization error with respect to the weight vector of the neuron, $\vec{w}_{yx}$. The significance measure used by LabelSOM is expressed as:

$$sig(A_l, M_{yx}) = -\left(\sum_{\vec{z}_s \in M_{yx}} \sqrt{(z_{sl} - w_{yxl})^2}\right) \tag{6.7}$$

where z_{sl} is the l^{th} component of weight vector $\vec{z}_s$, and higher significance values indicate attributes with greater significance. The LabelSOM approach's $sig(A_l, M_{yx})$ produces a

maximum significance value of 0.0 when, for the attribute under consideration, all the labeling data examples are perfectly consistent with the weight vector.

The LabelSOM significance measure requires all attributes to be continuous or ordinal, and is inappropriate for nominal values. Therefore, every attribute must have an intrinsic order, because of the measure's reliance on Euclidean quantization errors. This fact highlights an important consideration when selecting a significance measure: the measure must be appropriate for the types of attributes in the labeling data set.

The computation of a value for the numeric $sig(A_l, M_{yx})$ significance measure must be repeated for each attribute that is represented within the set of mapped labeling data examples of every neuron that is a constituent of the map grid of the SOM.

Step 3: Select Informative Attributes for Neurons

An attribute selection mechanism chooses sufficiently significant attributes for sub-labels. The attribute selection methods applicable to unsupervised example-based neuron labeling are the same as those used by unsupervised weight-based neuron labeling.

Step 4: Associate Values with Selected Attributes

Each sub-label typically also incorporates an associated value, which is usually calculated using the neuron's set of mapped examples. For standard continuous attributes, it is appropriate to use the mean or median value calculated over the mapped example set. In the case of attributes with a binary interpretation, it is sensible to choose the attribute value that occurs the most frequently within the mapped example set.

If the weight vectors and labeling example components are normalized, it is advisable to de-normalize the label values to the training data's original attribute ranges, to allow for easier sub-label interpretation. As is the case for the unsupervised weight-based labeling methods, it is also possible to use threshold values in place of raw attribute values, in order to create labels that are more readily interpretable [217].

Examples: Unsupervised Example-Based Neuron Labeling

Figure 6.8 depicts examples of an unsupervised example-based neuron labeling on the Iris data set SOM in Figure 4.1. The figure uses the sub-label encodings that are used

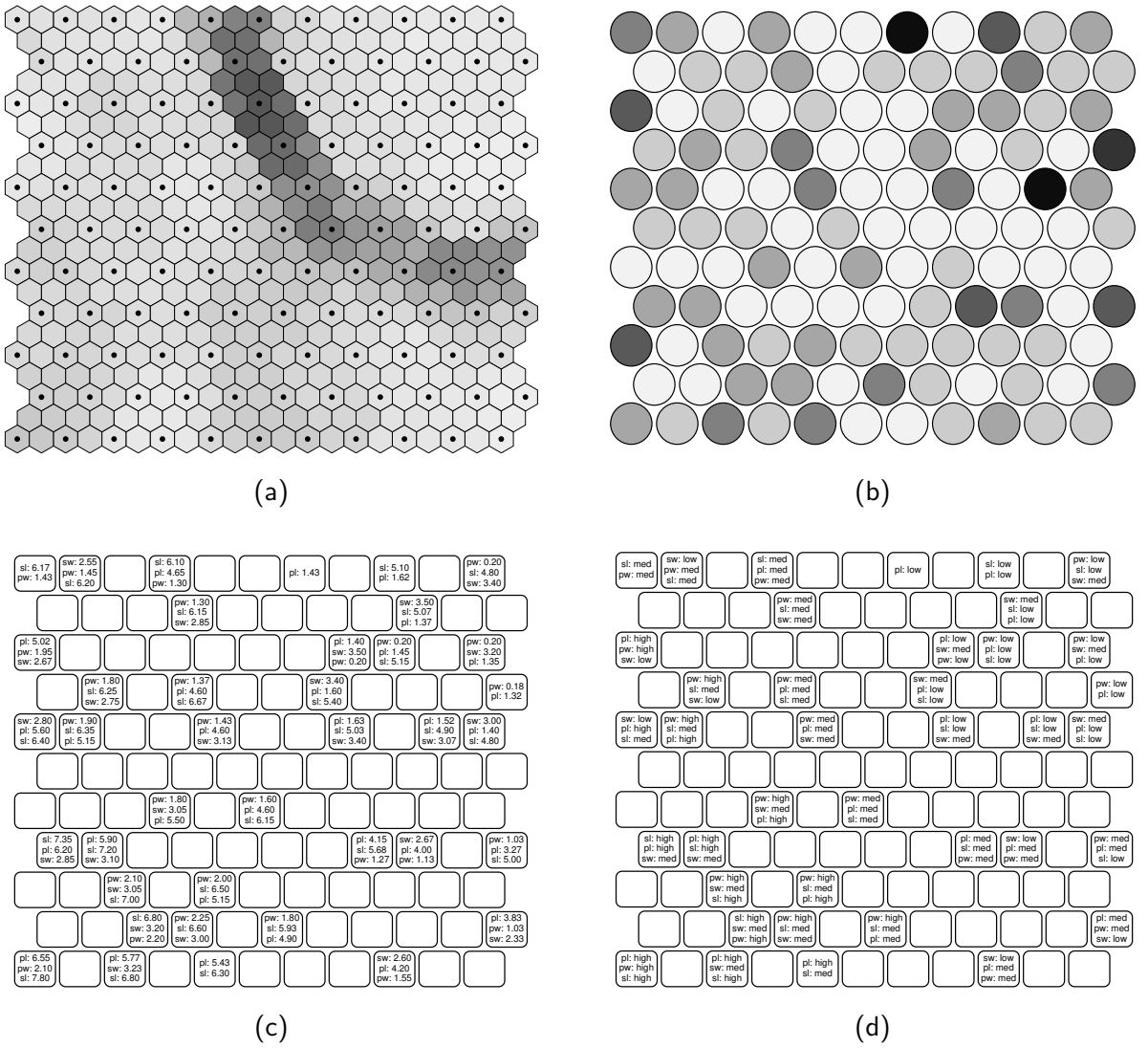


Figure 6.8: The Iris data set SOM of Figure 4.1, with unsupervised example-based neuron labels using the LabelSOM significance of Equation (6.7). The components `sepal_length`, `sepal_width`, `petal_length`, and `petal_width` are respectively denoted sl, sw, pl, and pw: (a) shows the map's U-matrix; (b) shows the training examples' data subset mapping; (c) shows neurons labeled with the full training data set, using a -0.15 significance threshold (where sub-labels require a minimum of two mapped labeling examples, no more than three sub-labels are allowed per neuron, and label values are de-normalized means over mapped labeling example sets); (d) shows the same labeling, using threshold values (where low, med and high respectively denote mean values in the lower, middle and upper third of each attribute range).

in Figures 6.6 and 6.7, where the sub-labels sl, sw, pl, and pw represent the attributes `sepal_length`, `sepal_width`, `petal_length`, and `petal_width`, respectively.

Figure 6.8 (a) shows the U-matrix of the trained map, indicating two emergent clusters. Figure 6.8 (b) illustrates a map grid augmented with a data subset mapping of the SOM's entire training set, where lighter gray shades indicate neurons with fewer BMU matches, and darker shades represent higher numbers of matches. Of interest is the relatively sparse mapping of training set examples over the map's surface, with many neurons receiving either no example mappings at all, or very few mappings.

Figure 6.8 (c) shows an unsupervised example-based neuron labeling, where the full training set is used for label building. The labeling uses a minimum threshold of -0.15, established via trial and error, on the LabelSOM significance measure of Equation (6.7). Sub-label values are means calculated over the labeling examples mapped to the label's neuron. At least two mapped examples are required per sub-label, ensuring labels are based on representative example sets. Very complex labels are avoided by allowing at most three sub-labels per neuron. The labeling method clearly suffers in the presence of sparse labeling sets, as evidenced by the large number of uncharacterized neurons.

Figure 6.8 (d) illustrates the same labeling as Figure 6.8 (c), but uses threshold-based labels in place of mean attribute values. The labeling scheme uses the encodings of low, med and high to respectively indicate the lower, middle and upper third of attribute ranges. It is clear that thresholds offer a more easily interpretable characterization of the neurons than weight aggregates do, at the expense of some sub-label detail.

6.3.4.2 Unsupervised Example-Based Cluster Labeling

Section 6.3.3 discussed how unsupervised weight-based cluster labeling is a simple extension of unsupervised weight-based neuron labeling, which uses emergent clusters to label groups of fairly homogeneous neurons. Similarly, unsupervised example-based cluster labeling [12] uses the basic approach of unsupervised example-based neuron labeling, but adapts the method so that the labeled units are emergent neuron groups. The technique is summarized, in the form of pseudocode, within Algorithm 6.8. The approach executes a five-step procedure, and each step is discussed separately. At the end of the discussion, some examples of unsupervised example-based cluster labeling are presented.

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence
Derive a discrete set of emergent clusters, $\mathcal{L} = \{S_1, S_2, \ldots, S_k\}$, of all $\vec{w}_{yx}$ in the map

for all *clusters* $S_i \in \mathcal{L}$ **do**
    Define an empty mapped example set, denoted N_i
    Associate N_i with S_i
end for

for all *labeling example vectors* $\vec{z}_s$ **do**
    Determine the BMU for $\vec{z}_s$, denoted n_{yx}, over all neurons in *map*
    Determine emergent cluster S_i, such that the BMU of $\vec{z}_s$ is in S_i, and add $\vec{z}_s$ to N_i
end for

for all *clusters* $S_i \in \mathcal{L}$ **do**
    for all *attributes* A_l *in* N_i **do**
        Use attribute A_l in N_i to compute $sig(A_l, N_i)$
        Associate the new significance value with A_l in N_i
    end for
end for

for all *clusters* $S_i \in \mathcal{L}$ **do**
    for all *sufficiently significant attributes* A_l *in* N_i **do**
        Build a sub-label using the name of A_l and aggregate z_{sl} value in N_i
        Add the new sub-label to the label of each $n_{yx} \in S_i$
    end for
end for

Algorithm 6.8: Pseudocode of the unsupervised example-based cluster labeling algorithm.

Step 1: Discover Emergent Clusters

In a similar fashion to the unsupervised weight-based cluster labeling technique, the first step that is performed by unsupervised example-based cluster labeling necessitates the discovery of emergent clusters of map neurons. For this purpose, it is once again appropriate to use any one of the methods that were discussed within Chapter 5.

Step 2: Map Labeling Examples to Clusters

As is the case for unsupervised example-based neuron labeling, unsupervised example-based cluster labeling requires that labeling examples be associated with each unit being labeled. In the case of unsupervised example-based neuron labeling, these units are individual neurons. For unsupervised example-based cluster labeling, labeling examples are mapped to sets associated with each emergent cluster, where set N_i is associated with emergent cluster S_i. It is possible to choose the mapped examples from either the training data set, or a separate labeling data set that was unseen during training.

Step 3: Compute Significance Values for Clusters

A statistical significance measure, denoted as $sig(A_l, N_i)$, is used to calculate a numerical indication of the significance attribute A_l has in the mapped example set N_i. Of course, many alternative significance measures for $sig(A_l, N_i)$ can plausibly be defined.

Azcarraga *et al* [12] propose a significance measure called the difference factor. This measure relies upon $out(N_i)$, which is a set of labeling data examples defined as:

$$out(N_i) = \left(\bigcup_{j=1}^{k} N_j \right) \setminus N_i \tag{6.8}$$

The $out(N_i)$ set therefore encompasses all the labeling examples that are not contained within the mapped example set N_i. In other words, $out(N_i)$ contains the complete set of labeling data examples that do not map into the emergent neuron cluster S_i.

The difference factor aims to assign a high significance to attributes that have a mean value over the labeling examples falling within a cluster, which is significantly higher or lower than the corresponding mean for labeling examples falling outside the cluster. According to the difference factor, the significance of attribute A_l in the set N_i is:

$$sig(A_l, N_i) = \frac{mean(A_l, N_i) - mean\big(A_l, out(N_i)\big)}{mean\big(A_l, out(N_i)\big)} \tag{6.9}$$

where $mean(A_l, N_i)$ is the mean value of A_l over the mapped example set N_i, and $mean\big(A_l, out(N_i)\big)$ is the mean value of A_l over the $out(N_i)$ set. The difference factor produces a significance value of 0.0 when the attribute means are equal to one another.

The attribute thus shows no difference between N_i and $out(N_i)$, and is considered to be insignificant. Highly significant attributes have large positive or negative values.

Significance measures must again be appropriate for the types of attributes in the labeling set, whether continuous, ordinal, or nominal. It is possible to adapt an unsupervised weight-based cluster labeling $sig(A_l, S_i)$ measure for example-based cluster labeling by computing the measure over N_i, rather than S_i. For example, the measures of Equations (6.3) to (6.5), as well as the K-S metric, are adaptable in this way. However, it should be noted that these measures are only applicable to continuous attributes within labeling data sets. Pearson's chi-squared statistic [180] is an alternative to the K-S statistic, which is appropriate for nominal attributes in labeling sets [188].

Step 4: Select Informative Attributes for Clusters

The attributes that produce values indicating higher significance are used as the basis of sub-labels for emergent map clusters. By extension, the neuron constituents that make up an emergent cluster receive the same sub-labels as the cluster itself does.

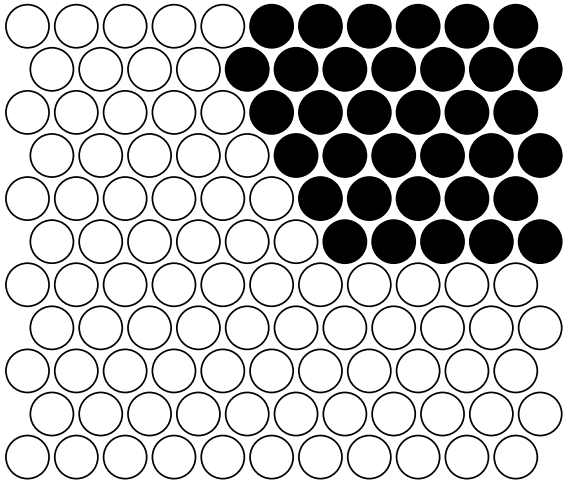
An appropriate attribute selection mechanism builds sub-labels, using sufficiently significant attributes that are chosen from each cluster's mapped example set. The attribute selection mechanisms for unsupervised example-based cluster labeling are the same as those that are available for unsupervised weight-based cluster labeling. All the neuron constituents of a cluster receive the same sub-labels as the cluster.

Step 5: Associate Values with Selected Attributes

In a similar fashion to unsupervised example-based cluster labeling, each selected attribute usually has an associated value. This value is typically the mean, median or most common value for the attribute in question, calculated over the set of mapped examples for the sub-label's cluster. Sub-label values should be de-normalized to the original attribute ranges, and threshold-based values often improve interpretability [217].

Examples: Unsupervised Example-Based Cluster Labeling

Figure 6.9 illustrates examples of unsupervised example-based cluster labeling on the Iris data set SOM from Figure 4.1, using the difference factor significance measure of

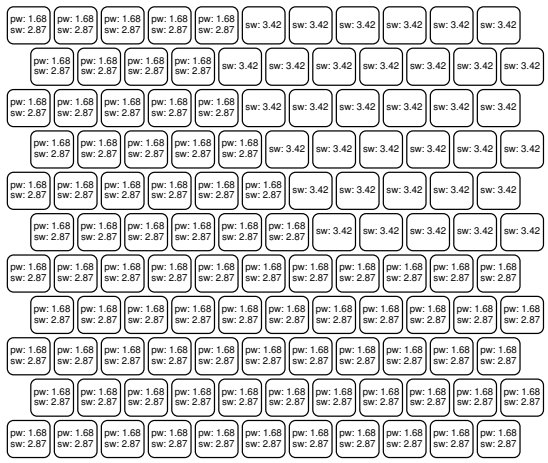


(a)

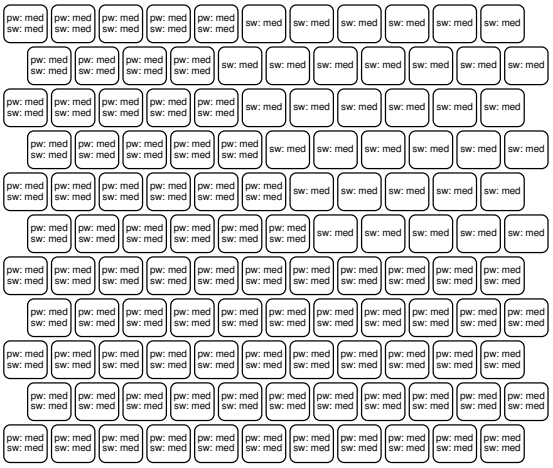
	Cluster 1 (○)		
Attr.	Mean N_i	Mean $out(N_i)$	$sig(A_l, N_i)$
sl	0.54	0.20	1.72
sw	0.36	0.59	-0.39
pl	0.66	0.08	7.30
pw	0.66	0.06	9.81

	Cluster 2 (•)		
Attr.	Mean N_i	Mean $out(N_i)$	$sig(A_l, N_i)$
sl	0.20	0.54	-0.63
sw	0.59	0.36	0.63
pl	0.08	0.66	-0.88
pw	0.06	0.66	-0.91

(b)



(c)



(d)

Figure 6.9: The Iris data set SOM of Figure 4.1, with unsupervised example-based cluster labels using the difference factor significance of Equation (6.9). The components `sepal_length`, `sepal_width`, `petal_length`, and `petal_width` are respectively denoted sl, sw, pl, and pw: (a) shows a visualization of two SOM-Ward discovered clusters; (b) shows, per cluster, the mean A_l value over N_i and $out(N_i)$, and attribute significances; (c) shows clusters labeled using attributes with significances differing more than one standard deviation from the cluster's mean significance, with sub-labels showing mean attribute values across clusters; (d) shows the same unsupervised labeling, with component value thresholds (where low, med, and high respectively denote mean values in the lower, middle, and upper third of each attribute range).

Equation (6.9). These examples use the attribute name encodings found in Figures 6.6, 6.7, and 6.8. That is to say, the sl, sw, pl and pw sub-labels represent the `sepal_length`, `sepal_width`, `petal_length`, and `petal_width` attributes, respectively.

Figure 6.9 (a) illustrates a grid-based map visualization that is augmented with a global similarity encoding. The visual encoding shows two emergent clusters of neurons, which were discovered by means of the SOM-Ward clustering algorithm.

Figure 6.9 (b) summarizes the statistical values that are relevant to the calculation of the difference factor significance values for each of the attributes in both discovered clusters. Specifically, the values of $mean\big(A_l, N_i\big)$, $mean\big(A_l, out(N_i)\big)$, and the final $sig(A_l, N_i)$ measure are shown for all attributes within each emergent cluster.

Figure 6.9 (c) shows an unsupervised example-based cluster labeling, based on the full training data set, and the significance values shown in Figure 6.9 (b). Cluster 1 has a mean significance value of 4.61, with a standard deviation of 4.11. The mean significance for cluster 2 is -0.45, with a standard deviation of 0.63. Selected sub-label attributes have significances that exceeded one positive or negative standard deviation from the relevant cluster's mean significance value. Sub-label values are de-normalized means of the attribute values for each selected attribute, calculated over the appropriate emergent cluster. Clusters are labeled uniformly, and no neurons have been left unlabeled.

Finally, Figure 6.9 (d) depicts the same unsupervised labeling as is shown in Figure 6.9 (c), but uses threshold-based sub-label values, rather than raw attribute value means. The sub-label values low, med, and high respectively represent the upper, middle, and lower third of the relevant attribute value's range. Threshold-based sub-labels are again easier to interpret than attribute value means, while sacrificing finer detail.

6.3.4.3 A Critical Analysis of Unsupervised Example-Based Labeling

Unsupervised example-based approaches and their weight-based counterparts are likely to produce fairly different labelings. Unsupervised weight-based labels are based on weights that approximate the training data. The sub-labels and values chosen by these methods thus also only approximately represent the training data. In contrast, example-based labels are derived from actual data examples. This implies that such labels are more direct representations of the data set attributes, rather than approximations.

In a similar fashion to supervised example-centric neuron labeling, both unsupervised example-based labeling methods are sensitive to labeling data quality. A sparse data set will produce smaller and less representative mapped example sets. Such sets will result in poor quality significance values, and therefore badly chosen sub-labels. Smaller sets of mapped examples also tend to result in less accurate sub-label values.

A labeling data set with many missing values also produces poor example mappings, which leads to poor sub-labels. It is thus prudent for unsupervised example-based labeling methods to require that attributes (or at least attributes chosen as sub-labels) have an analyst-specified minimum number of non-missing values among the mapped examples. The missing value threshold depends on the average number of mapped labeling examples per neuron or cluster, because large mapped example sets can accommodate more missing values before significance value reliability is compromised.

Unsupervised example-based cluster labeling is, however, less susceptible to the problems associated with sparse data sets and missing values than unsupervised example-based neuron labeling is. This is because of the aggregating effect of the emergent clusters, which group larger numbers of labeling data examples together than individual neurons do. This grouping produces mapped labeling example sets that are more representative, and within which a smaller proportion of missing values fall. In any event, this research very strongly recommends that validity analysis be performed by a human expert on any unsupervised example-based neuron or cluster labeling.

A further drawback associated with unsupervised example-based neuron labeling is that non-BMU neurons have no mapped examples, and thus receive no labels. Unsupervised example-based neuron labeling is therefore prone to partially labeling map grids in the presence of sparse data. In contrast, both the unsupervised weight-based labeling techniques ensure that sub-labels are applied to all neurons on the map.

The phenomenon of unlabeled neurons is also less severe for unsupervised example-based cluster labeling, because clusters containing no mapped examples are much less likely. This is because each emergent cluster's sub-labels are applied to all the cluster constituents, including neurons that are never BMUs for any labeling examples.

The above discussion suggests that the unsupervised example-based labeling methods should be preferred above unsupervised weight-based labeling when a comprehensive

enough data set is available. However, in the presence of a sparse or incomplete data set, the results of unsupervised weight-based labeling will be more trustworthy. Unsupervised example-based neuron labeling is also less appropriate for EDA than the other methods, because fragmented labelings are more difficult for humans to analyze.

A disadvantage shared by both unsupervised example-based and weight-based neuron labeling is that sub-labels tend to vary from neuron to neuron, producing irregular labelings. Such labelings are usually more difficult for human analysts to interpret. Both unsupervised example-based and weight-based cluster labeling produce large areas of uniform sub-labels, which are typically easier for human experts to analyze.

Unsupervised example-based cluster labeling also suffers from the problem of uniformly labeling heterogeneous clusters containing more than one class, in the same way that supervised example-centric cluster labeling and unsupervised weight-based cluster labeling do. Unsupervised example-based neuron labeling, of course, does not suffer from this problem, because the method does not rely on discovered emergent clusters.

Another drawback of both unsupervised example-based neuron and cluster labeling is the additional performance penalty incurred while mapping labeling examples to the map's neurons. The mapped examples can also negatively impact the memory resources used when sizable maps are labeled using a large number of examples, because of the memory resources required to store the necessary sets of mapped data examples.

As was the case for the unsupervised weight-based labeling techniques, both unsupervised example-based labeling approaches require relatively complicated significance measure calculations. It is possible for these computations to negatively affect the efficiency of the approaches, especially when very large sets of labeling examples are used. Large labeling example sets result in large sets of mapped examples, which will take more time to process into significance values. Large maps, consisting of many neurons, will exacerbate any problems related to slow significance value computation.

Unsupervised example-based cluster labeling's cluster discovery step increases computational cost, compared to both unsupervised weight-based neuron labeling and unsupervised example-based neuron labeling. The exact complexity introduced by the clustering phase depends on the clustering algorithm that is used. A detailed analysis of the comparative performance characteristics of the approaches is left to future work.

6.4 Applying Neuron Labeling to SOMs

Each of the supervised and unsupervised neuron labeling techniques, respectively discussed in Sections 6.2 and 6.3, can be applied to purely unsupervised, supervised, or semi-supervised SOMs. The supervision levels for SOM training are described in Section 3.3.4. Section 6.4.1 discusses the application of labeling methods to purely unsupervised SOMs, while Section 6.4.2 describes considerations for supervised SOM labeling, and Section 6.4.3 investigates labeling method application for semi-supervised SOMs.

6.4.1 Neuron Labeling for Purely Unsupervised SOMs

As has been noted previously, the majority of self-organizing maps applied in a practical setting are purely unsupervised in nature. As a consequence, the majority of neuron labeling approaches have been developed with purely unsupervised maps in mind. All of the neuron labeling techniques that have been discussed within this chapter can therefore be applied directly to entirely unsupervised SOMs, with no additional modifications.

6.4.2 Neuron Labeling for Supervised SOMs

A supervised SOM is simply an unsupervised SOM that incorporates classification attributes during training. As a result, all the labeling methods that are applicable for unsupervised SOMs are also relevant in the context of a supervised map. However, it is important to take note of the fact that any labeling methods that are employed will be biased by the classification information that is included during training.

Additionally, the trained map weights associated with the data set's classification attributes can be directly used for unsupervised labeling. A simple approach is to label each neuron with the attribute names and weight values associated with the data set's classification attributes. Unfortunately, such labels are fuzzy in nature, because the label values are continuous and provide no hard membership to one class or another.

Because fuzzy labels are often not desirable in a practical EDA or DM context, thresholds can be used to discretize the classification attribute weights. The selection of specific thresholds in such cases is dependent on the nature of the classification's encoding. Binary classification attributes normalized to a $[0.0, 1.0]$ range should use a threshold of

0.5, with weights below the threshold generating a label according to one classification, and weights greater than and equal to the threshold generating labels for the opposite classification. A non-binary nominal classification attribute requires multiple thresholds, equal to one fewer than the number of values allowed by the attribute.

It is also possible to use classification attribute weights to label clusters of neurons. One approach is to discretize the classification attribute weight values for each neuron in a cluster, and label the cluster with the discrete label that most commonly occurs within the cluster. Another alternative is to calculate the mean of each classification attribute's weight values across a cluster, and label the entire cluster using these means. Such mean values can also be discretized using threshold values. No experimental investigation into such techniques has yet been attempted, and is left to future research.

6.4.3 Neuron Labeling for Semi-Supervised SOMs

Semi-supervised SOMs are simply standard unsupervised SOMs that model the distribution of classification attributes over the map surface, but do not allow these attributes to bias the training process. Consequently, all of the labeling methods applicable for purely unsupervised SOMs are also valid for semi-supervised maps.

In addition, due to the inclusion of classification attributes, the unsupervised labeling methods for supervised SOMs are all also appropriate for semi-supervised SOM grids. An advantage that labeling specific to semi-supervised SOMs has over the same approaches on supervised maps is that, while labels are derived from the classification attributes, the classification information will not bias the map's data model in any way.

6.5 Neuron Labeling and High-Dimensional Data

As Section 3.6 notes, data sets with very high dimensionality have the potential to negatively affect the accuracy of a SOM's data model because of the problems that are introduced by the "curse of dimensionality" [16, 19, 216]. Neuron labels are always in some way derived from the map structure embodying the data model of a SOM. As a result, high-dimensional data sets are also likely to negatively affect the accuracy of any characterization provided by the labels of the neurons making up such inaccurate maps.

Due to the same problem with distances in high dimensions, any labeling approaches that are based on distance measures will themselves become less accurate as data set dimensionality increases. This problem affects all the labeling algorithms that require mappings of labeling data examples to neurons, or mappings of weight vectors to labeling examples. Furthermore, in the case of the unsupervised weight- and example-based labeling algorithms, this adverse effect extends to any significance measures or attribute selection schemes that use distances between weight vectors, distances between labeling data examples, or distances between weight vectors and labeling examples.

A concern unique to the unsupervised weight- and example-based neuron labeling algorithms is the number of sub-labels that can feasibly be used, which is limited by the ability of the human analyst to interpret the chosen number of attributes. This number is dependent on both the complexity of the data set and the expertise of the analyst in question, but is very unlikely to exceed ten in most practical cases. This limit is particularly troublesome for very high-dimensional data sets with few redundant attributes. In such cases the number of sub-labels describing each neuron becomes a tiny fraction of the available attributes, thus limiting the expressivity of the labeling. The labels also lose the ability to distinguish between neurons that have the same highly significant attributes, but differ in terms of less significant attributes. In such cases the same set of highly significant attributes will be selected for all neurons in question, reducing the ability of the labeling to differentiate between these map elements.

The above-mentioned potential pitfalls associated with SOM neuron labeling in the presence of high-dimensional data have yet to be analyzed. Real-world data sets are of particular interest for such investigations, and this work is left to future research.

6.6 Summary

This chapter overviewed the available map neuron labeling techniques. Section 6.1 presented an overview of the objectives that underlie neuron labeling, and the perspective on neuron labeling that is taken by this dissertation. Two categories of labeling techniques were described, namely supervised labeling techniques and unsupervised labeling methods. Section 6.2 provided further detail on specific supervised labeling techniques, while

Section 6.3 discussed the important factors related to unsupervised labeling. Section 6.4 dealt with the application of neuron labeling methods to self-organizing maps that are purely unsupervised, supervised, or semi-supervised. Finally, Section 6.5 discussed the limitations of neuron labeling in the face of very high-dimensional data sets.

The next chapter provides a detailed overview of the available SOM-based data mining algorithms. The chapter also introduces the novel HybridSOM rule extraction framework, which is the major contribution made by this dissertation's research work.

Chapter 7

SOM-Based Data Mining

The previous chapter described the various techniques available for addressing the task of labeling the neurons that make up a trained SOM. This chapter provides an overview of the automatic DM approaches that can be applied to SOMs. The overall distinction between the realms of EDA and DM was investigated in Chapter 2. The SOM-based DM methods in this chapter should be contrasted with the SOM-based EDA and visualization techniques described in Chapter 4. The emergent cluster discovery techniques, which were discussed in Chapter 6, are also referred to in parts of this chapter.

This chapter is arranged as follows: Section 7.1 outlines the underpinnings of SOM-based DM, and differentiates between supervised and unsupervised SOM-based DM. Section 7.2 outlines a supervised DM approach based on map region boundary detection, while Section 7.3 covers SIG*, the most widely cited unsupervised SOM-based DM algorithm. Section 7.4 describes this work's main contribution, an unsupervised rule extraction framework called HybridSOM. Section 7.5 covers miscellaneous SOM-based methods for DM, which are not this work's main focus. Section 7.6 justifies the viability of SOM-based DM in a practical setting. Finally, Section 7.7 summarizes the chapter.

7.1 The Philosophy of SOM-Based Data Mining

The premise of SOM-based data mining is similar to the one upon which SOM-based visualization is founded, which is described in Section 4.2: because the map structure of

a trained SOM approximately models characteristics of the data set used during training, a visualization of the map (or map-related features) reveals aspects of this training data. Similarly, SOM-based DM approaches are based on a trained map that is assumed to form a weight vector model of the underlying training data. SOM-based DM methods build a model either directly from the map's weight vectors, or from training data that is related to the map's weight structure. The model is typically a rule set.

This research identifies two categories of SOM-based DM techniques, defined by the reliance of the approach on knowledge of the classes present in the training data:

- *Supervised SOM-based data mining* is based on algorithms that require *a priori* knowledge of the classes that are present within the training data set. Clearly, these techniques cannot be used if no class information is present. It should also be noted that classification information is often unavailable in a practical data mining setting. In addition, the reliance of supervised SOM-based DM methods on classification information often biases DM approaches in this category.

- *Unsupervised SOM-based data mining* requires no prior knowledge of the training data classes. Techniques in this category are applicable both in situations where classification attributes are present, and when no such attributes are available. These methods are thus more general than their supervised counterparts. Another advantage of unsupervised SOM-based DM is that knowledge extraction is not biased by classification information, even when this information is present.

If it is assumed that the training data subset is representative of the general data landscape being modeled, the overall objective of a SOM-based DM technique is to build a model that represents the general characteristics of data examples that were not presented during SOM training (i.e., the model generalizes to unseen examples).

This dissertation investigates three data mining algorithms for SOMs. The only supervised SOM-based data mining method identified during this research work's literature survey is the *boundary-based rule extraction algorithm*, which is described in Section 7.2. Two unsupervised SOM-based data mining methodologies also exist: the widely-cited *SIG* algorithm* is covered in Section 7.3, and the novel *HybridSOM framework* is proposed in Section 7.4. Figure 7.1 illustrates the categorization of these methods.

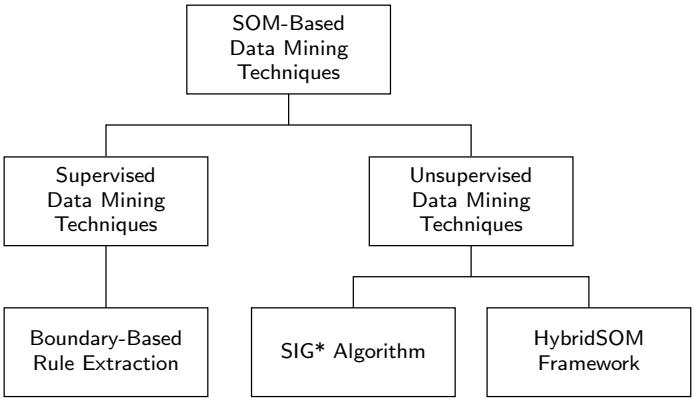


Figure 7.1: A general overview of the categorization of the supervised and unsupervised SOM-based data mining techniques that are discussed within this dissertation.

The main focus of this research work is on unsupervised SOM-based DM algorithms, because these techniques are applicable to a wider variety of data sets, and better maintain the overall philosophy of unsupervised training that SOMs promote. As such, the remainder of this discussion focuses more on the SIG* and HybridSOM methods.

7.2 Boundary-Based Rule Extraction

The boundary-based rule extraction algorithm is a supervised SOM-based DM approach that was proposed in 2006 by Malone *et al* [162]. Section 7.2.1 gives an overview of the approach, while a more detailed discussion of the algorithm's mechanics is undertaken in Section 7.2.2. Lastly, a critique of the method is provided in Section 7.2.3.

7.2.1 An Overview of the Approach

Boundary-based rule extraction focuses on the algorithmic detection of boundaries within a SOM's U-matrix and various component plane visualizations. Salient attributes are detected using correlations between the U-matrix and component plane boundaries. The boundaries within the selected component planes are then used to define attribute value conditions for rules that differentiate between the areas separated by the boundaries. The technique is outlined visually in Figure 7.2, and by means of pseudocode in Algorithm 7.1.

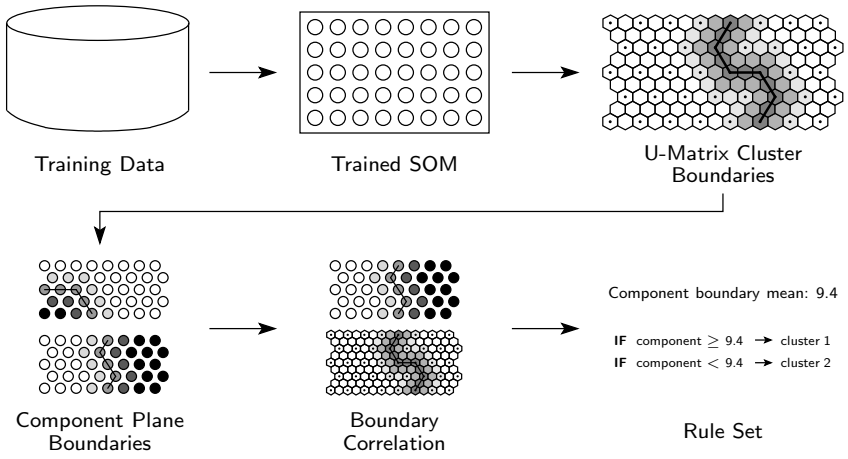


Figure 7.2: The overall procedure for the boundary-based rule extraction algorithm.

7.2.2 The Rule Extraction Procedure

The rule extraction process is separated into four steps. The first step detects cluster boundaries from the U-matrix of a trained map. The second step performs a similar boundary detection process on each of the SOM's component planes. Thirdly, each individual component plane boundary is compared to the U-matrix cluster boundaries, with the aim of selecting the component plane boundaries that closely match any one of the U-matrix boundaries. Finally, each component plane's boundaries are used to define a new rule's antecedent conditions. Each step is described in greater detail, below.

Step 1: Compute Candidate Cluster Boundaries

The first step of the boundary-based rule extraction process requires a SOM to be trained on an I-dimensional training data set. A U-matrix of the fully trained SOM must then be generated, as described in Section 4.3.1.1, after which a set of candidate boundaries on the U-matrix must be found. Each candidate boundary is a set of links on the U-matrix, joining adjacent neurons that are on the borders between emergent map clusters. Every neuron on the map is considered as a potential candidate boundary member.

A quantitative statistical measure is required to judge each neuron's potential for forming part of a candidate U-matrix boundary. While a variety of statistical measures could conceivably be used for this purpose, the original work of Malone *et al* [162]

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence

Generate a U-matrix for *map*, denoted *umat*, as described in Section 4.3.1.1

while *umat has insufficient boundaries to separate one area per class* $C_m \in \mathcal{C}$ **do**
    Generate a new candidate boundary on the U-matrix of *map*
end while

for all *attributes* $A_l \in \mathcal{D}_T$ **do**

    Generate a component plane for A_l, denoted $plane_l$, as described in Section 4.3.2.1

    while $plane_l$ *has insufficient boundaries to separate one area per class* $C_m \in \mathcal{C}$ **do**
        Generate a new candidate boundary for the component plane of A_l
    end while

end for

Define an empty rule set, denoted *rule_set*

for all *candidate boundaries,* $umat\_bound_u$*, on the U-matrix* **do**

    Define $rule_{u1}$, with a consequent that predicts the first area separated by $umat\_bound_u$
    Define $rule_{u2}$, with a consequent that predicts the second area separated by $umat\_bound_u$

    for all *component plane visualizations,* $plane_l$*, of map* **do**
        for all *candidate boundaries,* $plane\_bound_{l\hat{m}}$*, on* $plane_l$ **do**

            Compute the similarity between $plane\_bound_{l\hat{m}}$ and $umat\_bound_u$

            if *computed similarity is within an error threshold* **then**
                Compute the mean value along $plane\_bound_{l\hat{m}}$, denoted $plane\_mean_{l\hat{m}}$
                Add an antecedent condition, based on $plane\_mean_{l\hat{m}}$, to $rule_{u1}$
            end if

        end for
    end for

    Add $rule_{u1}$ to *rule_set*, merging $rule_{u1}$ with rules already in *rule_set* if necessary
    Add $rule_{u2}$ to *rule_set*, merging $rule_{u2}$ with rules already in *rule_set* if necessary

end for

Algorithm 7.1: Pseudocode of the boundary-based rule extraction algorithm.

suggests using a statistic called the boundary difference value (BDV). For neuron n_{yx}, the BDV is calculated relative to any two neurons adjacent to n_{yx}. The neuron, n_{yx}, and its two neighbors form a candidate boundary segment. The BDV is defined as:

$$BDV(cand_{yx}) = \frac{mean(cand_{yx}) - mean(cand'_{yx})}{range(cand'_{yx})} \tag{7.1}$$

where $cand_{yx}$ is a set containing two candidate neurons that are neighbors of n_{yx}, the set $cand'_{yx}$ contains the neurons that are neighbors of n_{yx} and are not in $cand_{yx}$, $mean(cand_{yx})$ and $mean(cand'_{yx})$ denote the mean distances between $\vec{w}_{yx}$ and the neuron weight vectors in $cand_{yx}$ and $cand'_{yx}$, respectively, and $range(cand'_{yx})$ defines the range of the distances between $\vec{w}_{yx}$ and the weight vectors within the $cand'_{yx}$ set (calculated as the difference between the largest distance and the smallest distance).

For map neuron n_{yx}, a BDV statistic is calculated for every pair of adjacent neurons, thus considering every possible candidate boundary segment through n_{yx}. Once the BDV for every candidate boundary segment through neuron n_{yx} has been computed, the maximum of these values is selected as the final BDV for n_{yx}. This process is repeated across the grid, producing an optimal BDV statistic value for every neuron.

Finally, the neuron with the highest BDV on the map is identified, in order to serve as a starting point for a new candidate boundary. Subsequent neurons with the highest BDV values neighboring the partially-formed boundary are added to the boundary. The process is repeated until a complete candidate boundary has been formed. The boundary-forming procedure is repeated on the remaining non-boundary neurons of the map's U-matrix. Boundaries are formed until the number of delimited areas is equivalent to the number of classes that are present in the map's original training data set.

Step 2: Compute Candidate Component Boundaries

The second step of the rule extraction process finds candidate boundaries for each of the map's components. This procedure requires a component plane to be generated, as Section 4.3.2.1 discusses, for each attribute making up the SOM's data model.

Once a component plane has been generated, a set of candidate boundaries must be computed to link neurons separating areas of similar weights. The detection of compo-

nent plane boundaries is similar to the previously described boundary detection process on the map's U-matrix, and uses a measure that is similar to the BDV of Equation (7.1). The BDV statistic used for component plane boundaries, however, uses the distances between adjacent component plane values, rather than U-matrix distances.

Other than the minor adaptation of the BDV statistic, the boundary detection procedure is the same as the detection performed on the trained map's U-matrix. For a single component plane, sufficient boundaries must be generated to separate a number of areas that is equivalent to the number of classes in the map's training data. Finally, the entire detection process is repeated for each of the component planes in turn.

Step 3: Find Matching Boundaries

Following the mapping of candidate boundaries on the SOM's U-matrix and component planes, the rule extraction process attempts to detect the most descriptive attributes that characterize emergent clusters of neurons on the map's U-matrix. The basic premise underlying this step of the procedure is as follows: homogeneous areas on component planes, which correlate (or partially correlate) to emergent clusters on the U-matrix, can be assumed to have an influence on the formation of these emergent clusters.

The detection of salient attributes is achieved by means of a search that identifies close matches between the U-matrix and component plane boundaries. The search compares each U-matrix candidate boundary to every component plane boundary. A user-specified similarity threshold is used to determine whether the compared boundaries are close enough to one another in map space. Attribute boundaries that satisfy the threshold requirement are selected to form rule conditions during the algorithm's final step.

Step 4: Construct Rules

The final step of the boundary-based rule extraction algorithm is the construction of descriptive rules. These rules are based on the U-matrix and component plane boundaries that have been defined as similar during the previous step of the algorithm.

As a first step, the candidate U-matrix boundaries are considered one by one. Attribute value conditions are defined for each of the component plane boundaries that

have been matched to the U-matrix boundary under consideration. The mean of the attribute values along each selected component plane boundary is also computed.

Two conditions are typically derived from a computed component boundary mean, each describing one of the emergent neuron clusters separated by the matching U-matrix boundary. The two generated attribute value conditions are of the general form:

$$plane\_attribute_l \geq plane\_mean_{l\hat{m}}$$
$$plane\_attribute_l < plane\_mean_{l\hat{m}}$$

where $plane\_attribute_l$ is the attribute name for component plane l, and $plane\_mean_{l\hat{m}}$ is the mean attribute value over candidate boundary $\hat{m}$ on component plane l.

Finally, the generated conditions are added to the antecedent of a rule that predicts one of the areas separated by the U-matrix boundary that is currently being considered. The consequent of the rule refers either to the appropriate emergent cluster, or a descriptive label that has been applied to the emergent cluster. All the conditions that refer to the same emergent cluster should also be aggregated into a single rule.

7.2.3 A Critique of the Approach

To the author's knowledge, the boundary-based algorithm has not been analyzed in detail, or compared to other methods. Only initial reasoning on the approach's strong and weak points is thus feasible, and further analysis is left to future work.

The method's comparison of SOM visualizations is similar to the EDA map analysis discussed in the previous chapter. Therefore, the boundary-based approach is relatively intuitive. Because of the emergent characteristics of trained SOMs, it is possible that the technique could offer advantages over traditional DM methods. However, further analysis is required to confirm the existence of any such advantages.

The first drawback of the method is simply that the algorithm is supervised, meaning that the approach cannot mine data sets that have no associated classification data. The algorithm's supervised nature also biases the produced rule set results.

The approach's second drawback lies in the relative complexity of the algorithm's implementation, as is evident from the previous description of the technique. This complexity may hinder the correct or timeous implementation of the algorithm.

Thirdly, the procedure is likely to be time-consuming, because multiple boundaries must be detected on several map visualizations. Algorithm efficiency is likely to suffer greatly in the presence of very high-dimensional training data, because many component planes will be generated, and each plane must also be searched for boundaries.

Finally, the technique depends on the presence of clear U-matrix and component plane boundaries. It is thus likely that poor or erratic results will be produced if boundaries are not present, fragmented, or do not fully separate emergent neuron clusters.

7.3 The SIG* Algorithm

The SIG* algorithm was developed in 1991 by Alfred Ultsch [238]. The algorithm became part of a KDD system called REGINA [239], and applied to problem domains such as medical diagnosis, environmental evaluation and industrial process analysis.

Section 7.3.1 overviews the algorithm. Section 7.3.2 outlines the algorithm's first main sub-procedure, which builds characterizing rules. Section 7.3.3 covers the second sub-procedure of the algorithm. Section 7.3.4 explains how SIG* rules can be converted into standard production rules. Finally, Section 7.3.5 critiques the SIG* algorithm.

7.3.1 An Overview of the Approach

The SIG* algorithm is a relatively complex data mining method, consisting of a pipelined sequence of several steps. The sequence of steps that the algorithm executes are summarized in diagrammatic form in Figure 7.3, and as pseudocode in Algorithm 7.2.

The algorithm's first phase begins by using an I-attribute data set to train a SOM until convergence. Once training is completed, discrete emergent neuron clusters must be found. While the original algorithm's description uses exploratory clustering aided by a U-matrix, any of the discrete cluster discovery methods described in Chapter 5 are appropriate for finding k emergent clusters, denoted as $\mathcal{L} = \{S_1, S_2, \ldots, S_k\}$.

Each discovered emergent neuron cluster, S_i, must then receive a label, L_i, using any of the cluster-based labeling techniques from Chapter 6. Either supervised labeling techniques (i.e., example-centric cluster labeling) or unsupervised labeling methods (i.e., exploratory cluster labeling, unique cluster labeling, unsupervised weight-based

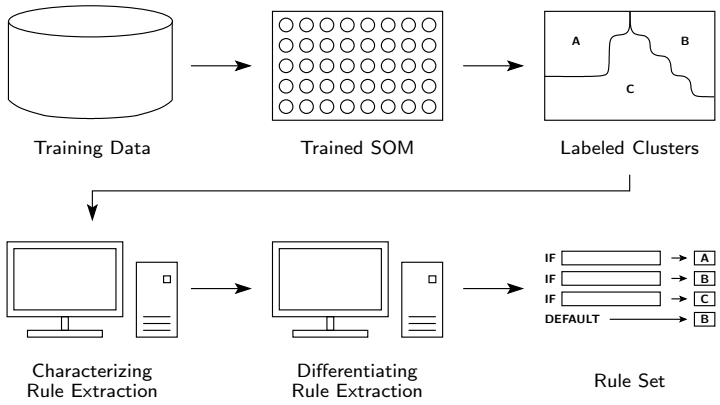


Figure 7.3: The overall component interactions within the SIG* algorithm.

cluster labeling, or unsupervised example-based cluster labeling) are applicable. Additionally, in a situation where the SOM's training is either supervised or semi-supervised, it is also possible to use the cluster labeling methods that were described in Section 6.4. Because the SIG* algorithm extracts rules from emergent clusters, it is impossible to use any of the labeling methods that attach characterizations to individual neurons.

The first phase of the SIG* algorithm reaches a conclusion with a mapping of all the SOM's training data examples to the discovered emergent neuron clusters. This is done by determining a BMU for each training example. Each training example is then mapped to the emergent cluster within which the BMU of the example falls.

Once the first algorithmic phase has been completed, the rule extraction procedure commences. The primary mechanism at the core of the SIG* rule extraction process is embodied in the remaining two general phases of the algorithm: the *characterizing rule construction procedure* and the *differentiating condition construction procedure*.

The characterizing rule construction sub-procedure builds a *characterizing rule* for each unique training example set and the set's corresponding discovered emergent neuron cluster. The general objective of each characterizing rule is to summarize the essential characteristics of the training example set that the rule describes. The characterizing rule building process is discussed in greater detail within Section 7.3.2.

The differentiating condition construction sub-procedure conditionally refines certain characterizing rules, by adding additional *differentiating conditions* to pairs of characterizing rules that are judged to have overlapping descriptions. Two characterizing rules

Create and initialize a SOM, denoted *map*, consisting of $Y \times X$ neurons
Train *map* on an I-attribute training set, denoted $\mathcal{D}_T$, until convergence

Phase 1: Map training examples to emergent neuron clusters
Derive a discrete set of emergent clusters, $\mathcal{L} = \{S_1, S_2, \ldots, S_k\}$, of all $\vec{w}_{yx}$ in the map
Apply a label, L_i, to each cluster, $S_i \in \mathcal{L}$, using a cluster-based neuron labeling method

for all *clusters* $S_i \in \mathcal{L}$ **do**
    Define an empty mapped example set, denoted N_i
    Associate N_i with S_i
end for

for all *training example vectors* $\vec{z}_s$ **do**
    Determine the BMU for $\vec{z}_s$, denoted n_{yx}, over all neurons in *map*
    Determine emergent cluster S_i, such that the BMU of $\vec{z}_s$ is in S_i, and add $\vec{z}_s$ to N_i
end for

Phase 2: Define a characterizing rule for each emergent cluster
Define an empty rule set, denoted as *rule_set*, which contains no rules
call function AddCharacterizingRules($\mathcal{D}_T$, $\mathcal{L}$, *rule_set*), given in Algorithm 7.3

Phase 3: Specialize any overlapping rule pairs
for all *clusters* $S_i \in \mathcal{L}$ *and* $S_j \in \mathcal{L}$, *such that* $S_i \neq S_j$ **do**
    if *the rules predicting* L_i *and* L_j *in rule_set classify at least one common example* **then**
        call function AddDifferentiatingRule($\mathcal{D}_T$, S_i, S_j, *rule_set*), given in Algorithm 7.4
    end if
end for

Algorithm 7.2: Pseudocode of the SIG* algorithm's overall structure.

overlap if the sets of training examples that they classify are not distinct, as Figure 7.4 shows. Section 7.3.3 deals with differentiating condition construction in more detail.

7.3.2 Characterizing Rule Construction Procedure

The method for building the characterizing rules that each describe one of the emergent clusters of a trained SOM relies on a single *significance matrix*. The matrix records the relative significance of each of the components of the training examples that map to every map cluster, where every component corresponds to a unique attribute.

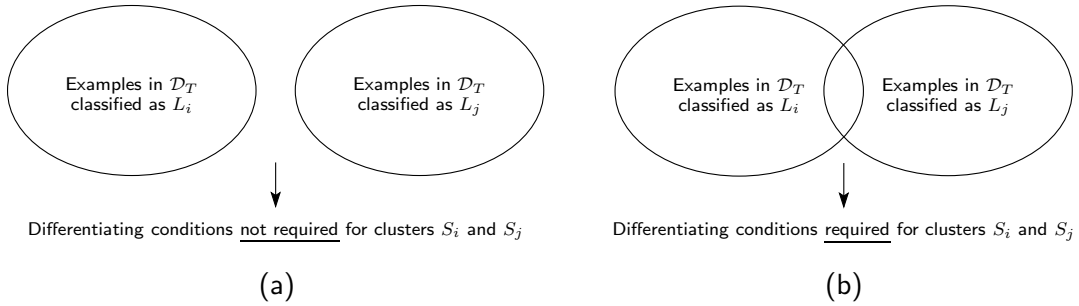


Figure 7.4: The condition governing when SIG* builds differentiating conditions for clusters S_i and S_j: (a) shows no overlap between the training examples classified as L_i and L_j, thus requiring no differentiating conditions for S_i and S_j; (b) shows an overlap between the training examples classified as L_i and L_j, thus requiring conditions differentiating between S_i and S_j.

The characterizing rule construction procedure has three continuous-valued user parameters. Firstly, a characterizing rule threshold value, denoted as θ_{char}, is a percentage in the range $(0, 100]$, and influences the number of attributes that are chosen for each characterizing rule. Lower threshold values result in fewer attributes per characterizing rule, while higher values cause more attributes to be selected.

The second and third parameters are the low and high characterizing condition parameters, respectively denoted ψ_{char} and ϕ_{char}. These parameters are both components in the calculation of attribute value ranges for rule conditions. Both parameters are in the range $(0, \infty)$, and affect the range of values included in each characterizing rule's attribute conditions. Higher characterizing condition parameter values result in wider attribute condition bounds, while lower values cause narrower bounds.

The characterizing rule building procedure consists of four sequential steps, outlined in Algorithm 7.3. The first step populates the significance matrix with initial values. The second and third steps process the matrix using cell marking and value normalization. The final step uses the significance matrix to build the conditions of the rules that describe each emergent neuron cluster. Each step is elaborated upon individually.

Step 1: Populate the Characterizing Significance Matrix

Prior to the construction of characterizing rules, an empty $I \times k$ characterizing significance matrix must be defined. The significance matrix consists of I rows, where I is the

begin function AddCharacterizingRules($\mathcal{D}_T$, $\mathcal{L}$, *rule_set*) **:**

Define an empty $I \times k$ matrix of continuous significance values, denoted *mat_char*
Define user-specified algorithmic parameters θ_{char}, ψ_{char} and ϕ_{char}

Step 1: Populate the characterizing significance matrix with significance values
for all *attributes* $A_l \in \mathcal{D}_T$ *and clusters* $S_i \in \mathcal{L}$ **do**
Set mat_char_{li} to $sig(A_l, N_i)$, the significance of A_l in example set N_i linked with S_i
end for

Step 2: Mark appropriate cells in the characterizing significance matrix
for all *attributes* $A_l \in \mathcal{D}_T$ **do**
Mark all cells in row l of *mat_char*, where $mat\_char_{li} = \max_{\forall j}\{mat\_char_{lj}\}$
end for

for all *clusters* $S_i \in \mathcal{L}$ **do**

Step 3: Normalize the values in the significance matrix column for cluster S_i
Calculate the total for column i in matrix *mat_char*, and denote it $total\_char_i$
for all *attributes* $A_l \in \mathcal{D}_T$ **do**
Update $mat\_char_{li} = (mat\_char_{li} \div total\_char_i) \cdot 100$
end for

Step 4: Define a rule for cluster S_i, using the most significant attributes of S_i
Add $rule_i$, an empty antecedent that predicts label L_i of S_i, to *rule_set*
Define, for column i, an attribute set $select\_char_i$ and an accumulator $sum\_char = 0$

while $sum\_char < \theta_{char}$ **do**
Find $A_l \notin select\_char_i$, such that $\forall A_{l'} \notin select\_char_i : mat\_char_{li} \geq mat\_char_{l'i}$
Add A_l to $select\_char_i$, and update $sum\_char = sum\_char + mat\_char_{li}$
end while

Add to $select\_char_i$ all $A_l \notin select\_char_i$ that are marked in column i of *mat_char*

for all *attributes* $A_l \in select\_char_i$ **do**
Compute $l\_char(A_l, S_i)$ and $h\_char(A_l, S_i)$ using Equations (7.2) and (7.3)
Add conjunctive condition, $A_l \in \big[l\_char(A_l, S_i), h\_char(A_l, S_i)\big]$, to $rule_i$
end for
end for

end function

Algorithm 7.3: Pseudocode of the SIG* algorithm's procedure for adding characterizing rules (based on a training data set, $\mathcal{D}_T$, and a neuron cluster set, $\mathcal{L}$) to a rule set denoted as *rule_set*.

number of attributes in the original training data set (and thus the number of training example dimensions in the map). The number of matrix columns, k, is equivalent to the number of discovered emergent neuron clusters on the map. The value stored in the cell at row l and column i of the significance matrix is denoted as mat_char_{li}. The value stored at mat_char_{li} numerically represents the relative significance of attribute A_l within the set of mapped examples, N_i, which is associated with cluster S_i.

