Comparison of LEM2 and a Dynamic Reduct Classification Algorithm

Masters thesis

by

Ola Leifler

Reg nr: LiTH-IDA-EX-02-93

9th October 2002
Comparison of LEM2 and a Dynamic Reduct Classification Algorithm

Master’s thesis

performed in Artificial Intelligence & Integrated Computer Systems Division, Dept. of Computer and Information Science at Linköpings universitet

by Ola Leifler

Reg nr: LiTH-IDA-EX-02-93

Supervisor: Dr. Marcin Szczuka
Faculty of Mathematics, Informatics and Mechanics
at Warsaw University

Examiner: Professor Patrick Doherty
Department of Computer and Information Science
at Linköpings universitet

Linköping, 9th October 2002
Abstract

This thesis presents the results of the implementation and evaluation of two machine learning algorithms [Baz98, GB97] based on notions from Rough Set theory [Paw82]. Both algorithms were implemented and tested using the Weka [WF00] software framework. The main purpose for doing this was to investigate whether the experimental results obtained in [Baz98] could be reproduced, by implementing both algorithms in a framework that provided common functionalities needed by both. As a result of this thesis, a Rough Set framework accompanying the Weka system was designed and implemented, as well as three methods for discretization and three classification methods.

The results of the evaluation did not match those obtained by the original authors. On two standard benchmarking datasets also used previously in [Baz98] (Breast Cancer and Lymphography), significant results indicating that one of the algorithms performed better than the other could not be established, using the Students t-test and a confidence limit of 95%. However, on two other datasets (Balance Scale and Zoo) differences could be established with more than 95% significance. The “Dynamic Reduct Approach” scored better on the Balance Scale dataset whilst the “LEM2 Approach” scored better on the Zoo dataset.

Keywords: Machine Learning, Rough Sets, LEM2, Dynamic Reducts
Acknowledgements

First of all, I’d like to thank my supervisor Marcin Szczuka who tirelessly provided me with help about both theoretical and practical matters. Patrick Doherty, my examiner, gave valuable comments on the report. Marcus Bjäreland helped me get the first pieces of the report together. Björn Tikkanen, my opponent, and Tobias Nurmiranta gave useful insights about how to write a report that someone can actually have a fair chance of understanding. As I progressed in my work, many others were harassed with questions about everything from Java, \LaTeX{} and Rough Set issues and I would like to take this opportunity to thank you all for your patience.
## Contents

1 Introduction 

1.1 Task ........................................ 2  
1.2 Rationale .................................... 2  
1.3 Intended audience ............................. 2  
1.4 Structure of this work ........................ 3  
\hspace{1em} 1.4.1 Using the Weka system ......... 3  
\hspace{1em} 1.4.2 Embrace and Extend ............ 3  
1.5 Structure of this thesis ...................... 4  
1.6 About the report ............................ 5  

2 Machine Learning .............................. 7  

2.1 Learning ..................................... 8  
2.2 Classification ............................... 8  
2.3 Rules ........................................ 10  
2.4 Restrictions ................................. 11  
2.5 Common issues ............................... 12  
2.6 Taxonomy .................................... 13  
\hspace{1em} 2.6.1 Why decision rules? ............... 14  
\hspace{1em} 2.6.2 Why Rough Sets? ................. 14  
2.7 Weka ........................................ 15  
\hspace{1em} 2.7.1 Manipulating input ................. 15  
\hspace{1em} 2.7.2 Representing knowledge .......... 16  
\hspace{1em} 2.7.3 Classification algorithms ......... 16  
\hspace{1em} 2.7.4 Evaluation methods ............... 18
3 Rough Sets
3.1 Background ......................................... 22
3.2 Notation .............................................. 23
3.3 Rough Set basics .................................... 23
  3.3.1 Reds .............................................. 24
        Use of reducts .................................. 25
  3.3.2 Relative reducts ................................. 25
  3.3.3 Approximations ................................. 26
        Use of approximations .......................... 28
  3.3.4 Decision rules ................................. 28
        Use of decision rules .......................... 28
3.4 Dynamic Reducts .................................... 28
  3.4.1 Background ..................................... 29
  3.4.2 Dynamic Re duct theory ......................... 30
        Use of dynamic reducts ........................ 32
3.5 LEM2 specifics ..................................... 32

4 Algorithms ........................................... 35
4.1 Notation ............................................ 36
4.2 The approaches .................................... 36
4.3 Rough Set Praxis .................................... 39
  4.3.1 Re duct Computation ........................... 40
  4.3.2 Dynamic Re duct Computation .................... 41
        Stability calculation .......................... 41
4.4 Filtering ............................................ 42
4.5 Missing values ..................................... 43
        Computational complexity ....................... 44
4.6 Discretization ...................................... 44
  4.6.1 Discernibility .................................. 44
  4.6.2 Local Discretization ............................ 46
  4.6.3 Global Discretization ......................... 47
        Computational complexity ....................... 47
  4.6.4 Dynamic Discretization ......................... 50
        Computational complexity ....................... 51
4.7 Model construction .................................. 53
  4.7.1 LEM2 ............................................ 53
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational complexity</td>
<td>55</td>
</tr>
<tr>
<td>4.7.2 Rule approximation</td>
<td>56</td>
</tr>
<tr>
<td>Computational complexity</td>
<td>57</td>
</tr>
<tr>
<td>4.8 Classification</td>
<td>58</td>
</tr>
<tr>
<td>4.8.1 The Voter</td>
<td>58</td>
</tr>
<tr>
<td>Computational complexity</td>
<td>59</td>
</tr>
<tr>
<td>4.8.2 Rule negotiations</td>
<td>60</td>
</tr>
<tr>
<td>4.8.3 Basic negotiation methods</td>
<td>60</td>
</tr>
<tr>
<td>Computational complexity</td>
<td>60</td>
</tr>
<tr>
<td>4.8.4 Stability strength</td>
<td>62</td>
</tr>
<tr>
<td>Computational complexity</td>
<td>63</td>
</tr>
<tr>
<td>4.9 Remarks on computational complexity</td>
<td>64</td>
</tr>
<tr>
<td>4.9.1 LEM2 time complexity</td>
<td>64</td>
</tr>
<tr>
<td>4.9.2 DR time complexity</td>
<td>64</td>
</tr>
<tr>
<td>4.9.3 Computational feasibility</td>
<td>64</td>
</tr>
<tr>
<td>4.10 Further Reading</td>
<td>65</td>
</tr>
</tbody>
</table>

## 5 Implementation

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Implementation philosophy</td>
<td>67</td>
</tr>
<tr>
<td>5.2 Fast vectors</td>
<td>68</td>
</tr>
<tr>
<td>5.3 Core datastructures</td>
<td>69</td>
</tr>
<tr>
<td>5.4 Decision Rules</td>
<td>70</td>
</tr>
<tr>
<td>5.5 Cutpoints</td>
<td>71</td>
</tr>
<tr>
<td>5.6 Rough Set functions</td>
<td>72</td>
</tr>
<tr>
<td>5.7 Reduct computation</td>
<td>73</td>
</tr>
<tr>
<td>5.8 Model construction</td>
<td>73</td>
</tr>
<tr>
<td>5.9 Filters</td>
<td>73</td>
</tr>
</tbody>
</table>

## 6 Testing

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Comparing learning algorithms</td>
<td>75</td>
</tr>
<tr>
<td>6.1.1 Significance</td>
<td>76</td>
</tr>
<tr>
<td>6.2 The datasets</td>
<td>76</td>
</tr>
<tr>
<td>6.2.1 Dataset origins</td>
<td>78</td>
</tr>
<tr>
<td>6.3 Experiment setup</td>
<td>78</td>
</tr>
<tr>
<td>6.3.1 System setup</td>
<td>79</td>
</tr>
<tr>
<td>6.4 Evaluation</td>
<td>79</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This masters thesis was written as the final project of the Computer Science Programme for the fulfillment of a *Datavetenskaplig magisterexamen*\(^1\) at Linköpings universitet, Sweden. All work presented in this paper was performed at the Computer Science Department (IDA) as part of the WITAS UAV [DGK+00] project\(^2\). It was supervised by Marcin Szczuka of Warsaw University.

In this introduction the purpose of this work will be stated, along with the motivation for doing it and how the actual work has been carried out. Since this chapter will set the scene for the following parts, notions not properly defined until later will be briefly referred. It is hoped that the reader is to a certain degree already familiar with some of the concepts and that the rest will be fairly easy to find information about in the following chapters.

\(^1\)approximately a Master of Science in Computer Science

\(^2\)However, there are no immediate applications in WITAS of the results presented here. This is to be considered as basic research in the area of machine learning, an area which is of general interest for the WITAS project
1.1 Task

The purpose of this work was to investigate the performance of two machine learning algorithms, here called LEM2 (Learning from Examples, Module 2) and DR (Dynamic Reducts). They were to be compared to each other primarily, but also to other learning algorithms. The parameters considered most important for such a comparison were quality of the generated rule sets, but also the computational feasibility of the methods. Two articles were used as the main source of reference, namely [GB97, Baz98], where the algorithms are described and the results used for comparison are presented.

1.2 Rationale

Why should it be considered interesting to do this kind of a comparison? First of all, both algorithms that were investigated in this work operate on similar datasets and use common theoretical underpinnings. Therefore, it could be interesting to do a fair and thorough investigation of them, including implementing them from scratch, to compare their behaviour in similar environments and excluding possible differences in performance that may arise from the use of different internal datastructures or different evaluation procedures. This way, the core algorithms and nothing more will be compared while unimportant parts are kept invariant between them.

Our hypothesis is that the tendencies\textsuperscript{3} obtained previously should be reproduced here, even if the exact numbers in the experiments might differ from previous experiments. This is an empirical study of two algorithms, so the focus of this work will not be on theoretical comparisons, though the computational complexity of both algorithms will be analyzed.

1.3 Intended audience

This thesis is intended for people literate in basic computer science and mathematical notations. It is not necessary to be a computer science major,\footnote{\textsuperscript{3}such as the relation between the experimental results from running each of the learning algorithms}
but being able to follow algorithmic descriptions, understand concepts from Object Oriented Programming and interpret UML diagrams will help. No prior knowledge of either Rough Sets or machine learning is required to understand the contents.

1.4 Structure of this work

Originally, no more constraints were made about this project than that it should involve a comparison of two algorithms. Since both algorithms use Rough Set techniques, it was considered important to properly implement a common set of Rough Set functionalities that could be used by both algorithms. Thus, a common framework was devised for them. However, the common needs of these two machine learning algorithms were not limited to a set of Rough Set functionalities, but also included methods for accessing and storing datasets, doing evaluation of the performance of the algorithms and preferably a nice graphical interface to them. All these needs led to the usage and later extensions of the Weka machine learning framework.

1.4.1 Using the Weka system

The Weka machine learning framework [WF00] was designed and implemented at Waikato University, New Zealand. Weka is short for Waikato Environment for Knowledge Analysis, but is also the name of a flightless bird found in New Zealand. The software contains much of the common functionality needed to support machine learning algorithms. There are methods for filtering the datasets, applying a learning algorithm to a set of data and evaluating the results of running the algorithm. Therefore, Weka was chosen as a basis for implementing the algorithms at hand, since it was thought that much overhead could be spared if some existing system could be used with slight adaptations.

1.4.2 Embrace and Extend

Though Weka contains lots of functionalities that common learning algorithms need, it was originally written in the early days of Java, when op-
timization was crucial if you wanted fast versions of algorithms. This lead to customizations and specialized datatypes that were not in the standard set of classes. By not using common classes and interfaces, it is difficult to currently use modern Java functionalities like generic data manipulation algorithms. This, along with a low level of abstraction in the implementation of Weka, made it imperative to extend the framework in order to have a common platform that the algorithms could be built upon.

1.5 Structure of this thesis

The structure of this thesis will basically follow that of the whole thesis’ project, as follows:

**Machine Learning** Introduction of the scientific discipline of machine learning and also a description of the learning situations this work is focused on (chapter 2).

**Rough Sets** Basics about the theory of Rough Sets and the somewhat more advanced concepts used in the algorithms that are to be compared (chapter 3).

**Algorithms** All the parts of the algorithms that have been investigated and are of interest to us will here be explained in more detail, along with notes on their implementation in Weka (chapter 4).

**Implementation** Description of the actual implementation of the algorithms and the supporting functionalities (chapter 5).

**Testing** The actual comparison that has been carried out will be presented, according to this outline (chapter 6):

1. Methods of comparison.
2. Datasets used.
3. Reliability of the results.

**Discussion** Finally, the results will be discussed and a summary of the work will be given in chapter 7.

\[e.g.\] those in the `java.util.Collections`-class
1.6 About the report

This report was typeset in \LaTeX \[\text{Lam94}\] along with the Pseudocode \[\text{KS99}\] package for displaying the algorithms. It is available on the World Wide Web at \url{http://www.ida.liu.se/~olale/exjobb/thesis/thesis.pdf} (pdf-format).
1.6. About the report
Chapter 2

Machine Learning

The purpose of this work is to investigate algorithms that learn by experience. The scientific field studying such algorithms is called Machine Learning (ML).

Exactly what it means for an algorithm to be learning something may not be obvious, but the definition given in [Mit97] is fairly good:

**Definition 1 (Learning)** A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

It is only “fairly good” since definition 1 does not allow for the learning program to **construct** programs that are in turn evaluated by the measure P. In general, the program constructs some kind of model of the input data, much as we do as humans. This model however, can be more or less dependent on the program that it was constructed from, to work. Here, much emphasis will be on models that are completely independent of the programs from which they are derived.

Very generally speaking a program should improve its performance. The experience could be just about anything, but in this work we will be interested in what programs can do when they train by example. More specifically, they will be presented with a number of examples that illustrate...
something and then the algorithm is supposed to derive a set of rules that in a more compact way describes what it has been presented with. The purpose of those rules is to perform predictions about unknown examples and classify them as belonging to a certain “decision class”. This could for instance mean that an algorithm is given a set of data sampled from a population of medical patients already diagnosed as either having an ailment or not. The task is then to diagnose new patients, given a similar set of data. It could be thought of as a form of pattern recognition – to see what combinations of medical values or symptoms pertain to a certain disease.

2.1 Learning

In applying the scientific method, some criteria must be established for what may be considered a well-posed learning problem. There are basically three demands that are placed on such a learning problem.

1. The resulting program must have a restricted set of tasks to perform.

2. There must be a well-defined performance measure that can be used on the resulting program.

3. Experience must be gained from some well-defined source.

For example, there might be the situation where the resulting algorithm is supposed to be a world chess champion. Then, the task to perform is playing chess, the performance measure is its ability to win matches and the experience could come from playing games against itself (example from [Mit97], page 3).

To construct a general learning problem, we have a set of choices as described in table 2.1. We must define what it is we are interested in at each stage.

2.2 Classification

A schematic view of a learning process is given in figure 2.1. The steps in the figure are not always necessarily part of what is generally called a
First comes the extraction of a sample dataset (hopefully a representative sample of the whole set) from the universe, a sample that is used to build a model (also called a training dataset). Usually, there are some kind of constraints on the data during the model construction phase, depending on how the construction is performed. To satisfy these constraints, the training dataset is modified so that the requirements are fulfilled while hopefully retaining all characteristics of the original set. In this thesis,
filtering methods that use ideas from the Rough Set theory will be investigated.

After filtering our data a model is built, which can be said to be the internal representation of the patterns inherent in the dataset. The reason for constructing this internal model by computer is that the universe, from which examples are later drawn, is so big that we simply cannot create a good enough model ourselves, or at least not in reasonable time. This model is then used later to tell things about data, other than the training set, drawn from the universe. Let’s use the example from the beginning of this chapter, the one with the medical patients. The expectations are that the program should tell us whether a patient has a certain disease or not. When asking such questions to the program, we cannot always expect to receive correct answers, since we only presented the learning process with a subset of the entire set of possible symptoms that patients may display. Thus, errors arise. Commonly, the ratio between correct and incorrect decisions made by the program is used as one of the measures of the quality of the produced model, a measure that is of course vital to ascertain before we can actually use our algorithm in reality to do things that people depend on. Naturally, there are always practical limitations on how good a model built from incomplete data can be so we cannot hope to find an oracle-like algorithm that produces perfect models, but depending on the difficulty of the problem domain, we may come pretty close.

2.3 Rules

Machine learning is about generating hypotheses/models that can have predictive or explanatory ability in a certain situation. It is desired that the algorithms create a model that can be used either to explain some scenario or to predict something. In the general case we would have to make decisions about how to obtain data to train our program with, what hypothesis or function our program should generate and what measurements to use for evaluating our hypothesis. Each of these decisions has already been made in this thesis.

\footnote{which is called classification, here w.r.t. whether a patient has a disease or not}
The algorithms will be trained on static sets of data that represent a set of observations, or objects. These sets are used to create a function that assigns a decision value to each object, for instance whether an observed person is a man or woman, what brand an observed car has or how warm it is outdoors, given some information about the object. This function will be represented as a set of propositional logical formulae of the form

\[(\text{forecast} = \text{cloudy}) \land \ldots \land (\text{sunny} = \text{yes}) \Rightarrow (\text{playtennis} = \text{yes})\]

where the available information (the antecedent) is on the left-hand side of the implication, and the consequence is on the right-hand side.

Since access is given to datasets that are provided by others ([WF00]), the algorithms can use these datasets to generate a rule-based predictor that assigns a decision value to each new instance that is not in the training dataset\(^2\). Thus, both the learning material, the representation of the hypothesis function and the evaluation model are predetermined.

### 2.4 Restrictions

The representation of the hypothesis or model alone may not imply that our learning situation is a restriction to the general learning scenario, but both the requirements on the data used for training and the evaluation model may introduce restrictions on the applicability of the algorithms and results of this thesis.

- One notable issue about the datasets in the thesis is that they are assumed to be relevantly constructed and crafted or engineered by humans. Irrelevant data will confuse the algorithms but they will still generate hypotheses, albeit of low quality. This is a common requirement if the training set is provided by sources external to the program.

\(^2\)The number of correctly classified instances among the new instances is commonly used as a measure of the predictive quality of the algorithm
Let’s say that each object in our set of training data is a vector \( \{x_1, \ldots, x_m\} \), corresponding to medical data about a patient. In our approach we are not able to model correlations or dependencies between any \( x_i, x_j \) such as \( x_i < x_j \) or \( x_i + 2 = 4x_j \).

