ABSTRACT

A classifier is a piece of software that is able to categorize objects for which the class is unknown. The task of automatically generating classifiers by generalizing from examples is an important problem in many practical applications. This problem is often referred to as supervised concept learning, and has been shown to be relevant in e.g. medical diagnosis, speech and handwriting recognition, stock market analysis, and other data mining applications.

The main purpose of this thesis is to analyze current approaches to evaluate classifiers as well as supervised concept learners and to explore possible improvements in terms of alternative or complementary approaches. In particular, we investigate the metric-based approach to evaluation as well as how it can be used when learning. Any supervised concept learning algorithm can be viewed as trying to generate a classifier that optimizes a specific, often implicit, metric (this is sometimes also referred to as the inductive bias of the algorithm). In addition, different metrics are suitable for different learning tasks, i.e., the requirements vary between application domains. The idea of metric-based learning is to both make the metric explicit and let it be defined by the user based on the learning task at hand.

The thesis contains seven studies, each with its own focus and scope. First, we present an analysis of current evaluation methods and contribute with a formalization of the problems of learning, classification and evaluation. We then present two quality attributes, sensitivity and classification performance, that can be used to evaluate learning algorithms. To demonstrate their usefulness, two metrics for these attributes are defined and used to quantify the impact of parameter tuning and the overall performance. Next, we refine an approach to multi-criteria classifier evaluation, based on the combination of three metrics and present algorithms for calculating these metrics. In the fourth study, we present a new method for multi-criteria evaluation, which is generic in the sense that it only dictates how to combine metrics. The actual choice of metrics is application-specific. The fifth study investigates whether or not the performance according to an arbitrary application-specific metric can be boosted by using that metric as the one that the learning algorithm aims to optimize. The subsequent study presents a novel data mining application for preventing spyware by classifying End User License Agreements. A number of state-of-the-art learning algorithms are compared using the generic multi-criteria method. Finally, in the last study we describe how methods from the area of software engineering can be used to solve the problem of selecting relevant evaluation metrics for the application at hand.
On the Metric-based Approach to Supervised Concept Learning

Niklas Lavesson
On the Metric-based Approach to Supervised Concept Learning

Niklas Lavesson
“something might perhaps be made out on this question by patiently acum-
mulating and reflecting on all sorts of facts which could possibly have any
bearing on it. After five years’ work I allowed myself to speculate on the
subject, and drew up some short notes”

– Charles Darwin - The Origin of Species (1859)
A classifier is a piece of software that is able to categorize objects for which the class is unknown. The task of automatically generating classifiers by generalizing from examples is an important problem in many practical applications. This problem is often referred to as supervised concept learning, and has been shown to be relevant in e.g. medical diagnosis, speech and handwriting recognition, stock market analysis, and other data mining applications. The main purpose of this thesis is to analyze current approaches to evaluate classifiers as well as supervised concept learners and to explore possible improvements in terms of alternative or complementary approaches. In particular, we investigate the metric-based approach to evaluation as well as how it can be used when learning. Any supervised concept learning algorithm can be viewed as trying to generate a classifier that optimizes a specific, often implicit, metric (this is sometimes also referred to as the inductive bias of the algorithm). In addition, different metrics are suitable for different learning tasks, i.e., the requirements vary between application domains. The idea of metric-based learning is to both make the metric explicit and let it be defined by the user based on the learning task at hand. The thesis contains seven studies, each with its own focus and scope. First, we present an analysis of current evaluation methods and contribute with a formalization of the problems of learning, classification and evaluation. We then present two quality attributes, sensitivity and classification performance, that can be used to evaluate learning algorithms. To demonstrate their usefulness, two metrics for these attributes are defined and used to quantify the impact of parameter tuning and the overall performance. Next, we refine an approach to multi-criteria classifier evaluation, based on the combination of three metrics and present algorithms for calculating these metrics. In the fourth study, we present a new method for multi-criteria evaluation, which is generic in the sense that it only dictates how to combine metrics. The actual choice of metrics is application-specific. The fifth study investigates whether or not the performance according to an arbitrary application-specific metric can be boosted by using that metric as the one that the learning algorithm aims to optimize. The subsequent study presents a novel data mining application for preventing spyware by classifying End User License Agreements. A number of state-of-the-art learning algorithms are compared using the generic multi-criteria method. Finally, in the last study we describe how methods from the area of software engineering can be used to solve the problem of selecting relevant evaluation metrics for the application at hand.
Acknowledgments

It is my great privilege to get this opportunity to thank all of whom have helped and supported me throughout the course of writing this thesis.

I cannot overstate the gratitude to my supervisor, Professor Paul Davidsson, for sharing his valuable insights and ideas, and for always being supportive and encouraging. I also want to convey my deepest appreciation to Dr. Fredrik Wernstedt who, alongside Professor Davidsson, introduced me to machine learning and to Professor Henrik Boström for his valuable comments and suggestions during the defense of my Licentiate’s thesis.

Furthermore, I would like to acknowledge my assistant supervisor, Dr. Stefan Johansson, and the rest of my colleagues at the Distributed and Intelligent Systems Laboratory (DISL) for contributing to a stimulating research environment and many interesting discussions. Especially, I would like to thank my dear friend and colleague, Marie Persson, for always being supportive and also for her wit and great sense of humor.

To my parents, Annette and Lars, I state my immense gratitude for providing me and my brothers with love, confidence and support, and for always helping me in any way they possibly can.

Last but not least, I would like to thank Gong. Thank you for being my loved one and for letting me be part of your world. This thesis would have been impossible to write without you.

Ronneby, November 2008
Niklas Lavesson
Preface

This compilation thesis comprises seven papers. These are listed below and will be referenced in the text by the associated Roman numerals. Each previously published paper has been revised to conform to the thesis template and to address minor indistinctions and errors.


The author of the thesis is the main contributor to all of these papers. In addition, the following papers are related to the thesis:


Papers II, IV, and VII are extended versions of papers VIII, IX, and XII, respectively. Moreover, Paper VI is an extended synthesis of papers X and XI.
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td>Preface</td>
<td>v</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Concept Learning Algorithms</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 Evaluation of Classifiers and Learning Algorithms</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Related Work</td>
<td>5</td>
</tr>
<tr>
<td>1.2.1 Evaluation Methods and Metrics</td>
<td>6</td>
</tr>
<tr>
<td>1.2.2 Modification of the Algorithmic Bias</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Research Questions</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Research Methods</td>
<td>9</td>
</tr>
<tr>
<td>1.4.1 Theoretical Approaches</td>
<td>9</td>
</tr>
<tr>
<td>1.4.2 Empirical Approaches</td>
<td>9</td>
</tr>
<tr>
<td>1.5 Contributions</td>
<td>10</td>
</tr>
<tr>
<td>1.5.1 RQ1: What Are the Characteristics of the Supervised Concept Learning, Classification, and Evaluation Problems and Solutions?</td>
<td>10</td>
</tr>
<tr>
<td>1.5.2 RQ2: What Is the Impact of Learning Algorithm Parameter Tuning and How Can It Be Measured?</td>
<td>10</td>
</tr>
<tr>
<td>1.5.3 RQ3: How Can Multiple Criteria Be Considered During Evaluation?</td>
<td>11</td>
</tr>
<tr>
<td>1.5.4 RQ4: How Can Learning Be Optimized Toward Arbitrary Metrics?</td>
<td>13</td>
</tr>
<tr>
<td>1.5.5 RQ5: How Can Evaluation Metrics Be Systematically Selected for Different Applications?</td>
<td>14</td>
</tr>
<tr>
<td>1.6 Conclusions</td>
<td>15</td>
</tr>
<tr>
<td>1.7 Future Work</td>
<td>15</td>
</tr>
<tr>
<td>1.7.1 Metric-based Evaluation</td>
<td>15</td>
</tr>
<tr>
<td>1.7.2 Metric-based Learning</td>
<td>16</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>2 Paper I</td>
<td>Metric Selection</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td></td>
</tr>
<tr>
<td>2.1.1 Related Work</td>
<td></td>
</tr>
<tr>
<td>2.1.2 Outline</td>
<td></td>
</tr>
<tr>
<td>2.2 Framework</td>
<td></td>
</tr>
<tr>
<td>2.2.1 The Classification Problem</td>
<td></td>
</tr>
<tr>
<td>2.2.2 The Learning Problem</td>
<td></td>
</tr>
<tr>
<td>2.2.3 The Evaluation Problems</td>
<td></td>
</tr>
<tr>
<td>2.3 Evaluation Method Taxonomy</td>
<td></td>
</tr>
<tr>
<td>2.4 Evaluation Methods</td>
<td></td>
</tr>
<tr>
<td>2.4.1 Classifier Evaluation</td>
<td></td>
</tr>
<tr>
<td>2.4.2 Algorithm Configuration Evaluation</td>
<td></td>
</tr>
<tr>
<td>2.4.3 Algorithm Evaluation</td>
<td></td>
</tr>
<tr>
<td>2.5 Conclusions</td>
<td></td>
</tr>
<tr>
<td>3 Paper II</td>
<td></td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td></td>
</tr>
<tr>
<td>3.2 Quality Attribute Metrics</td>
<td></td>
</tr>
<tr>
<td>3.3 Experiment Design</td>
<td></td>
</tr>
<tr>
<td>3.3.1 Featured Algorithms</td>
<td></td>
</tr>
<tr>
<td>3.3.2 Procedure</td>
<td></td>
</tr>
<tr>
<td>3.4 Results</td>
<td></td>
</tr>
<tr>
<td>3.5 Discussion</td>
<td></td>
</tr>
<tr>
<td>3.6 Related Work</td>
<td></td>
</tr>
<tr>
<td>3.7 Conclusions and Future Work</td>
<td></td>
</tr>
<tr>
<td>4 Paper III</td>
<td></td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td></td>
</tr>
<tr>
<td>4.2 Measure-based Evaluation</td>
<td></td>
</tr>
<tr>
<td>4.3 Cross-validation Evaluation</td>
<td></td>
</tr>
<tr>
<td>4.4 A Multi-dimensional Measure Function</td>
<td></td>
</tr>
<tr>
<td>4.4.1 Similarity</td>
<td></td>
</tr>
<tr>
<td>4.4.2 Simplicity</td>
<td></td>
</tr>
<tr>
<td>4.4.3 Subset Fit</td>
<td></td>
</tr>
<tr>
<td>4.4.4 A Weighted Measure Function</td>
<td></td>
</tr>
<tr>
<td>4.5 Experiments</td>
<td></td>
</tr>
<tr>
<td>4.6 Conclusions and Future Work</td>
<td></td>
</tr>
<tr>
<td>5 Paper IV</td>
<td></td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td></td>
</tr>
<tr>
<td>5.1.1 Multi-Criteria Evaluation</td>
<td></td>
</tr>
<tr>
<td>5.2 Generic Multi-Criteria Methods</td>
<td></td>
</tr>
<tr>
<td>5.2.1 The Efficiency Method</td>
<td></td>
</tr>
<tr>
<td>5.2.2 The SIM Method</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>5.2.3</td>
<td>The Measure-Based Method</td>
</tr>
<tr>
<td>5.3</td>
<td>Analysis</td>
</tr>
<tr>
<td>5.4</td>
<td>A New Generic MC Method</td>
</tr>
<tr>
<td>5.5</td>
<td>Case study</td>
</tr>
<tr>
<td>5.5.1</td>
<td>CEF Evaluation</td>
</tr>
<tr>
<td>5.6</td>
<td>Conclusions and Future Work</td>
</tr>
<tr>
<td>6</td>
<td>Paper V</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Motivation</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Outline</td>
</tr>
<tr>
<td>6.2</td>
<td>Background</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Inherent Metrics</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Evaluation Metrics</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Definitions</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Related Work</td>
</tr>
<tr>
<td>6.3</td>
<td>Metric-based Learning</td>
</tr>
<tr>
<td>6.3.1</td>
<td>A Metric-based One Rule Inducer</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Metrics for Metric-based Algorithms</td>
</tr>
<tr>
<td>6.4</td>
<td>Experiments</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Metrics</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Data Sets</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Experiment 1</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Experiment 2</td>
</tr>
<tr>
<td>6.5</td>
<td>Discussion</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusions and Future Work</td>
</tr>
<tr>
<td>7</td>
<td>Paper VI</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Background</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Related Work</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Scope and Aim</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Outline</td>
</tr>
<tr>
<td>7.2</td>
<td>EULA Classification</td>
</tr>
<tr>
<td>7.2.1</td>
<td>The EULA Classification Task</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Supervised Concept Learning</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Representation</td>
</tr>
<tr>
<td>7.3</td>
<td>Data Sets</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Data Collection</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Data Representation</td>
</tr>
<tr>
<td>7.4</td>
<td>Experiments</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Algorithm Selection and Configuration</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Evaluation of Classifier Performance</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Experimental Procedure</td>
</tr>
<tr>
<td>7.5</td>
<td>Results</td>
</tr>
</tbody>
</table>
Technological advances have spawned a change of lifestyle and expanded the focus of the global economy from production of physical goods to manipulation of information. As a consequence, we rely more and more heavily on databases. The number, and especially the size, of these databases grow quickly. In fact, it is argued that stored data is doubling every nine months (Kargupta, Joshi, Sivakumar, & Yesha, 2004). It is therefore becoming increasingly hard to extract useful information. Kargupta et al. (2004) note that data mining technologies have been shown to perform well at this task in a wide variety of science, business, and technology areas.

Data mining, or knowledge discovery, draws on work conducted in a variety of areas such as: machine learning, statistics, and high performance computing. The main problem studied is how to find useful information in large quantities, or otherwise complex types, of data. Although the nature of this problem can be very different across applications, one of the most common tasks is that of identifying structural patterns in data that can then be used to categorize the data into a distinct set of categories (Witten & Frank, 2005). If these patterns can actually distinguish between different categories of data this implies that they have captured some generalized characteristics of each category. As it turns out, the area of machine learning provides a number of approaches to automatically learn this kind of concepts by generalizing from categorized data.

Evaluation of learning algorithms (and the learned concept descriptions) is required both to ensure that a problem is solved sufficiently well by an approach, and to select the most appropriate learning algorithm out of the many available algorithms. The aim of this thesis is to highlight some of the central issues of evaluation and, more importantly, to present new evaluation methods and metrics for a specific class of machine learning tasks referred to as supervised concept learning.

The remainder of this chapter is organized as follows: first we give some background on supervised concept learning and evaluation and this is followed by a review of related
work. We then present the research questions and discuss the research methods used to investigate these questions. Finally, we describe the contributions and end with conclusions and directions for future work.

1.1 Background

A classifier can be viewed as a piece of software that is able to classify data instances for which the class, or category, is unknown. The task of automatically generating classifiers by generalizing from examples is an important problem in many practical applications, for example: prediction, diagnosis, pattern recognition, control of vehicles, and so forth. More specifically, supervised concept learning as been applied to: text categorization (Sebastiani, 2002), junk email filtering (Sahami, Dumais, Heckerman, & Horvitz, 2001), authorship attribution (Diederich, Kindermann, Leopold, & Paass, 2003), fraud detection (Hilas & Mastorocostas, 2008), and prediction of diabetic control status (Huang, McCullagh, Black, & Harper, 2007). This problem is often referred to as supervised concept learning.

1.1.1 Concept Learning Algorithms

A supervised concept learner is an algorithm for learning classifiers. There exist a vast number of learning algorithms but in essence they all have the same basic functionality: given a set of examples of instances with known classes (a data set), the algorithm uses some assumptions to generalize from the examples and it then outputs a classifier, which is a mapping between instances and classes. The set of assumptions is what distinguishes different learning algorithms and is generally referred to as the inductive bias. Without prior assumptions a learner would have no rational basis for generalizing from examples (Mitchell, 1997a), i.e., the bias is what makes an algorithm prefer one generalization over another.

The weather problem is represented by a small data set that can be used to explain the generic supervised concept learning task. This data set features 14 instances, as shown in Table 1.1. Each instance describes the weather on one particular day using the four attributes: outlook, temperature, humidity, and windy. The concept we want to learn is \textit{days for which we would like to go outdoors to play some game}. The target attribute, play, represents the choice of going outdoors to play the game.

The weather data set represents a binary classification problem in the sense that there are only two possible classes; yes or no. In addition to the target attribute, we can observe that the four input attributes are also nominal; outlook and temperature both have three possible outcomes, whereas humidity and windy have two each. In total, this gives us \(3 \times 3 \times 2 \times 2 = 36\) possible combinations, out of which 14 are represented in Table 1.1.

We can now formulate two assumptions to generalize from the weather data; (i) a classifier is represented with one rule that is defined by one test or conjunctions of
Table 1.1: The Weather data set

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>yes</td>
</tr>
</tbody>
</table>

several tests, and (ii) a classifier should classify the provided examples as correctly as possible. Each test involves comparing an attribute to a constant, e.g., if Outlook = sunny. The tests together constitute the precondition for the rule and the conclusion of the rule gives the class that applies to instances covered by the rule.

In analyzing these assumptions, we observe that the first assumption restricts the set of possible classifiers because a single conjunctive rule cannot describe all classifiers (this is the representational bias) and the second describes how to generate and compare (rank) classifiers (this is the algorithmic bias). Thus, the inductive bias can be divided into the representational bias and the algorithmic bias.

An interesting aspect of the algorithmic bias is that it in most cases includes a performance element. Essentially, this performance element includes one or more evaluation metrics (in our case: accuracy). Since the term evaluation metric usually refers to the metric used for evaluating the output of a learning algorithm, it is necessary to make a distinction: we refer to the metric in the performance element as the learning metric and the metric used for evaluating the generated classifier as the evaluation metric.

The learning metric is used to compare different rules, however, we also need a way to systematically generate candidate rules in order to have something to compare. This systematic generation can often be described as a search through the space of possible classifiers. A simple algorithm could, for example, begin by randomly generating a rule and calculating the number of correct classifications. It could then continue to randomly generate a new rule and compare the performance of this rule to that of the first rule. If
the new rule is better, the first rule is discarded. This process continues until we reach a certain threshold, e.g., at least 80 per cent correct classifications or after a given number of iterations, e.g., 1,000. One classifier that could be generated by our learning algorithm is:

\[
\text{if (Humidity = normal) and (Windy = false) then Play = yes else Play = no}
\]

This classifier would classify 9 out of 14 instances correctly, i.e., it would achieve an accuracy of 64.3 per cent on the available data set. However, the number of correctly classified instances on the data set that was used to for learning the classifier is not a particularly useful evaluation metric. In fact, it will be a very optimistic estimate of generalization performance. Hence, it is important that the evaluation is conducted on an independent test set, i.e., a data set that has not been used for learning the classifier.

To conclude, the inductive bias is what guides a supervised concept learner when generalizing from examples. The set of assumptions that constitute the inductive bias can vary greatly between different algorithms. There is no set of assumptions that are superior for all problems (Wolpert, 2001) but some assumptions work better than others for a particular problem (King, Feng, & Sutherland, 1995).

There are quite a few possible applications for supervised concept learning and one typical example is that of trying to detect the presence or absence of a particular cardiac disorder or to distinguish between different cardiac disorders based on, for example, electrocardiogram recordings\(^1\) (ECGs). A learning algorithm can automatically generate a classifier that makes suggestions about undiagnosed cases by observing some examples of successful diagnoses based on ECG recordings.

### 1.1.2 Evaluation of Classifiers and Learning Algorithms

Evaluation is a central concept of supervised concept learning research and most evaluation tasks concern performance assessment. In fact, most definitions of learning rely on some notion of improved performance. Thus, various performance metrics are the natural dependent variables for machine learning experiments (Langley, 1988).

There is a need for systematic ways to evaluate how well different learning methods work and to compare one with another (Witten & Frank, 2005). In the early days of machine learning some researchers recognized the fact that learning is a central phenomenon in human cognition and therefore suggested that machine learning methods could be evaluated in terms of their ability to explain human learning (Langley, 1986). This can be related to the original artificial intelligence concept of understanding human intelligence by modeling artificial intelligence. However, a large amount of the supervised concept learning research that has been carried out seems to be more focused towards solving tasks that either previously have been done by humans or cannot be done

---

by humans, rather than to explain human behavior. We will here focus on evaluation criteria and metrics that can be used to evaluate the performance of such tasks.

We make a distinction between evaluation criteria and evaluation metrics; the latter are measured to evaluate the former. Criteria, or quality attributes as we refer to them later, represent a higher abstraction. The performance criterion can be broken down into several sub criteria (space, time, correctness, and so forth) and each of these criteria can be evaluated using several metrics. As an example, the performance of a classifier can be measured in terms of the size of its representation, the time it takes to classify, or the correctness of its classifications. Examples of other criteria include: learnability (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1989), complexity (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1987), comprehensibility/understandability (Dehuri & Mall, 2006; Langley, 1986), and interestingness (Freitas, 1998).