The first main step that the SIG* algorithm uses for building a characterizing rule set involves populating each cell of the characterizing significance matrix with an initial value. Each matrix cell value, mat_char_{li}, is derived from a statistical significance measure, which is represented as $sig(A_l, N_i)$. The $sig(A_l, N_i)$ measure must be chosen to compute a numeric representation of the significance of attribute A_l within the set of mapped training examples that is linked with emergent neuron cluster S_i.

Ultsch's original work does not elaborate on the exact nature of the significance value, beyond mentioning that a statistical method is likely to be used. This dissertation's previous discussion on example-based cluster labeling, presented in Section 6.3.4.2, has mentioned several appropriate $sig(A_l, N_i)$ measures. These measures are all applicable to the SIG* algorithm's computations of characterizing matrix values.

Figure 7.5 illustrates a hypothetical example of the process for populating a characterizing significance matrix with values. The example is based on a SOM trained on a data set with four attributes. The example also assumes that three emergent neuron clusters have been discovered, and that the set of the SOM's original training examples have been mapped to these clusters. The matrix thus has four rows and three columns. This example is extended throughout the remainder of this section and the next.

Step 2: Mark Cells in the Characterizing Significance Matrix

The second step of the characterizing rule extraction procedure begins processing the characterizing significance matrix by marking highly significant cells within the matrix. The marking process considers each row of the characterizing significance matrix separately, and flags the single cell that holds the highest significance value for that row. The marks associated with matrix cells are used to indicate attributes that are forced to appear as part of the characterizing rule of an emergent neuron cluster.

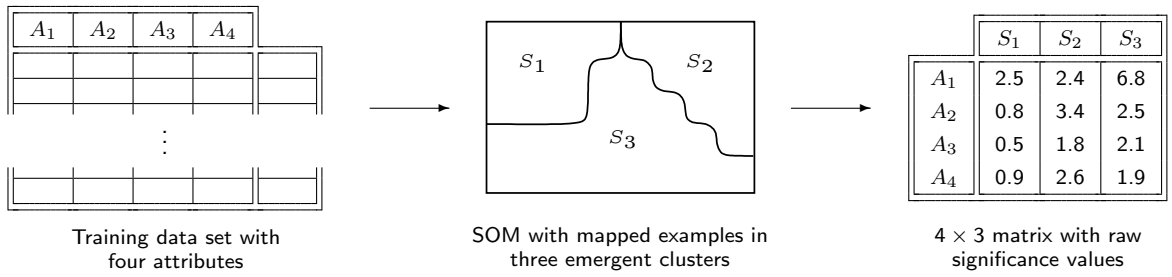


Figure 7.5: A hypothetical example of a SIG* characterizing rule significance matrix, populated with initial raw significance values. The matrix is for a SOM trained on a data set with four attributes (A_1, A_2, A_3, and A_4), with three emergent clusters (S_1, S_2, and S_3).

The objective of the marking procedure is to ensure that the cluster for which a particular attribute is the most significant will include that attribute in the cluster's characterizing rule description. A side-effect of the marking process is that every attribute in the SOM's data model will definitely be included in the rule set.

Figure 7.6 shows the outcome of the cell marking procedure for the example characterizing significance matrix of Figure 7.5. The original un-augmented significance matrix appears to the left of the figure, while the marked matrix is on the right. Each marked cell is annotated with an asterisk symbol. This example illustrates that it is possible for some cluster columns to contain more than one marked cell. Similarly, certain cluster columns might have no matrix cells that are marked by this step of the algorithm.

Step 3: Normalize the Characterizing Significance Matrix

The characterizing rule construction method's third step completes the processing of the characterizing significance matrix, prior to the actual rule building procedure commencement. The process normalizes cell values in the characterizing significance matrix, so that all the attributes within an emergent cluster can be compared in terms of a contribution to the cluster's description, instead of absolute significance value magnitudes.

The normalization process first requires totals to be calculated for all the raw significance values in each matrix column. Each cell value is then re-computed as the percentage that the cell's value contributes to the total calculated for the cell's column. Finally, each cell's computed percentage replaces the raw significance value.

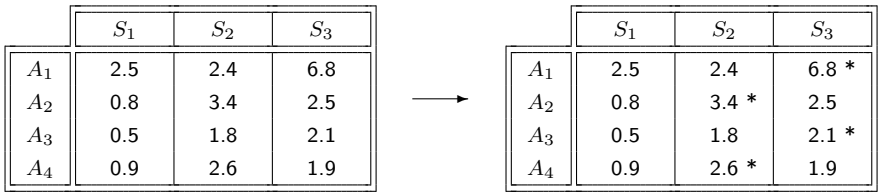


Figure 7.6: The marking of appropriate cells within the initially populated SIG* characterizing matrix of Figure 7.5. In each row of the significance matrix, the cell containing the maximum significance value is marked. Marked cells are indicated with an asterisk.

Figure 7.7 illustrates the normalization of the marked matrix shown in Figure 7.6. The original matrix of raw significance values is shown on the left, with the totals of each column's significance values included at the bottom of the table. The normalized table is shown to the right of the figure. Both tables again use the asterisk symbol to indicate the matrix cells that were marked during the previous algorithmic step.

Step 4: Define a Rule for Each Cluster

The final step of the SOM's characterizing procedure builds up one rule that describes each emergent cluster on the map structure. The rule definition procedure commences by selecting maximally descriptive attributes for each cluster, and constructing rule conditions out of these attributes. The process uses the marked cells and normalized values that were computed during steps 2 and 3 of the characterizing procedure.

The algorithm iterates through each column of the characterizing matrix, and selects attributes one at a time. Selected attributes are associated with the cluster represented by the current column. A user-defined parameter, called the characterizing threshold and represented as θ_{char}, is required. For each matrix column, attributes are selected in descending order of normalized significance. A cumulative total is kept for each matrix column, which stores the sum of the normalized significance values for the selected attributes. Attributes are added to the set of selected attributes for a column until the cumulative total reaches or exceeds the specified characterizing threshold.

Clearly, the most descriptive attributes are selected first. Should one or more attributes have a particularly high significance in a column, fewer will be selected. Con-

	S_1	S_2	S_3
A_1	2.5	2.4	6.8 *
A_2	0.8	3.4 *	2.5
A_3	0.5	1.8	2.1 *
A_4	0.9	2.6 *	1.9
Total	4.7	10.2	13.3

Matrix with marked significance values

	S_1	S_2	S_3
A_1	53.2%	23.5%	51.1% *
A_2	17.0%	33.3% *	18.8%
A_3	10.6%	17.7%	15.8% *
A_4	19.2%	25.5% *	14.3%

Matrix with normalized significance values

Figure 7.7: The normalization of values within the marked SIG* characterizing matrix of Figure 7.6. Each value is normalized as a percentage of the total significance in the value's column. Significance totals per column are indicated in the non-normalized matrix.

versely, if all attributes have fairly low significances, many will be chosen. The characterizing threshold affects the number of attributes selected for each characterizing rule. More attributes will tend to be selected to meet the requirements of a high threshold. Conversely, fewer attributes will be chosen before a low threshold is exceeded.

Once every column's attributes have been selected, subject to the characterizing threshold, the SIG* algorithm takes marked matrix cells into consideration. Each column's unselected attributes are investigated. If any marked attributes have not been chosen for a column, those attributes are also selected for the column's cluster.

Figure 7.8 illustrates the selection of attributes from the marked and normalized significance matrix of Figure 7.7. The normalized and marked significance matrix is shown to the left of the figure, while the right of the diagram lists the attributes that are selected for each cluster. A characterizing threshold of 50% was used. Each selected attribute is followed by an annotation in parentheses. If the annotation is a normalized significance value, the attribute significance was the deciding factor for selection. If the annotation is the keyword marked, attribute selection occurred only because the attribute's cell was marked during step 2. Also shown for each cluster, is the total of the normalized significances for cells chosen only due to the characterizing threshold.

After the attribute selection process has been completed, the characterizing rules that describe the essential characteristics of the map are finally built. A rule is generated for each discovered emergent cluster, where the label of the cluster is used as the consequent of the rule. For each emergent cluster, every attribute that was selected is used to create

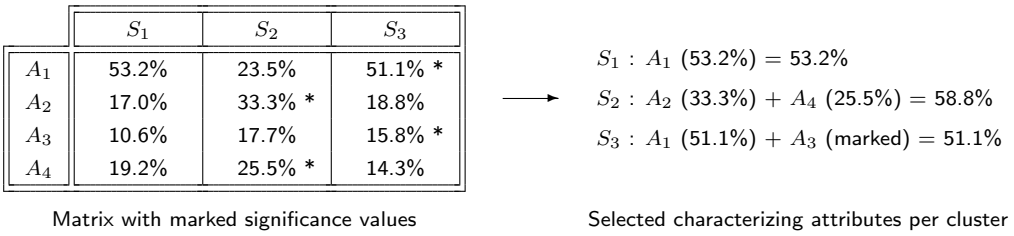


Figure 7.8: Characterizing attribute selection, using the normalized SIG* characterizing matrix of Figure 7.7, and a 50% threshold. Selected attribute significances and cumulative significances are shown. Attributes listed as marked are chosen only due to a marked cell.

one antecedent condition. The condition that is created for attribute A_l within the characterizing rule for emergent cluster S_i has the following general form:

$$A_l \in \left[l\_char(A_l, S_i), h\_char(A_l, S_i)\right]$$

where $l\_char(A_l, S_i)$ denotes the lower bound of the characterizing rule's attribute condition, and $h\_char(A_l, S_i)$ represents the upper bound for the attribute's value condition. All of the selected attribute tests for a rule are linked by logical conjunctions.

The lower bound for a condition on the value of attribute A_l, for the characterizing rule that predicts the label that is associated with cluster S_i, is defined as follows:

$$l\_char(A_l, S_i) = mean(A_l, N_i) - \left(\psi_{char} \cdot dev(A_l, N_i)\right) \tag{7.2}$$

where $mean(A_l, N_i)$ is the mean value for attribute A_l over the mapped example set N_i, and $dev(A_l, N_i)$ is the standard deviation of attribute A_l over example set N_i. Similarly, the upper bound on the condition for attribute A_l in the rule for cluster S_i is:

$$h\_char(A_l, S_i) = mean(A_l, N_i) + \left(\phi_{char} \cdot dev(A_l, N_i)\right) \tag{7.3}$$

The parameters ψ_{char} and ϕ_{char} respectively affect the lower and upper bounds of the condition. Smaller values result in narrower value ranges, and larger values produce wider value ranges. The separation between the two parameters and equations allows independent fine-tuning of the two bounds relative to the attribute's mean value.

Figure 7.9 depicts the example rule set that is generated from the selected attributes shown in Figure 7.8. The rules predict each of the three emergent clusters present in the original SOM, and show a condition for each of the selected attributes. The exact condition bounds have been omitted from the figure, in order to simplify the example.

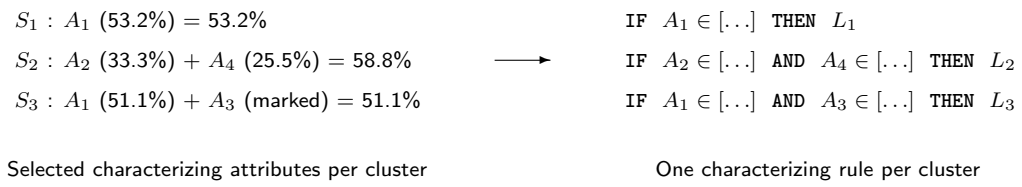


Figure 7.9: The construction of a SIG* characterizing rule for each emergent cluster, using the selected characterizing attributes of Figure 7.8. The actual bounds selected for the attribute conditions, which form part of each characterizing rule's antecedent, are omitted.

7.3.3 Differentiating Condition Construction Procedure

Rules with overlapping conditions will not be able to differentiate between similar training examples. Consequently, the SIG* algorithm attempts to specialize pairs of characterizing rules that have overlapping descriptions. Pairs of overlapping rules are identified when both rules independently match any training set examples. All possible pairs of characterizing rules are compared, to determine which require specialization.

Unfortunately, the details of the differentiating condition building sub-procedure are not clearly specified in the original literature on SIG* [243] (the process is described simply as an "analog algorithm", similar to the characterizing rule building procedure). This chapter's discussion on the differentiating procedure is therefore based as closely as possible on Ultsch's original algorithm, but may differ in some respects.

The differentiating condition building procedure has three parameters. The first parameter, θ_{diff}, specifies a continuous differentiating threshold percentage in the $(0, 100]$ range, which adjusts the number of attributes selected for differentiating conditions. Higher values for θ_{diff} result in a greater number of attributes being selected, while lower values incorporate a smaller number of attributes into the conditions.

The second and third differentiating condition construction parameters are referred to as ψ_{diff} and ϕ_{diff}, and are continuous values in the range $(0, \infty)$. As the ψ_{char} and ϕ_{char} parameters do for characterizing conditions, both ψ_{diff} and ϕ_{diff} affect the range of values that are incorporated into differentiating condition attribute tests. Higher parameter values result in wider attribute value condition boundaries, and lower values cause narrower ranges. The two parameters again allow separate fine-tuning of the upper and lower condition bounds, in relation to the attribute's mean.

The SIG* algorithm's differentiating condition building sub-procedure is illustrated in Algorithm 7.4. The entire procedure consists of three steps, and is repeated for every characterizing rule pair that is judged to overlap. The first step involves building and populating a differentiating significance matrix for the rule pair. The second step normalizes the significance values that are stored in the differentiating matrix. The final sub-procedure step performs the actual specialization of both the overlapping rules. Each of the differentiating procedure's steps is investigated separately, in more detail.

Step 1: Populate the Differentiating Significance Matrix

For any overlapping pair of characterizing rules, an empty differentiating significance matrix must be defined prior to rule specialization. The differentiating significance matrix has one column and I rows. Each matrix row represents an attribute in the SOM's model. Only one column is required because the matrix stores significance values that tell the overlapping clusters apart, rather than describe the clusters separately.

The first step of the rule differentiation sub-procedure populates a differentiating significance matrix for a selected pair of overlapping characterizing rules. In contrast to the significance matrix for characterizing rules, the differentiating significance matrix cell mat_diff_{l1}, in column l, holds a value representing the significance of attribute A_l in terms of telling the overlapping example sets N_i and N_j apart. It should be noted that the example sets N_i and N_j are linked to clusters S_i and S_j, respectively. The differentiating matrix values are computed using a statistic represented as $sig(A_l, N_i, N_j)$.

While Ultsch unfortunately does not describe the specific nature of the differentiating significance statistic, it is possible to utilize any measure that compares the attribute values that make up two mapped sets of training examples. To perform such a comparison, it is possible to modify the $sig(A_l, N_i)$ significance measures for unsupervised example-based cluster labeling that are based on a comparison between the N_i set and the $out(N_i)$ set of Equation (6.8). The modified statistics should, however, simply compare the example sets N_i and N_j, rather than the N_i and $out(N_i)$ sets of examples.

Figure 7.10 illustrates a hypothetical example of the differentiating matrix construction procedure, based on the example characterizing rules defined in Figure 7.9. The example assumes that an overlap exists between the examples classified by the char-

begin function AddDifferentiatingRule($\mathcal{D}_T$, S_i, S_j, *rule_set*) **:**

Define an empty $I \times 1$ matrix of continuous significance values, denoted *mat_diff*
Define user-specified algorithmic parameters θ_{diff}, ϕ_{diff} and ψ_{diff}

Step 1: Populate the differentiating significance matrix with significance values

for all *attributes* $A_l \in \mathcal{D}_T$ **do**
Set mat_diff_{l1} to $sig(A_l, N_i, N_j)$, the significance of A_l in telling N_i and N_j apart
end for

Step 2: Normalize the values in the significance matrix

Calculate the column total for matrix *mat_diff*, and denote it *total_diff*
for all *attributes* $A_l \in \mathcal{D}_T$ **do**
Update $mat\_diff_{l1} = (mat\_diff_{l1} \div total\_diff) \cdot 100$
end for

Step 3: Specialize rules for S_i and S_j, using their most significant attributes

Retrieve $rule_i$ predicting L_i and $rule_j$ predicting L_j from *rule_set*
Define empty differentiating condition expressions $cond\_diff_i$ and $cond\_diff_j$
Define an attribute set *select_diff* and an accumulator $sum\_diff = 0$

while $sum\_diff < \theta_{diff}$ **do**
Find $A_l \notin select\_diff$, such that $\forall A_{l'} \notin select\_diff : mat\_diff_{l1} \geq mat\_diff_{l'1}$
Add A_l to *select_diff*, and update $sum\_diff = sum\_diff + mat\_diff_{l1}$
end while

for all *attributes* $A_l \in select\_diff$ **do**
Calculate $l\_diff(A_l, S_i)$ and $h\_diff(A_l, S_i)$ using Equations (7.4) and (7.5)
Add a disjunctive condition, $A_l \in \left[l\_diff(A_l, S_i), h\_diff(A_l, S_i)\right]$, to $cond\_diff_i$
Calculate $l\_diff(A_l, S_j)$ and $h\_diff(A_l, S_j)$ using Equations (7.4) and (7.5)
Add a disjunctive condition, $A_l \in \left[l\_diff(A_l, S_j), h\_diff(A_l, S_j)\right]$, to $cond\_diff_j$
end for

Add $cond\_diff_i$ as a new conjunctive condition to $rule_i$
Add $cond\_diff_j$ as a new conjunctive condition to $rule_j$

end function

Algorithm 7.4: Pseudocode of the SIG* algorithm's procedure for differentiating the characterizing rules for clusters S_i and S_j (based on a training set, denoted as $\mathcal{D}_T$), and an existing set of characterizing rules previously generated from $\mathcal{D}_T$, which is denoted as *rule_set*.

acterizing rules for clusters S_1 and S_3, but not between any other pairs of rules. A differentiating condition must therefore be built for both of the overlapping rules, using a single differentiating significance matrix as a base. The matrix has four rows, each of which is associated with one of the attributes represented by the SOM.

Step 2: Normalize the Differentiating Significance Matrix

The second step of the differentiating condition building process requires the normalization of the values contained in the differentiating matrix. The normalization of a differentiating significance matrix is similar to the normalization of the characterizing significance matrix. A significance value total is calculated for the single differentiating matrix column, and cell values are normalized as a percentage of the total.

Figure 7.10 shows the outcome of the value normalization for the significance matrix used in the differentiating condition attribute selection example. The total for the single matrix column is shown in the non-normalized matrix. To the right of the unmodified matrix of raw significance values appears the normalized matrix of percentages.

Step 3: Specialize the Rules for the Overlapping Clusters

The final step of the SIG* algorithm's specialization procedure involves the actual construction of the differentiating conditions. Once built, the differentiating conditions must also be added to the two existing characterizing rules that are being specialized.

First, attributes are selected to make up the differentiating conditions. Attributes are chosen in descending order of normalized significance, starting with the highest value. A total of all the selected normalized significance values is kept. Attributes are selected until the total reaches or exceeds the value of the θ_{diff} parameter.

Figure 7.10 shows the attribute selection step for the rule specialization example, which used a 50% differentiating significance threshold. Attribute A_2 and A_4 are thus chosen, with a normalized significance total of 61.3% exceeding the threshold.

After the selection of appropriate attributes for an overlapping pair of characterizing rules, a separate differentiating condition is built for each of the two rules. Each differentiating condition consists of several value boundaries, each associated with one of the selected differentiating attributes. A boundary associated with an attribute has the

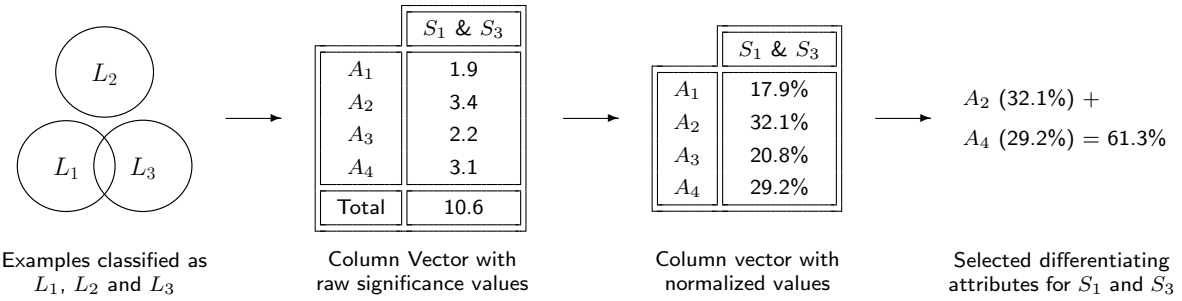


Figure 7.10: Differentiating attribute selection for the characterizing SIG* rules of Figure 7.9, given an overlap between examples classified as L_1 and L_3. In the normalized matrix, values are a percentage of the column's total significance. A differentiating threshold of 50% is used, and the selected attributes are annotated with their normalized significance values.

same form as the attribute conditions used in characterizing rules. The general structure of an attribute condition within a differentiating condition is as follows:

$$A_l \in \left[l\_diff\,(A_l, S_i), h\_diff\,(A_l, S_i) \right]$$

The differentiating condition bounds are similar to characterizing bounds. A differentiating condition's lower bound on A_l, for the characterizing rule of S_i, is:

$$l\_diff\,(A_l, S_i) = mean(A_l, N_i) - \left(\psi_{diff} \cdot dev(A_l, N_i) \right) \tag{7.4}$$

In a similar fashion, the upper value boundary on the differentiating condition for attribute A_l, defined for the characterizing rule that predicts S_i, is computed as:

$$h\_diff\,(A_l, S_i) = mean(A_l, N_i) + \left(\phi_{diff} \cdot dev(A_l, N_i) \right) \tag{7.5}$$

The ψ_{diff} and ϕ_{diff} parameters have effects that are similar to the influence that the user-defined ψ_{char} and ϕ_{char} parameters have on characterizing condition boundaries. Smaller parameter values reduce condition ranges, while larger values result in a wider range. As was the case for the characterizing rule conditions, the two separate parameters allow for a separate fine-tuning of the upper and lower bounds.

The constructed attribute conditions for a specialized characterizing rule are combined into a complete differentiating condition. The construction of such complete conditions simply involves linking the differentiating attribute conditions using logical disjunctions. According to the original algorithm description, disjunctions are used because

characterizing rule conditions should generally be stronger descriptions than differentiating conditions. As a final step, the constructed differentiating condition is simply added to the appropriate characterizing rule's antecedent, using a logical conjunction.

Figure 7.11 illustrates how the example's selected differentiating attributes from Figure 7.10 are combined with the original characterizing rules shown in Figure 7.9, to form a finally specialized rule set. Two differentiating conditions of the same general form are built from the set of selected attributes. The differentiating conditions are then added to the rules predicting L_1 and L_3, the labels of the overlapping clusters S_1 and S_3. To illustrate the form of the rules more clearly, the exact condition bounds are omitted.

7.3.4 Converting SIG* Rules into Production Rules

Many DM algorithms, including those compared to SIG* in Chapter 8, output production rules with simple conjunctions of attribute tests as antecedents. The SIG* algorithm's rule set is clearly not in this form. It is thus often necessary to convert SIG* rules into general production rules, to allow a fair structural comparison of algorithm results.

Should a characterizing rule contain any differentiating conditions, the rule antecedent must be converted into a Boolean expression that is composed only out of a conjunction of attribute value conditions. Standard Boolean algebra [267] is used to perform the translation. For example, the following SIG* characterizing rule:

$$\begin{aligned}\texttt{IF}\ A_1 \in [\ldots]\ \texttt{AND}\ \big(A_2 \in [\ldots]\ \texttt{OR}\ A_3 \in [\ldots]\big)\\ \texttt{AND}\ \big(A_4 \in [\ldots]\ \texttt{OR}\ A_5 \in [\ldots]\big)\ \texttt{THEN}\ L_1\end{aligned}$$

is converted into the following set of separate rules, where each rule has an antecedent that consists of only a conjunction of conditions on the values of the attributes:

$$\begin{aligned}&\texttt{IF}\ A_1 \in [\ldots]\ \texttt{AND}\ A_2 \in [\ldots]\ \texttt{AND}\ A_4 \in [\ldots]\ \texttt{THEN}\ L_1\\ &\texttt{IF}\ A_1 \in [\ldots]\ \texttt{AND}\ A_2 \in [\ldots]\ \texttt{AND}\ A_5 \in [\ldots]\ \texttt{THEN}\ L_1\\ &\texttt{IF}\ A_1 \in [\ldots]\ \texttt{AND}\ A_3 \in [\ldots]\ \texttt{AND}\ A_4 \in [\ldots]\ \texttt{THEN}\ L_1\\ &\texttt{IF}\ A_1 \in [\ldots]\ \texttt{AND}\ A_3 \in [\ldots]\ \texttt{AND}\ A_5 \in [\ldots]\ \texttt{THEN}\ L_1\end{aligned}$$

The reader should note that the converted set of rules is generally more complex than the original SIG* rule, both in terms of the number of rules and the number of attribute

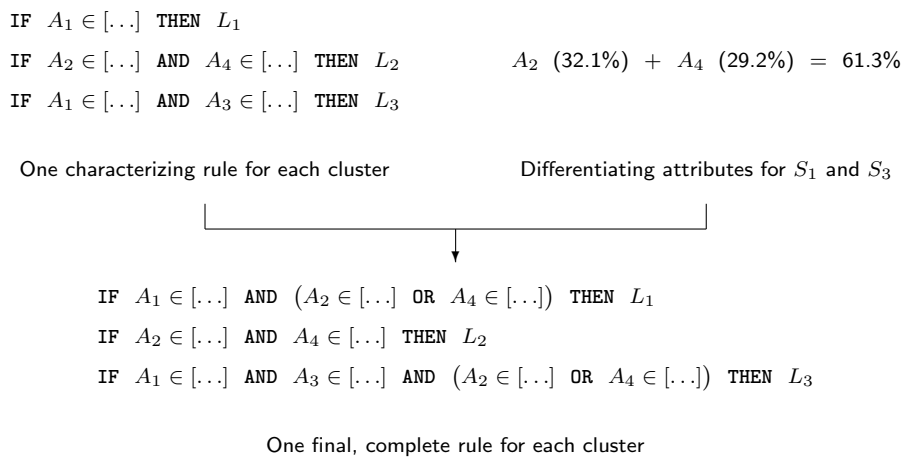


Figure 7.11: The construction of final SIG* rules, using the characterizing rules of Figure 7.9, and the differentiating attributes for clusters S_1 and S_3 from Figure 7.10. Differentiating conditions are added to the rules predicting L_1 and L_3. Actual attribute ranges are omitted.

conditions. This effect is exacerbated by a large number of differentiating conditions, as well as by differentiating conditions that consist of many attribute tests.

Many DM algorithms, including the algorithms that the SIG* algorithm is compared against in Chapter 8, use only equalities and inequalities in attribute conditions. Clearly, the SIG* algorithm's bounded attribute conditions have a different syntax.

Bounded conditions are easily translatable into two equality-based conditions, which are linked by a logical conjunction. The first condition tests the lower bound of the range, while the second condition tests the upper bound. For example, the condition:

$$A_1 \in [0.2, 5.1]$$

will be converted into the following equivalent logical conjunction, which contains two separate production rule conditions on the value of the same data attribute, A_1:

$$(A_1 \texttt{ >= 0.2) AND (}A_1 \texttt{ <= 5.1)}$$

It should, of course, be borne in mind that this translation necessarily increases the complexity of the rule set, in terms of the number of conditions contained in each of the rules. This effect is greater when rules have many conditions, and is further magnified if many rules exist due to the conversion of antecedents into conjunctions of conditions.

7.3.5 A Critique of the Approach

The SIG* algorithm was the only SOM-based data mining algorithm for more than a decade, and is consequently an important baseline algorithm against which all SOM-based rule extraction algorithms should be analyzed and measured. However, this research work's analysis of the SIG* approach, which is based on the interpretation of the algorithm as it was described within this chapter, has identified several drawbacks and shortcomings associated with the technique. This section discusses these drawbacks theoretically, and some are experimentally investigated within Chapter 8.

Firstly, the SIG* algorithm bases rules on discovered emergent clusters of neurons. As Chapter 5 illustrated, there are many potential drawbacks to different clustering algorithms. Any drawback linked to a clustering algorithm used by SIG* will implicitly be associated with the operation of the SIG* implementation. Drawbacks include the time-consuming nature of an exploratory clustering approach, the need to select an appropriate number of clusters that a partitional algorithm should produce, and the level at which a hierarchical technique's dendrogram should be cut.

Real-world SOM models also often include emergent clusters that are insignificant (because the cluster is very small, very few data examples fall within the cluster, or the cluster is simply uninteresting). While data mining and exploratory data analysis tools often eliminate insignificant rules, SIG* requires the inclusion of a characterizing rule for every emergent cluster, regardless of how insignificant the cluster may be.

In the third place, because of the cell marking procedure used during the construction of characterizing rules, every attribute is used to describe at least one emergent cluster. Redundant or unimportant data set attributes will, however, not contribute to a description of a data set. Many data mining and exploratory data analysis methods eliminate unimportant attributes. However, SIG* enforces the inclusion of even insignificant attributes in the rule set, leading to potential rule description redundancy.

Fourthly, adding differentiating conditions to both rules in an overlapping pair, rather than just one, is likely to introduce a large degree of redundancy to the rule set. Furthermore, differentiating conditions are generated for rule pairs that have even a single classified example in common. The specialization of a rule to eliminate a single positive example is also likely to require a great deal of detail. It is thus likely that very minor

overlaps between rules will result in a large increase in rule set complexity. Moreover, when a characterizing rule overlaps with several other rules, the differentiating conditions for every overlapping pair will cause an even greater complexity increase.

Related to the fourth drawback, it is also likely that the addition of many differentiating conditions will lead to an over-specialization of rules. The result of such an over-specialization will be a rule set that very closely models the SOM's training data, while not generalizing well to unseen data examples. The rule set will thus perform less accurately on examples that are not available during map training.

In the sixth place, component mean and standard deviation calculations over mapped training example sets form the basis for attribute value conditions. It is likely that means and standard deviations will be an inaccurate approximation of the underlying component distributions for examples mapped to very irregular clusters. Rule performance is therefore likely to degrade when describing these types of clusters.

In a similar vein to the previous drawback, the accuracy of extracted rule sets is likely to become a concern if a relatively small number of training examples fall within one or more emergent clusters on the map. Means and standard deviations for these clusters will therefore be calculated on fewer example observations, which will reduce the accuracy of the derived characterizing rule attribute conditions.

The execution time of the SIG* algorithm is also of some concern. The very complex characterizing and differentiating processes (which involve repeated matrix-building and significance calculations) increase execution time. Several clustering algorithms are fairly time complex, and will also slow down SIG* if used as part of the algorithm.

Finally, the SIG* algorithm is incredibly complicated in comparison to many other rule extraction algorithms, as is attested to by the lengthy description of the approach. This potentially results in a very cumbersome and time-consuming algorithm implementation process, and a widened scope for programmatic errors to be introduced.

7.4 The HybridSOM Framework

The HybridSOM approach is a novel method, introduced by this research work. HybridSOM is not an algorithm in the true sense of the word, but rather a framework that

allows for the combination of any rule extraction algorithm with a SOM. The framework is thus a highly adaptable general approach to SOM-based rule extraction.

A broad overview of the approach is presented in Section 7.4.1, while Section 7.4.2 provides a more detailed outline of the rule extraction procedure. Finally, an analysis of the framework's strengths and weaknesses is presented within Section 7.4.3.

7.4.1 An Overview of the Approach

The HybridSOM framework is predicated on three facts. Firstly, the SOM's weight vectors model the characteristics of the map's training data set. Secondly, SOMs typically contain fewer weight vectors than the number of training examples [68], meaning that the weight vector model is less complex than the training data set. Thirdly, a SOM's weight vectors are equivalent in structure to unlabeled training set examples.

Based on these three assumptions, it is possible to hypothesize that an arbitrary DM algorithm's application to a trained SOM's weight vectors will be roughly equivalent to the DM algorithm's application on the map's original training data examples. Furthermore, it can be hypothesized that the rule set that is produced by a DM algorithm executed on a trained SOM's weight vectors should be similar to the rules produced if the same DM algorithm is executed on the map's raw training data.

The overall structure of the HybridSOM framework's operation is presented graphically in Figure 7.12. A pseudocode approach outline is also provided in Algorithm 7.5.

7.4.2 The Rule Extraction Procedure

The HybridSOM framework includes a SOM component, and is configured with a neuron labeling method and a rule extraction algorithm. The labeling technique is not required to apply a label to every neuron in the map. As a result, any one of the neuron labeling approaches that were discussed in Chapter 6 is appropriate for HybridSOM. Furthermore, any rule extraction algorithm that processes a data set using the table-based data model is usable, including rule induction approaches (e.g., CN2 [38] and AQ15 [171]), decision tree building techniques (e.g., C4.5 [193], ID3 [190], and ASSISTANT 86 [30]), and computational intelligence methodologies (e.g., Ant-Miner [179] and GPMCC [186]).

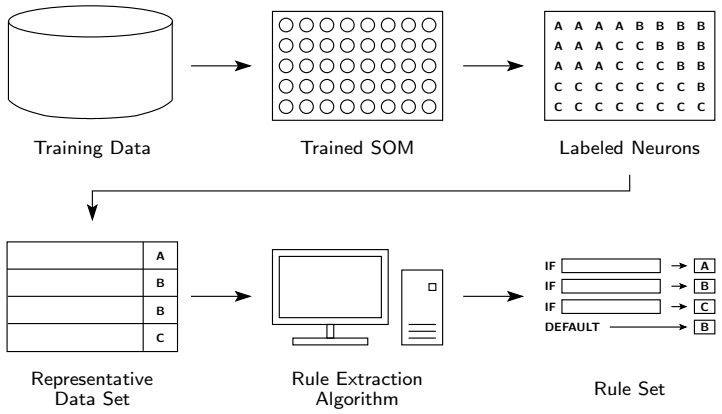


Figure 7.12: The component interactions within the HybridSOM framework.

Create and initialize a SOM, *map*, and train it on an I-attribute training set, $\mathcal{D}_T$
Apply a label to the neurons corresponding to every $\vec{w}_{yx}$ that make up *map*
Define a data set, *input*, with the same I attributes that are found in $\mathcal{D}_T$
for all *labeled weight vectors,* $\vec{w}_{yx}$*, in the map* **do**
    Define a new example, *record*, and add it to *input*
    Add all w_{yxv} to *record*, as values for their corresponding attribute values
    Add $label_{yx}$ to *record*, as the record classification
end for
Execute rule extractor on *input*, producing a rule set, *output*

Algorithm 7.5: Pseudocode of the HybridSOM rule extraction DM framework.

The HybridSOM method begins by simply initializing and training a SOM on a training data set. Once training has been completed, the map is prepared for mining by labeling the neurons of the map structure, using the chosen labeling method.

After map preparation, a representative proxy data set is built from the map's weight vectors and labels. Each weight vector becomes a new example in the proxy data set, where each weight becomes an attribute value. The label associated with the neuron of a weight vector becomes the classification of the example built from that weight vector. Any unlabeled weight vectors are simply discarded from the proxy data set.

Because map weights correlate to training data attributes, the descriptive attributes of the proxy data set have exactly the same structure as the descriptive portion of the training data set. The correlation between the classification information in the proxy data set and the original training data set obviously depends on the results of the neuron labeling scheme used. When a supervised labeling approach is used, proxy data set classifications will have the same form as those found in the original training data set. If unsupervised labeling approaches are used, it is possible that proxy data set classifications will differ from the original training data.

Once the new data set has been constructed, HybridSOM's final step simply executes the chosen rule extraction algorithm on the proxy data set. The rule set produced by the data mining algorithm then becomes the final output of the framework.

7.4.3 A Critique of the Approach

The HybridSOM framework has a number of advantages. Firstly, the hybrid nature of the framework allows any neuron labeling scheme and any rule extraction algorithm to be used for SOM-based data mining. This introduces a level of flexibility that is not present in either the boundary-based rule extraction algorithm, or SIG*.

Simplicity is the second advantage of the framework, as is demonstrated by the compact algorithm description. The framework's implementation is thus quick and easy, and poses less of a problem than SIG* or the boundary-based method. This is particularly advantageous when a pre-existing general-purpose rule extraction algorithm is available, in which case very little effort is required to use the algorithm with a SOM.

The third advantage of the HybridSOM framework lies in the relative execution efficiency of the approach, because the method requires only a single iteration through the map's neurons, and a simple copy operation on weights and neuron labels.

The framework is not without drawbacks, however. Firstly, the quality of a HybridSOM configuration's output rule set (in terms of rule set accuracy and description complexity) is strongly linked to the rule set quality that the rule extraction algorithm is capable of producing. Thus, if a DM algorithm that is not suited to a particular data domain is used with a SOM trained on such data, HybridSOM's output rules will also perform poorly. It is usually difficult to ascertain whether a DM algorithm is suited to a

certain data set, making appropriate rule extractor selection challenging. By extension, a *generally poor* rule extractor will also produce poor HybridSOM results.

A second drawback associated with HybridSOM is that the framework's overall result quality is also partially linked to the neuron labeling algorithm that the framework is configured with. Inaccurate or unintuitive labels will result in substandard proxy rule set classification information, leading to poor rule extraction algorithm results.

The final drawback of the HybridSOM technique is that the simplicity and time complexity of the neuron labeling and rule extraction algorithms are entirely separate from the framework itself. Consequently, if the implementation of either algorithm is complex, the overall implementation complexity of a HybridSOM configuration will increase. Similarly, if either a very time complex labeling algorithm or rule extractor is chosen, the overall time complexity of the framework configuration will also be adversely affected.

7.5 Miscellaneous Approaches

Several miscellaneous SOM-based data mining techniques have been proposed, which produce different forms of knowledge to the previously discussed methods in this chapter. Further investigation of these miscellaneous approaches is left to future research.

Hung and Huang [116] propose a method similar to the characterizing rule building of SIG*. A one-dimensional SOM is used, contiguous groups of equivalent labels define clusters, and a weighted boundary between clusters defines rule conditions. This method does not scale to two-dimensional maps, and is thus not considered further.

It is possible to generate fuzzy rule sets from trained SOMs. Fuzzy rules differ greatly from production rules in both structure and interpretation, and are thus excluded from this dissertation's focus. Alfred Ultsch has proposed an extension to the SIG* algorithm [242], which builds and outputs sets of fuzzy rules. Pedrycz and Card [181] have also proposed a method to give fuzzy descriptions to neuron clusters.

Vesanto and Hollmén [260] have broadly outlined approaches for optimizing rule condition bounds, based on a classification error function. However, the original publication does not describe the exact nature of the optimization procedure. Further investigation into the effects of attribute value optimization is therefore left to future work.

Work by Fung *et al* [89] has used SOMs as a component in the initial data preparation phase of a complex classification scheme that uses a system of several feedforward neural networks. The SOM is only a minor component in the overall system, and this research thus does not consider the method to be a SOM-based data mining approach.

Siponen *et al* [223] have done some work on evaluating the significance of individual rules in a SOM-derived rule set. The proposed rule significance measure is calculated for a particular rule, relative to a discovered emergent neuron cluster that the rule predicts. As a result, the significance measure is not applicable to rules that do not describe emergent clusters, such as those derived by the HybridSOM framework. The measure's application is thus not general enough to be considered further within this work.

Vesanto and Hollmén [260] have also presented some research on the visualization of rules that describe emergent neuron clusters. This dissertation focuses on rule set performance, rather than the visual representation of rules, and therefore does not focus further on this or any other visualization methods for SOM-derived rules or rule sets.

7.6 The Viability of SOM-based Data Mining

As is evident from this chapter, there are not many SOM-based data mining approaches. This is due largely to the fact that the SOM is a modeling technique suited primarily to visual representations that are interpretable by humans. This means that a SOM's use is traditionally considered very well suited to EDA, but less applicable to DM.

SOM training is very time complex, especially when training examples have many attributes and when the map is large [129]. SOM execution time further suffers in the presence of large training sets. There are thus many data mining methods that are less time complex, and also produce adequate rule sets. As outlined in Section 3.6, the "curse of dimensionality" [16, 19, 216] is also problematic for high-dimensional data.

In addition, it should be recalled that the SOM produces an *approximate* model of the map's training data. As a result, SOM-based rule extractors are essentially built upon a rough approximation of the training data, rather than the unmodified and more precise original set of training data. This dissertation's research work therefore hypothesizes that SOM-based DM algorithms will inevitably produce rules that are at least somewhat less

accurate than rules extracted directly from the raw training data. The degree of accuracy degradation will, of course, depend on the accuracy of the SOM's model.

The question then arises as to whether SOM-based data mining is a worthwhile approach justifying further investigation, and whether SOM-based DM is suitable for any real-world applications. This dissertation proposes that the following general advantages are associated with rule extraction based on a sufficiently trained self-organizing map:

1. In problem domains such as medical data analysis, SOMs have been found to be capable of modeling problem spaces for which other algorithms cannot generate adequate rule sets [243]. Under such conditions, it is possible for one of the SOM-based rule extraction approaches to produce rule sets with improved performance in comparison to other approaches, despite an increase in time complexity.

2. A SOM is a very versatile approach, because trained maps can also be used for EDA purposes. It is possible to train a SOM once, and then use the map as a single platform for a variety of exploratory tasks, *as well as* a basis for automatic rule set generation. It is possible to use such a SOM-based rule set in addition to the exploratory findings. It is also possible to use generated rule sets as additional evidence to verify the results of an EDA exercise, or vice versa.

3. Very large data sets often cause problems for DM algorithms [71], which can exhaust available hardware or software resources. However, because SOM training uses one data example at a time, large data sets do not prevent the training of moderately-sized maps. Furthermore, practical SOMs typically contain far fewer neurons than training examples, meaning that a SOM can substantially reduce the time complexity of algorithms that derive rules from map weight vectors.

4. A SOM's model tends to smooth out noise in the training data set [146]. It is therefore possible that a rule extractor that is based on a SOM's constituent weight vectors will degrade more gracefully in the presence of noise, when compared to a rule extraction algorithm that is executed on a raw training data set.

The last two advantages are only applicable to SOM-based rule extraction algorithms that are based directly on the weight vectors of a trained SOM, such as the boundary-based method and the HybridSOM framework. The SIG* approach is based on training

examples mapped to a SOM structure, and thus does not have these two advantages. Of course, the advantages described under points 1 and 4 require further empirical research to confirm or reject, but this analysis is deferred until future research studies.

7.7 Summary

This chapter dealt with the DM-oriented aspect of SOM use. DM applications are relatively limited due to the fact that SOMs lend themselves primarily to the visual interpretation of EDA, rather than automatic rule extraction. Section 7.1 discussed the philosophy underlying SOM-based DM. Three SOM-based DM approaches were discussed in this chapter. Firstly, the boundary-based rule extraction algorithm was described in Section 7.2. Secondly, the SIG* algorithm was discussed in Section 7.3. Finally, HybridSOM is a novel framework proposed by this research, and discussed in Section 7.4. Section 7.5 outlined miscellaneous approaches that are related to DM with SOMs, and Section 7.6 discussed the practical viability of using a SOM for automatic DM.

The next chapter presents the results of the empirical analysis performed for this work. The chapter experimentally compares configurations of the HybridSOM framework to the SIG* algorithm implementation that was described in this chapter. In addition, the classic CN2 rule induction algorithm and the C4.5 decision tree building technique are compared against. As previously mentioned, this research focuses on unsupervised SOM-based DM, and the boundary-based method is thus not included in the comparison.

Chapter 8

Experimental Results

The previous chapter described the SOM-based DM techniques that can be used for rule extraction. This chapter describes the procedures and statistical measures used during the empirical study conducted for this research. Thereafter, the experimental results obtained during the empirical investigation are presented and discussed.

The experimental analysis performed for this dissertation is split into two separate investigations. The first investigates the performance of the supervised neuron labeling methods for SOMs, which are covered in Chapter 6. The second explores the performance of the SOM-based DM techniques, which are described within Chapter 7.

This chapter is arranged as follows: Section 8.1 discusses the experimental procedures that are applicable to both the aforementioned investigations. Section 8.2 describes the data sets used as a basis for the experiments. Sections 8.3 and 8.4 present and interpret the experimental results for the supervised neuron labeling approaches and the SOM-based DM methods, respectively. Finally, Section 8.5 summarizes the chapter.

8.1 General Experimental Procedure

This section discusses details of the experimental procedure that both the aforementioned investigations share. Section 8.1.1 describes the cross-validation procedure used in the experiments. Section 8.1.2 covers the general analysis techniques and statistics that were used to compare the performance characteristics of the algorithms under investigation.

8.1.1 Cross-Validation Procedure

The performance of each investigated technique was assessed using several classification problems. In other words, both the supervised neuron labeling methods and the SOM-based data mining algorithms were used to apply labels to data examples, and the task performance of each approach was measured. Each classification problem used examples from a well-known benchmark data set. The sets are described in Section 8.2.

It is possible to use several measures to assess the performance of algorithms on classification problems. The specific performance measures used for assessing the various approaches in the two investigations are discussed in Sections 8.3.3 and 8.4.3.

It is insufficient to judge the performance of a technique using only a single simulation on a classification problem [68]. Consequently, several simulations are required, upon which all reported results are based. It is also well known that analysis performed on an algorithm's training data gives an over-estimation of algorithmic performance [157]. Cross-validation [10] is a family of techniques used to judge how the results of a statistical analysis on an algorithm generalize to a data set not presented during training.

The k-fold cross-validation method [174] divides the benchmark data set upon which an algorithm is being assessed into k disjoint subsets, each containing the same number of data examples. A set of k simulations is then executed, where each simulation trains the algorithm on a *unique* training set of $k - 1$ subsets. One subset is not presented during training, and is referred to as the *test data set* for the simulation. Across the k simulations, each data subset is therefore used as a test set in one simulation.

Any performance measures computed using a test set are more indicative of the algorithm's true performance for the simulation on data outside the training set (i.e., the generalization performance). Because the cross-validation uses every data subset as a test set, the full training set is effectively used as a test set. The mean of any test set performance measure over the cross-validation is then an overall estimate of the algorithm's performance on data that is independent of the training set. A representation for the variance of the same measure over the simulations, such as standard deviation, estimates the consistency of the algorithm's performance on unseen data.

All analyses in this chapter used k-fold cross-validation. This choice is based on the fact that cross-validation only assumes that the data examples within the underlying

data sets are identically distributed, and that the training set examples and validation set examples in each simulation are independent [10]. The k-fold cross-validation technique in particular is very widely used, and has reasonable time complexity compared to other cross-validation techniques, such as leave-one-out cross-validation.

For each algorithm tested on a benchmark data set in this chapter, 30-fold cross-validation was performed throughout (in other words, $k = 30$). This decision was based on the fact that the benchmark data sets that were experimented upon are relatively small. This means that a smaller number of folds, which is typically used for large data sets, results in training sets that are too small for adequate training. Conversely, a larger number of folds results in smaller test sets, which are less representative.

When k does not divide perfectly into the number of examples in the full data set, a group of fewer than k examples is left after the subset division. This group is too small to form a test set. The procedure used for these experiments simply added all remaining examples to the training set for each simulation in the cross-validation.

To avoid bias related to the original order of data set examples, each data set was randomly shuffled once, prior to any experiments, and all cross-validations were performed on the same shuffled data set. As a consequence, any results produced by simulations with the same training and test sets are dependent. The Fisher-Yates shuffle [79, 141] was used, because this shuffling algorithm produces an unbiased permutation of the data set. The shuffling algorithm used the MT19937 implementation of the Mersenne Twister pseudo-random number generator [90, 166], because this generator is very commonly used in practice [163] and is considered to be appropriate for simulations [90].

8.1.2 Algorithmic Performance Comparison Procedure

The reported experiments aim to draw general conclusions about the relative performance of the investigated algorithms. A statistical hypothesis test [55] draws conclusions about statistical populations by analyzing sets of samples drawn from the populations. For these experiments, the samples are performance measure values obtained over 30-fold cross-validated simulations. Specifically, the hypothesis test used in this investigation must determine whether any observed performance differences between algorithms cross-validated on the same data set are due only to chance, at a specific significance level.

Demšar [53] suggests using the two-sample Wilcoxon signed-rank statistical hypothesis test [268] to compare two algorithms that are cross-validated on the same data set. The two-sample test is designed for dependent samples, and is thus appropriate for this investigation. This is because, as noted in the previous section, results derived from the same training and test sets in a cross-validation are dependent.

The two-sample Wilcoxon signed-rank test determines a p-value, which is a statistic computed from the two samples of performance values. The p-value is calculated using the ranks of the absolute differences between pairs of performance measure values produced by simulations of the two algorithms on the same training and test data subsets. The p-value computation also uses the sign function of the differences.

The use of the Wilcoxon signed-rank test is appropriate because the test is non-parametric. This means that the test does not assume that the populations represented by the sets of performance measure samples belong to a specific distribution.

The two-tailed version of the test was used for all reported experiments. This test simply indicates whether the population distributions of performance measure values for two algorithms cross-validated on the same data sets differ from one another. The null hypothesis for the two-tailed test is that the population median of paired differences for the two performance measure samples is 0.0. The null hypothesis implies that the populations of performance values are not significantly different from one another, and that the algorithms do not differ with respect to the performance measure and data set under investigation. The alternative hypothesis is that the median is not 0.0, implying a significant difference in performance. In this case, the mean values of the performance measure for the two algorithms were compared to draw conclusions.

The null hypothesis of the test is rejected when the p-value of the test is less than the significance level chosen for the investigation. For the experiments reported here, a significance level of 0.05 was decided upon prior to any experimental analysis. This significance level implies that the null hypothesis is incorrectly rejected only 5% of the time. When performing multiple hypothesis tests, as this investigation does, the family-wise error rate increases, and the probability of incorrectly rejecting the null hypothesis rises. To reduce the family-wise error rate, a Bonferroni correction [21, 22, 61, 62] was performed on the critical value used for the tests. This correction is considered to be

relatively conservative, meaning that the populations of performance measure values are guaranteed to be different if the test indicates the presence of a significant result.

The two-sample, two-tailed Wilcoxon signed-rank test used during this chapter's analysis is the implementation provided within the coin add-on package [113, 114] for the R programming language [198]. The exact conditional distribution of the statistic under the null hypothesis was computed using the Streitberg-Röhmel shift algorithm [228, 229]. During the computation of the test's p-value, it is possible for no differences between ranks to occur. Such differences of 0.0 are handled according to the technique proposed by Pratt [187], whereby all absolute differences are initially ranked, and the ranks that correspond to differences of 0.0 are discarded, but the remaining ranks remain unchanged.

8.2 Experimental Data Sets

This section provides details on the experimental data sets that were used as a platform for the analysis of the techniques investigated in this chapter. All data sets are freely available for download from the UCI Machine Learning Database Repository [3].

8.2.1 Iris Plants Data Set

This section discusses the Iris plants data set. Background information on the data set is given in Section 8.2.1.1, while Section 8.2.1.2 describes the data preparation that was performed on the Iris plants data set prior to the experimental analysis.

8.2.1.1 Data Set Background

The Iris plants data set was produced in 1936 by Sir Ronald Aylmer Fisher, for his work on taxonomic problems [77]. This data set is considered to be a very simple learning domain. Despite this simplicity, the set is a very common benchmark for pattern recognition and data mining applications, and was thus included in the analysis.

The data set consists of tuples that represent individual examples of Iris plants. The descriptive attributes of the data set consist of four attributes, namely `sepal_length`, `sepal_width`, `petal_length`, and `petal_width`. Each attribute value is continuous,

and denotes a measurement that is taken in centimeters. No attribute values are missing for any of the examples making up the data set. Table 8.1 shows the most important characteristics related to the descriptive attributes of the data set.

Each example is classified by means of a single nominal attribute. A classification is either `Iris_Setosa`, `Iris_Versicolor` or `Iris_Virginica`. The distribution of data set examples between these three classes is shown within Table 8.2.

The `Iris_Setosa` class is known to be linearly separable from the other classes, and `Iris_Versicolor` and `Iris_Virginica` are not linearly separable from one another. High correlations to the example classification exist for `petal_length` and `petal_width`, with a moderate correlation for `sepal_length`, and no correlation for `sepal_width`.

8.2.1.2 Data Preparation

As for all the benchmark data sets, a 30-fold cross-validation was performed for each experiment involving the Iris plants data set. Table 8.3 shows the division of examples between the training and test sets that resulted from this procedure.

As is noted in Section 2.3.2, data requires pre-processing before any data analysis or mining begins. Consequently, the basic data preparation techniques that are discussed in Section 2.4 had to be applied to all data sets, where necessary.

Because no attribute values are missing, no replacement strategy was required. This study aims to investigate the performance of the various approaches in the presence of noise, thus no noise reduction was performed. No obvious data set inconsistencies, which require cleaning, exist in the data set. Because no nominal descriptive attributes are present, no binary encoding of nominal values was required either.

Table 8.1 clearly shows that the attributes of the Iris plants data set vary over very different ranges, which necessitates the use of a data normalization procedure. Min-max normalization, which is described in Section 2.4.2.1, preserves the original relationships that are observable between the attribute values within the unmodified data set. Consequently, this normalization approach was chosen to scale all attribute values into the range $[0.0, 1.0]$. No nominal descriptive attributes are present, and therefore no binary encoding was required. Finally, because all the tested labeling and rule extraction methods require nominal classes, no modification of the classification attribute was necessary.

Table 8.1: The characteristics of data attributes in the Iris plants data set.

Attribute	Data Type	[Min, Max]	Range	Missing
sepal_length	Continuous	[4.3, 7.9]	3.6	0
sepal_width	Continuous	[2.0, 4.4]	2.4	0
petal_length	Continuous	[1.0, 6.9]	5.9	0
petal_width	Continuous	[0.1, 2.5]	2.4	0

Table 8.2: The distribution of example classifications in the Iris plants data set.

Classification	Example Count	Data Set Percentage
Iris_Setosa	50	33.$\dot{3}$%
Iris_Versicolor	50	33.$\dot{3}$%
Iris_Virginica	50	33.$\dot{3}$%
Total Examples	150	100.0%

Table 8.3: Data subsets for 30-fold cross-validations on the Iris plants data set.

Set of Examples	Data Examples
Entire data set	150
Test set	5
Training set	145
Examples not used in any test sets	0

8.2.2 Ionosphere Data Set

This section discusses details related to the ionosphere data set. Section 8.2.2.1 provides brief background details on the data set. Section 8.2.2.2 describes the data preparation that was performed on the data set prior to the experimental work.

8.2.2.1 Data Set Background

This data set was produced by the Space Physics Group of Johns Hopkins University in 1989. The set contains radar data collected by a system located in Goose Bay, Labrador. The system consisted of a phased array of 16 high-frequency radar antennae with a total

transmitted power to the order of 6.4 kW. The antennae targeted free electrons in the ionosphere, in order to detect any form of structure within the ionosphere.

An autocorrelation function was used to process the received signals. The arguments of the function were the time of a pulse and the pulse number. There were 17 pulse numbers for this system. A tuple corresponds to a radar return, and is described by two attributes per pulse number, resulting in 34 attributes. Each attribute corresponds to the complex values that were returned by the autocorrelation function through which the signal was processed, and are continuous values in the range $[0, 1]$. No missing attributes are present. Table 8.4 summarizes the attribute characteristics. This data was chosen due to the large number of attributes, in comparison to the other sets.

A total of 351 radar returns were recorded in the data set, each of which was classified as either `good`, or `bad`. A `good` return signifies that evidence of some type of structure in the ionosphere was shown, while a `bad` return indicates that no structure was found because the radar signals passed through the ionosphere. These classifications are encoded into an additional binary-valued, nominal classification attribute. The distribution of data set tuples between these two classes is illustrated in Table 8.5.