The dataset is assumed to have examples of all possible decisions that can be made. If there are only examples that belong to one of two possible decision classes, other approaches must be used (see [Mug96]) since our algorithms will in such cases generate hypotheses that classify all objects into one decision class only.

However, general issues about uncertainty are addressed.

\subsection*{2.5 Common issues}

Commonly when we feed data from real situations into a computer and intend to use the data as a basis for generating hypotheses, we introduce uncertain data. The reason is that we simply don’t have complete information and want to be able to reason with whatever is available.

The most common types of uncertainties in data are:

\textbf{Missing values} Some values may be missing in the observations we have made. Returning to our medical example, some instruments used for testing the patients may have been broken at some points in time, resulting in incomplete information about those patients.

\textbf{Ambiguous decisions} Two observations may be equal except that the decision values assigned to them differ. For example, two patients with the same symptoms may be diagnosed differently by two different doctors. This may also be called \textit{ambiguity}.

The types of problems that can be modeled are confined to such problems that can be encoded as independent observations with a limited set of attributes in each observation. The types of patterns that can be found are those that associate a fixed combination of attribute values to a certain decision.
Also, when constructing rules from a set of data, a common problem is that one tends to overfit the data, which means that if a small training set is given, the learning algorithms tend to have difficulties generalizing the knowledge about the world to encompass unseen examples. Rather, they induce patterns that are only valid for this small training set and are not supported by instances not in the set. This is a fundamental problem which is nearly impossible to solve since a good generalization would require knowledge about how one may generalize whilst retaining a sound hypothesis, something that in turn requires knowledge derived from more objects than those in the training set. Reductio ad absurdum.

2.6 Taxonomy

Dividing ML algorithms into classes may be done in several ways, roughly corresponding to the range of possible options one has when constructing a learning problem, as described in table 2.1. The reason for trying to introduce a taxonomy here is that it may provide the reader with a broader perspective of what constitutes a ML process and a better understanding of the limitations inherent in the approaches used here.

When classifying objects, ML algorithms of today use for example the concepts of decision trees, neural networks or rule sets (see [Mit97]) as the internal representation for the knowledge they accumulate. The formalism used to describe the knowledge may be within the Fuzzy Set or Rough Set theories. The algorithms for classification may be purely probabilistic, as in Bayesian learning [CPS98], or use one or more of the concepts mentioned above. Pre- and postprocessing, like preparation of the data sets and validation of the results, may or may not be part of each specific ML algorithm. All in all, there are no clear distinctions between different algorithms, or even what a ML algorithm is and is not. Does preprocessing and discretization of attribute values (see [FI93]) count as part of an algorithm for learning or not? In this thesis I will be satisfied with having answered the questions of table 2.1 according to table 2.2. Thus we specialize ourselves to only deal with a rather simple, but well-defined version of a learning problem.
2.6. Taxonomy

<table>
<thead>
<tr>
<th>Issue</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of experience</td>
<td>Use tables of predefined observations</td>
</tr>
<tr>
<td>Target function</td>
<td>Construct a function that assigns decision values to our observations</td>
</tr>
<tr>
<td>Representation of the learned function</td>
<td>Construct a set of propositional logic formulae</td>
</tr>
<tr>
<td>Learning algorithm</td>
<td>Our two approaches to compare</td>
</tr>
</tbody>
</table>

Table 2.2: Learning problem in this thesis

2.6.1 Why decision rules?

Rule sets, i.e. sets of formulas in propositional logics of the form

\[(a_1 = v_1) \land \ldots \land (a_n = v_n) \Rightarrow d\]

are argued to be one of the more promising representations of knowledge since:

- It is easy to understand what the rules say and to figure out whether they make sense or not. Each of the pairs \(a_i = v_i\) have a meaning - they directly relate to real-world concepts.

- Since the rules can be understood by humans, they can also be manipulated by humans, which in effect means that we can incorporate knowledge we already possess into the knowledge base in an efficient manner.

- Unlike decision trees, no duplication of rules take place. In decision trees one might have several subtrees that are exact copies of each other.

2.6.2 Why Rough Sets?

One of the principal motives for originally developing the rough set concepts was the elegance with which one could treat uncertain knowledge. In one of the original articles on the subject ([Paw82]) it is argued that common
problems in AI (particularly learning scenarios) can be formulated elegantly using rough set concepts. The expressive power is limited to only deal with certain or uncertain, not more or less uncertain knowledge, but it is still very useful. Also, the models that result from using a Rough Set technique are sets of rules, a representation of knowledge that offers great possibilities for inspection and manipulation by humans since the knowledge is in an explicit and readable form.

2.7 Weka

As a software framework for ML algorithms, Weka offers a wide range of possibilities. It has been designed both to be used as an introduction to ML, a tool for ML scientists who want proven implementations of standard algorithms that they can use and also for those who wish to extend it with a piece of ML software of their own. Therefore, it contains lots of functionalities that are commonly used, some of which will be explained here.

2.7.1 Manipulating input

As a common format for all the input data, WEKA uses a simple ASCII-format called the ARFF-format for the input. The attributes may be nominal\(^3\), numerical or strings. Unknown or missing attribute values are represented by a “?”.

The input may need to be filtered, which can be done either as a supervised or an unsupervised process. In supervised mode, the user may tell the filter which attribute will later be used as a decision attribute, which is not done in the unsupervised process. As an example of a filter, missing values in a dataset may be filled with the mean or average value of the attribute prior to classification.

\(^3\)the value range of the attribute is a finite, typically small, set of values where each one has a special meaning. For example, an attribute Occupation could have the value range \{thief, farmer and doctor\}
2.7.2 Representing knowledge

The internal representation of data can be either the same as for the input - as a so called decision table. Another method is to construct various kinds of trees that encode the knowledge in a treelike structure, or as some kind of rule set. However, none of these encodings except the decision table are made generally available as a structure that can be reused by other algorithms. A simple form of representation that might be useful for some algorithms is the instance-based representation, where the very data is used as is, though not the whole decision table necessarily.

2.7.3 Classification algorithms

When it comes to actually classifying data, WEKA offers several kinds of learning algorithms. There are decision tree learners such as a variant of C4.5 (see [Qui93]) called j48, that use a treelike structure for their internal representation where each node in the tree branches on some particular attribute describing the data. Second, there are so called support vector machines which classify data by taking the decision of the nearest object in the data, given by some distance function based on the concept of hyperplanes (see [WF00]). Moreover, there are methods for clustering data into chunks that share some features, performing numerical prediction and more.

What is of main interest in this report however, are the algorithms that construct decision rules (see section 3.3) to represent their model of the world. This is due to the fact that the results presented in chapter 6 will only be compared to those of other rule-based learning algorithms that are present in Weka. The reason for not choosing more algorithms to compare results with is that this thesis’ main conjecture states that the tendencies seen previously should be reproduced, so the only really interesting comparisons would be between the two approaches we will study.

As of version 3.2 of the Weka system, the following rule-producing methods are included: DecisionTable [Koh95], OneR [Hol93], Prism [Cen87] and ZeroR.

Let’s have a look at what they do:

**DecisionTable** A decision table in this context is exactly what we will
also call a decision table (see sections 2.7.2 and 3.3). However, the algorithm used in this class was constructed with the purpose of evaluating the representation of this model produced rather than evaluating a rule induction technique. The author of this algorithm argued that reducts used in Rough Set techniques were unimportant, or at least not necessary in order to extract a set of attributes used for rule induction. He uses another method for extracting a subset of the attributes later used for classification, which has not been studied in detail for this work.

**OneR** Constructs a very simple model where only one attribute is considered. It constructs a number of rules of the form $\text{forecast} = \text{cloudy} \Rightarrow \text{playtennis} = \text{yes}$. The attribute with the least error rate for these kinds of rules is chosen as the resulting condition attribute.

**Prism** Creates rules for each decision class by adding to a new rule the attribute-value pairs that maximizes the ratio between the number of objects supporting the rule and the number of objects matching the rule, until no matches with different decision value are given by the rule (see section 3.3). All objects matched by a completed rule are then removed and the process is repeated, until no more objects from the decision class remain. **Prism** is a predecessor to LEM2 which works reasonably well on some datasets but may still overfit the data by producing too specific rules since the stop requirement is quite restrictive (see [CGB94]).

**ZeroR** An algorithm which, in analogy to OneR, considers no attributes during classification but only assigns to all objects either the most frequent decision value if the decision attribute is nominal, or the mean decision value if the decision attribute is numeric. However, ZeroR serves a purpose in benchmarking since some algorithms in certain cases generate so specific rules that the predictive quality may even be worse than that of ZeroR, which is a strong indication that the algorithm needs modification.

All of these learning algorithms use their own separate internal representations of the rules they create. None of them share methods for manipulating rules or with each other. Thus, their internals were not considered
for reuse during the implementation of the LEM2 classification algorithm. However, since both the DecisionTable and Prism classes shared some ideas with LEM2, they served later as reference implementations during the testing stage of this project.

2.7.4 Evaluation methods

In ML, evaluation of an algorithm is mostly done by monitoring the predictive quality of the model produced by the algorithm. In a real-world situation, one might have a set of objects for which the decision values were already known, and another set for which there are no known decision values. Then, the problem would of course be to assign correct decisions to the unseen, unclassified objects. However, to obtain some information about how well an algorithm works, we would need to be able to measure the error rate. If we gave our program a set of data to classify, the classification of which was unknown even to humans, and got some resulting classification, then we could hardly tell how good or bad this classification would be. Thus, the most common way of evaluating a learning algorithm is through the use of cross-validation, where the training set, with decision values given, is divided into several parts. Then during evaluation, all but one are used to generate the model and the remaining one to evaluate the model. This process is repeated for every part of the original set. Since it is likely that the quality of the model improves with the size of the training set it is given, usually only a small fraction of the original set is left out of the learning process, making the number of such distinct subsets rather large if cross-validation is to be used. *N-fold cross-validation* and *leave-one-out* are by far the most common methods for evaluating learners. In N-fold cross-validation one N:th of the table is left out of the learning process and later used for testing, which is repeated N times. Leave-one-out does just what the name implies - leave one instance in the dataset out during training and use it for testing. The latter method is good on large datasets because there is only one way (when the ordering is not important) to extract N objects one at a time from a set of N objects so statistical methods can give more precise answers as to the quality of the process.

Weka offers both N-fold cross-validation, the use of separate training and testing sets as well, as something called percentage split, which is
simply a method where you split the training set into two parts, one used for training and one for testing, the sizes of which are determined by the percentage. All algorithms later tested used a ten-fold cross-validation procedure for evaluation.
Chapter 3

Rough Sets

This chapter will introduce all the basic concepts of Rough Sets that we will need in order to reason about the algorithms that were implemented as a part of this work. Most importantly, the ideas described in section 3.4.2 (dynamic reducts) will be used in algorithm 4.6.4 (a filtering technique). Apart from that, minimal use of the concepts of approximations as described in definitions 3 and 4 occurs in algorithm 4.7.2 (the LEM2 algorithm). All other Rough Set theory presented here serves as a general background to the reader and has been of importance for implementing the general Rough Set framework, though not all parts were used in the final testing setups. The mathematical definitions in this chapter are not necessary to understand in order to follow the algorithms in chapter 4. They are included for those who better understand mathematical set-notations than free text. However, all concepts are explained in plain text as well as by examples, so those who do not feel comfortable reading mathematical formulae should be able to understand them as well.

When we later refer to “Rough Sets” we will mean some of the concepts pertaining to the Rough Set theory as explained below.
3.1 Background

Rough Sets introduce a set theory, originally devised by Zdzisław Pawlak in 1982 [Paw82], that takes uncertainties into account. In normal set theory, membership in a set is “crisply” defined so that every object is either in a set or not. In many cases it is not sufficient to have a theory that only allows such positive or negative knowledge about the world. It is rather desired that one can reason about both inconsistencies in data as well as lack of data. Several proposals for theoretical frameworks dealing with uncertainties have been put forth, ranging from approaches only concerned about the reasoning itself such as fuzzy logic [CPS98] to methods for representing uncertain knowledge, like the Rough Set method. The rough set approach to the uncertainty problem is to introduce equivalence relations between objects in the universe and derive the concepts of sets from the definition of this relation.

Another approach to dealing with approximate sets called fuzzy sets stating that each object may be included in a set with the probability $P : 0 \leq P \leq 1$ has also been proposed, though some authors argue that the fuzzy set theory often collapses into a two-valued logic in practice [Elk94]. Having discrete rather than continuous properties has seemed like a nice idea since it allows for the use of traditional (discrete) logical reasoning frameworks to be used.

<table>
<thead>
<tr>
<th></th>
<th>Occupation</th>
<th>Age</th>
<th>Shoesize</th>
<th>Credibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>thief</td>
<td>35</td>
<td>42</td>
<td>High</td>
</tr>
<tr>
<td>$u_2$</td>
<td>doctor</td>
<td>45</td>
<td>44</td>
<td>Medium</td>
</tr>
<tr>
<td>$u_3$</td>
<td>thief</td>
<td>35</td>
<td>41</td>
<td>Low</td>
</tr>
<tr>
<td>$u_4$</td>
<td>farmer</td>
<td>23</td>
<td>46</td>
<td>High</td>
</tr>
<tr>
<td>$u_5$</td>
<td>thief</td>
<td>53</td>
<td>46</td>
<td>High</td>
</tr>
<tr>
<td>$u_6$</td>
<td>thief</td>
<td>53</td>
<td>45</td>
<td>Low</td>
</tr>
<tr>
<td>$u_7$</td>
<td>thief</td>
<td>37</td>
<td>42</td>
<td>Medium</td>
</tr>
<tr>
<td>$u_8$</td>
<td>doctor</td>
<td>49</td>
<td>44</td>
<td>Low</td>
</tr>
<tr>
<td>$u_9$</td>
<td>doctor</td>
<td>49</td>
<td>44</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 3.1: Example of a decision table
In the definitions below we will use table 3.1 as an example to illustrate the concepts presented. The table is a representation of a decision system (see section 3.3) and we will sometimes refer to it by calling it a decision table. It is supposed to illustrate a police officer’s problem of finding out what witnesses are the most credible ones when making an investigation. From previous knowledge, the officer knows that the subjects $u_1, u_2, \ldots, u_9$ can be said to have credibility according to table 3.1. Also, some data has been collected regarding their occupation, age and shoesize. Now he wants to use this information when deciding whether to believe witnesses or not in the future and preferably, he would like computer support to help him.

### 3.2 Notation

Standard mathematical notation will be used throughout this section along with one convention - all lowercase letters are related to their uppercase counterparts, usually by being elements in the sets that are denoted with an uppercase letter. Also, uppercase letters will also be related to their boldcase counterparts, such as the letters $T$ and $\mathbf{T}$, in one way or another.

### 3.3 Rough Set basics

This section will introduce a description of Rough Set theory, or at least the parts we will be interested in. It may be used for reference later.

An information system is a pair $S = \langle U, A \rangle$ where $U$ is a non-empty finite set of objects called the universe and $A$ is a non-empty finite set of attributes such that $a : U \mapsto V_a$ for every $a \in A$. The set $V_a$ is called the value domain of $a$.

The concepts related to information systems can be illustrated using a simple table, and table 3.1 will be used throughout this chapter to explain the concepts.

**Example 1** The attributes of the information system depicted in table 3.1 are:

$$A = \{ \text{Occupation, Age, Shoesize, Credibility} \}$$
Though the attribute *Credibility* may be seen as the attribute that depends on the other two, it is just an attribute.

A decision system is an information system where one of the attributes is a decision attribute and the rest are then called condition attributes. The decision attribute is the one we want our program to collect information about, so that we can ask it about what value should be assigned to an object with no value on that attribute. Decision systems will here be noted as \( \langle U, A \cup \{d\} \rangle \) where \( d \) is the decision attribute. We will by using \( S \) refer to a general decision system. In our example above, using a table to represent the decision system, *Credibility* is the decision attribute. In fact, we will later on refer to “tables” when we talk about concrete decision systems (decision systems as implemented in a computer program).

Restricting ourselves to merely one attribute as the decision attribute is not a specialization of a more general case. Any learning situation where there are more than one decision attribute (say \( \{d_1 \ldots d_k\} \)) and all those attributes are nominal (taking a finite number of values) can be reduced into a single decision attribute with cardinality \(|d_1| \cdot |d_2| \cdot \ldots \cdot |d_k|\) where \(|d_i|\) is the size of the value domain of \( d_i \).

### 3.3.1 Reducts

Given any set \( B \subseteq A \) and an information system \( \langle U, A \rangle \), we can define a binary relation called the indiscernibility relation. This relation states that all objects \( u_i, u_j \) that have the same values on the attributes in \( B \) are indiscernible, or in the \( IND_S(B) \)-relation to each other.

**Definition 2 (Indiscernibility Relation)**

\[
IND_S(B) = \{(u, u') \in U^2, S = \langle U, A \cup \{d\} \rangle : \forall a \in B a(u) = a(u')\}
\]

Please note that \( IND_S(B) \) is an equivalence relation. The partition of \( U^1 \) as defined by \( B \) will be denoted \( U/B \) and the equivalence classes introduced by \( B \) will be denoted \([u]_B\). In particular, \([u]_d\) will be called the decision classes of the decision system.

---

1 That is, the division of \( U \) into a number of distinct subsets where each subset is an equivalence class according to the attributes in \( B \).
A super-reduct $B$ is a set of attributes in an information system $S$ such that $\text{IND}_S(A) = \text{IND}_S(B)$. Super-reducts are thus sets of attributes that keep the characteristics of our data. We reduce unnecessary attributes and only keep those that are required to tell two objects that have different values on some attribute in $A$ apart.

**Example 2** In table 3.1 (here representing an information system $S = \langle U, A \cup \{d\}\rangle$), objects $u_8$ and $u_9$ are identical, that is, they have equal values on all attributes in $A$. Another way to put it is that $[u_8]_A = [u_9]_A = \{u_8, u_9\}$. The set $B = \{\text{Occupation, Age}\}$ is not a super-reduct since it would make objects $u_5$ and $u_6$ impossible to differentiate, although they have different values on the attributes Shoesize and Credibility. In other words, $\text{IND}_S(A) \neq \text{IND}_S(B)$. Therefore, the only possible super-reduct of $S$ would be $A$.

If $B$ is a super-reduct and no proper subset $B'$ of $B$ can induce the equivalence relation $\text{IND}_S(B')$ equal to $\text{IND}_S(A)$, then $B$ is a reduct, or proper reduct as we will sometimes call it. The set of all reducts of $S$ is denoted $\text{RED}(S)$. Reducts are thus minimal sets of attributes that allow us to discern objects from each other. There may be many such sets that are minimal, but different in size compared to each other.