In the specific area of supervised concept learning, traditional evaluation methods have been focused toward assessing performance in terms of the predictive accuracy of classifiers; that is, their classification accuracy on data that was not available during the learning phase. However, theoretical and empirical research has raised a number of issues concerning the validity of such assessments. For example: the accuracy metric assumes an equal class distribution (the number of instances must be roughly the same for each class) and equal misclassification costs (the cost of misclassifying a particular class is identical to the cost of misclassifying any of the other classes). Several candidate performance metrics have been proposed. However, it is argued that no metric is superior for all problems (Caruana & Niculescu-Mizil, 2006). In addition, Drummond (2006) argues that the metric we use should represent something we care about. Assuming we want to study the complexity of different algorithms, it would make little sense to just focus on finding the most accurate classifier.

Except for the choices of evaluation criteria and metrics, it is necessary to choose an assessment method. Statistical methods such as cross-validation (Stone, 1974) and bootstrap (Jain, Dubes, & Chen, 1987) are frequently used for the purpose of assessing different performance metrics. However, there is no standard method for systematic selection of criteria or metrics on the basis of problem type or application. Nor is there any standard method for evaluating multiple arbitrary metrics in order to capture application-specific trade-offs.

### 1.2 Related Work

The related work can be divided into two parts. The first part focuses on how to perform evaluations and which metrics to use for quantifying different aspects of performance or other relevant evaluation criteria. The second part is perhaps more narrow and deals with the question of how to optimize learning algorithms by modifying their learning metric(s).
1.2.1 Evaluation Methods and Metrics

Quinlan (1986) wrote a seminal paper on decision tree induction and also explicitly discussed accuracy as a metric for classifier performance. Meanwhile, Langley (1988) pointed out the need to investigate other evaluation metrics as well. In what was arguably the largest empirical machine learning study of its time, King et al. (1995) performed a comparison of 17 supervised learners on 12 real-world data sets. King et al. used accuracy, misclassification cost, training time, comprehensibility, and ease-of-use as evaluation metrics. More recently, Caruana and Niculescu-Mizil (2006) performed another large-scale empirical study that includes state-of-the-art algorithms and metrics.

Originating from signal detection theory, Receiver Operating Characteristic (ROC) analysis (Egan, 1975) was introduced to the machine learning community at large by Provost and Fawcett (1997). It is now a common evaluation method in supervised concept learning. This is much due to the work of Provost, Fawcett, and Kohavi (1998) in demonstrating the benefits of ROC analysis compared to accuracy estimation. ROC curves can be represented as single quantities by using the Area Under the ROC Curve (AUC) metric, which was popularized by Fawcett (2003). In addition to AUC and the aforementioned metrics, there exist several other metrics that are more or less used in different domains, such as: the F-measure (Rijsbergen, 1979) and Recall/Precision (Witten, Moffat, & Bell, 1999). In addition to ROC analysis, other visualization techniques include: lift charts (Berry & Linoff, 1997) and cost curves (Drummond & Holte, 2000).

Nakhaeizadeh and Schnabl (1997) were among the first to present an approach to evaluate arbitrary multiple criteria using a multi-criteria metric. Their method was based on Data Envelopment Analysis (DEA), which is a non-parametric method that originates from operations research. Similarly, Andersson, Davidsson, and Lindén (1999) presented an approach called measure-based evaluation and specified a measure function that consisted of three weighted metrics that each captured the performance according to a certain inductive bias. Moreover, Soares, Costa, and Brazdíl (2000) presented the Simple and Intuitive Measure (SIM), which introduced the concept of acceptable ranges for metrics. Setting a range for a metric enables us, amongst other things, to disqualify a classifier completely if the score it achieves for a particular metric does not suffice for our particular application.

Multiple arbitrary criteria can also be evaluated without transforming the results to a single quantity. Instead, the criteria can be ranked or visualized. Freitas (2004) performs a review of multi-objective optimization in data mining and compares three approaches: (i) transforming the multi-objective problem into a single-objective one, (ii) the Lexicographic approach, and (iii) the Pareto approach. He notes one single argument in favor of single-quantity multi-criteria evaluation; conceptual simplicity and ease-of-use. The possible disadvantages listed are; ad hoc weight setting, mixing different units of measurement, and mixing non-commensurable criteria. For the Lexicographic approach, which is a ranking method, Freitas notes that it recognizes the non-commensurability with different criteria but it introduces a new ad hoc parameter by requiring the user
to specify a tolerance-threshold for each criterion. Finally, he notes that the Pareto approach does not suffer from the disadvantages of the other approaches but it does not single out the best classifier (rather it can recommend a set of classifiers). A classifier is Pareto optimal when it is not possible to improve one objective without deteriorating at least one of the other. A set of Pareto optimal classifiers constitute the Pareto front.

In addition, Japkowicz, Sanghi, and Tischer (2008) present a projection-based evaluation framework that enables visualization of the performance of multiple classifiers on multiple domains. The projection is computed by aggregating performance matrices, such as the confusion matrix, from multiple domains into one vector for each classifier. By applying a projection and a distance measure, the classifier performance vectors can be visualized in two-dimensional space.

1.2.2 Modification of the Algorithmic Bias

Many researchers have studied the relationship between the algorithmic and the representational bias. For example, see the study on evaluation and selection of biases in machine learning (Gordon & Desjardins, 1995) or the study about the need for biases in learning generalizations (Mitchell, 1980). Researchers in the field of meta learning have pointed out the importance of going beyond the engineering goal of producing more accurate learners to the scientific goal of understanding learning behavior (Giraud-Carrier, Vilalta, & Brazdil, 2004).

A number of studies address the question of how to optimize algorithms toward different objectives. In particular, we distinguish between two approaches. The first approach is to optimize a particular metric by replacing the learning metric. Notable examples of such studies include: the optimization of the Area Under the ROC Curve (AUC) using decision trees (Ferri, Flach, & Hernandez-Orallo, 2002) and gradient-descent (Herschtal & Raskutti, 2004), as well as the optimization of the F-measure using support vector machines (Musicant, Kumar, & Özgür, 2003).

The second approach is to optimize more than one metric, either by replacing the learning metric of an existing algorithm with a multi-criteria metric, or by developing a new algorithm that optimizes such a metric. For example, the support vector machines algorithm has been generalized to optimize multi-criteria non-linear performance metrics (Joachims, 2005), and dynamic bias selection has been implemented for prediction rule discovery (Suzuki & Ohno, 1999). Additionally, Andersson et al. (1999) describe how to implement hill-climbing-based learning algorithms that optimize multi-criteria metrics.

1.3 Research Questions

The main purpose of this thesis is to analyze current approaches to evaluate classifiers as well as supervised concept learners and to explore possible improvements in terms of
alternative or complementary approaches. In particular, we investigate the metric-based approach to evaluation as well as how it can be used during the learning phase. The underlying idea of the metric-based approach is to focus on which metrics are appropriate for the application at hand. Metric-based evaluation is the process of evaluating classifiers and learning algorithms using application-specific metrics. Analogously, metric-based learning algorithms are those that try to optimize application-specific metrics. We start by investigating a very fundamental research question:

**RQ1. What are the characteristics of the supervised concept learning, classification, and evaluation problems and solutions?**

The reason for studying this question is that terms that refer to key concepts are sometimes used ambiguously, perhaps due to the multi-disciplinary nature of machine learning. A formalization of the stated problems could simplify the description of current evaluation methods using a consistent terminology.

We have noticed that the evaluation of learning algorithms are often performed in a rather ad hoc fashion, in particular with regard to the choice of the algorithm parameter settings. Therefore, we have chosen to investigate the following research question:

**RQ2. What is the impact of learning algorithm parameter tuning and how can it be measured?**

There may be several important criteria to consider when evaluating classifiers or learning algorithms for different learning tasks. The question is how to evaluate multiple criteria. We formulate the research question:

**RQ3. How can multiple criteria be considered during evaluation?**

When investigating this question we focus specifically on methods that return a single quantity, that is, multi-criteria metrics. One rationale behind this focus is that such methods can be used as input for the next research question.

Existing supervised concept learners rely on learning metrics to guide the selection of, or search for, classifiers on the basis of observed training data. Some learning algorithms have been optimized toward a specific metric by adapting them for this particular metric. We consider the possibility to create algorithms for which the learning metric can be selected on the basis of the problem at hand. Thus, we investigate:

**RQ4. How can learning be optimized toward arbitrary metrics?**

Evaluation metrics are often selected based on standard practice, e.g.: Precision and Recall are used in information retrieval and the Area Under the ROC Curve is used in medical applications. There exist, to our knowledge, no generic approach to identify relevant evaluation criteria and select appropriate metrics for a particular application. Hence, we formulate the last research question:

**RQ5. How can evaluation metrics be systematically selected for different applications?**
1.4 Research Methods

Each research methodology is associated with a set of assumptions about how to learn and what to learn from research inquiries (Creswell, 2003). Machine learning is generally categorized as a branch of computer science but analogously to artificial intelligence it is clearly a multidisciplinary area of research. This means that there is no obvious association with any one research methodology. However, similarly to many other branches of computer science, machine learning research tends to lean toward the scientific method in sense that there is a tendency to focus on the empirical support of claims. In addition, the adherence to scientific principles such as reproducibility and repeatability seems to become increasingly favored by the community.

The research questions have been approached using a mixed methods approach (Creswell, 2003), that is, the choice of whether to use a quantitative method, a qualitative method, or both have been evaluated on the basis of suitability for each research question.

1.4.1 Theoretical Approaches

Analysis form a substantial part of the research conducted in this thesis. A quantitative analysis approach is taken to define mathematical models or formal frameworks for, e.g.: learning algorithms, evaluation metrics and methods, whereas a qualitative approach is used to model and define relationships between concepts and to pursue questions that may be difficult to formalize with variables. We use a quantitative analysis approach to RQ1 and a more qualitative analysis approach to RQ5. The remaining research questions (RQ2, RQ3, and RQ4) are addressed by using quantitative analysis as well as experimentation.

1.4.2 Empirical Approaches

Experimentation is the fundamental research method of empirical computer science. Unlike scientists in, e.g., physics, medicine, and biology, computer scientists often have the advantage of performing experiments in a controlled and designed environment. Amongst other things, this makes it possible to more easily, and less costly, run several iterations with slight changes in configurations. In addition, the objects of investigation in computer science are artifacts (computer-related phenomena) that change concurrently with the development of theories describing them and simultaneously with the growing practical experience in their usage (Dodig-Crnkovic, 2002).

In this thesis, we conduct experiments to investigate hypotheses and sub questions that address different aspects of some of the main research questions (RQ2, RQ3, and RQ4). The experimental results are either subjected to statistical analysis (RQ3 and RQ4 as addressed by papers VI and V, respectively) or presented as proof of concepts (RQ3 as addressed by papers III and IV).
1.5 Contributions

In this section we address each research question individually and, in the process, summarize the included papers.

1.5.1 RQ1: What Are the Characteristics of the Supervised Concept Learning, Classification, and Evaluation Problems and Solutions?

RQ1 is addressed in Paper I, which essentially is an analysis of evaluation methods. However, we first analyze the concept learning problem domain and formalize the problems of learning, classification, and evaluation.

We distinguish between the evaluation of classifiers, algorithms, and algorithm configurations. By an algorithm configuration we mean an algorithm with a particular parameter setting. In contrast, an algorithm evaluation is an overall evaluation of an algorithm over multiple data sets or multiple parameter settings. These three types of evaluation are then used as top nodes in a taxonomy. Each type is then further divided into a general and a specific branch. The first branch denotes methods that can be used independently of which candidates are evaluated, whereas the second branch denotes methods that are tailored for a limited set of candidates or one specific candidate.

We use the formal framework to describe 18 evaluation methods and categorize each method using the taxonomy. The main conclusion is that the framework lets us describe methods from different fields (e.g.: machine learning, statistics, and so forth) with one single terminology and that the taxonomy simplifies the categorization of existing evaluation methods. We hypothesize that this categorization can help in identifying directions for future work since it makes explicit which parts of the taxonomy has few existing methods.

1.5.2 RQ2: What Is the Impact of Learning Algorithm Parameter Tuning and How Can It Be Measured?

RQ2 is addressed in Paper II. In contrast to, what could be perceived as the majority of, other machine learning papers about evaluation this paper focuses on algorithm evaluation instead of algorithm configuration evaluation or classifier evaluation. Specifically, we investigate the impact of learning algorithm parameter tuning. For this purpose, we introduce the concept of Quality Attributes (QAs) to represent evaluation criteria; a QA typically represents a property of interest and can often be evaluated using several different metrics.

We define two QAs, which we refer to as sensitivity and classification performance. Our notions of these QAs are that sensitivity represents the susceptibleness of an algorithm in terms of tuning and classification performance represents the overall perfor-
mance of an algorithm. In order to focus on different aspects of these QAs, we define two metrics for sensitivity (variance and range) and two metrics for performance (average and best). Ideally, these metrics should be calculated from the complete population of configurations. However, for most algorithms the parameter space is too large and thus we approximate the metrics by sampling a suitable number of configurations symmetrically around the default configuration provided by the Weka machine learning workbench.

An experiment, featuring four algorithms and eight data sets, shows that each metric captures a different aspect of its quality attribute. Interesting conclusions can be drawn from using the metrics. For example, a certain algorithm might be sensitive (the impact of tuning is high) but at the same time performing poorly on average. An algorithm can be sensitive in different ways. For example, it might perform extremely well for a particular configuration but otherwise very poorly (the range is wide) or it might perform differently for every configuration (the variance is high). We make the assumption that the average classification performance represents the performance obtained from a standard practice configuration. Our conclusion is then that, for the particular set of algorithms and data sets, parameter tuning seems to be more important than the choice of algorithm since the lowest best performance is higher than the highest average performance. We also conclude that some algorithms are more robust to parameter change than others, i.e., for these algorithms the impact of parameter tuning is lower than for the others.

1.5.3 RQ3: How Can Multiple Criteria Be Considered During Evaluation?

RQ3 is addressed by papers III, IV, and VI. First, we investigate the concept of metric-based evaluation by analyzing a suggested metric function for classifier performance that combines three weighted metrics; subset fit, similarity, and simplicity. Each metric represents a common learning bias: instances should be classified correctly, similar instances should be classified similarly, the representation of the classifier should be as simple as possible. The weight for each metric can be adjusted to capture a suitable trade-off.

In the suggested metric function, the algorithms for calculating similarity and simplicity are only defined for two-dimensional data sets but ideas on how to calculate these metrics for higher dimensions are suggested. Based on these suggestions, we present new algorithms for calculating similarity and simplicity for $n$-dimensional data sets and compare a set of classifiers as a proof of concept. The difficulty of calculating similarity and simplicity is associated with the original idea for these metrics; that they should be captured in an algorithm-independent way. For example: we cannot define simplicity based on the number of tree nodes or the number of neurons because those numbers would only be available for certain classes of algorithms (decision trees and neural net-
The new metrics are calculated by systematically sampling and classifying points in instance space. On the one hand, simplicity is defined as the average number of crossed decision borders when traveling across a fictional grid laid out on the instance space (this is rather simple to imagine in two or three dimensions). Similarity, on the other hand, is defined as the average distance between each correctly classified instance and its closest decision border subtracted by the average distance of each incorrectly classified instance and its closest decision border. Thus, an optimal similarity score would imply no misclassified instances and a maximum margin between each instance and its closest decision border.

As a proof of concept we run the new metric function for the same algorithm configurations and on the same data set used in the original study, yielding corresponding differences between the different classifiers. In addition, we plot the decision space to show graphically what is described by the metrics. We conclude that the new algorithms seem to correspond to the old algorithms, but they work for higher dimensions. Another conclusion is that the computational effort will be too high for using the metrics when evaluating over large data sets (in terms of the number of instances and attributes). However, the overall conclusion is that the metric function can capture multiple aspects of performance that are not revealed during standard accuracy-based evaluation.

In Paper IV, we argue that real-world problems often require us to trade off several criteria but the specific set of important criteria differs across applications. We note that, while there are multi-criteria evaluation methods available, they are usually defined for a specific set of metrics.

We define a generic method for multi-criteria evaluation to be a method that defines how to trade off and combine metrics but leaves the choice of metrics to the user. We then review the literature and find three multi-criteria methods that fit this definition. The three generic methods are analyzed and synthesized into a new generic multi-criteria metric, denoted the Candidate Evaluation Function (CEF). A CEF metric can consist of an arbitrary number of metrics.

Each metric has a weight, which represents its relative importance, and an acceptable range. If the lowest acceptable score is not reached by a candidate (an algorithm or classifier), it will be disqualified (the complete score will be set to zero). If the candidate score for a certain metric is higher than the highest acceptable score, the metric score is set to the maximum value (one). Each metric is normalized according to the best and worst scores obtained by the studied sets of algorithms and data sets. We demonstrate the use of CEF with a case study. The main conclusion is that CEF provides the necessary functionality (weighting, normalization, and ranges) for a general class of evaluation problems. However, the normalization can be performed using the acceptable range in order to get a smooth distribution between zero and one and to avoid skewness.

In paper VI, we present a novel data mining approach to prevent spyware and demonstrate how to benefit from generic multi-criteria evaluation when comparing candidate
algorithms for this particular problem. Our approach is based on the assumption that, to avoid legal repercussions the vendors are required to inform users about associated spyware via the End User License Agreement (EULA) prior to the installation of software. This information is intentionally written in a way that makes it easy to overlook. We therefore investigate the approach to automatically discriminate between legitimate software and spyware associated software by mining EULAs.

We have compiled a collection that consists of 996 EULAs out of which 9.6% are associated with spyware. We convert the collection into two representations; a bag-of-words model and a meta model that includes a set of EULA statistics. We then compare the performance of 17 learning algorithms to that of a majority class baseline algorithm and the results show that most algorithms significantly outperform the baseline. Moreover, we show that the bag-of-words model is significantly better than the meta EULA model, at least for the studied algorithms and the selected evaluation metric. In addition, we suggest an application-specific CEF metric and demonstrate how it can be used to evaluate the studied algorithms.

We conclude that automatic EULA classification can be used to assist users in making informed decisions about aborting or installing applications. In addition, we conclude that CEF can be used to evaluate application-specific trade-offs but appropriate metrics, weights and ranges should be selected systematically.

1.5.4 RQ4: How Can Learning Be Optimized Toward Arbitrary Metrics?

The fourth research question (RQ4) is addressed in Paper V, which focuses on improving performance by modifying the algorithmic bias of learning algorithms. We argue that a major part of the algorithmic bias of existing learning algorithms can be expressed explicitly as a learning metric.

No evaluation metric is superior for all classes of problems and thus the choice of evaluation metrics should be dependent on the application at hand. We therefore assume that this argument also holds for the learning metric. Many studies compare different learning algorithms in order to find the most suitable algorithm for some particular problem. Instead, we examine the relationship between learning metric performance and evaluation metric performance. In other words, we compare variations of a certain algorithmic bias for one particular representational bias instead of comparing different conjunctions of representational and algorithmic biases.

Our comparison is aimed at answering questions, such as: would the performance according to evaluation metric $x$ improve for an algorithm if its learning metric, $y$, is replaced with $x$. For the purpose of our investigation, we develop A Metric-based One Rule Inducer (AMORI), which uses CEF as the learning metric. This means that any single metric, or combinations of metrics, that can be defined as a CEF metric can be used as part of the algorithmic bias.
We compare three instances of AMORI, each using a particular CEF metric. In addition, the AMORI instances are compared to two standard rule inducers (Ripper and One Rule). Each CEF metric is based on a common trade-off: accuracy, the F-measure, and AUC. The experimental results, obtained from 19 UCI data sets, show that each AMORI instance perform better than the two other instances and One Rule if its learning metric is also used for evaluation. Meanwhile, Ripper has the ability to generate sets of rules and also performs post-processing. Thus, we expected Ripper to be very competitive. However, Ripper did not significantly outperform any of the AMORI instances on their respective metrics.

1.5.5 RQ5: How Can Evaluation Metrics Be Systematically Selected for Different Applications?

The last research question (RQ5) is addressed in Paper VII, which elaborates on and further motivates the use of generic multi-criteria methods but is primarily aimed at developing a more systematic approach to selecting relevant metrics for the application at hand.

Many studies compare different metrics for evaluating the same criteria, e.g., performance. However, we argue that it becomes difficult to select appropriate metrics for an application if it is not clear which are the relevant criteria. We therefore suggest a framework for metric selection. In this framework, we break down the process of evaluating candidates, e.g., algorithms or classifiers, for a particular application into a set of subsequent steps.

The first step is to describe the application by identifying relevant quality attributes (evaluation criteria). This process can be divided into two sub steps; (i) use a taxonomy of generic quality attributes as inspiration to find relevant attributes, and (ii) identify additional quality attributes that may be more specific to the application. The next step is to assess the importance of quality attributes, i.e., to prioritize them. The list of prioritized attributes can then be used as a basis for conducting the next step; to find suitable metrics for each attribute. The identified metrics should then be assigned weights and acceptable ranges. The weight for each metric represents the importance of the attribute it evaluates. The acceptable ranges should be established based on the application objectives or requirements. For example, accuracy should be over 70% but we do not care about improvements over 90%.