8.2.2.2 Data Preparation

As in all other cases reported here, 30-fold cross-validations were performed in all experiments involving the ionosphere data set. The breakdown of the examples between training sets and test sets for the cross-validation is given in Table 8.6.

For the same reasons considered in the preparation of the Iris plants data set, no missing attribute replacement, noise reduction, or nominal value encoding was performed. Similarly, the nominal classification attribute needed no further processing. Because the attribute values of this data set are all in the same range, no data normalization is required. In summary, therefore, no data cleaning was necessary for this data set.

8.2.3 The Monk's Problems Data Sets

This section provides a discussion on the three monk's problems data sets. Section 8.2.3.1 gives the most important background information relating to the data sets, and Section 8.2.3.2 lists the data preparation steps that were required for each set.

Table 8.4: The characteristics of data attributes in the ionosphere data set.

Attribute	Data Type	[Min, Max]	Range	Missing
1 to 34	Continuous	[0.0, 1.0]	1.0	0

Table 8.5: The distribution of example classifications in the ionosphere data set.

Classification	Example Count	Data Set Percentage
good	225	64.1%
bad	126	35.9%
Total Examples	351	100.0%

Table 8.6: Data subsets for 30-fold cross-validations on the ionosphere data set.

Set of Examples	Data Examples
Entire data set	351
Test set	11
Training set	340
Examples not used in any test sets	21

8.2.3.1 Data Set Background

The monk's problems [233] are artificial data sets created by Sebastian B. Thrun, Tom Mitchell and John Cheng (all from Carnegie Mellon University) for the 2nd European Summer School on Machine Learning, which was held in Belgium during the summer of 1991. The sets were used to compare several learning algorithms. The suite contains three data sets, each with the same attribute domain, but different problem characteristics. The sets were devised for several research teams, each of which advocated a particular technique, thus reducing bias in favor of any specific learning strategy.

The three monk's problems data sets are identical in basic structure. Each data example represents a hypothetical monk. The same nominal attributes are present in all three data sets, and describe six characteristics of the monks. The sets contain no missing attribute values. The relevant attribute characteristics are shown in Table 8.7.

Table 8.7: The characteristics of data attributes in the monk's problems data sets.

Attribute	Data Type	Legal Values	Missing
head_shape	Nominal	{round, square, octagon}	0
body_shape	Nominal	{round, square, octagon}	0
is_smiling	Nominal	{yes, no}	0
holding	Nominal	{sword, balloon, flag}	0
jacket_color	Nominal	{red, yellow, green, blue}	0
has_tie	Nominal	{yes, no}	0

All three data sets contain examples representing the full 432 possible attribute value combinations. Each example is classified, using a binary nominal classification attribute, as either `monk`, or `not_monk`. Each problem uses a different classification condition:

- **Problem 1:** This data set describes a condition in standard disjunctive normal form (DNF). The condition for an example to be classified as `monk` is:

  $$(\texttt{head\_shape} = \texttt{body\_shape})\ \texttt{OR}\ (\texttt{jacket\_color} = \texttt{red})$$

  No noise is added to this data set. The distribution of the examples between the two possible classes within this data set is shown in Table 8.8.

- **Problem 2:** This problem is similar to a parity problem, and is difficult to describe in simple DNF or CNF. The condition for an example to be `monk` is:

  exactly two of all six attributes have their first value

  As was the case with the first problem, the data set is noise-free. The distribution of data set examples between the two classes is tabulated in Table 8.9.

- **Problem 3:** The condition of the third problem is in DNF, and is more complex than the first problem's condition. The condition for a classification as `monk` is:

  $$(\texttt{jacket\_color} = \texttt{green}\ \texttt{AND}\ \texttt{holding} = \texttt{sword})\ \texttt{OR}$$
  $$(\texttt{jacket\_color} \neq \texttt{blue}\ \texttt{AND}\ \texttt{body\_shape} \neq \texttt{octagon})$$

  Table 8.10 shows this data set's class distribution. In addition, 5% class noise is added to the training data. The test set, obviously, remains noise-free.

Table 8.8: The distribution of example classifications in the monk's problem 1 data set.

Classification	Example Count	Data Set Percentage
monk	216	50.0%
not_monk	216	50.0%
Total Examples	432	100.0%

Table 8.9: The distribution of example classifications in the monk's problem 2 data set.

Classification	Example Count	Data Set Percentage
monk	142	32.87%
not_monk	290	67.13%
Total Examples	432	100.0%

Table 8.10: The distribution of example classifications in the monk's problem 3 data set.

Classification	Example Count	Data Set Percentage
monk	228	52.78%
not_monk	204	47.22%
Total Examples	432	100.0%

8.2.3.2 Data Preparation

As for the previous data sets, a 30-fold cross-validation was performed for all experiments involving one of the monk's problems data sets. Table 8.11 shows the division of examples between training and test sets, and is the same for all three data sets.

For the same reasons that are discussed in Sections 8.2.1.2 and 8.2.2.2, no attribute replacement or noise reduction strategies were applied to the three sets. The nominal classification attribute in each set also required no further processing.

The need for nominal attributes to be encoded into binary-valued sub-attributes is discussed in Section 2.4. Due to this requirement, binary attribute encoding was performed for all three data sets. After this encoding was completed, all attribute values varied over the range $[0.0, 1.0]$, and consequently no normalization was required.

Table 8.11: Data subsets for 30-fold cross-validations on the monk's problems data sets.

Set of Examples	Data Examples
Total data set	432
Test set	14
Training set	418
Examples not used in any test sets	12

In the case of the original third data set, a pre-composed training set is provided, which has 5% noise added. However, the cross-validation process requires training sets drawn over the entire data set. Consequently, a noise-free base data set was constructed. For each of the training sets in a 30-fold cross-validation, 5% random class noise was then introduced. This was achieved by randomly selecting 20 training examples from each clean training set of 418 data examples. The class of each chosen data example was then changed to the complement of the correct classification. To ensure comparability between simulations performed on the third monk's problem, the same randomly modified training set was used in corresponding simulations over all cross-validations.

8.2.4 Pima Indians diabetes Data Set

This section discusses the Pima Indians diabetes data set, the final data set used in this investigation. The salient details relating to the data set are given in Section 8.2.4.1, and Section 8.2.4.2 covers the cleaning steps required before experimentation.

8.2.4.1 Data Set Background

This data is a subset of a larger data set, and originated from the National Institute of Diabetes and Digestive and Kidney Diseases. The set contains only female patients who were at least 21 years old, of Pima Indian heritage, and lived near Phoenix, Arizona, in the United States of America. Eight attributes, summarized in Table 8.12, describe each example. The data set owners did not indicate any missing attribute values.

A total of 768 examples are classified by a binary nominal attribute as either `positive` or `negative`. These classes indicate that the patient tested either positive or negative

Table 8.12: The characteristics of data attributes in the Pima Indians diabetes data set.

Attribute	Data Type	[Min, Max]	Range	Missing
Number_of_times_pregnant	Continuous	[0.0, 17.0]	17.0	0
2-Hour_plasma_glucose_concentration	Continuous	[0.0, 199.0]	199.0	0
Diastolic_blood_pressure	Continuous	[0.0, 122.0]	122.0	0
Triceps_skin_fold_thickness	Continuous	[0.0, 99.0]	99.0	0
2-Hour_serum_insulin	Continuous	[0.0, 846.0]	846.0	0
Body_mass_index	Continuous	[0.0, 67.1]	67.1	0
Diabetes_pedigree_function	Continuous	[0.078, 2.42]	2.342	0
Age	Continuous	[21.0, 81.0]	60.0	0

for diabetes according to World Health Organization criteria. A test was positive when a 2-hour post-load plasma glucose measurement of 200 mg/dl, or more, was recorded at any examination. Because the classifications are nominal, as was the case for all the other data sets, no modification of the classification attributes was necessary. Table 8.13 shows the distribution of examples between the two classes. The data set was selected because it represents a real-world application with a heterogeneous attribute set.

8.2.4.2 Data Preparation

Once again, 30-fold cross-validations were performed in all experiments involving the Pima Indians diabetes data set. The breakdown of data set examples between training and test sets over every simulation is shown in Table 8.14.

No missing attribute replacement, or noise reduction strategies were employed, for the same reasons that these cleaning operations were not necessary in the previously discussed data sets. It was also unnecessary to modify the classification attributes in any way, because the tested algorithms all require nominal classes.

As in the instances of the ionosphere and Iris data sets, all the attributes in this set are continuous. No binary encoding of attribute values was therefore needed. However, because the values of the various attributes in the data set have differing ranges, attribute value normalization was necessitated. Min-max normalization was used to scale all attribute values into a $[0, 1]$ range, as in the case of the Iris plants data set.

Table 8.13: The distribution of example classifications in the Pima Indians diabetes data set.

Classification	Example Count	Data Set Percentage
`positive`	268	34.9%
`negative`	500	65.1%
Total Examples	768	100.0%

Table 8.14: Data subsets for 30-fold cross-validations on the Pima Indians diabetes data set.

Set of Examples	Data Examples
Total data set	768
Test set	25
Training set	743
Examples not used in any test sets	18

8.3 Analysis of Supervised Labeling Techniques

This section describes the details surrounding the first experimental investigation performed during this investigation, which compared the supervised labeling approaches discussed in this dissertation. Section 8.3.1 presents the overall objectives that the analysis sets out to address. Section 8.3.2 provides details on the specific implementations of the algorithms that were used for the experiments. The performance measures that were used to assess algorithmic performance are outlined in Section 8.3.3. Section 8.3.4 describes the parameter optimization procedure that was carried out for every tested algorithm, while Section 8.3.5 presents the results of the parameter optimizations. In Section 8.3.6, the relative performance characteristics of the optimized labeling algorithms are compared, while Section 8.3.7 discusses the results of the performance comparison.

8.3.1 Objectives of the Analysis

These experiments compared the three supervised labeling algorithms, namely example-centric neuron labeling, example-centric cluster labeling, and weight-centric neuron labeling. Two variants of the example-centric cluster labeling technique were included: one

using Ward clustering, the second using k-means clustering. These clustering algorithms are very commonly used for SOM neuron labeling, justifying the analysis of both.

The Davies-Bouldin index, described in Section 5.2, was used to produce an optimal number of clusters. The method described in Section 5.3.1 was used for Ward clustering, while Section 5.3.2 describes the technique used by k-means clustering.

To assess labeling algorithm performance, each algorithm was applied to a simple classification exercise. First, a SOM was trained on a training data set. The tested labeling algorithm was then applied to the converged map. During assessment, a BMU was determined for each data example in a test or training set. Each data example's BMU label was compared to the actual example classification, where a match constituted a correct classification, and a mismatch denoted a misclassification. Any data example that mapped to an unlabeled BMU was considered to be unclassified.

This classification task was performed for each experimental data set discussed in Section 8.2. Classifications were performed and assessed on both the training and test data set of each simulation in a 30-fold cross-validation. Several performance aspects, related both to the labels themselves and the classifications, were assessed. The facets of performance that were focused upon are discussed within Section 8.3.3.

The primary objective of this analysis was to scrutinize the performance differences between the investigated approaches, as measured on the tested data sets. Additionally, this study aimed to determine whether performance was detrimentally affected as a result of the potential for neurons to be left unlabeled when example-centric neuron labeling and (to a lesser extent) example-centric cluster labeling were used for classification.

8.3.2 Implementations of the Algorithms

This analysis used a SOM implementation that extends version 3.2 of SOM_PAK[1]. The original SOM_PAK software [148] was developed under the supervision of Teuvo Kohonen, and is widely used in the literature. This dissertation also considers SOM_PAK to be a reference implementation, because of Kohonen's direct involvement.

[1] The source implementation files and installation documentation for version 3.2 of the SOM_PAK software are available for free download at `http://cis.legacy.ics.tkk.fi/hynde/lvq/tars/`. This version is an unofficial release, but addresses only software stability issues present in the previous version.

Unfortunately, SOM_PAK suffers from the following limitations, which made the original software package less suitable for the reported experimental analysis:

- Data shuffling is biased, and uses a weak pseudo-random number generator.
- The only available stopping condition is an iteration limit specified by the user.
- The implemented decay functions for $\eta(t)$ and $\sigma(t)$ depend on the iteration limit.

Furthermore, because SOM_PAK version 3.2 was developed in 1997, several compatibility issues arise when compiling the source program using modern compilers.

Due to these shortcomings, the original implementation of SOM_PAK was adapted[2], to correspond with the discussion on SOMs given in Chapter 3. In addition to adapting the program code for a modern compiler, four main modifications were made:

- A Fisher-Yates shuffle [79, 141], which used the MT19937 implementation of the Mersenne Twister pseudo-random number generator [90, 166], was used to reorder the data set of training examples after the completion of each epoch.
- The hypercube-based weight initialization scheme proposed by Su *et al* has been demonstrated to produce good map training performance [231]. This technique was therefore used to provide initial weight values for all simulations.
- Two stopping conditions were used. Firstly, training ceased when the standard deviation of the training quantization error over a 30 iteration window, calculated according to Equation (3.9), dropped below a 0.0001 limit. Secondly, a training iteration cut-off of 100 000 ensured termination in a reasonable time frame.
- Exponential decay functions for $\eta(t)$ and $\sigma(t)$ were implemented according to Equation (3.11) in Section 3.5.2, and Equation (3.12) in Section 3.5.3, respectively.

The stopping conditions for stochastic training were intentionally chosen to be relatively strict. This resulted in longer training times, but ensured that every SOM had completely converged prior to the application of any neuron labeling algorithms to the maps.

[2] The modified SOM_PAK implementation, which was used throughout all of the reported experiments, is available for free download at `http://cirg.cs.up.ac.za/resources/som_pak.tar.gz`.

In every simulation, SOM_PAK was configured to use a hexagonal map lattice. The stochastic training process used the smooth Gaussian kernel neighborhood function, as described by Equation (3.6) within Section 3.3.2. These choices are common in SOM literature, and are therefore justified for use in the reported experiments. In order to prevent any *a priori* bias resulting from classification attribute values in the training data set, a fully unsupervised SOM was used in all simulations.

The three tested labeling algorithms were custom implemented for the purposes of this analysis, and were also used during the investigation into the performance of SOM-based rule extractors. The implementations of example-centric neuron labeling, example-centric cluster labeling, and weight-centric neuron labeling followed the algorithmic descriptions presented in Sections 6.2.1, 6.2.2, and 6.2.3, respectively.

As previously noted, example-centric cluster labeling was tested with both the Ward and k-means clustering algorithms. These clustering algorithms were respectively implemented according to the descriptions in Sections 5.3.1 and 5.3.2 of this dissertation.

8.3.3 Algorithmic Performance Measures

Several facets of performance were measured for each algorithm tested by means of a cross-validation on a specific experimental data set. These measures provide insight into two aspects of performance: the error rate of the various labeling techniques when performing the classification task outlined in Section 8.3.1, as well as the proportion of neurons in the map lattice that were left unlabeled by each technique. This section describes the measures recorded for each cross-validation that was performed for a labeling algorithm applied to the classification of a benchmark data set.

Three measures were used to assess the error performance of each labeling technique over a cross-validation on the training data set. Firstly, the *overall training set error* mean ($\mathcal{E}_T$) and standard deviation ($\mathcal{S}_T$) were measured as the percentage of classification errors. Two additional error rates, which are both components of the overall training set error, are reported in order to provide further insight into the training performance. Firstly, the mean ($\mathcal{E}_{TM}$) and standard deviation ($\mathcal{S}_{TM}$) of the *training set error due to misclassified data examples* were measured as the percentage of classifications that were applied, but were incorrect. Secondly, the mean ($\mathcal{E}_{TU}$) and standard deviation ($\mathcal{S}_{TU}$) of

the *training set error due to unclassified data examples* recorded the percentage of cases in which no classifications could be applied, due to unlabeled BMUs.

Three measures were also used to characterize test set error performance. Firstly, the *overall test set error* mean ($\mathcal{E}_G$) and standard deviation ($\mathcal{S}_G$), were recorded as the percentage of all classification errors. The overall test set error was sub-divided into the mean ($\mathcal{E}_{GM}$) and standard deviation ($\mathcal{S}_{GM}$) of the *test set error due to misclassified data examples*, measured as the percentage of incorrectly applied classifications, and the mean ($\mathcal{E}_{GU}$) and standard deviation ($\mathcal{S}_{GU}$) of the *test set error due to unclassified data examples*, which is the percentage of classifications that were not applied.

Test set error is generally considered to be more informative than training set error, because this measure provides insight into the performance of a technique on data that is not presented during training. Test set error therefore provides insight into how well the error performance of each of the assessed techniques generalizes.

The final performance measure that was recorded described the proportion of map neurons with no assigned labels, measured as the *percentage of unlabeled neurons*. As for the training and test error measure, both the mean ($\mathcal{E}_U$) and standard deviation ($\mathcal{S}_U$) of this measure were recorded for each algorithm executed on a particular data set.

8.3.4 Parameter Optimization Procedure

All the algorithms compared in this section have parameters that affect algorithmic performance. As suggested by the no free lunch theorem [270], no set of algorithmic parameters is optimal for all problems [58]. It was thus necessary to tune the parameters of all the investigated algorithms for each classification problem, so that near-optimal results were achieved in all cases. To ensure a fair algorithmic comparison, it was necessary to apply the same optimization procedure to all the tested algorithms.

The optimization approach applied to each algorithm and data set pair was based on a method proposed by Franken [82]. The technique was originally applied to parameter tuning for particle swarm optimization (PSO) [136, 219], but is applicable to any algorithm with tunable parameters. The method uses Sobol' sequences to generate candidate parameter configurations, and parallel coordinate plots to visualize the search space of possible parameter settings. Sections 8.3.4.1 and 8.3.4.2 discuss these two components.

8.3.4.1 Generating Candidate Parameter Configurations

A naïve parameter optimization approach is called *one-factor-at-a-time design*. The values of all parameters are fixed, except the one currently being tuned. Cross-validations are then performed for varying values of the tuned parameter, and the setting producing the most desirable cross-validated performance (in terms of one or more measures) is chosen as the optimal value. This parameter setting is then fixed, and the process is repeated for each of the remaining parameters, until all have been optimized.

One-factor-at-a-time design is essentially a greedy search of the parameter space, and thus does not take parameter dependencies into account. This is because it is impossible to adapt a previously fixed parameter in response to a newly optimized one, should a relationship exist between the two. Even if the former parameter is re-optimized, it is quite possible that additional dependencies will be affected in turn. The optimization process is also likely to be sensitive to the initial values of the parameters. For these reasons, one-factor-at-a-time design was not used during this investigation.

Factorial design is another simple approach to optimizing parameters, and requires a cross-validation for every combination of parameter values that is possible. The appropriate performance measures are then analyzed for every cross-validation, and the settings with the most desirable performance are selected as optimal. While this approach does consider the interactions between parameters, the optimization process is usually computationally intractable, due to the large number of parameter value combinations that must be compared. The time complexity increases as a larger number of parameters are optimized, and more possible settings for each parameter are investigated. This drawback made factorial design an unattractive choice for this work.

A *random sampling* of the parameter space generates a set of random points in the parameter space. While inter-parameter relationships are not ignored, a random sample simply ensures that every point in the search space has the same probability of being analyzed, and does not imply that the search space is sampled evenly. This means that it is possible for better parameter settings to be unintentionally overlooked. One way of mitigating the likelihood of this scenario is to increase the number of random samples, but this will necessarily increase the computational complexity of the optimization further. For these reasons, random sampling was also discarded for these experiments.

Sobol' sequences [225] are low-discrepancy sequences of quasi-random points in a multi-dimensional space. The quasi-random points in the sequence have similar characteristics to randomly sampled points, but are designed to fill a unit hypercube as uniformly as possible. While other low-discrepancy sequences exist, studies have been inconclusive as to whether one is clearly superior [93]. However, a very efficient technique exists to generate Sobol' sequences using Gray code binary encoding [124, 125].

The *cycle length* of a Sobol' sequence is the number of points in the sequence, where a longer cycle length gives a greater level of detail. The cycle length must be a power of two, to ensure an even coverage of the hypercube at a certain detail level.

The points in a Sobol' sequence are used to create a set of candidate parameter values, by associating each dimension of a point with a parameter value. Every dimension value, which is in the range $[0, 1]$, is then scaled to the appropriate range for the parameter represented by the dimension, using min-max normalization. For ordinal parameters, the scaled value is also rounded. Quasi-random parameter values are preferable to purely random ones, because the former sample the parameter space more uniformly.

The sampled parameter settings used for the reported experiments were produced using a Sobol' sequence generator that uses the Gray code binary encoding approach [154], which was implemented by Frances Kuo, one of the originators of the technique. Every candidate configuration was produced using the generator configuration recommended by Kuo. A cycle length of 512 was chosen for all the investigated algorithm and data set combinations. This cycle length was chosen because 512 cycles produced well-performing configurations, and the next largest cycle length (that is, 1 024 configurations) resulted in a prohibitively time complex optimization process over the conducted experiments.

8.3.4.2 Visualizing the Search Space of Parameter Settings

Once the candidate parameter configurations were generated, the performance of each had to be assessed. This was achieved by performing a cross-validation for each parameter configuration, and analyzing the performance measures of each.

The overall test data set error was of primary interest for neuron labeling technique optimization, because this measure indicates the proportion of classification errors that can be expected on data that is unseen. The overall training error is an overly optimistic

measure of classification performance, and was therefore of secondary importance during parameter optimization. The percentage of unlabeled neurons was not used, because it was unclear whether fewer unlabeled neurons affected SOM-based classification (however, Section 6.2.1 argues that fewer unlabeled neurons are desirable for EDA).

Parallel coordinate plots [121] were used as a basis for the performance analysis of the parameter space for an algorithm executed on a data set. A parallel coordinate plot is a popular visualization for many high-dimensional data records in two dimensions. While the 1980s saw an increase in the popularity of these plots, it is possible to identify early examples of these visualizations from the late nineteenth century [107].

A parallel coordinate plot consists of a series of parallel lines, where each line represents a dimension of the visualized data. The endpoints of a line represent the maximum and minimum that values for the associated dimension range between, with intermediate values appropriately scaled between the endpoints. One data record is represented as a single set of connected line segments (or a polyline), with vertices located on each vertical line. Each vertex is positioned at the location representing the data record value for the appropriate dimension. It is then possible to overlay an arbitrary number of data records on the same plot, giving an overview of the entire data set.

For all of this dissertation's parameter optimizations, parallel coordinate plots were used to represent the parameter values of the algorithm being investigated, and one or more performance measures that were being optimized for. The parallel lines of the plot were oriented vertically, with one line representing each algorithmic parameter. The upper and lower endpoints of these lines indicated the upper and lower attribute value bounds. The rightmost parallel lines were reserved for the applicable mean performance measures, where the upper and lower bounds represented the maximum and minimum mean performance measures produced over all the examined parameter configurations. For the parameter optimization of the supervised neuron labeling techniques, the mean of the overall test set error was the only performance measure represented.

Each polyline represented a candidate parameter configuration and the associated mean test set error. The results were first sorted in ascending order of the overall test set error mean, with ties settled first in favor of test set standard deviation, then mean training set error, and finally training set error standard deviation. To avoid dense and

uninterpretable visualizations, the plots only visualized the configurations responsible for the first 20% of the sorted results (that is, the best performing 20% of the configurations). Finally, the first configuration in the sorted list was deemed to be optimal.

8.3.5 Results of Parameter Optimization

This subsection presents the results of the parameter optimizations for the four tested supervised neuron labeling techniques, namely example-centric neuron labeling, example-centric cluster labeling configured with Ward clustering, example-centric cluster labeling configured with k-means clustering, and weight-centric neuron labeling. Each approach was optimized using parallel coordinate plots, for the classification task described in Section 8.3.1, run on the six experimental data sets discussed in Section 8.2.

Due to the fact that none of the supervised neuron labeling techniques have algorithmic parameters, the experiments involving these approaches only required the optimization of parameters for the underlying SOM. For each of the analyzed data sets, Table 8.15 summarizes the value ranges within which candidate parameter settings varied.

For simplicity, square maps were used throughout all experiments, meaning that one value was optimized for both Y and X. The minimum grid size was 2×2, because a map consisting of a single neuron is not meaningful, even for a non-emergent feature map. The maximum tested map dimension for each data set produced the largest grid that still contained fewer neurons than the number of training examples. This decision was based on the implied maximum number of neurons in an emergent feature map, noted in Section 3.4.3, which is equivalent to the number of training examples.

The initial kernel width of each parameter configuration was a percentage of the map grid dimension for the configuration in question, in the range $(0, 1]$. The initial kernel width was thus greater than 0.0, with a maximum value of the largest grid dimension, but never exceeded the grid width and height in any parameter configuration. The remaining parameters used the same ranges for all data sets, and were chosen to balance a reasonable execution time with an adequate coverage of the parameter space.

This chapter presents all performance and optimized parameter values with three fractional digits. Figure 8.1 shows an example of a parallel coordinate plot for the parameters of a supervised neuron labeling algorithm. The plot visualizes the parameter

Table 8.15: The ranges within which parameter values varied for supervised neuron labeling.

Parameter	Symbol	Data Type	Data Set	Range
Map dimensions	Y, X	Ordinal	Iris plants Ionosphere Monk's problems Pima Indians diabetes	[2, 12] [2, 18] [2, 20] [2, 27]
Initial learning rate	$\eta(0)$	Continuous	All	[0.0, 10.0]
Learning rate decay constant	τ_1	Continuous	All	(0.0, 1 500.0]
Kernel width	$\sigma(0)$	Continuous	All	(0.0, Y]
Kernel width decay constant	τ_2	Continuous	All	(0.0, 100.0]

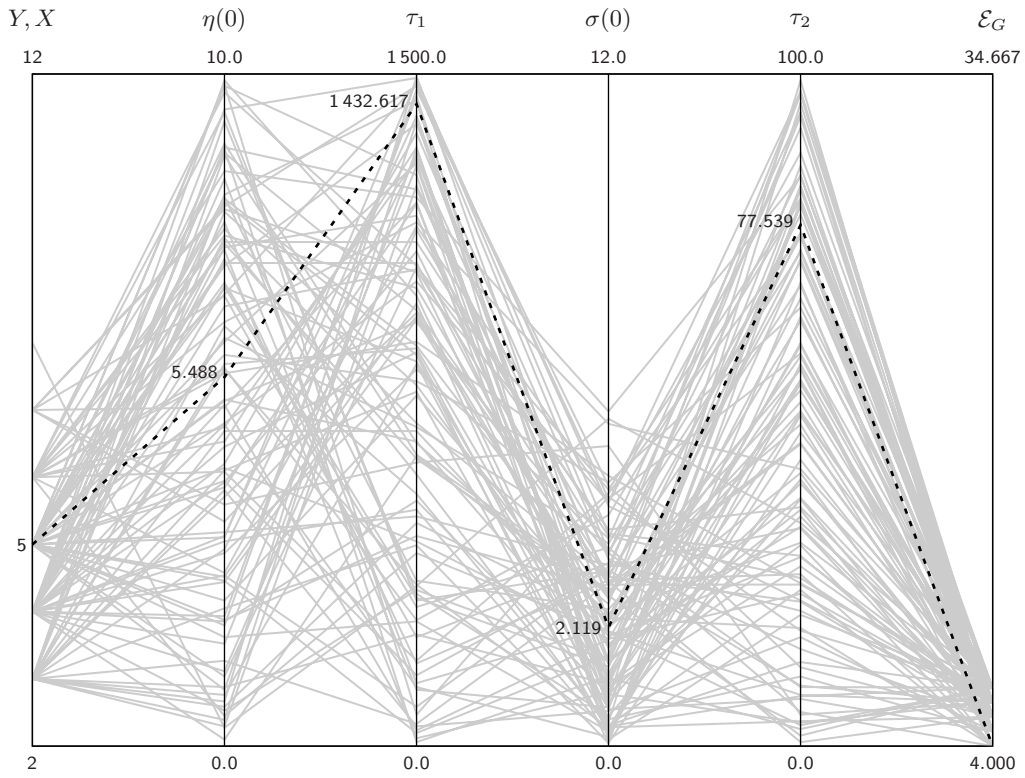


Figure 8.1: The parallel coordinate plot of the parameter value optimization performed for example-centric neuron labeling, when applied to the Iris plants experimental data set.

space for example-centric neuron labeling, when it was applied to the Iris plants data set. The polylines representing the best performing 20% of the configurations are gray, while the most optimal parameter setting is represented by a dashed line. Labels at the vertices of the dashed polyline note the respective optimal parameter values. The optimal parameter settings produced a mean overall test set error of 4.000%, while the worst mean overall test set error over all configurations was 34.667%.

Parallel coordinate plots are also useful for more detailed analysis of the parameter space. For example, Figure 8.1 illustrates that the configurations responsible for the best results generally had map dimensions in the lower half of the tested range, paired with lower initial kernel widths. The initial learning rate and both the decay constants did not show strong correlations to good performance. Furthermore, the crossed polylines between the initial learning rate and the learning rate decay constant indicate that these parameters are inversely correlated. A study involving such detailed analysis is, however, beyond the scope of this dissertation's focus, and is left to future work.

Table 8.16 shows the optimal parameter configurations for example-centric neuron labeling applied to the six experimental data sets. Tables 8.17 and 8.18 show the same information for example-centric cluster labeling, respectively configured with Ward clustering and k-means clustering. Finally, Table 8.19 represents the optimal parameter configurations for weight-centric neuron labeling. The mean of the overall test set error for each optimal parameter configuration is shown in the last row in all cases.

8.3.6 Comparison of Algorithmic Performance

This subsection presents and discusses the results of the statistical comparison performed between the analyzed supervised neuron labeling algorithms, tested on each of the experimental data sets. To facilitate this discussion, all the results related to one of the performance measures described in Section 8.3.3 are described separately. The statistical analysis procedure described in Section 8.1.2 was followed throughout.

The remainder of this section is organized as follows: Section 8.3.6.1 discusses performance related to the overall training set error. The training set error resulting from misclassified and unclassified data examples are presented in Sections 8.3.6.2 and 8.3.6.3, respectively. Section 8.3.6.4 focuses on the overall test set error, while Sections 8.3.6.5

Table 8.16: Optimal parameters for the example-centric neuron labeling algorithm.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
Y, X	5	7	14	15	11	12
$\eta(0)$	5.488	3.848	6.055	8.867	9.941	2.070
τ_1	1 432.617	1 209.961	849.609	1 365.234	577.148	1 376.953
$\sigma(0)$	2.119	1.818	10.445	1.348	3.029	9.797
τ_2	77.539	59.570	9.766	31.641	83.008	52.734
$\mathcal{E}_G$	4.000%	11.515%	20.000%	20.238%	26.429%	26.133%

Table 8.17: Optimal parameters for example-centric cluster labeling with Ward clustering.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
Y, X	9	15	16	19	16	12
$\eta(0)$	4.180	2.148	6.133	2.695	4.492	2.578
τ_1	5.859	638.672	1 330.078	1 470.703	1 271.484	1 160.156
$\sigma(0)$	8.402	2.520	8.563	0.074	10.438	8.719
τ_2	66.016	64.453	87.109	71.484	50.391	88.281
$\mathcal{E}_G$	18.000%	8.182%	27.619%	29.524%	28.333%	24.933%

Table 8.18: Optimal parameters for example-centric cluster labeling with k-means clustering.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
Y, X	4	18	19	18	17	17
$\eta(0)$	8.145	1.953	0.625	4.551	3.164	9.688
τ_1	893.555	1 253.906	1 031.250	73.242	1 119.141	1 453.125
$\sigma(0)$	2.633	7.453	3.563	4.816	16.801	2.656
τ_2	79.102	82.031	31.250	5.664	1.172	78.125
$\mathcal{E}_G$	21.333%	9.394%	33.571%	32.381%	27.619%	25.200%

Table 8.19: Optimal parameters for the weight-centric neuron labeling algorithm.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
Y, X	12	17	14	15	19	24
$\eta(0)$	4.609	0.918	6.562	8.867	9.629	3.770
τ_1	574.219	594.727	46.875	1 365.234	61.523	424.805
$\sigma(0)$	10.781	14.045	4.813	1.348	4.639	17.297
τ_2	96.094	87.305	34.375	31.641	67.383	72.852
$\mathcal{E}_G$	3.333%	15.152%	31.667%	29.762%	27.619%	29.467%

and 8.3.6.6 respectively investigate the test set error due to misclassified and unclassified data examples. Lastly, Section 8.3.6.7 analyzes the proportion of neurons in the map structure that were left unlabeled by each supervised neuron labeling technique.

8.3.6.1 Overall Training Set Error

Tables 8.20 to 8.25 represent each experimental data set, comparing the performance of every supervised neuron labeling method in terms of overall training set error. Table 8.20 shows the comparison for the Iris plans data set, Table 8.21 represents the ionosphere data set, Tables 8.22 to 8.24 respectively portray the three monk's problems data sets, and Table 8.25 provides details on the Pima Indians diabetes data set.

Each of these tables compares all the supervised neuron labeling algorithms against one another for a specific experimental data set. Each row and column of a table represents a labeling algorithm, as well as the mean and standard deviation of the overall training set errors produced by the neuron labeling algorithm in question.

The intersection of a row and column shows, for a pair of algorithms, the p-value determined when the two algorithms were compared using the two-sample Wilcoxon signed-rank test described in Section 8.1.2. The p-values were assessed at a 0.05 significance level, with a Bonferroni correction. A square appears above a p-value indicating that the pair of algorithms were not significantly different according to the overall training set error. When a p-value indicates that a significant difference was evident, an arrow points to the row or column of the algorithm with the lowest overall training set error.

Table 8.20: Statistical comparison of the overall training set error for the supervised neuron labeling methods executed on the Iris plants data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	3.655	0.705	25.057	10.178	16.989	9.022
Example-Centric Cluster (k-Means)	$\uparrow$ 1.863×10^{-9}					
$\mathcal{E}_T$ / $\mathcal{S}_T$	25.057 / 10.178					
Example-Centric Cluster (Ward)	$\uparrow$ 1.863×10^{-9}		$\leftarrow$ 6.411×10^{-3}			
$\mathcal{E}_T$ / $\mathcal{S}_T$	16.989 / 9.022					
Weight-Centric Neuron	$\leftarrow$ 4.768×10^{-6}		$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}	
$\mathcal{E}_T$ / $\mathcal{S}_T$	2.575 / 0.700					

Table 8.21: Statistical comparison of the overall training set error for the supervised neuron labeling methods executed on the ionosphere data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	10.137	1.221	7.912	1.391	8.598	1.961
Example-Centric Cluster (k-Means)	$\leftarrow$ 2.503×10^{-6}					
$\mathcal{E}_T$ / $\mathcal{S}_T$	7.912 / 1.391					
Example-Centric Cluster (Ward)	$\leftarrow$ 1.485×10^{-3}		$\square$ 0.097			
$\mathcal{E}_T$ / $\mathcal{S}_T$	8.598 / 1.961					
Weight-Centric Neuron	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}	
$\mathcal{E}_T$ / $\mathcal{S}_T$	15.216 / 1.626					

Table 8.22: Statistical comparison of the overall training set error for the supervised neuron labeling methods executed on the monk's problem 1 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	18.126	1.084	26.643	2.781	24.848	1.924
Example-Centric Cluster (k-Means)						
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}				
26.643	2.781					
Example-Centric Cluster (Ward)						
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}	$\leftarrow$ 3.147×10^{-3}			
24.848	1.924					
Weight-Centric Neuron						
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}	$\square$ 0.030		$\uparrow$ 5.748×10^{-6}	
27.974	2.416					

Table 8.23: Statistical comparison of the overall training set error for the supervised neuron labeling methods executed on the monk's problem 2 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	15.766	0.657	32.305	1.353	22.400	1.796
Example-Centric Cluster (k-Means)						
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}				
32.305	1.353					
Example-Centric Cluster (Ward)						
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}	$\leftarrow$ 1.863×10^{-9}			
22.400	1.796					
Weight-Centric Neuron						
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}	$\leftarrow$ 3.725×10^{-9}		$\square$ 0.017	
23.947	2.693					

Table 8.24: Statistical comparison of the overall training set error for the supervised neuron labeling methods executed on the monk's problem 3 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	23.828	1.792	26.619	2.278	25.000	1.460
Example-Centric Cluster (k-Means)	$\uparrow$ 1.006×10^{-5}					
$\mathcal{E}_T$ / $\mathcal{S}_T$	26.619 / 2.278					
Example-Centric Cluster (Ward)	$\square$ 0.019		$\leftarrow$ 5.657×10^{-3}			
$\mathcal{E}_T$ / $\mathcal{S}_T$	25.000 / 1.460					
Weight-Centric Neuron	$\leftarrow$ 6.333×10^{-8}		$\leftarrow$ 5.588×10^{-9}		$\leftarrow$ 1.863×10^{-9}	
$\mathcal{E}_T$ / $\mathcal{S}_T$	19.769 / 1.873					

Table 8.25: Statistical comparison of the overall training set error for the supervised neuron labeling methods executed on the Pima Indians diabetes data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	20.588	0.971	24.047	1.083	25.276	1.287
Example-Centric Cluster (k-Means)	$\uparrow$ 1.863×10^{-9}					
$\mathcal{E}_T$ / $\mathcal{S}_T$	24.047 / 1.083					
Example-Centric Cluster (Ward)	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 9.290×10^{-4}			
$\mathcal{E}_T$ / $\mathcal{S}_T$	25.276 / 1.287					
Weight-Centric Neuron	$\uparrow$ 1.304×10^{-7}		$\leftarrow$ 1.675×10^{-6}		$\leftarrow$ 3.725×10^{-9}	
$\mathcal{E}_T$ / $\mathcal{S}_T$	22.391 / 1.121					

These results indicate that example-centric neuron labeling outperformed all the other techniques on half of the data sets (the first and second monk's problems, and the Pima Indians diabetes data set), and was outperformed by only one competing algorithm in two cases (the Iris plants and third monk's problem data sets). Example-centric neuron labeling was not the worst performing technique for any of the data sets.

Weight-centric neuron labeling outperformed all the other approaches on two of the data sets (the Iris plants and third monk's problem data sets), and was outperformed by a single algorithm in two cases (the second monk's problem and the Pima Indians diabetes data set). However, the weight-centric approach performed the worst in two cases (the ionosphere data set and monk's problem 1), where example-centric cluster labeling with k-means clustering performed equally poorly in the latter case.

Example-centric cluster labeling generally underperformed. Both cluster-based approaches jointly outperformed the other two methods on only the ionosphere data set. The k-means configuration performed the worst overall in three cases (Iris plants, and monk's problems 2 and 3), and matched the worst performance of weight-centric labeling on the first monk's problem. Ward clustering performed worse than all other methods on the Pima Indians diabetes set, narrowly outperformed by k-means.

Furthermore, the example-centric neuron labeling method outperformed the weight-centric algorithm on four data sets (ionosphere, monk's problems 1 and 2, and Pima Indians diabetes), while the opposite only occurred for the Iris plants and monk's problem 3 sets. Based on this analysis, example-centric neuron labeling is deemed the superior approach over the benchmark data sets, when considering overall training error.

Focusing on only cluster labeling, Ward clustering outperformed k-means clustering in four cases (on the Iris plants data set, and all of the monk's problems), while the opposite was true for only one data set (the Pima Indians diabetes data set). The two approaches were indistinguishable in one further case (on the ionosphere data set).

8.3.6.2 Training Set Error Due to Misclassified Data Examples

To more deeply analyze the differences in error-based performance on training data, Tables 8.26 to 8.31 compare the labeling techniques in relation to the training classification error due only to misclassified examples. Table 8.26 shows the results for the Iris plants

Table 8.26: Statistical comparison of the training set error due to misclassified data examples for the supervised neuron labeling methods executed on the Iris plants data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	3.655	0.705	25.057	10.178	16.989	9.022
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$					
25.057	10.178	$\uparrow$ 1.863×10^{-9}				
Example-Centric Cluster (Ward)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$ 1.863×10^{-9}	$\leftarrow$ 6.411×10^{-3}			
16.989	9.022					
Weight-Centric Neuron						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\leftarrow$ 4.768×10^{-6}	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}	
2.575	0.700					

Table 8.27: Statistical comparison of the training set error due to misclassified data examples for the supervised neuron labeling methods executed on the ionosphere data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	10.137	1.221	7.912	1.391	8.598	1.961
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\leftarrow$ 2.503×10^{-6}				
7.912	1.391					
Example-Centric Cluster (Ward)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\leftarrow$ 1.485×10^{-3}	$\square$ 0.097			
8.598	1.961					
Weight-Centric Neuron						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$ 1.863×10^{-9}	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}	
15.216	1.626					

Table 8.28: Statistical comparison of the training set error due to misclassified data examples for the supervised neuron labeling methods executed on the monk's problem 1 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	18.126	1.084	26.643	2.781	24.848	1.924
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$				
26.643	2.781	1.863×10^{-9}				
Example-Centric Cluster (Ward)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$	$\leftarrow$			
24.848	1.924	1.863×10^{-9}	3.147×10^{-3}			
Weight-Centric Neuron						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$	$\square$		$\uparrow$	
27.974	2.416	1.863×10^{-9}	0.030		5.748×10^{-6}	

Table 8.29: Statistical comparison of the training set error due to misclassified data examples for the supervised neuron labeling methods executed on the monk's problem 2 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	15.766	0.657	32.305	1.353	22.400	1.796
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$				
32.305	1.353	1.863×10^{-9}				
Example-Centric Cluster (Ward)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$	$\leftarrow$			
22.400	1.796	1.863×10^{-9}	1.863×10^{-9}			
Weight-Centric Neuron						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$	$\leftarrow$		$\square$	
23.947	2.693	1.863×10^{-9}	3.725×10^{-9}		0.017	

Table 8.30: Statistical comparison of the training set error due to misclassified data examples for the supervised neuron labeling methods executed on the monk's problem 3 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	23.828	1.792	26.619	2.278	25.000	1.460
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$ 1.006×10^{-5}				
26.619	2.278					
Example-Centric Cluster (Ward)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\square$ 0.019	$\leftarrow$ 5.657×10^{-3}			
25.000	1.460					
Weight-Centric Neuron						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\leftarrow$ 6.333×10^{-8}	$\leftarrow$ 5.588×10^{-9}		$\leftarrow$ 1.863×10^{-9}	
19.769	1.873					

Table 8.31: Statistical comparison of the training set error due to misclassified data examples for the supervised neuron labeling methods executed on the Pima Indians diabetes data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	20.588	0.971	24.047	1.083	25.276	1.287
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$ 1.863×10^{-9}				
24.047	1.083					
Example-Centric Cluster (Ward)						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$ 1.863×10^{-9}	$\uparrow$ 9.290×10^{-4}			
25.276	1.287					
Weight-Centric Neuron						
$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\uparrow$ 1.304×10^{-7}	$\leftarrow$ 1.675×10^{-6}		$\leftarrow$ 3.725×10^{-9}	
22.391	1.121					

data set, while Table 8.27 illustrates the ionosphere data set results. Tables 8.28 to 8.30 compare performance on the three monk's problem data sets, respectively. Finally, Table 8.31 outlines the results for the Pima Indians diabetes data set.

These tables clearly show that the results for training set errors due to data example misclassifications are identical to those obtained during the analysis of the overall training set error. Because of this, all the findings from the previous section also hold for training misclassifications viewed in isolation. Example-centric neuron labeling is thus considered to be the best performing labeling approach in terms of the training set error resulting solely from misclassifications, when considering the investigated data sets.

8.3.6.3 Training Set Error Due to Unclassified Data Examples

In contrast to the previous section, Tables 8.32 to 8.37 show the compared performance for the training errors stemming from only unclassified data examples. An example was only unclassified when mapped to an unlabeled neuron. Tables 8.32 to 8.33 summarize the results for the Iris plants and ionosphere data sets, respectively. The comparisons for the three monk's problem data sets are outlined in Tables 8.34 to 8.36. Lastly, Table 8.37 lists the results that were obtained for the Pima Indians diabetes data set.

When an annotation of N/A appears at the intersection of a row and column, this indicates that no p-value could be calculated for the corresponding two algorithms. This was due to the fact that all the paired differences between the two populations of performance values were 0.0. In other words, the simulations of both algorithms performed identically. In such a case, therefore, the conclusion is obviously that no significant difference can be determined between the performance of the two algorithms.

From the tabulated statistics, it is clear that the results were identical for all of the neuron labeling algorithms on every experimental data set. Additionally, the means and standard deviations for the training set error due to unclassified data examples are 0.0 in all cases, indicating that no training data examples were left unclassified by any of the algorithms over all simulations. This is expected in the case of the weight-centric neuron labeling algorithm, because this approach guarantees a label for every neuron. However, it is interesting to note that the same behavior is exhibited by the other neuron labeling approaches, because all these methods have the potential for leaving neurons unlabeled.

Table 8.32: Statistical comparison of the training set error due to unclassified data examples for the supervised neuron labeling methods executed on the Iris plants data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000
Example-Centric Cluster (k-Means)	□ N/A					
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						
Example-Centric Cluster (Ward)	□ N/A		□ N/A			
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						
Weight-Centric Neuron	□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						

Table 8.33: Statistical comparison of the training set error due to unclassified data examples for the supervised neuron labeling methods executed on the ionosphere data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000
Example-Centric Cluster (k-Means)	□ N/A					
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						
Example-Centric Cluster (Ward)	□ N/A		□ N/A			
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						
Weight-Centric Neuron	□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						

Table 8.34: Statistical comparison of the training set error due to unclassified data examples for the supervised neuron labeling methods executed on the monk's problem 1 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000
Example-Centric Cluster (k-Means)	□ N/A					
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$	0.000 / 0.000					
Example-Centric Cluster (Ward)	□ N/A		□ N/A			
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$	0.000 / 0.000					
Weight-Centric Neuron	□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$	0.000 / 0.000					

Table 8.35: Statistical comparison of the training set error due to unclassified data examples for the supervised neuron labeling methods executed on the monk's problem 2 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000
Example-Centric Cluster (k-Means)	□ N/A					
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$	0.000 / 0.000					
Example-Centric Cluster (Ward)	□ N/A		□ N/A			
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$	0.000 / 0.000					
Weight-Centric Neuron	□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$	0.000 / 0.000					

Table 8.36: Statistical comparison of the training set error due to unclassified data examples for the supervised neuron labeling methods executed on the monk's problem 3 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000
Example-Centric Cluster (k-Means)	□ N/A					
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						
Example-Centric Cluster (Ward)	□ N/A		□ N/A			
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						
Weight-Centric Neuron	□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						

Table 8.37: Statistical comparison of the training set error due to unclassified data examples for the supervised neuron labeling methods executed on the Pima Indians diabetes data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000
Example-Centric Cluster (k-Means)	□ N/A					
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						
Example-Centric Cluster (Ward)	□ N/A		□ N/A			
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						
Weight-Centric Neuron	□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{TU}$ $\mathcal{S}_{TU}$						
0.000 0.000						

8.3.6.4 Overall Test Set Error

The attention of this experimental investigation now turns away from the analysis of the training data set, and focuses on algorithm performance for the sets of test data. While the previous sections delineate how well the algorithms model the underlying training data, this section and the two that follow indicate performance characteristics for data unseen during the training process. These results thus better indicate the real-world performance that can be expected from the labeling algorithms.

Tables 8.38 to 8.43 present the comparisons of the supervised neuron labeling techniques with respect to the overall classification error produced for the test sets across all of the experimental data sets. Table 8.38 compares the algorithms on the Iris plants data set, and Table 8.39 does the same for the ionosphere data set. Tables 8.40, 8.41, and 8.42 respectively present the results for the three monk's problems, while Table 8.43 summarizes the results for the Pima Indians diabetes experimental set.

From the results in these tables, it is clear that there was no significant statistical difference between the performance recorded for any of the algorithms when tested on two of the experimental data sets (the third monk's problem data set, and the Pima Indians diabetes data set). In the case of the ionosphere data set, nearly all the algorithms performed equally well, with only example-centric cluster labeling configured using Ward clustering clearly outperforming weight-centric neuron clustering.

It is also observable that example-centric neuron labeling outperformed all the other labeling algorithms in two of the six data sets (the first and second monk's problem data sets). In one further instance (on the Iris plants data set), this approach outperformed both the example-centric cluster labeling variants, but was not distinguishable from weight-centric neuron labeling. No other labeling algorithm performed better than example-centric neuron labeling on any of the investigated data sets.

In contrast, only the Iris plants data set saw weight-centric neuron labeling outperforming the two cluster labeling configurations, which example-centric neuron labeling also achieved. Weight-centric labeling performed equivalently to the worst performing algorithms on two of the data sets (the first and second monk's problems).

Both the example-centric cluster labeling configurations again underperformed. The approaches were statistically indistinguishable from one another for all experimental data

Table 8.38: Statistical comparison of the overall test set error for the supervised neuron labeling methods executed on the Iris plants data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	4.000	8.137	21.333	19.605	18.000	20.578
Example-Centric Cluster (k-Means)	$\uparrow$ 1.659×10^{-4}					
$\mathcal{E}_G$ / $\mathcal{S}_G$						
21.333 / 19.605						
Example-Centric Cluster (Ward)	$\uparrow$ 4.135×10^{-3}		$\square$ 0.621			
$\mathcal{E}_G$ / $\mathcal{S}_G$						
18.000 / 20.578						
Weight-Centric Neuron	$\square$ 1.000		$\leftarrow$ 3.357×10^{-4}		$\leftarrow$ 6.104×10^{-4}	
$\mathcal{E}_G$ / $\mathcal{S}_G$						
3.333 / 7.581						

Table 8.39: Statistical comparison of the overall test set error for the supervised neuron labeling methods executed on the ionosphere data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	11.515	10.923	9.394	9.982	8.182	8.390
Example-Centric Cluster (k-Means)	$\square$ 0.262					
$\mathcal{E}_G$ / $\mathcal{S}_G$						
9.394 / 9.982						
Example-Centric Cluster (Ward)	$\square$ 0.049		$\square$ 0.421			
$\mathcal{E}_G$ / $\mathcal{S}_G$						
8.182 / 8.390						
Weight-Centric Neuron	$\square$ 0.208		$\square$ 0.026		$\uparrow$ 8.202×10^{-4}	
$\mathcal{E}_G$ / $\mathcal{S}_G$						
15.152 / 12.016						

Table 8.40: Statistical comparison of the overall test set error for the supervised neuron labeling methods executed on the monk's problem 1 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	20.000	12.358	33.571	12.178	27.619	10.895
Example-Centric Cluster (k-Means)						
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 4.187×10^{-6}				
33.571	12.178					
Example-Centric Cluster (Ward)						
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 2.730×10^{-3}		$\square$ 0.048		
27.619	10.895					
Weight-Centric Neuron						
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 2.352×10^{-4}		$\square$ 0.254		$\square$ 0.158
31.667	11.817					

Table 8.41: Statistical comparison of the overall test set error for the supervised neuron labeling methods executed on the monk's problem 2 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	20.238	10.116	32.381	11.369	29.524	12.405
Example-Centric Cluster (k-Means)						
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 2.131×10^{-4}				
32.381	11.369					
Example-Centric Cluster (Ward)						
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 1.441×10^{-3}		$\square$ 0.236		
29.524	12.405					
Weight-Centric Neuron						
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 8.988×10^{-4}		$\square$ 0.087		$\square$ 0.998
29.762	9.395					

Table 8.42: Statistical comparison of the overall test set error for the supervised neuron labeling methods executed on the monk's problem 3 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	26.429	12.178	27.619	9.329	28.333	8.894
Example-Centric Cluster (k-Means)						
$\mathcal{E}_G$ / $\mathcal{S}_G$	□ 0.922					
27.619 / 9.329						
Example-Centric Cluster (Ward)						
$\mathcal{E}_G$ / $\mathcal{S}_G$	□ 0.493		□ 0.692			
28.333 / 8.894						
Weight-Centric Neuron						
$\mathcal{E}_G$ / $\mathcal{S}_G$	□ 0.955		□ 0.883		□ 0.803	
27.619 / 12.118						

Table 8.43: Statistical comparison of the overall test set error for the supervised neuron labeling methods executed on the Pima Indians diabetes data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	26.133	9.951	25.200	7.513	24.933	7.478
Example-Centric Cluster (k-Means)						
$\mathcal{E}_G$ / $\mathcal{S}_G$	□ 0.532					
25.200 / 7.513						
Example-Centric Cluster (Ward)						
$\mathcal{E}_G$ / $\mathcal{S}_G$	□ 0.559		□ 0.937			
24.933 / 7.478						
Weight-Centric Neuron						
$\mathcal{E}_G$ / $\mathcal{S}_G$	□ 0.060		□ 0.026		□ 0.016	
29.467 / 8.565						

sets. The cluster labeling approaches were the worst performing techniques in half of the investigated data sets (Iris plants, and the first two monk's problems), and showed no significant differences from the other methods in the remaining three.

Finally, example-centric neuron labeling outperformed its weight-centric counterpart in two of the six cases (on the first and second monk's problem data sets). Example-centric neuron labeling performed as well as the weight-centric approach on the remaining four investigated data sets. To summarize the outcome of the analysis, this investigation again points to example-centric neuron labeling as the superior approach within the context of overall test set accuracy performance on the analyzed benchmark data sets.

8.3.6.5 Test Set Error Due to Misclassified Data Examples

Once again, a finer-grained analysis on the test data set error performance was desired. As in Section 8.3.6.2, only the classification errors due to incorrect data classification were considered in Tables 8.44 to 8.49, but the test set was focused on instead. Table 8.44 summarizes performance on the Iris plants data set. Table 8.45 gives the ionosphere data set results, while Tables 8.46 to 8.48 investigate the monk's problems. Table 8.49 summarizes algorithm performance for the Pima Indians diabetes data set.

While the p-values for the Wilcoxon signed ranks tests on the misclassification-based test set error differ from those observed for the overall test set error, the final conclusions of the analyses for the two measures are identical. As a result, the conclusions drawn from the overall test set error comparisons presented in the previous section also hold for the errors stemming from only misclassified test set data examples.

Looking at the results holistically, the example-centric neuron labeling algorithm was superior to the other approaches when considering the test set error arising from only misclassified data examples on the investigated experimental data sets. The two variants of the supervised example-centric cluster labeling algorithm both always performed equally poorly when any performance differences between techniques could be discerned.