**Use of reducts**

In algorithm 4.6.4 we will use the notion of reducts to compute how to best divide the range of a numerical attribute into a set of intervals, so that the objects in the training set can still be discerned from one another by the set of intervals introduced.

### 3.3.2 Relative reducts

Although the definitions of reducts and discernibility used above are the basic definitions, we will mostly deal with decision systems $\langle U, A \cup \{d\}\rangle$ in this thesis and also be most interested in attribute sets $B \subseteq A$ such that $\text{IND}_S(B) = \text{IND}_S(\{d\})$. If $B$ is a minimal such set, then we will say that $B$

---

脚注:

1. i.e., $[u_5]_B = [u_6]_B = \{u_5, u_6\}$
2. also called discretization, see section 4.6
3.3. Rough Set basics

Table 3.2: Example of relative reducts

<table>
<thead>
<tr>
<th></th>
<th>a₁</th>
<th>a₂</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>u₁</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>u₂</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>u₃</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>u₄</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>u₅</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>u₆</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

is a relative reduct. That is, B is a minimal set of attributes that allows to tell all objects with different decision values apart. However, there may be objects that have different values on attributes in $A \setminus B$ that are considered indiscernible by the attributes in B.

**Example 3** Using table 3.2 to illustrate a decision system, $\{a_1\}$ would be a relative reduct since no two objects with the same value of $a_1$ have different decision values. $\{a_2\}$ on the other hand would not be a reduct since $a_2(u_1) = a_2(u_3)$ but $d(u_1) \neq d(u_3)$.

The reason for using relative reducts is of course that we want to find a subset of attributes that is important for us when determining the decision values of objects, in which case it doesn’t really matter that $a_2(u_1) \neq a_2(u_2)$ when using the relative reduct $\{a_1\}$ since $u_1$ and $u_2$ are classified the same.

The set of all reducts of $S$ relative to a decision attribute $d$ will be denoted $\text{RED}(S,d)$.

### 3.3.3 Approximations

Given the information system $\langle U, A \rangle$ and a set $B \subseteq A$ a B-lower approximation to a set of objects $X \subseteq U$ is the following set:

**Definition 3 (Lower Approximation)**

$$X_B = \{ u \in U : [u]_B \subseteq X \}$$
Rough Sets

That is, all elements of the equivalence class of \( u \), as determined by the attributes in \( B ([u]_B) \), must be in the set \( X \). Since all approximations are determined by a subset of attributes, we will by writing the lower approximation mean the B-lower approximation when we talk about the lower approximation to a set of objects and the subset \( B \) is given.

**Example 4** Algorithmically, we iterate through the rows of a decision table and for each row extract all those rows that have equal values as the current one on the columns referred to by \( B \). If all the rows are contained in \( X \), then we add the current row to the set that will become \( X_B \).

Using table 3.1 again, we define a set of objects \( X \) as \{\( u_1, u_4, u_5 \}\}, and a set \( B \) of attributes as \{\text{Age}\}. We can clearly see that \( X \) is the set of all objects that have the decision value High. \( X_B \) is \{\( u_4 \}\} since \([u_4]_B = \{u_4\}\) but \([u_1]_B = [u_5]_B = \{u_1,u_3,u_5,u_6,u_7\}\) \( \not\subseteq X \).

The upper approximation to a set is defined in a similar manner. All equivalence classes \([u]_B\) that have a non-empty intersection with a set \( X \) are in the B-upper approximation to \( X \).

**Definition 4 (Upper Approximation)**

\[
X_B = \{ u \in U : [u]_B \cap X \neq \emptyset \}
\]

**Example 5** Algorithmically, we iterate through the rows of a decision table and for each row extract, from the whole table, all those rows that have equal values as the current one on the columns referred to by \( B \). If only some of those rows are contained in \( X \), then we add the current row to the set that will become \( X_B \).

Following example 4 and the algorithm above, \( X_B \) is \{\( u_1,u_3,u_4,u_5,u_6,u_7 \}\). \([u_1]_B = \{u_1,u_3,u_5,u_6,u_7\}\) and \([u_1]_B \cap X = \{u_1,u_5\}\), so therefore all the elements in \([u_1]_B\) will be included in \( X_B \). Also, \([u_4]_B \cap X = \{u_4\}\) so \( u_4 \) will included in \( X_B \).

A positive region is the union of the lower approximations to the decision classes \([u]_{(d)}\). If \( C^1, \ldots , C^{[d]} \) are the decision classes of \( U \), then the B-positive region is \( C^1_B \cup \ldots \cup C^{[d]}_B \).

---

4 from the whole table
5 Assuming that \( V_d = \{1, \ldots , [d]\} \) and \( C' = \{ u \in U : d(u) = i \} \)
3.4. Dynamic Reducts

Use of approximations
Approximations are used in the implementation of LEM2 in algorithm 4.7.2.

3.3.4 Decision rules
Decision rules are propositional logic formulae of the form:

\[ a_1 = v_{a_1} \land \ldots \land a_j = v_{a_j} \Rightarrow d = v_d \]

where the \( a_i \)'s represent attributes and the \( v_{a_i} \)'s represent values of those attributes, just as in the example in section 3.3.4. Everything on the left-hand side of \( \Rightarrow \) is the set of conditions of the rule and to the right of \( \Rightarrow \) is the decision. The objects in table \( S \) that match a decision rule \( r \) are those that have the same values \( v_{a_1} \ldots v_{a_j} \) on the attributes \( a_1 \ldots a_j \) as the conditions in rule \( r \). Of those objects, the ones that also have the decision value \( v_d \) are said to support the rule.

Being formal, we may introduce some mathematical concepts that add some stringency to our reasoning. A Rule \( r \) is a pair \( \langle C, d_i \rangle \) where \( C \) is supposed to simply be a set of attribute-value pairs and \( d_i \) a decision value. A RuleSet \( \text{Rules}_{d_i} = \{ r : r = \langle C, d_i \rangle \} \) is a set of rules that all have as a consequent the same decision value. All the rules induced from a decision system is a set of such sets, \( \text{Rules} \). We can define it like this: \( \text{Rules} = \{ \text{Rules}_{d_i} : d_i \in V_d \} \).

Use of decision rules
Decision rules are generated by LEM2 and used to classify objects. All methods for negotiating between conflicting rules as described by the algorithms in section 4.8.2 act upon such rules.

3.4 Dynamic Reducts

Now, we will consider the dynamic reducts and the theory governing them. For the sake of simplicity, if the word “dynamic” has no other obvious meaning, it will from now on refer to the concepts of dynamic reducts.
3.4.1 Background

Reducts, as presented in section 3.3, are minimal sets of attributes that allow us to discern objects. It is considered that reducts are better if they contain fewer attributes. This is because we wish to build a model that is useful when we encounter unseen data and it is argued in [BSS94] that if we can make do with less attributes (and thus have a more compact model of our domain) we are better suited for making predictions. Thus it is considered interesting to find short reducts (reducts with few attributes). The problem of finding the shortest reduct is NP-hard [SR92] and algorithms that rely on heuristics to find short reducts have been thoroughly investigated. However, as a basis for decision rules\(^6\), reducts that are merely short do not always yield the better results than other reducts when they are used to classify data. For the sake of gaining predictive performance Bazan argues in [BSS94] that the reducts that occur in many subsets of the entire universe or, as he calls it, are as stable as possible, should be used. This is the main conjecture and motivation for finding dynamic reducts. The stability is thus the frequency with which reducts of a decision system also occur, as reducts, in “subsystems” of that decision system (see definition 5).

In [Baz98] Bazan claims that one way to obtain these stable reducts is to look at subsets of a training set and only add those reducts that are reducts of the whole set and also occur in more than a certain percentage of the subsets. In this way, given that the subsets are large enough (which translates into having a large enough training set), we could better simulate what would happen when the algorithm is subjected to unknown objects. The reason that is given is that by looking at reducts frequently occurring in subsets of the training set, we can make a statement like “these reducts are most frequently the ones that distinguish objects in this domain, for any given set of objects that we look at”. If we indeed find such a set, it is intuitively thought that the reducts identify patterns not only present in one specific set of objects, but in many.

\(^6\) by using reducts as conditional attributes in rules
3.4.2 Dynamic Reduct theory

We define Dynamic Reducts in terms of the set of reducts of a decision system $S$ and a set of reducts of the subsystems $S'$ of that decision system. A subsystem $S'$ is defined as:

**Definition 5 (Subsystem $S'$ of $S$)** Let $S$ be a decision system $\langle U, A \cup \{d\}\rangle$. Then $S'$ is a subsystem of $S$ iff $S' = \langle U', A \cup \{d\}\rangle, U' \subseteq U$.

The “powerset” $G$ of decision system $S$ can then be defined as follows:

**Definition 6 (Powerset $G$ of $S$)** Let $S$ be a decision system $\langle U, A \cup \{d\}\rangle$. Then $G$ is a powerset of $S$ iff $G = \{\langle U', A \cup \{d\}\rangle : U' \subseteq U\}$

**Definition 7 (Family of subsystems)** Let $G$ be a powerset of $S$ as given by definition 6. Then $F$ is a family of subsets iff $F \subseteq G$.

A dynamic reduct of $S$ is a reduct of $S$ that is also a reduct for subsystems $S'$ of $S$. Naturally, the definition would be quite useless if we required that some set of attributes would discern objects from each other in every subsystem of the original decision system. Therefore, we introduce $F, F \subseteq G$ that is a set of subsystems of $S$ and we say that a reduct $R$ of $S$ is a $F$-dynamic reduct if it is also a reduct for all the subsystems in $F$. If we use a decision table again to illustrate our decision system, then we pick out a number of sets, where each set is a number of rows. For each of these sets, we require that $R$ is a reduct for those rows in the table, i.e. a minimal set of attributes that properly\(^7\) discerns all objects in the set.

We will use $DR(S, F)$ as the short-hand notation of the set of reducts that are both reducts of the whole decision system $S$ and also for all the subsystems in $F$.

**Definition 8 (F-dynamic Reducts of $S$)**

$$DR(S, F) = RED(S, d) \cap \bigcap_{S' \in F, F \subseteq G} RED(S', d)$$

\(^7\)according to how a reduct is supposed to work (see section 3.3.1)
Definition 8 means that $DR(S,F)$ is the set of reducts in the original decision system that are also reducts of all the subsystems in the family of subsystems $F, F \subseteq G$. Please note that from now on, we are only talking about relative reducts when referring to reducts.

This requirement can be relaxed so that a dynamic reduct does not have to be a reduct of all subsystems in $F$ but only in many enough of them. This set is called the set of $(F, \varepsilon)$-dynamic Reducts and will be denoted $DR_\varepsilon(S,F)$, where $\varepsilon$ regulates to what degree a reduct of $S$ needs to be a reduct of the subsystems in $F$. It can be defined as follows:

**Definition 9 ($(F,\varepsilon)$-dynamic Reducts of $S$)**

$$DR_\varepsilon(S,F) = \{R \in RED(S,d) : \frac{|\{S' \in F : R \in RED(S',d)\}|}{|F|} \geq 1 - \varepsilon\}$$

In definition 9, we only require that a reduct in $S$ should also be a reduct in more than $(1 - \varepsilon) \cdot |F|$ subsystems in $F$.

If we cannot compute the reducts of the whole system, we may have to settle for the reducts we can compute for the subsystems. The concepts $F$-dynamic Reducts and $(F, \varepsilon)$-dynamic Reducts are then replaced with the concepts $F$-generalized Dynamic Reducts and $(F, \varepsilon)$-generalized Dynamic Reducts. The definitions of these two concepts are identical to those of their non-generalized counterparts, except that they don’t include the demand that the reducts should be reducts for the whole system $S$.

Later, by referring to a Dynamic Reduct of a decision system $S$, we will mean a reduct $R$ such that $R \in DR_\varepsilon(S,F)$ for some family $F$ and some $\varepsilon$, $0 \leq \varepsilon \leq 1$.

The reducts can be sorted with respect to how many subsystems they are reducts for, or equivalently, according to how stable they are:

**Definition 10 (Stability coefficient of a reduct)**

$$Stability(R) = \frac{|\{S' \in F : R \in RED(S',d)\}|}{|F|}$$

In general, only the most stable reduct will be of interest. Worth noticing is that this measure does not take into account the fact that the subsystems $S'$ come in different sizes.
3.5. LEM2 specifics

In the articles describing the various incarnations of LEM2 (see [CGB94] and [GB97]), some concepts are introduced that require explanation.

Some notation\(^8\) we use is described in table 3.3.

Using that notation, we introduce the concept of a minimal complex:

**Definition 11 (Minimal Complex of X)** Let \( S = \langle U, A \cup \{d\} \rangle \) be a decision system and let \( X \) be a subset of \( U \). Then \( T \) is a minimal complex of \( X \) if the following conditions hold:

- \( T \neq \emptyset \)
- \( [T]_U \neq \emptyset \)
- \( [T]_U \subseteq X \)
- there exists no \( T' \) such that \( T' \subset T \) and \( T' \) is a minimal complex

The semantics of a minimal complex is that \( T \) should be “the most general description of objects in the subsystem \( X \)”. It is equal to an equivalence class \([u]_B\) if we choose \( B \) correctly. Actually, the concept of minimal complex

---

\(^8\)In the original articles of the author, the letter \( t \) is used to denote the pair \( \langle a, v \rangle \) in table 3.3
is closely related to a lower approximation $X_B$ in the following way: First, extract from the pairs $\langle a, v \rangle$ in $T$ the attributes $a$ and let them form the set $B^9$. Since the second requirement states that $[T]_U \neq \emptyset$, we know that there are some objects $u \in U$ such that $[u]_B = [T]_U$. That is, the objects in $[u]_B$ have the values of all the attributes in $B$ as given by the pairs in $T^{10}$. The third requirement on $T$ is that $[T]_U$ should be a subset of $X$. Since it is clear that there exists a $u$ such that $[u]_B = [T]_U$, we can say that $[T]_U \subseteq X_B$.

**Definition 12 (Local Covering)** Let $S = \langle U, A \cup \{d\} \rangle$ be a decision system and let $X$ be a subset of $U$. Then $T$ is a local covering of $X$ if it meets the following conditions:

- $T \neq \emptyset$
- $T = \{T : T$ minimal complex$\}$
- $\bigcup_{T \in T} [T]_U = X$
- there exists no $T'$ such that $T' \subset T$ and $T'$ is a local covering

We say that $T$ is a minimal set of minimal complexes that exactly covers $X$. It consists of a number of equivalence classes $[u_1]_{B_1}, \ldots, [u_l]_{B_l}$, where $B_i, 1 \leq i \leq l$ is the set of the attributes contained in the minimal complex $T_i, T_i \in T$.

If the set $X$ is a subset of $U$ such that $d(x) = d_i$ for all $x \in X$, then we say that a minimal complex is actually the condition set (see section 3.3) of a rule, the decision of which is the same as for the objects in $X$. The local covering then is a set of such rule conditions, something that we will later refer to as a RuleSet when the set is paired with a decision value. The name covering comes from the fact that $T$ is supposed to contain the descriptions $T$ such that they cover the entire input set $X$ and nothing more than $X$. It is in no way required that $X$ should actually be a decision class however, but it might be easier to think of it that way since we will not use the concepts of coverings and complexes for any other purpose in this thesis.

---

9 In set-notation, $B$ would be $\{a : \langle a, v \rangle \in T\}$

10 See the set-notation describing $[T]_U$ in table 3.3
3.5. LEM2 specifics
Chapter 4

Algorithms

This chapter describes the algorithms that are the main focus of this work. The pseudocode has been designed to faithfully reflect the implementations and has been constructed after the actual implementation phase.

The implementation began with some algorithms for computing reducts and upper/lower approximations, the implementations of which will be presented in section 4.3. These later served as a framework for the filtering and classification algorithms. Apart from those, we will follow the structure given in figure 4.1\(^1\) when we describe the algorithms in this chapter. Thus, we continue by discussing the methods used for preparing (filtering) the data for the model construction part in sections 4.4-??. After that, the methods for actually constructing a model are explained in section 4.7 and last, the methods for rule negotiation (classification of data) are described in section 4.8.

It is worth noticing that LEM2 as an algorithm (as described in [GB97]) actually only corresponds to algorithm 4.7.1 in this chapter. The rest of “The LEM2 Approach” (see section 4.2) has been composed by Marcin Szczuka, the thesis’ supervisor, in part due to that the details of the ML system LERS (Learning from Examples using Rough Sets) in which LEM2 was originally implemented have not been fully disclosed by the author.

\(^1\)Same as figure 2.1
4.1 Notation

The function \( \text{argmax}_{i \in I} f(i) \) will be used to extract the object \( i \) from set \( I \) with the maximal value of \( f(i) \), correspondingly for \( \text{argmin}_{i \in I} f(i) \). In the cases where several \( i \) are maximal w.r.t. \( f(i) \), \( \text{argmax} \) will choose one randomly unless otherwise specified. By writing \( \text{max}_{i \in I} f(i) \), we mean a function that will return the maximal value of \( f \).

In the algorithms below, we will use some conventions from the previous chapter as explained in table 4.1. When various parts of the algorithms are described statements will be referred to by their line numbers, shown in the left margin.