The last step is to perform multi-criteria evaluation to assess a candidate or to rank several candidates. Based on the framework, we suggest how the Goal/Question/Metric (GQM) method can be used in conjunction with pairwise comparison and CEF as an approach to metric selection. We conclude that, by using the concept of quality attributes, we can determine suitable metrics and metric weights by using methods that are applied in the area of software engineering, e.g., such as: pairwise comparison, GQM, and the Analytic Hierarchy Process.
1.6 Conclusions

Evaluation methods are essential for machine learning research and its applications. However, the motivations and requirements may vary. In research the evaluation can be more focused towards the testing of hypotheses about algorithm performance or the investigation of properties such as algorithm complexity and learnability, whereas for many applications the evaluation is more focused toward reliable performance estimates. For example, this applies to decision support systems for which subsequent decisions have serious consequences. The reliability of results is always important but even if an accuracy score is reliable it still measures only one property of performance.

The primary purpose of this thesis was to analyze current approaches to evaluate classifiers as well as supervised concept learners and to explore possible improvements in terms of alternative or complementary approaches. In seven separate studies we have explored different aspects pertaining to this purpose. We have used a metric-based approach, that is, we have focused on how to represent application objectives and important trade-offs by evaluating or learning with appropriate metrics. Our work demonstrates several benefits with multi-criteria metrics. It has been argued that there are some problems with using multi-criteria metrics, e.g., that the setting of weights is often ad hoc.

However, we argue that such issues can be avoided or partly resolved by using a systematic method for metric selection and evaluation configuration. In addition, we conclude that multi-criteria metrics can be used as learning metrics, i.e., as part of the algorithmic bias in order to capture application relevant trade-offs during the learning phase. The interpretation of multi-quantity results of an evaluation (e.g.: a ROC curve) is subjective. The multi-quantity results may provide more information than single-quantity approaches but they often require sound judgment. Thus, some might require a human expert in order to be interpreted correctly. Many of the existing supervised learning algorithms are designed to interpret single quantities to rank classifiers based on data, thus replacing their performance element with a multi-quantity evaluation method could be very difficult or even impossible.

1.7 Future Work

We divide the future work into three subject categories; metric-based evaluation, metric-based learning, and application-specific metric selection.

1.7.1 Metric-based Evaluation

The evaluation method taxonomy can be refined by adding branches that separate multi-criteria from single-criteria methods. Moreover, the multi-criteria branch can be further divided into: single-quantity and multi-quantity methods. In addition, the categorization
of evaluation methods can be extended with more methods and a more distinct separation between method and metric can also be established. 

In order to increase the precision of the sensitivity and classification performance metrics, a better configuration sampling technique can be employed. It would also be interesting to perform in-depth studies of algorithms that are notoriously difficult to configure due to their vast parameter space (e.g.: back-propagated neural networks). Additional metrics for sensitivity can also considered. For example: one plausible metric is the average performance difference between adjacent configurations. This metric is related to the assumption that smooth distributions would simplify hill-climbing-based optimization of the particular algorithm.

We believe that learning algorithm independent metrics for common learning biases, such as similarity and simplicity, should be studied more deeply. The algorithms proposed for calculating these metrics are computationally intensive, thus we can identify two basic directions for future work: (i) to develop more efficient algorithms or alternative approximations, and (ii) to further assess the scientific value of the proposed metrics and the present way in which they are calculated.

### 1.7.2 Metric-based Learning

In order to further establish whether or not it is beneficial to use identical learning and evaluation metrics, future work on metric-based learning should compare a more diverse set of metrics. AMORI was compared using accuracy, the F-measure, and AUC, which are all based on the same basic trade-off. Another direction that seems promising, is to compare different representational biases using identical learning metrics. We hypothesize that this would make it possible to measure the impact of metric-based learning for different conjuctions of representational biases and learning metrics.

In addition, more research is required in order to understand how to select an appropriate learning metric based on a particular problem. As observed in Paper V, some learning metrics seem to be more suitable than others for certain data sets. A preliminary step in this direction could be to establish metrics for some of the widely used data sets (e.g.: from the UCI machine learning repository). These metrics can be defined to promote behaviors desirable for typical applications related to each data set. For example: a heart disease data set can be associated with a cost-based metric that punishes undetected presence of cardiac disorder more severely than the false detection of disorder. These data set metrics could be used as benchmark evaluation metrics when comparing different algorithms and, related to metric-based learning, they could also be used as learning metrics.

Another direction is to employ the multi-criteria functionality of the CEF metric to be able to trade-off several metrics during the learning phase of AMORI.
1.7.3 Metric Selection

We suggest the development of a machine learning oriented quality attribute taxonomy to facilitate the mapping from application objectives to relevant evaluation metrics. This taxonomy can, for example, be based on the software engineering ISO standard\(^2\) list of quality attributes. In addition, we hypothesize that the terminal nodes of such a taxonomy could be used to describe suitable metrics. Another direction is to investigate various systematic approaches to prioritize quality attributes.

We also believe that it would be possible to develop a software-based tool for application-oriented evaluation. This tool should preferably be compatible with a suitable software library of machine learning algorithms, e.g. the Weka machine learning workbench\(^3\), to enable the evaluation of common algorithms and the processing of standard formatted data sets.

---

\(^2\)ISO/IEC FDIS 9126-1:2000 (E)

\(^3\)Weka (Witten & Frank, 2005) is a machine learning experimentation environment written in Java and it is publicly available from [http://www.cs.waikato.ac.nz/ml/weka](http://www.cs.waikato.ac.nz/ml/weka)
2.1 Introduction

In supervised learning, the objective is to generalize from training instances defined by a set of inputs and the correct output (Russell & Norvig, 2003). Some supervised learning problems involve generating a classifier. In these problems, the data instances belong to different classes and the task of a learning algorithm is to find out how to classify new, unseen instances correctly. One example of a supervised classifier learning problem is that of finding out how to classify unseen instances of three types of Iris plants by observing descriptions of 150 already classified plants (50 of each type) (Anderson, 1935). The data given in this case are the petal width and length, as well as the sepal width and length, of each plant.

Research in machine learning and related areas has resulted in the development of many different supervised learning algorithms for classification problems but the no free lunch theorems state that no algorithm can be superior on all problems with respect to classification accuracy (Wolpert, 2001). Hence, when presented with some particular
learning problem it would be wise to select an appropriate algorithm for that problem. This type of selection is usually performed by evaluating a number of candidate algorithms or classifiers. There exist a large number of evaluation methods and the question of when to use which method is still open.

In this study we analyze 18 evaluation methods for learning algorithms and classifiers and show how to discriminate between these methods with the help of an evaluation method taxonomy which is based on several criteria. We also define and apply a formal framework to ensure that all methods are described using the same terminology even though they may have been originally introduced in various areas of research.

2.1.1 Related Work

We have not been able to identify any studies dedicated to evaluation methods for learning algorithms and classifiers, which is the motivation for this study. There exists work that is related to supervised learning evaluation methods but does not consider both algorithm and classifier evaluation. Some of the studies focus on the evaluation of classifiers and compare different metrics, cf., Andersson et al. (1999); Caruana and Niculescu-Mizil (2004); Nakhaeizadeh and Schnabl (1997).

In the research area of mathematical statistics there has been a number of studies that have compared metrics for model evaluation, which is a term often used in the machine learning community and is ambiguously applied to address both algorithm and classifier evaluation. Mathematical statistics has contributed much to the research of evaluation metrics, cf., Akaike (1973); Murata, Yoshizawa, and Amari (1994); Raftery (1986); Schwartz (1978); Sugiyama and Ogawa (2001). However, the majority of these metrics are useful only in evaluating regression functions and often not applicable to classifiers, i.e., discrete functions.

In the area of experimental psychology, Myung and Pitt (2002) describe how criteria from the philosophy of science can be applied in model evaluation. Criteria that are categorized by Myung as qualitative are: explanatory adequacy, interpretability, and faithfulness; and quantitative criteria that could be used are: falsifiability, goodness of fit, complexity, and generalizability. Other related work shift focus from evaluation methods to the statistical validity of experimental comparisons of classifiers and algorithms (Dietterich, 1998; Salzberg, 1997) and finally, there exists work in machine learning that uses evaluation methods as a fundamental part of experiments but in which the main focus is understanding or enhancing learning algorithms, cf., Giraud-Carrier et al. (2004); King et al. (1995); Vilalta and Drissi (2002).

2.1.2 Outline

In Section 2.2 we specify the general framework which we will use to describe all evaluation methods in a uniform manner. Section 2.3 then introduces the evaluation method
taxonomy. This is followed by presentations of all featured evaluation methods in Section 2.4. The last section includes results and conclusions.

2.2 Framework

To describe the different evaluation approaches consistently, we decided to use a general framework of formal definitions. As no existing framework was found suitable for this purpose we had to develop such a framework. Of course, it has some similarities with existing frameworks and ways to describe supervised learning formally, (cf., Mitchell, 1997b; Russell & Norvig, 2003). Given a particular supervised learning problem we have that:

**Definition 1.** An instance, $i$, is an $n$-tuple defined by a set of attributes, $A = \{a_1, a_2, \ldots, a_n\}$, where each attribute, $a_i$, can be assigned a value, $v \in V_i$ and $V_i$ is the set of all possible values for attribute $a_i$.

We now define:

**Definition 2.** The instance space, $I$, is the set of all possible instances (given a set of attributes and their possible values).

The number of dimensions, $n$, of $I$ is equal to the number of attributes that defines each instance, and so:

$$I = V_1 \times V_2 \times \cdots \times V_n .$$

(2.1)

Instances can be labeled with a class, $k$, and this labeling effectively partitions the instance space into different groups of instances. Let $K$ be the set of all valid classes for instances from $I$.

**Definition 3.** A classifier, $c$, is a mapping, or function, from instances, $i \in I$, to classes, $k \in K$ (Fawcett, 2003):

$$c : I \rightarrow K .$$

(2.2)

**Definition 4.** The classifier space, $C$, is the set of all possible classifiers, i.e., all possible mappings between $I$ and $K$.

Using the instance and classifier spaces, we now identify three separate problems related to supervised learning: classification, learning, and evaluation. We begin by formulating the classification problem and define the classified set; a set containing a selection of instances from $I$ paired with a classification for each instance. Using the definition of classified sets we then formulate the learning problem and discuss inductive bias and its relation to $C$. Finally, we define algorithm evaluation, algorithm configuration evaluation, and classifier evaluation.

---

1We assume that each $V_i$ is a finite set but the framework as such is not limited in theory to this assumption.
2.2.1 The Classification Problem

In the last section we defined classifiers and the classifier space. Let us now define classification:

**Definition 5.** Classification is the assignment of a class, $k \in K$, to an instance, $i \in I$.

The goal is of course to associate an instance with the correct class. However, we need to discriminate between three different types of classification: true, data set, and classifier classification. The true classification is always correct but often exists only in theory. The data set classification is usually assumed to be correct and a number of data set classifications is typically used as input to learning algorithms to generate classifiers. The third type is the classification made by a classifier. It is commonly compared to the data set classification to measure the accuracy of the classifier.

Given that we have a classifier, $c$, the three types of classification can be expressed as:

$$c(i) = k \quad \text{(2.3)}$$

$$c^*(i) = k \quad \text{(2.4)}$$

$$c^T(i) = k \quad \text{(2.5)}$$

where $T$ is a classified set:

**Definition 6.** A classified set of instances is a set, $T$, of pairs, such that each pair consists of an instance, $i \in I$, and a class, $k$:

$$T = \{ <i, k> \mid i \in I, k \in K \} \quad \text{(2.6)}$$

2.2.2 The Learning Problem

Using the definitions provided in the last section, the learning problem can now be formulated as follows:

**Definition 7.** Given a classified set of instances, $T$, we say that learning is the process of selecting a classifier from $C$.

Usually, $T$ is very small compared to $I$. In addition, $C$ is often too large to search through exhaustively and thus it needs to be restricted by some means. Without prior assumptions about the learning target, or inductive bias, a learner has no rational basis

---

2 This process is sometimes called model selection. The term, originating from statistics, is frequently but also ambiguously used in the machine learning community. Consequently, it is often followed by some definition, e.g., “the objective is to select a good classifier from a set of classifiers” (Kohavi, 1995), “a mechanism for [...] selecting a hypothesis among a set of candidate hypotheses based on some pre-specified quality measure” (Ratsaby, Meir, & Maiorov, 1996).
for classifying new, unseen instances (Mitchell, 1997b). Inductive bias can be viewed as a rule or method that causes an algorithm to choose one classifier over another (Mitchell, 1980). The inductive biases of learning algorithms can be divided into two categories; representational and procedural. The representational bias defines which subset of $C$ that the algorithm considers and the procedural bias determines the order of traversal of $C$ when searching for, or selecting, a classifier in that subspace (Gordon & Desjardins, 1995). Formally we say that:

**Definition 8.** The bias of an algorithm with configuration $a$ and the input of $T$ defines $C_a \subset C$.

Inductive bias, as a concept, is related to the conservation law of generalization performance (Schaffer, 1994) and the no free lunch theorems for supervised learning (Wolpert, 2001). The latter tells us that “for any two learning algorithms, there are just as many situations […] in which algorithm one is superior to algorithm two as vice versa”. This implies that no matter what kind of biases are used in two different algorithms, they will perform equally well averaged over all possible tasks of learning. However, in most real-world cases, the interest is in finding out how suitable an algorithm is for a particular problem or a limited set of problems. Thus, for these cases the inductive biases of learning algorithms has to be considered.

### 2.2.3 The Evaluation Problems

Having formulated the learning and classification problems, we now concentrate on the evaluation problems. Evaluation is used to determine how appropriate an algorithm or a classifier is for a particular problem. We formulate the classifier evaluation problem as follows:

**Definition 9.** Given a classifier, $c \in C$, and a classified set, $T$, assign a value representing its performance, $v$, according to one or more evaluation metrics. We can express this as a function:

$$e(c, T) = v$$  \hspace{1cm} (2.7)

There are many different types of criteria that can be used for classifier evaluation but the two most common types are goodness of fit and complexity (Myung & Pitt, 2002). The goodness of fit of classifiers is typically measured as the fraction of misclassified instances (Equation 2.9) on a particular data set. The misclassification cost function called 0-1 loss (Equation 2.10) awards 1 to each incorrect classification and 0 to a correct one. The sum of misclassifications is then divided by the total number of classified instances. It is also common to express the goodness of fit as accuracy by using Equation 2.8.

$$\beta(c, T) = 1 - \alpha(c, T) \hspace{1cm} \text{where}$$  \hspace{1cm} (2.8)
Error: $\alpha(c, T) = \frac{1}{|T|} \sum_{i \in T} \delta(i^k, c(i))$ and

$$0\text{-}1 \text{ loss: } \delta(c^T(i), c(i)) = \begin{cases} 1 & c^T(i) \neq c(i) \\ 0 & \end{cases} . \quad (2.10)$$

Two key issues concerning the goodness of fit criterion are estimation bias and estimation variance (Mitchell, 1997b). The goodness of fit on training data is usually a poor estimate of the goodness of fit on new, unseen instances (Witten & Frank, 2005). $T$ is often used to define two subsets; the training set, $T_s \subset T$, and the test set, $T_t \subset T$. If evaluation is performed using instances from the training set ($T_s = T_t$), the estimate will often be optimistically biased. In order to get a less biased estimate, the evaluation must be performed on instances that were not used for training ($T_s \cap T_t = \emptyset$).

The 0-1 loss function assumes equal misclassification costs but for many real-world problems there is a huge difference in cost associated with different misclassifications. For instance, failing to detect an environment-threatening oil slick is a far greater cost than a false alarm (Witten & Frank, 2005). An alternative metric, the TP/false alarm rate, can be used instead of the 0-1 loss function to separate between instances that are correctly classified as belonging to a class and instances that are false alarms. This is further discussed in Section 2.4.1.

The goodness of fit is usually straightforward to use and it is by far the most common criterion by which classifiers are evaluated. Complexity as an evaluation criterion, on the other hand, is harder to explain because it can be measured in many different ways. For instance, it can be defined as the number of nodes in a decision tree (Andersson et al., 1999) or the number of basis functions for a support vector machine (Keerthi, Chapelle, & DeCoste, 2006). Complexity can also be defined more generally so that it might be used in comparisons between classifiers of different type.

Some evaluation methods highlight a trade-off between complexity and the goodness of fit, (cf., Akaike, 1973). This is based on the bias plus variance formula (BV), which concerns the relationship with classifier complexity and the estimated accuracy on new, unseen instances. According to BV, a classifier of too high complexity may score well on training data but will vary much in its classifications of test data (low bias, high variance) as opposed to a too simple classifier, which will be more consistent in its predictions but might not be able to partition the instance space as detailed as that of $c^*$, thus it will always have an unacceptably high error on test data (high bias, low variance).

Many evaluation methods tend to favor simpler classifiers as opposed to more complex ones on the basis that they are more prone to overfitting3, (cf., Rissanen, 1978). However, Domingos (1999) questions that a complex classifier automatically leads to overfitting and less accuracy and argues that the success of classifier ensembles shows

---

3In terms of our framework, an overfitted classifier scores a very high accuracy on $T_s$ but it has an unacceptably low accuracy when classifying other instances. The goal is to find a classifier with an acceptable accuracy for typical instances of the problem domain.
that large error reductions can systematically result from significantly increased complexity, since a classifier ensemble is effectively equivalent to a single classifier, but a much more complex one. Moreover, in one study about using overfitting avoidance as bias and its relation to prediction accuracy it is argued that overfitting avoidance as a bias may in fact lead to less accuracy (Schaffer, 1993).

We now leave the discussion about classifier evaluation and focus on algorithm evaluation. Learning algorithms are often evaluated indirectly by evaluating a number of generated classifiers. To add to the complexity of evaluation, it is very common for a learning algorithm to have a number of configuration parameters which may affect the classifier selection process, e.g., the number of neurons and layers of a back-propagation neural network. Taking this into account, we may formulate the algorithm configuration evaluation problem as follows:

**Definition 10.** Given a learning algorithm with a particular parameter configuration, \( a \in A \), and a classified set, \( T \), assign a value representing its performance, \( v \), according to one or more evaluation criteria:

\[
e(a, T) = v .
\]  
(2.11)

A more general way would be to consider all configurations of an algorithm but it is quite unusual. We define the algorithm evaluation:

**Definition 11.** Given a learning algorithm with a set of parameter configurations, \( A \), and a classified set, \( T \), assign a value representing its performance, \( v \), according to one or more evaluation criteria:

\[
e(A, T) = v .
\]  
(2.12)

### 2.3 Evaluation Method Taxonomy

We are about to specify an evaluation method taxonomy which, together with the formal framework, have three evident purposes. It should (i) simplify finding candidate evaluation methods for a particular evaluation problem, (ii) help in making fair comparisons of candidate evaluation methods, and (iii) present the state-of-the-art in learning algorithm and classifier evaluation thus simplifying the identification of areas in need of research.

At the highest level of our taxonomy, which can be viewed in Figure 2.1, we address the type of evaluation and discriminate between classifier evaluation, algorithm configuration evaluation, and algorithm evaluation. This is partially inspired by the taxonomy of statistical questions within the machine learning area of research that is proposed by Dietterich (1998) and in which a discrimination is being made between classifier and algorithm evaluation.

We then discriminate between general or specific methods in terms of the range of application: a general classifier evaluation method can be used for most or all types of
classifiers whereas a specific method is only applicable to a limited number of classifier types or perhaps only one singular type. For instance, one method might only measure the goodness of fit making it highly generic, while another method uses a complexity measure which is dependent of the underlying representation of the classifier, e.g. networks, making it much more specific.

In addition to the taxonomy we also present a list of performance metrics, or criteria, in Table 2.1.

### 2.4 Evaluation Methods

We now present the featured evaluation methods in alphabetical order using the definitions from our framework. The presentation is divided into three subsections, one for each evaluation target.
2.4.1 Classifier Evaluation

The Data Envelopment Analysis (DEA) (Nakhaeizadeh & Schnabl, 1997) evaluation method is used for multi-criteria classifier evaluation. The idea is to separate different performance criteria into a positive and a negative group. The positive group should include all criteria for which a high score is preferable and vice versa. Each criterion is also weighted according to its relevance to the problem at hand and this weighting is done objectively by an optimization algorithm. The total positive score is divided with the total negative score and the result is called the efficiency score. In one example (Nakhaeizadeh & Schnabl, 1997) the DEA method is used with two negative metrics, training and testing time, and one positive metric, accuracy. However, the DEA method is not restricted to these specific measures.