8.3.6.6 Test Set Error Due to Unclassified Data Examples

As opposed to the analysis of the erroneous test set classifications that the previous section concentrated on, this part of the discussion focuses on test set errors caused only

Table 8.44: Statistical comparison of the test set error due to misclassified data examples for the supervised neuron labeling methods executed on the Iris plants data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	3.333	7.581	21.333	19.605	18.000	20.578
Example-Centric Cluster (k-Means)	↑ 4.578×10^{-5}					
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	21.333 / 19.605					
Example-Centric Cluster (Ward)	↑ 1.617×10^{-3}		□ 0.621			
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	18.000 / 20.578					
Weight-Centric Neuron	□ 1.000		← 3.357×10^{-4}		← 6.104×10^{-4}	
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	3.333 / 7.581					

Table 8.45: Statistical comparison of the test set error due to misclassified data examples for the supervised neuron labeling methods executed on the ionosphere data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	11.212	10.319	9.394	9.982	8.182	8.390
Example-Centric Cluster (k-Means)	□ 0.299					
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	9.394 / 9.982					
Example-Centric Cluster (Ward)	□ 0.056		□ 0.421			
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	8.182 / 8.390					
Weight-Centric Neuron	□ 0.143		□ 0.026		↑ 8.202×10^{-4}	
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	15.152 / 12.016					

Table 8.46: Statistical comparison of the test set error due to misclassified data examples for the supervised neuron labeling methods executed on the monk's problem 1 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	18.810	12.226	33.095	12.369	27.381	10.951
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$				
33.095	12.369	8.665×10^{-6}				
Example-Centric Cluster (Ward)						
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$		$\square$		
27.381	10.951	5.854×10^{-4}		0.061		
Weight-Centric Neuron						
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$		$\square$		$\square$
31.667	11.817	8.643×10^{-5}		0.314		0.155

Table 8.47: Statistical comparison of the test set error due to misclassified data examples for the supervised neuron labeling methods executed on the monk's problem 2 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	19.524	9.177	32.381	11.369	29.048	12.147
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$				
32.381	11.369	5.356×10^{-5}				
Example-Centric Cluster (Ward)						
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$		$\square$		
29.048	12.147	4.749×10^{-4}		0.182		
Weight-Centric Neuron						
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$		$\square$		$\square$
29.762	9.395	1.184×10^{-4}		0.087		0.890

Table 8.48: Statistical comparison of the test set error due to misclassified data examples for the supervised neuron labeling methods executed on the monk's problem 3 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	25.952	12.082	27.381	9.205	28.333	8.894
Example-Centric Cluster (k-Means)	□ 0.701					
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	27.381 / 9.205					
Example-Centric Cluster (Ward)	□ 0.357		□ 0.659			
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	28.333 / 8.894					
Weight-Centric Neuron	□ 0.811		□ 0.827		□ 0.803	
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	27.619 / 12.118					

Table 8.49: Statistical comparison of the test set error due to misclassified data examples for the supervised neuron labeling methods executed on the Pima Indians diabetes data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	25.467	9.937	25.067	7.570	24.933	7.478
Example-Centric Cluster (k-Means)	□ 0.770					
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	25.067 / 7.570					
Example-Centric Cluster (Ward)	□ 0.904		□ 0.859			
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	24.933 / 7.478					
Weight-Centric Neuron	□ 0.030		□ 0.025		□ 0.016	
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	29.467 / 8.565					

by data examples that remained unclassified due to map neurons without associated labels. Tables 8.50 to 8.55 compare the performance of the respective neuron labeling approaches in terms of this error measure. Table 8.50 presents the algorithmic comparison for the Iris plants data set. The comparative algorithmic performance for the ionosphere data set is outlined in Table 8.51. The three monk's problem data sets are compared in Tables 8.52, 8.53, and 8.54, respectively. Lastly, Table 8.55 includes the statistics related to the Pima Indians diabetes experimental data set.

It is clear that all the neuron labeling algorithms produced comparable performance when it came to the error due to unclassified data examples on the test data sets. It should, however, be noted that p-values could be computed for most pairs of compared algorithms, as opposed to the analysis on training set errors attributed to unclassified examples presented in Section 8.3.6.3, which concluded that every algorithm's results were identical. When a p-value is present for a pair of algorithms, this indicates that at least one of the approaches produced some unclassified examples from the test data. It must be noted, however, that the number of unclassified data examples in all cases were negligible, as evidenced by the very low performance measure means and standard deviations observed for all labeling algorithms throughout the tables.

In contrast, p-values were not computable only when two approaches both produced no unclassified test set examples. This situation arose between the weight-centric labeling method and the k-means based cluster labeling configuration on one occasion (for the second monk's problem), and between the weight-centric approach and the cluster labeling method based on Ward clustering in two instances (on the third monk's problem and the Pima Indians diabetes data sets). For a further two data sets (namely, the Iris plants and ionosphere sets) both example-centric cluster labeling approaches and the weight-centric neuron labeling method left no test set examples unclassified.

8.3.6.7 Percentage of Unlabeled Neurons

The percentage of unlabeled neurons produced per map is the final performance measure to be considered in the investigation of the supervised neuron labeling algorithms. The results reported in the previous section indicate that unlabeled map neurons did not have a very detrimental effect on the performance of a SOM used for automatic classi-

Table 8.50: Statistical comparison of the test set error due to unclassified data examples for the supervised neuron labeling methods executed on the Iris plants data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.667	3.651	0.000	0.000	0.000	0.000
Example-Centric Cluster (k-Means)	□					
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	1.000					
0.000 / 0.000						
Example-Centric Cluster (Ward)	□		□			
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	1.000		N/A			
0.000 / 0.000						
Weight-Centric Neuron	□		□		□	
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	1.000		N/A		N/A	
0.000 / 0.000						

Table 8.51: Statistical comparison of the test set error due to unclassified data examples for the supervised neuron labeling methods executed on the ionosphere data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.303	1.660	0.000	0.000	0.000	0.000
Example-Centric Cluster (k-Means)	□					
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	1.000					
0.000 / 0.000						
Example-Centric Cluster (Ward)	□		□			
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	1.000		N/A			
0.000 / 0.000						
Weight-Centric Neuron	□		□		□	
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	1.000		N/A		N/A	
0.000 / 0.000						

Table 8.52: Statistical comparison of the test set error due to unclassified data examples for the supervised neuron labeling methods executed on the monk's problem 1 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	1.190	3.294	0.476	2.608	0.238	1.304
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$					
0.476	2.608	□ 0.375				
Example-Centric Cluster (Ward)						
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ 0.312		□ 1.000		
0.238	1.304					
Weight-Centric Neuron						
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ 0.125		□ 1.000		□ 1.000
0.000	0.000					

Table 8.53: Statistical comparison of the test set error due to unclassified data examples for the supervised neuron labeling methods executed on the monk's problem 2 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.714	2.179	0.000	0.000	0.476	1.812
Example-Centric Cluster (k-Means)						
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$					
0.000	0.000	□ 0.250				
Example-Centric Cluster (Ward)						
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ 1.000		□ 0.500		
0.476	1.812					
Weight-Centric Neuron						
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ 0.250		□ N/A		□ 0.500
0.000	0.000					

Table 8.54: Statistical comparison of the test set error due to unclassified data examples for the supervised neuron labeling methods executed on the monk's problem 3 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.476	1.812	0.238	1.304	0.000	0.000
Example-Centric Cluster (k-Means)	□ 1.000					
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	0.238 / 1.304					
Example-Centric Cluster (Ward)	□ 0.500		□ 1.000			
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	0.000 / 0.000					
Weight-Centric Neuron	□ 0.500		□ 1.000		□ N/A	
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	0.000 / 0.000					

Table 8.55: Statistical comparison of the test set error due to unclassified data examples for the supervised neuron labeling methods executed on the Pima Indians diabetes data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.667	1.516	0.133	0.730	0.000	0.000
Example-Centric Cluster (k-Means)	□ 0.125					
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	0.133 / 0.730					
Example-Centric Cluster (Ward)	□ 0.062		□ 1.000			
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	0.000 / 0.000					
Weight-Centric Neuron	□ 0.062		□ 1.000		□ N/A	
$\mathcal{E}_{GU}$ / $\mathcal{S}_{GU}$	0.000 / 0.000					

fication tasks on the data sets that were investigated. However, these observations do not present a complete picture. While unlabeled neurons were not observed to present a large problem for the algorithmic SOM analysis reported within this investigation, the exploratory interpretation of maps by humans often suffers greatly as the number of unlabeled neurons increases, producing a less consistent representation.

Tables 8.56 to 8.61 compare the performance of each neuron labeling approach in terms of the percentage of map neurons that were left unlabeled. The Iris plants data set is investigated in Table 8.56, while Table 8.57 focuses on the ionosphere data set. Tables 8.58, 8.59, and 8.60 respectively represent the three monk's problem experimental data sets, and Table 8.61 represents the Pima Indians diabetes data set.

Example-centric neuron labeling was outperformed by all the other algorithms on every data set, except for Iris plants (where example-centric neuron and Ward-based cluster labeling jointly performed the worst). The mean percentage was relatively low for the ionosphere and Pima Indians diabetes data sets (3.810% and 3.264%, respectively). However, the labeling approach produced substantially worse mean performance on the remaining data sets, ranging from 19.467% for Iris plants, to 49.807% on the second monk's problem. These percentages were much higher than those observed for the other methods, and represent a major potential challenge to human data analysts.

At the other extreme, weight-centric neuron labeling outperformed all the other methods, except on the Iris and Pima Indians diabetes sets. In the former case, weight-centric labeling matched the performance of example-centric cluster labeling with k-means, while outperforming the other two methods. In the latter, weight-centric labeling was equivalent to Ward-based cluster labeling, while outperforming the other methods.

Cluster labeling performed better than example-centric neuron labeling and worse than weight-centric labeling, except on the Iris data. The k-means method outperformed Ward clustering on half of the data sets (Iris, and monk's problems 1 and 3), the opposite occurred for the Pima Indians diabetes set, and all other sets showed no difference.

8.3.7 Discussion

Taking into consideration the analyses of all the performance measures discussed previously, a number of general conclusions are now offered in relation to the holistic per-

Table 8.56: Statistical comparison of the percentage of unlabeled neurons for the supervised neuron labeling methods executed on the Iris plants data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$
	19.467	3.104	0.000	0.000	11.481	16.308
Example-Centric Cluster (k-Means)	$\leftarrow$ 1.863×10^{-9}					
$\mathcal{E}_U$ / $\mathcal{S}_U$	0.000 / 0.000					
Example-Centric Cluster (Ward)	$\square$ 0.011		$\uparrow$ 6.104×10^{-5}			
$\mathcal{E}_U$ / $\mathcal{S}_U$	11.481 / 16.308					
Weight-Centric Neuron	$\leftarrow$ 1.863×10^{-9}		$\square$ N/A		$\leftarrow$ 6.104×10^{-5}	
$\mathcal{E}_U$ / $\mathcal{S}_U$	0.000 / 0.000					

Table 8.57: Statistical comparison of the percentage of unlabeled neurons for the supervised neuron labeling methods executed on the ionosphere data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$
	3.810	1.912	1.883	1.558	1.526	1.452
Example-Centric Cluster (k-Means)	$\leftarrow$ 7.228×10^{-5}					
$\mathcal{E}_U$ / $\mathcal{S}_U$	1.883 / 1.558					
Example-Centric Cluster (Ward)	$\leftarrow$ 4.208×10^{-6}		$\square$ 0.431			
$\mathcal{E}_U$ / $\mathcal{S}_U$	1.526 / 1.452					
Weight-Centric Neuron	$\leftarrow$ 3.725×10^{-9}		$\leftarrow$ 5.960×10^{-8}		$\leftarrow$ 9.537×10^{-7}	
$\mathcal{E}_U$ / $\mathcal{S}_U$	0.000 / 0.000					

Table 8.58: Statistical comparison of the percentage of unlabeled neurons for the supervised neuron labeling methods executed on the monk's problem 1 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$
	43.469	3.065	1.607	1.240	7.018	3.141
Example-Centric Cluster (k-Means)						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}				
1.607	1.240					
Example-Centric Cluster (Ward)						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}		$\uparrow$ 1.304×10^{-8}		
7.018	3.141					
Weight-Centric Neuron						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 5.960×10^{-8}		$\leftarrow$ 1.863×10^{-9}
0.000	0.000					

Table 8.59: Statistical comparison of the percentage of unlabeled neurons for the supervised neuron labeling methods executed on the monk's problem 2 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$
	49.807	1.885	6.101	4.077	6.944	3.522
Example-Centric Cluster (k-Means)						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}				
6.101	4.077					
Example-Centric Cluster (Ward)						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}		$\square$ 0.499		
6.944	3.522					
Weight-Centric Neuron						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 2.384×10^{-7}		$\leftarrow$ 1.863×10^{-9}
0.000	0.000					

Table 8.60: Statistical comparison of the percentage of unlabeled neurons for the supervised neuron labeling methods executed on the monk's problem 3 data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$
	32.700	4.967	2.134	1.767	6.771	3.392
Example-Centric Cluster (k-Means)						
$\mathcal{E}_U$	$\mathcal{S}_U$					
2.134	1.767	$\leftarrow$ 1.863×10^{-9}				
Example-Centric Cluster (Ward)						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}		$\uparrow$ 1.416×10^{-6}		
6.771	3.392					
Weight-Centric Neuron						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 2.980×10^{-8}		$\leftarrow$ 1.863×10^{-9}
0.000	0.000					

Table 8.61: Statistical comparison of the percentage of unlabeled neurons for the supervised neuron labeling methods executed on the Pima Indians diabetes data set.

	Example-Centric Neuron		**Example-Centric Cluster (k-Means)**		**Example-Centric Cluster (Ward)**	
	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$	$\mathcal{E}_U$	$\mathcal{S}_U$
	3.264	1.317	0.242	0.447	0.208	0.552
Example-Centric Cluster (k-Means)						
$\mathcal{E}_U$	$\mathcal{S}_U$					
0.242	0.447	$\leftarrow$ 1.863×10^{-9}				
Example-Centric Cluster (Ward)						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}		$\square$ 0.670		
0.208	0.552					
Weight-Centric Neuron						
$\mathcal{E}_U$	$\mathcal{S}_U$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 7.812×10^{-3}		$\square$ 0.125
0.000	0.000					

formance of each of the supervised neuron labeling algorithms. These observations were used to select the neuron labeling approach used by the SOM-based data mining algorithms analyzed in the following section. While it is possible to use the HybridSOM framework in conjunction with any of the labeling methods, only a method based on a cluster discovery algorithm is appropriate for the SIG* algorithm.

For this investigation, example-centric neuron labeling was usually superior to the other labeling approaches in terms of both training and test set error measures. This performance advantage was, however, less overwhelming in the case of the test set analysis. Interestingly, this classification performance advantage was observed despite the example-centric neuron labeling algorithm very clearly introducing the highest proportion of unlabeled neurons over almost all the experimental analyses.

It has been previously noted that weight-centric neuron labeling will tend to apply less accurate labels to neurons that do not map closely to any labeling data examples. Weight-centric labeling also applies labels based on only a single data example, while example-centric neuron labeling often uses several data examples to decide a neuron label. Furthermore, unlabeled neurons produced by example-centric neuron labeling are typically interpolating neurons, which usually do not attract any mapped data examples, and are thus less likely to be involved in example classification. These factors are proposed as contributors to the phenomenon of example-centric neuron labeling generally outperforming weight-centric labeling on the tested experimental data sets.

The tested example-centric cluster labeling configurations performed generally poorly throughout the experiments that were conducted, in terms of both training and test set classification error. These methods did leave some unlabeled neurons over the experiments, but certainly far fewer than example-centric neuron labeling did.

Example-centric cluster labeling results in large groups of neurons that are labeled uniformly. These uniform areas tend to mask much of the finer-grained detail found in the map, and are thus likely to result in mislabeling. This effect is proposed as a factor contributing to the poor training and test classification performance of the example-centric cluster labeling methods, which was observed during the experiments.

These observations suggest that example-centric neuron labeling should be preferred when classifying data examples using a SOM. This advantage is likely to be carried over

to any algorithm relying on a map labeled by example-centric neuron labeling. This is because a labeling with a superior classification ability models the underlying data more accurately, and an algorithm using this model will benefit by extension.

The percentage of neurons left unlabeled by example-centric neuron labeling is concerning when maps are to be analyzed by human experts. Use of example-centric neuron labeling for EDA is therefore strongly discouraged by this research work. Instead, the use of weight-centric neuron labeling is suggested, because of the fully labeled map that the algorithm guarantees, and the poor performance of the cluster labeling approach that was observed over this experimental investigation. Example-centric cluster labeling should only be used when a higher-level, less detailed visualization is required.

Finally, the relative performance of the two example-centric cluster labeling configurations must be considered. Ward clustering and k-means clustering produced very similar results throughout the experiments. The Ward clustering technique generally outperformed the k-means method in terms of training error, but no significant difference was evident between the configurations when test set error was concerned. The k-means configuration was found to outperform the Ward technique when the percentage of unlabeled neurons were considered for the experiments that were conducted.

Based on the very similar performance of the example-centric cluster labeling configurations, it is not possible to offer definitive practical guidelines as to the use of any specific clustering algorithm. Future research will compare a wider variety of clustering techniques, in the hope of better understanding the relevant performance characteristics. The SIG* algorithm implementation used during the experimentation reported in Section 8.4 was configured with Ward clustering, based simply on the training error advantage observed during this investigation, and the fact that unlabeled neurons were not observed to have a strong detrimental effect on classification performance.

8.4 Analysis of SOM-Based Data Mining Techniques

The focus of this section is on the second experimental analysis performed for this dissertation, which investigated the relative performance of the SOM-based data mining approaches. The SOM-based techniques were compared to one another, as well as two

classical rule extraction algorithms, in order to give a detailed picture of the algorithmic performance characteristics. Section 8.4.1 outlines the broad objectives of this investigation. Section 8.4.2 discusses the implementation of the investigated algorithms, while Section 8.4.3 describes the measures used to characterize the various performance aspects of the investigated algorithms. The procedure followed by the optimization of algorithmic parameters is detailed in Section 8.4.4, while Section 8.4.5 presents the actual outcomes of the parameter optimization. Finally, the statistical comparison of the investigated algorithms is discussed in Section 8.4.6, and Section 8.4.7 draws final conclusions.

8.4.1 Objectives of the Analysis

This experimental investigation analyzed the performance characteristics of the HybridSOM framework, described in Section 7.4, and the SIG* algorithm, discussed in Section 7.3. Both these algorithms are unsupervised, in the sense that neither requires prior knowledge about the class structure of the underlying training data. The boundary-based rule extraction algorithm, which is outlined in Section 7.2, is excluded from this analysis because the approach requires prior knowledge of the number of classes in the training data, and is thus supervised in nature. The comparison of supervised and unsupervised approaches is not fair, because the former class of techniques incorporates *a priori* knowledge, which the latter category does not have access to.

The SOM-based approaches were also compared against two classical rule extraction algorithms used for data mining, namely CN2 [38] and C4.5 [193]. Both CN2 and C4.5 are well known and widely used in the literature, and thus sensible choices for this analysis. The CN2 algorithm is described in Appendix A, while Appendix B provides details on the C4.5 algorithm. Additionally, the HybridSOM framework must be configured with a rule extraction algorithm. For this investigation, two such HybridSOM configurations were tested, which gave broader insight into the performance of the framework. The CN2 and C4.5 algorithms were again used as the rule extractors in the two versions of HybridSOM. This decision allowed for the identification of any performance degradation or improvement that could be ascribed only to the framework itself.

The outcomes of the analysis performed on the neuron labeling algorithms were used to inform the decision as to which labeling schemes were used during this data mining

algorithm analysis. Example-centric neuron labeling was thus used by the HybridSOM framework. The SIG* algorithm uses labeled clusters, and example-centric cluster labeling configured with Ward clustering was used for that purpose. The labeling algorithm implementations were identical to the ones used for the previous analysis.

A classification task, similar to the one used in the previous analysis, was employed to analyze the data mining algorithms. Each approach was trained in order to generate a rule set. In the case of the SOM-based algorithms, neuron labeling was performed using the training data set. When the algorithm was assessed for accuracy, the generated rule set was used to classify training or test set data examples. Correct classifications occurred when the class of the data example matched the class predicted by the rule set, while misclassifications happened when there was a mismatch. If an approach failed to predict a class, the data example was considered to be unclassified.

The data mining algorithms were applied to the same benchmark data sets that were used when analyzing the neuron labeling approaches. The performance of each rule extraction algorithm was assessed over a 30-fold cross-validation, in the same way as the investigation of the labeling algorithms did. The only differences to the performance assessment lay in the nature of the measures used to assess each cross-validation. Each of the measures that were used is described in detail within Section 8.4.3.

The most important objective of the analysis was to determine the general performance characteristics for the assessed rule extraction algorithms when tested on the investigated experimental data sets. These characteristics were to be used to compare the data mining approaches. The investigation also aimed to assess any performance advantages or disadvantages introduced specifically by the HybridSOM framework, after discarding the effects of the rule extractors with which the framework was configured.

8.4.2 Implementations of the Algorithms

The implementation details for the SOM and rule extraction algorithms were exactly the same as those used in the labeling algorithm experiments. The relevant details of these algorithmic implementations are described in Section 8.3.2.

Both CN2 and C4.5 have standard implementations, outlined in Appendices A and B, respectively. While other implementations exist, the standard versions were developed

by the originators of the approaches. These implementations were thus considered to be authoritative, and were not modified. The CN2 method was set up to build ordered rule sets, thus allowing a fair comparison to the ordered rules produced by C4.5.

The SIG* algorithm used during these experiments had to be custom developed, because no standard implementation of the algorithm is available. The algorithm developed for this experimental analysis precisely followed the description provided in Section 7.3. This implementation used two significance measures to populate the significance matrices of the algorithm, both suggested for unsupervised example-based cluster labeling in Section 6.3.4.2. The K-S measure [5, 6], discussed in Section 6.3.3.2, was used for data sets with only continuous attributes (i.e., the Iris plants, ionosphere, and Pima Indians diabetes data sets). For data sets with only nominal attributes (i.e., all the monk's problem data sets), Pearson's chi-squared statistic [180] was used instead.

It is critical to remember that the rules produced by SIG* were converted into standard production rule sets according to the procedure outlined in Section 7.3.4. This conversion was performed in order to facilitate sensible comparisons of SIG* rules to the rule sets that were produced by the other rule extraction algorithms.

It should be noted that the original description of the SIG* algorithm includes no provision for a default rule, and no such feature was added to this work's implementation. It was thus possible for the approach to leave data examples unclassified.

Finally, the original SIG* literature does not describe how the order of rules is determined. Because rule order is essential to production rule set evaluation, an ordering scheme had to be introduced. The approach used here sorted the rules into an ascending sequence, ordered by *false positives*. A false positive occurs when a data example is matched by a rule, but does not belong to the class predicted by that rule.

The HybridSOM framework was implemented according to the description given in Section 7.4. The implementation of the framework simply wrapped the previously discussed standard implementations of the CN2 and C4.5 rule extraction algorithms.

8.4.3 Algorithmic Performance Measures

The SOM-based data mining algorithm analysis shared most of the performance measures used to analyze the labeling algorithms, which Section 8.3.3 describes. These mea-

sures are the mean ($\mathcal{E}_T$) and standard deviation ($\mathcal{S}_T$) of the *overall training set error*, the mean ($\mathcal{E}_{TM}$) and standard deviation ($\mathcal{S}_{TM}$) of the *training set error due to misclassified data examples*, the mean ($\mathcal{E}_{TU}$) and standard deviation ($\mathcal{S}_{TU}$) of the *training set error due to unclassified data examples*, the mean ($\mathcal{E}_G$) and standard deviation ($\mathcal{S}_G$) of the *overall test set error*, the mean ($\mathcal{E}_{GM}$) and standard deviation ($\mathcal{S}_{GM}$) of the *test set error due to misclassified data examples*, and the mean ($\mathcal{E}_{GU}$) and standard deviation ($\mathcal{S}_{GU}$) of the *test set error due to unclassified data examples*. The mean and standard deviation of the percentage of unlabeled neurons were not used during this analysis, because the final rule sets were of interest, as opposed to the structure of the SOM itself.

An additional three measures were introduced, all of which assessed different aspects of the rule set complexity. The complexity of a rule set is a factor of both the size of the rule set, in terms of the number of rules making up the rule set, and the size of the rules themselves, described using the number of conditions making up each rule.

The first new measure is the *total number of conditions per rule set*, which represents the overall rule set complexity. The mean ($\mathcal{E}_{FT}$) and the standard deviation ($\mathcal{S}_{FT}$) of the measure, each computed over a cross-validated experiment, were recorded. A default rule was counted as a single condition (one that matches any data example), in order to differentiate the methods that produce default rules from those that do not.

The second measure analyzed the complexity of only the rule set by tallying the *number of rules per rule set*. Once again, this measure was represented by a mean ($\mathcal{E}_R$) and standard deviation ($\mathcal{S}_R$) calculated over a cross-validation. The rule count measure considered a default rule, in the event that one was present, to be a single rule.

Finally, the *average number of conditions per rule* were recorded using the mean ($\mathcal{E}_{FA}$) and standard deviation ($\mathcal{S}_{FA}$) for a cross-validated experiment. In a similar fashion to the previous two measures, a default rule was seen as a rule with one condition.

In general, excessive rule set complexity is undesirable, because complex rule sets are difficult to understand and manage. There is, however, typically a tradeoff between error performance and rule set complexity [119]. Rule set complexity must usually increase to improve accuracy, while simpler rule sets are normally less accurate. A balance between these two aspects of performance must therefore be found. Of course, this trend does not always hold. Simple rule sets are sometimes highly accurate, and redundancy introduced

into a complex rule set will not improve accuracy. It is even possible for very complex rule sets to reduce accuracy, due to the introduction of noise into the rule model.

8.4.4 Parameter Optimization Procedure

Once again, all the algorithms compared in this analysis have parameters that needed to be optimized to ensure near-optimal results. The optimization procedure followed for each algorithm was the same. The optimization procedure for this study was, however, somewhat more complex, due to the differing nature of the parameters between the approaches, the inclusion of SOM parameters that required optimization for all the SOM-based data mining methods, and the need to consider rule set complexity.

The method based on Franken's work, which uses Sobol' sequences to generate potential parameter configurations and parallel coordinate plots to visualize the parameter space, was again applied for the reported experiments. The procedure was, however, complicated by the fact that several of the algorithmic parameters are not continuous. Section 8.4.4.1 discusses the procedure that was followed to generate the candidate parameter configurations, while Section 8.4.4.2 covers the parameter space visualization.

8.4.4.1 Generating Candidate Parameter Configurations

The parameter tuning of the data mining algorithms used the same Sobol' sequence that was used for the labeling algorithm optimization. Dimension values in the Sobol' sequence were scaled as previously described for continuous and ordinal parameters. The same cycle length of 512 was used for all algorithms optimized for this analysis.

CN2 and C4.5 each have one binary parameter, respectively describing the error estimate used by CN2, and the test heuristic of C4.5. The Sobol' sequence dimensions correlating to these parameters were rounded to the nearest integer. A value of 0.0 denoted the first parameter setting, while a 1.0 value represented the second.

C4.5 builds a decision tree, which is converted into a rule set. One C4.5 parameter configures, for each generated decision tree node, the minimum number of training set examples that must be classified by at least two outcomes of the node. Because values larger than the available number of training examples are meaningless, these experiments capped the parameter setting at the training set size for the data set being used.

In C4.5, a pessimistic error rate guides the pruning of redundant rule set conditions. Fisher's exact test, with a continuous confidence level in the range $[0, 100]$, is optionally also used during this pruning. For this parameter, the appropriate dimension of the Sobol' sequence was scaled to a $[-100, 100]$ range, using min-max normalization. If the scaled value fell within the $[0, 100]$ range, the parameter was used with the scaled value as the setting. If a negative scaled value was produced, the parameter was not used. This ensured that the parameter was unused in approximately half the candidate parameter settings, while the parameter value range was adequately sampled in the remaining cases.

8.4.4.2 Visualizing the Search Space of Parameter Settings

Once again, the parameter space visualization and optimization procedure required a cross-validation to be performed for each parameter configuration. The previously discussed tradeoff between accuracy and complexity had to be taken into account during optimization. For this reason, overall test set error was used as the primary indicator of configuration quality, followed by the total number of rule set conditions. The overall training set error was of the least concern during parameter optimization.

Parallel coordinate plots again visualized the parameter space for each algorithm applied to a specific data set. Algorithmic parameters were visualized to the left of each plot. The rightmost two parallel plot lines respectively represented the configuration's means of the overall test set accuracy and the total conditions per rule set.

The optimization procedure first sorted all the evaluated parameter configurations into ascending order, based on the mean test set error. Ties were settled according to the test error standard deviation, followed by the mean total rule set condition count, the standard deviation of the condition count, the mean training error, and finally the standard deviation of the training error. The most optimal 20% of parameter configurations in this list were selected from this sorted list to form a near-optimal sublist.

All the pairs of unique total condition count mean and standard deviation values in the near-optimal sublist were recorded. The pairs were arranged in ascending order of condition count means, with the standard deviation breaking ties. The first 20% of the pairs formed a set of optimal condition counts. The optimal set of configurations were then considered to be the near-optimal parameter configurations with total condition

count means and standard deviations that appeared in the set of optimal condition counts. Thus the near-optimal parameter settings were pruned of configurations with sub-optimal rule set complexity, to form the optimal parameter configuration set. The first configuration in the optimal parameter set was judged most optimal.

In some cases this parameter optimization procedure selects a configuration with relatively high mean test error. In such cases, it is typically the case that configurations with better test error performance produce overly large rule sets, which was undesirable.

8.4.5 Results of Parameter Optimization

This section presents the parameter optimizations of the DM methods applied to the data sets listed in Section 8.2 for the classification task described in Section 8.4.1. The DM methods were the standard CN2 and C4.5 algorithms, the HybridSOM framework combined first with CN2 and then with C4.5, and SIG*. Parallel coordinate plots were again used to optimize the candidate algorithmic parameter configurations.

Each DM algorithm had a unique set of parameters to optimize. The parameter value ranges for CN2 and C4.5 were dictated by the values allowed by the algorithm implementations. The SOM and SIG* parameter ranges were chosen to balance time complexity with adequate exploration of the parameter space. Tables 8.62 and 8.63 respectively show the parameters and associated value ranges for CN2 and C4.5. Tables 8.64 and 8.65 show the parameter ranges that are applicable to the HybridSOM framework configured with the CN2 and C4.5 algorithms, respectively. Finally, Table 8.66 shows the parameter value ranges for this dissertation's SIG* implementation.

In the case of the HybridSOM framework configurations and the SIG* algorithm, the SOM parameters were not optimized independently. Instead, the SOM parameters were treated as part of the rule extraction approach in question. This decision was made because the underlying interactions between the algorithmic parameters are not currently well understood. The decisions surrounding the tested SOM parameter ranges and the interactions between these parameters were identical to those discussed in Section 8.3.5, which were used during the supervised neuron labeling algorithm analysis.

The parallel coordinate plots of the parameter spaces for each of the DM algorithms were similar to those used during the neuron labeling algorithm optimizations. Only the

Table 8.62: The ranges within which parameter values varied for the CN2 algorithm.

Parameter	Symbol	Data Type	Data Set	Range
Error estimate	*cn2-err*	Ordinal	All	{Naïve, Laplacian}
Size of the *star* set	*cn2-star*	Continuous	All	[1, 200]
Significance threshold	*cn2-sig*	Continuous	All	[0.0, 200.0]

Table 8.63: The ranges within which parameter values varied for the C4.5 algorithm.

Parameter	Symbol	Data Type	Data Set	Range
Minimum examples in two outcomes	*c4.5-min*	Ordinal	Iris plants Ionosphere Monk's problems Pima Indians diabetes	[1, 145] [1, 340] [1, 418] [1, 743]
Test heuristic	*c4.5-heur*	Nominal	All	{Gain, Gain ratio}
Pessimistic error confidence	*c4.5-pess*	Continuous	All	[0.0, 100.0]
Fisher threshold (optional)	*c4.5-Fisher*	Continuous	All	[0.0, 100.0]
Rule pruning redundancy	*c4.5-redun*	Continuous	All	(0.0, 10 000.0]

Table 8.64: The ranges within which parameter values varied for HybridSOM using CN2.

Parameter	Symbol	Data Type	Data Set	Range
Map dimensions	Y, X	Ordinal	Iris plants Ionosphere Monk's problems Pima Indians diabetes	[2, 12] [2, 18] [2, 20] [2, 27]
Initial learning rate	$\eta(0)$	Continuous	All	[0.0, 10.0]
Learning rate decay constant	τ_1	Continuous	All	(0.0, 1 500.0]
Kernel width	$\sigma(0)$	Continuous	All	(0.0, Y]
Kernel width decay constant	τ_2	Continuous	All	(0.0, 100.0]
Error estimate	*cn2-err*	Ordinal	All	{Naïve, Laplacian}
Size of the *star* set	*cn2-star*	Continuous	All	[1, 200]
Significance threshold	*cn2-sig*	Continuous	All	[0.0, 200.0]

Table 8.65: The ranges within which parameter values varied for HybridSOM using C4.5.

Parameter	Symbol	Data Type	Data Set	Range
Map dimensions	Y, X	Ordinal	Iris plants Ionosphere Monk's problems Pima Indians diabetes	[2, 12] [2, 18] [2, 20] [2, 27]
Initial learning rate	$\eta(0)$	Continuous	All	[0.0, 10.0]
Learning rate decay constant	τ_1	Continuous	All	(0.0, 1 500.0]
Kernel width	$\sigma(0)$	Continuous	All	(0.0, Y]
Kernel width decay constant	τ_2	Continuous	All	(0.0, 100.0]
Minimum examples in two outcomes	$c4.5\text{-}min$	Ordinal	Iris plants Ionosphere Monk's problems Pima Indians diabetes	[1, 145] [1, 340] [1, 418] [1, 743]
Test heuristic	$c4.5\text{-}heur$	Nominal	All	{Gain, Gain ratio}
Pessimistic error confidence	$c4.5\text{-}pess$	Continuous	All	[0.0, 100.0]
Fisher threshold (optional)	$c4.5\text{-}Fisher$	Continuous	All	[0.0, 100.0]
Rule pruning redundancy	$c4.5\text{-}redun$	Continuous	All	(0.0, 10 000.0]

Table 8.66: The ranges within which parameter values varied for the SIG* algorithm.

Parameter	Symbol	Data Type	Data Set	Range
Map dimensions	Y, X	Ordinal	Iris plants Ionosphere Monk's problems Pima Indians diabetes	[2, 12] [2, 18] [2, 20] [2, 27]
Initial learning rate	$\eta(0)$	Continuous	All	[0.0, 10.0]
Learning rate decay constant	τ_1	Continuous	All	(0.0, 1 500.0]
Kernel width	$\sigma(0)$	Continuous	All	(0.0, Y]
Kernel width decay constant	τ_2	Continuous	All	(0.0, 100.0]
Characterizing threshold	θ_{char}	Continuous	All	[0.0, 100.0]
High characterizing constant	ϕ_{char}	Continuous	All	[0.0, 3.0]
Low characterizing constant	ψ_{char}	Continuous	All	[0.0, 3.0]
Differentiating threshold	θ_{diff}	Continuous	All	[0.0, 100.0]
High differentiating constant	ϕ_{diff}	Continuous	All	[0.0, 3.0]
Low differentiating constant	ψ_{diff}	Continuous	All	[0.0, 3.0]

final set of pruned optimal parameter settings were represented in all cases. Dashed lines again represented the most optimal parameter settings determined for each algorithm and data set combination. The last two parallel plot lines in each visualization represented, respectively, the means of the overall test set error and total rule set conditions.

The parameters for the basic CN2 rule extraction algorithm were simply tested over the full range of values that the standard implementation supported. Figure 8.2 shows an example parallel coordinate plot of the parameter space that was explored for the CN2 algorithm executed on the Iris plants data set. From the plot, it is clear that the optimal parameter settings produced a mean test set error of 2.667%, and a mean of 4.833 rule set conditions. The worst mean test error over all simulations was 86.000%, while the smallest and largest mean number of conditions were 1 and 20.900, respectively.

Figure 8.3 illustrates an example of the parallel coordinate plot of the parameter space for the standard C4.5 algorithm, when it was tested on the Iris plants data set. The parameters were tested over the full range of values accepted by the standard implementation. The optimal mean test set error was 5.333%, while a mean rule set condition count of 5.000 was achieved. The test error means varied between 4.000% and 86.000%, while the mean condition counts varied between 1.000 and 8.100 at the extremes.

The HybridSOM framework configured with the CN2 algorithm was also optimized using parallel coordinate plots, where Figure 8.4 again shows an example of a plot for the Iris plants data set. The parameters ranged over the values used for the SOMs of the labeling algorithm analysis, and the previously mentioned CN2 settings. The optimal mean test set error was 34.000%, where the minimum and maximum over all configurations were 11.212% and 62.424%, respectively. The mean number of rule set conditions for the selected parameter configuration was 2.000, while the lowest and highest mean number of conditions were respectively 1.000 and 21.333 over all configurations.

Figure 8.5 depicts an example of a parallel coordinate plot for the parameter space for the HybridSOM framework configured with the C4.5 rule extraction algorithm, once again for the Iris plants data set. The tested parameters once again ranged over the values previously used for the SOMs of the labeling algorithm optimizations, and the C4.5 parameters described above. The optimal configuration produced a mean overall test set error of 14.667%, as well as a 4.733 mean for the total conditions per rule set.

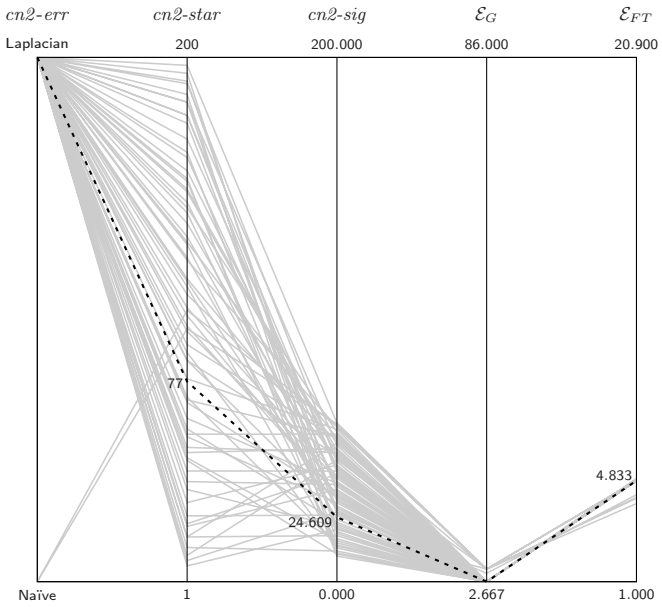


Figure 8.2: The parallel coordinate plot of the parameter value optimization performed for the CN2 algorithm, when applied to the Iris plants experimental data set.

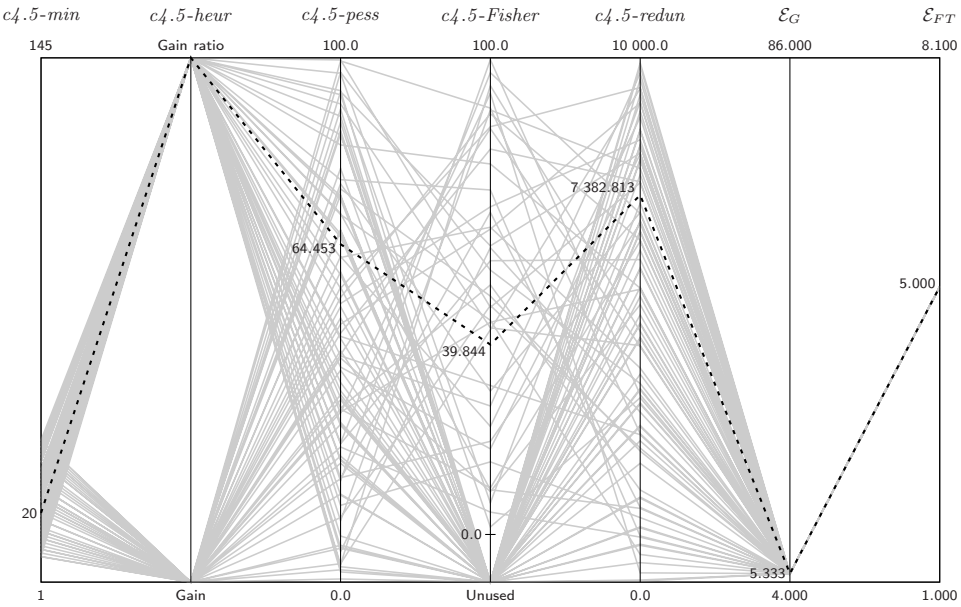


Figure 8.3: The parallel coordinate plot of the parameter value optimization performed for the C4.5 algorithm, when applied to the Iris plants experimental data set.

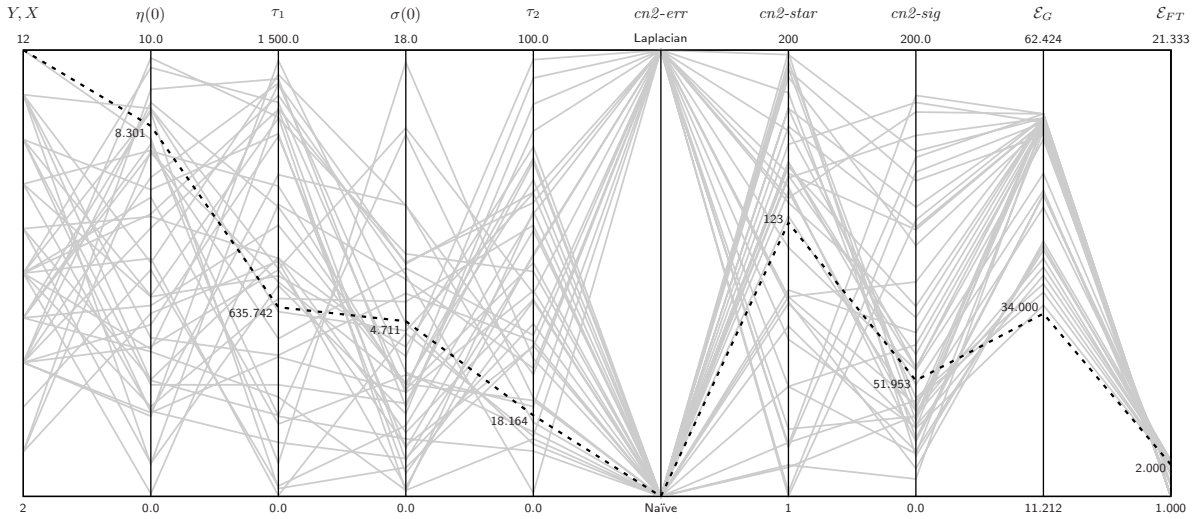


Figure 8.4: The parallel coordinate plot of the parameter value optimization performed for HybridSOM configured with CN2, when applied to the Iris plants experimental data set.

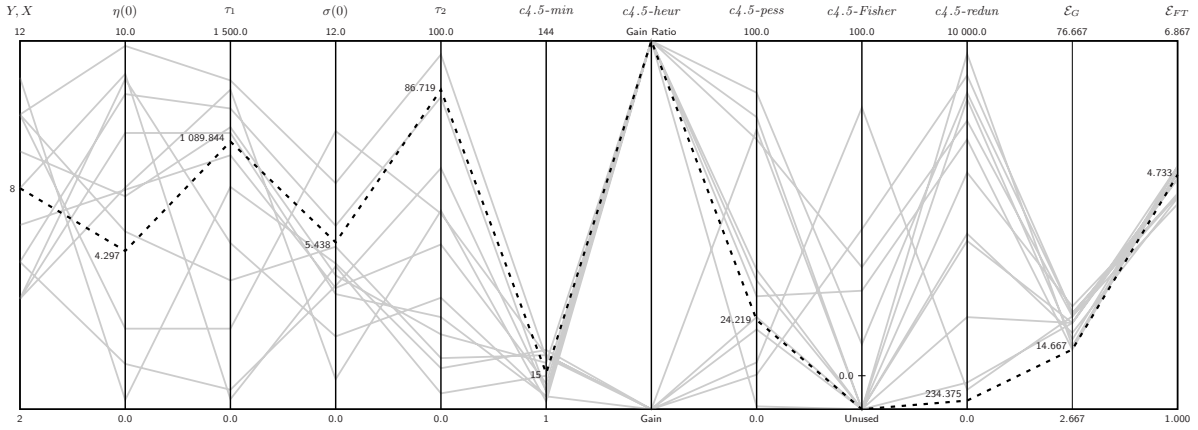


Figure 8.5: The parallel coordinate plot of the parameter value optimization performed for HybridSOM configured with C4.5, when applied to the Iris plants experimental data set.

The minimum test set error over the entire set of candidate parameter settings was 2.667%, while a maximum value of 76.667% was generated. The smallest mean number of conditions per rule set was 1.000, with a maximum of 6.867 conditions.

Finally, Figure 8.6 presents an example of the parallel coordinate plot visualization of the SIG* DM algorithm parameter space. The example again illustrates the parameter optimization for the Iris plants data set. The optimal configuration's mean overall test set

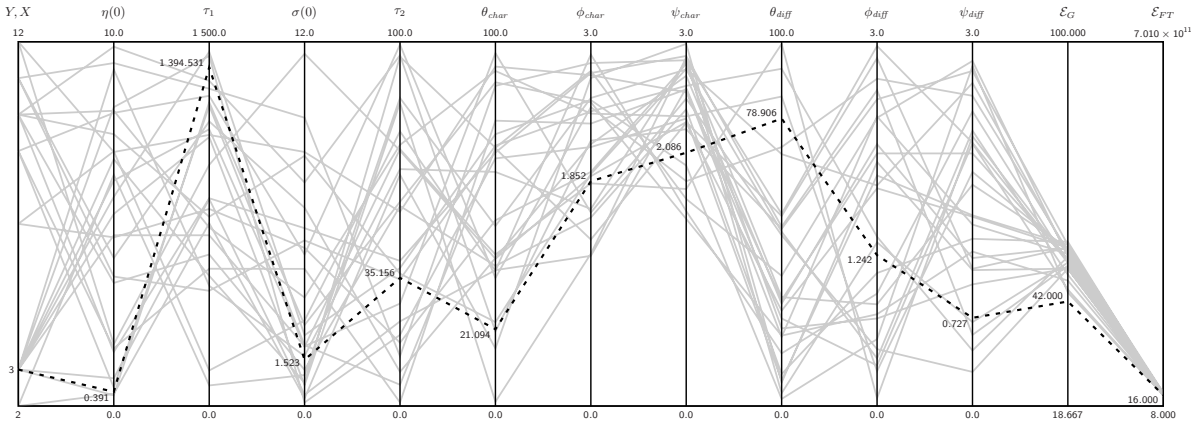


Figure 8.6: The parallel coordinate plot of the parameter value optimization performed for the SIG* algorithm, when applied to the Iris plants experimental data set.

error was 42.000%, with the full set of configurations producing a minimum of 18.667% and a maximum of 100.000%. The best configuration also produced a mean rule set condition count of 16.000, where the minimum and maximum mean over the entire set of candidate parameter settings were 8.000 and 7.010×10^{11}, respectively.

Tables 8.67 to 8.71 summarize the optimal parameter settings for each algorithm, when applied to the benchmark data sets. The parameters for the basic CN2 and C4.5 algorithms are respectively shown in Tables 8.67 and 8.68. Tables 8.69 and 8.70 provide the parameters for the HybridSOM framework, configured with the CN2 and C4.5 algorithms, respectively. Finally, Table 8.71 presents the SIG* rule extraction algorithm parameters. The last two rows in each of these tables list the mean of the overall training set error and the mean of the total condition count for the optimal configurations.

8.4.6 Comparison of Algorithmic Performance

This section presents a detailed discussion on the results of the comparative analyses of the investigated rule extraction algorithms. All of the performance measures described in Section 8.4.3 are taken into account in separate sections. The statistical comparison procedure described in Section 8.1.2 was followed throughout the investigation.

The discussion of the experiments on the DM algorithms is structured as follows: Section 8.4.6.1 investigates the overall training set error performance of the algorithms,

Table 8.67: Optimal parameters for the basic CN2 rule extraction algorithm.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
cn2-err	Laplacian	Naïve	Laplacian	Laplacian	Laplacian	Laplacian
cn2-star	77	185	45	129	196	140
cn2-sig	24.609	28.516	69.141	14.844	160.156	103.906
$\mathcal{E}_G$	2.667%	7.576%	0.000%	22.857%	2.857%	26.667%
$\mathcal{E}_{FT}$	4.833	4.967	8.000	24.667	3.000	2.933

Table 8.68: Optimal parameters for the basic C4.5 rule extraction algorithm.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
c4.5-min	20	42	93	11	56	97
c4.5-heur	Gain ratio	Gain	Gain ratio	Gain ratio	Gain ratio	Gain ratio
c4.5-pess	64.453	43.359	43.945	82.031	92.969	64.453
c4.5-Fisher	39.844	85.156	Unused	67.188	1.563	39.844
c4.5-redun	7 382.813	4 492.188	1 972.656	6 484.375	3 515.625	7 382.813
$\mathcal{E}_G$	5.333%	9.697%	24.286%	32.381%	2.857%	24.667%
$\mathcal{E}_{FT}$	5.000	5.000	3.000	56.600	5.000	3.500

Table 8.69: Optimal parameters for the HybridSOM framework configured with CN2.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
Y, X	12	8	13	11	20	3
$\eta(0)$	8.301	7.793	0.410	5.000	0.039	9.668
τ_1	635.742	125.977	38.086	750.000	462.891	407.227
$\sigma(0)$	4.711	1.109	11.096	5.500	11.016	2.104
τ_2	18.164	81.055	50.195	50.000	57.422	24.805
cn2-err	Naïve	Naïve	Laplacian	Naïve	Naïve	Naïve
cn2-star	123	128	68	101	163	165
cn2-sig	51.953	16.797	125.391	100.000	25.781	4.297
$\mathcal{E}_G$	34.000%	18.788%	46.429%	32.857%	38.333%	30.267%
$\mathcal{E}_{FT}$	2.000	2.567	1.000	1.000	1.733	1.667

Table 8.70: Optimal parameters for the HybridSOM framework configured with C4.5.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
Y, X	8	7	20	19	16	25
$\eta(0)$	4.297	8.965	9.043	9.336	0.156	4.727
τ_1	1 089.844	676.758	1 438.477	685.547	1 429.688	322.266
$\sigma(0)$	5.438	3.322	10.273	12.098	12.750	3.613
τ_2	86.719	30.273	43.555	80.078	89.063	85.547
c4.5-min	15	12	106	151	44	101
c4.5-heur	Gain ratio	Gain ratio	Gain ratio	Gain	Gain	Gain
c4.5-pess	24.219	34.961	88.867	11.328	42.188	48.047
c4.5-Fisher	Unused	Unused	Unused	Unused	Unused	36.719
c4.5-redun	234.375	722.656	3 769.531	7 929.688	7 968.750	3 945.313
$\mathcal{E}_G$	14.667%	20.000%	37.857%	32.857%	19.048%	26.400%
$\mathcal{E}_{FT}$	4.733	3.133	1.933	1.000	3.067	3.200

Table 8.71: Optimal parameters for the SIG* rule extraction algorithm.

Parameter	Iris Plants	Ionosphere	Monk's Problem 1	Monk's Problem 2	Monk's Problem 3	Pima Indians Diabetes
Y, X	3	8	18	11	9	7
$\eta(0)$	0.391	9.258	4.063	9.980	2.070	6.094
τ_1	1 394.531	1 423.828	1 171.875	266.602	1 376.953	1 007.813
$\sigma(0)$	1.523	2.781	10.688	8.744	7.348	0.109
τ_2	35.156	93.359	9.375	27.930	52.734	29.688
θ_{char}	21.094	8.203	40.625	60.742	55.078	45.313
ϕ_{char}	1.852	1.746	2.156	2.596	2.215	2.672
ψ_{char}	2.086	1.957	2.719	2.572	2.613	2.484
θ_{diff}	78.906	13.672	28.125	78.711	98.047	14.063
ϕ_{diff}	1.242	2.168	2.156	2.725	1.324	2.297
ψ_{diff}	0.727	2.684	1.219	2.865	1.090	2.578
$\mathcal{E}_G$	42.000%	16.970%	33.571%	32.857%	27.143%	31.467%
$\mathcal{E}_{FT}$	16.000	9.403×10^{11}	3.201×10^{9}	34.000	8.449×10^{13}	1.242×10^{101}

while Sections 8.4.6.2 and 8.4.6.3 analyze the contributions of misclassified and unclassified data examples to this error. The results of the test set error investigation are described in Section 8.4.6.4, where Sections 8.4.6.5 and 8.4.6.6 focus on the test set errors that were due to misclassified and unclassified data examples, respectively. The three rule set complexity measures are focused on in the final three sections, with Section 8.4.6.7 considering the total rule set conditions, Section 8.4.6.8 discussing the total rule count, and Section 8.4.6.9 looks at the average number of conditions per rule.

8.4.6.1 Overall Training Set Error

Tables 8.72 to 8.77 outline the comparative statistics used to analyze the overall training set error for the investigated DM algorithms executed on the experimental data sets. The Iris plants data set is focused on in Table 8.72, while Table 8.73 investigates the ionosphere data set. Tables 8.74, 8.75, and 8.76 investigate the three monk's problems. Finally, the Pima Indians diabetes data set analysis is summarized in Table 8.77.

It is possible for a hypothesis test to conclude that two algorithms are significantly different (which denotes that the results represent clearly distinct populations) while the means and standard deviations of the results signify that the measures themselves are almost identical. In such a case, the differences between the algorithms are *statistically significant*, while being subjectively judged to be *practically insignificant* in a real-world sense [66]. This analysis assumed practically insignificant performance differences when the means and standard deviations of algorithms were equivalent to a precision of three decimal digits. Table 8.75 shows that this situation arose for the overall training error measured on the second monk's problem for HybridSOM configured with CN2, and SIG*. Table 8.76 shows the same outcome for basic CN2 and C4.5 on monk's problem 3. Table footnotes show the mean and standard deviation differences in these cases.

The SOM-based rule extraction algorithms generally performed poorly in comparison to the basic CN2 and C4.5 algorithms. None of the SOM-based methods performed the best, or equivalent to the best performing approaches, on any of the data sets.

The results indicate that the basic CN2 algorithm was the best performing technique. CN2 outperformed all five of the other rule extraction approaches on three data sets (the Iris plants, ionosphere, and first monk's problem data sets), and jointly outperformed

Table 8.72: Statistical comparison of the overall training set error for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	2.138	0.491	3.977	0.348	33.678	2.097	14.345	10.718
Unmodified C4.5								
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 7.451×10^{-9}						
3.977	0.348							
HybridSOM (CN2)								
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}				
33.678	2.097							
HybridSOM (C4.5)								
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\leftarrow$ 8.009×10^{-8}		
14.345	10.718							
SIG* algorithm								
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 3.725×10^{-9}		$\uparrow$ 1.863×10^{-9}
41.862	1.148							

Table 8.73: Statistical comparison of the overall training set error for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	6.049	0.455	9.118	0.278	18.873	5.097	20.941	4.276
Unmodified C4.5								
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}						
9.118	0.278							
HybridSOM (CN2)								
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}				
18.873	5.097							
HybridSOM (C4.5)								
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\square$ 0.061		
20.941	4.276							
SIG* algorithm								
$\mathcal{E}_T$	$\mathcal{S}_T$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\square$ 0.035		$\leftarrow$ 7.994×10^{-6}
16.451	3.127							

Table 8.74: Statistical comparison of the overall training set error for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	0.000	0.000	25.024	0.395	50.120	0.425	38.349	12.723
Unmodified C4.5	$\uparrow$							
$\mathcal{E}_T$ / $\mathcal{S}_T$	1.863×10^{-9}							
25.024 / 0.395								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_T$ / $\mathcal{S}_T$	1.863×10^{-9}		1.863×10^{-9}					
50.120 / 0.425								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_T$ / $\mathcal{S}_T$	1.863×10^{-9}		3.052×10^{-5}		7.358×10^{-4}			
38.349 / 12.723								
SIG* algorithm	$\uparrow$		$\leftarrow$		$\leftarrow$		$\leftarrow$	
$\mathcal{E}_T$ / $\mathcal{S}_T$	1.863×10^{-9}		2.316×10^{-4}		1.863×10^{-9}		5.718×10^{-7}	
23.158 / 2.328								

Table 8.75: Statistical comparison of the overall training set error for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	21.300	10.146	19.769	1.812	32.871	0.429	32.871	0.429
Unmodified C4.5	$\square$							
$\mathcal{E}_T$ / $\mathcal{S}_T$	0.413							
19.769 / 1.812								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_T$ / $\mathcal{S}_T$	2.384×10^{-7}		1.863×10^{-9}					
32.871 / 0.429								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\square$			
$\mathcal{E}_T$ / $\mathcal{S}_T$	4.470×10^{-8}		1.863×10^{-9}		6.233×10^{-3}			
32.871 / 0.429								
SIG* algorithm	$\uparrow$		$\uparrow$		$\uparrow$ *		$\square$	
$\mathcal{E}_T$ / $\mathcal{S}_T$	2.980×10^{-8}		1.863×10^{-9}		5.188×10^{-4}		6.233×10^{-3}	
32.871 / 0.429								

* Means differ by 3.081×10^{-15}, while standard deviations differ by 9.841×10^{-16}, indicating practically insignificant result differences.