4.2 The approaches

This section describes the composition of algorithms in the two approaches. In table 4.2 the algorithms for filtering the datasets and constructing the model used in each of them are listed. There are three things to note:

1. The Selection of attributes refers to selecting a subset of the total set of attributes (i.e. perhaps also nominal attributes) for the discretization process (see section 4.6).
<table>
<thead>
<tr>
<th>Name</th>
<th>Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = \langle U, A \cup {d} \rangle$</td>
<td>a decision system</td>
</tr>
<tr>
<td>$U$</td>
<td>A set of objects. Also called universe</td>
</tr>
<tr>
<td>$</td>
<td>U</td>
</tr>
<tr>
<td>$u \in U$</td>
<td>an element in $U$, also referred to as an object</td>
</tr>
<tr>
<td>$A$</td>
<td>the set of all attributes in a decision system</td>
</tr>
<tr>
<td>$[u]_B$</td>
<td>${u_i \in U : \forall a \in B(a(u_i) = a(u))}$ It is assumed that $B \subseteq A$.</td>
</tr>
<tr>
<td>$</td>
<td>A</td>
</tr>
<tr>
<td>$a \in A$</td>
<td>an attribute in $A$ that acts as a function $a : U \mapsto V_a$</td>
</tr>
<tr>
<td>$V_a$</td>
<td>the value domain of attribute $a$</td>
</tr>
<tr>
<td>$d$</td>
<td>decision attribute or a function that assigns a decision value to an object</td>
</tr>
<tr>
<td>$</td>
<td>d</td>
</tr>
<tr>
<td>$D$</td>
<td>Set of cuts after the discretization process (see section 4.6)</td>
</tr>
<tr>
<td>$\langle a, v \rangle$</td>
<td>a pair of an attribute $a$ and a value $v \in V_a$</td>
</tr>
<tr>
<td>$T$</td>
<td>A set of pairs $\langle a, v \rangle$</td>
</tr>
<tr>
<td>$\langle a, v \rangle_U$</td>
<td>${u \in U : a(u) = v}$</td>
</tr>
<tr>
<td>$[T]_U$</td>
<td>${u \in U : \forall \langle a_i, v_i \rangle \in T(a_i(u) = v_i)}$</td>
</tr>
</tbody>
</table>

Table 4.1: Conventions
4.2. The approaches

<table>
<thead>
<tr>
<th></th>
<th>Dynamic Approach</th>
<th>The LEM2 Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing values</td>
<td>Rough Set method</td>
<td>Rough Set method</td>
</tr>
<tr>
<td>replacement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection of</td>
<td>some nominal +</td>
<td>some nominal +</td>
</tr>
<tr>
<td>attributes</td>
<td>all numeric</td>
<td>all numeric</td>
</tr>
<tr>
<td></td>
<td>all numeric</td>
<td></td>
</tr>
<tr>
<td>Discretization</td>
<td>Dynamic Discret-</td>
<td>Global Discret-</td>
</tr>
<tr>
<td></td>
<td>ization (algo-</td>
<td>ization (algo-</td>
</tr>
<tr>
<td></td>
<td>rithm 4.6.4)</td>
<td>4.6.3)</td>
</tr>
<tr>
<td></td>
<td>Fayyad-Irani [FI93]</td>
<td>Fayyad-Irani [FI93]</td>
</tr>
<tr>
<td>Rule Creation</td>
<td>LEM2 + rule</td>
<td>LEM2 + rule</td>
</tr>
<tr>
<td></td>
<td>approx.</td>
<td>approx.</td>
</tr>
<tr>
<td></td>
<td>No rules</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(algorithm 4.8.1)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Filtering and building a model

2. The Rough Set method for replacing missing values refers to a method described in section 4.5.

3. The rule approximation used together with LEM2 was not included in the original LEM2-algorithm but is a commonly used method for improving predictive performance (see section 4.7.2).

The “Dynamic Approach” as described here is actually only a subset of all the sub-algorithms that were included originally by Bazan in his comparison of dynamic techniques and other ML techniques. However, this subset was considered to be the most interesting aspects of the original Dynamic Approach taken by Bazan.

Table 4.3 lists the different options available in each approach once rules have been constructed and objects are to be classified. Observe that once the common subparts of both algorithms have been factored out, they only differ in the choice of discretization method and rule negotiation. Bazan originally made experiments where also the rule construction phase of his algorithm used the notions of dynamic reducts but that part, along with
4.3 Rough Set Praxis

Using discernibility matrices and discernibility functions (see [SR92]) were the original way used for constructing reducts. These concepts were very straightforward to implement and use but the time and space complexity of the methods soon made it imperative to find new and more efficient Rough Set algorithms.

In [NN96] some significantly more effective Rough Set algorithms were presented. The main idea in the article was to manipulate the equivalence classes \([u]_B\) (the objects in \(IND_S(B)\)-relation to each other) of a decision system as given by a subset of the set of attributes. They showed that both reduct calculation and discretization could be efficiently performed using these equivalence classes. For instance, one of the algorithms calculates a reduct in \(O(k^2 \cdot n \cdot \log_2 n)\) time, where \(k\) is the number of attributes and \(n\) the number of objects in the dataset. I will not explain the details of these basic algorithms, but it should be mentioned that all of those described in [NN96] (including methods for computing single reducts, upper/lower approximations and equivalence classes modulo attribute sets) were implemented as a part of this thesis work. The only algorithm described separately here is algorithm 4.3.1 that computes a set of proper

---

2 reference: personal conversation
4.3. Rough Set Praxis

Algorithm 4.3.1 Calculate \( n \) reducts from a super-reduct \( R \)

\[
\text{RandomPermutation}(A)
\]
1  /* \( A = a[1 \ldots m] \) */
2  for \( i \leftarrow m \) to 2
3  do \( r \leftarrow \text{RandomNumberIn}(1,i) \)
4  Swap\((a_r,a_i)\)
5  return \( A \)

\[
\text{ProperReducts}(R,S,n)
\]
1  \( R \leftarrow \emptyset \)
2  for \( i \leftarrow 1 \) to \( n \)
3  do \( R \leftarrow \text{RandomPermutation}(R) \)
4  for each \( a \in R \)
5  do if \( \text{IsReduct}(R \setminus \{a\},S) \)
6    then \( R \leftarrow R \setminus a \)
7  \( R \leftarrow R \cup R \)
8  return \( R \)

reducts. The algorithm was considered best suited here since it is used both as a stand-alone method for computing reduct sets and integrated in the dynamic discretization algorithm (algorithm 4.6.4).

4.3.1 Reduct Computation

In order to construct reducts of a decision system, algorithm 4.3.1, from article [Wró98], was implemented. The use of these reducts in this thesis is to extract a set of proper reducts from a super-reduct. This procedure is used in the algorithm describing the dynamic discretization (algorithm 4.6.4).

The attributes in the super-reduct \( R \) are randomized, to ensure that we have a equal chance of computing every possible proper reduct \( R' \subset R \). \( \text{IsReduct}(R) \) is a function that returns \text{true} if \( R \) is a reduct of \( S \), false otherwise, by checking whether the positive region of \( R \) equals \( |U| \). That is, if all objects can be certainly classified using the attributes in \( R \). The
parameter $n$ to algorithm 4.3.1 was not used in [Wró98], and the reason it is included here as a limit on the number of iterations to perform is that other stop conditions used in a more exhaustive search cause the time complexity of the algorithm to become exponential (for example, if one would try all subsets of $R$ and see if they are proper reducts).

### 4.3.2 Dynamic Reduct Computation

One of the things that make dynamic reduct computation challenging is the fact that so many parts of the computation may severely compromise the overall performance of the system if implemented poorly. Therefore, great care was taken to ensure that if not the best, then at least the same methods as the original authors used, would also be incorporated in this version of a dynamic reduct algorithm. In this section we will present the details of how the theory of dynamic reducts of Bazan were used in this thesis. However, as mentioned in table 4.2 the Dynamic Approach is composed of several parts and we will deal with them separately in sections 4.4, 4.8.4 and 4.7. In this section the basics of the Dynamic Reduct Approach, that did not fit well into the other sections, are explained.

As stated in table 4.2 and table 4.3, the Dynamic Reduct Approach consists of two steps where *dynamic techniques* are actually used:

- discretization
- rule negotiation

In these two steps we use some of the techniques described in section 3.4.2. Computing the stability of a $(F, \varepsilon)$-dynamic reduct is the most complicated and time-consuming step in the Dynamic Approach. Algorithm 4.6.4 (dynamic reduct discretization) is the one that utilizes this calculation.

#### Stability calculation

In order to calculate the stability$^3$ of a reduct, we need to do the following:

$^3$line 17 of algorithm 4.6.4, dynamic reduct discretization
1. **Determine minimal subtable size** A dynamic reduct is a reduct not only for the original system but also for a set of subsystems (see section 3.4.2). The size of each of these subsystems\(^4\) should not matter in theory, but in practice it does. Therefore, some heuristics have to be applied in order to determine the proper subtable size. The subtables should be small enough to facilitate the calculation of reduct stability, but large enough to be representative samples of the training set.

2. **Determine family size** As Bazan discusses in [Baz96] a certain number of subtables should be generated in order to make statistical statements about the quality of the dynamic reducts.

3. **Determine stability of a reduct** Using the measurement of reduct stability as defined in definition 10, the stability of the cuts are determined.

### 4.4 Filtering

Now let’s turn to how one can filter the data so that various models of it can be constructed.

We will be interested in generating rules that can classify objects in a dataset. The construction of these rules assumes that all attributes only take a limited, small number of values\(^5\) and that all attribute values can be used when constructing a model (i.e. no attribute values will be considered special during the model construction phase, not even missing values). If there are numeric\(^6\) attributes in our dataset they will be converted into nominal attributes with values corresponding to intervals in the range of the numerical attributes. To maintain the characteristics of the data, the

---

\(^4\)Given the decision system \(\langle U, A \cup \{d\} \rangle\), the size of the system is \(|U|\)

\(^5\)It can be argued that also numerical attributes could be used, but since the predictive quality resulting from using such attributes as conditions in a decision rule is very low, it is rarely used.

\(^6\)One may also want to discretize nominal attributes so as to decrease the number of values they range over. This can be done by simply filtering the nominal attributes as if they were numeric.
discretization will be performed so that objects with different decision values will still be considered different after the conversion.

4.5 Missing values

Methods for filling in missing values in datasets are already available in Weka but this method, called ReplaceMissingValuesFilter, doesn’t take advantage of the fact that the decision attribute of the dataset might be known during the filtering stage. Since the outcome of a learning process depends much on how well the different parts of the process are suited for each other, it was decided that a filter for dealing with missing values that used the information about which attribute was going to be used as the decision attribute, should be implemented.

The previously implemented class for filtering missing values in Weka simply filled the missing attribute values by taking the most frequent attribute value of nominal attributes and the mean value of numeric attributes. This class was modified to become a “supervised filter”, with knowledge of the decision attribute, called RSReplaceMissingValues.

It is based on the following algorithm devised by Marcin Szczuka and assumes that the algorithm is dealing with tables like the ones in 3.1 and 3.2: for each object $u_i$ (row in the table), for nominal attributes fill in the most frequent value of the attribute in the class $[u]_d$ (see table 4.1). For numeric attributes, fill the missing value with the mean value of the attribute in $[u]_d$. This is thought to better suit the learning process later, when it is intended that data should be grouped according to the decision classes. However, one could object that this could introduce an unnatural bias towards the predominant attribute value in a dataset with many unknown values for that attribute. From an engineering point of view, the solution would simply be to disregard such an uncertain attribute since it would be unsuited for inclusion in a final model anyway. In our datasets, no such attribute existed.
Computational complexity

The above mentioned method for filling missing values will first have to calculate the frequencies of the values for each nominal attribute \(O(m \cdot n)\), assuming at most \(m\) nominal attributes and \(n\) objects in the decision system) and then update the objects with missing attribute values \(O(n)\). Thus, in total it will take \(O(m \cdot n)\) time.

4.6 Discretization

Discretization is a technique for converting a possibly large number of values into a smaller set of values. You can picture the situation as in figure 4.2 where the small circles are supposed to be objects plotted along the horizontal axis of an attribute. Between each plotted attribute value we introduce potential cutpoints, some of which are later used to discern the objects from each other and separate them into a discrete number of intervals. The method most commonly used for calculating the quality of a cutpoint is the Fayyad-Irani method (see [FI93]) which employs information gain as a quality function. However, other measures of quality have been proposed, among them the Rough Set-based method which uses discernibility (see [NS95]).

4.6.1 Discernibility

Since we are only interested in separating objects that have different decision values, each cut in our representation is given information about how many objects from each decision class are to the left and to the right of

![Figure 4.2: Cutpoints](image-url)
Algorithms

<table>
<thead>
<tr>
<th>Name</th>
<th>Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a,c)$</td>
<td>A cutpoint $c$ on an attribute $a$ dividing all objects in a decision system in two parts - ${u \in U : a(u) &lt; c}$ and ${u \in U : a(u) &gt; c}$</td>
</tr>
<tr>
<td>$D$</td>
<td>A set of cuts $(a, c)$</td>
</tr>
<tr>
<td>$AllCuts_S$</td>
<td>All possible cuts on the decision system $S$</td>
</tr>
<tr>
<td>$L$</td>
<td>Short explanation: $U/{(a, c) \in D}$ or $U/B, B \subseteq A$ (see section 3.3.1). The first case perhaps needs some lengthier explanation: First, let $D$ be a vector of cuts. Second, let $df : U \mapsto \text{boolean}^{</td>
</tr>
<tr>
<td>$l^X(a,c), r^X(a,c)$</td>
<td>number of elements that are to the left/right of the cut $(a, c)$ in the equivalence class $X$</td>
</tr>
<tr>
<td>$l^X_i(a,c), r^X_i(a,c)$</td>
<td>number of elements with decision value $i$ in equivalence class $X$ that are to the left/right of the cut $(a, c)$</td>
</tr>
<tr>
<td>$c_j$</td>
<td>a numerical value indicating where the cut is made</td>
</tr>
</tbody>
</table>

Table 4.4: Discernibility Conventions

The cut, i.e. how many pairs of objects with different decision values that are discerned from each other. The algorithm where this measure is later used sequentially deals with each attribute and the set of cuts that may be introduced on that attribute. By assuming that we can totally order all objects so that they primarily are sorted on the value of the current attribute and secondly in some arbitrary order, one can use algorithm 4.6.1 to calculate the discernibility value of a cut in constant time, and in $O(m \cdot n)$ (see table 4.1) time for the whole table.

Before turning to the details of algorithm 4.6.1, internal notation used in the algorithm is explained in table 4.4.
Algorithm 4.6.1 Calculate the number of objects discerned by a cut

\[
W_L(a, c_j)
\]

1. \( N \leftarrow 0 \)
2. \( \text{if } j = 1 \)
3. \( \text{then for each } X \in L \)
4. \( \text{do } N \leftarrow N + (l^X(a, c) \cdot r^X(a, c)) - \sum_{i=1}^{d} l_i^X(a, c) \cdot r_i^X(a, c) \)
5. \( \text{else } \)
6. \( \text{between } \leftarrow \{ x \in (X \in L) : c_{j-1} \leq a(x) < c_j \} \)
7. \( \text{cut}_{old} \leftarrow (a, c_{j-1}) \)
8. \( \text{cut}_{new} \leftarrow (a, c_j) \)
9. \( \text{for each } x \in \text{Between} \)
10. \( \text{do } t \leftarrow d(x) \)
11. \( N \leftarrow W_L(\text{cut}_{old}) + (r^X_t(\text{cut}_{old}) - l^X_t(\text{cut}_{old})) \)
12. \( N \leftarrow N - (r^X_t(\text{cut}_{old}) - l^X_t(\text{cut}_{old})) \)
13. \( \text{return } N \)

Please note that the pseudocode hides the fact that in the actual implementation, each cutpoint actually contains information about how many objects are to the left and to the right of it, so the loop at the lines 8-12 will accumulate this information about each successive cut. Thus, \( l^X_t(\text{cut}) \) can be calculated in constant time. The proof that this algorithm is correct can be found in [NN96].

As a comment to line 8, if \( |\text{between}| > 1 \) we iterate through all virtual cutpoints that could be introduced for the set of objects between two cutpoints (the curved lines in figure 4.3) and consider only the last one of them to be a “real” cut, later compared to others when deciding on the resulting set of cuts.

### 4.6.2 Local Discretization

Algorithm 4.6.2 is described in [NN96]. It works by finding a maximally discerning cut (see algorithm 4.6.1) from the set of all possible cuts (\( \text{AllCuts} \))
and then dividing the dataset into two subsets as long as there are objects with different decision values.

In line 2, we partition the data into equivalence classes $[u]_B$, where $B$ is the set of attributes that will not be discretized.

### 4.6.3 Global Discretization

In algorithm 4.6.3, we work with decision classes and check each consecutive cut that is added to the final set $D$ against all objects that are not completely separated into equivalence classes uniform w.r.t. decision value by the current set of cuts. We split the decision classes into smaller and smaller parts until they are uniform with respect to the decision values of the objects. Comparing this approach to algorithm 4.6.2, one can see that it will be able to generate a potentially larger set of cuts, since we take into account the partitioning induced by all cuts and not only the latest one.

---

7 one set where all objects are to the left of the cut (line 5) and one where all are to the right of the cut (line 6)

8 In reality, there are not only numerical attributes or only nominal ones in decision systems, but quite often both. In such situations we can not require the dataset to be split entirely by the set of cuts. In such cases we first divide the data into equivalence classes modulo the attributes that are not to be discretized
Algorithm 4.6.2 Local Discretization

**NumClasses**(*S*)
1. **return** number of decision classes in *S*

**Traverse**(*S*)
1. if **NumClasses**(*S*) > 1
2. then 
   
   \[
   (a^*, c^*) \leftarrow \text{argmax}_{(a,c) \in \text{AllCuts}_S} W_{U/B}(a,c)
   \]
3. \[D \leftarrow D \cup \{(a^*, c^*)\}\]
4. \[\text{AllCuts}_S \leftarrow \text{AllCuts}_S \setminus \{(a^*, c^*)\}\]
5. \[U_1 \leftarrow \{x \in U : a^*(x) < c\}\]
6. \[U_2 \leftarrow \{x \in U : a^*(x) \geq c\}\]
7. Traverse(*U*₁)
8. Traverse(*U*₂)

**LocalDiscretization**(*S*)
1. /* B is the set of attributes that will not be discretized */
2. **AllCuts**ₜₛ ← the set of all possible cuts in *S*
3. \[D \leftarrow \emptyset\]
4. Traverse(*U*)
5. Discretize *S* using the cuts in *D*
Algorithm 4.6.3 Global Discretization

\texttt{NumClasses}(S)
1 \hspace{1em} \textbf{return} \hspace{0.5em} \text{number of decision classes in } S

\texttt{GlobalDiscretization}(S)
1 /* B is the set of attributes that will \textit{not} be discretized */
2 \texttt{AllCuts}_S \leftarrow \text{the set of all possible cuts in } S
3 \texttt{D} \leftarrow \emptyset
4 \texttt{L} \leftarrow U/B
5 \text{repeat}
6 \hspace{1em} a^*, c^* \leftarrow \arg\max_{(a,c) \in \text{AllCuts}_S} W_L(a,c)
7 \hspace{1em} \texttt{D} \leftarrow \texttt{D} \cup \{(a^*, c^*)\}
8 \hspace{1em} \texttt{AllCuts}_S \leftarrow \texttt{AllCuts}_S \setminus \{(a^*, c^*)\}
9 \hspace{1em} \text{for each } X \in \texttt{L}
10 \hspace{2em} \texttt{L} \leftarrow \texttt{L} \setminus \{X\}
11 \hspace{1em} \text{if } \texttt{NumClasses}(X) > 1
12 \hspace{2em} \text{then } X_1 \leftarrow \{x \in X : a^*(x) \leq c^*\}
13 \hspace{2em} X_2 \leftarrow \{x \in X : a^*(x) > c^*\}
14 \hspace{1em} \texttt{L} \leftarrow \texttt{L} \cup \{X_1, X_2\}
15 \hspace{1em} \text{until } \texttt{L} = \emptyset
16 \hspace{1em} \text{Discretize } S \text{ using the cuts in } \texttt{D}
Computational complexity

As described in [NN96], the time complexity of algorithm 4.6.3 is \(O(m^2 \cdot n^2)\).\(^9\)

4.6.4 Dynamic Discretization

The dynamic method of discretization devised by Bazan and used in his comparison article [Baz98] was implemented as in algorithm 4.6.4. In the article, the algorithm itself was only briefly sketched and what is presented as an algorithm for dynamic discretization here is something Bazan in person has explained.