The Holdout (HLD) (Duda, Hart, & Stork, 2000) method is a general method for evaluating classifiers using goodness of fit as a performance criterion. The data set, $T$, is divided into two subsets, training and testing. A classifier is generated using the training data and evaluated using 0-1 loss on the testing data. An alternate version is to divide the data into three subsets by adding the validation set to the two already mentioned subsets. The applied learning algorithm is used on the training data until it has generated a classifier that scores well on the validation data. The classifier is then evaluated on the testing data. The holdout method is closely related to cross-validation techniques, which are nothing more than combinations of multiple holdout evaluations.

Measure-based evaluation (MBE) (Andersson et al., 1999) is a general classifier evaluation method without pre-decided performance criteria. The idea is to define a measure function of classifier performance by combining a set of weighted criteria relevant to a particular problem.

The concept of measure functions for generalization performance allows for the definition of learning problems as computational problems. Measure-based evaluation provides a way to evaluate certain qualitative properties of classifiers, unlike more common methods which only performs a quantitative evaluation by means of accuracy calculation. Some related studies also make use of the geometric or measurable qualities of the classifier and instance spaces. One such study proposes geometric strategies for detecting overfitting and performing robust classifier selection using simple metric-based intuitions (Schuurmans, 1997).

An example measure function (MF) has been suggested (Andersson et al., 1999) and it is this example that is featured in the categorization of evaluation methods in this study. MF is based on three criteria: subset fit (training accuracy), similarity and decision space complexity.

Each criterion is weighted depending on the problem and the similarity measure is divided into one negative and one positive group depending on if the calculation was performed on an incorrectly or correctly classified instance. The two groups can be weighted differently. It is argued that these three criteria capture the inherit biases found in most learning algorithms. Equation 2.13 describes the example measure function. $a_0$, 

27
\(a_1\) and \(a_2\) are the weights of subset fit, i.e., accuracy (see Equation 2.8), similarity (\(\text{simi}\)) and simplicity (\(\text{simp}\)) respectively. \(k_1\) and \(k_2\) are the weights of the similarity measure for correctly classified instances (\(\text{simi}_1\)) and incorrectly classified instances (\(\text{simi}_2\)) respectively. As already mentioned, the example measure function uses the total length of the decision borders as a measure of simplicity. A low value (indicating a simple partitioning of the decision space) is typically wanted and that is why \(\text{simp}\) is subtracted from the subset fit and similarity measures.

\[
\text{mf} = a_0\beta(c, T) + a_1(k_1\text{simi}_1 + k_2\text{simi}_2) - a_2\text{simp}.
\]  

(2.13)

The Minimum Description Length principle (MDL) (Rissanen, 1978) is an approximate measure of Kolmogorov complexity (Kolmogorov, 1965), which defines simplicity as “the length of the shortest program for a universal Turing machine that correctly reproduces the observed data” (Russell & Norvig, 2003). MDL is based on the insight that any regularity in the data can be used to compress the data, i.e., describing it using fewer symbols than the number of symbols needed to describe the data literally (Witten & Frank, 2005). MDL is claimed to embody a form of Ockham’s Razor (Tornay, 1938) as well as protecting against overfitting (Grünwald, 2005). It is argued that there is no guarantee that MDL will choose the most accurate classifier (Domingos, 1999) when used as a basis for selection. Furthermore, some researchers note that it is hard to use MDL in practice since one must decide upon a way to code the so-called theory, i.e., the description, so that the minimum description length can be calculated (Witten & Frank, 2005) and this coding has to be done differently for each algorithm. Consequently, MDL is a specific classifier evaluation method with complexity as the only performance criterion.

Receiver Operating Characteristic (ROC) analysis (Egan, 1975) was first used in signal detection theory and later introduced to the machine learning community (Provost & Fawcett, 1997). It is a graphical technique that can be used to evaluate classifiers or algorithm configurations. For a two-class prediction \(K = \{k_1, k_2\}\), where one \(k\) is chosen as the target, there are four possible outcomes: an instance can be correctly classified as either belonging to the target \(k\) (true positive) or not belonging to the target \(k\) (true negative), or it can be incorrectly classified as belonging to the target \(k\) (false positive) or not belonging to the target \(k\) (false negative). The two kinds of error (false positives and negatives) can have different costs associated with them. In order to plot a ROC curve the featured instances must be ranked according to the probability that they belong to a particular class.

Most classifiers only output information about the predicted class, not the probability that an instance belongs to a particular class. Because of this fact, we label ROC analysis as a specific evaluation technique. The ROC curve is plotted by starting from origo, reading from the ranking list (starting at the highest ranked \(i \in I\)), and moving up for each true positive and right for each false positive. The vertical axis shows the percentage of true positives and the horizontal axis shows the percentage of false positives (Witten & Frank, 2005). Lift charts, a very similar technique, displays the number of true positives...
on the vertical axis and the sample size on the horizontal axis.

A ROC plot, cannot be used as an evaluation method in the sense described in Section 2.2.3, since it does not return a value indicating how good a classifier is. However, the area under the ROC curve (AUC) can be used for this purpose, i.e. to measure \( v \). (cf., Hand & Till, 2001; G. M. Weiss & Provost, 2001). As with ROC plots, the standard AUC can only be used for problems where \(|K| = 2\), however it has been generalized to work for problems where \(|K| \geq 2\) (Hand & Till, 2001).

Comparisons have been made between ROC/AUC and other evaluation methods (Provost & Fawcett, 1997; Provost et al., 1998). In one such study (Provost et al., 1998) it was argued that accuracy-based evaluation of machine learning algorithms cannot be justified, since accuracy maximization assumes that the class distribution is known for \( I \). However, for benchmark data sets it is often not known whether the existing distribution, i.e., the class distribution of \( T \), is the natural distribution (the class distribution of \( I \)). Also, accuracy estimation does not take cost into account and often the cost of misclassifying different \( k \) is not equal. A significant example would be that of learning to predict if a patient suffers from a life threatening disease or not, e.g., misclassifying a healthy patient as ill is perhaps not as bad as misclassifying an ill patient as healthy. Accuracy-based evaluation does not take this perspective into account.

The Squared error, Accuracy and ROC area (SAR) (Caruana & Niculescu-Mizil, 2004) evaluation method is another multi criteria method for classifier evaluation. The method is based on an experiment in which ten performance metrics were used to evaluate classifiers generated by a large number of algorithms and data sets. The metrics were compared in terms of correlation in metric space and the three most consistent and well-behaved metrics were combined into one evaluation function. The SAR method combines two goodness of fit measures with the area under the ROC curve. The last metric makes the method specific in that it only works for classifiers which can output.

The Vapnik-Chervonenkis Bound (VCB) (Vapnik, 1995; Vapnik & Chervonenkis, 1971) is a specific classifier evaluation method based on an algorithm complexity metric, the VC dimension, and the goodness of fit of a classifier selected by that algorithm (Burges, 1998). The VC dimension, for an algorithm with a particular configuration \( a \), is a measure of its so-called capacity. The VC dimension has been defined for different learning algorithms since it depends on the inductive bias, which often differs between algorithms and configurations. The VC dimension of an algorithm \( a \), is related to the ability of the classifiers of \( C_a \) to shatter instances (Mitchell, 1997b), i.e., the size of \( C_a \) compared to \( C \). Shattering is the process of discriminating between instances, i.e., assigning different classes to different instances. The VC dimension of an algorithm is thus the size of the largest subset of \( I \) that can be shattered using its classifiers (Vapnik, 1995). The VC dimension itself is a specific method for evaluating algorithms using a complexity criterion.
2.4.2 Algorithm Configuration Evaluation

There exist a number of methods for measuring the goodness of fit of algorithm configurations, e.g., cross-validation (CV) (Stone, 1974), jackknife (JK) (Quenouille, 1956), and bootstrap (BS) (Efron & Tibshirani, 1993; Jain et al., 1987). JK is a special case of CV (Duda et al., 2000) and will only be briefly mentioned in that context.

A CV test is prepared by partitioning $T$ into $n$ number of subsets ($T_1, T_2, \ldots, T_n$), where each subset is called a fold. For each $n$, training is performed on all subsets except one, e.g., $T_s = T_1 \cup T_2 \cdots \cup T_{n-1}$. The omitted subset is used for evaluation of the selected classifier, $c$, e.g. $T_t = T_n$. The training and evaluation steps are repeated until all subsets have been used for evaluation once. The special case for which $n = |T|$ is called leave-one-out, n-fold CV, or jackknife. One of the most common CV configurations is the stratified 10-fold CV (Witten & Frank, 2005). Stratification ensures that each class is properly represented in each fold with respect to the class distribution over $T$, i.e., if 30% of $T$ belong to class $k$, each subset should consist of roughly 30% class $k$ instances.

BS is based on the statistical procedure of sampling with replacement. Preparation of a BS evaluation is performed by sampling $n$ instances from $T$. Contrary to CV, the same instance can be selected several times. In one particular setup, 0.632 bootstrap (Witten & Frank, 2005), instances are sampled $|T|$ times to create a training set, $T_s$. If $T$ is reasonably large, it has been shown that $|T_s| \approx 0.632|T|$. The instances that were never picked/sampled are put into the test set, $T_t$.

CV and BS have been studied extensively and the two methods have been compared with the main conclusion that 10-fold CV is the recommended method for classifier selection (Kohavi, 1995). It has been has shown that leave-one-out is almost unbiased, but it has a high estimation variance, leading to unreliable estimates (Efron, 1983). In related work, 0.632 BS and leave-one-out have been compared by Bailey and Elkan, and the experimental results contradicted those of earlier papers in statistics which advocated the use of BS. This work also concluded that BS has a high bias and a low variance, while the opposite holds for cross-validation (Bailey & Elkan, 1993). CV is one of the most popular methods for evaluating algorithm configurations (Witten & Frank, 2005) and it is often used in conjunction with classifier evaluation methods. For instance, CV has been used instead of holdout to produce ROC curves, using techniques such as pooling (Bradley, 1997) and averaging (Swets & Pickett, 1982), with the objective of making them more reliable in the sense that the result does not depend as much on a particular training/test set partition.

2.4.3 Algorithm Evaluation

Lavesson and Davidsson (2006) have proposed Algorithm Sensitivity (SEN) and Algorithm Classification Performance (PER) as algorithm evaluation methods. SEN tries to measure the impact of parameter tuning, i.e., how much the performance of an algorithm can be affected by its configuration, and PER tries to measure the overall accuracy of an
algorithm. The metrics used by SEN and PER depend on the evaluation of all possible configurations of an algorithm to cover as many classifiers as possible from $C$. In most practical cases this is impossible and an estimation is therefore suggested which can be performed by selecting a subset of the configurations symmetrically around a known default configuration, e.g., determined by meta-learning techniques. Each configuration is evaluated using 10-fold cross-validation, however, the metrics as such are not limited to this particular method.

2.5 Conclusions

Evaluation is fundamental to machine learning (Mitchell, 1997b) and a key to making real progress in data mining (Witten & Frank, 2005). Nevertheless, it is hard to find any studies dedicated to the comparison of evaluation methods. There exists work that is related to supervised learning evaluation methods but does not consider both algorithm and classifier evaluation and when comparing work from, e.g., statistics and machine learning one realizes that different terminologies are used. As a consequence, it is hard to compare methods from various areas of research and it can even be difficult to find out if it is meaningful to compare two methods, i.e., it is unclear if the methods address the same problem. In this study, we have defined and applied an evaluation method taxonomy (shown in Figure 2.1) and a formal framework (Section 2.2). The taxonomy helps to discriminate between methods in terms of the target they address, the prerequisites and possible dependencies they have, and the criteria with which they use to measure performance. The formal framework is used to present and discuss methods using a singular terminology, thus enabling comparison and deeper understanding of evaluation methods from different fields of research including experimental psychology, machine learning, and statistics.

Together, the framework and taxonomy should (i) simplify finding candidate evaluation methods for a particular evaluation problem, (ii) help in making fair comparisons of candidate evaluation methods, and (iii) present the state-of-the-art in learning algorithm and classifier evaluation thus simplifying the identification of areas in need of research.

We have analyzed 18 evaluation methods for learning algorithms and classifiers using the framework and categorized them according to the criteria defined in the evaluation method taxonomy. The results of the categorization can be viewed in Table 2.2. It is evident that very few methods evaluate more than one configuration of an algorithm. Our guess is that, in most real world cases, the evaluation of multiple configurations is still performed by running the evaluation method of choice for each configuration. This is why our taxonomy distinguishes between algorithm evaluation and algorithm configuration evaluation. Furthermore, the results support the claim that goodness of fit is the most common metric.
<table>
<thead>
<tr>
<th>Evaluation Method</th>
<th>Target Metric used</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm Classification Performance (PER)</td>
<td>Vapnik-Chervonenkis Bound (VCB)</td>
<td>G</td>
</tr>
<tr>
<td></td>
<td>Vapnik-Chervonenkis Dimension (VCD)</td>
<td>G</td>
</tr>
<tr>
<td>ROC Analysis (ROC)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Minimum Description Length (MDL)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Measure-based Evaluation (MBE)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Lift Chart</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Jackknife (JK)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Holdout (HLD)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Data Envelopment Analysis (DEA)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>CV ROC</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>CV Lift Chart</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>CV AUC</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Cross-validation (CV)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Boosting</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Area Under the ROC Curve (AUC)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Algorithm Sensitivity (SEN)</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Algorithm Classification Performance (PER)</td>
<td></td>
<td>S</td>
</tr>
</tbody>
</table>

Table 2.2: Evaluation Methods
Quantifying the Impact of Learning Algorithm Parameter Tuning

Niklas Lavesson & Paul Davidsson

3.1 Introduction

It has been argued that many of the well-known learning algorithms often generate classifiers of comparable quality (Mitchell, 1997b; Weideman, Manry, Hung-Chun, & Wei, 1995; Witten & Frank, 2005). However, the relation between algorithm parameter tuning and classifier performance is seldom discussed.

In order to solve a learning task there is typically only a limited amount of resources available and different situations put emphasis on different requirements. For instance, when the time available for preparation is short, a robust learning algorithm may be useful, capable of generating a reasonably good classifier with a minimum amount of parameter tuning. For other problems it could be more important to select an algorithm capable of generating a classifier with the best classification accuracy possible (using careful parameter tuning), with the possible drawback of getting worse results if less care is taken when setting the parameters.

We present a systematic comparison between four algorithms. One rationale behind this study is to investigate the impact of the configuration of algorithms on classifier performance. Instead of just finding the algorithm that generates the most accurate classifier
on some data, it may be important to look at the variation of performance, or sensitivity of a learning algorithm (sensitivity is here seen as the inverse of robustness). Obviously this is a complex issue and it has been argued that, theoretically, no algorithm is superior on all possible induction problems (Schaffer, 1994; Wolpert, 2001). In fact, it is argued that any particular configuration of an algorithm is as good as any other taken over all possible problems. In order to investigate this issue, quantifiers of performance as well as more systematic ways to compare algorithms and algorithm configurations are needed. This paper presents two quality attributes for algorithm assessment; sensitivity and classification performance. As this study will show, these two attributes reflect different aspects of quality related to learning problems and there exist several possible candidate metrics for assessing the quality attributes.

In the next section, we describe the quality attributes and the metrics used to assess them. Then the experiment procedure and results are presented and this is followed by a discussion about quality attributes and how they may be used for algorithm selection in a systematic way. The last sections feature related work as well as conclusions and pointers to future work.

### 3.2 Quality Attribute Metrics

We now introduce sensitivity and classification performance as learning algorithm quality attributes and present formal definitions of two metrics for each attribute. The metrics for classification performance capture the average and best performance and the metrics for sensitivity capture the average impact, and the degree of impact, of parameter tuning.

Given a learning algorithm $a$, let $C_a$ be the set of all possible parameter configurations and $c_a \in C_a$ a particular configuration. In addition, $D$ is a set of known instances and $f$ an evaluation function $f(c_a, D)$. Since $C_a$ is usually very large it is not feasible to evaluate all elements in practice. Instead a subset, $Q_a \subset C_a$, is used as an estimate. One should be careful when choosing $Q_a$ since this set has a large impact on the assessment of the quality attributes.

Classification performance is probably the most used quality attribute in current practice. It is typically measured in terms of the classification accuracy of a classifier generated by a learning algorithm. The average performance metric and its estimate are defined as:

$$p_1(C_a, D) = \frac{\sum_{x \in C_a} f(x, D)}{|C_a|},$$

(3.1)

$$\hat{p}_1 = p_1(Q_a, D) = \frac{\sum_{x \in Q_a} f(x, D)}{|Q_a|}.$$  

(3.2)

The second metric, best performance and its estimate are defined as:

$$p_2 = \max_{x \in C_a} f(x, D).$$

(3.3)
\[
\hat{p}_2 = \max_{x \in Q_a} f(x, D).
\] (3.4)

We define algorithm sensitivity as being inverse proportionate to robustness, i.e., the impact of tuning a sensitive algorithm is high, while the opposite holds for a robust algorithm. Two metrics are defined for evaluating the sensitivity attribute. \( s_1 \) uses the standard deviation of the evaluation scores to capture the average impact of tuning. The metric \( s_1 \) and its estimate \( \hat{s}_1 \) are defined as follows:

\[
s_1(C_a, D) = \sqrt{\frac{\sum_{x \in C_a} (f(x, D) - p_1(C_a, D))^2}{|C_a|}} \quad (3.5)
\]

\[
\hat{s}_1 = s_1(Q_a, D) = \sqrt{\frac{\sum_{x \in Q_a} (f(x, D) - \hat{p}_1(Q_a, D))^2}{|Q_a| - 1}}. \quad (3.6)
\]

The metric \( s_2 \) uses the statistical range, i.e., the difference between the highest and lowest scores, to capture the degree of impact. \( s_2 \) and its estimate are defined as:

\[
s_2 = p_2 - \min_{x \in C_a} f(x, D) \quad (3.7)
\]

\[
\hat{s}_2 = \hat{p}_2 - \min_{x \in Q_a} f(x, D). \quad (3.8)
\]

### 3.3 Experiment Design

The aim is to investigate the impact of algorithm parameter tuning by assessing learning algorithm quality attributes and to find out whether there is a trade-off between these attributes. The experiment is run using the Weka machine learning workbench (Witten & Frank, 2005).

#### 3.3.1 Featured Algorithms

The four algorithms; Back-Propagation (BP) (Rumelhart, Hinton, & Williams, 1986), K-Nearest Neighbor (KNN) (Aha, Kibler, & Albert, 1991), Support Vector Machines (SVM) (Cortes & Vapnik, 1995), and C4.5 (Quinlan, 1993) were selected for this study. The motivation is that these four algorithms are all widely used and, in addition they belong to four different families of learning algorithms; neural network learners, instance-based learners, kernel machines and decision tree learners. Many studies on supervised learning include one or more of these algorithms and they are all discussed extensively in the literature, cf., Mitchell (1997b); Russell and Norvig (2003).

The number of parameters and the extent to which it is possible to influence the performance of the resulting classifier vary between different algorithms. Some allow very extensive tuning to adjust to a specific problem while others are completely unconfigurable. The parameter intervals were chosen by selecting limits close to, and symmetrically around, the default values of Weka. In case Weka used the lowest possible setting
as the default value this setting was chosen as a lower bound. One of the most complex algorithms in this respect is BP (MultilayerPerceptron in Weka) and many researchers have suggested configuration guidelines for different problems (Bebis & Georgiopoulos, 1995).

A subset of the BP algorithm parameters and the network structure were selected for this particular study; the training time, hidden layers and neurons in each hidden layer. The justification for choosing these parameters is that they all are relative to the data set on which the algorithm operates; hence they should be optimized for a particular problem. Other parameters like momentum and learning rate are often set to very similar values independent of the data set. Common values for momentum and learning rate are 0.2 and 0.3 respectively (Mitchell, 1997b; Witten & Frank, 2005). The Weka default value for training time is 500 epochs and the number of hidden neurons is defined by

\[ n = (\alpha + \beta)/2 \]

where \( \alpha \) is the number of classes and \( \beta \) is the number of attributes for a particular data set. Table 3.1 describes the configurations of BP used in the experiment. The time and layers parameters both have static intervals but the neurons parameter interval is dynamic and varies with the number of classes and attributes of different data sets. As a constraint the lower bound of the hidden neurons interval is 1. This means that even if \( n \) is lower than 7 the lower bound will not be lower than 1, e.g., if \( \alpha = 2 \) and \( \beta = 3 \) then \( n = 2 \) which would make the lower limit \( n - 6 = 2 - 6 = -4 \) if it was not restricted.

Regarding the KNN algorithm (IBk in Weka), the number of neighbors, \( k \), to use
Table 3.3: C4.5 parameter configurations

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidence threshold</td>
<td>[0.02...0.5] step size: 0.02</td>
</tr>
<tr>
<td>Minimum instances per leaf</td>
<td>[1...4] step size: 1</td>
</tr>
<tr>
<td>Folds for pruning</td>
<td>[2...5] step size: 1</td>
</tr>
<tr>
<td>Pruning</td>
<td>{no, yes}</td>
</tr>
<tr>
<td>Reduced-error pruning</td>
<td>{no, yes}</td>
</tr>
<tr>
<td>Subtree raising</td>
<td>{no, yes}</td>
</tr>
<tr>
<td>Sample size</td>
<td>808</td>
</tr>
</tbody>
</table>

for a particular problem can be difficult to know beforehand and a number of methods for finding an optimal value have been proposed. The Weka default value of $k$ is 1. The KNN algorithm can also be configured to weight instances differently and two weighting techniques included in the Weka implementation are inverse-distance weighting ($w = 1/distance$) and similarity weighting ($w = 1 − distance$). Equal weighting ($w = distance$) is the default technique. The KNN configurations are shown in Table 3.2.