Table 8.76: Statistical comparison of the overall training set error for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	7.289	0.365	7.289	0.365	42.400	10.618	21.930	2.866
Unmodified C4.5	$\leftarrow$ *							
$\mathcal{E}_T$ / $\mathcal{S}_T$	1.863×10^{-9}							
7.289 / 0.365								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_T$ / $\mathcal{S}_T$	3.725×10^{-9}		1.863×10^{-9}					
42.400 / 10.618								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_T$ / $\mathcal{S}_T$	3.725×10^{-9}		3.725×10^{-9}		8.009×10^{-8}			
21.930 / 2.866								
SIG* algorithm	$\uparrow$		$\uparrow$		$\leftarrow$		$\uparrow$	
$\mathcal{E}_T$ / $\mathcal{S}_T$	1.863×10^{-9}		1.863×10^{-9}		7.758×10^{-6}		5.588×10^{-9}	
29.147 / 5.238								

* Means differ by 6.951×10^{-8}, while standard deviations differ by 3.480×10^{-9}, indicating practically insignificant result differences.

Table 8.77: Statistical comparison of the overall training set error for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$	$\mathcal{E}_T$	$\mathcal{S}_T$
	25.953	1.711	25.141	0.407	31.175	4.232	26.563	1.339
Unmodified C4.5	$\leftarrow$							
$\mathcal{E}_T$ / $\mathcal{S}_T$	3.725×10^{-9}							
25.141 / 0.407								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_T$ / $\mathcal{S}_T$	1.118×10^{-8}		1.863×10^{-9}					
31.175 / 4.232								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_T$ / $\mathcal{S}_T$	1.207×10^{-3}		2.980×10^{-8}		1.598×10^{-5}			
26.563 / 1.339								
SIG* algorithm	$\uparrow$		$\uparrow$		$\square$		$\uparrow$	
$\mathcal{E}_T$ / $\mathcal{S}_T$	4.470×10^{-8}		1.863×10^{-9}		0.719		1.863×10^{-8}	
30.740 / 1.808								

the other methods in conjunction with the C4.5 algorithm on a further two data sets (the second and third monk's problems). For the remaining Pima Indians diabetes experimental data set, CN2 was outperformed only by the raw C4.5 algorithm.

The raw C4.5 algorithm performed slightly worse than CN2 did, but better than the SOM-based approaches. Unmodified C4.5 performed better than all other rule extractors on the Pima Indians diabetes data set, and equivalently to CN2 on two data sets (the previously mentioned second and third monk's problems). On the Iris and ionosphere data sets, C4.5 was only outperformed by CN2. When considering the first monk's problem, C4.5 only performed better than the HybridSOM configurations, while also performing worse than both the CN2 and SIG* rule extraction algorithms.

In almost all cases the basic CN2 and C4.5 algorithms outperformed the SOM-based techniques. Unmodified CN2 performed better than all the SOM-oriented techniques on every data set in the analysis. The basic C4.5 approach achieved the same result on all but the first monk's problem, where the method was outperformed by SIG*.

The best performing SOM-based approach was the HybridSOM framework configured with C4.5. On the Iris plants, third monk's problem, and Pima Indians diabetes data sets this HybridSOM configuration exhibited average performance by outperforming two approaches, while being outperformed by the other two. The first monk's problem saw the C4.5 configured framework performing better than only one approach. For the two remaining data sets (the monk's problem 2 and ionosphere sets), the C4.5-based configuration performed equivalently to the methods that performed the worst.

The SIG* algorithm performed slightly worse than the HybridSOM configuration that used C4.5. On the first monk's problem, SIG* was the second best performing technique, beaten only by CN2. The third monk's problem saw SIG* performing better than only one approach. For the Iris data set, SIG* performed worse than every other technique, while performing equivalently to the worst performing approaches for the last three sets (ionosphere, the second monk's problem, and Pima Indians diabetes).

The CN2 configuration of HybridSOM fared the worst out of all the algorithms. Considering the general performance of CN2-based HybridSOM, this method was outperformed by every other approach on monk's problems 1 and 3. The HybridSOM configuration was equivalent to the worst performing algorithms in three cases (ionosphere,

monk's problem 2, and Pima Indians diabetes). For ionosphere and monk's problem 2, HybridSOM with CN2 was equivalent to the other SOM-based methods, while in the Pima Indians diabetes set the configuration was equivalent to only SIG*. All algorithms, save SIG*, outperformed HybridSOM with CN2 on the Iris plants data set.

To consolidate the above analysis, the SOM-based rule extractors were compared directly with one another. Firstly, the two HybridSOM frameworks were compared. On four of the data sets the performance of the C4.5 configured framework exceeded the performance of HybridSOM configured with CN2 (for Iris plants, monk's problems 1 and 3, and Pima Indians diabetes). The remaining two instances, namely the ionosphere and monk's problem 2 data sets, exhibited characteristics under which both of the HybridSOM configurations performed equivalently to one another.

Next, the SIG* algorithm was compared to the two HybridSOM configurations. Three data sets (Iris plants, monk's problem 3, and Pima Indians diabetes) exhibited better performance from the C4.5 configured HybridSOM framework, in comparison to SIG*, while the opposite was true only within the first monk's problem. In the outstanding two experimental data sets, namely ionosphere and monk's problem 2, SIG* and HybridSOM with C4.5 performed equally well. SIG* outperformed CN2 configured HybridSOM on two data sets (the first and third monk's problems), and the opposite was true for only the Iris plants data set. In the remaining three cases (the ionosphere, monk's problem 2 and Pima Indians diabetes data sets), the methods performed equivalently.

Finally, the analysis also focused on the performance of each HybridSOM configuration in relation to the algorithm with which the framework was configured. HybridSOM, when configured with C4.5, was outperformed by the unmodified C4.5 algorithm on all six of the benchmark data sets. Additionally, all six experimental data sets showed that CN2 combined with HybridSOM performed worse than CN2 on its own.

8.4.6.2 Training Set Error Due to Misclassified Data Examples

Tables 8.78 to 8.83 compare the DM methods in terms of the training error due to only misclassified data. Table 8.78 outlines the Iris plants data set, while Table 8.79 shows the results for the ionosphere data set. Tables 8.80 to 8.82 represent the three monk's problems. Finally, the Pima Indians diabetes data set results are presented in Table 8.83.

Table 8.78: Statistical comparison of the training set error due to misclassified data examples for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	2.138	0.491	3.977	0.348	33.678	2.097	14.345	10.718
Unmodified C4.5	$\uparrow$							
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	7.451×10^{-9}							
3.977 / 0.348								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		1.863×10^{-9}					
33.678 / 2.097								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		1.863×10^{-9}		8.009×10^{-8}			
14.345 / 10.718								
SIG* algorithm	$\uparrow$		$\uparrow$		$\leftarrow$		$\uparrow$	
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}		1.598×10^{-5}	
26.184 / 1.409								

Table 8.79: Statistical comparison of the training set error due to misclassified data examples for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	6.049	0.455	9.118	0.278	18.873	5.097	20.941	4.276
Unmodified C4.5	$\uparrow$							
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}							
9.118 / 0.278								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		1.863×10^{-9}					
18.873 / 5.097								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\square$			
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		1.863×10^{-9}		0.061			
20.941 / 4.276								
SIG* algorithm	$\uparrow$		$\uparrow$		$\square$		$\leftarrow$	
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		1.863×10^{-9}		0.013		4.422×10^{-6}	
16.147 / 3.118								

Table 8.80: Statistical comparison of the training set error due to misclassified data examples for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	0.000	0.000	25.024	0.395	50.120	0.425	38.349	12.723
Unmodified C4.5	$\uparrow$							
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}							
25.024 / 0.395								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		1.863×10^{-9}					
50.120 / 0.425								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		3.052×10^{-5}		7.358×10^{-4}			
38.349 / 12.723								
SIG* algorithm	$\uparrow$		$\leftarrow$		$\leftarrow$		$\leftarrow$	
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		8.242×10^{-6}		1.863×10^{-9}		2.049×10^{-7}	
22.376 / 2.433								

Table 8.81: Statistical comparison of the training set error due to misclassified data examples for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	21.300	10.146	19.769	1.812	32.871	0.429	32.871	0.429
Unmodified C4.5	$\square$							
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	0.413							
19.769 / 1.812								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	2.384×10^{-7}		1.863×10^{-9}					
32.871 / 0.429								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\square$			
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	4.470×10^{-8}		1.863×10^{-9}		6.233×10^{-3}			
32.871 / 0.429								
SIG* algorithm	$\uparrow$		$\uparrow$		$\uparrow$ *		$\square$	
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	2.980×10^{-8}		1.863×10^{-9}		5.188×10^{-4}		6.233×10^{-3}	
32.871 / 0.429								

* Means differ by 3.081×10^{-15}, while standard deviations differ by 9.841×10^{-16}, indicating practically insignificant result differences.

Table 8.82: Statistical comparison of the training set error due to misclassified data examples for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	7.289	0.365	7.289	0.365	42.400	10.618	21.930	2.866
Unmodified C4.5	$\leftarrow$ *							
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}							
7.289 / 0.365								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	3.725×10^{-9}		1.863×10^{-9}					
42.400 / 10.618								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	3.725×10^{-9}		3.725×10^{-9}		8.009×10^{-8}			
21.930 / 2.866								
SIG* algorithm	$\uparrow$		$\uparrow$		$\leftarrow$		$\uparrow$	
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.863×10^{-9}		1.863×10^{-9}		7.758×10^{-6}		5.588×10^{-9}	
29.147 / 5.238								

* Means differ by 6.951×10^{-8}, while standard deviations differ by 3.480×10^{-9}, indicating practically insignificant result differences.

Table 8.83: Statistical comparison of the training set error due to misclassified data examples for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$	$\mathcal{E}_{TM}$	$\mathcal{S}_{TM}$
	25.953	1.711	25.141	0.407	31.175	4.232	26.563	1.339
Unmodified C4.5	$\leftarrow$							
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	3.725×10^{-9}							
25.141 / 0.407								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.118×10^{-8}		1.863×10^{-9}					
31.175 / 4.232								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	1.207×10^{-3}		2.980×10^{-8}		1.598×10^{-5}			
26.563 / 1.339								
SIG* algorithm	$\uparrow$		$\uparrow$		$\leftarrow$		$\uparrow$	
$\mathcal{E}_{TM}$ / $\mathcal{S}_{TM}$	7.702×10^{-6}		9.313×10^{-9}		1.900×10^{-3}		2.287×10^{-4}	
28.367 / 1.704								

As Table 8.75 demonstrates for the overall training error, Table 8.81 shows that SIG* and HybridSOM configured with CN2 differed statistically significantly on monk's problem 2, but that the differences were slight, and not practically significant according to the criteria mentioned in Section 8.4.6.1. Furthermore, as in Table 8.76 from the previous section, Table 8.82 reveals statistically significant differences between unmodified CN2 and C4.5 for monk's problem 3, which could not be judged practically significant.

The results described in this section were predominantly similar to those discussed in the previous section, where the overall training error results were outlined. Once again, none of the SOM-based rule extraction techniques were capable of outperforming all the other data mining algorithms for any of the investigated experimental data sets.

The results for the training error due to misclassifications were compared to those for the basic training error. Although several statistics differed, the overall interrelationships between CN2 and the other rule extraction algorithms were the same for both measures. As a result, the comparative performance analysis drew the same conclusions, and the CN2 rule extraction algorithm was deemed to be the superior algorithm amongst the analyzed approaches, when considering the investigated experimental data sets.

The performance of the basic C4.5 algorithm in relation to the other rule extractors was also identical to what was observed in the context of the overall training set error. As a result of this, C4.5 was again considered to be the second best performing approach in terms of the training set error due to misclassification that was observed during the reported analysis, where C4.5 performed slightly worse than the CN2 algorithm.

Once again, as was observed during the basic training error analysis, unmodified CN2 and C4.5 outperformed the SOM-based approaches in almost all cases. Exactly the same relationships were observed, where CN2 outperformed every SOM-based approach in all cases, and C4.5 succeeded in achieving the same result on all but one data set.

The interrelationships between C4.5 configured HybridSOM and the other data mining algorithms were also the same as those observed in the previous section. Just as was seen when the overall training error was analyzed, the HybridSOM framework combined with C4.5 thus clearly performed the best out of the SOM-based algorithms.

SIG* was again the second best performing technique amongst the SOM-oriented approaches. The relationships between SIG* and the other algorithms was very similar to

what was observed when the overall training error was concerned. The only differences were found in the Iris and Pima Indians diabetes data sets. In the former case, SIG* outperformed one technique, instead of being outperformed by all the other techniques. In the latter case, SIG* again performed better than one algorithm, rather than performing the worst in conjunction with HybridSOM's CN2 configuration. Due to these differences, SIG* much more definitively outperformed the CN2 configured HybridSOM framework than was observed during the analysis of the basic training set error.

Finally, CN2 combined with HybridSOM performed the worst out of the data mining algorithms. As was the case for the SIG* algorithm, the only differences between the overall training error and the training error due to misclassification were observed on the Iris plants and Pima Indians diabetes data sets. On the Iris data set CN2 configured HybridSOM was outperformed by every other technique, instead of outperforming one method. For the Pima Indians diabetes data set, HybridSOM with CN2 also underperformed in relation to all other approaches, as opposed to performing equivalently to the SIG* algorithm. These differences mean that CN2-based HybridSOM performed even more poorly than was observed during the overall training error analysis.

The performance interrelationships between the SOM-based methods were again analyzed in more detail. When the two HybridSOM methods were compared, the relationship demonstrated during the basic training error analysis was again observed.

When focusing on the relationship between the SIG* algorithm and the two HybridSOM configurations, the relationships between SIG* and HybridSOM configured with C4.5 were unchanged from what was seen in the overall training error analysis. The relationship between SIG* and CN2-based HybridSOM differed somewhat, however. On the Iris data set, SIG* was observed to outperform HybridSOM with CN2, as opposed to the opposite, which was the case for the overall training error. The Pima Indians diabetes data set revealed that SIG* also outperformed HybridSOM with CN2, rather than performing equivalently. These differences resulted in SIG* outperforming HybridSOM with CN2 four times, and the approaches performing equivalently twice.

The last analysis for the training error due to misclassified data compared each HybridSOM configuration to the unmodified version of the algorithm with which the framework was configured. C4.5 again outperformed the C4.5 version of HybridSOM for every

analyzed data set. The CN2 rule extraction algorithm combined with HybridSOM was also always outperformed by the CN2 algorithm alone. This was exactly the same as the behavior that was observed during the analysis of the overall training set error.

8.4.6.3 Training Set Error Due to Unclassified Data Examples

The errors resulting only from unclassified training data examples are presented and analyzed in Tables 8.84 to 8.89. Table 8.84 focuses on the performance interrelationships observed on the Iris plants data set, while Table 8.85 investigates the performance within the ionosphere experimental data set. The performance characteristics of the algorithms on the three monk's problem benchmark data sets are respectively outlined in Tables 8.86, 8.87, and 8.88. Finally, Table 8.89 considers the training error performance due to unclassified data examples for the Pima Indians diabetes data set.

When attention is focused on the error due to only the unclassified training set examples, it was readily apparent that the SIG* rule extraction algorithm underperformed in relation to the other data mining techniques when the methods were tested on the investigated benchmark data sets. No Wilcoxon signed-rank test significance values could be computed when comparing the basic CN2 algorithm, the unmodified C4.5 algorithm, and both HybridSOM framework configurations to one another. This indicates that the approaches performed identically in terms of this performance measure. The means and standard deviations of 0.0 further point to the fact that none of these methods produced any unclassified training set examples. This was because CN2 and C4.5 produce rule sets with default rules, guaranteeing a classification for all presented data.

Four experimental data sets exhibited SIG* performance that was worse than all the other rule extraction techniques in a statistically significant manner. This was the case for the Iris plants, ionosphere, first monk's problem, and Pima Indians diabetes data sets. While the effect was not overwhelming in most of these instances, more than 15% of the training set data examples in the Iris plants data set remained unclassified. In only two of the data sets, namely the second and third monk's problems, there was no statistically significant difference between SIG* and the other data mining approaches. In fact, the SIG* algorithm executed on these two data sets produced no unclassified training data examples at all, exactly like the other tested rule extraction algorithms.

Table 8.84: Statistical comparison of the training set error due to unclassified data examples for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A						
0.000	0.000							
HybridSOM (CN2)								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A				
0.000	0.000							
HybridSOM (C4.5)								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A		□ N/A		
0.000	0.000							
SIG* algorithm								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	↑ 1.863×10^{-9}		↑ 1.863×10^{-9}		↑ 1.863×10^{-9}		↑ 1.863×10^{-9}
15.678	0.991							

Table 8.85: Statistical comparison of the training set error due to unclassified data examples for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A						
0.000	0.000							
HybridSOM (CN2)								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A				
0.000	0.000							
HybridSOM (C4.5)								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A		□ N/A		
0.000	0.000							
SIG* algorithm								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	↑ 6.104×10^{-5}		↑ 6.104×10^{-5}		↑ 6.104×10^{-5}		↑ 6.104×10^{-5}
0.304	0.412							

Table 8.86: Statistical comparison of the training set error due to unclassified data examples for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A						
0.000	0.000							
HybridSOM (CN2)								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A				
0.000	0.000							
HybridSOM (C4.5)								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A		□ N/A		
0.000	0.000							
SIG* algorithm								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	↑ 7.451×10^{-9}		↑ 7.451×10^{-9}		↑ 7.451×10^{-9}		↑ 7.451×10^{-9}
0.781	0.547							

Table 8.87: Statistical comparison of the training set error due to unclassified data examples for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A						
0.000	0.000							
HybridSOM (CN2)								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A				
0.000	0.000							
HybridSOM (C4.5)								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A		□ N/A		
0.000	0.000							
SIG* algorithm								
$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	□ N/A		□ N/A		□ N/A		□ N/A
0.000	0.000							

Table 8.88: Statistical comparison of the training set error due to unclassified data examples for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5	□ N/A							
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$								
0.000 / 0.000								
HybridSOM (CN2)	□ N/A		□ N/A					
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$								
0.000 / 0.000								
HybridSOM (C4.5)	□ N/A		□ N/A		□ N/A			
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$								
0.000 / 0.000								
SIG* algorithm	□ N/A		□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$								
0.000 / 0.000								

Table 8.89: Statistical comparison of the training set error due to unclassified data examples for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$	$\mathcal{E}_{TU}$	$\mathcal{S}_{TU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5	□ N/A							
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$								
0.000 / 0.000								
HybridSOM (CN2)	□ N/A		□ N/A					
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$								
0.000 / 0.000								
HybridSOM (C4.5)	□ N/A		□ N/A		□ N/A			
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$								
0.000 / 0.000								
SIG* algorithm	↑ 1.863×10^{-9}		↑ 1.863×10^{-9}		↑ 1.863×10^{-9}		↑ 1.863×10^{-9}	
$\mathcal{E}_{TU}$ / $\mathcal{S}_{TU}$								
2.373 / 0.643								

8.4.6.4 Overall Test Set Error

Tables 8.90 to 8.95 present the overall test set classification error, which was assessed on the experimental data sets. Table 8.90 illustrates results for the Iris plants data set. The ionosphere data set results are summarized in Table 8.91, while Tables 8.92 to 8.93 respectively focus on the three monk's problem data sets. Lastly, Table 8.90 considers the overall test set error performance on the Pima Indians diabetes set.

Table 8.94 shows that there was a difference between the unmodified CN2 and C4.5 algorithms that was statistically significant. However, this difference was not considered practically significant, based on the criteria that Section 8.4.6.1 mentioned.

In general, the overall test set error was much more uniform than the overall training set error was observed to be. All of the data sets saw several groups of equivalently performing rule extraction algorithms. This indicates that it is more difficult to tell the approaches apart from one another when it comes to test set performance.

The unmodified CN2 algorithm again performed generally the best out of the five tested rule extraction techniques in the context of the investigated data sets. CN2 outperformed all the other approaches on two data sets, namely the first and second monk's problems. On the remaining four benchmark data sets (Iris plants, ionosphere, monk's problem 3, and Pima Indians diabetes) CN2 performed equivalently to the other best performing rule extraction algorithms. The CN2 method was not outperformed by any of the other algorithms across all the tested experimental data sets.

The basic C4.5 algorithm did not perform as well as CN2, but still outperformed the SOM-based rule extraction approaches over the full set of benchmark data sets. C4.5 did not outperform every algorithm on any data set, but did perform equivalently to the best performing CN2 algorithm in four cases (the Iris plants, ionosphere, monk's problem 3, and Pima Indians diabetes data sets). The basic C4.5 method was outperformed by only the CN2 algorithm in two cases, on the first and second monk's problem experimental data sets. In the latter data set, C4.5 jointly performed the worst, in conjunction with all the SOM-based data mining methods that were experimented upon.

The unmodified CN2 and C4.5 algorithms generally outperformed all the SOM-based rule extraction approaches. The CN2 and C4.5 techniques both successfully performed better than all the SOM-based algorithms on the Iris plants, ionosphere, and monk's

Table 8.90: Statistical comparison of the overall test set error for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	2.667	6.915	5.333	10.417	34.000	20.443	14.667	13.830
Unmodified C4.5	□ 0.219							
$\mathcal{E}_G$ / $\mathcal{S}_G$								
5.333 / 10.417								
HybridSOM (CN2)	↑ 1.192×10^{-7}		↑ 1.907×10^{-6}					
$\mathcal{E}_G$ / $\mathcal{S}_G$								
34.000 / 20.443								
HybridSOM (C4.5)	↑ 1.526×10^{-5}		↑ 3.723×10^{-3}		← 7.771×10^{-4}			
$\mathcal{E}_G$ / $\mathcal{S}_G$								
14.667 / 13.830								
SIG* algorithm	↑ 1.490×10^{-8}		↑ 2.384×10^{-7}		□ 0.209		↑ 8.449×10^{-6}	
$\mathcal{E}_G$ / $\mathcal{S}_G$								
42.000 / 23.104								

Table 8.91: Statistical comparison of the overall test set error for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	7.576	7.581	9.697	8.586	18.788	8.247	20.000	9.369
Unmodified C4.5	□ 0.725							
$\mathcal{E}_G$ / $\mathcal{S}_G$								
9.697 / 8.586								
HybridSOM (CN2)	↑ 2.384×10^{-7}		↑ 1.088×10^{-4}					
$\mathcal{E}_G$ / $\mathcal{S}_G$								
18.788 / 8.247								
HybridSOM (C4.5)	↑ 2.235×10^{-7}		↑ 3.815×10^{-6}		□ 0.741			
$\mathcal{E}_G$ / $\mathcal{S}_G$								
20.000 / 9.369								
SIG* algorithm	↑ 2.377×10^{-4}		↑ 1.044×10^{-3}		□ 0.181		□ 0.277	
$\mathcal{E}_G$ / $\mathcal{S}_G$								
16.970 / 13.018								

Table 8.92: Statistical comparison of the overall test set error for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	0.000	0.000	24.286	11.804	46.429	12.688	37.857	16.369
Unmodified C4.5								
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 1.863×10^{-9}						
24.286	11.804							
HybridSOM (CN2)								
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 1.863×10^{-9}	$\uparrow$ 5.513×10^{-7}					
46.429	12.688							
HybridSOM (C4.5)								
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 1.863×10^{-9}	$\uparrow$ 6.104×10^{-5}		$\square$ 0.048			
37.857	16.369							
SIG* algorithm								
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 1.863×10^{-9}	$\square$ 0.019		$\leftarrow$ 1.180×10^{-4}		$\square$ 0.133	
33.571	10.300							

Table 8.93: Statistical comparison of the overall test set error for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	22.857	11.317	32.381	10.895	32.857	12.805	32.857	12.805
Unmodified C4.5								
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 8.794×10^{-4}						
32.381	10.895							
HybridSOM (CN2)								
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 3.967×10^{-4}	$\square$ 0.903					
32.857	12.805							
HybridSOM (C4.5)								
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 1.375×10^{-4}	$\square$ 0.841		$\square$ 0.046			
32.857	12.805							
SIG* algorithm								
$\mathcal{E}_G$	$\mathcal{S}_G$	$\uparrow$ 5.887×10^{-4}	$\square$ 0.919		$\square$ 0.112		$\square$ 0.047	
32.857	12.805							

Table 8.94: Statistical comparison of the overall test set error for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	2.857	4.023	2.857	4.023	38.333	16.512	19.048	9.441
Unmodified C4.5	$\leftarrow$ *							
$\mathcal{E}_G$ / $\mathcal{S}_G$	9.766×10^{-4}							
2.857 / 4.023								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_G$ / $\mathcal{S}_G$	3.725×10^{-9}		1.863×10^{-9}					
38.333 / 16.512								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_G$ / $\mathcal{S}_G$	7.451×10^{-9}		7.451×10^{-9}		1.151×10^{-6}			
19.048 / 9.441								
SIG* algorithm	$\uparrow$		$\uparrow$		$\leftarrow$		$\square$	
$\mathcal{E}_G$ / $\mathcal{S}_G$	1.863×10^{-9}		1.863×10^{-9}		9.553×10^{-4}		0.012	
27.143 / 14.335								

* Means differ by 2.725×10^{-8}, while standard deviations differ by 3.837×10^{-8}, indicating practically insignificant result differences.

Table 8.95: Statistical comparison of the overall test set error for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$	$\mathcal{E}_G$	$\mathcal{S}_G$
	26.667	8.683	24.667	8.604	30.267	9.017	26.400	8.700
Unmodified C4.5	$\square$							
$\mathcal{E}_G$ / $\mathcal{S}_G$	0.175							
24.667 / 8.604								
HybridSOM (CN2)	$\square$		$\uparrow$					
$\mathcal{E}_G$ / $\mathcal{S}_G$	0.040		2.356×10^{-3}					
30.267 / 9.017								
HybridSOM (C4.5)	$\square$		$\square$		$\square$			
$\mathcal{E}_G$ / $\mathcal{S}_G$	0.961		0.130		0.074			
26.400 / 8.700								
SIG* algorithm	$\uparrow$		$\uparrow$		$\square$		$\uparrow$	
$\mathcal{E}_G$ / $\mathcal{S}_G$	4.742×10^{-5}		7.153×10^{-7}		0.221		1.726×10^{-3}	
31.467 / 8.320								

problem 3 data sets. Conversely, neither of the two unmodified techniques were outperformed by any of the SOM-based algorithms throughout the experiments.

In almost all cases no SOM-based data mining technique outperformed both the other two SOM-based methods, indicating that no SOM-oriented approach was overwhelmingly superior in this analysis. The only exception was the Iris data set, where C4.5 configured HybridSOM was able to outperform both SIG* and HybridSOM combined with CN2. This was not observed within the overall training error analysis, where most data sets exhibited a statistically superior SOM-based algorithm.

The HybridSOM framework, when used with C4.5, performed the best out of the SOM-based techniques, and was the only SOM-based method to perform as well as the overall best performing techniques in one case (for the Pima Indians diabetes data set). In one further case, on the Iris plants data set, the C4.5-based HybridSOM configuration demonstrated average performance, outperforming two methods (HybridSOM with CN2 and SIG*) while being outperformed by the other two. In one more instance, for the third monk's problem, the C4.5 HybridSOM configuration outperformed one method (HybridSOM with CN2), while being outperformed by two (raw CN2 and C4.5). However, the C4.5-based framework was equivalent to the worst performing approaches in half of the data sets (the ionosphere, and monk's problems 1 and 2 data sets).

The SIG* algorithm, in turn, performed slightly better than the CN2 configured HybridSOM framework. The SIG* algorithm never outperformed all of the other approaches, but did succeed in being outperformed by only the CN2 algorithm on the first monk's problem. Additionally, SIG* was outperformed by only the two unmodified CN2 and C4.5 rule extraction algorithms in the case of the third monk's problem. In the remaining four data sets (namely Iris plants, ionosphere, monk's problem 2, and Pima Indians diabetes) SIG* was equivalent to the worst performing methods.

HybridSOM configured with CN2 was the worst performing approach out of those tested on the analyzed data sets. In almost all cases the CN2-based HybridSOM configuration was equivalent to the worst performing techniques for the data set in question. This was the case for Iris plants, ionosphere, monk's problem 1, monk's problem 2, and Pima Indians diabetes sets. In the remaining instance, the third monk's problem, the CN2 configured framework was outperformed by every other DM algorithm.

In support of the above analysis, the SOM-based approaches were again compared directly to one another, starting with a comparison between the two HybridSOM variants. Two data sets, namely the Iris plants and third monk's problem data sets, saw C4.5 configured HybridSOM outperform its CN2-based counterpart, while the remaining four data sets showed equivalent performance between the two configurations. The CN2-based framework never performed better than C4.5-based HybridSOM. The reader should note that the difference in performance was not as clearly delineated as was observed during the overall and misclassified data analysis of the training error.

When compared directly to SIG*, the C4.5 HybridSOM configuration performed better on two data sets (Iris and Pima Indians diabetes), while the techniques were equivalent in the remaining four cases. HybridSOM configured with the CN2 algorithm underperformed in comparison to the SIG* rule extraction algorithm in two of the six cases, namely on the first and third monk's problem data sets. In the remaining four investigated data sets, the two approaches were equivalent to one another.

Finally, a direct comparison was performed between each HybridSOM variant and the rule extractor with which the framework was configured. The CN2 variation on the HybridSOM framework was outperformed by unmodified CN2 in five of the six cases (on the Iris, ionosphere, and all three monk's problem data sets), whereas the two techniques performed equivalently on the Pima Indians diabetes data set. The basic C4.5 algorithm outperformed HybridSOM configured with C4.5 in four cases (on the Iris plants, ionosphere, monk's problem 1, and monk's problem 3 data sets), while the remaining two data sets (the second monk's problem, and the Pima Indians diabetes data set) showed equivalent performance between the two. The two unmodified approaches were therefore clearly better performing than their HybridSOM counterparts.

8.4.6.5 Test Set Error Due to Misclassified Data Examples

Tables 8.96 to 8.101 outline the classification error that was attributable only to misclassified test set data. The Iris plants data set is examined in Table 8.96, while Table 8.97 summarizes algorithmic performance on the ionosphere data set. The monk's problem data sets are considered in Tables 8.98, 8.99, and 8.100, respectively. Finally, Table 8.101 tabulates the performance achieved on the Pima Indians diabetes data set.

Table 8.96: Statistical comparison of the test set error due to misclassified data examples for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	2.667	6.915	5.333	10.417	34.000	20.443	14.667	13.830
Unmodified C4.5	□							
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	0.219							
5.333 / 10.417								
HybridSOM (CN2)	↑		↑					
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	1.192×10^{-7}		1.907×10^{-6}					
34.000 / 20.443								
HybridSOM (C4.5)	↑		↑		←			
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	1.526×10^{-5}		3.723×10^{-3}		7.771×10^{-4}			
14.667 / 13.830								
SIG* algorithm	↑		↑		□		↑	
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	8.345×10^{-7}		1.836×10^{-5}		0.384		4.232×10^{-3}	
27.333 / 18.557								

Table 8.97: Statistical comparison of the test set error due to misclassified data examples for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	7.576	7.581	9.697	8.586	18.788	8.247	20.000	9.369
Unmodified C4.5	□							
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	0.725							
9.697 / 8.586								
HybridSOM (CN2)	↑		↑					
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	2.384×10^{-7}		1.088×10^{-4}					
18.788 / 8.247								
HybridSOM (C4.5)	↑		↑		□			
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	2.235×10^{-7}		3.815×10^{-6}		0.741			
20.000 / 9.369								
SIG* algorithm	↑		↑		□		□	
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	1.057×10^{-3}		2.778×10^{-3}		0.107		0.095	
16.061 / 13.007								

Table 8.98: Statistical comparison of the test set error due to misclassified data examples for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	0.000	0.000	24.286	11.804	46.429	12.688	37.857	16.369
Unmodified C4.5								
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$ 1.863×10^{-9}						
24.286	11.804							
HybridSOM (CN2)								
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$ 1.863×10^{-9}	$\uparrow$ 5.513×10^{-7}					
46.429	12.688							
HybridSOM (C4.5)								
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$ 1.863×10^{-9}	$\uparrow$ 6.104×10^{-5}		$\square$ 0.048			
37.857	16.369							
SIG* algorithm								
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$ 1.863×10^{-9}	$\square$ 0.622		$\leftarrow$ 7.283×10^{-7}		$\square$ 5.268×10^{-3}	
28.333	10.185							

Table 8.99: Statistical comparison of the test set error due to misclassified data examples for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	22.857	11.317	32.381	10.895	32.857	12.805	32.857	12.805
Unmodified C4.5								
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$ 8.794×10^{-4}						
32.381	10.895							
HybridSOM (CN2)								
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$ 3.967×10^{-4}	$\square$ 0.903					
32.857	12.805							
HybridSOM (C4.5)								
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$ 1.375×10^{-4}	$\square$ 0.841		$\square$ 0.046			
32.857	12.805							
SIG* algorithm								
$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\uparrow$ 5.887×10^{-4}	$\square$ 0.919		$\square$ 0.112		$\square$ 0.047	
32.857	12.805							

Table 8.100: Statistical comparison of the test set error due to misclassified data examples for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	2.857	4.023	2.857	4.023	38.333	16.512	19.048	9.441
Unmodified C4.5	$\leftarrow$ *							
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	9.766×10^{-4}							
2.857 / 4.023								
HybridSOM (CN2)	$\uparrow$		$\uparrow$					
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	3.725×10^{-9}		1.863×10^{-9}					
38.333 / 16.512								
HybridSOM (C4.5)	$\uparrow$		$\uparrow$		$\leftarrow$			
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	7.451×10^{-9}		7.451×10^{-9}		1.151×10^{-6}			
19.048 / 9.441								
SIG* algorithm	$\uparrow$		$\uparrow$		$\leftarrow$		$\square$	
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	1.863×10^{-9}		1.863×10^{-9}		9.553×10^{-4}		0.012	
27.143 / 14.335								

* Means differ by 2.725×10^{-8}, while standard deviations differ by 3.837×10^{-8}, indicating practically insignificant result differences.

Table 8.101: Statistical comparison of the test set error due to misclassified data examples for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$	$\mathcal{E}_{GM}$	$\mathcal{S}_{GM}$
	26.667	8.683	24.667	8.604	30.267	9.017	26.400	8.700
Unmodified C4.5	$\square$							
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	0.175							
24.667 / 8.604								
HybridSOM (CN2)	$\square$		$\uparrow$					
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	0.040		2.356×10^{-3}					
30.267 / 9.017								
HybridSOM (C4.5)	$\square$		$\square$		$\square$			
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	0.961		0.130		0.074			
26.400 / 8.700								
SIG* algorithm	$\square$		$\uparrow$		$\square$		$\square$	
$\mathcal{E}_{GM}$ / $\mathcal{S}_{GM}$	6.752×10^{-3}		1.068×10^{-4}		0.694		0.048	
29.467 / 8.819								

The overall test set error for the third monk's problem, illustrated within Table 8.94, showed that the statistically significant difference between raw CN2 and C4.5 was not deemed practically significant using the criteria of Section 8.4.6.1. Similarly, Table 8.100 indicates that there was also no practically significant difference between these methods on the same data set in terms of the test error due to misclassification.

The results for the misclassification-based test error were almost identical to those reported for the overall test error. The only exception was for the Pima Indians diabetes data, where almost all the methods performed equivalently. The conclusions presented here are thus largely the same as those outlined in the previous section.

The CN2 algorithm was once again the best performing rule extractor tested on the experimental data sets. The performance interrelationships between CN2 and the other approaches for the misclassification-based test error were exactly the same as those observed for the overall test set error on all date sets, except for Pima Indians diabetes. In the latter case, instead of CN2 outperforming SIG*, the two approaches performed equivalently. CN2 was therefore not as clearly superior to SIG*.

Similarly, the general interrelationships between the unmodified C4.5 technique and the other rule extraction algorithms, in terms of the misclassified example test set error, were identical to those observed for the overall test set error. The C4.5 algorithm was thus determined to be the second best performing rule extraction method, trailing behind the basic CN2 algorithm, while outperforming the SOM-based rule extractors.

The interrelationships between the SOM-based rule extraction techniques and the basic CN2 and C4.5 algorithms were again the same as those that were observed for the overall test set classification error. The unmodified CN2 and C4.5 algorithms again outperformed all the SOM-based algorithms for half of the experimental data sets, and did not underperform in relation to any of the SOM-based approaches at all.

Again, no SOM-based technique showed overwhelmingly better performance than the other SOM-based methods. The performance relationships between the SOM-based techniques were exactly the same as those observed for the overall test error.

In relation to the other SOM-based rule extractors, C4.5-based HybridSOM exhibited the general performance relationships that were seen seen during the overall test set error analysis. The only deviation was for the Pima Indians diabetes data set, where

HybridSOM with C4.5 was equivalent to SIG*, and not superior. Therefore, although C4.5-based HybridSOM was again situated as the best performing SOM-based DM algorithm, this trend was not as clearly illustrated as for the overall test error.

The SIG* algorithm again performed slightly worse than C4.5-based HybridSOM, while producing better performance than HybridSOM configured with CN2. The general performance relationships of the SIG* algorithm were the same as those observed during the overall test set error analysis, except that the algorithm performed equivalently to CN2 and C4.5-based HybridSOM, rather than worse, for the Pima Indians diabetes data set. This means that SIG* performed slightly better in terms of test error due to misclassifications than was observed in the case of the overall test error.

The last SOM-based rule extractor, the HybridSOM framework configured with CN2, also had the same general performance interactions with the other algorithms as was noted during the overall test error assessment. The CN2 configured framework was thus the weakest algorithm in terms of the test set error due to misclassifications.

As in the analysis of the previous section, the direct relationships between the SOM-based methods were also analyzed. Directly comparing the two HybridSOM variations once again showed identical performance relationships between the two approaches. This confirmed that the C4.5-based HybridSOM variant outperformed the CN2 configured version, albeit not as decisively as observed during the training error analysis.

The direct relationships between SIG* and the CN2 version of HybridSOM were unchanged from the overall test error analysis, confirming that SIG* was also superior in this analysis. When directly comparing C4.5 configured HybridSOM and SIG*, however, the HybridSOM configuration only outperformed SIG* on the Iris data set, and the approaches performed indistinguishably in all other cases. Consequently, while the holistic analysis of the algorithms indicated that there was still a general performance advantage associated with C4.5-based HybridSOM, the direct comparison illustrated that neither of the approaches clearly outperformed the other. This suggests that the advantage was not as clear when the misclassification-based test error was concerned.

To supplement this analysis, each HybridSOM configuration was once again compared to the basic rule extraction algorithm with which it was combined. The exact same relationships that were observed during the analysis of the previous section were again

present during the analysis of the misclassification-based test error. The conclusion that the HybridSOM configurations underperformed against the base methods thus still holds.

8.4.6.6 Test Set Error Due to Unclassified Data Examples

Tables 8.102 to 8.107 consider the final measure related to the test set error, namely the test set classification error due to only unclassified data examples. Table 8.102 investigates the Iris plants data set, while the ionosphere data set is summarized in Table 8.103. Tables 8.104, 8.105, and 8.106 focus on the three monk's problems, respectively. Finally, Table 8.107 outlines the results achieved on the Pima Indians diabetes set.

In very much the same way as the training set error due to unclassified data examples demonstrated, the SIG* algorithm clearly underperformed in relation to all the other DM algorithms when the test set error due to unclassified data examples was considered for the data sets under investigation. Once again, the unmodified CN2 and C4.5 algorithms, as well as the HybridSOM configurations that used these two algorithms, did not leave any test set data examples unclassified. This was, of course, due to the default rules that the CN2 and C4.5 algorithms both incorporate into their rule set output.

The SIG* algorithm did not underperform as drastically as was observed within the analysis of the training error due to unclassified data examples. It was only observed in half of the data sets that SIG* performed worse in a statistically significant fashion. The cases in question were the Iris plants, first monk's problem, and Pima Indians diabetes data sets. For the remaining three instances, namely the ionosphere and last two monk's problem data sets, all the tested data mining algorithms performed equivalently to one another. In the case of the second and third monk's problems, the SIG* technique left no test set examples unclassified. The SIG* algorithm did fail to apply some classifications in the case of the ionosphere data set, although this number was negligible.

For the data sets that demonstrated an underperforming SIG* algorithm, the percentage of unclassified test set examples was again not exceptionally high. The SIG* algorithm left only approximately 2% of the test set examples unclassified when the algorithm was run on the Pima Indians diabetes data set, while just over 5% of the test set was unclassified in the first monk's problem data set. The Iris data set, however, saw almost 15% of the test set examples processed by SIG* remain without a classification.

Table 8.102: Statistical comparison of the test set error due to unclassified data examples for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5								
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ N/A						
0.000	0.000							
HybridSOM (CN2)								
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ N/A		□ N/A				
0.000	0.000							
HybridSOM (C4.5)								
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ N/A		□ N/A		□ N/A		
0.000	0.000							
SIG* algorithm								
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	↑ 3.815×10^{-6}		↑ 3.815×10^{-6}		↑ 3.815×10^{-6}		↑ 3.815×10^{-6}
14.667	13.830							

Table 8.103: Statistical comparison of the test set error due to unclassified data examples for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5								
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ N/A						
0.000	0.000							
HybridSOM (CN2)								
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ N/A		□ N/A				
0.000	0.000							
HybridSOM (C4.5)								
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ N/A		□ N/A		□ N/A		
0.000	0.000							
SIG* algorithm								
$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	□ 0.250		□ 0.250		□ 0.250		□ 0.250
0.909	2.774							

Table 8.104: Statistical comparison of the test set error due to unclassified data examples for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5	□ N/A							
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
HybridSOM (CN2)	□ N/A		□ N/A					
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
HybridSOM (C4.5)	□ N/A		□ N/A		□ N/A			
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
SIG* algorithm	↑ 6.104×10^{-5}		↑ 6.104×10^{-5}		↑ 6.104×10^{-5}		↑ 6.104×10^{-5}	
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
5.238 6.480								

Table 8.105: Statistical comparison of the test set error due to unclassified data examples for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5	□ N/A							
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
HybridSOM (CN2)	□ N/A		□ N/A					
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
HybridSOM (C4.5)	□ N/A		□ N/A		□ N/A			
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
SIG* algorithm	□ N/A		□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								

Table 8.106: Statistical comparison of the test set error due to unclassified data examples for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5	□ N/A							
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
HybridSOM (CN2)	□ N/A		□ N/A					
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
HybridSOM (C4.5)	□ N/A		□ N/A		□ N/A			
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
SIG* algorithm	□ N/A		□ N/A		□ N/A		□ N/A	
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								

Table 8.107: Statistical comparison of the test set error due to unclassified data examples for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$	$\mathcal{E}_{GU}$	$\mathcal{S}_{GU}$
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unmodified C4.5	□ N/A							
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
HybridSOM (CN2)	□ N/A		□ N/A					
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
HybridSOM (C4.5)	□ N/A		□ N/A		□ N/A			
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
0.000 0.000								
SIG* algorithm	↑ 2.441×10^{-4}		↑ 2.441×10^{-4}		↑ 2.441×10^{-4}		↑ 2.441×10^{-4}	
$\mathcal{E}_{GU}$ $\mathcal{S}_{GU}$								
2.000 2.519								

8.4.6.7 Total Number of Conditions per Rule Set

Tables 8.108 to 8.113 outline the first performance measure that characterizes rule set complexity, namely the total number of conditions per rule set. This measure is the most general of the complexity measures, and assesses overall complexity while the other complexity measures focus on specific aspects of the rule sets. Table 8.108 focuses on the Iris plants data set. Table 8.109 considers the performance on the ionosphere data set, while Tables 8.110, 8.111, and 8.112 summarize the three monk's problem data sets. Lastly, Table 8.113 outlines the results for the Pima Indians diabetes data set.

It should be noted that better performance in any complexity measure does not necessarily point to a superior approach. It is possible for rule sets with very few conditions to classify data inaccurately. The results presented here should thus be read in conjunction with the conclusions of the analysis, which are presented in Section 8.4.7.

HybridSOM configured with CN2 was overwhelmingly the superior approach when considering the total number of conditions per rule set that were produced by the assessed algorithms when tested on the experimental data sets. The framework configuration outperformed all the other approaches in five of the six data set cases, namely the Iris plants, ionosphere, monk's problems 1 and 3, and Pima Indians diabetes data sets. In the final case, on the third monk's problem, the two HybridSOM configurations were indistinguishable, and outperformed all the other data mining methods. Of course, in all cases HybridSOM with CN2 outperformed the unmodified CN2 method.

The C4.5 configuration of the HybridSOM framework performed substantially worse than the CN2-based configuration, while still producing better performance than SIG* and the basic CN2 and C4.5 implementations. The C4.5 and CN2 configurations performed equivalently on the second monk's problem, with both outperforming all the other rule extraction methods. In two cases (the ionosphere and monk's problem 1 data sets) C4.5 configured HybridSOM was outperformed by the CN2-based variant, but performed better than all the other methods. In the remaining four data sets (Iris, monk's problem 3, and Pima Indians diabetes), HybridSOM with C4.5 performed equivalently to algorithms that were outperformed by only CN2-based HybridSOM.

The unmodified CN2 algorithm performed worse than both the HybridSOM configurations, but better than the C4.5 and SIG* techniques. CN2 did not perform better

Table 8.108: Statistical comparison of the total number of conditions per rule set for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$
	4.833	0.461	5.000	0.000	2.000	0.000	4.733	0.691
Unmodified C4.5	□							
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	0.125							
5.000 / 0.000								
HybridSOM (CN2)	←		←					
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.863×10^{-9}		1.863×10^{-9}					
2.000 / 0.000								
HybridSOM (C4.5)	□		□		↑			
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	0.656		0.125		1.863×10^{-9}			
4.733 / 0.691								
SIG* algorithm	↑		↑		↑		↑	
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}	
16.000 / 0.000								

Table 8.109: Statistical comparison of the total number of conditions per rule set for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$
	4.967	0.183	5.000	0.000	2.567	0.679	3.133	0.507
Unmodified C4.5	□							
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.000							
5.000 / 0.000								
HybridSOM (CN2)	←		←					
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.863×10^{-9}		1.863×10^{-9}					
2.567 / 0.679								
HybridSOM (C4.5)	←		←		↑			
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	7.451×10^{-9}		7.451×10^{-9}		1.957×10^{-3}			
3.133 / 0.507								
SIG* algorithm	↑		↑		↑		↑	
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}	
1 491.467 / 614.011								

Table 8.110: Statistical comparison of the total number of conditions per rule set for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$
	8.000	0.000	3.000	0.000	1.000	0.000	1.933	1.015
Unmodified C4.5	$\leftarrow$ 1.863×10^{-9}							
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$								
3.000 / 0.000								
HybridSOM (CN2)	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}					
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$								
1.000 / 0.000								
HybridSOM (C4.5)	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 3.052×10^{-5}		$\uparrow$ 1.221×10^{-4}			
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$								
1.933 / 1.015								
SIG* algorithm	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}	
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$								
1 737.400 / 342.554								

Table 8.111: Statistical comparison of the total number of conditions per rule set for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$
	24.667	13.520	56.600	7.356	1.000	0.000	1.000	0.000
Unmodified C4.5	$\uparrow$ 1.863×10^{-9}							
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$								
56.600 / 7.356								
HybridSOM (CN2)	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}					
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$								
1.000 / 0.000								
HybridSOM (C4.5)	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}		$\square$ N/A			
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$								
1.000 / 0.000								
SIG* algorithm	$\uparrow$ 1.157×10^{-3}		$\leftarrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}	
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$								
34.000 / 0.000								

Table 8.112: Statistical comparison of the total number of conditions per rule set for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$
	3.000	0.000	5.000	0.000	1.733	0.980	3.067	0.365
Unmodified C4.5	$\uparrow$							
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.863×10^{-9}							
5.000 / 0.000								
HybridSOM (CN2)	$\leftarrow$		$\leftarrow$					
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	3.815×10^{-6}		1.863×10^{-9}					
1.733 / 0.980								
HybridSOM (C4.5)	$\square$		$\leftarrow$		$\uparrow$			
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.000		3.725×10^{-9}		3.815×10^{-6}			
3.067 / 0.365								
SIG* algorithm	$\uparrow$		$\uparrow$		$\uparrow$		$\uparrow$	
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}	
1 556.067 / 584.961								

Table 8.113: Statistical comparison of the total number of conditions per rule set for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$	$\mathcal{E}_{FT}$	$\mathcal{S}_{FT}$
	2.933	0.365	3.500	1.592	1.667	0.606	3.200	0.610
Unmodified C4.5	$\square$							
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	0.125							
3.500 / 1.592								
HybridSOM (CN2)	$\leftarrow$		$\leftarrow$					
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.490×10^{-8}		7.451×10^{-9}					
1.667 / 0.606								
HybridSOM (C4.5)	$\square$		$\square$		$\uparrow$			
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	0.125		0.438		3.725×10^{-9}			
3.200 / 0.610								
SIG* algorithm	$\uparrow$		$\uparrow$		$\uparrow$		$\uparrow$	
$\mathcal{E}_{FT}$ / $\mathcal{S}_{FT}$	1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}	
315.667 / 126.061								

than all of the other approaches on any of the data sets, and also did not jointly perform the best with one or more of the other algorithms. CN2 was, however, outperformed by only CN2-based HybridSOM in three cases (Iris, monk's problem 3, and Pima Indians diabetes), but performed equivalently to other algorithms in all of these cases. For the second monk's problem, CN2 outperformed two approaches, while performing worse than two other algorithms. On the first monk's problem data set, CN2 only performed better than the SIG* method. Finally, CN2 and C4.5 performed equivalently in the case of the ionosphere data set, and both methods outperformed SIG* only.

The best performance achieved by the basic C4.5 technique was on the Iris Plants and Pima Indians diabetes data sets, where C4.5 was equivalent to CN2 and C4.5-based HybridSOM, and was only outperformed by CN2 configured HybridSOM. The first monk's problem saw C4.5 outperforming two approaches and performing worse than the other two algorithms. Only the SIG* algorithm was outperformed by C4.5 for the third monk's problem. The same was true for the ionosphere data set, although CN2 was indistinguishable from C4.5 in this case. Finally, for monk's problem 2, C4.5 was outperformed by all the other rule extraction algorithms that were explored.

Finally, SIG* clearly exhibited the worst performance out of all the algorithms. Both the means and standard deviations for this measure were substantially higher than the other algorithms in most cases, indicating both poor and erratic performance. The best performance achieved by SIG* was on the second monk's problem data set, where the approach outperformed only the worst performing C4.5 algorithm. On all other data sets SIG* was outperformed by every other analyzed DM technique.

Two direct comparisons between the SOM-based algorithms were also conducted. The two HybridSOM methods were compared first, and the second monk's problem saw the two approaches performing equivalently. On all other data sets, the CN2-based configuration produced better results than the C4.5 configured version.

The final direct comparison for this performance measure was between each of the HybridSOM framework variations and the algorithm with which the framework configuration in question was combined. HybridSOM configured with C4.5 outperformed unmodified C4.5 in four cases (for the ionosphere data set and all the monk's problems), while both were equivalent in two cases (on the Iris plants and Pima Indians diabetes

data sets). Additionally, the CN2 variant of HybridSOM was observed to produce better performance than the unmodified CN2 algorithm on ever data set in the analysis.

8.4.6.8 Number of Rules per Rule Set

Tables 8.114 to 8.119 provide the summarized statistics for the analysis performed on the complexity performance measure that represents the number of rules making up a rule set. Table 8.114 focuses on the Iris data set, while Table 8.115 outlines the ionosphere data set. Tables 8.116, 8.117, and 8.118 respectively present results for the three monk's problems, and Table 8.119 presents results for the Pima Indians diabetes data set.

The CN2 configured HybridSOM framework once again definitively outperformed all the other tested approaches. On four of the six data sets, the HybridSOM and CN2 combination outperformed all the other techniques. The data sets in question were the ionosphere, the first and third monk's problem, and the Pima Indians diabetes sets. In the remaining cases, namely the Iris plants data set and the second monk's problem, the CN2-based framework performed equivalently to the best performing algorithms.

The second best performing algorithm in the analysis was the basic CN2 algorithm. This technique demonstrated performance exceeded by only one algorithm in two of the six cases. The data sets in question were the monk's problem 3 and Pima Indians diabetes data sets. In another two instances, on the Iris plants and second monk's problem data sets, the performance of CN2 was surpassed by a single group of equivalently performing algorithms. These groupings contained two techniques in the former case, and three methods in the latter. CN2 was observed to outperform only SIG* in the remaining two instances, namely the ionosphere and first monk's problem benchmark data sets.

The HybridSOM framework set up with C4.5 demonstrated relatively erratic behavior over the analyzed data sets, producing fairly good results in some cases, and particularly poor comparative performance on other data sets. Nevertheless, this HybridSOM variant generally performed better than unmodified C4.5 and SIG*. While never outright performing better than all the other tested algorithms, the second monk's problem saw the C4.5-based framework perform equivalently to two other algorithms, which jointly outperformed both C4.5 and CN2. On the ionosphere and first monk's problem data sets, C4.5-based HybridSOM was outperformed by only the CN2-based

Table 8.114: Statistical comparison of the number of rules per rule set for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$
	3.000	0.000	4.000	0.000	2.000	0.000	3.867	0.346
Unmodified C4.5	$\uparrow$							
$\mathcal{E}_R$ / $\mathcal{S}_R$	1.863×10^{-9}							
4.000 / 0.000								
HybridSOM (CN2)	$\leftarrow$		$\leftarrow$					
$\mathcal{E}_R$ / $\mathcal{S}_R$	1.863×10^{-9}		1.863×10^{-9}					
2.000 / 0.000								
HybridSOM (C4.5)	$\uparrow$		$\square$		$\uparrow$			
$\mathcal{E}_R$ / $\mathcal{S}_R$	2.980×10^{-8}		0.125		1.863×10^{-9}			
3.867 / 0.346								
SIG* algorithm	$\leftarrow$		$\leftarrow$		$\square$		$\leftarrow$	
$\mathcal{E}_R$ / $\mathcal{S}_R$	1.863×10^{-9}		1.863×10^{-9}		N/A		1.863×10^{-9}	
2.000 / 0.000								

Table 8.115: Statistical comparison of the number of rules per rule set for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$
	4.967	0.183	4.000	0.000	2.433	0.504	3.067	0.254
Unmodified C4.5	$\leftarrow$							
$\mathcal{E}_R$ / $\mathcal{S}_R$	3.725×10^{-9}							
4.000 / 0.000								
HybridSOM (CN2)	$\leftarrow$		$\leftarrow$					
$\mathcal{E}_R$ / $\mathcal{S}_R$	1.863×10^{-9}		1.863×10^{-9}					
2.433 / 0.504								
HybridSOM (C4.5)	$\leftarrow$		$\leftarrow$		$\uparrow$			
$\mathcal{E}_R$ / $\mathcal{S}_R$	3.725×10^{-9}		7.451×10^{-9}		7.629×10^{-6}			
3.067 / 0.254								
SIG* algorithm	$\uparrow$		$\uparrow$		$\uparrow$		$\uparrow$	
$\mathcal{E}_R$ / $\mathcal{S}_R$	1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}	
16.600 / 3.440								

Table 8.116: Statistical comparison of the number of rules per rule set for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$
	5.000	0.000	3.000	0.000	1.000	0.000	1.933	1.015
Unmodified C4.5								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\leftarrow$ 1.863×10^{-9}						
3.000	0.000							
HybridSOM (CN2)								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}				
1.000	0.000							
HybridSOM (C4.5)								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 3.052×10^{-5}		$\uparrow$ 1.221×10^{-4}		
1.933	1.015							
SIG* algorithm								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}
83.433	8.169							

Table 8.117: Statistical comparison of the number of rules per rule set for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$
	8.900	4.950	15.033	1.732	1.000	0.000	1.000	0.000
Unmodified C4.5								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\uparrow$ 3.498×10^{-6}						
15.033	1.732							
HybridSOM (CN2)								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}				
1.000	0.000							
HybridSOM (C4.5)								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}		$\square$ N/A		
1.000	0.000							
SIG* algorithm								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}		$\square$ N/A		$\square$ N/A
1.000	0.000							

Table 8.118: Statistical comparison of the number of rules per rule set for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$
	2.000	0.000	4.000	0.000	1.367	0.490	3.033	0.183
Unmodified C4.5								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\uparrow$ 1.863×10^{-9}						
4.000	0.000							
HybridSOM (CN2)								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\leftarrow$ 3.815×10^{-6}		$\leftarrow$ 1.863×10^{-9}				
1.367	0.490							
HybridSOM (C4.5)								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\uparrow$ 1.863×10^{-9}		$\leftarrow$ 3.725×10^{-9}		$\uparrow$ 1.863×10^{-9}		
3.033	0.183							
SIG* algorithm								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}
24.567	3.390							

Table 8.119: Statistical comparison of the number of rules per rule set for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$	$\mathcal{E}_R$	$\mathcal{S}_R$
	1.967	0.183	3.233	0.728	1.600	0.498	3.100	0.305
Unmodified C4.5								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\uparrow$ 1.863×10^{-9}						
3.233	0.728							
HybridSOM (CN2)								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\leftarrow$ 9.766×10^{-4}		$\leftarrow$ 1.863×10^{-9}				
1.600	0.498							
HybridSOM (C4.5)								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\uparrow$ 1.863×10^{-9}		$\square$ 0.438		$\uparrow$ 1.863×10^{-9}		
3.100	0.305							
SIG* algorithm								
$\mathcal{E}_R$	$\mathcal{S}_R$	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}
12.700	3.196							

framework. HybridSOM with C4.5 achieved average performance on the third monk's problem, by outperforming two approaches while performing worse than two others. For the Pima Indians diabetes data set, C4.5-oriented HybridSOM performed equivalently to one method, where both approaches only outperformed the worst performing SIG* technique. Lastly, C4.5 configured HybridSOM performed the worst out of all the methods tested on the Iris data set, in conjunction with the equivalent C4.5 algorithm.