When discretizing a table with this algorithm, we work with reducts, or more specifically, reducts that are sets of cuts rather than sets of attributes. To explain this, we consider a new decision system \(S_{\text{ext}} = \langle U, A_{\text{nom}} \cup A_{\text{bin}} \cup \{d\} \rangle\) with both the original, nominal attributes \((A_{\text{nom}})\) and a set of new binary attributes \((A_{\text{bin}})\) corresponding to a set of cuts on \(S\) such that each attribute \(a_c\) in the attribute set \(A_{\text{bin}}\) corresponds to a cut \(\langle a, c \rangle\) in \(D\), the set of possible cuts on \(S\). Then the values of the new attributes \(a_c\) for each object \(u\) are like this:

\[
\begin{align*}
    a_c(u) &= 0 \quad \text{iff} \quad a(u) < c \\
    a_c(u) &= 1 \quad \text{iff} \quad a(u) > c
\end{align*}
\]

The cuts in \(S\) can thus be considered as attributes in this new decision system. It might be that the values of the numerical attributes are not enough to tell objects with different decision values apart, just as in the previous discretization algorithms. To remedy this, we can imagine that we split \(S_{\text{ext}}\) into a number of subsystems \(S'_{\text{ext}} \in \{ ([u_i]_{A_{\text{nom}}}, A_{\text{bin}} \cup \{d\}) : u_i \in U \}\). Then we say that the property of being a super-reduct of \(S_{\text{ext}}\) is equivalent to being a super-reduct to \(S'_{\text{ext}}\) for all \(u_i \in U\).

Algorithm 4.6.4 first generates a set of cuts \(D\) that are supposed to suffice to separate the objects into their respective decision classes. That is...

\(^9\)In [NN96], the expression is \(O(m \cdot n(|P| + \log_2 n))\) where \(P\) is supposed to be the set of cuts, which of course can be limited by \(m \cdot n\). In complexity analysis, we simply disregard terms of lower rank since they are unimportant, so the \(\log_2 n\) term will disappear.
Algorithm 4.6.4 Dynamic Discretization

\begin{algorithm}
\caption{DynamicDiscretization($S, n$)}
\begin{algorithmic}[1]
  \STATE /* bestCut is the best cut on each attribute */
  \STATE $D \leftarrow \emptyset$
  \STATE $AllCuts_S \leftarrow$ the set of all possible cuts in $S$
  \FOR {each $a \in A$}
    \STATE bestCut $\leftarrow$ GetBestCut($AllCuts_S, a$)
    \STATE $D \leftarrow D \cup \{\text{bestCut}\}$
    \STATE $AllCuts_S \leftarrow AllCuts_S \setminus \{\text{bestCut}\}$
  \ENDFOR
  \WHILE {D is not a super-reduct of $S$}
    \FOR {each $a \in A$}
      \IF {D is not a super-reduct of $S$}
        \STATE bestCut $\leftarrow$ GetBestCut($AllCuts_S, a$)
        \STATE $AllCuts_S \leftarrow AllCuts_S \setminus \{\text{bestCut}\}$
        \STATE $D \leftarrow D \cup \{\text{bestCut}\}$
      \ENDIF
    \ENDFOR
    \STATE $S_{\text{binary}} \leftarrow \text{Convert}(S, AllCuts_S)$
    \STATE $R \leftarrow \text{ProperReducts}(D, S_{\text{binary}}, n)$
    \STATE $R^* \leftarrow \text{argmax}_{R \in R} \text{Stability}(R)$
  \ENDWHILE
  \STATE Discretize $S$ using the cuts in $R^*$
\end{algorithmic}
\end{algorithm}

done by first using a greedy method (GetBestCut($cuts, a$)) choosing the most discerning cut on each attribute $a$ from the possible set of cuts $cuts$ and then iterating through the set of attributes and successively adding cuts until the data has been completely separated into equivalence classes (in the binary table described above). After this, a set of proper reducts (line 15) on $S_{\text{binary}}$ is computed from this set (see the paragraph above) and selects the most stable one as the final set of cuts to use for discretization. The $\text{Convert}$-procedure simply creates the new “binary” table.

### Computational complexity

To obtain a super-reduct, we may have to add all the possible cutpoints, possibly $m \cdot n$. The set $AllCuts_S$ of cuts can be initialized as a sorted set of
4.6. Discretization

cuts in $O(m \cdot n \cdot \log_2 n)$ time\footnote{Each of the $m \cdot n$ cuts can be put in a bin associated with the attribute of the cut. Assuming that the bins are sorted then insertion can be made by performing some binary search which doesn’t require more than $\log_2 n$ time. Multiply the two expressions and you get $m \cdot n \cdot \log_2 n$} but that only has to be done once. However, until a super-reduct has been obtained, we must check before each iteration if $\mathbf{D}$ constitutes a super-reduct, which takes $O(m \cdot n \cdot \log_2 n)$ time\footnote{if the efficient algorithm from [NN96] for computing the positive region, as a check for whether an attribute set is a super-reduct, is used}.

Thus the construction of a super-reduct will have time complexity $O(m^2 \cdot n^2 \cdot \log_2 n)$. When performing line 15, ideally all proper reducts would have to be calculated, which in a naïve way could be implemented by checking every every element in the powerset of $\mathbf{D}$ and see if it is a super-reduct. If it is a super-reduct, then it is inserted into $\mathbf{R}$ only if no subset of it already exists in $\mathbf{R}$. This algorithm would have an exponential time complexity $O(m^2 \cdot n \cdot 2^m \cdot n)$. The relaxation made in the experiments in chapter 6 (and in algorithm 4.6.4) was that the computation for finding proper reducts was only allowed to run for a constant number of iterations. During each iteration, the cuts in $\mathbf{D}$ were permuted and trimmed down to a proper reduct. The trimming was performed as in algorithm 4.3.1 by successively removing cuts and checking whether the positive region was kept constant. Iterating through at most $m \cdot n$ cuts and performing a $O(m \cdot n \cdot \log_2 n)$ calculation every time (positive region calculation from [NN96] again) a constant number of times gives us a time complexity of $O(m^2 \cdot n^2 \cdot \log_2 n)$, just as for the construction of the super-reduct.

Now to the dynamic part - calculating the stability of the proper reducts. During the experiments, this calculation was done on a constant number of subsystems, the sizes of which were at least a constant number $k$ ($0 < k \leq 1$) times the size of the original system (see section 4.3.2). For each proper reduct, at most a constant number of them (as given in the paragraph above), testing whether they were proper reducts on each of the subsystems only had time complexity $O(m^2 \cdot n^2 \cdot \log_2 n)$. Thus, the whole algorithm would have the time complexity $O(m^2 \cdot n^2 \cdot \log_2 n)$, although the many hidden constants may introduce significant differences between this approach and others of the same computational complexity.
4.7 Model construction

Now, we will focus on the third row of table 4.2, i.e. how to construct a model. As listed in that table, there are two options, namely the “No Rules” and “LEM2” options. We will only look at the “LEM2” option, since “No Rules” mean that we do not use an internal model but rather use the entire training set as a basis for our decisions on unknown data. “No Rules” correspond to algorithm 4.8.1 explained in section 4.8.

4.7.1 LEM2

Algorithms 4.7.1 and 4.7.2 describe the core of LEM2. LEM2 is the learning algorithm that, along with the dynamic reduct algorithms of Bazan, is the main focus in this thesis. Its first inception was as a part of the LERS learning system [GB92] which, in turn, was a development of the earlier classification algorithm PRISM [Cen87]. Grzymala-Busse claims to achieve much better results than PRISM in the general case [CGB94] by using techniques for reducing the risk of overfitting.

The input to LEM2 is a subsystem $X$ (see definition 5) to a decision system $S = \langle U, A \cup \{d\} \rangle$. First, we try to cover a set of objects with pairs $t$ (short for $\langle a, v \rangle$ in table 4.1) and create a Minimal Complex called Condition (see definition 11) out of these (line 7). Then, we prune the complex of unnecessary pairs (lines 12-14) and add it to a Local Covering called Covering (see definition 12). This covering is later reduced (lines 16-17) to only contain necessary complexes. These two pruning steps are what makes this algorithm interesting regarding the issue of overfitting. By removing parts of the final rule set that are redundant, the author of LEM2 has tried to improve the predictive performance of the original PRISM. However, this very algorithm has been developed on its own and has now only a few concepts in common with PRISM.

The expression $[\langle d, d_i \rangle]_U$ denotes decision class $i$, that is, all objects in $U$ that have decision value $d_i$.\footnote{the value of attribute $d$}
Algorithm 4.7.1 The original LEM2 rule construction algorithm

\textbf{Weight}(\textit{Pair, G})
\begin{algorithmic}[1]
  \State \textbf{return} \(|\text{[pair]}_U \cap G|
\end{algorithmic}

\textbf{BestPair}(\textit{P, G})
\begin{algorithmic}[1]
  \State \texttt{best} $\leftarrow \text{argmax}_{p \in P} \text{Weight}(p, G)$
  \State \textbf{return} \texttt{best}
\end{algorithmic}

\textbf{LEM2}(\textit{X})
\begin{algorithmic}[1]
  \State \texttt{/* $X =$ subsystem of $S.S = \langle U, A \cup \{d\}$ */}
  \State \texttt{G} $\leftarrow U'$
  \State \texttt{Covering} $\leftarrow \emptyset$
  \While {\texttt{G} \neq \emptyset}
    \State \texttt{Allrelevant} $\leftarrow \{t : \text{[t]}_U \cap G \neq \emptyset\}$
    \While {\texttt{Condition} = \emptyset \text{ or } \text{[Condition]}_U \not\subseteq U'}
      \State \texttt{t} $\leftarrow \text{BestPair(Allrelevant, G)}$
      \State \texttt{Condition} $\leftarrow \text{Condition} \cup \{t\}$
      \State \texttt{G} $\leftarrow \text{[t]}_U \cap G$
      \State \texttt{Allrelevant} $\leftarrow \{t : \text{[t]}_U \cap G \neq \emptyset\}$
    \EndWhile
    \For {each \texttt{t} $\in \text{Condition}$}
      \State \If {\text{[Condition} – \{t\}]}_U \subseteq X}
        \State \texttt{Condition} $\leftarrow \text{Condition} – \{t\}$
        \State \texttt{Covering} $\leftarrow \text{Covering} \cup \{\text{Condition}\}$
      \EndIf
    \EndFor
  \EndWhile
  \For {each \texttt{Condition} $\in \text{Covering}$}
    \State \If {\bigcup_{S \in \text{Covering} – \{\text{Condition}\}[S]}_U = U'}
      \State \texttt{Covering} $\leftarrow \text{Covering} – \{\text{Condition}\}$
    \EndIf
  \EndFor
  \State \textbf{return} \texttt{Covering}
\end{algorithmic}
Algorithm 4.7.2 Our LEM2 rule construction algorithm

LEM2 CLASSIFIER$(S)$

1 /* Calculate rules from the decision system */
   $S = \langle U, A \cup \{d\} \rangle$ */
2 $Rules \leftarrow \emptyset$
3 for each $\langle Class, d_i \rangle \in \{\langle (d, d_i) \rangle_{U, d_i} : d_i \in V_d \}$
4    do $Rules \leftarrow Rules \cup \{\{\text{LEM2}(\text{Class}_i), d_i \}\}$
5      $Rules \leftarrow Rules \cup \{\{\text{LEM2}(\text{Class}_d), d_i \}\}$
6

Computational complexity

The computational complexity of algorithms 4.7.2 and 4.7.1 are easily computed. First, it can be assessed that the LEM2 algorithm is run exactly $|d|$ times, where $|d|$ is the number of decision classes. The main iteration in the LEM2 algorithm (while $G$ is non-empty) is performed at most $n$ times, where $n$ is the number of objects in the training set. Since we may have the whole set as the upper approximation to every decision class, we have to perform the outer loop of the LEM2 algorithm at most $|d| \cdot n$ times. In the covering part (beginning at line 7) we have to iterate through all possible pairs of attributes and values $(n \cdot m)$ to extract the ones relevant to $G$ and of those find the best one. The pruning steps (lines 12-14 and 16-17 have computational complexities $O(m \cdot n)$ and $O(n^2)$) respectively ($\text{Condition}$ contains $m$ elements at most and $\text{Covering}$ contains no more than $n$ elements). Put another way, the total space required by $\text{Covering}$ will not exceed that of the whole table which a constant factor times $n \cdot m$.

Given the information above, it can be concluded that the whole algorithm will have computational complexity $O(|d| \cdot n^3)$, assuming that $n > m$. This applies to the model-building part of the process, so it is only performed once. For the classification part, see section 4.8 below.
4.7.2 Rule approximation

Although LEM2 tries to deal with overfitting by reducing the conditions it builds, sometimes the final set of rules is overly specific to the training set. A greedy algorithm is used for building our rule conditions and once such a set is obtained, we only prune our set (lines 12-14 in algorithm 4.7.1) by removing attribute-value pairs that let the rules stay 100% consistent with the input. Also, when we decide on removing rules (or conditions) from the entire set Covering, we only remove those that keep the equivalence class of Covering ([Covering]_U) absolutely intact. This requirement, along with the fact that the training set may be but a small subset of the entire dataset, can lead to the generation of an overfitted set of rules. It can either be that each rule contains attribute-value pairs that only contribute by excluding a single object from the set that matches the rule, or that some rules match approximately the same sets of objects. By letting the ratio of the number of supporting to the number of matching objects of a rule (see section 3.3.4) be less than 1, it is hoped that the rules will be slightly better at classifying unseen objects. The other way of generalizing a rule set, by removing entire rules, is far more risky and was proven not to be fruitful during an initial round of tests of various approximation techniques.

Rule approximation techniques have been well studied and a good summary can be seen in [˚Ago99]. We will use a standard version of rule approximation that works independently of the rule-producing algorithm, and the method is described in algorithm 4.7.3. We remove attribute-value pairs in each rule while the quotient between the number of supporting objects and the number of matching objects is higher than some threshold value. We will later when referring to LEM2 actually use LEM2 accompanied with this algorithm for rule reduction, and with ε set to 0.8\(^{13}\).

Some supporting functions that we use:

- **Matching** : \( \bigcup_{d_i \in V_d} \text{Rules}_{d_i} \rightarrow \mathbf{P}(U) \) returns those \( u \in U \) (from the decision system \( \langle U, A \cup \{ d \} \rangle \) that rule \( r \in \text{Rules}_{d_i} \) was constructed from) that match the conditions in \( r \). We will use subscripts to clarify which decision system we are working with.

\(^{13}\)based on empirical studies
Algorithm 4.7.3 Rule approximation algorithm

\textbf{STABILITY}(\textit{Rule})
1 \textbf{return} \frac{|\text{SUPPORTING}(\textit{Rule})|}{|\text{MATCHING}(\textit{Rule})|}

\textbf{APPROXIMATE RULES}(R, \varepsilon)
1 \textbf{for each} \textit{Rules}_{di} \in R \\
2 \quad \textbf{do for each} \langle \textit{Covering}, \textit{Class} \rangle \in \textit{Rules}_{di} \\
3 \quad \quad \textbf{do for each} \textit{Condition} \in \textit{Covering} \\
4 \quad \quad \quad \textbf{do for each} \textit{t} \in \textit{Condition} \\
5 \quad \quad \quad \quad \textbf{do if} \textit{STABILITY}(\textit{Condition} \setminus \{t\}) \geq 1 - \varepsilon \\
6 \quad \quad \quad \quad \textbf{then} \textit{Condition} \leftarrow \textit{Condition} \setminus \{t\} \\
7 \textbf{return} R

- \textbf{SUPPORTING} : \bigcup_{d_i \in V_d} \textit{Rules}_{d_i} \mapsto \textbf{P}(\textit{U}) is a function that returns the objects (from the decision system \langle \textit{U}, \textit{A} \cup \{d\} \rangle that rule \textit{r} \in \textit{Rules}_{d_i} was constructed from) that match \textit{r} and have the same decision value as \textit{r}.

\textbf{Computational complexity}

Given that the rules are produced by the LEM2 algorithm (algorithm 4.7.1) we know two things:

- The maximal size of a rule is \textit{m} since it can contain at most \textit{m} attribute-value pairs.

- There can be at most \textit{n} rules since each rule will cover at least one object.

Computing the stability of a rule can be done in \textit{O(n)} time and thus, the complexity of this rule approximation technique is \textit{O(m \cdot n^2)}. 
4.8 Classification

Here we shall describe the algorithms used for classifying objects once the models are built (see table 4.3). First, we describe an algorithm that does not use an internal model to classify objects, but rather relies on the information available in the entire training set (algorithm 4.8.1). After that, we demonstrate a number of techniques for negotiating among the rules created by LEM2 (algorithm 4.7.2) when the rules are to be used to classify unseen examples.

4.8.1 The Voter

The Voter-algorithm is implemented just to see if rule construction is of importance or not in the Dynamic Approach. The algorithm is also used in [Baz98] and therefore considered relevant as a point of reference.

In short, the idea with algorithm 4.8.1 is to find those objects (line 6) that have at least one attribute value in common with the one to be classified. AL will thus hold the set of pairs \( \{(a,v) : a \in A, v \in V_a \} \) such that \( a(u_i) = v \land a(u_t) = v \) for every object \( u_i \) during each iteration (lines 4-16) (\( u_t \) is the object to be classified). EQUAL returns the set of attributes with equal values in both the objects given as arguments. For each object that partially matches, we scan the set of objects again and for every additional partial match that is a more or equally exact match than AL (line 10). Then, we divide the number of such objects by the total number of objects that are in the decision class of \( u_i \) to obtain the weight used when voting\(^{14}\).