The C4.5 algorithm (J48 in Weka) induces pruned or unpruned decision tree classifiers. Configuration possibilities include the type of pruning, the confidence threshold for pruning, the number of folds used for reduced error pruning and the minimum number of instances per leaf. Subtree raising is a post-pruning technique that raises the subtree of the most popular branch. The most popular branch is defined as the branch that has the highest number of training instances. Reduced-error pruning is performed by splitting the data set into a training and validation set. The smallest version of the most accurate subtree is then constructed by greedily removing the node that less improves the validation set accuracy. The J48 default configuration can be summarized as follows. Pruning is on and the technique used is subtree raising. The confidence threshold for pruning is 0.25, the minimum instances per leaf parameter is set to 2 and the number of folds for reduced error pruning value is set to 3. The configuration possibilities of J48 are further detailed in Table 3.3. For Support Vector Machines (SVM), SMO in Weka supports two different kernels: polynomial and radial basis function. These kernels have different sets of parameters that can be configured for a specific problem. Configurations have been chosen so that each parameter interval lies in proximity of the Weka defaults. One parameter is the kernel, which can be set to either polynomial or radial basis function. The gamma parameter is only used by the radial basis function whereas the exponent and lower order terms parameters are only used by the polynomial kernel. The complexity constant determines the trade-off between the flatness and the amount by which misclassified samples are tolerated (Howley & Madden, 2004). The SVM parameters are shown in Table 3.4.
Table 3.4: SVM parameter configurations

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>[0.1...1.0] step size: 0.1</td>
</tr>
<tr>
<td>Kernel function</td>
<td>{polynomial, radial basis}</td>
</tr>
<tr>
<td>Gamma</td>
<td>[0.005...0.060] step size: 0.005</td>
</tr>
<tr>
<td>Lower order terms</td>
<td>{true, false}</td>
</tr>
<tr>
<td>Exponent</td>
<td>[0.5...2.5] step size: 0.5</td>
</tr>
<tr>
<td>Sample size</td>
<td>200</td>
</tr>
</tbody>
</table>

Table 3.5: Data set overview

<table>
<thead>
<tr>
<th>Data set</th>
<th>Instances (I)</th>
<th>Classes</th>
<th>Attributes (A)</th>
<th>I * A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast-cancer</td>
<td>286</td>
<td>2</td>
<td>9</td>
<td>2,574</td>
</tr>
<tr>
<td>Contact-lenses</td>
<td>24</td>
<td>3</td>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>Iris</td>
<td>150</td>
<td>3</td>
<td>4</td>
<td>600</td>
</tr>
<tr>
<td>Labor</td>
<td>57</td>
<td>2</td>
<td>16</td>
<td>912</td>
</tr>
<tr>
<td>Lymph</td>
<td>148</td>
<td>4</td>
<td>19</td>
<td>2,812</td>
</tr>
<tr>
<td>Soybean</td>
<td>683</td>
<td>19</td>
<td>35</td>
<td>23,905</td>
</tr>
<tr>
<td>Weather</td>
<td>14</td>
<td>2</td>
<td>4</td>
<td>56</td>
</tr>
<tr>
<td>Zoo</td>
<td>101</td>
<td>7</td>
<td>18</td>
<td>1,818</td>
</tr>
</tbody>
</table>

3.3.2 Procedure

Eight data sets from the UCI machine learning repository were used in the experiments. The number of instances (I), classes and attributes (A) of each data set are shown in Table 3.5. The last column of this table, I * A, could be regarded as a measure of problem size. For each of the four algorithms, a large number of different parameter configurations were evaluated using 10-fold cross-validation (10CV) on each of the eight data sets. 10CV is performed by dividing the data set into 10 folds or partitions. Classifiers are then generated by training on nine folds and validated on the tenth until all folds have been used for validation. The 10CV score is defined as the mean of all validation test scores. We use 10CV as evaluation function for the metrics and the justification is that it is perhaps the most common classifier evaluation method and extensive tests on different data sets with different techniques have shown that ten is about the right number of folds to get the best estimate of accuracy. Other versions of cross-validation exist, e.g. leave-one-out, but the drawbacks of this procedure include a high computational cost and it also implies a non-stratified sample (Witten & Frank, 2005) because the folds will
not have the same class distribution as the complete data set. For instance, if a data set has 100 instances and the class distribution is 50 class A and 50 class B instances, this distribution (50% A and 50% B) will be very different from a leave-one-out testing fold which will be either 100% class A or 100% class B.

3.4 Results

The classification performance and sensitivity of each algorithm for each data set, measured according to the four metrics, are presented in Table 3.6. In addition, Figure 3.1 and Figure 3.2 contain box plots (one for each data set) that show the distribution of 10CV values for the four algorithms. Each box plot indicates the highest and lowest 10CV value as well as the median and the upper and lower quartiles.

We see that both the highest mean, \( \hat{p}_1 \), (Table 3.6) and the highest median (Figure 3.1 and Figure 3.2) are lower than the lowest max value, \( \hat{p}_2 \), for all eight data sets. Assuming that the mean and median values are approximations of the performance of the default configuration of an algorithm (that is, given that little effort is spent on parameter tuning the mean/median would be the expected value), then this would suggest that it is more important to tune the parameters of an arbitrary algorithm than choosing a particular algorithm. At least this holds for the classification tasks and algorithms studied in this experiment.

We have investigated the relation between the two quality attributes, e.g., whether there is a trade-off between classification performance and robustness, i.e., if sensitive algorithms have high performance. If this trade-off was present there should be a strong positive correlation between the classification performance and sensitivity metrics. However this is not the case. With a significance level of 0.05 (95% confidence) and a sample size of 8 the correlation coefficient must be greater than 0.707 to establish a nonzero correlation. Using this level of significance we could establish a correlation between the two classification performance metrics (0.73) and the two sensitivity metrics (0.84) but no correlation could be established between a classification performance metric and a sensitivity metric.

When comparing the highest sensitivity and classification performance scores between the algorithms across all data sets, it can be observed that KNN has the highest \( \hat{p}_1 \) on a majority of the data sets while SVM only has the highest \( \hat{p}_1 \) on one data set. The opposite holds for \( \hat{p}_2 \). BP has the highest \( \hat{s}_1 \) and \( \hat{s}_2 \) on a majority of the data sets, followed by C4.5 and SVM.

In six out of eight data sets, the range (\( \hat{s}_2 \)) is greater for BP and SVM than for KNN and C4.5. This would suggest that BP and SVM are more sensitive to the choice of parameter setting, which in turn indicates that BP and SVM are harder to tune. For at least four out of eight data sets, the best scoring algorithm is also the worst scoring. This strengthen the argument made that there is little, or no correlation between sensitivity and performance.
Table 3.6: Experiment results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data set</th>
<th>Performance</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\hat{p}_1$</td>
<td>$\hat{p}_2$</td>
</tr>
<tr>
<td>BP</td>
<td>Breast-cancer</td>
<td>0.69</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>Contact-Lenses</td>
<td>0.67</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>Iris</td>
<td>0.79</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>Labor</td>
<td>0.83</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>Lymph</td>
<td>0.75</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>Soybean</td>
<td>0.81</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>Weather</td>
<td>0.64</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>Zoo</td>
<td>0.76</td>
<td>0.97</td>
</tr>
<tr>
<td>C4.5</td>
<td>Breast-cancer</td>
<td>0.70</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>Contact-lenses</td>
<td>0.76</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>Iris</td>
<td>0.94</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>Labor</td>
<td>0.78</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td>Lymph</td>
<td>0.75</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>Soybean</td>
<td>0.88</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Weather</td>
<td>0.61</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>Zoo</td>
<td>0.89</td>
<td>0.96</td>
</tr>
<tr>
<td>KNN</td>
<td>Breast-cancer</td>
<td>0.71</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>Contact-lenses</td>
<td>0.72</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>Iris</td>
<td>0.95</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>Labor</td>
<td>0.85</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>Lymph</td>
<td>0.82</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>Soybean</td>
<td>0.83</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>Weather</td>
<td>0.67</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>Zoo</td>
<td>0.92</td>
<td>0.96</td>
</tr>
<tr>
<td>SVM</td>
<td>Breast-cancer</td>
<td>0.71</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>Contact-lenses</td>
<td>0.64</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>Iris</td>
<td>0.92</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>Labor</td>
<td>0.80</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>Lymph</td>
<td>0.78</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>Soybean</td>
<td>0.86</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>Weather</td>
<td>0.63</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>Zoo</td>
<td>0.78</td>
<td>0.97</td>
</tr>
</tbody>
</table>
Both $\hat{s}_1$ and $\hat{s}_2$ are very high for BP on the Iris and Soybean data sets. However, $\hat{s}_1$ is higher than $\hat{s}_2$ for Iris, and the opposite holds for Soybean. For Iris, it can be observed that the lowest value is a part of general low performance while the lowest value for Soybean seems to be an outlier. This suggests that both sensitivity metrics are important, since they can be used to measure different aspects of sensitivity.

### 3.5 Discussion

The metrics $\hat{p}_1$ and $\hat{p}_2$ tell little about to which extent the performance can be influenced by tuning the learning algorithm. In order to capture both the performance and to what extent an algorithm is sensitive to parameter change, two different quality attributes have to be assessed. A sensitive algorithm may be a better choice for experts or researchers searching for optimal performance, while a robust algorithm may provide less experienced users with a method that is easy to use and still produce good results.

It is hard to formulate a strict measure of the impact that algorithm parameters have on classifier performance. On the one hand, it can be a dependency problem. If one parameter is dependent on the value of another and changing the first parameter impacts classifier performance, it could be an indirect impact caused by the second parameter. On the other hand, there is a degree of randomness that affects the evaluation. For example, the weights of a neural network are initialized to random values before training. This could influence the performance of the learned classifier differently between runs. There is also a small element of randomness involved when using 10CV. Estimation bias and variance also need to be considered if the training set or the test set is small. However, there are certain measures that may indicate the degree of impact. $\hat{s}_2$ reveals the upper and lower limits of the impact of tuning. However, it does not explain which parameter had the largest impact. $\hat{s}_1$ reveals information about how results are scattered between the upper and lower limits. These two metrics capture different aspects of sensitivity, i.e., how much the output (performance) is related to the input (configuration) of an algorithm.

As shown in this study, the quality attributes and their metrics can be used to better understand the behavior of learning algorithms. In addition, they can be used for selecting which algorithm to use in a particular situation. One approach would be to apply the Analytic Hierarchy Process (AHP) (Saaty & Vargas, 2001), which is a multi-criteria decision support method stemming from Management Science. One of the cornerstones in AHP is to evaluate a set of alternatives based on a particular blend of criteria, i.e., considering a specific trade-off situation. The first step in AHP is to set up a hierarchy of the criteria that are being evaluated. This means that one criterion can be broken down into several sub-criteria, and the evaluation of the different alternatives is done by weighing all levels of this decision support hierarchy. In our case, the criteria would be based on quality attributes (and metrics), and their importance for the application at hand.
Figure 3.1: Box plots of 10CV scores on four data sets: breast-cancer, contact-lenses (1st row), iris, labor (2nd row). The dotted line represents the performance of a majority class model.
Figure 3.2: Box plots of 10CV scores on four data sets: lymph, soybean (1st row), weather, zoo (2nd row). The dotted line represents the performance of a majority class model.
3.6 Related Work

Meta-learning research focuses on automatic ways to induce meta-knowledge from experience. The objective is learning how to learn by using algorithms to analyze and interpret prior experiments and data set characteristics in order to improve the quality of learning (Giraud-Carrier & Keller, 2002). Many researchers have recognized the fact that algorithm optimization can be very time consuming. Often there is a need for domain-knowledge and expertise in order to generate good classifiers. Meta-learning studies have also brought forth different methods for enhancing the performance of existing techniques, e.g., automatic tuning of SVM kernel parameters (Chapelle, Vapnik, Bousquet, & Mukherjee, 2002) and cascade-correlation for automatically creating and training neural networks (Fahlman & Lebiere, 1990). Other enhancing methods, such as bagging, boosting and stacking, combine several classifiers to increase performance, cf., Breiman (1996); Freund and Schapire (1996); Witten and Frank (2000); Wolpert (1992). Whereas meta learning is concerned with learning how to learn, this study addresses issues such as how to quantify the potential and need of meta-learning for different learning algorithms.

Several methods exist for both classifier and learning algorithm evaluation. For example, classifiers can be evaluated using measure functions (Andersson et al., 1999) or holdout (Duda et al., 2000) while algorithms are commonly evaluated using cross-validation (Stone, 1974) or bootstrap (Efron & Tibshirani, 1993). Large-scale empirical studies have been conducted to examine the behavior of such evaluation methods and the metrics they use, cf., Caruana and Niculescu-Mizil (2004). Whereas this study investigates different performance metrics to measure the quality of classifiers (the output from the learning algorithm using a particular parameter setting), we define metrics which are used to measure the quality of the algorithm itself.

Some work has focused on the sensitivity of algorithms, e.g., sensitivity analysis which aims at determining how variation of the input influences the output of a system (Bousquet & Elisseeff, 2002). Instead of regarding data sets as the only input for learning, we consider algorithm parameter configuration to be an important part of the input as well.

3.7 Conclusions and Future Work

We have studied the impact of learning algorithm optimization by means of parameter tuning and argue that some algorithms are more robust to parameter change than others. Two quality attributes that facilitate the impact analysis were defined. One attribute, sensitivity, captures the extent to which it is possible to affect performance by tuning parameters and the other captures the classification performance of an algorithm. We then presented two candidate metrics for each quality attribute and showed that these metrics complete each other by revealing different aspects of sensitivity and classifica-
tion performance. The metrics depend on the evaluation of all possible configurations of an algorithm. In most practical cases this is impossible and we therefore suggest an estimation that can be performed by selecting a subset of the configurations symmetrically around a known default configuration, e.g., determined by meta-learning techniques. We have evaluated each configuration using cross-validation; however the metrics as such are not restricted to this particular method. Our results indicate that parameter tuning is often more important than the choice of algorithm and we provide quantitative support to the assertion that some algorithms are more robust than others. Moreover, the results suggest that no trade-off exists between sensitivity and classification performance.

In future work, we intend to refine the method for configuration subset selection to increase the accuracy of the metrics estimations and evaluate the sensitivity and classification performance using a wider selection of algorithms and data sets.

Other metrics are also considered, e.g., one plausible metric for sensitivity could be the average performance difference between adjacent configurations. An assumption related to this metric is that smooth distributions would benefit optimization of the particular algorithm. We also intend to investigate the use of AHP and other approaches that could be used for systematic selection of learning algorithms.
4.1 Introduction

The choice of which method to use for classifier performance evaluation is dependent of many attributes and it is argued that no method satisfies all the desired constraints (Mitchell, 1997b). This means that, for some applications, we need to use more than one method in order to get a reliable evaluation. Another possible consequence is that, for a specific class of problems, there are some methods that are more suitable than others. It has been argued that the construction of classifiers often involves a sophisticated design stage, whereas the performance assessment that follows is less sophisticated and sometimes very inadequate (Adams & Hand, 2000). Sometimes the wrong evaluation method for the problem at hand is chosen and other times too much confidence is put into one individual method.

Also, it is important to note that many methods measure performance based solely on the classification accuracy of a limited set of given instances. Examples of such methods include cross-validation (Stone, 1974), bootstrap (Efron & Tibshirani, 1993) and holdout (Duda et al., 2000). There are some alternative evaluation methods that take
into account more than just accuracy, e.g., ROC plots and lift charts (Witten & Frank, 2005). Moreover, Bebis and Georgiopoulos (1995) defined a fitness function consisting of one generalization term and one network size term for use with a genetic algorithm when optimizing neural networks. However, this method is classifier-specific and cannot be used for other types of classifiers than neural networks.

The use of measure functions for evaluating classifier performance was proposed by Andersson et al. (1999). They argued that this alternative approach is able to handle some of the problems with the methods mentioned above. In experiments it was shown that a measure function could also be used to build a custom classifier targeted towards solving a specific class of problems. We will here present an implementation based on the original measure function example.

The implementation is able to evaluate classifiers generated from data sets with any number of attributes, contrary to the original function which is limited to classifiers generated from two data set attributes. In the next two sections, we introduce measure-based evaluation and cross-validation evaluation, which will be used for comparisons. Section 4.4 gives the details of the measure function implementation, and this is followed by a section about measure function configuration and example usage. Then some experiments using the current implementation are described. The positive and negative aspects of measure-based classifier performance evaluation as well as future work are discussed in the last section.

### 4.2 Measure-based Evaluation

According to Andersson et al. (1999) a measure function assigns a value describing how good the classifier is at generalizing from the data set, for each possible combination of data set and classifier. They also argue that most popular learning algorithms try to maximize implicit measure functions made up of one or more of three heuristics: subset fit (known instances should be classified correctly), similarity (similar instances should be classified similarly) and simplicity (the partitioning of the decision space should be as simple as possible). Most evaluation methods only take subset fit (or a version of subset fit where instances not present in the training set are also included) into consideration but the measure function proposed by Andersson et al. (1999) build upon all three heuristics. There is often a trade-off between the three heuristics and the measure function reveals this trade-off. The measure function helps in analyzing the learned classifier and the partitioning of the decision space. For example, consider the two classifiers illustrated in Figure 4.1. Both classifiers would get the highest score in a cross-validation test (since all instances are correctly classified by both classifiers). However, some might regard the classifier making the simple division of the instance space using a straight line as better than the more complex one. A measure function would make it possible to capture this, e.g., by including simplicity and similarity as evaluation aspects. Below, similarity and simplicity will be given a more detailed explanation. However, it is important to note
Figure 4.1: Two-dimensional decision spaces of two classifiers, one illustrated by the dashed line, and the other illustrated by the curves. The letters indicate the positions of instances from two categories (A and B).

that the concept of measure functions is not limited to the three heuristics discussed, but is a much more general concept where any aspect of the classifiers’ division of the instance space may be taken into account.

It is mentioned earlier in this section that some algorithms use the similarity heuristic, i.e., they try to classify similar instances similarly. The question is how to define similarity as a metric for classifier performance. A classifier partitions the instance space into a number of areas corresponding to different classes. From now on, we will refer to each border between such areas as a decision border, and to the entire region covering all such areas as the decision space. According to Andersson et al. (1999) one often used heuristic of similarity is that the decision borders should be centered between clusters of instances belonging to different classes (e.g., Support Vector Machines try to accomplish this by finding the maximum margin hyperplane, cf., Witten and Frank (2005)). Thus, in general terms, the distance between each instance and its closest decision border could work as a measure of similarity.

The simplicity heuristic is used, either because of the belief that a simple solution reduces the chance of overfitting the data, or because of the belief that one should always pick the simplest of two equally good solutions (according to the theory of Ockham’s razor, cf., Tornay (1938)). According to Andersson et al. (1999) simplicity is often measured with respect to a particular representation scheme. One example of measuring simplicity this way is to count the number of nodes in an induced decision tree. However, the measure function need to be general enough to be applicable to any classifier independently on how it is represented and which learning algorithm was used to train it. Thus, it should focus on the decision space rather than the representation, or structure,
4.3 Cross-validation Evaluation

Cross-validation (CV) tests exist in a number of versions but the general idea is to divide the training data into a number of partitions or folds. A classifier is then trained using all folds except one and evaluated on the remaining fold using classification accuracy as a measure for classifier performance. This procedure is repeated until all partitions have been used for evaluation once. Perhaps, it would be better to refer to cross-validation as an algorithm evaluation method, rather than a classifier evaluation method, since many classifiers are trained and evaluated during one test. Some of the most common types used are ten-fold, n-fold, and bootstrap CV (Witten & Frank, 2005).

The difference between these three types of CV lies in the way that data is selected for training and testing. The n-fold CV test, where \( n \) represents the number of instances in the data set, is sometimes referred to as jackknife (Duda et al., 2000) or leave-one-out (Witten & Frank, 2005). It is performed by leaving one instance out for testing and training on all the other instances. This procedure is then repeated until all instances have been left out once.

Somewhat different from the other two mentioned versions of CV, bootstrap is based on the statistical method of sampling with replacement. The data set is sampled \( n \) times to build a training set of \( n \) instances. Because of the use of replacement, some instances will be picked more than once and the instances that are never picked are used for testing. Even though CV tests can give a hint of how well a certain classifier will perform on new instances, it does not provide us with much analysis of the generalization capabilities nor any analysis of the decision regions of the classifier. These and other bits of information extracted from a learned classifier may work as valuable clues when trying to determine how well the classifier would perform on new, previously unseen, instances. It has been argued that the design of ten-fold CV introduces a source of correlation since one uses examples for training in one trial and for testing in another (Maloof, 2002).