Next, the unmodified C4.5 algorithm was investigated, and found to broadly underperform in relation to all the algorithms, with the exception of SIG*. The best performance produced by C4.5 was on the ionosphere and first monk's problem data sets, where C4.5 exhibited the median performance by producing better results than two approaches and worse results in relation to two more on both sets. Unmodified C4.5 only succeeded in outperforming SIG* in the case of the third monk's problem. The Pima Indians diabetes data set found C4.5 performing equivalently to C4.5-based HybridSOM, where both algorithms outperformed only the SIG* algorithm. The C4.5 method performed worse than every other algorithm on the monk's problem 2 data set. Finally, C4.5 again performed equivalently to C4.5 configured HybridSOM on the Iris data set, however neither algorithm outperformed any other rule extractors.

The worst performing approach was the SIG* technique. In only two instances, on the Iris and monk's problem 2 data sets, SIG* performed as well as the other approaches that performed the best in terms of rule count. In the majority of cases, however, SIG* was outperformed by every other method. This was the case four times, under the ionosphere, monk's problems 1 and 3, and Pima Indians diabetes data sets.

Several direct comparisons between pairs of approaches were again performed, to further strengthen the results of the analysis. Firstly, the two HybridSOM configurations were compared against one another. CN2 configured HybridSOM outperformed the C4.5-based configuration five times, on the Iris plants, ionosphere, first and third monk's problem, and Pima Indians diabetes data sets. In the only outstanding case, for monk's problem 2, the C4.5-based method outperformed HybridSOM with CN2. These results support the previous observation that the CN2 configuration was superior.

Next, the two HybridSOM configurations were compared to the SIG* algorithm. In four instances (the ionosphere, monk's problem 1 and 3, and Pima Indians diabetes data

sets) the CN2-based HybridSOM configuration outperformed SIG*, while the opposite was never true. The two techniques were indistinguishable from one another in the remaining two cases, namely the Iris plants data set and the second monk's problem. The C4.5 HybridSOM configuration was also compared directly to SIG*. The HybridSOM configuration also outperformed SIG* on the same four data sets that the CN2-based HybridSOM variant achieved the same result. The opposite relationship held for the Iris data set only, and C4.5-based HybridSOM and SIG* had equivalent performance characteristics on the remaining data set, namely the second monk's problem. Once again, these results supported the assessment presented earlier in this section.

In the final comparison the two HybridSOM configurations were each contrasted to the rule extraction algorithm on which they were respectively based. HybridSOM with CN2 outperformed the unmodified CN2 algorithm on all six experimental data sets. The C4.5-oriented HybridSOM configuration outperformed the basic C4.5 implementation four times, for the ionosphere data set and all the monk's problems. The opposite relationship was never the case, although the Iris plants and Pima Indians diabetes data sets exhibited equivalent performance between HybridSOM combined with C4.5 and the raw C4.5 algorithm. Each HybridSOM configuration thus tended to produce fewer rules per rule set than the basic algorithm with which the framework was combined.

8.4.6.9 Average Number of Conditions per Rule

The results for the final performance measure, namely the average number of conditions per rule, are summarized in Tables 8.120 to 8.125. This measure focuses on the typical complexity of the rules contained in a rule set. The Iris plants data set is investigated in Table 8.120, while Table 8.121 focuses on the ionosphere data set. The three monk's problem data sets are dealt with in Tables 8.122, 8.123, and 8.124, respectively. The Pima Indians diabetes data set was the last to be investigated, in Table 8.125.

For this performance measure there was little observable difference between the performance of the two versions of HybridSOM, within the bounds of this experimental analysis. Focusing first on the CN2 configuration, the approach outperformed all the other algorithms on the Iris data set, and was outperformed by only one method while performing equivalently to another for the third monk's problem. On the other hand, the

Table 8.120: Statistical comparison of the average number of conditions per rule for the rule extraction algorithms executed on the Iris plants data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$
	1.611	0.154	1.250	0.000	1.000	0.000	1.217	0.086
Unmodified C4.5	$\leftarrow$ 1.490×10^{-8}							
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.250 / 0.000								
HybridSOM (CN2)	$\leftarrow$ 3.725×10^{-9}		$\leftarrow$ 1.863×10^{-9}					
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.000 / 0.000								
HybridSOM (C4.5)	$\leftarrow$ 9.313×10^{-9}		$\square$ 0.125		$\uparrow$ 2.980×10^{-8}			
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.217 / 0.086								
SIG* algorithm	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}	
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
8.000 / 0.000								

Table 8.121: Statistical comparison of the average number of conditions per rule for the rule extraction algorithms executed on the ionosphere data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$
	1.000	0.000	1.250	0.000	1.050	0.132	1.017	0.063
Unmodified C4.5	$\uparrow$ 1.863×10^{-9}							
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.250 / 0.000								
HybridSOM (CN2)	$\square$ 0.125		$\leftarrow$ 4.172×10^{-7}					
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.050 / 0.132								
HybridSOM (C4.5)	$\square$ 0.500		$\leftarrow$ 7.451×10^{-9}		$\square$ 0.250			
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.017 / 0.063								
SIG* algorithm	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}	
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
86.067 / 20.250								

Table 8.122: Statistical comparison of the average number of conditions per rule for the rule extraction algorithms executed on the monk's problem 1 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$
	1.600	0.000	1.000	0.000	1.000	0.000	1.000	0.000
Unmodified C4.5	$\leftarrow$ 1.863×10^{-9}							
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.000 / 0.000								
HybridSOM (CN2)	$\leftarrow$ 1.863×10^{-9}		$\square$ N/A					
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.000 / 0.000								
HybridSOM (C4.5)	$\leftarrow$ 1.863×10^{-9}		$\square$ N/A		$\square$ N/A			
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.000 / 0.000								
SIG* algorithm	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}	
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
20.713 / 2.759								

Table 8.123: Statistical comparison of the average number of conditions per rule for the rule extraction algorithms executed on the monk's problem 2 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$
	2.754	0.270	3.761	0.115	1.000	0.000	1.000	0.000
Unmodified C4.5	$\uparrow$ 1.863×10^{-9}							
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
3.761 / 0.115								
HybridSOM (CN2)	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}					
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.000 / 0.000								
HybridSOM (C4.5)	$\leftarrow$ 1.863×10^{-9}		$\leftarrow$ 1.863×10^{-9}		$\square$ N/A			
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
1.000 / 0.000								
SIG* algorithm	$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}		$\uparrow$ 1.863×10^{-9}	
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$								
34.000 / 0.000								

Table 8.124: Statistical comparison of the average number of conditions per rule for the rule extraction algorithms executed on the monk's problem 3 data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$
	1.500	0.000	1.250	0.000	1.183	0.245	1.008	0.046
Unmodified C4.5	$\leftarrow$							
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$	1.863×10^{-9}							
1.250 / 0.000								
HybridSOM (CN2)	$\leftarrow$		$\square$					
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$	3.815×10^{-6}		0.200					
1.183 / 0.245								
HybridSOM (C4.5)	$\leftarrow$		$\leftarrow$		$\leftarrow$			
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$	1.863×10^{-9}		3.725×10^{-9}		9.766×10^{-4}			
1.008 / 0.046								
SIG* algorithm	$\uparrow$		$\uparrow$		$\uparrow$		$\uparrow$	
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$	1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}	
63.172 / 22.066								

Table 8.125: Statistical comparison of the average number of conditions per rule for the rule extraction algorithms executed on the Pima Indians diabetes data set.

	Unmodified CN2		**Unmodified C4.5**		**HybridSOM (CN2)**		**HybridSOM (C4.5)**	
	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$	$\mathcal{E}_{FA}$	$\mathcal{S}_{FA}$
	1.483	0.091	1.049	0.155	1.033	0.127	1.025	0.076
Unmodified C4.5	$\leftarrow$							
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$	1.490×10^{-8}							
1.049 / 0.155								
HybridSOM (CN2)	$\leftarrow$		$\square$					
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$	1.490×10^{-8}		0.875					
1.033 / 0.127								
HybridSOM (C4.5)	$\leftarrow$		$\square$		$\square$			
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$	3.725×10^{-9}		0.438		1.000			
1.025 / 0.076								
SIG* algorithm	$\uparrow$		$\uparrow$		$\uparrow$		$\uparrow$	
$\mathcal{E}_{FA}$ / $\mathcal{S}_{FA}$	1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}		1.863×10^{-9}	
23.885 / 4.197								

C4.5 configuration outperformed all other methods for the third monk's problem benchmark data. When tested on the Iris data set, C4.5-based HybridSOM performed better than half of other approaches, while underperforming in relation to the other half. In four instances both HybridSOM variations together performed equivalently to the other best performing techniques. The data sets that produced this equivalent performance were ionosphere, the first two monk's problems, and Pima Indians diabetes.

Trailing behind the two HybridSOM variants, in terms of the average conditions per rule, were the basic CN2 and C4.5 algorithms. Both approaches produced relatively similar results, although C4.5 produced slightly better overall performance.

The best performance for C4.5 was seen on the monk's problem 1 and Pima Indians diabetes data sets. In these cases, C4.5 had performance equivalent to both HybridSOM configurations, which were also jointly the best performers. On the Iris data set, C4.5 performed better than all the other techniques, except for CN2-oriented HybridSOM. C4.5 executed on the third monk's problem was also outperformed by only one other algorithm, but produced equivalent performance to one technique as well. For the ionosphere data set, C4.5 was outperformed by a single group of three equivalent algorithms, and outperformed the SIG* approach. C4.5 performed relatively poorly on the second monk's problem data set, producing better performance than only SIG*.

Turning one's attention to the unmodified CN2 technique, it is clear that the best performance for the approach was seen on the ionosphere data set, where CN2 performed equivalently to the HybridSOM versions, and all three algorithms outperformed the other rule extraction methods. For the second monk's problem, CN2 took up the middle ground of performance, underperforming against the two HybridSOM configurations, but performing better than the other two approaches. The third monk's problem and Pima Indians diabetes data sets both saw CN2 being outperformed by one group of three equivalently performing techniques, while outperforming SIG*. Finally, CN2 performed poorly on the Iris data set, where the algorithm outperformed only SIG*.

The SIG* algorithm was unequivocally the overall worst performing algorithm within the investigation. The trend of poor SIG* performance was also observed for the other complexity measures. SIG* underperformed in relation to every other method on all six of the data sets used during this analysis. The SIG* implementation also produced

substantially more conditions per rule than the other techniques, particularly on the ionosphere and third monk's problem data sets. The standard deviations of the performance measure were also higher, indicating more erratic results overall.

Finally, to strengthen support for the above findings, three sets of direct comparison between pairs of algorithms were performed. Firstly, the two HybridSOM configurations were compared to one another. The configuration that used CN2 performed equivalently to the C4.5-based approach in four cases, on the ionosphere, monk's problem 1 and 2, and Pima Indians diabetes data sets. On the Iris data set, the CN2-based technique performed better than the C4.5 configuration, while the opposite was true for the third monk's problem. This consolidates the earlier observation that the HybridSOM variants performed relatively similarly in terms of overall performance characteristics.

Next, CN2 and C4.5 were compared directly. The C4.5 algorithm performed better than CN2 four times, on the Iris, monk's problems 1 and 3, and Pima Indians diabetes data sets. In the two other instances, for the ionosphere data set and the second monk's problem, the opposite was true. These results confirm that C4.5 was superior, which was not as clear when the overall performance of each method was considered.

The final comparison was between each HybridSOM configuration and the rule extraction algorithm used to configure the framework in question. CN2-based HybridSOM outperformed the unmodified CN2 algorithm in five of the six cases, for the Iris plants and Pima Indians diabetes data sets, and all of the monk's problems. On the ionosphere data set, both techniques demonstrated indistinguishable performance. Considering the C4.5 based HybridSOM method, basic C4.5 is outperformed by C4.5-based HybridSOM in three instances, namely the ionosphere data set, and monk's problems 2 and 3. The opposite relationship occurred only for the Iris data set. The two approaches performed equivalently in two cases, for the first monk's problem, and Pima Indians diabetes set. Both HybridSOM configurations thus had better performance, although the C4.5-based framework was somewhat less clearly superior to the unmodified C4.5 method.

8.4.7 Discussion

The focus now shifts to the general findings stemming from the experimental results that were presented in the previous section. This section discusses, in fairly general terms,

the strengths and shortcomings associated with the rule extraction techniques, all of which were observed within the scope of the experimental work that was performed. Several practical recommendations, which follow directly from these findings, are also presented. The results reported on in the previous sections paint an interesting picture of the relative performance characteristics of the investigated approaches.

Of particular interest were the performance relationships of the HybridSOM configurations in relation to one another, and the basic CN2 and C4.5 algorithms with which the frameworks were combined. There was also a very clear interplay between the classification accuracy measures and the indicators of rule set complexity. These performance aspects and interrelationships are discussed in detail through the remainder of this discussion, in the context of each investigated SOM-based DM technique.

The SOM-based techniques generally produced significantly worse classification accuracy than the unmodified CN2 and C4.5 algorithms, both in terms of the overall training and test error, and the training and test error due to only misclassifications. This result stems from the fact that interpolating units are typically introduced into the data model created by emergent feature maps. These neurons do not directly represent part of the data model, but appear as a separation between emergent clusters. The introduction of interpolating units is useful in the context of EDA, because these neurons delineate the extent of emergent clusters. Interpolating units, however, introduce what amounts to noise into SOM models. This noise is typically not problematic during EDA, because visualizations tend to focus on general map characteristics, rather than finer details. For SOM-based data mining, however, rules are extracted directly from the detailed data model of the map. Because the classification performance of any rule extraction algorithm is known to degrade in the presence of noise [276], it is very likely that all SOM-based data mining techniques operate with this performance disadvantage.

Interestingly, when considering overall and misclassification-based training and test set accuracy, the basic CN2 algorithm generally outperformed unmodified C4.5, whereas the C4.5-based HybridSOM configuration tended to perform better than HybridSOM with CN2. Sahami [209] suggests that C4.5 performs better than CN2 in the presence of noise, due to several pruning strategies incorporated in the former algorithm. This research proposes that the mechanism underlying the stronger performance of C4.5 con-

figured HybridSOM is the ability of C4.5 to better handle noise. It has been noted that a SOM-based data model introduces noise through the inclusion of interpolating units. If an algorithm performs well in the presence of noise, as C4.5 does, it is likely that the algorithm performance will degrade more gracefully when extracting knowledge from a noisy SOM model. Conversely, a method that deals poorly with noise, like CN2, is likely to exhibit more drastic performance degradation when combined with a SOM. Further investigation of this hypothesis is, however, deferred to future analysis.

Considering overall and misclassification error on both the training and test data sets, the SIG* algorithm produced relatively poor performance. The approach did, however, succeed in holding its own in relation to the other SOM-based rule extraction approaches. SIG* did typically succeed at outperforming the CN2-based HybridSOM framework, but tended to underperform with respect to C4.5 configured HybridSOM.

Focusing on the training and test error that resulted from only unclassified data, all the rule extraction algorithms other than SIG* never left any data examples unclassified. This was due to the techniques catering for the inclusion of default rules. The inclusion of inaccurate default rules can also be ruled out, due to the strong performance of the CN2 and C4.5 algorithms when only misclassified data was considered.

The SIG* algorithm, in contrast, is capable of leaving data examples unclassified. In several instances SIG* produced statistically significant higher error rates due to unclassified examples. Even so, there were also a number of cases in which SIG* performed equivalently to the other approaches. The magnitude of the effect of unclassified data is therefore case dependent, varying between data sets. Nonetheless, the experiments demonstrated the potential danger associated with the lack of SIG* support for default rules. Future research should therefore conduct further testing, to determine the most sensible means of including such default rules in SIG* implementations.

The simplest rule sets, in terms of all three complexity measures, were produced by the two HybridSOM versions. This means that HybridSOM produced rule sets that tended to consist of fewer rules, where each rule also contained fewer conditions on average. Between the two HybridSOM variants, the CN2 implementation tended to produce the least complex output. When these findings are interpreted in conjunction with classification error performance, however, it becomes clear that the typically simpler rule

sets produced generally poor classification performance. This is further highlighted by the fact that the CN2 configuration of HybridSOM was responsible for both the simplest rule sets and the worst training and test set classification errors, in terms of overall error as well as the error attributed to only classification mistakes. Similarly, the C4.5 configuration was generally the better performing of the HybridSOM implementations, and also produced slightly more complex rule sets. The output rule sets produced by the HybridSOM framework therefore tended to be pathologically simple.

In contrast to the HybridSOM configurations, the basic CN2 and C4.5 rule extractors produced more complex rule sets. This complexity is linked to the generally significantly superior test and training classification performance of these rule extractors.

Finally, when focusing upon the SIG* algorithm, it is readily apparent that the technique generated exceptionally complex rule sets in comparison to the other algorithms. This complexity was again witnessed in terms of all three complexity measures, meaning that SIG* rule sets were much larger, and that the rules themselves were more complex. Taken in conjunction with the general classification accuracy observed for SIG*, it is apparent that the technique had relatively poor accuracy performance, despite the much greater complexity of the algorithmic output. In the case of SIG*, therefore, there was a great deal of redundancy in the rule sets produced by the algorithm.

There are a number of reasons for the increased complexity, which were suggested in Section 7.3.5. One is the fact that SIG* develops a characterizing rule for every discovered emergent cluster, even if a cluster is insignificant or uninteresting. Secondly, the attribute marking procedure for characterizing rules guarantees that every attribute is included in the final rule set, regardless of the importance of the attribute. Another contributor to rule complexity is the rule differentiation method, which is too aggressive, and tends to add a large number of additional conditions to both overlapping rule pairs, without testing whether these conditions are already included in the rules. Another factor is the conversion of the SIG* rule sets into a list of production rules. This conversion was, of course, essential to the experimental work presented here, but it should be noted that unconverted SIG* rules will be more succinct. It is important to realize, however, that the non-standard nature of unmodified SIG* rule sets presents potential challenges in a practical setting, both for rule analysis and integration into expert systems.

Many of these problems are easy to address. Firstly, it is possible to employ feature selection methods to only add highly significant attributes to characterizing and differentiating rules. Insignificant emergent clusters should also not be considered for development into rules. The rule differentiation procedure is also an area to focus on in order to simplify rule specializations. Possible optimizations include limiting rule specialization to only rule pairs with greater overlap, only specialize one of the rules in an overlapping pair, and ensuring that duplicate or subsumed conditions are never added to characterizing rules. These improvements are, however, left to further research.

Given the above discussion, SOM-based data mining is not recommended for small to moderately sized data sets, because stand-alone rule extractors will produce adequate results on such data. SOM-based methods should also not be used when classification accuracy is of primary importance. The use of SOM-based data mining is, however, sensible when any of the advantages described within Section 7.6 are desirable properties.

8.5 Summary

This chapter presented a detailed discussion of the results produced during the experimental analysis of this dissertation. The discussion investigated both supervised neuron labeling techniques and SOM-based data mining methods in conjunction with the classical CN2 and C4.5 rule extraction algorithms. Section 8.1 introduced the experimental procedure that was followed for by both broad analyses. The benchmark data sets that were used as a basis for both technique comparisons were described in Section 8.2. The results achieved for the supervised neuron labeling and data mining algorithm investigations were outlined and discussed in Sections 8.3 and 8.4, respectively.

The final chapter of this dissertation presents the concluding remarks for this research work. The main findings of the research are listed and discussed, with a focus on the novel contributions made by this work. The chapter also focuses upon future directions that research emanating from the work presented within this dissertation will take.

Chapter 9

Conclusions

This chapter discusses the conclusions arrived at by this dissertation. The objectives that this research set out to achieve are reiterated in Section 9.1, while the methodology used in support of the research objectives is summarized in Section 9.2. The main novel contributions introduced by this research are summarized in Section 9.3, while the experimental findings of this research are summarized under Section 9.4. Finally, future research topics that are suggested by this work are enumerated under Section 9.5.

9.1 Objectives

To contextualize the discussion of this conclusion, the research objectives of this research work are briefly reiterated. These objectives are summarized as follows:

- To broadly contextualize, in a general and real-world practical situation, the place of the SOM algorithm within the related fields of EDA and DM.
- To describe the most prevalent categories of SOM-based EDA and the related data visualization techniques, which are possible in a practical situation.
- To describe and analyze approaches for labeling SOM neurons, due to the fact that such labeling is important for many EDA and DM techniques based on SOMs.
- To investigate existing DM methods based on SOMs, which can be found in the literature, and explore the possibility of new SOM-based DM techniques.

9.2 Methodology

The methodology that was used within this dissertation is reiterated, to provide context for this discussion. The components of the methodology included the following:

- A thorough literature survey covering SOMs, the existing neuron labeling methods for SOMs, and the SOM-based EDA and DM approaches that have been developed.
- Algorithms and working prototypes for the new SOM-based neuron labeling and DM techniques that have been proposed by this dissertation's research work.
- Experimental analyses covering several performance categories linked to the existing and proposed neuron labeling and DM techniques based on SOMs.

9.3 Summary of Contributions

The first contribution of this work was the specification of several broad categories, which classify the SOM-based EDA techniques that Chapter 4 discussed. These categories were described in Section 4.5, and encompass characterization, feature selection, sensitivity analysis, interpolation, and trend analysis. It is also possible to use these groupings for the categorization of new SOM-based EDA techniques, as well as to serve as a basis for expansion should radically new directions for SOM-based EDA be developed.

Secondly, this research presented a detailed survey of neuron labeling approaches for SOMs, which had not previously been systematically taxonomized in this way. This dissertation also thoroughly discussed the general behavior associated with each identified neuron labeling method, as well as the broad tradeoffs for each technique.

Supervised neuron labeling, which derives labels from a set of classified labeling data examples, was discussed in Section 6.2. These approaches include example-centric neuron labeling, example-centric cluster labeling, and weight-centric neuron labeling.

Unsupervised neuron labeling techniques were described in Section 6.3, and do not use classified labeling data. The unsupervised techniques include exploratory labeling, unique cluster labeling, unsupervised weight-based neuron labeling, unsupervised example-based neuron labeling, and unsupervised example-based cluster labeling. The

last three methods are algorithmic, and the descriptions in this work were general, where interchangeable elements were derived from specific labeling algorithms.

One of the main contributions of this research was a novel unsupervised neuron labeling algorithm called unsupervised weight-based cluster labeling. The technique is generic in nature, with a number of interchangeable elements. Broadly speaking, the approach derives descriptive labels for emergent clusters, each of which consists of sub-labels derived from the weight vectors making up the emergent cluster in question. Examples illustrated some labelings produced by the technique, and the approach was critically discussed. Section 6.3.3.2 provided further details on the technique.

Details peculiar to the labeling of unsupervised and semi-supervised SOMs were also outlined. In particular, techniques were suggested for exploiting the classification attributes incorporated within such maps in order to build neuron labels.

The second main focus of this dissertation was on SOM-based DM algorithms. Two existing rule extraction algorithms were identified, namely the boundary-based rule extraction technique and the SIG* algorithm. The former approach was discussed in Section 7.2, and is supervised in the sense that the number of classes in the training data needs to provided as an algorithmic input. SIG*, which was described in Section 7.3, does not require knowledge of the data set classes, and is thus entirely unsupervised. Both of the approaches were described and critically discussed in detail.

The most important contribution of this research work was the introduction of the HybridSOM framework, which allows for any rule extraction algorithm to be executed on the data model of a trained SOM. Like the SIG* algorithm, HybridSOM is unsupervised, because no training data pre-classifications are relied upon. The framework was thoroughly described and critically discussed within Section 7.4 of this work.

Several recommendations related to SOM-based data mining, in a general sense, were arrived at and presented in Section 7.6. The primary drawbacks that were discussed include the additional time complexity introduced by SOM training, and the fact that SOM-based DM is always based on an approximate data model. The advantages of SOM-based DM include possibly improved performance in traditionally challenging domains, the versatility of the SOM for both DM and EDA, improved handling of very large data sets, and potentially improved performance in the presence of data set noise.

9.4 Summary of Experimental Findings

Two related experimental investigations were performed through the course of this research. The first focused on the relative performance of the three identified supervised neuron labeling algorithms, and included two versions of example-centric cluster labeling, one configured with k-means clustering, and the other with Ward clustering. The second set of experiments comparatively analyzed only the unsupervised SOM-based DM techniques. As such, the boundary-based rule extraction approach was excluded from the analysis, and only SIG* and HybridSOM were focused on. In addition, two classical rule extraction algorithms, CN2 and C4.5, were used as baselines against which the SOM-based approaches were compared. Two different HybridSOM variants were investigated, one configured with CN2, and the other with the C4.5 algorithm.

Supervised neuron labeling was assessed by means of a classification task in Section 8.3. The analysis found that example-centric neuron labeling was the superior method within the conducted experiments when training and test set classification accuracy was concerned. It was interesting to note that example-centric neuron labeling also produced the highest proportion of unlabeled map neurons, because this indicated that unlabeled neurons were not indicative of poor classification performance. Poor classification error performance was generally observed from the example-centric cluster labeling approaches, where the Ward clustering algorithm generally performed better.

Section 8.4 investigated the DM algorithms. The SOM-based approaches generally performed fairly poorly against the classical techniques when focusing on classification performance, while producing less complex rule sets. The drawbacks of the SIG* algorithm were twofold. Firstly, the algorithm generally failed to classify a proportion of training and test data, due to the lack of default rules in the algorithmic results. Secondly, SIG* produced very complex data sets, which were highly redundant.

9.5 Future Work

Several possible avenues for future work have been identified through the course of this dissertation's writing. This section briefly lists and summarizes these potential areas of investigation focus, which will be scrutinized during upcoming research exploration.

Firstly, supervised neuron labeling for SOMs offers interesting opportunities for further research. Some avenues of investigation identified by this research are:

- The author intends to analyze a wider variety of clustering algorithms in more detail, to assess the impact of each on example-centric cluster labeling.
- Extended work will investigate and empirically analyze the feasibility of the extension to example-centric neuron labeling that was proposed by Kayacık and Zincir-Heywood [135], and which was described in Section 6.2.1.
- An experimental investigation should be made into the use of supervised labeling methods that utilize multiple label mappings, as suggested in Section 6.2.4.
- Sensitivity analysis of the SOM parameters should be conducted in the context of the neuron labeling approaches discussed in this dissertation.

Unsupervised SOM neuron labeling is a relatively unexplored topic, which is likely to bear fruit in the future. Some examples of possible research include the following:

- An analysis considering additional clustering algorithms is also necessary to assess the impact of each on unsupervised weight- and example-based cluster labeling.
- An analysis of significance measures and attribute selection schemes for unsupervised weight- and example-based labeling is necessary, to find optimal methods.
- An important area of future research will focus on the development of an approach to empirically analyze unsupervised neuron labeling algorithm performance, followed by a thorough investigation into all performance aspects of the algorithms.

The following ideas for future investigation relate to some of the miscellaneous neuron labeling approaches that were not the main focus of the presented research work:

- A detailed analysis of the miscellaneous SOM-specific cluster discovery algorithms described in Section 5.3.3 is required. The application of these techniques to SOM labeling and rule extraction has to be assessed and empirically analyzed.
- The use of classification attributes when labeling supervised and semi-supervised maps, as was described in Section 6.4, is also of interest to future research.

SOM-based DM also offers a number of areas that are likely to offer very fruitful ground for future research efforts. These topics include the following:

- An analysis of the boundary-based rule extractor, from Section 7.2, is needed. Adaptations of this method for unsupervised DM should be investigated.
- The miscellaneous SOM-based DM approaches, which were briefly described in Section 7.5, should be empirically investigated and more extensively compared.
- As Section 7.6 noted, Ultsch [243] states that SOM-based DM can model certain types of data that other approaches cannot. Further research is required to confirm these results, and to establish the underlying characteristics of such data sets.
- The performance degradation of SOM-based DM in the presence of noise, which Section 7.6 suggested as an advantage of these methods, should be investigated.
- The improvements to SIG*, which are proposed in Section 8.4.7, should be empirically analyzed to determine whether the algorithm's performance can be improved.
- Section 8.4.7 suggested that configuring HybridSOM with more noise-resilient DM algorithms is likely to lead to better rule set accuracy. This hypothesis should be more thoroughly investigated with a variety of different stand-alone DM algorithms.
- Methods aiming to improve HybridSOM accuracy should be investigated. For example, the benefit of reducing model noise by stripping interpolating units from maps is a possibly fruitful avenue for further exploration. Another possibility is to incorporate a weighting, which adjusts the contribution that each neuron weight vector in the proxy data set has in the training process of the rule extractor. One possibility is to base each weighting on the number of data examples in the receptive field of the neuron in question. A similar approach has been used in patch learning [101, 275], which has been effectively applied to large data sets.
- As for the neuron labeling approaches, a study of SOM parameter sensitivity should be performed for the SOM-based DM approaches discussed in this work.

Lastly, the possible problems introduced by very high-dimensional data (noted in Sections 3.6, 6.5, and 7.6) must be investigated, especially for neuron labeling and DM.

Bibliography

[1] M. Ackerman and J. Moore. When is clustering perturbation robust? 22 January 2016. Available on-line at `arXiv:1601.05900v1 [cs.LG]`

[2] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In J. Van den Bussche and V. Vianu, editors, *Proceedings of the 8th International Conference on Database Theory*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434, London, England, 4–6 January 2001. Springer. Available on-line at `doi:10.1007/3-540-44503-X_27`

[3] D. W. Aha, C. L. Blake, S. J. Hettich, E. J. Keogh, C. J. Merz, and P. M. Murphy. UCI repository of machine learning databases, 1998. University of California, Irvine, Department of Information and Computer Sciences, Irvine, California, United States of America. Available on-line at `doi:10.17616/R3T91Q` (accessed: 13 February 2004)

[4] D. Alahakoon, S. K. Halgamuge, and B. Srinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614, May 2000. Available on-line at `doi:10.1109/72.846732`

[5] E. Alhoniemi. *Unsupervised Pattern Recognition Methods for Exploratory Analysis of Industrial Process Data*. Doctoral thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory of Computer and Information Science, P. O. Box 5400, FIN-02015 HUT, Espoo, Finland, 2002.

[6] E. Alhoniemi and O. Simula. Interpretation and comparison of multidimensional data partitions. In M. Verleysen, editor, *Proceedings of the 9th European Sympo-*

sium on Artificial Neural Networks, pages 277–282, Bruges, Belgium, 25–27 April 2001. D-Facto.

[7] R. Amarasiri, D. Alahakoon, and K. A. Smith. HDGSOM: A modified growing Self-Organizing Map for high dimensional data clustering. In M. Ishikawa, S. Hashimoto, M. Paprzycki, E. Barakova, K. Yoshida, M. Köppen, D. W. Corne, and A. Abraham, editors, *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems*, pages 216–221, Kitakyushu, Japan, 5–8 December 2004. IEEE. Available on-line at `doi:10.1109/ICHIS.2004.52`

[8] C. Apté, B. Liu, E. P. D. Pednault, and P. Smyth. Business applications of data mining. *Communications of the ACM, Special Issue: Evolving Data Mining into Solutions for Insights*, 45(8):49–53, August 2002. Available on-line at `doi:10.1145/545151.545178`

[9] C. Apté, S. Weiss, and G. Grout. Predicting defects in disk drive manufacturing: A case study in high-dimensional classification. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, pages 212–218, Orlando, Florida, United States of America, 1–5 March 1993. IEEE Computer Society. Available on-line at `doi:10.1109/CAIA.1993.366608`

[10] S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010. Available on-line at `doi:10.1214/09-SS054`

[11] M. Attik, L. Bougrain, and F. Alexandre. Self-Organizing Map initialization. In W. Duch, J. Kacprzyk, E. Oja, and S. Zadrożny, editors, *Artificial Neural Networks: Biological Inspirations: Proceedings of the 15th International Conference on Artificial Neural Networks, Part I*, volume 3696 of *Lecture Notes in Computer Science*, pages 357–362, Warsaw, Poland, 11–15 September 2005. Springer. Available on-line at `doi:10.1007/11550822_56`

[12] A. Azcarraga, M.-H. Hsieh, S.-L. Pan, and R. Setiono. Improved SOM labeling methodology for data mining applications. In O. Maimon and L. Rokach, editors, *Soft Computing for Knowledge Discovery and Data Mining*, pages 45–75. Springer, 2008. Available on-line at `doi:10.1007/978-0-387-69935-6_3`

[13] M.-F. Balcan, A. Blum, and A. Gupta. Approximate clustering without the approximation. In C. Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1068–1077, New York City, New York, United States of America, 4–6 January 2009. Available on-line at doi:10.1137/1.9781611973068.116

[14] H.-U. Bauer and K. R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on Neural Networks*, 3(4):570–579, July 1992. Available on-line at doi:10.1109/72.143371

[15] H.-U. Bauer and T. Villmann. Growing a hypercubical output space in a self-organizing feature map. *IEEE Transactions on Neural Networks*, 8(2):218–226, March 1997. Available on-line at doi:10.1109/72.557659

[16] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, New Jersey, United States of America, 1961.

[17] J. C. Bezdek and N. R. Pal. Some new indexes of cluster validity. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 28(3):301–315, June 1998. Available on-line at doi:10.1109/3477.678624

[18] S. Bhattacharyya. Evolutionary algorithms in data mining: Multi-objective performance modeling for direct marketing. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 465–473, Boston, Massachusetts, United States of America, 20–23 August 2000. ACM. Available on-line at doi:10.1145/347090.347186

[19] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, New York City, New York, United States of America, 1995.

[20] J. Blackmore and R. Miikkulainen. Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 450–455, San Francisco, California, United States of America, 28 March–1 April 1993. Available on-line at doi:10.1109/ICNN.1993.298599

[21] C. E. Bonferroni. Il calcolo delle assicurazioni su gruppi di teste. In *Studi in Onore del Professore Salvatore Ortu Carboni*, pages 13–60. Bardi, Rome, Italy, 1935. Publication in Italian. Quoted by Moyé [175].

[22] C. E. Bonferroni. Teoria statistica delle classi e calcolo delle probabilità. In *Pubblicazioni del R. Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, volume 8, pages 3–62. Libreria Internazionale Seeber, 1936. Publication in Italian. Quoted by Moyé [175].

[23] R. Boswell. Manual for CN2 version 6.1. Manual TI/P2154/RAB/4/1.5, The Turing Institute Limited, Glasgow, Scotland, January 1990. Distributed as part of the downloadable version 6.1 release of the CN2 program package.

[24] E. Boudaillier and G. Hebrail. Interactive interpretation of hierarchical clustering. *Intelligent Data Analysis*, 2(3):229–244, 1998. Available on-line at `doi:10.1016/S1088-467X(98)00026-2`

[25] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. The Wadsworth Statistics/Probability Series. CRC Press, 1984.

[26] G. Cabanes and Y. Bennani. Learning the number of clusters in Self Organizing Map. In G. K. Matsopoulos, editor, *Self-Organizing Maps*. InTech, 2010. Available on-line at `doi:10.5772/9164`

[27] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. The Morgan Kaufmann Series in Interactive Technologies. Morgan Kaufmann, San Francisco, California, United States of America, 1999.

[28] G. Carlsson and F. Mémoli. Characterization, stability and convergence of hierarchical clustering methods. *Journal of Machine Learning Research*, 11:1425–1470, 2010.

[29] C. Carter and J. Catlett. Assessing credit card applications using machine learning. *IEEE Expert*, 2(3):71–79, Fall 1987. Available on-line at `doi:10.1109/MEX.1987.4307093`

[30] B. Cestnik, I. Kononenko, and I. Bratko. ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning: Proceedings of the 2nd European Working Session on Learning, Part One: Learning from Examples in the Presence of Noise*, pages 31–45, Bled, Yugoslavia, May 1987. Sigma Press. Quoted by Quinlan [193].

[31] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. The Wadsworth Statistics/Probability Series. Wadsworth Press, Belmont, California, United States of America, 1983.

[32] G. J. Chappell and J. G. Taylor. The temporal Kohønen map. *Neural Networks*, 6(3):441–445, 1993. Quoted by Koskela *et al* [152]. Available on-line at `doi:10.1016/0893-6080(93)90011-K`

[33] Y. Cheng. Clustering with competing self-organizing maps. In *International Joint Conference on Neural Networks*, volume 4, pages 785–790, Baltimore, Maryland, United States of America, 7–11 June 1992. IEEE. Available on-line at `doi:10.1109/IJCNN.1992.227222`

[34] H. Chernoff. The use of faces to represent points in n-dimensional space graphically. Technical Report 71, Stanford University, Department of Statistics, Stanford, California, United States of America, 27 December 1971.

[35] K.-H. Chuang, M.-J. Chiu, and C.-C. Lin. Model-free functional MRI analysis using Kohonen clustering neural networks and fuzzy c-means. *IEEE Transactions on Medical Imaging*, 18(12):1117–1128, December 1999. Available on-line at `doi:10.1109/42.819322`

[36] P. Clark. Knowledge representation in machine learning. In Y. Kodratoff and A. Hutchinson, editors, *Machine and Human Learning: Advances in European Research*, Michael Horwood Series in Artificial Intelligence, pages 35–49. Kogan Page, London, England, 1989.

[37] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In Y. Kodratoff, editor, *Proceedings of Machine Learning: European Working Session*

on Learning, Part 3: Numeric and Statistical Approaches, volume 482 of *Lecture Notes in Computer Science*, pages 151–163, Porto, Portugal, 6–8 March 1991. Springer. Available on-line at doi:10.1007/BFb0017011

[38] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, March 1989. Available on-line at doi:10.1007/BF00116835

[39] A. Corradini and H.-M. Gross. A hybrid stochastic-connectionist architecture for gesture recognition. In *Proceedings of the International Conference on Information Intelligence and Systems*, pages 336–341, Bethesda, Maryland, United States of America, 31 October–3 November 1999. IEEE Computer Society. Available on-line at doi:10.1109/ICIIS.1999.810286

[40] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995. Available on-line at doi:10.1007/BF00994018

[41] I. G. Costa, F. de A. T. de Carvalho, and M. C. P. de Souto. Stability evaluation of clustering algorithms for time series gene expression data. In *Proceedings of the First Brazilian Workshop on Bioinformatics*, Gramado, Brazil, 18 October 2002.

[42] M. Cottrell, J.-C. Fort, and G. Pagès. Theoretical aspects of the SOM algorithm. *Neurocomputing*, 21(1–3):119–138, 6 November 1998. Available on-line at doi:10.1016/S0925-2312(98)00034-4

[43] M. Cottrell and P. Letrémy. Missing values: Processing with the Kohonen algorithm. In J. Janssen and P. Lenca, editors, *Proceedings of the XIth International Symposium on Applied Stochastic Models and Data Analysis, Part V: Clustering*, pages 489–496, Brest, France, 17–20 May 2005.

[44] M. Cottrell, M. Olteanu, F. Rossi, and N. Villa-Vialaneix. Theoretical and applied aspects of the self-organizing maps. In E. Merényi, M. J. Mendenhall, and P. O'Driscoll, editors, *Advances in Self-Organizing Maps and Learning Vector Quantization: Proceedings of the 11th International Workshop, Part 1*, volume 428 of *Advances in Intelligent Systems and Computing*, pages 3–26, Houston,

Texas, United States of America, 6–8 January 2016. Springer. Available on-line at doi:10.1007/978-3-319-28518-4_1

[45] D. Crevier. *AI: The Tumultuous History of the Search for Artificial Intelligence*. BasicBooks, New York City, New York, United States of America, 1993.

[46] S. K. Das. *Deductive Databases and Logic Programming*. International Series in Logic Programming. Addison-Wesley, 1992.

[47] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(1–4):131–156, 1997. Available on-line at doi:10.1016/S1088-467X(97)00008-5

[48] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, April 1979. Available on-line at doi:10.1109/TPAMI.1979.4766909

[49] E. de Bodt, M. Cottrell, and M. Verleysen. Statistical tools to assess the reliability of self-organizing maps. *Neural Networks*, 15(8–9):967–978, October–November 2002. Available on-line at doi:10.1016/S0893-6080(02)00071-0

[50] G. Deboeck. Picking mutual funds with self-organizing maps. In G. Deboeck and T. Kohonen, editors, *Visual Explorations in Finance with Self-Organizing Maps*, Springer Finance, chapter 3, pages 39–58. Springer, 1998. Available on-line at doi:10.1007/978-1-4471-3913-3_3

[51] G. Deboeck. Public domain vs. commercial tools for creating neural self-organizing maps. *PC AI*, 13(1):27–33, January/February 1999.

[52] G. Deboeck and T. Kohonen, editors. *Visual Explorations in Finance with Self-Organizing Maps*. Springer Finance. Springer, 1998. Available on-line at doi:10.1007/978-1-4471-3913-3

[53] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006. Electronic journal. Available on-line at http://www.jmlr.org/papers/v7/demsar06a (accessed: 15 January 2012)

[54] J. S. Deogun, V. V. Raghavan, A. Sarkar, and H. Sever. Data mining: Trends in research and development. In T. Y. Lin and N. Cercone, editors, *Rough Sets and Data Mining: Analysis of Imprecise Data*, part I, chapter 2, pages 9–45. Kluwer Academic Publishers, Norwell, Massachusetts, United States of America, 1996. Available on-line at doi:10.1007/978-1-4613-1461-5_2

[55] J. L. Devore. *Probability and Statistics for Engineering and the Sciences*. Brooks/ Cole, Boston, Massachusetts, United States of America, eighth edition, 2012.

[56] T. G. Dietterich. Machine learning. In J. F. Traub, editor, *Annual Review of Computer Science*, volume 4, pages 255–306. Annual Reviews, Palo Alto, California, United States of America, June 1990. Available on-line at doi: 10.1146/annurev.cs.04.060190.001351

[57] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In N. Cercone, T. Y. Lin, and X. Wu, editors, *Proceedings IEEE International Conference on Data Mining*, pages 107–114, San Jose, California, United States of America, 29 November–2 December 2001. Available on-line at doi:10.1109/ICDM.2001.989507

[58] F. Dobslaw. Recent development in automatic parameter tuning for metaheuristics. In J. Šafránková and J. Pavlů, editors, *Proceedings of the 19th Annual Conference of Doctoral Students, Part I: Mathematics and Computer Sciences*, pages 54–63, Prague, Czech Republic, 1–4 June 2010. MatfyzPress.

[59] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, October 2012. Available on-line at doi: 10.1145/2347736.2347755

[60] C. Douligeris, A. Pitsillides, and D. Panno. Computational intelligence in telecommunications networks. Guest editorial, *Computer Communications*, 25(16):1413–1414, 1 October 2002. Available on-line at doi:10.1016/S0140-3664(02)00042-7

[61] O. J. Dunn. Confidence intervals for the means of dependent, normally distributed variables. *Journal of the American Statistical Association*, 54(287):613–621, September 1959. Available on-line at doi:10.1080/01621459.1959.10501524

[62] O. J. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, March 1961. Available on-line at doi:10.1080/01621459.1961.10482090

[63] D. M. Dutton and G. V. Conroy. A review of machine learning. *The Knowledge Engineering Review*, 12(4):341–367, December 1997. Available on-line at doi:10.1017/S026988899700101X

[64] R. C. Eberhart. Overview of computational intelligence. In H. K. Chang and Y. T. Zhang, editors, *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 20, number 3, pages 1125–1129, Hong Kong, China, 29 October–1 November 1998. Available on-line at doi:10.1109/IEMBS.1998.747069

[65] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003. Available on-line at doi:10.1007/978-3-662-05094-1

[66] S. M. Ellis and H. S. Steyn. Practical significance (effect sizes) versus or in combination with statistical significance (p-values). Research note, *Management Dynamics: Journal of the Southern African Institute for Management Scientists*, 12(4):51–53, 2003.

[67] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Pearson, seventh edition, 2016.

[68] A. P. Engelbrecht. *Computational Intelligence: An Introduction*. John Wiley & Sons, Chichester, England, second edition, 2007. Available on-line at doi:10.1002/9780470512517

[69] P. Esling and C. Agon. Time-series data mining. *ACM Computing Surveys*, 45(1), November 2012. Available on-line at doi:10.1145/2379776.2379788

[70] D. Fasulo. An analysis of recent work on clustering algorithms. Technical Report 01-03-02, University of Washington, Department of Computer Science and Engineering, Seattle, Washington, United States of America, 26 April 1999.

[71] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3), Fall 1996. Available on-line at doi:10.1609/aimag.v17i3.1230

[72] W. Feller. On the Kolmogorov-Smirnov limit theorems for empirical distributions. *The Annals of Mathematical Statistics*, 19(2):177–189, June 1948. Errata published in [73]. Available on-line at doi:10.1214/aoms/1177730243

[73] W. Feller. Errata: "On the Kolmogorov-Smirnov limit theorems for empirical distributions". *The Annals of Mathematical Statistics*, 21(2):301–302, June 1950. Available on-line at doi:10.1214/aoms/1177729850

[74] C. S. Fertig, A. A. Freitas, L. V. R. Arruda, and C. Kaestner. A fuzzy beam-search rule induction algorithm. In J. M. Żytkow and J. Rauch, editors, *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, volume 1704 of *Lecture Notes in Artificial Intelligence*, pages 341–347, Prague, Czech Republic, 15–18 September 1999. Springer. Available on-line at doi:10.1007/b72280

[75] F. Fessant and S. Midenet. Self-organising map for data imputation and correction in surveys. *Neural Computing & Applications*, 10(4):300–310, April 2002. Available on-line at doi:10.1007/s005210200002

[76] R. A. Fisher. On the interpretation of χ^2 from contingency tables, and the calculation of P. *Journal of the Royal Statistical Society*, 85(1):87–94, January 1922. Available on-line at doi:10.2307/2340521

[77] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, September 1936. Available on-line at doi:10.1111/j.1469-1809.1936.tb02137.x

[78] R. A. Fisher. *Statistical Methods, Experimental Design, and Scientific Inference*. Oxford Science Publications. Oxford University Press, New York City, New York, United States of America, 1990.

[79] R. A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultural and Medical Research*. Oliver and Boyd, Edinburgh, Scotland, 1938.

[80] A. Flexer. On the use of self-organizing maps for clustering and visualization. *Intelligent Data Analysis*, 5(5):373–384, 2001.

[81] J. C. Fort. SOM's mathematics. *Neural Networks*, 19(6–7):812–816, July–August 2006. Available on-line at doi:10.1016/j.neunet.2006.05.025

[82] N. Franken. Visual exploration of algorithm parameter space. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 389–398, Trondheim, Norway, 18–21 May 2009. Available on-line at doi:10.1109/CEC.2009.4982973

[83] A. A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computation: Theory and Applications*, Natural Computing Series, part II, pages 819–845. Springer, 2003. Available on-line at doi:10.1007/978-3-642-18965-4_33

[84] J. H. Friedman. A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*, 26(4):404–408, April 1977. Correspondence. Available on-line at doi:10.1109/TC.1977.1674849

[85] B. Fritzke. Growing cell structures—A self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994. Available on-line at doi:10.1016/0893-6080(94)90091-4

[86] B. Fritzke. Growing Grid—A self-organizing network with constant neighbourhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13, September 1995. Available on-line at doi:10.1007/BF02332159

[87] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. The MIT Press, 1995.

[88] T. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, February 2011. Available on-line at doi:10.1016/j.engappai.2010.09.007

[89] C. C. Fung, K. W. Wong, and D. Myers. An intelligent data analysis approach using self-organising-maps. In *Proceedings of the 11th International Conference on Neural Information Processing*, volume 2, pages 735–738b, Perth, Australia, 16–20 November 1999. Available on-line at doi:10.1109/ICONIP.1999.845687

[90] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual*. Network Theory Limited, third edition, January 2009.

[91] N. R. Garge, G. P. Page, A. P. Sprague, B. S. Gorman, and D. B. Allison. Reproducible clusters from microarray research: Whither? In J. D. Wren and W. Slikker, Jr, editors, *Proceedings of the Second Annual MidSouth Computational Biology and Bioinformatics Society Conference. Bioinformatics: A Systems Approach*, Little Rock, Arkansas, United States of America, 7–9 October 2004. Available on-line at doi:10.1186/1471-2105-6-S2-S10

[92] M. N. Garofalakis and R. J. Miller. Very large databases. In J. G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*, volume 23, pages 133–139. John Wiley & Sons, 1999. Available on-line at doi:10.1002/9780470050118.ecse443

[93] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Statistics and Computing. Springer, 233 Spring Street, New York City, New York, United States of America, second edition, 2003. Available on-line at doi:10.1007/b97336

[94] A. Gisbrecht and B. Hammer. Data visualization by nonlinear dimensionality reduction. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(2):51–73, March/April 2015. Available on-line at doi:10.1002/widm.1147

[95] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, March 1997. Available on-line at doi:10.1023/A:1009726021843

[96] G. Guimarães. Temporal knowledge discovery with Self-organizing Neural Networks. *The International Journal of Computers, Systems and Signals*, 1(1):5–16, 2000.

[97] G. Guimarães, V. S. Lobo, and F. Moura-Pires. A taxonomy of Self-organizing Maps for temporal sequence processing. *Intelligent Data Analysis*, 7(4):269–290, September 2003.

[98] M. Hagenbuchner, A. Sperduti, and A. C. Tsoi. A self-organizing map for adaptive processing of structured data. *IEEE Transactions on Neural Networks*, 14(3):491–505, May 2003. Available on-line at doi:10.1109/TNN.2003.810735

[99] E. Häkkinen and P. Koikkalainen. The neural data analysis environment. In T. Kohonen, editor, *Proceedings of the Workshop on Self-Organizing Maps*, pages 69–74, Espoo, Finland, 4–6 June 1997. Helsinki University of Technology.

[100] A. Hämäläinen. Using genetic algorithm in self-organizing map design. In D. W. Pearson, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 364–367, Alès, France, 1995. Springer. Quoted by Bauer and Villmann [15]. Available on-line at doi:10.1007/978-3-7091-7535-4_95

[101] B. Hammer and A. Hasenfuss. Topographic mapping of large dissimilarity data sets. *Neural Computation*, 22(9):2229–2284, September 2010. Available on-line at doi:10.1162/NECO_a_00012

[102] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaïane. DBMiner: A system for mining knowledge in large relational databases. In E. Simoudis, J. Han, and U. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 250–255, Portland, Oregon, United States of America, 2–4 August 1996. AAAI Press.

[103] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, third edition, 2012.

[104] G. Heidemann, A. Saalbach, and H. Ritter. Semi-automatic acquisition and labelling of image data using SOMs. In M. Verleysen, editor, *Proceedings of the 11th European Symposium on Artificial Neural Networks*, pages 503–508, Bruges, Belgium, 23–25 April 2003. D-side.

[105] L. Herrmann and A. Ultsch. Label propagation for semi-supervised learning in Self-Organizing Maps. In *Proceedings of the 6th International Workshop on Self-Organizing Maps*, Bielefeld, Germany, 3–6 September 2007. Available on-line at `doi:10.2390/biecoll-wsom2007-113`

[106] L. Hetel, J.-L. Buessler, and J.-P. Urban. Superposition-based order analysis in self-organizing maps. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 1, pages 787–792, Budapest, Hungary, 25–29 July 2004. Available on-line at `doi:10.1109/IJCNN.2004.1380020`

[107] F. W. Hewes and H. Gannett. *Scribner's Statistical Atlas of the United States Showing by Graphic Methods their Present Condition and their Political, Social and Industrial Development*, plate 151. Charles Scribner's Sons, 743 and 745 Broadway, New York City, New York, United States of America, 1883.

[108] J. Himberg. Enhancing the SOM-based data visualization by linking different data projections. In L. Xu, L.-w. Chan, I. King, and A. Fu, editors, *Proceedings of the First International Symposium on Intelligent Data Engineering and Learning:*

Perspectives on Financial Engineering and Data Mining, pages 427–434, Hong Kong, China, 14–16 October 1998. Springer.

[109] P. E. Hoffman. *Table Visualizations: A Formal Model and its Applications*. Sc.D. dissertation, University of Massachusetts Lowell, Department of Computer Science, Lowell, Massachusetts, United States of America, 1999.

[110] P. E. Hoffman and G. G. Grinstein. A survey of visualizations for high-dimensional data mining. In U. Fayyad, G. G. Grinstein, and A. Wierse, editors, *Information Visualization in Data Mining and Knowledge Discovery*, volume 104–4 of *The Morgan Kaufmann Series in Data Management Systems*, chapter 2, pages 47–82. Morgan Kaufmann, 14 September 2001.

[111] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning and Discovery*. Computational Models of Cognition and Perception. The MIT Press, September 1986.

[112] M. Holsheimer and A. P. J. M. Siebes. Data mining: The search for knowledge in databases. Technical Report CS-R9406, Centrum voor Wiskunde en Informatica, P. O. Box 94079, NL-1090 GB Amsterdam, The Netherlands, 1994.

[113] T. Hothorn, K. Hornik, M. A. van de Wiel, and A. Zeileis. A Lego system for conditional inference. *The American Statistician*, 60(3):257–263, 2006. Available on-line at `doi:10.1198/000313006X118430`

[114] T. Hothorn, K. Hornik, M. A. van de Wiel, and A. Zeileis. Implementing a class of permutation tests: The coin package. *Journal of Statistical Software*, 28(8):1–23, 13 November 2008. Available on-line at `doi:10.18637/jss.v028.i08`

[115] G. P. Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, 14(1):55–63, January 1968. Available on-line at `doi:10.1109/TIT.1968.1054102`

[116] C. Hung and J.-J. Huang. Mining rules from one-dimensional self-organizing maps. In *Proceedings of the International Symposium on Innovations in Intelligent Sys-*

tems and Applications, pages 292–295, Istanbul, Turkey, 15–18 June 2011. Available on-line at doi:10.1109/INISTA.2011.5946078

[117] C. Hung and S. Wermter. A dynamic adaptive self-organising hybrid model for text clustering. In X. Wu, A. Tuzhilin, and J. Shavlik, editors, *Proceedings of the Third IEEE International Conference on Data Mining*, pages 75–82, Melbourne, Florida, United States of America, 19–22 November 2003. Available on-line at doi:10.1109/ICDM.2003.1250905

[118] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, second edition, August 2004.