The method tends to give high weights to objects that match the tested object by the attributes in a relative reduct of \( S \). That is, if \( u_i \) has the same values of \( \{a_1, a_2 \ldots a_l\} \) as \( u_t \) and \( \{a_1, a_2 \ldots a_l\} \) partitions the table into classes in which all objects have the same decision, then the decision of \( u_i \) is likely to be chosen winner. While fulfilling the condition of partitioning so that each partition has only (approximately) one decision value, the less attributes that are matched by \( u_i \), the more objects could be expected to be included in \( OL \) (line 13) implying a higher ratio \( \frac{|OL|}{n} \) and that

\(^{14}\)The constant \( \alpha \) in algorithm 4.8.1 is used as a “cut-off” constant for the ratio \( \frac{|OL|}{n} \). If the ratio is less than \( \alpha \), we do not consider the currently matching object \( u_i \).
Algorithm 4.8.1 The Voter

\textbf{VOTER}(S, u_t, \alpha)

1. for \( i \leftarrow 1 \) to \( n \)
2. \hspace{1em} do \( OA[i] \leftarrow 0 \)
3. for \( i \leftarrow 1 \) to \( n \)
4. \hspace{1em} do if \( OA[i] = 0 \)
5. \hspace{2em} then \( AL \leftarrow \text{EQUAL}(u_t, u_i) \)
6. \hspace{2em} if \( |AL| > 0 \)
7. \hspace{3em} then \( OL \leftarrow \emptyset \)
8. \hspace{2em} count \leftarrow 0
9. for \( j \leftarrow 1 \) to \( n \)
10. \hspace{2em} do if \( AL \subseteq \text{EQUAL}(u_t, u_j) \)
11. \hspace{3em} then \( count \leftarrow count + 1 \)
12. \hspace{3em} if \( d(u_i) = d(u_j) \)
13. \hspace{4em} then \( OL \leftarrow OL \cup \{u_j\} \)
14. \hspace{2em} if \( \frac{|OL|}{count} \geq \alpha \)
15. \hspace{3em} then for each \( u_i \in OL \)
16. \hspace{4em} do \( OA[i] \leftarrow 1 \)
17. \( u^* \leftarrow \text{argmax}_{u_i \in U} \frac{|\{u_j \in U: d(u_j) = d(u_i) \land OA[j] = 1\}|}{|\{u_j \in U: d(u_j) = d(u_i)\}|} \)
18. \( d^* \leftarrow d(u^*) \)
19. return \( d^* \)

\( AL \in \text{RED}(S, d) \). In the end, we choose the decision value of the object \( u_i \) in the decision table with the highest weight (line 18). If there are several objects with the same value of the expression at line 18, we choose one by random.

**Computational complexity**

The Voter builds no model, so algorithm 4.8.1 is instead the one used for classifying objects and it is thus called \( O(n) \) times. There are two major loops, namely lines 4-16 and 10-13. Both loops are performed \( n \) times, which gives us a computational complexity for the classification process of
4.8. Classification

4.8.2 Rule negotiations

When using the rule set produced by LEM2, each object is matched against the rules and sometimes it matches several of them. In such a case, we have to use some kind of heuristics to determine what rule to trigger, a technique that is called rule negotiation. Since the only ambiguity that it is interesting to solve is the one that arises when an unseen object matches decision rules with different decision values and therefore all algorithms calculate the strengths of sets of rules, where each set of rules is supposed to imply the same decision value.

Regarding notation, we will in addition to Matching and Supporting (see section 4.7.2) use two functions in these algorithms:

- **Decision**: $\text{Rules} \mapsto V_d$ takes as input a $\text{RuleSet}$ (see section 3.3.4) and returns the decision value.

- **MatchingRules**: $(\text{Rules} \times U) \mapsto \text{Rules}$ is a function that returns a $\text{RuleSet}$ matching an object from a $\text{RuleSet}$ and an object $u \in U$.

The algorithms that generate rules are assumed, without loss of generality, to generate a set of sets of rules $R$ where $|R| = |d|$ and every element $R_i \in R$ is a set of rules associated with a certain decision class $d_i$, just as defined in section 3.3.4.

4.8.3 Basic negotiation methods

These two methods for calculating the strength of a rule, Simple Strength (algorithm 4.8.2) and Maximal Strength (algorithm 4.8.3) need no further explanation. They simply return the quotient of two simple expressions.

**Computational complexity**

Just like the Voter algorithm, these two algorithms are run $O(n)$ times during each evaluation process. Computing the Simple Strength requires $O(n^3)$. 
Algorithm 4.8.2 Simple rule strength

\[ \text{SIMPLEVALUE}(\text{Rules}_{d_i}, o) \]
\[ \begin{align*}
1 & \quad \text{return} \frac{|\text{MATCHINGRULES}(\text{Rules}_{d_i}, o)|}{|\text{Rules}_{d_i}|} \\
\end{align*} \]

\[ \text{SIMPLE STRENGTH}(S, R, o) \]
\[ \begin{align*}
1 & \quad \text{BestClass} \leftarrow \arg\max_{r \in R} \text{SIMPLEVALUE}(\text{Rules}_{d_i}, o) \\
2 & \quad d^* \leftarrow \text{DECISION}(\text{BestClass}) \\
3 & \quad \text{return} \; d^* \\
\end{align*} \]

Algorithm 4.8.3 Maximal rule strength

\[ \text{MAXVALUE}(\text{Rules}_{d_i}, o) \]
\[ \begin{align*}
1 & \quad \text{return} \max_{r \in \text{MATCHINGRULES}(\text{Rules}_{d_i}, o)} \frac{|\text{SUPPORTING}(r)|}{|\{u \in U : d(u) = \text{DECISION}(\text{Rules}_{d_i})\}|} \\
\end{align*} \]

\[ \text{MAXIMAL STRENGTH}(S, R, o) \]
\[ \begin{align*}
1 & \quad \text{BestClass} \leftarrow \arg\max_{\text{Rules}_{d_i} \in R} \text{MAXVALUE}(\text{Rules}_{d_i}, o) \\
2 & \quad d^* \leftarrow \text{DECISION}(\text{BestClass}) \\
3 & \quad \text{return} \; d^* \\
\end{align*} \]
the computation of \texttt{SIMPLEVALUE} for each decision class, which takes $O(n)$ time by performing a simple linear search. However, the space requirements of $R$ is linear w.r.t. $n$ since there not more rules than objects, so in total, the algorithm will only require $O(n)$ time. The whole classification process, using $k \cdot n$ testing objects ($k$ constant), would thus require $O(n^2)$ time. The \textsc{Maximal Strength} computation requires iteration through the whole table $S$ for each class, so this algorithm requires $O(|d| \cdot n)$ time. However, assuming that the process is initialized by calculating the number of supporting objects for each rule and the number of objects in each decision class, the computation would be reduced for each tested object to $O(n)$ time. Thus, there would be a total time complexity of $O(n^2)$ for the whole classification process using the \textsc{Maximal Strength} measure.

\subsection*{4.8.4 Stability strength}

The stability strength method is the second part of the dynamic approach, apart from the dynamic discretization, that uses ideas from the theory of dynamic reducts. However, there is no direct connection between this method and the concept of dynamic reducts, but it comes from the same article ([Baz98]). The reason for developing the stability strength measure for rules was that in analogy to reducts, rules that are applicable not only to the whole training set but also to a family of subsets of that set can be considered as more \textit{stable} than others. However, the expressions used here to calculate rule stability do not reveal this idea, since this is only an approximation to a more exact (and much more computationally expensive) method for calculating stability for rules (see [Baz98] for details).

In the stability strength calculation, we will use $dc^r_i$ to indicate how many objects that differ only by attribute $i$ from matching a rule $r$. That is,

$$dc^r_i = |\{u \in U : (\forall j \neq ia_j(u) = v^r_j) \land a_i(u) \neq v^r_i \land d(u) \neq d^r\}|$$

$v^r_i$ means the value of attribute $a_i$ in rule $r$. As a special case, $-1$ will denote the index of the decision attribute (see definition 13).
Algorithm 4.8.4 Stability rule strength

\textbf{Stability Strength}(R,o)

1. \textit{BestClass} $\leftarrow \text{argmax}_{\text{Rules}_{d_i} \in R}(\max_{r \in \text{Matching Rules}(\text{Rules}_{d_i}, o)} SC(r))$

2. \textit{d} $\ast$ $\leftarrow \text{Decision}(\text{BestClass})$

3. return \textit{d} $\ast$

\begin{definition} (Difference Coefficient)
\[ dc_1 = |\{u \in U : (\forall i \in \{1...m\} a_i(u) = v_i) \land d(u) \neq v_d\}| \]
\end{definition}

Definition 14 below has been used to implement what Bazan calls stability strength for rules. Please note that this stability coefficient is different from definition 10, stability coefficient for reducts.

\begin{definition} (Stability Coefficient for rules)
\[ SC(r) = \prod_{i=1}^{m} \left( 2^{dc_i} - 1 \right) \]
\[ \frac{dc_1 + \sum_{i=1}^{m} dc_i}{2} \]
\end{definition}

Now we are ready to define the function that calculates the stability strength for each Rule Set$^{15}$ Rules$_{d_i} \in$ Rules - algorithm 4.8.4.

\textbf{Computational complexity}

The stability coefficient for each rule can be calculated by iterating through the whole table for each attribute, giving a time complexity of $O(n \cdot m)$. This calculation need only be done once, since it does not involve any objects from the testing set. Thus, there would also here only have to be a linear search for the best rule for each object, giving a time complexity of $O(n^2)$.

$^{15}$see section 3.3.4
4.9 Remarks on computational complexity

Here we will present a summary of the complexity analyses performed in this chapter.

4.9.1 LEM2 time complexity

The LEM2 Approach will have a total time complexity of \(O(m^2 \cdot n^2) + O(|d| \cdot n^3) + O(m \cdot n^2) + O(n^2)\), for the discretization, rule generation, approximation and negotiation, respectively. In total, we can assert that the upper bound on this computation will be \(O(|d| \cdot n^3)\) as determined by the core LEM2 algorithm (algorithm 4.7.2).

4.9.2 DR time complexity

The upper bound on the time complexity of the Dynamic Reduct Approach later used in the evaluation is the greater of the two expressions \(O(m^2 \cdot n^2 \cdot \log_2 n) + O(n^3)\) and \(O(m^2 \cdot n^2 \cdot \log_2 n) + O(|d| \cdot n^3)\) corresponding to dynamic reduct discretization (see section 4.6.4) combined with the Voter algorithm and LEM2, respectively. Since \(O(|d| \cdot n^3)\) clearly subsumes \(O(n^3)\), we can determine that \(O(m^2 \cdot n^2 \cdot \log_2 n) + O(|d| \cdot n^3)\) is the better bound on the time complexity of the Dynamic Approach.

4.9.3 Computational feasibility

From the derived expressions in sections 4.9.1 and 4.9.2, we can determine that the LEM2 Approach has a tighter complexity bound on its computation than the Dynamic Approach, given that \(|d| \cdot n < m^2 \cdot \log_2 n\). This is a fairly reasonable assumption, especially since the expressions determining the complexity bound on the Dynamic Approach has many potentially large hidden constants. Since this is an empirical study, we will not use these results any further.
4.10 Further Reading

As supplementary reading to this chapter, I first and foremost recommend Son Nguyen’s article about the implementation of Rough Set algorithms in [NN96], but also the rule negotiation part (section 15 in [BNN+00]) could be of general interest.

Now that the parts of the implementation that are central in this work have been studied, the following chapters will deal with how these algorithms were fitted into the Weka framework, how they were evaluated and what the results were from those evaluations.
Chapter 5

Implementation

In this chapter, the implementation of the algorithms in chapter 4 will be described. Much of the work has focused on constructing a Rough Set framework that can be used for implementing various Rough Set algorithms, of which only parts were actually used in the comparison of the two approaches. If not stated otherwise, all of these classes are implemented in the Java package \texttt{weka.roughset}.

All in all, the implementation amounts to about 8000 lines of code. Not all methods or attributes implemented in the classes are listed in the UML-diagrams below, but the most important ones are. For more information about the details, the reader is referred to the Javadoc online documentation available at \url{http://www.ida.liu.se/~olale/exjobb/weka/weka-3-2/doc/}.

5.1 Implementation philosophy

During the implementation, the main focus has been to build well-structured and readable code that can be used and modified by others. No special considerations were made regarding speed or memory efficiency. This was made in accordance to the general goal of this thesis work - to produce fair implementations from the published descriptions of the algorithms and
compare them. However, an initial study was conducted (see section 5.2) just to establish what kind of performance loss one could expect if standard Java components were used without any optimizations, though it had no impact on the decision to use standard components.

For each concept in the original articles such as cutpoints, reducts, local coverings and minimal complexes, new classes were implemented and methods added to handle objects of these types. This type of modularization may have introduced additional overhead that didn’t exist in the original implementations. Hopefully however, the level of abstraction in the current implementation may facilitate reading the code and improving it.

5.2 Fast vectors

The standard Weka Instances class (for manipulating information systems in the form of tables), as well as other Weka classes, uses a custom-made data structure FastVector. This datatype is a fast, non-synchronized vector but it does not support standard interfaces that are convenient when manipulating data with standard algorithms. It was considered desirable to have a solution that used standard components as much as possible for data manipulation, since it generally allows for better solutions and more readable code. However, since learning tasks can be computationally very expensive, details such as which datastructures to use cannot be ignored and it was not clear what the exact costs for using standard components compared to custom-made ones would be. Therefore, some initial tests were run to see whether standard Java classes for storing data could be used.

In table 5.1 a small comparison is shown between two datatypes, where the numbers indicates the time it takes to sort sets of integers in ascending order, using either a custom-made Quicksort implementation or the standard java.Collections.sort-method on the FastVector or Vector classes respectively. We can see that the Vector class combined with the standard sorting algorithm is inferior to the custom-made datatype for datasets with up to one million elements, though the advantage is mostly through a constant overhead that is of lesser importance in large datasets. However, there is still a significant performance gain in choosing the customized data-
<table>
<thead>
<tr>
<th># elements</th>
<th>FastVector</th>
<th>Vector</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.0030</td>
<td>0.012</td>
<td>0.25</td>
</tr>
<tr>
<td>1000</td>
<td>0.01</td>
<td>0.034</td>
<td>0.29</td>
</tr>
<tr>
<td>10000</td>
<td>0.02</td>
<td>0.049</td>
<td>0.41</td>
</tr>
<tr>
<td>100000</td>
<td>0.378</td>
<td>0.541</td>
<td>0.70</td>
</tr>
<tr>
<td>1000000</td>
<td>5.749</td>
<td>7.357</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Table 5.1: FastVector analysis

type, though it comes at the expense of not being able to benefit from any of the future work by the Java development community on the standard components of the JDK.

Despite these results, it was decided to use the standard Java components since making readable and simple solutions was considered more valuable than fast code. However, as indicated in the test results presented in chapter 6, the algorithms are somewhat slower.

5.3 Core datastructures

The basic datastructures that were implemented in Weka were the classes RoughInstance and RoughInstances. In picture 5.1 we can see that the RoughInstance and RoughInstances classes inherit from Instance and Instances respectively. The need for these classes arise when working with discernibility. We want to compare only the values of a subset of the attributes (a possible reduct for example) on some objects, something that is not possible using the original datastructures and not easily added to them. As a test for equality, only a set of “active attributes” with or without the decision attribute are compared between two instances in our classes. This way, the data can be divided into a single equivalence class \([u]_B, B = \emptyset\) which could then successively be split into smaller ones by adding \(a \in A \setminus B\) to \(B\), thus creating smaller and smaller equivalence classes until some condition is met, such as that all classes are uniform with respect to the values of the decision attribute. This happens to be one of the uses for this class (along with others) in the implementation of algorithm 4.6.3,
global discretization.

5.4 Decision Rules

The diagram in figure 5.2 shows the classes involved in representing the generated set of rules. Actually, the LEM2 algorithm is implemented as a `addRules( ... )`-method in the class `Rules`.

As depicted in figure 5.2, we create a new set of rules for each decision value. Every rule has methods for calculating it’s strength by using some of the methods described in section 4.8.2, since it has access to the dataset it was constructed for.

The supporting classes `AttributeValuePair` and `AttributeValuePairs` implements the concepts of attribute-value pair and sets of such pairs, respectively.
5.5 Cutpoints

All discretization techniques use the MDCutPoints class to represent the set of cuts which in turn contains CutPoint objects. Algorithm 4.6.1\(^1\) is implemented in the updateValues() -methods of the CutPoint-class, which is called by the constructors of the class. Which one that is chosen is based on whether or not we construct the first cutpoint on an attribute or not.

By default, it is assumed that each new CutPoint has come arbitrary

\(^1\)Number of pairs of objects with different decision values separated by a cut, see section 4.6.1
5.6 Rough Set functions

Many of the algorithms in chapter 4 work with data that is or becomes partitioned in equivalence classes. The class `SortedClasses` implements the concept of equivalence classes as introduced by the \( \text{IND}_S \)-relation and contains methods for retrieving the attribute discerning most pairs of objects with different decision values in these classes (used for constructing reducts for instance) and splitting the classes by using a cutpoint, for example. This class was also used in the `RoughSet` class which implements all the basic Rough Set functionalities such as computing lower and upper approximations to a concept and the size of a positive region. It also contains functions pertaining to the Dynamic Approach, such as calculating the minimal size of a subtable family (see statement 2 in the description of stability calculation in section 4.3.2).
5.7 Reduct computation

Both methods for computing a single reduct as well as a set of reducts from a super-reduct (algorithm 4.3.1) were implemented in RelativeReductComputation. As supporting datastructures, the Reduct class holds information about a set of attributes and a set of objects the attributes are associated with, and the class Reducts contains sets of such objects.

5.8 Model construction

In the weka.classifiers-package, a generic RuleClassifier with the possibility to approximate rules and handle generic options for a rule-based classification algorithm has been added. Also, LEM2 and Voter, implementing algorithms 4.7.2 and 4.8.1 respectively are placed in this package.