4.4 A Multi-dimensional Measure Function

We will now present an implementation of the example measure function suggested by (Andersson et al., 1999), which includes a number of modifications. The main improvement is that the new version supports evaluation of classifiers learned from data sets with more than two attributes. To accomplish this, new algorithms for similarity and simplicity had to be designed and implemented. In addition, this version uses Weka (Witten & Frank, 2005), a popular machine learning workbench, which makes it compatible both with the Weka ARFF data set standard and the learning algorithms included in this workbench (this is further discussed in Section 4.4.4). From now on we refer
to the implementation of the measure function as MDMF (Multi-Dimensional Measure Function).

In order to compute the similarity and the simplicity components of MDMF, the instance space has been normalized. The instances of the data set are normalized by dividing their attribute values by the highest found value for each attribute.

4.4.1 Similarity

To measure similarity we need to calculate the distance, $d$, between each instance and its closest decision border. If an instance was correctly classified the similarity contribution of that instance is positive, otherwise it is negative. The reason for applying this rule is that correctly classified instances should preferably reside as far away as possible from the decision borders, as mentioned in the description of the similarity heuristic in Section 4.2. Incorrectly classified instances reside on the wrong side of the decision border so the contribution they make to similarity should be negative. Preferably these instances should reside close to the border of the correct class. This is implemented by letting the contribution be less negative if the instance is close to the border of the correct class. The formulas used to calculate the contribution of $d$ to similarity are taken from the original measure function; the contributions from correctly and incorrectly classified instances, respectively, are described by Equation 4.1 and Equation 4.2.

$$1 - \frac{1}{2^{bd}}.$$ \hfill (4.1)

$$\frac{1}{2^{bd}} - 1.$$ \hfill (4.2)

The normalization constant, $b$, for the similarity function is chosen in the same way as for the original measure function; the square root of the number of instances in the data set is used. Both Equation 4.1 and Equation 4.2 describe functions with sigmoid-like behavior. They are asymptotic for large positive and negative values in order to capture the fact that instances very far from borders should not be affected by slight border adjusts.

In order to find the decision border closest to an instance, we start from the location of the instance and search outwards in all dimensions (one for each attribute). We cannot measure the length between an instance and a border in a trivial manner since it cannot be assumed that the classifier provides an explicit representation of the decision borders. By moving outwards in small steps and querying the classifier about the current position (by making it classify an imaginary instance at that position) it is easy to determine if we have entered a region belonging to a different class than that of the original instance which we calculate the distance for. If this is the case a decision border has been found. Figure 4.3 and Figure 4.4 describe the algorithms for calculating similarity in detail. For pseudo code on a higher abstraction level, see Figure 4.2. We will now explain the basic
for all instances do
    MoveToInstancePos()
    $SDist \leftarrow 0$
    while $SDist < 1$ do
        Increase($SDist$ with $RadiusIncrease$)
        for all dimensions except one do
            MoveFromInstancePos(either $-SDist$ or $SDist$)
            for $pos = -SDist$ to $SDist$ stepsize $SDist \times 2/10$ do
                MoveInExcludedDimension($pos$)
                if NewPrediction $\neq$ OldPrediction then
                    CalculateDistance()
                    Break()
                end if
            end for
        end for
    end while
end for

Figure 4.2: MDMF Similarity calculation

functionality of the algorithms used for the calculation of similarity. Each instance, $x$, is
classified and the result, $p$, is compared with the known class value, $a$, for that instance.

The search for a decision border starts at the position of $x$ and the size of the search
area is decided by $SDist$. $SDist$ increases with $RadiusIncrease$ (which is a configurable
parameter of MDMF) for every iteration.

For each new search area, the following procedure is repeated: one attribute is se-
lected as main attribute. All other attribute values are either set to $-SDist$ or $SDist$,
relative to the original instance position. The main attribute value is then set to $-SDist$
and the new instance position is classified. If the classification differs from the clas-
sification of the original instance position we have found a decision border, else the
main attribute value is incremented and the procedure is repeated until it reaches $SDist$.
The area has been searched completely when all attributes have been selected to main
attribute once and all non main attributes have been set to both $-SDist$ and $SDist$.
When the area search is complete a new iteration starts.

The distance between an instance and its closest decision border, $d$, is the Euclidean
distance between the original instance position and the position of the imaginary instance
defined by the current attribute values. Depending on the classification ($a = p$ or $a \neq p$)
the contribution to similarity is computed using either Equation 4.1 or Equation 4.2.

The computational effort needed to calculate similarity depends on the number of
attributes, the number of instances in the data set, and the number of search steps. The
Require: $I$ (A set of instances)
Require: $A$ (The set of attributes of $I$)
Require: $c$ (A classifier)

for all $x \in I$ do
    $p \leftarrow \text{PredictedClass}(x, c)$
    $a \leftarrow \text{ActualClass}(x, I)$
    if $a = p$ then
        Correct $\leftarrow$ true
    else
        $p \leftarrow a$
        Correct $\leftarrow$ false
    end if
    $SDist \leftarrow 0$
    while $a = p$ do
        $SDist \leftarrow SDist + RadiusIncrease$
        if $SDist > 1$ then
            break
        end if
        step $\leftarrow (SDist * 2)/10$
        for $mainAttr = 0$ to $|A|$ do
            for $z = 0$ to $\text{Power}(2, |A| - 1)$ do
                Search()
            end for
        end for
    end while
    if Correct then
        ContributePosValue(dist)
    else
        ContributeNegValue(dist)
    end if
end for

Figure 4.3: MDMF Similarity calculation
modifier ← 0
for subAttr = 0 to |A| do
    if subAttr! = mainAttr then
        attrDist[subAttr] ← direction[z >> modifier&1] * SDist
        x.set(subAttr, oldValue[subAttr] + attrDist[subAttr])
        modifier ← modifier + 1
    end if
end for
for pos = −SDist to SDist do
    attrDist[mainAttr] ← pos
    x.set(mainAttr, oldValue[mainAttr] + pos)
    p ← PredictedClass(x, c)
    if p! = a then
        dist ← |attrDist|
        break
    end if
    pos ← pos + step
end for

Figure 4.4: Similarity search function
complexity of the similarity calculation is $O(nas)$ where $n$ is the number of instances, $a$ the number of attributes, and $s$ the number of possible search steps in each dimension.

The Iris data set (Anderson, 1935) has four numeric attributes (and a nominal target attribute). To calculate similarity for a classifier learned from this data set we have to search in four dimensions (one for each attribute) to locate the closest decision border for each instance. Actually, this is just an approximation since the closest border may reside between two search steps. Iris consists of 150 instances so we need to perform an equal amount of individual searches through all four dimensions to calculate the similarity aspect of MDMF. The number of search steps is currently set to ten but this can be changed very easily, e.g., a higher value would result in a more detailed but slower search.

### 4.4.2 Simplicity

It has been suggested that simplicity can be measured by calculating the total size of the decision borders (Andersson et al., 1999). Contrary to subset fit and similarity, the simplicity value (the size of the decision borders) should be as low as possible. Although this is fairly simple to calculate for two or maybe even three dimensions for classifiers that have explicit decision borders, it becomes increasingly hard to calculate for a larger number of dimensions, and for classifiers with implicit decision borders.

Andersson et al. (1999) have proposed the following way to approximate the decision border size: a number of lines are chosen each so that they cross the instance space with a random direction and starting point. The decision border size is then approximated by the average number of decision borders encountered when traveling along the lines. The procedure is quite simple; at the start of the line the coordinates are used as attribute values for an imaginary instance and the learned classifier then classifies this instance and the result is stored. A small movement is made from the initial position in the direction indicated by the line.

These events are then repeated until the end of the line is reached and if the returned class value is not equal to the previously stored value, this would indicate that a decision border has been found. A problem with this method is the stochastic element; since the lines are chosen randomly the result varies between runs and this should be avoided for stability and consistency reasons.

To remedy this, we have implemented a deterministic simplicity calculation method. Simplicity is calculated by dividing the decision space into a grid and traveling along each grid line, querying the classifier for each position, in order to find out the average number of decision borders encountered. The size of the squares of the grid used for calculation of simplicity is changeable. A smaller size results in longer execution time but higher level of detail and vice versa. The complexity of the simplicity calculation is $O(ga)$ where $g$ is the number of grid squares per dimension and $a$ is the number of attributes.
Table 4.1: MDMF configuration

<table>
<thead>
<tr>
<th>Weight</th>
<th>Affects</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>Subset fit</td>
<td>1.0</td>
</tr>
<tr>
<td>$a_1$</td>
<td>Similarity</td>
<td>1.0</td>
</tr>
<tr>
<td>$a_2$</td>
<td>Simplicity</td>
<td>0.1</td>
</tr>
<tr>
<td>$k_1$</td>
<td>$\text{Simi}_1$</td>
<td>0.5</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$\text{Simi}_2$</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Detail</th>
<th>Affects</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square size</td>
<td>Simplicity</td>
<td>0.01</td>
</tr>
<tr>
<td>radius size</td>
<td>Similarity</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

### 4.4.3 Subset Fit

The subset fit measure is implemented as described in the original measure function: the learned classifier is evaluated using the training set and the subset fit measure result is calculated by dividing the number of correctly classified instances by the total number of instances. The complexity of the subset fit calculation is $O(n)$, where $n$ is the number of instances.

### 4.4.4 A Weighted Measure Function

MDMF was implemented in Java using the Weka machine learning workbench (Witten & Frank, 2005). Weka contains a collection of machine learning algorithms as well as tools for classification, visualization, regression and clustering. Weka implementations exist for many of the well-known learning algorithms and these can be evaluated with MDMF. MDMF also makes use of the ARFF data set file format used by Weka. MDMF is very flexible and can be configured in many different ways, depending on the problem at hand. The different parts of the function are weighted in order to provide a way to focus on one or more specific aspects when evaluating and comparing classifier results. Subset fit, similarity and simplicity can all be weighted individually by changing variables $a_0$, $a_1$ and $a_2$. Similarity is divided into two parts; the similarity of correctly classified instances and the similarity of incorrectly classified instances. These two parts of similarity can also be weighted differently and the variables $k_1$ and $k_2$ are used for this purpose.

$$ \text{MDMF} = a_0 \text{Subsetfit} + a_1 (k_1 \text{Simi}_1 + k_2 \text{Simi}_2) - a_2 \text{Simp} . $$

(4.3)

It is now possible to describe the measure function with Equation 4.3. Except for the tuning of metric weights, the level of detail for the simplicity and similarity calculations...
can also be altered. As described earlier, both simplicity and similarity are calculated by classifying different points in the instance space in order to find out size and layout of the decision borders. Working with many data set attributes as well as large numbers of instances greatly increases the number of predictions that has to be made. As we observed from the complexity analysis of the similarity and simplicity computations, there is a trade-off between the level of detail and the execution time (as with many computational problems) and the balance depends on the specific problem to be solved.

The MDMF configuration used in the example and the experiments of this article is detailed in Table 4.1. Up till now we have presented the concept of measure-based evaluation and the implementation of the measure function. Let us demonstrate the functionality with an illustrative example. The Weka implementation of C4.5 (Quinlan, 1993), which is called J48, is here used to induce two different decision trees from the Iris data set. The first decision tree is left unpruned and reduced error pruning is applied on the second. The results are presented in Table 4.2. It seems that 10CV and MDMF yield contradicting results. The unpruned decision tree obviously predicts with higher accuracy than the pruned tree in this example (according to 10CV). The complete MDMF result, including the values for each metric, is shown in Table 4.3. We can see that even though the accuracy (subset fit) is higher for the unpruned tree, the pruned version has a lower simplicity value indicating that it is a simpler, perhaps less overfitted, solution. Two-dimensional visualizations of the decision spaces of these two decision trees can be seen in Figure 4.5. Intuitively the decision space of the pruned tree looks less overfitted and less complicated. What is important to notice here is the way the total measure gets calculated by combining the different parts along with their individual weights. After the evaluation has been performed the weights can be changed over and over again to reflect different views or perspectives of classifier performance. Different parts can be

\begin{table}
\centering
\caption{Comparison between 10CV and MDMF}
\begin{tabular}{lcc}
\hline
Algorithm & 10CV & MDMF \\
\hline
J48 & 0.953 & 1.086 \\
J48-P & 0.947 & 1.101 \\
\hline
\end{tabular}
\end{table}

\begin{table}
\centering
\caption{Measure-based evaluation of C4.5}
\begin{tabular}{lccccc}
\hline
Algorithm & Subset & Similarity & Simp & MDMF \\
\hline
J48 & 0.980 & 0.756 & 2.724 & 1.086 \\
J48-P & 0.967 & 0.720 & 2.253 & 1.101 \\
\hline
\end{tabular}
\end{table}
Figure 4.5: Visualizations of two-dimensional decision spaces. The J48 (top) and J48-P (bottom) classifiers have been produced by the J48 algorithm.
Table 4.4: Measure-based evaluation of 10 classifiers

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Subset fit</th>
<th>Similarity</th>
<th>Simplicity</th>
<th>MDMF</th>
</tr>
</thead>
<tbody>
<tr>
<td>J48</td>
<td>0.980</td>
<td>0.756</td>
<td>2.724</td>
<td>1.086</td>
</tr>
<tr>
<td>J48-p</td>
<td>0.967</td>
<td>0.720</td>
<td>2.253</td>
<td>1.101</td>
</tr>
<tr>
<td>IB1</td>
<td>1.000</td>
<td>0.692</td>
<td>4.300</td>
<td>0.916</td>
</tr>
<tr>
<td>IB10</td>
<td>0.967</td>
<td>0.717</td>
<td>3.762</td>
<td>0.949</td>
</tr>
<tr>
<td>BP2-500</td>
<td>0.987</td>
<td>0.720</td>
<td>4.196</td>
<td>0.927</td>
</tr>
<tr>
<td>BP2-5000</td>
<td>0.987</td>
<td>0.728</td>
<td>4.124</td>
<td>0.938</td>
</tr>
<tr>
<td>BP2-25000</td>
<td>0.987</td>
<td>0.723</td>
<td>3.935</td>
<td>0.955</td>
</tr>
<tr>
<td>BP3-20000</td>
<td>1.000</td>
<td>0.708</td>
<td>3.772</td>
<td>0.977</td>
</tr>
<tr>
<td>BP3-32000</td>
<td>1.000</td>
<td>0.707</td>
<td>3.778</td>
<td>0.976</td>
</tr>
<tr>
<td>NBayes</td>
<td>0.960</td>
<td>0.721</td>
<td>6.213</td>
<td>0.699</td>
</tr>
</tbody>
</table>

excluded by setting their corresponding weight to zero. The default configuration, discussed in Section 4.4, is chosen so that each part contributes about equally much to the evaluation result.

4.5 Experiments

This section presents two experiments conducted using MDMF. First, we describe an MDMF evaluation of a set of classifiers generated by some well-known learning algorithms. This is followed by a MDMF execution time assessment.

Table 4.4 shows the classifier evaluation results. The default configuration of the measure function has been used. Four learning algorithms have been used to generate a set of classifiers for the evaluation experiment. The decision tree algorithm (J48), which was introduced in an earlier example, has generated one pruned and one unpruned decision tree classifier (J48-P and J48, respectively). Both sub tree raising and reduced error pruning was applied on the first classifier. One of Weka’s nearest neighbor implementations (IBk) has been used to generate a classifier based on one neighbor (IB1) and another classifier based on ten neighbors (IB10). The back-propagation algorithm (NeuralNetwork\(^1\)) has been used to produce five different neural network classifiers (denoted BP\(x\)-\(y\)), each with a different combination of the number of hidden nodes (\(x = 2\) or \(x = 3\)) and the number of epochs for training (\(y = 500\), \(y = 5000\), \(y = 20000\), \(y = 25000\), or \(y = 32000\)). A Bayesian algorithm (NaiveBayes) has also been used to generate one classifier (NBayes). It has been argued that this algorithm performs com-

\(^1\)The name of this class has now been changed to MultilayerPerceptron in Weka after the publication of this paper.
parably with decision tree and neural network classifiers in some domains (Mitchell, 1997b). This makes it an interesting algorithm to use in the experiments concerning the evaluation of classifiers using a measure function.

The NBayes classifier is compared with the decision tree and neural network classifiers in order to find out if it performs comparably when measured with a measure function instead of a CV test. Let us first examine simplicity. It seems that the simplicity measure captured, quite well, the difference in decision space complexity between the pruned and unpruned decision tree. Figure 4.5 shows this difference visually for two selected dimensions.

The same pattern applies to the nearest neighbor classifiers. Figure 4.6 shows the difference visually between the decision spaces belonging to these two classifiers. The NBayes classifier scored the highest (worst) simplicity value and this can also be seen by inspecting the decision space in Figure 4.7 and comparing it to the decision spaces of the other classifiers.

When analyzing the neural network classifier results we can see that the simplest classifier is the one generated with three nodes and a training time of 20,000 epochs. When comparing with another neural network classifier (with the same size but trained for 32,000 epochs) the results indicate that the longer training time resulted in a classifier that may be more overfitted, since both similarity and simplicity values are worse than those of the first classifier.

In the execution time experiment a number of classifiers were evaluated using two different configurations of the simplicity detail level property \textit{square size}. Individual time measures are not important here, since they are tied to a very specific hardware platform, but rather what is interesting is the difference in execution time for the different configurations and the difference in execution time when evaluating different classifiers. The correlation coefficient is .99 when calculated from the two different time measure columns presented in Table 4.5. We can therefore assume that the increase in execution time is only related to the level of detail and not to which classifier is evaluated. It is easily noticed that the MDMF evaluation of the nearest neighbor classifiers is very slow. This is attributed to the fact that the instance-based algorithms are lazy learners. They process training data quickly, by just storing the instances. The actual work is carried out when a new instance should be classified.

Because of this fact, they have a tendency to be slow when classifying rather than learning. A large number of classifications must be performed in order to compute similarity and simplicity, thus a slow classifier will take time to evaluate. The classifier only needs to be generated once during the MDMF evaluation, hence a slow learner does not affect the evaluation time much. This could be compared with ten-fold CV, for which ten classifiers must be generated, one for each fold, in order to compute the test score.

Consequently, the reverse situation holds for CV tests; a slow learner has a more negative effect on the evaluation time than a slow classifier. For example, back propagation is evaluated very slowly with CV, but instance-based algorithms are evaluated quickly.
Figure 4.6: Visualization of a two-dimensional decision space. The classifiers IB1 (top) and IB10 (bottom) have been generated by the IBk algorithm.
Figure 4.7: Visualization of a two-dimensional decision space. The NBayes classifier have been generated by the NaiveBayes algorithm

Table 4.5: Time consumed by measure-based evaluation. The difference between J48 and J48-P is insignificant, hence only the average result is presented. Since the number of epochs a BP classifier has been trained does not impact the time consumed to evaluate it, only the average BP2 and BP3 results are presented

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10 grid</th>
<th>100 grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>J48</td>
<td>19 ms</td>
<td>353 ms</td>
</tr>
<tr>
<td>IB1</td>
<td>1738 ms</td>
<td>25905 ms</td>
</tr>
<tr>
<td>IB10</td>
<td>2103 ms</td>
<td>27501 ms</td>
</tr>
<tr>
<td>BP2</td>
<td>57 ms</td>
<td>850 ms</td>
</tr>
<tr>
<td>BP3</td>
<td>67 ms</td>
<td>943 ms</td>
</tr>
<tr>
<td>NBayes</td>
<td>83 ms</td>
<td>1148 ms</td>
</tr>
</tbody>
</table>
4.6 Conclusions and Future Work

Although cross-validation is the most frequently used method for classifier performance evaluation, we argue that it is important to consider that it only takes accuracy into account. We have shown, with a theoretical example in Section 4.2 and a practical experiment in Section 4.4, that measure functions may reveal aspects of classifier performance not captured by evaluation methods based solely on accuracy. By altering the weights or changing the level of detail the measure function evaluation can be biased towards different aspects of classifier performance, useful for a specific class of problems. The measure function implementation presented in this article, MDMF, supports evaluation of data sets with more than 2 attributes, making it useful for a larger amount of evaluation problems than the original measure function example. However, it is important to note that the concept of measure functions is not limited to the three heuristics discussed in this paper. Thus, the implementation could be further expanded with more heuristics in the future.

One important issue for future study concerns methods for determining the appropriate weights for a given domain (data set). Currently, MDMF only supports normalized data sets with numeric features and a nominal target attribute. This means that it can only evaluate classifiers learned from one specific type of data set. Although this type probably is one of the most common, it would be interesting to extend the capabilities of the measure function so that it can be used to evaluate classifiers on data sets with other attribute types as well, e.g., Boolean and nominal attributes.