[119] W. Iba, J. Wogulis, and P. Langley. Trading off simplicity and coverage in incremental concept learning. In J. Laird, editor, *Proceedings of the Fifth International Conference on Machine Learning*, pages 73–79, Ann Arbor, Michigan, United States of America, 12–14 June 1988. Morgan Kaufmann. Available on-line at doi:10.1016/B978-0-934613-64-4.50013-X

[120] J. Iivarinen, T. Kohonen, J. Kangas, and S. Kaski. Visualizing the clusters on the Self-Organizing Map. In C. Carlsson, T. Järvi, and T. Reponen, editors, *Proceedings of the Conference on Artificial Intelligence Research in Finland*, number 12 in Conference Proceedings of the Finnish Artificial Intelligence Society, pages 122–126, Helsinki, Finland, 29–31 August 1994.

[121] A. Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91, August 1985. Available on-line at doi:10.1007/BF01898350

[122] A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *IEEE Computer*, 29(3):31–44, March 1996. Available on-line at doi:10.1109/2.485891

[123] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999. Available on-line at doi:10.1145/331499.331504

[124] S. Joe and F. Y. Kuo. Remark on algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software*, 29(1):49–57, March 2003. Available on-line at `doi:10.1145/641876.641879`

[125] S. Joe and F. Y. Kuo. Constructing Sobol' sequences with better two-dimensional projections. *SIAM Journal on Scientific Computing*, 30(5):2635–2654, 2008. Available on-line at `doi:10.1137/070709359`

[126] S. Johnson. *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. Scribner, 2002.

[127] J. A. Kangas, T. K. Kohonen, and J. T. Laaksonen. Variants of self-organizing maps. *IEEE Transactions on Neural Networks*, 1(1):93–99, March 1990. Available on-line at `doi:10.1109/72.80208`

[128] S. Kaski. *Data Exploration Using Self-Organizing Maps*. Doctoral thesis, Acta Polytechnica Scandinavica, number 82 of *Mathematics, Computing and Management in Engineering Series*, Helsinki University of Technology, Department of Computer Science and Engineering, Espoo, Finland, February 1997.

[129] S. Kaski. Fast winner search for SOM-based monitoring and retrieval of high-dimensional data. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, volume 2, pages 940–945, Edinburgh, Scotland, 7–10 September 1999. IEE. Available on-line at `doi:10.1049/cp:19991233`

[130] S. Kaski, J. Kangas, and T. Kohonen. Bibliography of Self-Organizing Map (SOM) papers: 1981–1997. *Neural Computing Surveys*, 1:102–350, 1998.

[131] S. Kaski and T. Kohonen. Exploratory data analysis by the self-organizing map: Structures of welfare and poverty in the world. In A.-P. N. Refenes, Y. Abu-Mostafa, J. Moody, and A. Weigend, editors, *Neural Networks in Financial Engineering: Proceedings of the Third International Conference on Neural Networks in the Capital Markets*, pages 498–507, London, England, 11–13 October 1995. World Scientific.

[132] S. Kaski and K. Lagus. Comparing self-organizing maps. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks*, volume 1112 of *Lecture Notes in Computer Science*, pages 809–814, Bochum, Germany, 16–19 July 1996. Springer. Available on-line at doi:10.1007/3-540-61510-5_136

[133] S. Kaski, J. Venna, and T. Kohonen. Coloring that reveals high-dimensional structure in data. In *Proceedings of the 6th International Conference on Neural Information Processing*, volume 2, pages 729–734, Perth, Australia, 16–20 November 1999. IEEE. Available on-line at doi:10.1109/ICONIP.1999.845686

[134] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Mathematical Statistics. Wiley Interscience, 2005. Available on-line at doi:10.1002/9780470316801

[135] H. G. Kayacık and A. N. Zincir-Heywood. Using Self-Organizing Maps to build an attack map for forensic analysis. In *Proceedings of the International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*, pages 33:1–33:8, Oshawa, Ontario, Canada, 30 October–1 November 2006. ACM. Available on-line at doi:10.1145/1501434.1501474

[136] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, 27 November–1 December 1995. Available on-line at doi:10.1109/ICNN.1995.488968

[137] S. Y. Kim and J. W. Lee. Ensemble clustering method based on the resampling similarity measure for gene expression data. *Statistical Methods in Medical Research*, 16(6):539–564, 2007. Available on-line at doi:10.1177/0962280206071842

[138] K. Kiviluoto. Topology preservation in Self-Organizing Maps. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 294–299, Washington, District of Columbia, United States of America, 3–6 June 1996. Available on-line at doi:10.1109/ICNN.1996.548907

[139] K. Kiviluoto and P. Bergius. Analyzing financial statements with the Self-Organizing Map. In T. Kohonen, editor, *Proceedings of the Workshop on Self-Organizing Maps*, pages 362–367, Espoo, Finland, 4–6 June 1997. Helsinki University of Technology.

[140] K. Kiviluoto and P. Bergius. Two-level self-organizing maps for analysis of financial statements. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 1, Anchorage, Alaska, United States of America, 4–9 May 1998. Available on-line at doi:10.1109/IJCNN.1998.682260

[141] D. E. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms, page 145. Addison-Wesley, third edition, 1998.

[142] T. Kohonen. Self-organizing formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982. Available on-line at doi:10.1007/BF00337288

[143] T. Kohonen. Learning vector quantization for pattern recognition. Report TKK-F-A601, Helsinki University of Technology, Espoo, Finland, 1986. Quoted by Kohonen [146].

[144] T. Kohonen. The "neural" phonetic typewriter. *Computer*, 21(3):11–22, March 1988. Available on-line at doi:10.1109/2.28

[145] T. Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer, third edition, 1989. Available on-line at doi:10.1007/978-3-642-88163-3

[146] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, third edition, 2001. Available on-line at doi:10.1007/978-3-642-56927-2

[147] T. Kohonen. *MATLAB Implementations and Applications of the Self-Organizing Map*. Unigrafia Oy, Helsinki, Finland, 2014.

[148] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. SOM_PAK: The Self-Organizing Map program package. Report A31, SOM Programming Team, Laboratory of Computer and Information Science, Helsinki University of Technology, Rakentajanaukio 2 C, SF-02150 Espoo, Finland, January 1996.

[149] T. Kohonen, S. Kaski, K. Lagus, J. Salojärvi, J. Honkela, V. Paatero, and A. Saarela. Self organization of a massive document collection. *IEEE Transactions on Neural Networks*, 11(3):574–585, May 2000. Available on-line at doi:10.1109/72.846729

[150] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas. Engineering applications of the self-organizing map. *Proceedings of the IEEE*, 84(10):1358–1384, October 1996. Available on-line at doi:10.1109/5.537105

[151] A. Kolmogorov. On the empirical determination of a distribution law. In A. N. Shiryayev, editor, *Selected Works of A. N. Kolmogorov, Volume II: Probability Theory and Mathematical Statistics*, volume 26 of *Mathematics and its Applications (Soviet Series)*, pages 139–146. Kluwer Academic Publishers, 1992. Translated from Russian by G. Lindquist. Available on-line at doi:10.1007/978-94-011-2260-3_15

[152] T. Koskela, M. Varsta, J. Heikkonen, and K. Kaski. Temporal sequence processing using recurrent SOM. In L. C. Jain and R. K. Jain, editors, *Proceedings of the Second International Conference on Knowledge-Based Intelligent Electronic Systems*, Adelaide, Australia, 21–23 April 1998. Available on-line at doi:10.1109/KES.1998.725861

[153] M. A. Kraaijveld, J. Mao, and A. K. Jain. A non-linear projection method based on Kohonen's topology preserving maps. In *Proceedings of the 11th IAPR International Conference on Pattern Recognition*, volume 2, conference B: Pattern Recognition Methodology and Systems, pages 41–45, The Hague, The Netherlands, 30 August–3 September 1992. IEEE Computer Society Press. Available on-line at doi:10.1109/ICPR.1992.201718

[154] F. Y. Kuo. Sobol' sequence generator, with direction numbers obtained using the search criterion $D^{(7)}$ up to dimension 21201. Available on-line at `http://web.maths.unsw.edu.au/~fkuo/sobol/` (accessed: 13 January 2012)

[155] K. Lagus and S. Kaski. Keyword selection method for characterizing text document maps. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, volume 1, pages 371–376, Edinburgh, Scotland, 7–10 September 1999. IEE. Available on-line at `doi:10.1049/cp:19991137`

[156] J. Lampinen and E. Oja. Clustering properties of hierarchical self-organizing maps. *Journal of Mathematical Imaging and Vision*, 2(2–3):261–272, November 1992. Available on-line at `doi:10.1007/BF00118594`

[157] S. C. Larson. The shrinkage of the coefficient of multiple correlation. *Journal of Educational Psychology*, 22(1):45–55, January 1931. Quoted by Arlot and Celisse [10]. Available on-line at `doi:10.1037/h0072400`

[158] O. M. Lewis, J. A. Ware, and D. Jenkins. A novel neural network technique for the valuation of residential property. *Neural Computing & Applications*, 5(4):224–229, December 1997. Available on-line at `doi:10.1007/BF01424227`

[159] X. Li, J. Gasteiger, and J. Zupan. On the topology distortion in self-organizing feature maps. *Biological Cybernetics*, 70(2):189–198, December 1993. Available on-line at `doi:10.1007/BF00200832`

[160] H. Löffler-Wirth, E. Willscher, P. Ahnert, K. Wirkner, C. Engel, M. Loeffler, and H. Binder. Novel anthropometry based on 3D-bodyscans applied to a large population based cohort. *PLoS ONE*, 11(7): e0159887, 28 July 2016. Available on-line at `doi:10.1371/journal.pone.0159887`

[161] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1: Statistics, pages 281–297, Berkeley, California, United States of America, 1967. University of California Press.

[162] J. Malone, K. McGarry, W. Stefan, and C. Bowerman. Data mining using rule extraction from Kohonen self-organising maps. *Neural Computing and Applications*, 15(1):9–17, March 2006. Available on-line at doi:10.1007/s00521-005-0002-1

[163] S. Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/ CRC Machine Learning & Pattern Recognition Series. CRC Press, second edition, 2015.

[164] B. Martín-del-Brío, N. Medrano-Marqués, and J. Blasco-Alberto. Feature map architectures for pattern recognition: Techniques for automatic region selection. In D. W. Pearson, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 124–127, Alès, France, 18–21 April 1995. Springer. Quoted by Serrano-Cinca [217]. Available on-line at doi:10.1007/978-3-7091-7535-4_34

[165] T. Martinetz and K. Schulten. A "neural-gas" network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Proceedings of the International Conference on Artificial Neural Networks*, volume 1, pages 397–402, Espoo, Finland, 24–28 June 1991. North-Holland.

[166] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998. Available on-line at doi:10.1145/272991.272995

[167] R. Mayer, D. Merkl, and A. Rauber. Mnemonic SOMs: Recognizable Shapes for Self-Organizing Maps. In *Proceedings of the Fifth International Workshop on Self-Organizing Maps*, pages 131–138, Paris, France, 5–8 September 2005.

[168] J. A. McDonald, W. Stuetzle, and A. Buja. Painting multiple views of complex objects. *SIGPLAN Notices*, 25(10):245–257, October 1990. Available on-line at doi:10.1145/97945.97975

[169] D. Merkl, S. H. He, M. Dittenbach, and A. Rauber. Adaptive hierarchical incremental grid growing: An architecture for high-dimensional data visualization.

In *Proceedings of the 4th Workshop on Self-Organizing Maps*, pages 293–298, Kitakyushu, Japan, 11–14 September 2003.

[170] D. Merkl and A. Rauber. Alternative ways for cluster visualization in self-organizing maps. In T. Kohonen, editor, *Proceedings of the Workshop on Self-Organizing Maps*, pages 106–111, Espoo, Finland, 4–6 June 1997. Helsinki University of Technology.

[171] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The AQ15 inductive learning system: An overview and experiments. Report UIUCDCS-R-86-1260, University of Illinois at Urbana-Champaign, Department of Computer Science, Intelligent Systems Group, Illinois, United States of America, July 1986.

[172] G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, June 1985. Available on-line at doi:10.1007/BF02294245

[173] F. Molle and J. C. Claussen. Investigation of topographical stability of the concave and convex Self-Organizing Map variant. In S. Kollias, A. Stafylopatis, W. Duch, and E. Oja, editors, *Proceedings of the 16th International Conference on Artificial Neural Networks, Part I*, number 4131 in Lecture Notes in Computer Science, pages 208–215, Athens, Greece, 10–14 September 2006. Springer. Available on-line at doi:10.1007/11840817_22

[174] F. Mosteller and J. W. Tukey. Data analysis, including statistics. In G. Lindzey and E. Aronson, editors, *Handbook of Social Psychology*, volume 2, chapter 10, pages 80–203. Addison-Wesley, second edition, 1968.

[175] L. A. Moyé. *Multiple Analyses in Clinical Trials: Fundamentals for Investigators*. Statistics for Biology and Health. Springer, 2003. Available on-line at doi:10.1007/b97513

[176] F. Murtagh. Interpreting the Kohonen self-organizing feature map using contiguity-constrained clustering. *Pattern Recognition Letters*, 16(4):399–408, April 1995. Available on-line at doi:10.1016/0167-8655(94)00113-H

[177] N. J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, San Francisco, California, United States of America, 1998.

[178] M. Oja, S. Kaski, and T. Kohonen. Bibliography of Self-Organizing Map (SOM) papers: 1998–2001 addendum. *Neural Computing Surveys*, 3:1–156, 2003.

[179] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, August 2002. Available on-line at doi:10.1109/TEVC.2002.802452

[180] K. Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine, Series 5 (1876–1900)*, 50(302):157–175, 1900. Available on-line at doi:10.1080/14786440009463897

[181] W. Pedrycz and H. C. Card. Linguistic interpretation of self-organizing maps. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 371–378, San Diego, California, United States of America, 8–12 March 1992. Available on-line at doi:10.1109/FUZZY.1992.258643

[182] H. Petersohn. Assessment of cluster analysis and self-organizing maps. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):139–149, April 1998. Available on-line at doi:10.1142/S0218488598000124

[183] D. Polani. Measures for the organization of Self-Organizing Maps. In U. Seiffert and L. C. Jain, editors, *Self-Organizing Neural Networks: Recent Advances and Applications*, volume 78 of *Studies in Fuzziness and Soft Computing*, pages 13–44. Physica-Verlag, 2002. Available on-line at doi:10.1007/978-3-7908-1810-9_2

[184] M. Pöllä, T. Honkela, and T. Kohonen. Bibliography of Self-Organizing Map (SOM) papers: 2002-2005 addendum. TKK Report in Information and Computer Science TKK-ICS-R23, Helsinki University of Technology, Faculty of Information and Natural Sciences, Department of Information and Computer Science, P. O. Box 5400, FI-02015 TKK, Espoo, Finland, 2009.

[185] G. Pölzlbauer, M. Dittenbach, and A. Rauber. A visualization technique for Self-Organizing Maps with vector fields to obtain the cluster structure at desired levels of detail. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 1558–1563, Montréal, Québec, Canada, 31 July–4 August 2005. International Neural Network Society, IEEE Computational Intelligence Society. Available on-line at doi:10.1109/IJCNN.2005.1556110

[186] G. Potgieter. Mining continuous classes using evolutionary computing. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, October 2002.

[187] J. W. Pratt. Remarks on zeros and ties in the Wilcoxon signed rank procedures. *Journal of the American Statistical Association*, 54(287):655–667, September 1959. Available on-line at doi:10.2307/2282543

[188] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, third edition, 2007.

[189] D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 1999.

[190] J. R. Quinlan. Discovering rules by induction from large collections of examples. In D. Michie, editor, *Expert Systems in the Micro-Electronic Age*, pages 168–201. Edinburgh University Press, 1979. Quoted by Quinlan [193].

[191] J. R. Quinlan. Generating production rules from decision trees. In J. P. McDermott, editor, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, volume 1, pages 304–307, Milan, Italy, 23–28 August 1987. Morgan Kaufmann.

[192] J. R. Quinlan. Unknown attribute values in induction. In A. M. Segre, editor, *Proceedings of the Sixth International Workshop on Machine Learning*, pages 164–168, Ithaca, New York, United States of America, 26–27 June 1989. Morgan Kaufmann. Available on-line at doi:10.1016/B978-1-55860-036-2.50048-5

[193] J. R. Quinlan. *C4.5: Programs for Machine Learning*. The Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann, 1993.

[194] J. R. Quinlan. The minimum description length principle and categorical theories. In W. W. Cohen and H. Hirsh, editors, *Proceedings of the Eleventh International Conference on Machine Learning*, pages 233–241, New Brunswick, New Jersey, United States of America, 10–13 July 1994. Morgan Kaufmann. Available on-line at doi:10.1016/B978-1-55860-335-6.50036-2

[195] J. R. Quinlan. MDL and categorical theories (continued). In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 464–470, Tahoe City, California, United States of America, 9–12 July 1995. Morgan Kaufmann. Available on-line at doi:10.1016/B978-1-55860-377-6.50064-5

[196] J. R. Quinlan. Bagging, boosting, and C4.5. In W. J. Clancey and D. S. Weld, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Conference on Innovative Applications of Artificial Intelligence*, volume 1, pages 725–730, Portland, Oregon, United States of America, 4–8 August 1996. AAAI Press.

[197] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, January–June 1996. Available on-line at doi:10.1613/jair.279

[198] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. Available on-line at http://www.R-project.org/ (accessed: 11 February 2015)

[199] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, New York City, New York, United States of America, third edition, 2003.

[200] F. Ratle, A. Pozdnoukhov, V. Demyanov, V. Timonin, and E. Savelieva. Spatial data analysis and mapping using machine learning algorithms. In M. Kanevski,

editor, *Advanced Mapping of Environmental Data: Geostatistics, Machine Learning and Bayesian Maximum Entropy*, Geographical Information Systems Series, chapter 4, pages 95–148. Wiley-ISTE, 2008. Available on-line at doi:10.1002/9780470611463.ch4

[201] A. Rauber and D. Merkl. Automatic labeling of self-organizing maps: Making a treasure-map reveal its secrets. In N. Zhong and L. Zhou, editors, *Proceedings of the Third Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining*, volume 1574 of *Lecture Notes in Artificial Intelligence*, pages 228–237, Beijing, China, 26–28 April 1999. Springer. Available on-line at doi:10.1007/3-540-48912-6_31

[202] A. Rauber and D. Merkl. The SOMLib digital library system. In S. Abiteboul and A.-M. Vercoustre, editors, *Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries*, volume 1696 of *Lecture Notes in Computer Science*, pages 323–342, Paris, France, 22–24 September 1999. Springer. Available on-line at doi:10.1007/3-540-48155-9_21

[203] A. Rauber, D. Merkl, and M. Dittenbach. The Growing Hierarchical Self-Organizing Map: Exploratory data analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341, November 2002. Available on-line at doi:10.1109/TNN.2002.804221

[204] A. Rényi. On the theory of order statistics. *Acta Mathematica Academiae Scientiarum Hungarica*, 4(3–4):191–231, September 1953. Available on-line at doi:10.1007/BF02127580

[205] G. A. Ringland and D. A. Duce, editors. *Approaches to Knowledge Representation: An Introduction*. Knowledge-Based and Expert Systems Series. Research Studies Press, Letchworth, England, 1988. Quoted by Holsheimer and Siebes [112].

[206] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983. Available on-line at doi:10.1214/aos/1176346150

[207] H. Ritter. Self-Organizing Maps on non-euclidean spaces. In E. Oja and S. Kaski, editors, *Kohonen Maps*, pages 97–109. Elsevier Science, 1999. Available on-line at doi:10.1016/B978-044450270-4/50007-3

[208] S. Rüping, M. Porrmann, and U. Rückert. SOM hardware-accelerator. In T. Kohonen, editor, *Proceedings of the Workshop on Self-Organizing Maps*, pages 136–141, Espoo, Finland, 4–6 June 1997. Helsinki University of Technology.

[209] M. Sahami. Learning classification rules using lattices. Extended abstract in N. Lavrač and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *Lecture Notes in Artificial Intelligence*, pages 343–346, Heraclion, Crete, Greece, 25–27 April 1995. Springer. Available on-line at doi:10.1007/3-540-59286-5_83

[210] T. Samad and S. A. Harp. Self-organization with partial data. *Network: Computation in Neural Systems*, 3(2):205–212, May 1992. Available on-line at doi:10.1088/0954-898X/3/2/008

[211] S. Samarasinghe. *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*. Auerbach Publications, Boca Raton, Florida, United States of America, 2007.

[212] J. W. Sammon, Jr. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18(5):401–409, May 1969. Available on-line at doi:10.1109/T-C.1969.222678

[213] R. T. Santos, J. C. Nievola, and A. A. Freitas. Extracting comprehensible rules from neural networks via genetic algorithms. In X. Yao and D. B. Fogel, editors, *Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pages 130–139, San Antonio, Texas, United States of America, 11–13 May 2000. Available on-line at doi:10.1109/ECNN.2000.886228

[214] N. I. Sapankevych and R. Sankar. Time series prediction using Support Vector Machines: A survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, May 2009. Available on-line at doi:10.1109/MCI.2009.932254

[215] C. R. Schmidt, S. J. Rey, and A. Skupin. Effects of irregular topology in spherical self-organizing maps. *International Regional Science Review*, 34(2):215–229, April 2011. Available on-line at doi:10.1177/0160017610387297

[216] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley Series in Probability and Mathematical Statistics. Wiley Interscience, 1992. Available on-line at doi:10.1002/9780470316849

[217] C. Serrano-Cinca. Self organizing neural networks for financial diagnosis. *Decision Support Systems*, 17(3):227–238, July 1996. Available on-line at doi:10.1016/0167-9236(95)00033-X

[218] A. Sharma, R. Podolsky, J. Zhao, and R. A. McIndoe. A modified hyperplane clustering algorithm allows for efficient and accurate clustering of extremely large datasets. *Bioinformatics*, 25(9):1152–1157, 1 May 2009. Available on-line at doi:10.1093/bioinformatics/btp123

[219] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the International Conference on Evolutionary Computation (part of the IEEE World Congress on Computational Intelligence)*, pages 69–73, Anchorage, Alaska, United States of America, 4–9 May 1998. Available on-line at doi:10.1109/ICEC.1998.699146

[220] A. Shoshani. OLAP and statistical databases: Similarities and differences. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 185–196, Tucson, Arizona, United States of America, 11–15 May 1997. Available on-line at doi:10.1145/263661.263682

[221] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, sixth edition, 2011.

[222] O. Simula, P. Vasara, J. Vesanto, and R.-R. Helminen. The Self-Organizing Map in industry analysis. In L. C. Jain and V. R. Vemuri, editors, *Industrial Applications of Neural Networks*, volume 3 of *International Series on Computational Intelligence*, chapter 4, pages 87–112. CRC Press, 1999.

[223] M. Siponen, J. Vesanto, O. Simula, and P. Vasara. An approach to automated interpretation of SOM. In N. Allinson, H. Yin, L. Allinson, and J. Slack, editors, *Advances in Self-Organising Maps*, pages 89–94, Lincoln, England, 13–15 June 2001. Springer. Available on-line at doi:10.1007/978-1-4471-0715-6_13

[224] N. V. Smirnov. Otsenka raskhozhdeniya mezhdu empiricheskimi krivymi raspredeleniya v dvukh nezavisimykh vyborkakh. *Byulleten' Moskovskogo Universiteta*, 2:3–14, 1939. Publication in Russian. Quoted by Feller [72][1].

[225] I. M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967. Quoted by Gentle [93], and Joe and Kuo [124, 125]. Available on-line at doi:10.1016/0041-5553(67)90144-9

[226] S. Sohn and C. H. Dagli. Advantages of using fuzzy class memberships in self-organizing map and support vector machines. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 1886–1890, Washington, District of Columbia, United States of America, 15–19 July 2001. Available on-line at doi:10.1109/IJCNN.2001.938451

[227] T. Song, M. Jamshidi, R. R. Lee, and M. Huang. A novel weighted probabilistic neural network for MR image segmentation. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2501–2506, Waikoloa, Hawaii, United States of America, 10–12 October 2005. Available on-line at doi:10.1109/ICSMC.2005.1571524

[1] The original publication is unavailable. The majority of English sources, including Feller [72], reference incorrect publication information. According to Rényi [204], the correct publication information, transliterated in [224], is: Н. В. Смирнов. Оценка расхождения между эмпирическими кривыми распределения в двух независимых выборках. *Бюллетень Московского Университета*, 2:3–14, 1939.

[228] B. Streitberg and J. Röhmel. Exact distributions for permutations and rank tests: An introduction to some recently published algorithms. *Statistical Software Newsletter*, 12(1):10–17, 1986. Quoted by Hothorn *et al* [114].

[229] B. Streitberg and J. Röhmel. Exakte Verteilungen für Rang- und Randomisierungstests im allgemeinen c-Stichprobenproblem. *EDV in Medizin und Biologie / EDP in Medicine and Biology*, 18(1):12–19, 1987. Publication in German.

[230] M. Strickert and B. Hammer. Merge SOM for temporal data. *Neurocomputing*, 64:39–71, March 2005. Available on-line at `doi:10.1016/j.neucom.2004.11.014`

[231] M.-C. Su, T.-K. Liu, and H.-T. Chang. An efficient initialization scheme for the self-organizing feature map algorithm. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 1906–1910, Washington, District of Columbia, United States of America, 10–16 July 1999. International Neural Network Society, IEEE Neural Networks Council. Available on-line at `doi:10.1109/IJCNN.1999.832672`

[232] J. Surowiecki. *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations.* Anchor Books, New York City, New York, United States of America, August 2005.

[233] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. R. H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States of America, December 1991.

[234] J. Tuckova. The possibility of Kohonen self-organizing map applications in medicine. In *Proceedings of the 11th International Workshop of Electronics, Control, Measurement, Signals and Their Application to Mechatronics*, pages 17–22, Toulouse, France, 24–26 June 2013. IEEE. Available on-line at `doi:10.1109/ECMSM.2013.6648946`

[235] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley Series in Behavioral Science: Quantitative Methods. Addison-Wesley, Reading, Massachusetts, United States of America, 1977.

[236] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, volume II: The New Technologies*, volume 14 of *Principles of Computer Science Series*. Computer Science Press, Rockville, Maryland, United States of America, 1989.

[237] A. G. H. Ultsch. Control for knowledge-based information retrieval. Doctor of technical sciences dissertation, Swiss Federal Institute of Technology, Zürich, Switzerland, 1987. Quoted by Ultsch [241]. Available on-line at `doi:10.3929/ethz-a-000410196`

[238] A. Ultsch. Konnektionistische Modelle und ihre Integration mit wissensbasierten Systemen. Forschungsbericht 396, Technische Universität Dortmund, Fakultät für Informatik, Dortmund, Germany, 1991. Publication in German. Quoted by Ultsch and Korus [243].

[239] A. Ultsch. Self-organizing neural networks for knowledge acquisition. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 208–210, Vienna, Austria, 3–7 August 1992. John Wiley & Sons. Quoted by Ultsch and Korus [243].

[240] A. Ultsch. Self-Organizing Neural Networks perform different from statistical k-means clustering. In M. van der Meer, R. Schmidt, and G. Wolf, editors, *BMBF Statusseminar: Künstliche Intelligenz, Neuroinformatik und Intelligente Systeme*, pages 433–443, München, Germany, 17–19 April 1996. Bundesministerium für Bildung und Forschung.

[241] A. Ultsch. Data Mining and Knowledge Discovery with emergent Self-Organizing Feature Maps for multivariate time series. In E. Oja and S. Kaski, editors, *Kohonen Maps*, pages 33–46. Elsevier Science, Amsterdam, Holland, 1999. Available on-line at `doi:10.1016/B978-044450270-4/50003-6`

[242] A. Ultsch and K. U. Höffgen. Automatische Wissensakquisition für Fuzzy-Expertensysteme aus selbstorganisierenden neuronalen Netzen. Forschungsbericht 404, Technische Universität Dortmund, Fakultät für Informatik, Dortmund, Germany, 1991. Publication in German. Quoted by Ultsch and Korus [243].

[243] A. Ultsch and D. Korus. Automatic acquisition of symbolic knowledge from subsymbolic neural networks. In *Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing*, volume 1, pages 326–331, Aachen, Germany, 28–31 August 1995. European Laboratory for Intelligent Techniques Engineering Foundation.

[244] A. Ultsch and H. P. Siemon. Kohonen's self organizing feature maps for exploratory data analysis. In *Proceedings of the International Neural Networks Conference*, volume 1, pages 305–308, Paris, France, 9–13 July 1990. Kluwer Academic Publishers.

[245] X. Valero, F. Alías, D. Oldoni, and D. Botteldooren. Support Vector Machines and Self-Organizing Maps for the recognition of sound events in urban soundscapes. In C. Burroughs and S. Conlon, editors, *Proceedings of the 41st International Congress and Exposition on Noise Control Engineering*, volume 3, pages 2197–2206, New York City, New York, United States of America, 19–22 August 2012. Institute of Noise Control Engineering.

[246] A. Varfis and C. Versino. Clustering of european regions on the basis of socio-economic data — A Kohonen feature map approach. In D. Würtz and F. Murtagh, editors, *Proceedings of Parallel Problem Solving from Nature — Applications in Statistics and Economics*, pages 57–68, Zürich, Switzerland, December 1991.

[247] A. Varfis and C. Versino. Clustering of socio-economic data with Kohonen maps. *Neural Network World*, 2(6):813–834, 1992.

[248] T. Vatanen, M. Osmala, T. Raiko, K. Lagus, M. Sysi-Aho, M. Orešič, T. Honkela, and H. Lähdesmäki. Self-organization and missing values in SOM and GTM. *Neurocomputing*, 147:60–70, 5 January 2015. Available on-line at `doi:10.1016/j.neucom.2014.02.061`

[249] S. Vegas-Azcárate, T. Gautama, and M. M. Van Hulle. Topology preservation in topographic maps. Technical Report on Statistics and Decision Sciences TR05/14, Universidad Rey Juan Carlos, Grupo de Estadística y Ciencias de la Decisión, Móstoles, Spain, December 2005.

[250] P. F. Velleman and D. C. Hoaglin. Data analysis. In D. C. Hoaglin and D. S. Moore, editors, *Perspectives on Contemporary Statistics*, number 21 in MAA Notes, chapter 2, pages 19–39. The Mathematical Association of America, 1992.

[251] A. Vellido, P. J. G. Lisboa, and K. Meehan. Segmentation of the on-line shopping market using neural networks. *Expert Systems with Applications*, 17(4):303–314, November 1999. Available on-line at doi:10.1016/S0957-4174(99)00042-1

[252] S. Venkatasubramanian and S. Vassilvitskii. New developments in the theory of clustering. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, District of Columbia, United States of America, 25–28 July 2010. Video recording of tutorial. Available on-line at doi:10.1145/1835804.1866288

[253] J. Vesanto. SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126, August 1999. Available on-line at doi:10.1016/S1088-467X(99)00013-X

[254] J. Vesanto. Neural network tool for data mining: SOM Toolbox. In L. Yliniemi and E. Juuso, editors, *Proceedings of the Third International Symposium on Tool Environments and Development Methods for Intelligent Systems*, pages 184–196, Oulu, Finland, 13–14 April 2000. Oulun Yliopistopaino.

[255] J. Vesanto. Using SOM in data mining. Licentiate's thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Espoo, Finland, 5 April 2000.

[256] J. Vesanto. *Data Exploration Process Based on the Self-Organizing Map*. Doctoral dissertation, Acta Polytechnica Scandinavica, number 115 of *Mathematics and*

Computing Series, Helsinki University of Technology, Department of Computer Science and Engineering, Espoo, Finland, 16 May 2002.

[257] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600, May 2000. Available on-line at doi:10.1109/72.846731

[258] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. SOM Toolbox for Matlab 5. Report A57, SOM Toolbox Team, Helsinki University of Technology, P. O. Box 5400, FIN-02015 HUT, Espoo, Finland, 20 April 2000.

[259] J. Vesanto, J. Himberg, M. Siponen, and O. Simula. Enhancing SOM based data visualization. In T. Yamakawa and G. Matsumoto, editors, *Methodologies for the Conception, Design and Application of Soft Computing: Proceedings of the 5th International Conference on Soft Computing and Information/Intelligent Systems*, pages 64–67, Iizuka, Japan, 16–20 October 1998. World Scientific.

[260] J. Vesanto and J. Hollmén. An automated report generation tool for the data understanding phase. In A. Abraham and M. Köppen, editors, *Hybrid Information Systems*, volume 14 of *Advances in Soft Computing*, pages 611–625. Physica-Verlag, 2002. Available on-line at doi:10.1007/978-3-7908-1782-9_44

[261] J. Vesanto, M. Sulkava, and J. Hollmén. On the decomposition of the Self-Organizing Map distortion measure. In *Proceedings of the 4th Workshop on Self-Organizing Maps*, pages 11–16, Kitakyushu, Japan, 11–14 September 2003.

[262] T. Villmann, R. Der, M. Herrmann, and T. M. Martinetz. Topology preservation in self-organizing feature maps: Exact definition and measurement. *IEEE Transactions on Neural Networks*, 8(2):256–266, March 1997. Available on-line at doi:10.1109/72.557663

[263] T. Voegtlin. Recursive self-organizing maps. *Neural Networks*, 15(8–9):979–991, October–November 2002. Available on-line at doi:10.1016/S0893-6080(02)00072-2

[264] U. von Luxburg. Clustering stability: An overview. *Foundations and Trends in Machine Learning*, 2(3):235–274, March 2010. Available on-line at doi:10.1561/2200000008

[265] J. Wang, X. Wu, and C. Zhang. Support vector machines based on K-means clustering for real-time business intelligence systems. *International Journal of Business Intelligence and Data Mining*, 1(1):54–64, 2005. Available on-line at doi:10.1504/IJBIDM.2005.007318

[266] J. H. Ward, Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, March 1963. Available on-line at doi:10.1080/01621459.1963.10500845

[267] J. E. Whitesitt. *Boolean Algebra and its Applications*. Addison-Wesley, 1961.

[268] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, December 1945. Available on-line at doi:10.2307/3001968

[269] D. Wishart. 256 Note: An algorithm for hierarchical classifications. *Biometrics*, 25(1):165–170, March 1969. Available on-line at doi:10.2307/2528688

[270] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997. Available on-line at doi:10.1109/4235.585893

[271] X. Wu, C. Zhang, and S. Zhang. Database classification for multi-database mining. *Information Systems*, 30(1):71–88, March 2005. Available on-line at doi:10.1016/j.is.2003.10.001

[272] L. Zhang. *Learning the Structure of High-Dimensional Manifolds with Self-Organizing Maps for Accurate Information Extraction*. PhD thesis, Rice University, Houston, Texas, United States of America, May 2011.

[273] S. Zhang, C. Zhang, and X. Wu. *Knowledge Discovery in Multiple Databases*. Advanced Information and Knowledge Processing. Springer, 2004. Available on-line at doi:10.1007/978-0-85729-388-6

[274] X. Zhang and Y. Li. Self-organizing map as a new method for clustering and data analysis. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 2448–2451, Nagoya, Japan, 25–29 October 1993. IEEE. Available on-line at doi:10.1109/IJCNN.1993.714219

[275] X. Zhu, F.-M. Schleif, and B. Hammer. Patch processing for relational learning vector quantization. In J. Wang, G. G. Yen, and M. M. Polycarpou, editors, *Advances in Neural Networks: Proceedings of the 9th International Symposium on Neural Networks, Part 1*, volume 7367 of *Lecture Notes in Computer Science*, pages 55–63, Shenyang, China, 11–14 July 2012. Springer. Available on-line at doi:10.1007/978-3-642-31346-2_7

[276] X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210, November 2004. Available on-line at doi:10.1007/s10462-004-0751-8

[277] S. Zrehen. Analyzing Kohonen maps with geometry. In S. Gielen and B. Kappen, editors, *Proceedings of the International Conference on Artificial Neural Networks*, pages 609–612, Amsterdam, The Netherlands, 13–16 September 1993. Springer. Available on-line at doi:10.1007/978-1-4471-2063-6_167

Appendix A

The CN2 Algorithm

This appendix briefly overviews the CN2 rule induction algorithm, which was used during the experimentation reported on in Chapter 8 of this work. The original version of the CN2 algorithm was jointly designed by Peter Clark and Tim Niblett in 1989 [38]. In 1990 several further improvements were added to the algorithm [23]. The problems that were addressed by these additions, as well as the performance gains achieved, were discussed and experimentally analyzed by Clark and Boswell in 1991 [37]. This appendix covers the version of CN2 with these additions, which was used in the experimental work.

The remainder of this appendix is organized as follows: Section A.1 discusses the availability of the algorithm's implemented program package. CN2 derives a rule set in a single phase by performing a search for a "best" rule defined within a rule space. This process is discussed in Section A.2. Finally, Section A.3 summarizes the appendix.

A.1 Implementation Availability

There is no commercial version of CN2. The most recent version of CN2 (version 6.1 at the time of writing) is available for free download[1], and was implemented by Robin Boswell. A number of cross-platform porting issues were addressed by Rick Kufrin and Johannes Fuernkranz, allowing execution on a range of other operating systems.

[1] The source code, a Sparc executable and package documentation [23] are available for free download from Peter Clark's home page at `http://www.cs.utexas.edu/users/pclark/software/`.

A.2 Rule Set Generation

CN2 builds a rule set from a training set, $\mathcal{D}_T$, with several nominal or continuous descriptive attributes and one nominal classification attribute. Rules are ordered or unordered, where the former were used in the experiments, and are assumed from this point.

Section A.2.1 discusses the general philosophy underlying the CN2 algorithm. Section A.2.2 covers the search procedure that CN2 uses when building rule sets, while algorithmic features that are not applicable to this work are discussed in Section A.2.3.

A.2.1 Basic Philosophy

Section 2.1.2.4 discusses the production rule sets CN2 generates. A rule's antecedent consists of a single condition on an attribute value, or a conjunction of such conditions (called a *complex* in the original research work). It is important to note that the CN2 algorithm relies on the assumption that an empty complex classifies *all* examples.

The CN2 algorithm is based on a family of rule induction approaches known as AQ algorithms. An AQ algorithm generates a single (initially simple) rule for each class in training set $\mathcal{D}_T$. Each rule is specialized by iteratively increasing its complexity until all the training examples of the rule's class are covered, while no training examples of other classes are covered. The specialization of each rule may be seen as several parallel hill-climbing searches through a rule space, and is thus often called a *beam search*.

Unlike generic AQ algorithms, CN2's beam search is more general, because the search considers all possible rule specializations (including those that do not classify $\mathcal{D}_T$ examples perfectly). In addition, the rule specialization uses a heuristic as a cut-off, which facilitates a pruning of the search and prevents further specialization of poor rules.

A.2.2 Beam Search

This subsection deals with the details surrounding the beam search procedure that is employed by the CN2 algorithm. In order to guide the search process, the algorithm uses three heuristic tests, which evaluate every complex that has been specialized. Section A.2.2.1 describes these tests. The characteristics of the iterative algorithm that is executed in order to perform the actual beam search are discussed within Section A.2.2.2.

A.2.2.1 Heuristic Evaluation of Complexes

CN2 uses three heuristic tests to identify the best complex that has been generated for a subset of training data, and to guide pruning of the search. The tests focus on *complex accuracy*, *complex significance*, and *complex accuracy compared to a default rule*.

The first test, denoted $acc(comp_{\bar{a}}, comp_{\bar{b}})$, has a value of *true* when the first complex, $comp_{\bar{a}}$, is more accurate than the second, $comp_{\bar{b}}$, and a value of *false* otherwise:

$$acc(comp_{\bar{a}}, comp_{\bar{b}}) = \begin{cases} true & \text{if } acc(comp_{\bar{a}}) > acc(comp_{\bar{b}}) \\ false & \text{otherwise} \end{cases} \tag{A.1}$$

where $acc(comp_{\bar{a}})$ is a heuristic that computes a numeric value, which represents a measure of the accuracy that any particular complex, $comp_{\bar{a}}$, demonstrates.

The CN2 algorithm provides two alternative measures for the accuracy of a complex [23], namely a *naïve error estimate* and the *Laplacian error estimate*:

- The simplest accuracy measure used by CN2 is known as a naïve error estimate [23]. Using this error estimate, the accuracy of complex $comp_{\bar{a}}$ is calculated as:

$$acc(comp_{\bar{a}}) = \frac{cover(C_m, comp_{\bar{a}})}{cover(comp_{\bar{a}})} \tag{A.2}$$

  where C_m is the majority class within the data examples that are covered by $comp_{\bar{a}}$, $cover(C_m, comp_{\bar{a}})$ is the number of examples that are covered by $comp_{\bar{a}}$ and belong to C_m, and $cover(comp_{\bar{a}})$ is the total number of examples covered by $comp_{\bar{a}}$.

- The Laplacian error estimate [37] is a more advanced accuracy measure used by CN2. This estimate defines the accuracy of a complex, $comp_{\bar{a}}$, as follows:

$$acc(comp_{\bar{a}}) = \frac{cover(C_m, comp_{\bar{a}}) + 1}{cover(comp_{\bar{a}}) + \hat{c}(\mathcal{D}_T)} \tag{A.3}$$

  where $\hat{c}(\mathcal{D}_T)$ is the total number of classes that are present in the complete training data set. The Laplacian error estimate is the accuracy measure that is used by default within the latest available implementation of the CN2 algorithm.

The second heuristic test, which is denoted $significant(comp_{\bar{a}})$, evaluates the significance of a complex, $comp_{\bar{a}}$. This heuristic test estimates whether a complex expresses

a pattern or regularity that is not likely to exist as a result of chance. This heuristic test has a value of *true* if the complex is significant in accordance with a user-specified significance threshold denoted *cn2-sig*, and false otherwise, and is expressed as:

$$significant(comp_{\bar{a}}) = \begin{cases} true & \text{if } sig\_heur(comp_{\bar{a}}) > cn2\text{-}sig \\ false & \text{otherwise} \end{cases} \tag{A.4}$$

where $sig\_heur(comp_{\bar{a}})$ is a heuristic, which calculates a numeric value that represents an estimate of the significance that an arbitrary complex, $comp_{\bar{a}}$, exhibits.

The heuristic, $sig\_heur(comp_{\bar{a}})$, which measures complex significance in CN2, is based on the *likelihood ratio statistic*. This significance statistic is defined as follows:

$$sig\_heur(comp_{\bar{a}}) = 2 \sum_{m=1}^{\hat{c}(\mathcal{D}_T)} \left(actual_m \cdot \log\left(\frac{actual_m}{expected_m} \right) \right) \tag{A.5}$$

where $actual_m$ is the actual frequency distribution of examples covered by $comp_{\bar{a}}$ that belong to class C_m, and $expected_m$ is the expected frequency of examples belonging to class C_m (assuming that the complex $comp_{\bar{a}}$ were to select examples randomly).

The final heuristic test is denoted $beats\_default(comp_{\bar{a}})$. This test has a value of *true* when $comp_{\bar{a}}$ has a higher accuracy than a default rule that simply predicts the most prevalent class in the training data set, and *false* otherwise. This test is expressed as:

$$beats\_default(comp_{\bar{a}}) = \begin{cases} true & \text{if } acc(comp_{\bar{a}}) > acc(default) \\ false & \text{otherwise} \end{cases} \tag{A.6}$$

where the $acc(comp_{\bar{a}})$ measure used is the naïve error estimate that is defined in Equation (A.2), and *default* is an empty complex covering the full training data set.

A.2.2.2 Iterative Algorithm

The beam search is performed by an iterative algorithm, which this discussion breaks down into two parts. Firstly, the main body of the CN2 algorithm repeatedly generates rules, which are inserted into a final rule set. This rule set building algorithm is outlined in Algorithm A.1, and uses the algorithm's second part, a function shown in Algorithm A.2. This function searches for a complex that best describes the data examples from the training set that have not been covered by previously generated rules.

Define user-specified a maximum *star* size, denoted *cn2-star*
Define a training set, denoted $\mathcal{D}_T$
Define an empty rule set, denoted *rule_set*

repeat:
    Define a complex *best_comp* = `null`
    call function FindBestComplex($\mathcal{D}_T$, *best_comp*), outlined in Algorithm A.2

    if *best_comp* $\neq$ `null` **then**
        Define $\mathcal{D}''_T$ as the set of examples in $\mathcal{D}_T$ covered by *best_comp*
        Remove examples in $\mathcal{D}''_T$ from $\mathcal{D}_T$
        Define C_m as the most common class of examples in $\mathcal{D}''_T$
        Add a rule to *rule_set* with antecedent = *best_comp* and consequent = C_m
    end if
until *best_comp* = `null` **or** $\mathcal{D}_T$ *is empty*

Add a default rule to *rule_set*, predicting the majority class in $\mathcal{D}_T$

Algorithm A.1: Pseudocode of the CN2 rule set building algorithm.

The beam search follows a general-to-specific pattern of specializations on a set of complexes, *star*, which initially contains an empty complex. The size of *star* is limited to contain only the most acceptable complexes found so far. The search is performed on *star* by specializing each of the complexes in the set in every possible way.

A set, denoted *tests*, contains every possible valid attribute test. A test is generated for each value that a discrete attribute can take. In the case of a continuous attribute, one or more attribute conditions in the *tests* set determine whether the attribute's value is greater than or less than a threshold. The valid thresholds are simply the midpoints between every pair of consecutive values for the attribute in question, which are present in the training data set. A "less than" candidate condition and a "greater than" candidate condition are then defined for each threshold. Finally, the candidate conditions that represent local maxima in $acc(comp_{\bar{a}})$ values are added to *tests*. To avoid generating too many tests for each continuous attribute, only a fixed number of the "less than" and "greater than" tests with the highest $acc(comp_{\bar{a}})$ values are added to *tests*.

The set of specializations that are valid for each complex in *star* is denoted *new_star*. The *new_star* set is found by generating every possible complex that is a logical con-

```
begin function FindBestComplex(D_T, best_comp) :
    Define a set of complexes, star, containing only an empty complex
    Define tests, a set of all possible attribute tests
    while star contains complexes do
        Define an empty set of complexes, denoted new_star
        for all comp_ā ∈ star do
            for all test_ē ∈ tests, where test_ē is not tested in comp_ā do
                Specialize a complex in star, and store the result in new_star
                Create a new complex, defined as comp'_ā = comp_ā ∧ test_ē
                Store the most accurate (while still significant) complex in new_star
                if acc(comp'_ā, best_comp) and significant(comp'_ā) and beats_default(comp'_ā) then
                    Assign best_comp = comp'_ā
                end if
                Update and prune the new_star set to maintain the maximum size
                Add the newly specialized comp'_ā to the new_star set of complexes
                if size of new_star > cn2-star then
                    Remove comp_ā with lowest acc(comp_ā) from new_star
                end if
            end for
        end for
        Replace star with new_star
    end while
end function
```

Algorithm A.2: Pseudocode of the function within CN2 with the objective of finding the best complex that describes a subset of the training data, in terms of accuracy and significance.

junction between one complex in *star* and one attribute test in *tests*. Tests on attributes that are already in the complex being specialized are ignored, so that specializations already in *star* (i.e., unspecialized complexes) and contradictory complexes (i.e., complexes containing conjunctions that have a form similar to $a = 1 \wedge a = 2$) are excluded.

The heuristically best complex found thus far is maintained at each iteration. The best complex is defined in terms of the three heuristic tests described in Section A.2.2.

Firstly, the newly generated complex must be more accurate than the best complex found thus far, according to Equation (A.1). The specific accuracy heuristic used by this test is user configurable. Secondly, the new complex must be significant, in accordance with Equation (A.4). Finally, the constructed complex must be judged to have a better accuracy than that of a default rule, using Equation (A.6). A freshly generated complex that satisfies all three conditions becomes the new best complex.

Each specialized complex is added to the *new_star* set. A maximum size is defined for *new_star*, which is configurable by the user. Should the addition of the new complex result in the number of complexes in the *new_star* set exceeding the maximum size, the complex in *new_star* with the lowest $acc(comp_{\bar{a}})$ is removed from the set. Discarding such complexes prunes less promising portions of the search space.

Finally, the previous *star*, from which *new_star* was specialized, is replaced by *new_star*. The entire search process is repeated, until all possible specializations have been considered. This occurs as soon as an empty *star* set has been generated, which indicates that no further attribute tests can be added to any of the complexes already in *star*. At this point, the overall best complex represents the conjunction that most optimally describes the majority of training data examples in $\mathcal{D}_T$.

The final best complex then becomes the antecedent of a new rule, which predicts the majority class amongst the examples covered by the antecedent. The examples covered by the rule are then removed from $\mathcal{D}_T$. The beam search is repeated on the trimmed $\mathcal{D}_T$, thus finding an optimal rule complex describing the majority of examples not covered by the previously generated rule complexes. Searches continue until no further heuristically acceptable complexes can be found, or all the examples in $\mathcal{D}_T$ have been exhausted (in which case the generated rules cover every example in the training set).

As a final step, a default rule, which classifies all examples not covered by any of the preceding rules, is added to the rule set. This rule has a classification that is equivalent to the class that occurs most commonly in the full training set's data examples.

A.2.3 Additional Features

The CN2 algorithm additionally includes functionality that can be used to replace any attribute values that are missing in the original training data set. This replacement

mechanism is not considered further, because no missing attribute values occur in any of the data sets used during the experimental work that Chapter 8 presents. Clark and Niblett [38] provide further details on CN2's missing value replacement.

It is also possible to assign a "do not care" value to attributes. Such values indicate that an attribute is irrelevant for a particular data example. This feature was unused during the experimental work reported in Chapter 8, because no such values exist in any of the experimental data sets, and because none of the other tested algorithms support this attribute value. More detail is provided in the CN2 documentation [23].

In addition, CN2 can also generate unordered rule sets. However, because C4.5 generates only ordered rule sets, and to facilitate the accurate comparison of experimental results, CN2 was only used for ordered rule set building. The procedure that CN2 uses for building unordered rule sets is discussed by Clark and Boswell [37].

CN2 is also capable of ensuring that a test on an attribute is included in every rule condition that includes a test on another specified attribute. For example, it is possible for the algorithm to force any condition containing a test on the attribute `SALARY` to also contain a test for the `EMPLOYED` attribute (because people must be employed before they have salaries). This feature was also unused in the experiments of Chapter 8, because no assumptions were made about attribute interdependencies. The documentation for the CN2 algorithm [23] gives additional detail on this functionality.

Finally, it is possible to assign weights to individual training examples used by the CN2 algorithm. The inclusion of such a weight allows a data example to make either a greater or smaller contribution to the rule induction process. Again, this feature was ignored during the experimental work of Chapter 8, because appropriate weightings were unknown, and because this feature is unsupported by the other algorithms that were tested. The weighting of data examples is also discussed in the CN2 documentation [23].

A.3 Summary

This appendix provided a brief outline of the CN2 rule induction algorithm. The availability of CN2's latest implementation was briefly covered in Section A.1. The technique the algorithm employs for building rule sets was discussed in Section A.2, including the

algorithm's basic philosophy, the beam search carried out by the algorithm, and the additional features of the algorithm that were not used in this research.

The algorithm performs a so-called iterative beam search for heuristically suitable complexes (conjunctions of attribute conditions). The CN2 algorithm aims to find complexes that adequately describe a set of training examples. Each iteration of the algorithm generates a "best" complex, which is used as the antecedent of a rule. The consequent becomes the class of the majority of examples covered by the rule. Subsequent complexes (and their rules) are generated on the subset of training examples not covered by previous rules. A "best" complex is found by iteratively building a set of valid complex specializations. Complex specializations are evaluated according to heuristic tests, which estimate the accuracy and significance of a complex. During the specialization process, the worst complexes are culled, to simplify the search process.

Appendix B

The C4.5 Algorithm

This appendix describes the most important aspects of the C4.5 rule induction algorithm, used in the experiments described in Chapter 8. C4.5 was developed by J. Ross Quinlan, and evolved from ID3 [190], one of his earlier systems. His own summarizing book [193] describes C4.5 in detail. Several algorithmic improvements by Quinlan [194, 195, 197] are also included in the most recent revision of the program, namely release 8.

Section B.1 describes the availability of the implementation used in this work's empirical investigation. The C4.5 algorithm constructs decision trees from a data set (for a general description of decision trees, see Section 2.1.2.4). This process is described in Section B.2. The experiments outlined in this work required the generation of rule sets. C4.5 generates a rule set from an already-built decision tree. The rule building process is described in Section B.3. Finally, the appendix is summarized in Section B.4.

B.1 Implementation Availability

Release 8 of the original implementation of C4.5 is no longer actively maintained, but is freely available on-line for download[1]. The original algorithm has been extended in a variety of ways, and is currently available under the names C5.0 for Unix® and See5 for Microsoft® Windows®. Both implementations were only available commercially for

[1] The implementation source files and documentation for C4.5 Release 8 are available for free download from J. Ross Quinlan's home page at `http://www.rulequest.com/Personal/c4.5r8.tar.gz`.

many years, via Quinlan's company, RuleQuest Research. Recently, however, the source files for C5.0 were released to the open source software development community[2].

Unfortunately, the technical details of the algorithmic extensions within C5.0 have not been clearly documented in any published literature. Furthermore, the use of C5.0 and See5 appears to be much less prevalent in the literature than applications of C4.5. For instance, at the time of publication, C4.5 was discussed in 774 publications within the IEEE Xplore® Digital Library (available at `http://ieeexplore.ieee.org/`), while C5.0 and See5 only appeared in a combination of 89 publications. For these reasons, this work only focuses on C4.5, which was also used in the experiments of Chapter 8.

B.2 Decision Tree Building

In contrast to beam search algorithms (such as CN2, which is discussed in Appendix A), C4.5 builds a decision tree from which production rules must be further refined. The latter process is discussed in Section B.3. The tree building algorithm requires a training set, denoted $\mathcal{D}_T$. Each training example in $\mathcal{D}_T$ has descriptive attributes that are either nominal or continuous, and must have a single nominal classification attribute. The algorithm is recursive, using $\mathcal{D}_T$ to build a decision tree, $\mathcal{T}$, one node at a time.

To build a tree, a heuristic is required to quantify the quality of candidate attribute tests at newly generated node branches. These heuristic evaluations are described in Section B.2.1. Section B.2.2 covers the selection of appropriate candidate attribute tests. Section B.2.3 discusses the recursive algorithm that builds decision trees. Finally, algorithmic features not applicable in this work's context are covered in Section B.2.4.

B.2.1 Attribute Test Evaluation

The C4.5 tree building algorithm generates attribute tests for each node added to the decision tree being built. The technique relies on the evaluation of an attribute test, denoted t'. This test is generated for a subset of the training data, denoted $\mathcal{D}'_T$, as

[2] The implementation source files for C5.0 are available under the GNU General Public License for free download from RuleQuest Research at `http://www.rulequest.com/GPL/C50.tgz`.

Section B.2.3 describes. The test has $\hat{n}$ outcomes, $\mathcal{O} = \{O_1, O_2, \ldots, O_{\hat{n}}\}$, and partitions $\mathcal{D}'_T$ into $\hat{n}$ further subsets, $\{\mathcal{D}'_{T1}, \mathcal{D}'_{T2}, \ldots, \mathcal{D}'_{T\hat{n}}\}$, where each $\mathcal{D}'_{T\bar{m}}$ satisfies $O_{\bar{m}}$.