5.9 Filters

In package weka.filters, RSDiscretize implements algorithms 4.6.2 and 4.6.3, and DynamicReductDiscretize implements algorithm 4.6.4.
Chapter 6

Testing

This chapter presents the evaluation results of the new implementations of the algorithms described in chapter 4 gathered during a testing phase. In addition, we compare the performance of the new implementations with the implementations of the original authors. Our original hypothesis was that the tendencies of our results\footnote{The relations between the performance of one algorithm and another, such as that they achieve similar results} should not differ too much from the original results in [Baz98] since we use similar setups of algorithms and exactly the same datasets.

6.1 Comparing learning algorithms

We want to compare the performance of these algorithms on some datasets and see if we can establish the same tendencies as in [Baz98]. We will satisfy ourselves with noting whether or not the same differences, between the performance of the LEM2 approach and the Dynamic Approach, appear in this batch of tests as previously. Statistically verifying the difference has been done using a Student’s t-test\footnote{The t-test is a standard statistical method used to assess whether the means of two sets of datapoints are statistically different from each other} (see [WF00]). We use a table of
confidence limits for the Student’s Distribution with 9 degrees of freedom, corresponding to the fact that we will be using results that are the average of ten cross-validations. The t-test uses the mean values and variances (of the correctness values) of both approaches on the datasets. Since the two test series we will compare come from independent cross-validations, we cannot use the \textit{paired} version of the t-test, but only the non-paired. The main equation used in the non-paired t-test is as follows:

\[
    t = \frac{\bar{d}}{\sqrt{\sigma_d^2/k}}
\]

We use the variables $\bar{d}$, $\sigma_d^2$ and $k$ for the mean difference value, the variance of the difference and the number of tests conducted, respectively. The variance of the difference for a non-paired t-test can be approximated by simply adding the variances of the variables to be compared and dividing the sum with the number of tests (10 in this case) if the variables come from tests with equal amounts of values in them. This was the way the difference variance was obtained below, though the individual numbers are not shown.

\subsection{6.1.1 Significance}

We say that one learning approach is better than the other if we can set a 95\% confidence limit on the comparison of the results from the learning approaches, according to the Student’s t-test.

\section{6.2 The datasets}

The datasets in table 6.1 were used for testing the algorithms. The table lists how many attributes occur in the dataset (decision value excluded), the number of those that are numerical, how many attribute values that are missing in the dataset, the number of possible values for the decision attribute and the number of instances in the table. We will briefly describe each one of the sets.
First we have the Breast Cancer dataset. This has been used as a benchmarking dataset for machine learning algorithms ever since it was first released in 1988. The data represents information about patients that have been treated for breast cancer. Typically, the use of this dataset is to learn what information that can be useful when trying to determine if a patient's cancer will return or not. Some of the attributes represent intervals of numerical values, namely the age of the patients, the size of the tumors, and the number of inverted nodes. These attributes\(^3\) were considered well suited for discretization after being made into numerical attributes. In this way we could reduce the number of intervals and adapt them to the given classification of the patients. Another reason for choosing this dataset is that it was also chosen by the authors of the two approaches that were supposed to be evaluated in this work.

This set is similar to Breast Cancer and was obtained from the same source. It is related to medical data from lymphography tests taken at the same hospital. However, some of the attributes are numeric and it has more decision classes which makes it a bit more interesting. This dataset was also used by both the authors of our machine learning algorithms.

The Balance Scale data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight and

\(^3\)As dataset breast-cancer.arff from UCI [BM98], these attributes have indices 1, 3 and 4.
the right distance. The correct way to find the class is the greater of \((\text{left-distance} \times \text{left-weight})\) and \((\text{right-distance} \times \text{right-weight})\). If they are equal, it is balanced. The set is fairly large, but contains rather few attributes, as opposed to the lymphography dataset. Also, since the true model is a relation between attributes, it could be interesting to see how well the generated set of rules from the different algorithms can approximate that model since none of them can express the desired relation exactly as explained in section 2.4.

**Zoo** Last but not least is my personal favorite\(^4\) - Zoo. Zoo contains information about animals at a zoo, one from each species. Most of the attributes are boolean and relate to questions like whether or not the animal has feathers, is aquatic, produces milk or so. The idea is that the classification is supposed determine which family\(^5\) an animal is a member of. It is fairly easy to assess the quality of the rules produced by merely observing them, since all the attributes relate to common knowledge about animals that anyone could understand.

### 6.2.1 Dataset origins

All datasets come from the UCI Machine Learning Repository [BM98].

### 6.3 Experiment setup

All testing was conducted using the **Experimenter** environment in Weka which supports the use of a N-times M-fold cross-validation procedure. However, there is no way to obtain information about the models that the algorithms produce, so all tests were first performed using the **Explorer**, which cannot perform full-scale experiments but merely single runs of algorithms but allows for displaying the set of rules produced. The sizes of the produced rule sets below come from the rule sets constructed on the whole input dataset, not on the subsets later used for training.

\(^4\)My favorite mostly because the rules produced by the algorithms can be quite hilarious, such as that all animals that lack teeth and are predators are mammals :)  
\(^5\)i.e. mammals, fish or amphibians
Since the experiments were made with so many parameters, we will try to explain them with table 6.2 in an effort to make it easier to obtain an overview.

As one can deduce from tables 4.2, 4.3 and 6.2, the combinations \([R]^6\) \([N](D|F)^7\)[LS]V make up the Dynamic Approach and \([R]([N](M|F))L(I|X)\) make up the LEM2 approach\(^8\). That is, both approaches use the same method for replacing missing values, unless there are none ([R]). Then they may use discretization on all numeric and possibly some nominal (N) attributes with some of the discretization methods labelled D,M or F, if discretization is to be performed at all (hence the brackets). The model-constructing phase corresponds to the options L or V. If V (the Voter) is used (in the Dynamic Approach) no rule negotiation method is needed since no rules are produced. If the L (LEM2) option is used then we must use a rule negotiation method along with it, either I or X.

The two approaches implemented in this work were also compared to two related algorithms already available in Weka, namely Prism, or \([R][N]F\)P for short, and DecisionTable, \([R][N]F\)T.

### 6.3.1 System setup

All experiments were carried out on Sun Microsystems Ultra 10\(^{TM}\) workstations, equipped with 256 MiB of SDRAM, a 440 MHz Ultra SPARC IIi processor and the Sun OS 5.8 operating system. The code was compiled with IBM’s Java-compiler Jikes [CS02] against version 1.4.0 of the Java runtime environment JRE [Mic02] and version 3.2 of the Weka framework.

### 6.4 Evaluation

In section 2.7.4 we discussed in general terms the methods for evaluating learning algorithms. However, here we add some additional comments regarding the methodology used during this particular testing phase.

\(^6\) [item ] = item is optional  
\(^7\)( item1 | item2 ) = select item1 or item2, exclusively  
\(^8\) As you can see, LEM2 is used in both approaches. However, the pure LEM2 approach uses no dynamic discretization but rather the standard Rough Set algorithm based on the discernibility heuristic
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>RSReplaceMissingValues, replace unknown values (see section 4.5)</td>
</tr>
<tr>
<td>N</td>
<td>RSNumerizeNominal, make nominal attributes numeric (see section 4.4)</td>
</tr>
<tr>
<td>D</td>
<td>DynamicReductDiscretize, discretize numerical attributes (algorithm 4.6.4)</td>
</tr>
<tr>
<td>M</td>
<td>RSDiscretize, discretize numerical attributes (algorithm 4.6.3)</td>
</tr>
<tr>
<td>F</td>
<td>DiscretizeFilter, discretize numerical attributes (Fayyad-Irani’s method, see [FI93])</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>DecisionTable (see description in section 2.7.3)</td>
</tr>
<tr>
<td>P</td>
<td>Prism (see description in section 2.7.3)</td>
</tr>
<tr>
<td>L</td>
<td>LEM2 (algorithm 4.7.2). Reduce the rules using rule approximation option -A 0.8</td>
</tr>
<tr>
<td>V</td>
<td>Voter (algorithm 4.8.1)</td>
</tr>
<tr>
<td>S</td>
<td>Stability Strength (algorithm 4.8.4)</td>
</tr>
<tr>
<td>I</td>
<td>Simple Strength (algorithm 4.8.2)</td>
</tr>
<tr>
<td>X</td>
<td>Maximal Strength (algorithm 4.8.3)</td>
</tr>
</tbody>
</table>

Table 6.2: Symbols describing the test setup
Usually, when evaluating an algorithm some common heuristics are used to obtain precision and error rates that are more statistically reliable than if only a single cross-validation is used (see [WF00]). Below are listed three common practices for evaluation followed in our tests.

**Ten-fold cross-validation** N-fold cross-validation is the standard method to evaluate learners with only a single dataset. In practice, the number of iterations is set to ten, as it is a good tradeoff between precision and time-consumption.

**Stratification** When dividing the data into folds, or subsets, each fold may or may not have objects from a given decision class. To ensure that the algorithm has at least some chance of classifying instances correctly from the models produced on each fold, the process of stratification guarantees that each decision class is properly represented in both the training and testing sets the algorithm is given. Weka offers this possibility and it was used during the testing.

**Runs** The results from running two ten-fold cross-validations with the same algorithm on the same datasets can of course differ because of the random variation in the selection of folds. To even out these differences, evaluation through ten-fold cross-validation of a classification algorithm is commonly performed ten times and the average results are used.

### 6.4.1 Relaxation

Three relaxations were introduced on the dynamic reduct discretization, due to the computational complexity of the process. Before these relaxations were made, the dynamic reduct discretization took more than 70 hours to complete even one batch of ten times ten-fold cross-validations on the Breast Cancer dataset. Thus they were considered necessary.

First of all the calculation of minimal subtable\(^9\) size prior to the discretization was not conducted every time the algorithm was run. In table 6.3,\(^9\)

---

\(^9\)In analogy to the concept of subsystem (see section 3.4.2), a subtable is a subset of the rows in a table. The table is here supposed to correspond to a decision system.
we can see the probabilities for choosing \( k \) objects out of 20. Naturally, the probability of choosing a subtable that is approximately half the size of the original table is most likely since there are by far most combinations of that size. With this information, the decision was made to simply say that a certain percentage of the table would be the least allowed subtable size. In practice, this means that most generated subtables will either be half the size of the original table, with the approximate probabilities for selecting each size given in table 6.3, or the size we supply, if that size is greater than half the original table size. There are no theoretical underpinnings to support the idea that a certain lower limit on the minimal subtable size would be better than another, though empirical studies have indicated it in certain cases\(^{10}\). Moreover, the process of finding the minimal subtable size\(^{11}\) is computationally very expensive. Those two facts together motivated the dismissal of calculating minimal subtable size every time. Just as the parameter \( \varepsilon \) (used for choosing approximation level of rules) was set to a fixed value before the tests (see section 4.7.2), so was the parameter deciding the minimal size of the subset of the original table that would be used during stability calculation. Initial tests were made with different minimal subtable sizes and it was found that 0.7 times the size of the size of the original table should be used as the minimal subtable size, for the datasets at hand.

Secondly, it was decided that 20 proper reducts should be the maximum number of reducts generated out of each super-reduct in algorithm 4.3.1. This number was chosen after thorough tests on the datasets at hand. This may not be considered so much of a relaxation as setting a parameter to a fixed value though.

Finally, only a fix number of subtables were used during the stability calculation instead of the number of subtables statistically required (see section 4.3.2). This relaxation was discussed with the author and he verified that it often made little difference in practice to make use of the large number of statistically required subtables.

\(^{10}\)according to personal conversations with Bazan  
\(^{11}\)by performing a binary search to evaluate different sizes and settle for one where many enough of the reducts on decision systems of that size are also reducts of the original system
<table>
<thead>
<tr>
<th>Subtable size</th>
<th>Probability (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.002</td>
</tr>
<tr>
<td>2</td>
<td>0.018</td>
</tr>
<tr>
<td>3</td>
<td>0.109</td>
</tr>
<tr>
<td>4</td>
<td>0.462</td>
</tr>
<tr>
<td>5</td>
<td>1.479</td>
</tr>
<tr>
<td>6</td>
<td>3.696</td>
</tr>
<tr>
<td>7</td>
<td>7.393</td>
</tr>
<tr>
<td>8</td>
<td>12.014</td>
</tr>
<tr>
<td>9</td>
<td>16.018</td>
</tr>
<tr>
<td>10</td>
<td>17.620</td>
</tr>
<tr>
<td>11</td>
<td>16.018</td>
</tr>
<tr>
<td>12</td>
<td>12.014</td>
</tr>
<tr>
<td>13</td>
<td>7.393</td>
</tr>
<tr>
<td>14</td>
<td>3.696</td>
</tr>
<tr>
<td>15</td>
<td>1.479</td>
</tr>
<tr>
<td>16</td>
<td>0.462</td>
</tr>
<tr>
<td>17</td>
<td>0.109</td>
</tr>
<tr>
<td>18</td>
<td>0.018</td>
</tr>
<tr>
<td>19</td>
<td>0.002</td>
</tr>
<tr>
<td>20</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

Table 6.3: Probabilities of choosing a subtable of a certain size
6.5 Results

In this section the results that were obtained during the testing of each algorithm will be presented, sorted on the datasets. Each table of results will consist of four columns, describing four characteristics of each sub-experiment.

The first column (Scheme) describes the combination of algorithms used. Refer to table 6.2 to see what each letter means. Next comes the time (Time) taken to build a model of the whole dataset, in seconds. Note that this model is not used during testing, where only subsets of the original set were used for building models. The third column (Rules) tells us the number of rules produced when constructing a model of the whole dataset, if applicable. Finally we have the column (Correct) which was used as the performance measure - the fraction of objects in the testing set that were correctly classified using the model created. The best results in each category are displayed in bold letters. Those objects that were not correctly classified were classified incorrectly or not at all, depending on the algorithms.

6.5.1 Breast Cancer results

Table 6.4 shows the results of running the various schemes on the breast cancer dataset. Interesting to note is that the DecisionTable classification algorithm seems to work best of all, by using only 5 rules compared to the best results in the LEM2 approach which produced 23 times as many rules. Also, it is quite clear that the LEM2\(^{12}\) and RSDiscretize\(^{13}\) implementations are not the most efficient, taking up to 80 times longer to finish compared to the Weka schemes tested.

Statistical results

We could not establish any statistically significant difference between the LEM2 Approach and the Dynamic Approach. However, the difference

\(^{12}\)Pattern *L*
\(^{13}\)Pattern *M*
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Time</th>
<th>Rules</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weka</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNFT</td>
<td>.4</td>
<td>5</td>
<td>.7297</td>
</tr>
<tr>
<td>RNFP</td>
<td>.74</td>
<td>131</td>
<td>.6724</td>
</tr>
<tr>
<td>The LEM2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLI</td>
<td>64.54</td>
<td>189</td>
<td>.6389</td>
</tr>
<tr>
<td>RLX</td>
<td>71.61</td>
<td>189</td>
<td>.6529</td>
</tr>
<tr>
<td>RNMLI</td>
<td>71.48</td>
<td>167</td>
<td>.6321</td>
</tr>
<tr>
<td>RNMLX</td>
<td>69.24</td>
<td>167</td>
<td>.6439</td>
</tr>
<tr>
<td>RNFLI</td>
<td>36.56</td>
<td>117</td>
<td>.6029</td>
</tr>
<tr>
<td>RNFLX</td>
<td>27.41</td>
<td>117</td>
<td>.68</td>
</tr>
<tr>
<td>The Dynamic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLS</td>
<td>58.49</td>
<td>189</td>
<td>.6522</td>
</tr>
<tr>
<td>RNDLS</td>
<td>182.08</td>
<td>164</td>
<td>.667</td>
</tr>
<tr>
<td>RNFLS</td>
<td>42.82</td>
<td>117</td>
<td>.6902</td>
</tr>
<tr>
<td>RNDV</td>
<td>60.78</td>
<td>N/A</td>
<td>.6641</td>
</tr>
</tbody>
</table>

Table 6.4: Breast Cancer results

between the Dynamic Approach and the DecisionTable\textsuperscript{14} algorithm was significant.

### 6.5.2 Lymphography results

In table 6.5 we can see some indication that for the lymphography dataset, the LEM2 approach is better than DecisionTable, when coupled with the standard DiscretizeFilter\textsuperscript{15} discretization algorithm. The stability strength rule negotiation method alone does not improve the results of the LEM2 algorithm however.

**Statistical results**

When conducting the t-test it showed no significant differences between the Dynamic Approach and the LEM2 Approach. There is not even a significant difference between the best of the two approaches and the available Weka algorithms.

\textsuperscript{14}Pattern *T

\textsuperscript{15}Pattern *F*
6.5. Results

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Time</th>
<th>Rules</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weka Algorithms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>.65</td>
<td>44</td>
<td>.7493</td>
</tr>
<tr>
<td>FP</td>
<td>.22</td>
<td>43</td>
<td>.7535</td>
</tr>
<tr>
<td>The LEM2 Approach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLI</td>
<td>65.69</td>
<td>27</td>
<td>.7288</td>
</tr>
<tr>
<td>MLX</td>
<td>39.64</td>
<td>27</td>
<td>.738</td>
</tr>
<tr>
<td>FLI</td>
<td>47.94</td>
<td>32</td>
<td>.7192</td>
</tr>
<tr>
<td>FLX</td>
<td>61.33</td>
<td>32</td>
<td>.728</td>
</tr>
<tr>
<td>The Dynamic Approach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLS</td>
<td>65.97</td>
<td>28</td>
<td>.7332</td>
</tr>
<tr>
<td>DV</td>
<td>17.37</td>
<td>N/A</td>
<td>.7373</td>
</tr>
<tr>
<td>FLS</td>
<td>38.48</td>
<td>32</td>
<td>.7144</td>
</tr>
</tbody>
</table>

Table 6.5: Lymphography results

6.5.3 Balance Scale results

The Balance Scale dataset contains numerical values corresponding to values in an equation describing when something is in balance. In table 6.6 we can see that the \texttt{RSDiscretize} discretization algorithm evidently yields many more intervals than the standard discretization, and the rule sets produced by \texttt{LEM2} thus become much larger. However, the accuracies\textsuperscript{16} of each individual rule, as well as of the rules for each decision class, were much better than those generated from the dataset discretized by \texttt{Discretize-Filter}, according to initial tests with only one ten-fold cross-validation.

Statistical results

Here, we have very strong indications that the Dynamic Approach actually scores better. A significant difference in the mean values combined with small variances of the results give us a that indication.

6.5.4 Zoo results

Our final testing dataset shows some interesting differences compared to earlier results.