The method used to calculate simplicity in MDMF is very different from that of the original measure function, in which the total length of the decision borders was used as a simplicity measure. Since the new implementation supports $n$ dimensions it is not possible to measure the decision border lengths (at least not for dimensions higher than two). As a result another measure for simplicity has been used; the average number of decision borders encountered when traveling through decision space in any direction. Experiments have shown that this measure captures the difference between pruned and unpruned decision trees but there may be other measures of simplicity that are less complex and have even better qualities.

The measure function, as an evaluation method, could be combined with optimization methods such as genetic algorithms (Goldberg, 1989; Holland, 1975) or simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983) to produce classifiers optimized for a specific class of problems. Simple approaches of this type has been carried out using accuracy measures as fitness functions or thresholds (Chalup & Maire, 1999; Ho, Liu, & Liu, 2002). We plan to develop a public website that demonstrates the functionality of the multi-dimensional measure function. Users will be able to interactively supply data, choose a learning algorithm and configure both the algorithm and the measure function. The user would then have the possibility to view the resulting measure with the present configuration or change the different weights and see the changes in real time.
CHAPTER

FIVE

Paper IV

Generic Methods for Multi-Criteria Evaluation

Niklas Lavesson & Paul Davidsson

5.1 Introduction

We consider the problem for which a learning algorithm, given a set of training instances, generates a classifier that can be used to predict the class of new instances. The most frequently used criteria by which to assess classifier performance, and indirectly the performance of the applied learning algorithm, is classification accuracy. The most common metric for accuracy is the ratio between correctly classified instances and the total number of classified instances. However, serious concerns have been raised against the use of this metric as the only estimate for generalization performance (Provost et al., 1998). Additionally, when applying a learning algorithm to solve a real-world problem there are often other important factors that need to be considered. For instance, it might be crucial for a particular application that the training time and classification time of the employed algorithm are lower than some desired thresholds. For another application it might be essential that the classifier representation is easily interpreted by humans. When selecting algorithms for real-world applications, one usually has to be aware of, and properly balance, trade-offs between different criteria. One of the most common practices today is to focus on just one criterion, i.e., using only one evaluation metric.
(e.g., accuracy) or to design a custom multi-criteria evaluation scheme for the particular
application. Some of the drawbacks of the first method have already been presented
and, regarding the second method, we argue that a customizable, generic multi-criteria
method would be more effective and suitable for most applications since it simplifies
comparison and puts the focus on which criteria to assess for a particular application,
rather than on how to combine and trade-off criteria.

Next we will explain the concept of multi-criteria evaluation. In Section 5.2 we then
present three existing generic multi-criteria methods. An analysis and comparison of
these methods is carried out in Section 5.3. Building on the most useful parts of each
method we then present a new generic multi-criteria method in Section 5.4. In the next
section we present an interesting case study in which we apply the new method. We
finish with conclusions and pointers toward future work.

5.1.1 Multi-Criteria Evaluation

It is argued that some performance metrics are more appropriate than others for different
domains (Caruana & Niculescu-Mizil, 2006). This would imply that, analogous to the
no-free-lunch theorems for supervised learning (Wolpert, 1995), which stipulate that no
algorithm is superior on all problems, there is no single metric that is always superior for
performance assessment. One conclusion to draw from this is that one has to investigate
which criteria are relevant for a particular problem, and which metrics should be used to
assess these criteria. We argue that it is important to evaluate on the basis of all criteria
that are relevant for a particular problem. This should be accomplished by combining
criteria, in a systematic way, into one measure instead of comparing different criteria in
an ad-hoc manner.

We have distinguished between three types of candidates for evaluation: algorithms,
algorithm configurations, and classifiers (Lavesson & Davidsson, 2007). Algorithm
evaluation refers to the overall evaluation of an algorithm, covering potentially all, or
a selection, of its parameter settings, for a particular problem. An algorithm configura-
tion evaluation, on the other hand, refers to the evaluation of one particular parameter
setting for an algorithm. A classifier evaluation is an evaluation of one classifier on a
specific data set.

A criterion can usually be assessed using a variety of metrics, e.g., the complexity of
an induced tree can be assessed using a metric that simply counts the nodes, but a more
generic complexity metric might be based on the number of bits needed to represent
the classifier. In this study, we use the term multi-criteria (MC) evaluation for multi-
metric evaluation. However, it should be pointed out that even though an MC evaluation
involves a number of metrics, each addressing different criteria, some metrics could in
fact address the same criterion. Besides selecting which criteria and metrics to use, it
is also important to combine them in a suitable way and to balance trade-offs between
them according to the application at hand. As we will see later on, there are occasionally
conflicts between different criteria and one has to select an algorithm that balances the trade-offs in an acceptable manner. For instance, the most accurate classifier might take too long to generate.

Thus, there are a number of motivations for using an MC approach instead of the traditional single-criterion (SC) approach. However, MC evaluation is more complex than SC evaluation due to the computation and integration of several metrics. Additionally, it is often hard to compare different MC solutions since they might be hard coded for a particular application or described using different and sometimes ambiguous terminology. We therefore believe it is important to work toward a standardized and generic MC method for evaluation.

5.2 Generic Multi-Criteria Methods

Generic MC methods can be described as frameworks which contain rules for how to combine, weigh or in other ways balance multiple evaluation metrics. They are generic in the sense that arbitrary metrics can be integrated. Thus, the actual metrics are not predetermined, i.e., dictated by the design of the method. The choice of metrics could then be based on the problem at hand. We will soon review three existing generic MC methods. However, since many studies use different terminology, we will first discuss some of the fundamental elements and concepts of MC evaluation.

The output of an MC evaluation method can be either single-valued or multi-valued. A method based on single-valued output converts the multiple results into a single value by adding, dividing or multiplying the metric values. A multi-valued result, on the other hand, is often visualized with a graph (Freitas, 2004). However, we will focus only on methods with single-valued output.

The included metrics can be used to assess algorithm-level qualities like training time and memory consumption, or classifier-level qualities like complexity and classification accuracy. In order to simplify discussion, we define metrics as functions with a particular range (e.g., 0 to 1) and type (e.g., discrete or continuous). The input is usually a candidate algorithm, algorithm configuration or classifier and a data set. The output is referred to as the metric score or value for a particular candidate. Metrics can be explicitly weighted to balance trade-offs. Regardless of the existence of an explicit weight, a metric always has an implicit weight. For example, if a metric outputs a percentage, this output could be given in the range of 0 to 1 (implicit weight of 1) or from 0 to 100 (implicit weight of 100). A common procedure to minimize the impact of such implicit weights is to normalize the metric so that all metrics share the same range.

We now give some definitions that will be used to present and compare the existing generic MC methods. We define \( I \) as an index set over the set of included metrics, \( M \), and let \( c \) represent a candidate from a set of candidates, \( C \), while \( D \) represents a data set. Each metric, \( m_i \), can be associated with an explicit weight, \( w_i \), a lower metric bound, \( b_{i_l} \), and an upper metric bound, \( b_{i_u} \).
5.2.1 The Efficiency Method

One of the early studies that investigated the applicability of MC methods for classifier and algorithm evaluation (Nakhaeizadeh & Schnabl, 1997) focused on the fact that most candidates feature both positive and negative properties. Quite intuitively, if a positive property is assessed using some metric the value should be as high as possible and if a negative property is assessed the metric value should be as low as possible. The study introduced a generic MC method, called efficiency, which is based on Data Envelopment Analysis (DEA). With this approach, the efficiency of a candidate is defined as:

\[
\text{efficiency}(c, D) = \frac{\sum_{j \in J} w_j m_j(c, D)}{\sum_{k \in K} w_k m_k(c, D)}
\]  
\hspace{1cm} (5.1)

using \( J \) as an index set over the positive metrics in \( M \) and \( K \) as an index set over the negative metrics in \( M \). What really separates this method from the other is that the weight for each metric is selected automatically by an optimization algorithm with the goal of maximizing the efficiency score. The idea is that the method should find the most efficient candidate(s) out of a set of candidates by optimizing the efficiency of all candidates so that at least one candidate has an efficiency score equal to 1 and no candidate has a higher score. The candidates that achieve an efficiency score equal to 1 together form the efficiency frontier.

5.2.2 The SIM Method

The Simple and Intuitive Measure (SIM) (Soares et al., 2000) is based on the notion of distance between an evaluated candidate and the optimal candidate (for which all metric values are optimal). This distance, denoted the SIM score, is defined as:

\[
\text{SIM}(c, D) = \prod_I |m_i(c, D) - o_i|
\]  
\hspace{1cm} (5.2)

where \( o_i \) is the optimal value for \( m_i \). The SIM method does not allow explicitly setting the weights of metrics in order to balance trade-offs, because it is argued that these compromises are often not quantifiable. Instead, a slightly different approach is used, in which a bound, specifically defined for a particular application, is set for each metric. Adding the concept of bounds the original SIM can be refined to:

\[
\text{BSIM}(c, D) = \begin{cases} 
\text{SIM} & \forall i(m_i(c, D) \in [b_i^l, b_i^h]) \\
\infty & \text{otherwise}
\end{cases}
\]  
\hspace{1cm} (5.3)

The SIM score of a candidate indicates the dissimilarity to the optimal candidate and the BSIM score indicates the dissimilarity to the ideal candidate. Whereas SIM and
BSIM are mainly intended for candidate ranking, the Normalized Bounded SIM (NB-SIM) score is used for the purpose of evaluating a single candidate:

\[
\text{NBSIM}(c, D) = \frac{\text{BSIM}}{\prod_{i} |b_{i}^h - b_{i}^l|}
\]  

(5.4)

The NBSIM score is used to assess the quality of a candidate by representing the dissimilarity to the optimum as the proportion of its BSIM score to the SIM score of the minimally compliant candidate, i.e., the candidate for which all metrics yield their worst possible value. It is argued that the SIM method is adequate for application settings where user preferences are preferably defined as bounds for a set of equally important metrics.

5.2.3 The Measure-Based Method

Andersson et al. (1999) introduced a generic MC framework called measure-based evaluation. The main idea is to define a measure function (MF) for a particular problem. The purpose is to get an explicit distinction between problem formulation, i.e., specifying the measure function and problem solving, i.e., finding a candidate maximizing the measure function. The measure function assigns a value for each combination of problem and candidate describing how well the candidate solves the problem. Thus, as soon as a non-trivial measure function is defined (which is basically any function that does not give the same output for all cases) some candidates will perform better than others. As previously mentioned the measure function should capture, in an algorithm independent way, a suitable algorithm behavior with regard to a particular application. Andersson et al. present a measure function example which focuses on common heuristics or inductive biases of popular algorithms. For instance, the ID3 tree inducing algorithm tries to find the smallest tree (simplicity) that is consistent with the training data (accuracy). The example MF combines three properties: accuracy on the training data, similarity and simplicity (the inverse of complexity). The similarity property indicates to what extent instances that are close to each other geometrically get assigned the same class label. Measure-based evaluation is very generic in the sense that few restrictions are enforced with regard to weighting and integration. The provided example function can be generalized into a generic method that works for arbitrary metrics, as follows:

\[
\text{MF}(c, D) = \sum_{i} w_{i} m_{i}(c, D)
\]  

(5.5)

5.3 Analysis

Some concerns should be raised about efficiency and DEA since they could prove to be detrimental to the solution of some evaluation problems. Firstly, there is no point in
applying DEA to optimize the weights if the purpose is to evaluate one single candidate, since the objective of DEA optimization is to rank candidates. Secondly, since DEA automatically adjusts the metric weights to optimize the efficiency score it is not possible for the user to specify the weights to reflect the importance of different metrics. Thus, it imposes restrictions which makes it impossible to manually balance trade-offs.

The term efficiency may also be misleading sometimes since the top ranked candidate according to DEA might not be the most efficient for the application at hand. Intuitively, the most efficient candidate would be the candidate with the most attractive metric scores when taking important trade-offs into consideration. However, the candidate ranked highest by DEA merely represents a candidate for which the optimization algorithm was able to adjust the weights in such a way so that it scored the highest efficiency score. It is not hard to come up with a scenario which involves the ranking of candidates using two metrics (one of them being much more important to the application at hand than the other). A candidate which scores low for the more important metric and high for the other metric might still be considered the most efficient if its automatically selected weights yield the highest efficiency score. We also argue that the partitioning of metrics into positive and negative is a little bit arbitrary. For instance, accuracy could be a positive metric (as it is) or a negative metric if transformed to error rate (1-accuracy). Furthermore, the efficiency score is not defined for problems without any negative metrics (division by zero). Thus, it seems better to use the summation approach as done in measure-based evaluation to avoid this problem.

Considering the SIM method, it assumes that if a candidate is within bounds all criteria have equal importance but still claims that the compromise between different criteria is application-specific. We agree with this later claim and argue that SIM is overly restrictive for a generic MC method, which should include the possibility to specify weights. If a particular application depended on an equal metric importance idea this could easily be achieved by setting all explicit weights to the same value. It is also argued in the paper about SIM that the user may not be able to select explicit weights with such detail as required by DEA (Soares et al., 2000). This argument seems somewhat confusing since the explicit weights are tuned automatically by an optimization algorithm for the DEA method.

One of the problems that we identified with the SIM method has to do with the dissimilarity measure, i.e., the difference between the optimal candidate and the evaluated candidate. The most relevant candidate for the solution of a particular problem might not always be the (optimal) candidate for which all metrics output the maximum scores, but rather it could be the candidate with the best balanced metric results. We argue that the most valuable component of the SIM method is the bounding feature, followed by normalization. Bounding lets SIM make sure that real-world application constraints are enforced by assigning a score of \( \infty \) for a candidate of which at least one metric is out of bounds.

MF evaluation is the most generic of the three reviewed methods. In general it does
not enforce many restrictions on weighting or integration of metrics. As an example, the measure function instance provided in the study of measure-based evaluation (Andersson et al., 1999) is defined as the sum of accuracy and similarity subtracted by complexity, using different explicitly specified weights for each metric. Rather than focusing on how to integrate arbitrary metrics, the MF study essentially focuses on the presentation of the concept of measure-based evaluation, i.e., how to separate problem formulation and solution by establishing a measure function. Since normalization is not applied a metric with a high maximal value could totally dominate other metrics. There is no discussion about acceptable metric ranges either, which would mean that a measure function will output results as usual even though one metric might have been out of bounds, considering the application at hand.

In conclusion, we have identified a number of attractive properties of possible multi-criteria evaluation methods. First of all, it is important that an arbitrary number of metrics can be integrated and that the method itself does not dictate which metrics should be used. Secondly, one should be able to specify explicit weights for each metric to make it possible to properly represent trade-offs. Finally, one should be able to specify the acceptable range for each metric, pertaining to the application at hand.

5.4 A New Generic MC Method

We now present a new generic MC method, called the candidate evaluation function (CEF), based on a synthesis of the reviewed methods. Starting from MF evaluation, which is the most generic method, we employ explicit weighting and simple addition of metrics. We have included concepts from SIM, namely bounding and normalization. The concept of bounding enables us to check if application domain constraints are violated. The use of normalization makes sure all metrics have the same range, which is a prerequisite for meaningful weighting. We let the total result be zero if at least one metric violates a bound. We extend the bounding concept to give each metric not just a lower but also an upper bound. The bounds effectively generate an acceptable range. Metrics are normalized so that \( m_i(c, D) \in [0, 1] \) where 1 is the best value. If a metric value is higher than the upper bound \( m_i(c, D) > b^U_i \) then it is set to 1. We define CEF as:

\[
CEF(c, D) = \begin{cases} 
\sum_i w_i m_i(c, D) & \forall i (m_i(c, D) \in [b^L_i, 1]) \\
0 & \text{otherwise,} 
\end{cases} \tag{5.6}
\]

in which \( \sum_{i \in I} w_i = 1 \) assures that CEF \((c, D) \in [0, 1] \).
5.5 Case study

We now present a study investigating how data mining can be applied to prevent spyware (Boldt, Jacobsson, Lavesson, & Davidsson, 2008). The occurrence of spyware in applications available over the Internet has become very common. Ideally, countermeasures should not only remove unwanted software, but rather prevent spyware from ever entering computers. The presented study tries to take advantage of the fact that the vendors of spyware-hosting applications try to pose their software as legitimate. There are typically two objectives that counteract in that they want users to download applications that covertly install spyware but without any legal consequences. A common solution is to mention in the End User License Agreement (EULA) that spyware will indeed be installed but to give this information in a way that is hard to understand. The study addresses the spyware problem by mining EULAs of both legitimate (good) and spyware-hosting (bad) software looking for patterns in order to determine if it is possible to detect from the EULA whether the associated software hosts spyware or not. The conducted experiment involved evaluating 15 learning algorithms on a data set containing 50 good and 50 bad EULAs. The results, which can be viewed in Table 5.1, were promising and so a tool for spyware prevention was suggested. This tool could operate between the operating system and any application to be installed, by classifying the EULA as soon as it appears during installation and providing the user with recommendations about whether to install or not. The tool could also visualize which parts of the EULA that contributed to its classification. We need to make sure that the tool is accurate since it essentially should be able to detect a very high quantity of bad software.

5.5.1 CEF Evaluation

Analyzing the requirements it is clear that we need metrics for accuracy (of classifying both good and bad applications), time, and explainability. Mapping the requirements to the available experiment data and making informed choices (based on a limited set of data) about bounds and explicit weighting, we suggest the following setup. We let the true positives rate (TPR) and false positives rate (FPR) from the original experiment reflect the correctness of the classification of good software and bad software, respectively. Furthermore, we establish a very simple explainability metric (EXP) which assigns 0 to non human-understandable classifiers and 1 to classifiers that can be interpreted (e.g., rules or trees). Finally, we use the original testing time data (the number of seconds elapsed during the classification phase of the evaluation) as a metric representing the classification time (TET) of the tool. We present a simple instantiation of the CEF measure in Table 5.3. For this case study we have decided the lower and upper bound, as well as the explicit weight, for each metric after informal discussions with domain experts. We have selected acceptable lower bounds for TPR and FPR considering the small amount of data and we have adopted a weight scheme that simply ranks the met-
Table 5.1: Original experimental results: accuracy, true positives rate (TPR), false positives rate (FPR), and testing time (TET) for 15 algorithms. All results are presented with the mean of 10 runs of holdout using a 66% training set / 34% test set randomized split

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
<th>TPR</th>
<th>FPR</th>
<th>TET</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaBoostM1</td>
<td>73.82</td>
<td>0.72</td>
<td>0.24</td>
<td>0.00</td>
</tr>
<tr>
<td>DecisionStump</td>
<td>68.82</td>
<td>0.54</td>
<td>0.16</td>
<td>0.00</td>
</tr>
<tr>
<td>HyperPipes</td>
<td>76.47</td>
<td>0.91</td>
<td>0.38</td>
<td>0.07</td>
</tr>
<tr>
<td>IBk</td>
<td>77.94</td>
<td>0.71</td>
<td>0.15</td>
<td>0.13</td>
</tr>
<tr>
<td>J48</td>
<td>73.24</td>
<td>0.72</td>
<td>0.26</td>
<td>0.00</td>
</tr>
<tr>
<td>JRip</td>
<td>71.18</td>
<td>0.71</td>
<td>0.29</td>
<td>0.00</td>
</tr>
<tr>
<td>KStar</td>
<td>59.71</td>
<td>0.96</td>
<td>0.77</td>
<td>9.20</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>79.41</td>
<td>0.91</td>
<td>0.32</td>
<td>0.11</td>
</tr>
<tr>
<td>NaiveBayesMultinomial</td>
<td>87.94</td>
<td>0.88</td>
<td>0.12</td>
<td>0.00</td>
</tr>
<tr>
<td>PART</td>
<td>72.65</td>
<td>0.72</td>
<td>0.26</td>
<td>0.00</td>
</tr>
<tr>
<td>RandomForest</td>
<td>75.29</td>
<td>0.79</td>
<td>0.28</td>
<td>0.00</td>
</tr>
<tr>
<td>RBFNetwork</td>
<td>77.35</td>
<td>0.75</td>
<td>0.21</td>
<td>0.17</td>
</tr>
<tr>
<td>Ridor</td>
<td>67.65</td>
<td>0.63</td>
<td>0.28</td>
<td>0.00</td>
</tr>
<tr>
<td>SMO</td>
<td>83.53</td>
<td>0.78</td>
<td>0.11</td>
<td>0.00</td>
</tr>
<tr>
<td>VotedPerceptron</td>
<td>81.47</td>
<td>0.85</td>
<td>0.22</td>
<td>0.02</td>
</tr>
</tbody>
</table>
Table 5.2: Multi-criteria evaluation results. Each metric has been normalized (the range is from 0 to 1 and higher values are better). Five algorithms get a CEF score of 0 since they are out of bounds according to one or more criteria