The optimality of t' is measured using a heuristic, $heur(t')$, which is based on the *gain ratio criterion*, or the older *gain criterion*. Both criteria use the information gain achieved by partitioning $\mathcal{D}'_T$ according to t'. The test with the highest $heur(t')$ is optimal. Sections B.2.1.1 and B.2.1.2 cover the gain and gain ratio criteria, respectively.

B.2.1.1 Gain Criterion

The test evaluation heuristic based on the gain criterion is different for nominal and continuous candidate attribute tests. For nominal attributes, the heuristic is simply:

$$heur(t') = gain(t') \tag{B.1}$$

where $gain(t')$ represents the value produced by applying the original gain criterion to the candidate attribute test, t', for which the heuristic is being calculated.

The gain criterion favors continuous attributes with many distinct values in $\mathcal{D}'_T$. Thus, for continuous attributes, a penalty term is introduced [197], as follows:

$$heur(t') = gain(t') - penalty(t') \tag{B.2}$$

where $penalty(t')$ is the penalty term for attribute test t'. The penalty term is derived from the minimum description length (MDL) principle [206], and is defined as:

$$penalty(t') = \frac{\log_2\big(values(t') - 1\big)}{|\mathcal{D}'_T|} \tag{B.3}$$

where $values(t')$ denotes the number of distinct values throughout $\mathcal{D}'_T$ for the attribute evaluated by candidate test t', and $|\mathcal{D}'_T|$ is the total number of examples in $\mathcal{D}'_T$.

The basic gain criterion, used in the heuristic calculations for both nominal and continuous attributes, is defined for the candidate attribute test, t', as follows:

$$gain(t') = \begin{cases} info(\mathcal{D}'_T) - info_{t'}(\mathcal{D}'_T) & \text{if two or more } |\mathcal{D}'_{T\bar{m}}| > \textit{c4.5-min} \text{ exist} \\ 0 & \text{otherwise} \end{cases} \tag{B.4}$$

where $info(\mathcal{D}'_T)$ is the entropy of the $\mathcal{D}'_T$ data subset, $info_{t'}(\mathcal{D}'_T)$ is the expected entropy after t' partitions $\mathcal{D}'_T$, and *c4.5-min* is a user-specified algorithmic parameter value.

The condition attached to the gain criterion ensures that if an attribute test is to stand a chance of being considered optimal, the data subsets associated with at least two of the outcomes of the test must contain a user-defined minimum number of training data examples. Failing to enforce this requirement has been found to result in decision trees that tend to suffer from a generally reduced predictive ability [193].

The entropy measure $info(\mathcal{D}'_T)$, which is the average information required to identify the class of an example in a subset $\mathcal{D}'_T$ of examples, is calculated as follows:

$$info(\mathcal{D}'_T) = -\sum_{m=1}^{\hat{c}(\mathcal{D}'_T)} \frac{freq(C_m, \mathcal{D}'_T)}{|\mathcal{D}'_T|} \cdot \log_2 \left(\frac{freq(C_m, \mathcal{D}'_T)}{|\mathcal{D}'_T|} \right) \tag{B.5}$$

where $\hat{c}(\mathcal{D}'_T)$ denotes the number of classes found in $\mathcal{D}'_T$, and $freq(C_m, \mathcal{D}'_T)$ denotes the number of examples in the subset $\mathcal{D}'_T$ that belong to data example class C_m.

Finally, $info_{t'}(\mathcal{D}'_T)$ is the average expected information required to identify an example's class once data subset $\mathcal{D}'_T$ has been split by test t', and is defined as:

$$info_{t'}(\mathcal{D}'_T) = \sum_{\bar{m}=1}^{\hat{n}} \frac{|\mathcal{D}'_{T\bar{m}}|}{|\mathcal{D}'_T|} \cdot info(\mathcal{D}'_{T\bar{m}})$$

where $|\mathcal{D}'_{T\bar{m}}|$ is the number of examples within the $\mathcal{D}'_{T\bar{m}}$ data subset, and $info(\mathcal{D}'_{T\bar{m}})$ is the entropy, which is calculated using Equation (B.5), over the same data subset.

Attribute tests with many outcomes produce higher gain values. In the worst case, attributes with unique values produce the maximum possible gain. This is problematic, because tests on such attributes provide no insight into the underlying training data.

B.2.1.2 Gain Ratio Criterion

The gain ratio criterion counteracts the gain criterion's bias towards tests with many outcomes. The heuristic based on the gain ratio, for nominal and continuous tests, is:

$$heur(t') = \begin{cases} gain\_ratio(t') & \text{if } gain(t') \geq ave\_gain(\mathcal{D}'_T) \\ 0 & \text{otherwise} \end{cases} \tag{B.6}$$

where $gain\_ratio(t')$ is the gain ratio computed for test t', and $ave\_gain(\mathcal{D}'_T)$ is the average value of $gain(t')$, calculated over all the valid tests that C4.5 can evaluate on the $\mathcal{D}'_T$ data subset. The condition attached to the heuristic ensures that the gain ratio is only

used if the gain for a test is large. This is necessary because tests that are very close to trivial tend to have inflated gain ratios. C4.5 uses the gain ratio criterion by default. The gain criterion is, however, retained as a user-selectable alternative.

The gain ratio criterion is essentially the simple gain criterion, defined according to Equation (B.4), normalized to avoid the bias towards tests with many outcomes. The gain ratio, like the gain heuristic, must be computed differently for nominal and continuous attributes. For nominal attributes, the gain ratio is defined as:

$$gain\_ratio(t') = \frac{gain(t')}{split\_info(t')} \tag{B.7}$$

where $split\_info(t')$ is a measure of the information that can potentially be created through the act of splitting $\mathcal{D}'_T$ into $\hat{n}$ further separate subsets of data.

For continuous attributes, the gain ratio criterion must again incorporate a penalty term, due to the fact that the gain criterion has a bias towards continuous attributes with many unique values in $\mathcal{D}'_T$. The gain ratio in this case is defined as:

$$gain\_ratio(t') = \frac{gain(t') - penalty(t')}{split\_info(t')} \tag{B.8}$$

where $penalty(t')$ is the same penalty term that is used by the gain criterion heuristic for continuous valued attributes, and is defined according to Equation (B.3).

Finally, the $split\_info(t')$ measure for candidate attribute test t', by which the gain criterion value is normalized, is defined according to the following equation:

$$split\_info(t') = -\sum_{\bar{m}=1}^{\hat{n}} \frac{|\mathcal{D}'_{T\bar{m}}|}{|\mathcal{D}'_T|} \cdot \log_2 \left(\frac{|\mathcal{D}'_{T\bar{m}}|}{|\mathcal{D}'_T|} \right)$$

The gain ratio shifts the heuristic's bias away from tests with many outcomes, because such tests have increased $split\_info(t')$ values, resulting in lower gain ratio values. However, almost trivial tests have very small values for $split\_info(t')$, resulting in large gain ratios. The condition attached to Equation (B.6) counteracts this situation.

Because Equations (B.7) and (B.8) make use of the gain criterion of Equation (B.4), the gain ratio is subject to the same condition that the gain criterion is. In other words, if at least two of the data subsets partitioned by attribute test t' do not contain at least *c4.5-min* data examples, both the gain and gain ratio criteria have values of 0.

B.2.2 Attribute Test Selection

Whenever an attribute test must be added to a decision tree, the C4.5 algorithm must propose a set of possible candidate tests. One candidate is proposed for each attribute. It is possible to base a candidate on either a nominal or a continuous attribute.

All the candidates attribute tests are evaluated using one of the $heur(t')$ measures discussed in the previous section. Should the gain criterion be chosen, Equation (B.1) computes the heuristic values for nominal attribute tests, while Equation (B.2) is used for continuous valued tests. If the default gain ratio criterion is used, Equation (B.6) computes the heuristic values, but uses Equation (B.7) as the gain ratio for nominal attributes, and Equation (B.8) for tests on attributes with continuous values.

The nominal or continuous candidate test with the highest value for the appropriate heuristic is considered to be optimal, and is chosen to be added to the tree, as outlined in Section B.2.3. The processing of nominal and continuous attributes also differs in terms of the method that is used for determining the set, $\mathcal{O}$, of possible test outcomes.

B.2.2.1 Nominal Attributes

The simplest candidate tests that C4.5 can generate are for nominal attribute values, where each potential attribute value represents a possible test outcome. Each $O_{\bar{m}}$ is then equivalent to a value for the attribute in question, and $\hat{n}$ is the number of values.

B.2.2.2 Continuous Attributes

The C4.5 algorithm must construct a new test for each continuous attribute [197]. This is achieved by building a set of candidate threshold values, which define a number of possible tests on the same attribute. C4.5 uses an approach that is similar to the method CN2 uses to generate a set of candidate continuous valued test thresholds: thresholds are simply the midpoints between consecutive values that the attribute takes on, over the $\mathcal{D}'_T$ training example subset for which the attribute test is being generated.

Each of the possible threshold values is used to define a separate candidate test for the continuous attribute. A heuristic value based on the penalized gain criterion is calculated for each threshold-based candidate test, according to Equation (B.2), regardless

of whether gain or gain ratio is being used to compare attribute tests to one another. The threshold with the highest adjusted heuristic value is chosen for the condition of the candidate test on the attribute. The test has only two outcomes, which means that $n = 2$. The first outcome matches attribute values less than or equal to the threshold, while the second outcome covers attribute values greater than the threshold.

B.2.3 Recursive Divide and Conquer

C4.5 builds trees using a recursive divide and conquer approach. Each tree sub-branch is refined on the subset of examples classified by that branch. The algorithm is "greedy", with no backtracking to find a better solution once a sub-branch has been refined. Algorithm B.1 illustrates an outline of the tree building sub-algorithm of C4.5.

The recursive algorithm has three base cases and one recursive case, and is executed on a sub-tree that requires specialization, in conjunction with a data subset upon which the specialization will be based. Initially, the algorithm is executed on an empty node, which becomes the root of the decision tree, and the full training data set.

The algorithm begins by encountering a data subset, $\mathcal{D}'_T$, which contains examples from several classes. The algorithm then generates every attribute test that is valid for the data set, according to the procedures discussed in Section B.2.2. The optimal test, which has $\hat{n}$ outcomes, is found within the set of valid tests, according to the heuristics covered in Section B.2.1. Next, a non-leaf node, with $\hat{n}$ branches to sub-trees $\{\mathcal{T}'_1, \mathcal{T}'_2, \ldots, \mathcal{T}'_{\hat{n}}\}$, is defined for this test. The test sub-divides $\mathcal{D}'_T$ into further subsets, each of which satisfies one of the attribute test outcomes. The procedure is then repeated recursively on each new sub-tree, and the data subset that satisfies the test outcome leading to the new sub-tree. Consequently, each sub-tree is further refined.

The recursive algorithm's first base case is invoked when a data subset is reached that contains only examples belonging to a single class. A perfectly homogeneous data subset has thus reached this point in the tree. This means that the sub-tree must become a leaf node, which predicts the class of the training examples in the data subset.

The second base case handles a situation in which no examples satisfy a test outcome, and the outcome's new sub-tree thus has an empty data subset. The new sub-tree again becomes a leaf node, but with the most frequent class at the parent node as the outcome.

Define a decision tree, $\mathcal{T}$, consisting of an empty node
Initialize training set $\mathcal{D}_T$
call function BuildSubTree($\mathcal{T}$, $\mathcal{D}_T$)

begin function BuildSubTree($\mathcal{T}'$, $\mathcal{D}'_T$) **:**

if $\mathcal{D}'_T$ *contains 1 or more examples belonging only to a single class,* C_m **then**
Make $\mathcal{T}'$ a leaf node, identifying class C_m

else if $\mathcal{D}'_T$ *contains 0 examples* **then**
Identify the most frequent class C_m at the parent node
Make $\mathcal{T}'$ a leaf node, identifying class C_m

else if *every possible attribute test,* t'*, produces gain*$(t') \leq 0$ **then**
Identify the class, C_m, which is the most common in $\mathcal{D}'_T$
Make $\mathcal{T}'$ a leaf node, identifying class C_m

else
Define *candidates*, a set of all valid attribute tests, according to Section B.2.2
Associate the appropriate *heur*(t') from Section B.2.1 with each $t' \in$ *candidates*
Select attribute test, $t' \in$ *candidates*, with highest *heur*(t') within *candidates*
Identify the $\hat{n}$ outcomes, $\mathcal{O} = \{O_1, O_2, \ldots, O_{\hat{n}}\}$, of attribute test t'
Make $\mathcal{T}'$ a non-leaf node, with $\hat{n}$ branches for each $O_{\bar{m}} \in \mathcal{O}$
Define $\hat{n}$ empty sub-trees $\{\mathcal{T}'_1, \mathcal{T}'_2, \ldots, \mathcal{T}'_{\hat{n}}\}$ connected to each branch of $\mathcal{T}'$
Divide $\mathcal{D}'_T$ into $\hat{n}$ subsets $\{\mathcal{D}'_{T1}, \mathcal{D}'_{T2}, \ldots, \mathcal{D}'_{T\hat{n}}\}$, where each $\mathcal{D}'_{T\bar{m}}$ satisfies $O_{\bar{m}}$
for all $\mathcal{D}'_{T\bar{m}} \in \mathcal{D}_T$ *and* $\mathcal{T}'_{\bar{m}}$ *that corresponds to* $\mathcal{D}'_{T\bar{m}}$ **do**
call function BuildSubTree($\mathcal{T}'_{\bar{m}}$, $\mathcal{D}'_{T\bar{m}}$)
end for

end if

end function

Algorithm B.1: Pseudocode of the C4.5 sub-algorithm responsible for building trees.

The final base case, like the recursive case, deals with data subsets containing examples from a mixture of different classes. However, no tests have a gain criterion value above 0 for the data subset. Every test is thus insignificant, and no further refinement of the subtree will be fruitful. In this case, as for the first recursive case, the sub-tree becomes a leaf node predicting the most frequent class in the data subset.

Figure B.1 shows an example of the algorithm's operation. Using the full $\mathcal{D}_T$, test t'_1 is chosen. Examples satisfying the first outcome of t'_1 are used to select test t'_2. Finally, sets of examples satisfying the first and second outcomes of t'_2, and the second outcome of t'_1, consist of homogeneous data subsets, which are used to generate leaf nodes.

B.2.4 Additional Features

C4.5 supports missing value replacement, but the feature is unused because the benchmark data sets used in Chapter 8 lack such values. Consequently, the missing value replacement scheme used by C4.5 is not considered further in this dissertation.

It is also possible for C4.5 to create attribute tests for nominal attributes with outcomes that possibly include groups of values. Such tests effectively create disjunctions in rule conditions. Because the other algorithms considered in the experimental comparisons of Chapter 8 produce rule conditions that are simple conjunctions of attribute tests, grouped outcomes are not investigated further within this research work.

Tree pruning is also supported, and removes nodes from a generated decision tree to produce a more compact and understandable representation. However, the feature is unused during the experimental work, because the study focused on rule sets, and C4.5 builds rule sets from unpruned trees. Pruning is therefore not discussed further.

The process of windowing grows a decision tree from a small subset of training data, known as the window. Misclassified training examples from outside of the window are added to the window, and a new tree is built from the larger data set. This is repeated until a sufficiently accurate tree is found. However, windowing is essentially applicable to any rule extractor, and is therefore not considered in greater detail.

Finally, bagging and boosting has been tested with C4.5 [196]. Like windowing, these methods are also applicable to any rule extraction algorithm, including the other methods investigated in this work. Bagging and boosting are therefore not focused on.

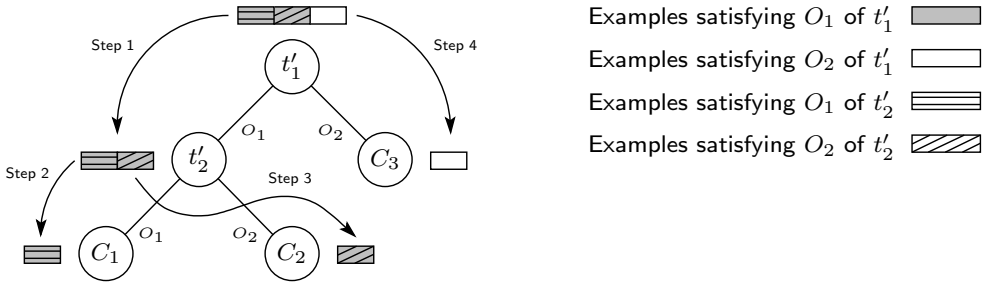


Figure B.1: An example of the recursive tree building performed by the C4.5 algorithm.

B.3 Generating a Rule Set

C4.5 uses a relatively complex method to translate an unpruned tree output into a set of production rules. The production rule set has the form described in Section 2.1.2.4, where the rule antecedents are only conjunctions of attribute tests. The rules are ordered, and a default rule is also generated to classify uncovered data examples.

The rule extraction process consists of several phases, each of which is discussed separately. Section B.3.1 discusses the initial extraction of rules from an unpruned decision tree, and the first pruning phase, which removes redundant conditions from the rules. Section B.3.2 discusses the second pruning phase, which involves the removal of redundant rules. The pruned rule set requires ordering, and the addition of a default rule, which Section B.3.3 discusses. The last step of the rule extraction process performs a final pruning of globally redundant rules, which is outlined in Section B.3.4.

B.3.1 Initial Rule Extraction and Condition Pruning

The algorithm first generates a rule for every leaf node in the tree. Each leaf node's outcome becomes the consequent of a rule. The rule's antecedent is the conjunction of non-leaf test conditions along the path traced from the tree's root to the leaf. Figure B.2 shows an example of a decision tree converted to an initial set of rules.

These initial rules are unoptimized, and generally relatively redundant in nature. As each initial rule is generated, all the redundant conditions in the rule antecedent are removed. Conditions are pruned one at a time in a greedy fashion, with no backtracking once a condition has been deleted, until no further conditions can be removed.

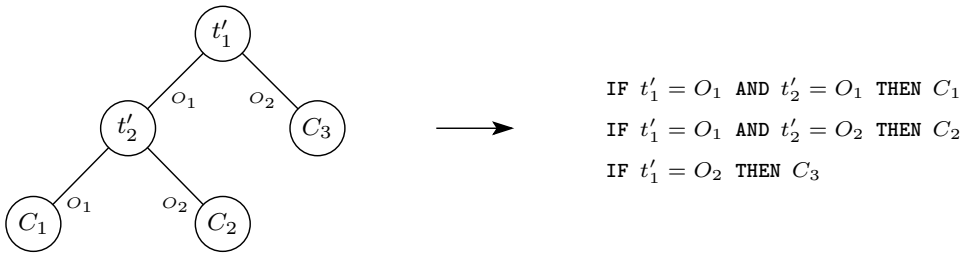


Figure B.2: An example of the initial conversion of a C4.5 decision tree into a rule set.

C4.5 must assess the importance of each condition, to decide which should be deleted at each pruning step. C4.5 has two measures for the desirability of a condition, namely the *pessimistic error rate*, and *Fisher's exact significance test*. The former method is always used for condition pruning. The latter is an older approach, used in addition to the pessimistic error rate when invoked by means of an algorithmic parameter.

A rule's error rate is the percentage of classification errors made by the rule over the full training set. The pessimistic error rate, $pess\_err(rule)$, of a rule, denoted *rule*, is a more accurate estimate of the rule's classification performance on unseen data, and is the upper confidence interval bound of the rule's error rate, at a user-specified confidence level denoted *c4.5-pess*. Condition *cond* is removable if the pessimistic error rate of the rule with *cond* removed does not exceed the pessimistic error rate for the original rule:

$$prunable(cond) = \begin{cases} true & \text{if } pess\_err(rule') \leq pess\_err(rule) \\ false & \text{otherwise} \end{cases}$$

where $rule'$ denotes the original unpruned rule with *cond* removed, and $pess\_err(rule')$ therefore represents the pessimistic error rate computed for this pruned rule. A lower value for the user-defined confidence level, which is used to calculate the pessimistic error rates in the equation, tends to result in more drastic pruning of conditions.

Fisher's exact significance test [76, 78] was the only method for judging condition pruning in earlier releases of C4.5 [191]. The test is based on all the data examples from the training set that are covered by the pruned rule from which the condition has been removed. Each data example either satisfies or does not satisfy the condition being considered for removal, and is either correctly or incorrectly classified. A *contingency table* records the numbers of examples in each category, as illustrated in Figure B.3.

Examples covered by $rule'$, which still satisfy $cond$, and belong to C_m	Examples covered by $rule'$, which still satisfy $cond$, and do not belong to C_m
Examples covered by $rule'$, which do not satisfy $cond$, and belong to C_m	Examples covered by $rule'$, which do not satisfy $cond$, and do not belong to C_m

Figure B.3: The format of the contingency tables used by Fisher's exact significance test. Pruning *cond* from *rule*, which predicts class C_m, produces a simplified rule, denoted $rule'$.

The value of the significance test, performed on the contingency table for $rule'$ (the rule produced by pruning *cond* from *rule*) is $exact\_sig(rule')$. The significance test value signifies the probability that the division of training data examples in the contingency table has occurred by chance. A condition is considered to be prunable when the significance value is greater than a user-specified significance threshold, as follows:

$$prunable(cond) = \begin{cases} true & \text{if } exact\_sig(rule') > c4.5\text{-}Fisher \\ false & \text{otherwise} \end{cases}$$

where *c4.5-Fisher* is the significance threshold, which is specified by the user as an algorithmic parameter. A lower value for the threshold results in heavier pruning.

All conditions are tested using the pessimistic error rate and, optionally, Fisher's significance test. A simplification must be performed if the tests performed on the conditions conclude that at least one condition is removable. In such a case, the algorithm records the pessimistic error rate produced when each condition is removed. The condition giving the lowest error rate is removed, regardless of whether the pessimistic error rate or Fisher's significance test concluded that the condition should be pruned.

The entire pruning process is repeated until no further conditions are deemed removable from the rule by the pessimistic error rate, as well as Fisher's significance test, if the test has been enabled by the user. Once a generated rule has been fully pruned of redundant conditions, the next rule is generated and pruned. The process is continued until a pruned rule has been created for every leaf node in the decision tree.

It is possible for every condition in a rule to be removed due to the condition pruning process. In such a case, the rule is naturally not added to the rule set. Furthermore,

should the pruning process result in a rule that is a duplicate of another rule already in the rule set, the pruned rule is also discarded. Finally, rules that are considered too inaccurate are omitted from the rule set. This final condition is assessed as follows:

$$accurate(rule) = \begin{cases} true & \text{if } pess\_err(rule) < default\_err(rule) \\ false & \text{otherwise} \end{cases}$$

where *rule* has undergone a complete cycle of antecedent condition pruning before possibly being added to the rule set, and $default\_err(rule)$ is computed as follows:

$$default\_err(rule) = 1 - \frac{class(\mathcal{D}_T, rule) + 1}{|\mathcal{D}_T| + 3}$$

where $class(\mathcal{D}_T, rule)$ is the number of data examples in $\mathcal{D}_T$ with the same class as the antecedent of *rule* predicts, and $|\mathcal{D}_T|$ is the number of data examples in $\mathcal{D}_T$.

Algorithm B.2 shows the process for building and pruning rules as pseudocode. Comments indicate the parts of the program responsible for rule building and pruning as Steps 1 and 2, respectively. Comments also mark the instructions performing the detection of prunable conditions and the actual pruning as Steps 2.1 and 2.2, respectively.

B.3.2 Pruning of Redundant Rules

Once all rule antecedents have been simplified, a second pruning phase is performed [195]. C4.5 finds entire rules that are deemed unnecessary, and removes these from the rule set. Rules with a predicted class in common are grouped together, for all classes.

The redundant rule pruning phase selects only a subset of the rules contained in each group, while eliminating the unselected rules. The MDL principle [206] is applied to guide the selection of rule subsets. This principle is also used in the calculation of the penalty term for the gain criterion, which was discussed in Section B.2.1.1.

The MDL method treats the selection of a rule subset as a two-class problem. The first class is the consequent of the rule group in question, and the second includes every other class in the training set. The process must choose a rule subset that covers training set examples of the first class, while not covering examples in the second class.

An MDL encoding is built up out of two components, namely a theory cost and an exception cost. The former encodes the rules in a subset, while the latter identifies

Build a decision tree, $\mathcal{T}$, according to Algorithm B.1
Define the set of $\hat{l}$ leaf nodes in $\mathcal{T}$ as $leaves = \{leaf_1, leaf_2, \ldots, leaf_{\hat{l}}\}$
Define a rule set, denoted *rule_set*

for all $leaf_{\bar{n}} \in leaves$ **do**

Step 1: Generate a new rule for each leaf in the decision tree
Define a new rule, denoted *rule*, with consequent = outcome of $leaf_{\bar{n}}$
Trace a route, denoted $\mathcal{R}$, from the root of $\mathcal{T}$ to $leaf_{\bar{n}}$
Add non-leaf test outcomes along $\mathcal{R}$ to the antecedent conjunction of *rule*

Step 2: Prune the conditions of the new rule
repeat:
Define the best condition for removal, denoted *best_cond*

Step 2.1: Determine if any conditions are prunable
for all $cond \in rule$ **do**
Compute *prunable*(*cond*), as described in Section B.3.1

if *cond has a lower pessimistic error rate than best_cond* **then**
Assign *best_cond* = *cond*
end if
end for

Step 2.2: Prune the condition with the lowest pessimistic error rate
if *prunable*(*cond*) = *true for one or more cond* $\in$ *rule* **then**
Prune *best_cond* from the antecedent of *rule*
end if
until *no further conditions can be removed*

Step 3: Add the new pruned rule to the rule set
if *rule contains conditions* ***and*** *rule* $\notin$ *rule_set* ***and*** *accurate*(*rule*) **then**
Add *rule* to *rule_set*
end if

end for

call function RefineRuleSet(*rule_set*), outlined in Algorithm B.3

Algorithm B.2: Pseudocode of the C4.5 sub-algorithm responsible for rule set building.

training examples that are misclassified by the subset. Furthermore, the exception cost is biased, in order to penalize subsets with predicted class distributions that differ from the observed class distributions in the original training data. The cost of encoding a rule increases in the presence of redundant attributes. To compensate for this, a user-specified redundancy factor, *c4.5-redun*, is incorporated into the MDL encoding.

Finally, for each rule group, the pruning process must select the subset of rules that minimize the biased MDL encoding. The selection involves a search, which is conducted differently depending on the number of rules in the group. If the group contains ten or fewer rules, every possible subset of rules that can be created is evaluated. The subset with the lowest MDL encoding then replaces the rule group being pruned.

When a group contains more than ten rules, several local greedy searches are performed to find a subset with an approximately minimal encoding. In this scenario, each search starts with a set, called the search set, containing rules randomly selected from the rule group. For each successive search, the probability of including each rule in the group increases from 0% to 100%, in increments of 10%. Thus no rules are selected for the first search, roughly 10% of the rules are chosen for the second search, approximately 20% of the rules are used for the third search, and so on, until the last search set includes all the rules in the group. Each initial search set must then be refined.

Each search begins by computing the MDL encoding for the initial search set. The objective of the search is to repeatedly either add or delete a single rule, until no further decrease in the MDL encoding for the search set can be achieved. A value is computed for each rule in the full group, which indicates the reduction of the MDL encoding that will result from the rule either being added to or removed from the search set. The rule with the largest reduction is then either added or removed, as appropriate, with no backtracking. This process is repeated until every possible rule addition and removal will result in no improvement to the MDL encoding of the search set.

As all of the local searches proceed, the search set with the lowest MDL encoding value over all the searches thus far is recorded. Once all the searches have been completed, the search set with the lowest MDL encoding represents the approximately optimal subset of rules for the rule group in question, and is finally chosen to replace the entire rule group being optimized. Step 1 in Algorithm B.3 illustrates the rule pruning process.

begin function RefineRuleSet(*rule_set*) **:**

Step 1: Prune redundant rules

for all $C_m \in$ *consequents of rule_set* **do**

Define a rule group, *group*(C_m), containing all rules in *rule_set* predicting C_m

if *group*(C_m) *contains ten or fewer rules* **then**

Generate every possible subset of the rules contained in *group*(C_m)

Replace *group*(C_m) with the subset giving the lowest MDL encoding

else

for all *prob* $\in \{0\%, 10\%, 20\%, \ldots, 100\%\}$ **do**

Build *search_set* storing each *rule* $\in$ *group*(C_m) with probability *prob*

repeat:

for all *rule* $\in$ *group*(C_m) **do**

Find reduction in *search_set* MDL for adding or removing *rule*

end for

Add or remove *rule* $\in$ *search_set* with maximal MDL reduction

until *no modification reduces the MDL encoding of search_set*

end for

Replace *group*(C_m) with the *search_set* that has the lowest MDL encoding

end if

end for

Step 2: Order rules and define a default rule

Order rule groups in ascending order of the number of false positives generated

Add a default rule to *rule_set*, predicting the majority class of unclassified examples

Step 3: Perform final rule pruning

while *a rule can be removed from rule_set to reduce misclassifications* **do**

Remove the first rule from *rule_set* that reduces misclassifications

end while

end function

Algorithm B.3: Pseudocode of the C4.5 sub-algorithm responsible for rule set refining.

B.3.3 Rule Ordering and Default Rule Definition

Once this series of optimizations has been completed, the resultant rule set is ordered. This is done by ordering the rule groups that classify each class, such that those groups with fewer *false positives* (examples that are covered by the rules, but are not members of the predicted class of the rules) are placed earlier in the list. A default classification is also added, which predicts the most prevalent class of the examples not classified by any rule (ties are resolved in favor of the class with the higher absolute frequency).

The ordering of rules within a rule group is unimportant, because all rules in a group predict the same class, and thus do not interfere with one another. The phase handling rule ordering and default rule construction is marked as Step 2 in Algorithm B.3.

B.3.4 Final Rule Pruning

Finally, the entire rule set is once again evaluated, in order to find additional rules that can be removed. If a rule's omission reduces the rule set's total number of misclassifications measured over the training data set, that rule is eligible for pruning. The first of these candidate rules is removed before the rule set is re-evaluated for further pruning. This step evaluates the entire rule set as a whole, based on the set's overall performance. Step 3 in the pseudocode of Algorithm B.3 marks the final rule pruning phase.

B.4 Summary

This appendix dealt with the basic operation of the C4.5 algorithm. Section B.1 briefly summarized the availability of the standard implementation of the C4.5 algorithm, while Sections B.2 and B.3, respectively, described the two phases this algorithm requires for rule set induction. The first phase generates an initial decision tree, while the second phase performs the translation of this tree into a concise production rule set.

The initial tree building phase uses a greedy, recursive divide and conquer algorithm. The algorithm defines a non-leaf node, which specifies an attribute test and the outcomes of the test, and then refines this test by adding sub-nodes. Leaf nodes are added when no further specialization is required, and a classification can be decided upon.

A decision tree built by the algorithm is refined into a set of rules by initially creating a single rule for every leaf node outcome. This initial set of rules is then refined by removing unnecessary conditions and rules. The rules are then ordered, after which a default rule is added. As a final step, a further rule pruning optimization is performed, with the objective of further optimizing the entire rule set's overall performance.

Appendix C

Derived Work

This appendix lists the academic publications derived from the work presented in this dissertation. Accepted publications are listed in Section C.1, while details of work in progress at the time of this dissertation's writing are contained in Section C.2:

C.1 Accepted Publications

- W. S. van Heerden and A. P. Engelbrecht. Exploratory data analysis and data mining using self-organising feature maps. In C. de Villiers and T. Alexander, editors, *Proceedings of the Postgraduate Research Symposium, Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 39–40, Pretoria, South Africa, 17 September 2003. SAICSIT.

  Summary: This paper outlines the distinction between EDA and DM, described in Section 2.5 of this dissertation. The five categories of SOM-based EDA that are discussed in Section 4.5 of this research work (namely characterization, feature selection, sensitivity analysis, interpolation, and trend analysis), are described. The SIG* algorithm (discussed in Section 7.3 of this dissertation) is broadly outlined, and the possibility of the algorithm generating inaccurate conditions on attribute values (mentioned in Section 7.3.5) is hypothesized. Finally, the paper presents a high-level description of the HybridSOM framework covered in Section 7.4.

- W. S. van Heerden and A. P. Engelbrecht. A comparison of map neuron labeling approaches for unsupervised self-organizing feature maps. In *Proceedings of the IEEE International Joint Conference on Neural Networks (part of the IEEE World Congress on Computational Intelligence)*, pages 2139–2146, Hong Kong, 1–8 June 2008. Available on-line at `doi:10.1109/IJCNN.2008.4634092`

  Summary: This paper discusses the three supervised neuron labeling techniques described in Section 6.2. Three of the unsupervised labeling techniques that were described in Section 6.3 of this research work (namely exploratory labeling, Serrano-Cinca's absolute weight value significance, and the LabelSOM method) are also outlined in the paper. An additional focus of the paper is on the application of neuron labeling to supervised, semi-supervised, and unsupervised SOMs, as investigated in Section 6.4 of this work. Finally, empirical results are presented, which compare example-centric cluster labeling (configured with both Ward clustering and k-means clustering) and weight-centric neuron labeling.

- W. S. van Heerden and A. P. Engelbrecht. HybridSOM: A generic rule extraction framework for self-organizing feature maps. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (part of the IEEE Symposium Series on Computational Intelligence)*, pages 17–24, Nashville, Tennessee, United States of America, 30 March–2 April 2009. Available on-line at `doi:10.1109/CIDM.2009.4938624`

  Summary: The principal contribution of this paper is a detailed description and critical discussion of the HybridSOM framework, as presented in Section 7.4 of this dissertation. The paper contrasts the HybridSOM framework to the SIG* algorithm, which is presented in Section 7.3 of this dissertation, and the boundary-based rule extraction algorithm proposed by Malone *et al*, which is investigated in Section 7.2 of this research. The paper experimentally compares the HybridSOM framework configured with CN2 and C4.5 to the stand-alone CN2 and C4.5 algorithms, while omitting a comparison to SIG*. Finally, the paper discusses the general viability of SOM-based rule extraction, as presented in Section 7.6.

- W. S. van Heerden and A. P. Engelbrecht. Unsupervised weight-based cluster labeling for self-organizing maps. In P. A. Estévez, J. C. Príncipe, and P. Zegers, editors, *Advances in Self-Organizing Maps: Proceedings of the 9th International Workshop*, volume 198 of *Advances in Intelligent Systems and Computing*, pages 45–54, Santiago, Chile, 12–14 December 2012. Springer. Available on-line at `doi:10.1007/978-3-642-35230-0_5`

  Summary: This paper differentiates the general classes of supervised and unsupervised neuron labeling schemes, which this dissertation details in Section 6.1 and graphically represents in Figure 6.1. The main contribution of this publication is the introduction of the unsupervised weight-based cluster labeling approach, which is discussed in Section 6.3.3.2 of this dissertation. The labeling method is described in detail, and several examples of the generated neuron labelings are presented. The paper also critically discusses unsupervised weight-based cluster labeling in terms of the advantages and disadvantages associated with the technique.

C.2 Work in Progress

- W. S. van Heerden and A. P. Engelbrecht. Unsupervised neuron labeling for self-organizing maps: A taxonomy of generalized techniques. Submitted for review to *ACM Computing Surveys*, December 2016.

  Summary: This article focuses on the SOM-based unsupervised neuron labeling methods that are presented in Section 6.3 of this dissertation. Unsupervised neuron labeling is differentiated from supervised neuron labeling, as described in Section 6.1. The novel taxonomy of unsupervised labeling methods, outlined in Figure 6.4, is described. The article also provides a detailed survey of the available approaches in the category of unsupervised neuron labeling, which are discussed in Section 6.3 of this dissertation. Particular attention is given to the significance measures and attribute selection schemes available for the unsupervised weight-based and example-based labeling methods described in Sections 6.3.3 and 6.3.4, respectively. All the described techniques are compared and critically discussed.

- W. S. van Heerden and A. P. Engelbrecht. An empirical analysis of supervised neuron labeling methods for self-organizing maps. Work in progress for journal submission.

  Summary: The primary objective of this article is to summarize the novel experimental work reported in Section 8.3 of this dissertation, which comparatively analyzed the performance of supervised neuron labeling techniques used on unsupervised self-organizing maps. The article also provides a brief background on the differences between supervised and unsupervised neuron labeling, which is covered in Section 6.1 of this dissertation, and an overview of the compared supervised neuron labeling algorithms, which are discussed in Section 6.2 of this research.

- W. S. van Heerden and A. P. Engelbrecht. An empirical investigation of unsupervised data mining algorithms based on self-organizing feature maps. Work in progress for journal submission.

  Summary: This article primarily focuses on reporting the results of the empirical investigation into the comparative performance of the SOM-based data mining approaches and classical data mining algorithms that is presented in Section 8.4 of this dissertation. Background is also provided on the SIG* and HybridSOM techniques used in the comparison, which are respectively discussed in Sections 7.3 and 7.4 of this work. The general justifications for the use of SOM-based data mining techniques, which is presented in Section 7.6, are also enumerated.

Appendix D

Symbol Definitions

This appendix lists and defines the mathematical symbols used in this dissertation. Symbols are listed in order of appearance, and grouped according to relevant chapters.

D.1 Foundations of EDA and DM

t	Moment in time or iteration
$\hat{S}_t$	State of an environment at time t
H	Name of a relation
$\tilde{n}$	Degree of a relation schema
A_l	Attribute l of a relation schema
$dom(A_l)$	Domain of attribute A_l
$\tilde{m}$	Number of tuples in a relation
$tup_{\bar{t}}$	Tuple $\bar{t}$ of a relation
f_l	Value l in a tuple
obj	An object encapsulated by a state within an environment
$\mathcal{V}$	Set of all possible states within an environment
$\mathcal{F}$	State transition function within an environment
$\mathcal{C}$	Set of all classes within a model
C_m	Class m in the set $\mathcal{C}$

B_m	Class condition for class C_m
$\mathcal{P}$	Classification function of a model
D_m	Class description of class C_m
$\mathcal{F}'$	Model transition function of a model
$\mathcal{A}$	Set of all attributes in a relation schema
$\mathcal{A}_{cls}$	Set of classification attributes in a relation schema
$\mathcal{A}_{des}$	Set of descriptive attributes in a relation schema
f'_l	Normalized value for f_l
$f_{l,min}$	Minimum possible value for f_l
$f_{l,max}$	Maximum possible value for f_l
$f'_{l,min}$	Minimum possible value f'_l should have
$f'_{l,max}$	Maximum possible value f'_l should have

D.2 Self-Organizing Feature Maps

I	Dimensionality of the input space for a SOM
$\mathcal{D}_T$	Training set containing training patterns
$\mathcal{D}$	Data set containing all available data patterns
P_T	Number of patterns in training set $\mathcal{D}_T$
$\vec{z}_s$	Data vector s in training set $\mathcal{D}_T$
$z_{s\hat{w}}$	Input parameter $\hat{w}$ of training vector $\vec{z}_s$
Y	Total number of rows within a SOM map grid
X	Total number of columns within a SOM map grid
n_{yx}	Neuron at row y and column x
c_{yx}	Map grid coordinate of neuron at row y and column x
$\vec{w}_{yx}$	Weight vector associated with neuron n_{yx}
w_{yxv}	Weight component v of weight vector $\vec{w}_{yx}$
V	Dimensionality of an arbitrary vector
$\vec{q}$	An arbitrary vector

$q_{\hat{p}}$	Component $\hat{p}$ of an arbitrary vector
n_{ba}	Best matching unit (BMU)
c_{yx}	Map grid coordinate of best matching unit (BMU)
$\vec{w}_{ba}$	Weight vector of best matching unit (BMU)
b	Row of best matching unit (BMU)
a	Column of best matching unit (BMU)
$\vec{w}_{yx}(t)$	Value of $\vec{w}_{yx}$ at training iteration t
$\Delta\vec{w}_{yx}(t)$	Change applied to $\vec{w}_{yx}$ at training iteration t
$\Delta w_{yxv}(t)$	Change applied to $\vec{w}_{yxv}$ at training iteration t
$h_{ba,yx}$	Neighborhood function for $\vec{w}_{yx}$ in relation to $\vec{w}_{ba}$
$Q_T(t)$	Training quantization error at training iteration t
W	Size of a sliding window
$\hat{t}$	Training iteration offset from the end of a sliding window
$\overline{Q}_T(t)$	Moving average of Q_T at iteration t
$d_T(t)$	Moving standard deviation of Q_T at iteration t
$\Delta w_{ave}(t)$	Average component weight change at iteration t
c'_{yx}	Map coordinate of $\hat{o}^{\text{th}}$ closest neuron to n_{yx} in map space
c''_{yx}	Map coordinate of $\hat{o}^{\text{th}}$ closest neuron to n_{yx} in weight space
$\vec{w}'_{yx}$	Weight vector of $\hat{o}^{\text{th}}$ closest neuron to n_{yx} in map space
$\vec{w}''_{yx}$	Weight vector of $\hat{o}^{\text{th}}$ closest neuron to n_{yx} in weight space
β	Factor for upper bound on the optimal neuron count
$\eta(t)$	Learning rate factor at training iteration t
τ_1	Constant affecting rate of the learning rate factor decay
$\sigma(t)$	Gaussian kernel width at training iteration t
τ_2	Constant affecting rate of the neighborhood radius decay

D.3 SOM-Based Visualization and EDA

$res(\vec{z}_s, \vec{w}_{yx})$	Euclidean response of $\vec{w}_{yx}$ to $\vec{z}_s$
$\vec{r}_{yx}$	Projection vector, which represents $\vec{w}_{yx}$
$weights$	Set of all weight weight vectors in a map grid
$\mathcal{Y}$	Set containing all unique weight vector pairs in a map grid
δ	Set of two non-equivalent weight vectors
$E(t)$	Sammon's mapping error function at iteration t
$r_{yx\hat{e}}$	Component $\hat{e}$ of projection vector $\vec{r}_{yx}$
$\Delta r_{yx\hat{e}}(t)$	Change applied to $r_{yx\hat{e}}$ at iteration t
φ	Constant factor applied to $\Delta r_{yx\hat{e}}(t)$
$\vartheta(\vec{w}_{ba})$	Mean distance between $\vec{w}_{ba}$ and all $\vec{z}_s$ mapped to n_{ba}

D.4 Emergent Neuron Cluster Discovery

$\mathcal{L}$	Set of clusters
k	Size of $\mathcal{L}$ (derived from k-means algorithm, cf. Section D.7)
S_i	Cluster i in $\mathcal{L}$
o_i	Number of weight vectors in S_i
$\vec{g}_i$	Centroid vector of S_i
$index$	Davies-Bouldin index
$intra(S_i)$	Intra-cluster distance within S_i
$inter(S_i, S_j)$	Inter-cluster distance between S_i and S_j
map	Map grid of a SOM, including weight vectors
$ward\_dist(S_i, S_j)$	Ward distance between clusters S_i and S_j

D.5 Map Neuron Labeling

$label_{yx}$	Label for n_{yx}
L_i	Label for S_i

M_{yx}	Set of $\vec{z}_s$ mapped to neuron n_{yx}
N_i	Set of $\vec{z}_s$ mapped to cluster S_i
$\vec{z}_e$	Data vector of the best matching example (BME)
$\bar{q}$	Size of a set of the closest data examples to a neuron
$sig(A_l, n_{yx})$	Significance of attribute A_l within n_{yx}
$sig(A_l, S_i)$	Significance of attribute A_l within S_i
$mean(w_{yxl}, S_i)$	Mean of w_{yxl} over all $\vec{w}_{yx}$ in S_i
$out(S_i)$	Set of all emergent clusters in $\mathcal{L}$, excluding S_i
$sig(A_l, M_{yx})$	Significance of attribute A_l within M_{yx}
$out(N_i)$	Set of all $\vec{z}_s$ not in N_i
$sig(A_l, N_i)$	Significance of attribute A_l within N_i
$mean(A_l, N_i)$	Mean of A_l over all $\vec{z}_s$ in N_i
$mean\big(A_l, out(N_i)\big)$	Mean of A_l over all $\vec{z}_s$ in $out(N_i)$

D.6 SOM-Based Data Mining

$rule\_set$	Rule set built by DM algorithm
$umat$	U-matrix of a trained SOM
$plane_l$	Component plane representing A_l
$rule\_set$	Set of rules
$umat\_bound_u$	Candidate boundary u on a U-matrix
$rule_{u1}$	Rule predicting first area separated by $umat\_bound_u$
$rule_{u2}$	Rule predicting second area separated by $umat\_bound_u$
$plane\_bound_{l\hat{m}}$	Candidate boundary $\hat{m}$ on $plane_l$
$plane\_mean_{l\hat{m}}$	Mean value along $plane\_bound_{l\hat{m}}$
$cand_{yx}$	Set of two candidate neurons that are neighbors of n_{yx}
$cand'_{yx}$	Set of neurons that are neighbors of n_{yx} and are not in $cand_{yx}$
$mean(cand_{yx})$	Mean distances between $\vec{w}_{yx}$ and weight vectors in $cand_{yx}$
$mean(cand'_{yx})$	Mean distances between $\vec{w}_{yx}$ and weight vectors in $cand'_{yx}$

$BDV(cand_{yx})$	Boundary difference value for $cand_{yx}$
$range(cand'_{yx})$	Range of distances between $\vec{w}_{yx}$ and weight vectors in $cand'_{yx}$
$plane\_attribute_l$	Attribute name for $plane_l$
θ_{char}	Characterizing threshold for SIG*
mat_char	Characterizing significance matrix for SIG*
ψ_{char}	Low characterizing condition parameter for SIG*
ϕ_{char}	High characterizing condition parameter for SIG*
mat_char_{li}	Characterizing significance matrix cell at row l and column i
$sig(A_l, N_i)$	Significance of A_l in N_i
$total\_char_i$	Total of column i in mat_char
$rule_i$	Rule corresponding to S_i
$select\_char_i$	Set of selected attributes for column i of mat_char
sum_char	Accumulator for column i of mat_char
$l\_char(A_l, S_i)$	Lower condition bound on A_l in SIG* characterizing rule for S_i
$h\_char(A_l, S_i)$	Upper condition bound on A_l in SIG* characterizing rule for S_i
$mean(A_l, N_i)$	Mean value of A_l over N_i
$dev(A_l, N_i)$	Standard deviation of A_l over N_i
θ_{diff}	Differentiating threshold for SIG*
mat_diff	Differentiating significance matrix for SIG*
ψ_{diff}	Low differentiating condition parameter for SIG*
ϕ_{diff}	High differentiating condition parameter for SIG*
mat_diff_{l1}	Differentiating significance matrix cell in row l
$sig(A_l, N_i, N_j)$	Significance of A_l in telling N_i and N_j apart
$total\_diff$	Column total of mat_char
$cond\_diff_i$	Differentiating condition expression i
$select\_diff$	Set of selected attributes for mat_diff
sum_diff	Accumulator for column of mat_diff
$l\_diff(A_l, S_i)$	Lower condition bound on A_l in SIG* differentiating rule for S_i

$h\_diff(A_l, S_i)$	Upper condition bound on A_l in SIG* differentiating rule for S_i
input	Intermediate data set built by HybridSOM
record	Record in *input*
output	Rule set produced by HybridSOM

D.7 Experimental Results

k	Number of folds in a k-fold cross validation (cf. Section D.4)
$\mathcal{E}_T$	Mean of overall training error
$\mathcal{S}_T$	Standard deviation of overall training error
$\mathcal{E}_{TM}$	Mean of training error due to misclassified $\vec{z}_s$
$\mathcal{S}_{TM}$	Standard deviation of training error due to misclassified $\vec{z}_s$
$\mathcal{E}_{TU}$	Mean of training error due to unclassified $\vec{z}_s$
$\mathcal{S}_{TU}$	Standard deviation of training error due to unclassified $\vec{z}_s$
$\mathcal{E}_G$	Mean of overall test error
$\mathcal{S}_G$	Standard deviation of overall test error
$\mathcal{E}_{GM}$	Mean of test error due to misclassified $\vec{z}_s$
$\mathcal{S}_{GM}$	Standard deviation of test error due to misclassified $\vec{z}_s$
$\mathcal{E}_{GU}$	Mean of test set error due to unclassified $\vec{z}_s$
$\mathcal{S}_{GU}$	Standard deviation of test set error due to unclassified $\vec{z}_s$
$\mathcal{E}_U$	Mean of unlabeled neuron percentage
$\mathcal{S}_U$	Standard deviation of unlabeled neuron percentage
$\mathcal{E}_{FT}$	Mean of total number of conditions per rule set
$\mathcal{S}_{FT}$	Standard deviation of total number of conditions per rule set
$\mathcal{E}_R$	Mean of number of rules per rule set
$\mathcal{S}_R$	Standard deviation of number of rules per rule set
$\mathcal{E}_{FA}$	Mean of average number of conditions per rule
$\mathcal{S}_{FA}$	Standard deviation of average number of conditions per rule
cn2-err	CN2 parameter for error estimate (see Appendix A)

cn2-star	CN2 parameter for size of *star* set (see Appendix A)
cn2-sig	CN2 parameter for significance threshold (see Appendix A)
c4.5-min	C4.5 parameter for $\vec{z}_s$ needed in two outcomes (see Appendix B)
c4.5-heur	C4.5 parameter for test heuristic (see Appendix B)
c4.5-pess	C4.5 parameter for pessimistic error confidence (see Appendix B)
c4.5-Fisher	C4.5 parameter for Fisher's exact test threshold (see Appendix B)
c4.5-redun	C4.5 parameter for rule pruning redundancy (see Appendix B)

D.8 The CN2 Algorithm

$comp_{\bar{a}}$	Complex $\bar{a}$ in a set of complexes
$acc(comp_{\bar{a}}, comp_{\bar{b}})$	Heuristic test comparing accuracy of $comp_{\bar{a}}$ and $comp_{\bar{b}}$
$acc(comp_{\bar{a}})$	Heuristic representing accuracy of $comp_{\bar{a}}$
$cover(C_m, comp_{\bar{a}})$	Number of examples covered by $comp_{\bar{a}}$ and belonging to C_m
$cover(comp_{\bar{a}})$	Total examples covered by $comp_{\bar{a}}$
$\hat{c}(\mathcal{D}_T)$	Number of classes present in $\mathcal{D}_T$
$significant(comp_{\bar{a}})$	Heuristic test for significance of $comp_{\bar{a}}$
$sig\_heur(comp_{\bar{a}})$	Heuristic representing significance of $comp_{\bar{a}}$
$actual_m$	Actual frequency for $\vec{z}_s$ belonging to C_m and covered by $comp_{\bar{a}}$
$expected_m$	Expected frequency for $\vec{z}_s$ belonging to C_m and covered by $comp_{\bar{a}}$
$beats\_default(comp_{\bar{a}})$	Heuristic test comparing accuracy of $comp_{\bar{a}}$ and a default rule
default	Empty complex covering all $\vec{z}_s$ in $\mathcal{D}_T$
star	Set of complexes searched by CN2
best_comp	Best complex found by CN2
$\mathcal{D}''_T$	Set of $\vec{z}_s$ in $\mathcal{D}_T$ covered by *best_comp*
tests	Set of all valid attribute tests
$test_{\bar{e}}$	Test $\bar{e}$ in *tests*
new_star	Set of specialized complexes
$comp'_{\bar{a}}$	Specialized complex built from $comp_{\bar{a}}$

D.9 The C4.5 Algorithm

$\mathcal{T}$	Decision tree built by C4.5
t'	Attribute test selected during recursive tree building
$\mathcal{D}'_T$	Training example subset created during recursive tree building
$\hat{n}$	Number of outcomes in t'
$\mathcal{O}$	Set of outcomes of t'
$O_{\bar{m}}$	Outcome $\bar{m}$ in $\mathcal{O}$
$\mathcal{D}'_{T\bar{m}}$	New subset $\bar{m}$ of $\mathcal{D}'_T$, generated when t' is applied to $\mathcal{D}'_T$
$heur(t')$	Heuristic measuring optimality of t'
$gain(t')$	Information gain caused by t'
$penalty(t')$	Penalty term for continuous attribute test t'
$values(t')$	Number of distinct values in $\mathcal{D}'_T$ for attribute evaluated by t'
$\lvert\mathcal{D}'_T\rvert$	Number of $\vec{z}_s$ in $\mathcal{D}'_T$
$info(\mathcal{D}'_T)$	Entropy over $\mathcal{D}'_T$
$info_{t'}(\mathcal{D}'_T)$	Expected entropy over $\mathcal{D}'_T$ after split by t'
$\hat{c}(\mathcal{D}'_T)$	Number of C_m in $\mathcal{D}'_T$
$freq(C_m, \mathcal{D}'_T)$	Number of $\vec{z}_s$ belonging to C_m in $\mathcal{D}'_T$
$\lvert\mathcal{D}'_{T\bar{m}}\rvert$	Number of $\vec{z}_s$ in $\mathcal{D}'_{T\bar{m}}$
$info(\mathcal{D}'_{T\bar{m}})$	Entropy over $\mathcal{D}'_{T\bar{m}}$
$gain\_ratio(t')$	Gain ratio caused by t'
$ave\_gain(\mathcal{D}'_T)$	Average $gain(t')$ over all valid tests that can be evaluated on $\mathcal{D}'_T$
$split\_info(t')$	Information potentially created by t' splitting $\mathcal{D}'_T$ into $\hat{n}$ subsets
$\mathcal{T}'$	Subtree of $\mathcal{T}$ refined during recursive tree building
$candidates$	Set of every valid t' during recursive tree building
$\mathcal{T}'_{\bar{m}}$	Subtree $\bar{m}$ of $\mathcal{T}'$ generated by t'
$rule$	New rule generated for $rule\_set$
$pess\_err(rule)$	Pessimistic error rate of $rule$
$cond$	Condition in the antecedent of $rule$

$prunable(cond)$	Heuristic test for prunability of *cond* from *rule*
$rule'$	Pruned version of *rule*, with *cond* removed
$exact\_sig(rule')$	Value of Fisher's exact significance test for $rule'$
$accurate(rule)$	Heuristic test for whether *rule* is too inaccurate for *rule_set*
$default\_err(rule)$	Default error against which $pess\_err(rule)$ is evaluated
$class(\mathcal{D}_T, rule)$	Number of $\vec{z}_s$ in $\mathcal{D}_T$ with same C_m as *rule* predicts
$\lvert\mathcal{D}_T\rvert$	Number of $\vec{z}_s$ in $\mathcal{D}_T$
leaves	Set of all leaf nodes in $\mathcal{T}$
$\hat{l}$	Number of leaf nodes in $\mathcal{T}$
$leaf_{\bar{n}}$	Leaf node $\bar{n}$ in *leaves*
$\mathcal{R}$	Route traced from root of $\mathcal{T}$ to $leaf_{\bar{n}}$
best_cond	Best *cond* to remove from *rule*
$group(C_m)$	Rule group containing every *rule* in *rule_set* predicting C_m
prob	Probability of *rule* inclusion in $group(C_m)$ as member of *search_set*
search_set	Subset of rules in $group(C_m)$ searched for lower MDL encoding

Appendix E

Acronym Definitions

This appendix alphabetically lists the acronyms used throughout this dissertation. Both existing acronyms and new acronyms introduced by this research are listed.

AI	Artificial intelligence
ANN	Artificial neural network
BDV	Boundary difference value
BME	Best matching example
BMU	Best matching unit
CI	Computational intelligence
CNF	Conjunctive normal form
DBMS	Database management system
DM	Data mining
DMQL	Data Mining Query Language
DNF	Disjunctive normal form
EDA	Exploratory data analysis

K-S	Kolmogorov-Smirnov
KDD	Knowledge discovery in databases
LVQ	Learning vector quantizer
MDL	Minimum description length
OLAP	Online analytical processing
PSO	Particle swarm optimization
SOM	Self-organizing feature map
SQL	Structured Query Language
SVM	Support Vector Machine
VQ-P	Vector quantization-projection

Index