\textsuperscript{16}as measured by the function \texttt{Stability} in algorithm 4.7.3
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Time</th>
<th>Rules</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weka</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>.23</td>
<td>9</td>
<td>.7201</td>
</tr>
<tr>
<td>FP</td>
<td>.28</td>
<td>41</td>
<td>.5361</td>
</tr>
<tr>
<td>The LEM2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLI</td>
<td>148.57</td>
<td>303</td>
<td>.734</td>
</tr>
<tr>
<td>MLX</td>
<td>149.93</td>
<td>303</td>
<td>.7404</td>
</tr>
<tr>
<td>FLI</td>
<td>2.84</td>
<td>15</td>
<td>.7418</td>
</tr>
<tr>
<td>FLX</td>
<td>2.87</td>
<td>15</td>
<td>.625</td>
</tr>
<tr>
<td>The Dynamic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLS</td>
<td>161.04</td>
<td>264</td>
<td>.7735</td>
</tr>
<tr>
<td>FLS</td>
<td>2.84</td>
<td>15</td>
<td>.6714</td>
</tr>
<tr>
<td>DV</td>
<td>129.52</td>
<td>N/A</td>
<td>.7831</td>
</tr>
</tbody>
</table>

Table 6.6: Balance Scale results

- The number of rules generated by DecisionTable is more than the number of rules produced by LEM2. This could be explained by the fact that different attributes are useful when determining whether or not an animal is a mammal than determining whether it is a bird or not. For instance, only mammals produce milk so knowing that would be enough for that class of animals. However, the "milk" attribute does not distinguish an insect from a fish since neither produce milk.

- Prism chooses to discriminate the animals by their names. Therefore it is unable to classify animals with unusual names and falls behind miserably in the number of correctly classified instances.

- The choice of rule negotiation method is quite irrelevant here as there is only one rule per class, very few attributes per rule and there are no instances such that they match a condition in a rule while not having the decision value of the rule. This can be seen clearly in the table.

- LEM2 fares very well against the standard algorithms and scores consistently above the levels of other methods. However, the error rate overall is very low and though the cross-validation is stratified, if we use ten folds during our cross-validation and there are only 4 (amphibian) or 5 (reptile) objects in each of the smaller decision classes,
### 6.5. Results

#### Scheme

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Scheme</th>
<th>Time</th>
<th>Rules</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weka Algorithms</td>
<td>FT</td>
<td>.4</td>
<td>16</td>
<td>.9146</td>
</tr>
<tr>
<td></td>
<td>FP</td>
<td>.15</td>
<td>41</td>
<td>.6233</td>
</tr>
<tr>
<td>The LEM2 Approach</td>
<td>MLI</td>
<td>3.11</td>
<td>10</td>
<td>.9464</td>
</tr>
<tr>
<td></td>
<td>MLX</td>
<td>3.13</td>
<td>10</td>
<td>.9464</td>
</tr>
<tr>
<td></td>
<td>FLI</td>
<td>2.92</td>
<td>10</td>
<td>.9235</td>
</tr>
<tr>
<td></td>
<td>FLX</td>
<td>2.89</td>
<td>10</td>
<td>.9235</td>
</tr>
<tr>
<td>The Dynamic Approach</td>
<td>DLS</td>
<td>3.92</td>
<td>10</td>
<td>.9364</td>
</tr>
<tr>
<td></td>
<td>FLS</td>
<td>2.87</td>
<td>10</td>
<td>.9245</td>
</tr>
<tr>
<td></td>
<td>DV</td>
<td>.92</td>
<td>N/A</td>
<td>.9346</td>
</tr>
</tbody>
</table>

Table 6.7: Zoo results

#### Algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Breast Cancer</th>
<th>Lymphography</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our LEM2</td>
<td>0.68</td>
<td>0.738</td>
</tr>
<tr>
<td>Orig. LEM2</td>
<td>0.70</td>
<td>0.81</td>
</tr>
<tr>
<td>Our Dyn.</td>
<td>0.6902</td>
<td>0.7373</td>
</tr>
<tr>
<td>Orig. Dyn.</td>
<td>0.772</td>
<td>0.843</td>
</tr>
</tbody>
</table>

Table 6.8: Comparison with previous results

it is not likely that we will always have examples of all classes during testing.

### Statistical results

For this dataset, we can say that LEM2 is the better approach with 95% significance.

#### 6.5.5 Previous results

In table 6.8 we have put together a summary of our results compared to those originally obtained with the algorithms we have used.

The second main article used for this thesis, [Baz98], uses a number of different combinations of preprocessing, classification and rule negotiation,
much in the same way as we do in table 6.2. The best versions of those combinations had error rates as in table 6.8, on the row labeled 'Orig. Dyn.' (short for Original Dynamic Approach). We can not say that our results support the idea that one approach was better on these two datasets in our tests, given a significance level of 95%, but then we have not either used exactly the same setups as in [Baz98].

Finally, some concluding remarks about the suitability of discretization and rule negotiation techniques:

**Discretization** It seems as if the standard Fayyad-Irani discretization method makes it more crucial to choose the right rule negotiation method, though improved precision may result. The Rough Set technique of using discernibility produces cuts of roughly equal quality. On both the breast-cancer and lymphography datasets, we can note that the Rough Set discretization makes it less important which rule negotiation method that is later chosen. The dynamic reduct discretization actually is worse on both those datasets, which could be due to the fairly small number of subtables used for stability calculation or the number of iterations used to find proper reducts of the super-reduct of cuts. However, it proved to be very competitive in the balance-scale tests.

**Rule Negotiation** As to the rule negotiation methods, it seemed as if the Simple Strength could actually be the best choice, as seen in the lymphography results. The dynamic version, Stability strength, was only able to do better than the rest (comparing schemes that differ only by rule negotiation, FL(I|X|S)) in the breast-cancer test.

Both these remarks do not hold statistical significance however, and the only thing we can say is that our results in two cases out of four indicate no significant differences between the two approaches.

### 6.6 A strange lesson

When the algorithms were first compared to each other in a single cross-validation setup, it happened that the Voter, independent of discretization
filter, scored consistently much better than any of the other algorithms. Repeated tests showed the very same pattern - that the Voter scored better than any of the algorithms published in literature. It turned out that for some reason unknown to us, the deterministic partitioning scheme that Weka uses during cross-validation favored the Voter on all datasets to the extent that it had error rates of 0% on some datasets. With 10x10-fold cross-validation however, the results became somewhat more realistic as demonstrated above.
Chapter 7

Discussion

In this final chapter, we will summarize and discuss the results, draw conclusions from the experiments and suggest directions for continued research.

7.1 Methodology criticism

Criticism can be directed towards the general goal of comparing learning algorithms and in particular towards the methodology used here. In [Sal99] it is claimed that much greater care must be taken when designing and conducting empirical experiments than what we have done. However, others (e.g. [Die98]) claim that conducting t-tests on multiple cross-validation only has a slight inclination towards producing Type I errors (falsely indicating that the null hypothesis should be rejected) and that it is a fully reliable comparison method. Also, in [Die98] it is claimed that averaging the results of only five runs of two-fold cross-validation suffices to produce statistically significant difference results.

7.1.1 Implementation

In the original implementation of LEM2, a rule negotiation method not implemented here was employed. It may have created a disadvantage to the
vanilla LEM2 approach to do this, though it was a decision made in cooperation with the supervisor of this project, Marcin Szczuka. His opinion on the matter was that it would probably not be of major significance.

7.1.2 Evaluation methods

In evaluating the classification algorithms, the different combinations of algorithms described in table 6.2 could be made even more fine-grained if parameters for rule approximation or dynamic reduct stability parameters were also included. However, to obtain results within reasonable time limits, initial tests were made to establish some default values of these parameters and they were not further investigated during the actual testing phase. This heuristic may have had an impact on the results.

7.1.3 Relaxations

The relaxations described in section 6.4.1 may have had a negative impact on the results. However, using our implementation of the Dynamic Approach they deemed necessary. The second relaxation was even necessary in order to avoid having an exponential time complexity of the algorithm (as explained in section 4.3.1).

7.1.4 Datasets

The results from the tests indicated several shortcomings regarding the datasets. They have very different characteristics, and perhaps they are also too small to show just how efficient the algorithms are. The only exception with respect to size is the balance-scale dataset which is reasonably large. No model for explaining the behavior observed is presented here however, and in order to obtain such a model a more thorough analysis that exhaustively tests all combinations of dataset characteristics would have to be made. We could then test how the algorithms react to these characteristics. Also, since we cannot claim that these sets are representative samples in a larger population, no generalizations of our results can be made.


7.2 Conclusion

The results obtained on these datasets, with these algorithms and restrictions, are inconclusive. As stated in section 6.5.5, the only thing we can say is that our results in only two cases\(^1\) out of four indicate significant differences between the two approaches, but not in the other two\(^2\). Though one approach or another may seem more suitable in a particular case, we cannot explain our results as well as we had predicted that we would be able to do - in short, the tendencies perceived by the original authors could not be reproduced in all cases.

7.3 Future Work

To recapitulate the hypothesis presented in section 1.2, we believed that it was possible to reconstruct the algorithms presented in [GB97, Baz98] using the written algorithmic descriptions of them. Also, by performing similar experiments as originally, we believed that we could obtain results in support of the same conclusions as in [Baz98].

The main tendency in [Baz98] was that Bazans Dynamic Approach scored much better than the LEM2 Approach. That tendency could not be established here, which we can attribute to several factors as listed in table 7.1. Table 7.1 identifies the factors, along with suggestions regarding how to continue this work\(^3\) by exploring in detail the way they affect the performance of the algorithms.

7.3.1 Explanatory models

In order to generate an explanatory model which would help us choose a suitable ML algorithm for each problem domain, one would have to perform much more exhaustive tests than the ones performed here. Those tests would have to use not standard benchmarking datasets, but rather datasets

---

\(^1\)Balance Scale and Zoo  
\(^2\)Breast Cancer and Lymphography  
\(^3\)where possible
## 7.3. Future Work

<table>
<thead>
<tr>
<th><strong>Issue</strong></th>
<th><strong>Suggested solution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The relaxations in the Dynamic Approach that were made (see section 6.4.1)</td>
<td>Exhaustive testing of the various options that were here set to default values in the Dynamic Approach.</td>
</tr>
<tr>
<td>The lack of differences in the implementation of internal datstructures and common functionalities between the approaches in the current implementation</td>
<td>The implementation details cannot be analyzed and compared to those of the original authors, since both original implementations are closed-source versions.</td>
</tr>
<tr>
<td>The Student’s t-test, which was not used in the original articles.</td>
<td>Not using the Student’s t-test would result in comparing the raw numbers presented in the tables of results in chapter 6.</td>
</tr>
<tr>
<td>The use of rule approximation (see section 4.7.2) in LEM2</td>
<td>Not approximating rules could be done by setting $\varepsilon$ in algorithm 4.7.3 (rule approximation) to 1.</td>
</tr>
<tr>
<td>The use of different rule negotiation methods than the ones originally used in [GB97]</td>
<td>Implementing the original rule strength algorithm employed in [GB97] could be done in the Rule-class (see section 5.4).</td>
</tr>
</tbody>
</table>

Table 7.1: Issues to be resolved in future work
generated randomly\textsuperscript{4}. For example, the following could be varied when generating these random sets:

- number of instances
- number of attributes
- number of missing values
- number of numerical attributes
- range of numerical and nominal attributes
- number of decision classes
- distributions of attribute values

By varying one of these parameters and running all applicable and available machine learning algorithms on each of the sets, there would be a much better chance of establishing common criteria that could be used for choosing a machine learning algorithm based on the problem domain at hand. This, however, is left for future work.

\textsuperscript{4}a small program for randomly generating datasets was developed as part of this project and could be used for this purpose
Bibliography


Index

algorithm
   evaluation of an .................. 18
ambiguity .......................... 12
approximation ..................... 26–28
   lower ........................... 27
   upper ........................... 27
attributes .......................... 23
AttributeValuePair ................. 70
AttributeValuePairs ............... 70

Balance Scale
   datasets .......................... 76
Breast Cancer
   datasets .......................... 76
classification ....................... 58
   Voter ............................ 58–60
condition attributes ............... 24
cut ................................. see cutpoint
CutPoint ............................ 71
   updateValues() .................. 71
cutpoints ............................ 44
data ................................ 76
decision attribute .................. 24
decision classes .................... 24
decision rules ....................... 28
approximation of ................... 56–57
maximal strength of ............... 60
negotiation ........................ 65
negotiation among ................. 60–64
simple strength of ................ 60
stability coefficient of .......... 63
stability strength of .............. 62–63
decision system ..................... 24
decision table ....................... 23
DecisionTable ....................... 16
difference coefficient ............. 63
discernibility ....................... 44–46
discretization ....................... 13, 44
   algorithms ....................... 46–52
dynamic ............................ 50
global ............................. 47
local .............................. 46
dynamic reduct ..................... see reduct
dynamic reduct stability ........... see reduct
   DynamicReductDiscretize ...... 73
fuzzy sets .......................... 22
indiscernibility relation ........... 24
leave-one-out ....................... 18
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEM2</td>
<td>73</td>
</tr>
<tr>
<td>local covering</td>
<td>33</td>
</tr>
<tr>
<td>lower approximation</td>
<td>33</td>
</tr>
<tr>
<td>approximation</td>
<td></td>
</tr>
<tr>
<td>Lymphography</td>
<td>76</td>
</tr>
<tr>
<td>datasets</td>
<td></td>
</tr>
<tr>
<td>machine learning</td>
<td>7</td>
</tr>
<tr>
<td>maximal strength</td>
<td>77</td>
</tr>
<tr>
<td>see decision rules</td>
<td></td>
</tr>
<tr>
<td>MDCutPoints</td>
<td>71</td>
</tr>
<tr>
<td>minimal complex</td>
<td>32</td>
</tr>
<tr>
<td>model construction</td>
<td>57</td>
</tr>
<tr>
<td>LEM2</td>
<td>53–55</td>
</tr>
<tr>
<td>model construction of</td>
<td>53</td>
</tr>
<tr>
<td>N-fold cross-validation</td>
<td>18</td>
</tr>
<tr>
<td>OneR</td>
<td>16</td>
</tr>
<tr>
<td>overfitting</td>
<td>12</td>
</tr>
<tr>
<td>positive region</td>
<td>16</td>
</tr>
<tr>
<td>Prism</td>
<td></td>
</tr>
<tr>
<td>Reduct</td>
<td>73</td>
</tr>
<tr>
<td>reduct</td>
<td>25</td>
</tr>
<tr>
<td>computation of</td>
<td>73</td>
</tr>
<tr>
<td>dynamic</td>
<td>28–32</td>
</tr>
<tr>
<td>stability coefficient</td>
<td>31</td>
</tr>
<tr>
<td>stability of</td>
<td>29</td>
</tr>
<tr>
<td>positive region of</td>
<td>27</td>
</tr>
<tr>
<td>proper</td>
<td>24</td>
</tr>
<tr>
<td>relative</td>
<td>25</td>
</tr>
<tr>
<td>super-reduct</td>
<td>24</td>
</tr>
<tr>
<td>Reducts</td>
<td>73</td>
</tr>
<tr>
<td>relative reduct</td>
<td>25</td>
</tr>
<tr>
<td>RelativeReductComputation</td>
<td>73</td>
</tr>
<tr>
<td>Rough Sets</td>
<td>22</td>
</tr>
<tr>
<td>RoughInstance</td>
<td>69</td>
</tr>
<tr>
<td>RoughInstances</td>
<td>69</td>
</tr>
<tr>
<td>RoughSet</td>
<td>72</td>
</tr>
<tr>
<td>RSDiscretize</td>
<td>73</td>
</tr>
<tr>
<td>RuleClassifier</td>
<td>73</td>
</tr>
<tr>
<td>Rules</td>
<td></td>
</tr>
<tr>
<td>addRules</td>
<td>70</td>
</tr>
<tr>
<td>Rules</td>
<td>70</td>
</tr>
<tr>
<td>significance</td>
<td></td>
</tr>
<tr>
<td>see t-test</td>
<td></td>
</tr>
<tr>
<td>t-test</td>
<td>76</td>
</tr>
<tr>
<td>simple strength</td>
<td></td>
</tr>
<tr>
<td>see decision rules</td>
<td></td>
</tr>
<tr>
<td>SortedClasses</td>
<td>72</td>
</tr>
<tr>
<td>stability strength</td>
<td></td>
</tr>
<tr>
<td>see decision rules</td>
<td></td>
</tr>
<tr>
<td>rules</td>
<td></td>
</tr>
<tr>
<td>stratification</td>
<td></td>
</tr>
<tr>
<td>N-fold cross-validation</td>
<td>81</td>
</tr>
<tr>
<td>super-reduct</td>
<td></td>
</tr>
<tr>
<td>see reduct</td>
<td></td>
</tr>
<tr>
<td>t-test</td>
<td>75</td>
</tr>
<tr>
<td>non-paired</td>
<td>75</td>
</tr>
<tr>
<td>paired</td>
<td>75</td>
</tr>
<tr>
<td>universe</td>
<td>23</td>
</tr>
<tr>
<td>upper approximation</td>
<td></td>
</tr>
<tr>
<td>see approximation</td>
<td></td>
</tr>
<tr>
<td>Voter</td>
<td>73</td>
</tr>
<tr>
<td>Weka</td>
<td>3</td>
</tr>
<tr>
<td>ZeroR</td>
<td>16</td>
</tr>
<tr>
<td>Zoo</td>
<td></td>
</tr>
<tr>
<td>datasets</td>
<td>76</td>
</tr>
</tbody>
</table>
This thesis presents the results of the implementation and evaluation of two machine learning algorithms [Baz98, GB97] based on notions from Rough Set theory [Paw82]. Both algorithms were implemented and tested using the Weka [WF00] software framework. The main purpose for doing this was to investigate whether the experimental results obtained in [Baz98] could be reproduced, by implementing both algorithms in a framework that provided common functionalities needed by both. As a result of this thesis, a Rough Set framework accompanying the Weka system was designed and implemented, as well as three methods for discretization and three classification methods.

The results of the evaluation did not match those obtained by the original authors. On two standard benchmarking datasets also used previously in [Baz98] (Breast Cancer and Lymphography), significant results indicating that one of the algorithms performed better than the other could not be established, using the Students t-test and a confidence limit of 95%. However, on two other datasets (Balance Scale and Zoo) differences could be established with more than 95% significance. The “Dynamic Reduct Approach” scored better on the Balance Scale dataset whilst the “LEM2 Approach” scored better on the Zoo dataset.
Copyright

Svenska

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under en länge tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är krävande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Ola Leifler
Linköping, 9th October 2002