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TPR</th>
<th>FPR</th>
<th>EXP</th>
<th>TET</th>
<th>CEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaBoostM1</td>
<td>0.72</td>
<td>0.76</td>
<td>0</td>
<td>1.00</td>
<td>0.67</td>
</tr>
<tr>
<td>DecisionStump</td>
<td>0.54</td>
<td>0.84</td>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HyperPipes</td>
<td>0.91</td>
<td>0.62</td>
<td>0</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>IBk</td>
<td>0.71</td>
<td>0.85</td>
<td>0</td>
<td>1.00</td>
<td>0.72</td>
</tr>
<tr>
<td>J48</td>
<td>0.72</td>
<td>0.74</td>
<td>1</td>
<td>1.00</td>
<td>0.81</td>
</tr>
<tr>
<td>JRip</td>
<td>0.71</td>
<td>0.71</td>
<td>1</td>
<td>1.00</td>
<td>0.79</td>
</tr>
<tr>
<td>KStar</td>
<td>0.96</td>
<td>0.23</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>0.91</td>
<td>0.68</td>
<td>0</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>NaiveBayesMultinomial</td>
<td>0.88</td>
<td>0.88</td>
<td>0</td>
<td>1.00</td>
<td>0.77</td>
</tr>
<tr>
<td>PART</td>
<td>0.72</td>
<td>0.74</td>
<td>1</td>
<td>1.00</td>
<td>0.81</td>
</tr>
<tr>
<td>RandomForest</td>
<td>0.79</td>
<td>0.72</td>
<td>0</td>
<td>1.00</td>
<td>0.68</td>
</tr>
<tr>
<td>RBFNetwork</td>
<td>0.75</td>
<td>0.79</td>
<td>0</td>
<td>1.00</td>
<td>0.71</td>
</tr>
<tr>
<td>Ridor</td>
<td>0.63</td>
<td>0.72</td>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SMO</td>
<td>0.78</td>
<td>0.89</td>
<td>0</td>
<td>1.00</td>
<td>0.75</td>
</tr>
<tr>
<td>VotedPerceptron</td>
<td>0.85</td>
<td>0.78</td>
<td>0</td>
<td>1.00</td>
<td>0.72</td>
</tr>
</tbody>
</table>
Table 5.3: A simple instantiation of the generic multi-criteria method, CEF, with four metrics (three continuous, one discrete). The original range is given, whereas the upper and lower bounds have been normalized. The lower bounds can be explained as follows: TPR must be higher than 70%, FPR must be lower than 30%, and testing time must not take longer than 1 second (1.0/9.20 ≈ 0.89). The explicit weights sum up to 1

<table>
<thead>
<tr>
<th>Metric</th>
<th>Type</th>
<th>Range</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPR</td>
<td>continuous</td>
<td>[0 . . 1]</td>
<td>0.70</td>
<td>1.0</td>
<td>2/7</td>
</tr>
<tr>
<td>FPR</td>
<td>continuous</td>
<td>[0 . . 1]</td>
<td>0.70</td>
<td>1.0</td>
<td>3/7</td>
</tr>
<tr>
<td>EXP</td>
<td>discrete</td>
<td>0, 1</td>
<td>0</td>
<td>1</td>
<td>1/7</td>
</tr>
<tr>
<td>TET</td>
<td>continuous</td>
<td>[0 . . 1]</td>
<td>0.89</td>
<td>1.0</td>
<td>1/7</td>
</tr>
</tbody>
</table>

5.6 Conclusions and Future Work

We investigate the concept of generic multi-criteria (MC) evaluation and perform a comparison of existing methods. We synthesize features from these methods into a new MC method, denoted the candidate evaluation function (CEF), and present a case study in which we apply it and show how to balance critical trade-offs according to the application at hand. The results show that CEF indeed provides a way to customize the evaluation so that several criteria can be balanced and assessed. It also provides a way to check that the requirements of an application are honored. There are several directions for future work. Firstly, we are investigating the concept of measure-based algorithms (Andersson et al., 1999) in which a measure function is defined for a particular problem and used as an inductive bias to reformulate the learning problem as an optimization
problem. We have generalized this idea to exchange the performance element of existing algorithms like back-propagated neural networks and rule learners with CEF. Secondly, we intend to develop a prototype of the spyware prevention tool and analyze performance on a larger data set in order to find the most suitable candidate algorithm(s) to include in the tool. Thirdly, we are working on a theoretical study of multi-criteria metrics which suggests the use of software engineering concepts like quality attributes and quality metrics to simplify taking the step from application requirements to a working multi-criteria evaluation.

Acknowledgments

The authors would like to thank Johan Holmgren at Blekinge Institute of Technology for fruitful discussions about the candidate evaluation function (CEF).
6.1 Introduction

One of the most common problems in data mining is that of generating classifiers from data sets consisting of instances where the class membership is known. The objective is to generate classifiers that can be used to accurately predict the class of unknown data instances of the same kind. Supervised concept learning algorithms have shown to be quite reliable in solving this task.

As a consequence, supervised concept learners are increasingly applied to solve real-world problems, for example by finding useful patterns in large databases.

6.1.1 Motivation

Experimental machine learning research often involves evaluating algorithms and classifiers using a particular evaluation metric due to its perceived or proven ability to measure predictive performance. However, the requirements of real-world applications often imply that a suitable trade-off between several important criteria is desired (Joachims,
2005). In addition, the importance of different criteria and the notion of what is a suitable trade-off vary between different applications.

The basic approach for handling this issue is to evaluate a number of algorithms using different metrics for all the important criteria and select the algorithm that is the most successful at balancing the relevant trade-off. For instance, one might decide on the algorithm that is the best at optimizing a certain single or multi-criteria performance metric that captures an application-specific trade-off.

A more sophisticated approach, however, is to try to increase the performance of the candidate algorithms during the actual learning phase by adapting them to optimize metrics relevant to the problem at hand, and then continue with the evaluation to be able to pick the best algorithm.

All supervised concept learning algorithms are biased as a consequence of their data structure (representational bias) and the way they select, or search for classifiers (algorithmic bias). This selection or search is augmented through the use of an embedded performance metric.

We investigate the relationship between the embedded metric and the target evaluation metric. That is, we are interested in finding out if it is possible to achieve better performance, given a certain metric, if the identical metric is also used as the embedded performance metric. If this is indeed the case, this type of metric-based learning could provide us with a new way to increase the potential of existing learning algorithms and it may also enhance our understanding of relationship between the representational and the algorithmic biases.

6.1.2 Outline

The remainder of this paper is organized as follows. In Section 6.2 we give a more detailed description of the problem domain. We describe the concept of embedded and evaluation metrics and discuss related work. We continue by introducing the concept of metric-based learning and describe a rule learning algorithm that is based on this concept in Section 6.3. The empirical experiments and their results are presented in Section 6.4. Finally, we analyze and discuss the results in Section 6.5 and end with conclusions and pointers toward future work in Section 6.6.

6.2 Background

We shall now discuss metrics in general and particularly the difference in using metrics inside algorithms as opposed to using them for evaluation. In the area of supervised concept learning, a metric is commonly used as a basis for evaluating or comparing learning algorithms, or their generated classifiers. The metric tries to capture to which extent an algorithm or classifier meets some given criterion. In a majority of the cases, the criterion is related to performance. There are several performance related criteria,
like time and space. However, the most common criterion for machine learning and data mining applications is arguably correctness. For instance, the accuracy metric is often used to measure the performance of a classifier in terms of correctness. However, other criteria like complexity, usefulness, and explainability are becoming increasingly popular.

Often there are several alternative metrics that can be used to evaluate a certain criterion. In this case we have to choose the most appropriate metric; perhaps one of the metrics has been proven empirically to be very successful, in general, at measuring the criterion? It is also quite common to choose a metric on the basis of its appropriateness for a particular application or a studied problem.

Finally, it is well-known that certain metrics are more popular than others in some areas of research. Some key characteristics of the data sets vary between different areas and researchers have to investigate which metrics best suit their particular domain. Although, we could speculate that some of area-specific choices of metrics can be attributed to tradition or a limited knowledge of what has been proven to work in similar research domains.

6.2.1 Inherent Metrics

Given an arbitrary data set it is possible to define the classifier space, that is, the complete set of possible classifiers for that particular data set. The inductive bias of a supervised concept learning algorithm narrows down the classifier space to a subset of classifiers, which can be described by the algorithm representation, and also dictates how to select a classifier from this subset. Thus, we can conceptually divide the inductive bias into two components: the representational bias that delimits the searchable space and the algorithmic bias that dictates how this space is searched.

For example, the network size and structure affect the representational bias of traditional back-propagated neural network learners while the back-propagation weight-adjusting training algorithm affects the algorithmic bias. Similarly, the allowed number of rules and the number of possible antecedents for each rule affect the representational bias of rule inducers while the rule generator and pruning methods affect the algorithmic bias.

An important aspect of these weight adjusters, rule generators, pruning techniques, and other training algorithms, is the ability to determine if one hypothesis is better than another. For this purpose, these algorithms make use of one or more inherent performance metrics.

For instance, back-propagated neural networks (Rumelhart et al., 1986) use mean-squared error or zero-one loss, rule learners such as Ripper (Cohen, 1995) and IREP (Fürnkranz & Widmer, 1994) use zero-one loss, coverage and information gain, and the C4.5 decision tree inducer (Quinlan, 1993) uses information gain and can optionally prune using zero-one loss and complexity.
Whereas the representational bias in most cases is rather explicit and easily understood by the user, the algorithmic bias is often implicit and difficult to grasp. The general goal of this work is to make the algorithmic bias more explicit, simple to comprehend, and even to a certain extent controllable by the user.

The underlying idea is to capture as much as possible of the algorithmic bias in an explicit metric which is used as a replacement for the inherent performance metric by the algorithm to generate the most appropriate classifier, given a set of application requirements that can be expressed as one or more application-specific metrics. We will refer to this class of supervised concept learning algorithms as metric-based algorithms.

### 6.2.2 Evaluation Metrics

The traditional method for evaluating classifiers has been to estimate the predictive accuracy via statistical tests, for example, cross-validation (Stone, 1974) or the bootstrap (Efron, 1979). More recently, it has been shown that performing ROC analysis is more suitable than estimating predictive accuracy for many problems (Provost et al., 1998) since ROC analysis does not depend on class distribution or misclassification cost.

However, algorithms designed to generate classifiers that maximize classification accuracy may not produce classifiers that achieve the best possible ROC result (Cortes & Mohri, 2004). Additionally, there are a large number of other metrics, which are more or less appropriate, for different classes of problems (Caruana & Niculescu-Mizil, 2006). In fact, real-world applications often have to meet several criteria, which would require the use of customized multi-criteria metrics (Joachims, 2005). Therefore, it seems plausible to assume that if the algorithmic bias of a learning algorithm is modified to optimize a particular metric it would possibly yield a better result compared to the original algorithm, when evaluated using that particular metric.

### 6.2.3 Definitions

We first define the instance space, \( I \), as the set containing all possible instances for a particular problem. Moreover, we define the class space, \( K \), as the set containing all possible classes, or outcomes, for this problem. For the purpose of our study, we may then use the common definition of a classifier, \( c \), as a mapping from instance space, \( I \), to class space, \( K \) (Lavesson & Davidsson, 2007):

\[
c : I \rightarrow K
\]  

(6.1)

The classifier space, \( C \), is then defined as the set of all possible classifiers, that is, all possible mappings between instance space and class space:

\[
C = I \times K
\]  

(6.2)
Additionally, we define a metric to be a function, \( m \), that assigns a scalar result value, \( v \), to each pair of classifier, \( c \in C \), and data set, \( D \):

\[
m(c, D) = v
\]  

(6.3)

The inductive bias of an algorithm, \( a \), can be divided into the representational bias and the algorithmic bias. The former limits the classifier space, thus it indirectly generates a subset, \( C_a \subset C \), of this space. The latter dictates how to search in, or select from \( C_a \).

The definition of a metric, as given in Equation 6.3, is sufficient for theoretical discussions about some of the most commonly used metrics. However, how do we know what requirements to impose on metrics that are to be used as embedded metrics for algorithms? For instance, the learning algorithm itself might restrict which metrics can be used in terms of the range and which information is provided as input for the metric. In addition, one may need to trade-off several metrics. To be able to easily change the inherent metric of an algorithm we need to add some generic mechanism to allow for embedding alternative metrics without changing the algorithm fundamentally.

In a recent paper (Lavesson & Davidsson, 2008) we identified a number of attractive properties of existing multi-criteria evaluation metrics and presented a generic multi-criteria metric that we designed with these properties in mind. This metric, called the Candidate Evaluation Function (CEF), has the main purpose of combining an arbitrary number of individual metrics into a single quantity.

CEF normalizes the metrics in order to get a uniform output domain and it is also possible to specify explicit weights for each metric to ensure that application-specific trade-offs can be properly represented. CEF itself does not dictate which metrics to use, it merely dictates how metrics are combined. Finally, CEF makes it possible to specify the acceptable range for each metric, pertaining to a particular application.

We define \( m_j \) as a metric with index \( j \) from an index set, \( J \), over the selected set of metrics. Each metric is associated with a weight, \( w_j \), and an acceptable range, \( r = [b_{lj}, b_{uj}] \). The lower bound, \( b_{lj} \), denotes the least desired acceptable score. Similarly, the upper bound, \( b_{uj} \), denotes the desired score. Note that, in the original CEF definition a metric was normalized according to the best and worst score of that particular metric obtained from the studied set of classifiers. The current normalization uses the lower and upper bound to generate a smooth distribution from 0 (least desired) to 1. CEF is
now defined as specified in Equation 6.4.

\[
CEF(c, D) = \begin{cases} 
0 : \exists j(\overline{m}_j(c, D) < 0) \\
\sum_{j \in J} w_j \overline{m}_j(c, D) \text{ otherwise}
\end{cases}
\]

where \( \sum_{j \in J} w_j = 1 \) and

\[
\overline{m}_j(c, D) = \begin{cases} 
1 : \frac{m_j - b_{j^*}}{b_{j^*} - b_j} > 1 \\
\frac{m_j - b_{j^*}}{b_{j^*} - b_j} \text{ otherwise}
\end{cases}
\]  

(6.4)

6.2.4 Related Work

Which performance metrics are important in a particular situation depends on the application. Thus, for a given real-world problem some specific metrics are often more important than others. Let us refer to these metrics as application-specific metrics. Supervised concept learning algorithms that are constructed to maximize a particular metric during the learning phase may not produce classifiers that achieve the best possible score according to the application-specific metrics.

For example, a certain application might depend on finding classification rules with few antecedents and that achieve a low false positives rate for a particular class while the selected supervised concept learning algorithm might be designed to maximize accuracy. If the class distribution is skewed in such a way that most instances do not belong to the application important class, the generated classifier might actually contribute negatively to the application metrics.

Optimizing Accuracy

A number of studies address the question of how to optimize supervised concept learning algorithms toward different objectives. We distinguish between three existing approaches. The first focuses on improving accuracy by enhancing the inherent metric of popular supervised learners. For instance, it has been shown that the convergence of a back-propagated neural network can be improved by replacing the error function with a log-likelihood cost function (Holt & Semnani, 1990).

Optimizing An Alternative Metric

The second approach goes one step further and tries to optimize metrics other than accuracy by replacing the inherent metric. Notable examples of such studies include the optimization of ROC using decision trees (Ferri et al., 2002) and gradient-descent (Herschtal & Raskutti, 2004), as well as optimization of the f-measure using support vector machines (Musicant et al., 2003).
Optimizing Multiple Metrics

The third approach aims to optimize more than one metric, either by replacing the inherent metric of an existing algorithm with a multi-criteria metric, or by developing a new algorithm that optimizes such a metric. For example, the support vector machines algorithm has been generalized to optimize multi-criteria non-linear performance metrics (Joachims, 2005), and dynamic bias selection has been implemented for prediction rule discovery (Suzuki & Ohno, 1999).

In addition, Andersson et al. (1999) present an approach called measure-based evaluation and describes how to implement hill-climbing learning algorithms that optimize a multi-criteria metric, called the measure function. By designing a measure function for a particular problem it is possible to get an explicit distinction between the problem formulation, that is, specifying the measure function, and problem solving: finding a classifier that maximizes the measure function. By making this distinction, it is possible to isolate the meta-knowledge necessary for classifier selection from the details of the learning algorithms.

6.3 Metric-based Learning

Many researchers have studied the relationship between the algorithmic and the representational bias. For example, see the study on evaluation and selection of biases in machine learning (Gordon & Desjardins, 1995) or an earlier study about the need for biases in learning generalizations (Mitchell, 1980). Additionally, another study (Lavesson & Davidsson, 2006) tries to empirically evaluate the generic performance of different learning algorithms by formulating metrics that summarize performance over a large set of classifiers generated by different algorithm configurations.

The focus of this study is very close to the mentioned relationship since we investigate the possibility of increasing the performance of classifiers generated by a learning algorithm with a certain representational bias by changing its algorithmic bias.

One question that is only slightly addressed in this study is how much impact this change will have for a certain representational bias. That is, how much potential there is in using a particular representational bias for metric-based learning. Obviously, one has to compare the performance of one particular embedded metric in conjunction with algorithms that use different representational biases in order to address that question more fully.

Researchers in the field of meta-learning have pointed out the importance of going beyond the engineering goal of producing more accurate learners to the scientific goal of understanding learning behavior (Giraud-Carrier et al., 2004). We believe that it is possible to move closer to this goal by conducting research into some of the core concepts of metric-based learning. However, in this study we focus on one specific representational bias only.
6.3.1 A Metric-based One Rule Inducer

In a seminal work in the area of machine learning (Holte, 1993) it was shown that very simple classification rules perform quite well on most commonly used data sets. The study featured a supervised concept learning algorithm, called One Rule that actually achieved results comparable to those achieved by the C4.5 decision tree inducer. However, the algorithm only used accuracy as an inherent metric and the evaluation results were also only provided for the accuracy metric.

We have developed A Metric-based One Rule Inducer (AMORI). As the name suggests it tries to generate one rule that is optimal, given a selected embedded metric. The pseudo code for the algorithm is given in Figure 6.1. We have implemented AMORI in Java by extending the Weka (Witten & Frank, 2005) Classifier class. Thus, it is compatible with the Weka ARFF data set format and can be evaluated using, for instance, the Weka Evaluation class.

The AMORI Algorithm

The algorithm takes a set of training instances as input and generates one rule, consisting of zero or more antecedents and one consequent. Each antecedent consists of an attribute, a conditional operator, and a value. The implementation currently only supports two-class (binary) target attributes and input attributes of types nominal and numeric, which are handled as follows.

The rule can only use a specific nominal attribute in an antecedent once (using the $=$ operator). However, attributes of numeric type can be used twice, since there are two operators available ($>$ and $\leq$). The possible values, which an attribute can be compared with, are taken from the training set. Thus, it is possible to calculate an upper bound on the number of possible rules that can be generated for a certain data set. The algorithm selects the minority class as consequent and classifies all uncovered instances as belonging to the majority class. After calculating the default rule score, that is, the performance score for an empty rule (consisting of zero antecedents and one consequent), the algorithm performs an exhaustive search in which each attribute is matched with the compatible operator(s) and the values that exist for the attribute in the training set. Each possible antecedent is added to the rule, prior to a temporary metric evaluation, and then removed again. When the exhaustive search is complete, the best found metric score is compared to the overall rule score (which is the default rule score during the first run).

If the best found metric score is worse than the overall rule score, the training is stopped. Otherwise, the corresponding antecedent is added to the rule and an exhaustive search is performed yet again, on the remaining attributes. For each attribute, the algorithm checks to see if it has been used before and omits any evaluation using that particular attribute if it is already in use, according to the usage restrictions expressed earlier.

The generated classifier assigns the class label that corresponds to the consequent of
Require: data - a set of training instances
Require: metric - a CEF metric

while running do
    consequent ⇐ leastFrequentClass
    ruleScore ⇐ evaluation(data, metric)
    for a = 0 to a < numAttributes do
        if a is not used then
            antd ⇐ bestAntd(a, data)
            rule.add(antd)
            tempScore ⇐ evaluation(data, metric)
            rule.removeLast()
            if tempScore > bestAntdScore then
                bestAntdScore ⇐ tempScore
                bestAttribute ⇐ a
                bestGlobalAntd ⇐ antd
            end if
        end if
    end for
    if bestAttributeScore < ruleScore then
        running ⇐ false
    else
        rule.add(bestGlobalAntd)
        mark a as used
    end if
end while

Figure 6.1: Pseudo code for A Metric-based One Rule Inducer (AMORI)
the created rule for instances that are covered by the rule. If an instance is not covered by the rule, the classifier instead assigns the majority class.

**AMORI Complexity**

The representational bias of AMORI limits the classifier space, $C$, into a subset, $C_{AMORI}$. In order to calculate the number of possible classifiers, we first define a general rule for a data set, $D$, of $x$ attributes:

$$r = \alpha_1 \land \ldots \land \alpha_x$$

(6.5)

The possible number of antecedents for a particular nominal attribute, $y$, can then be defined as $|\alpha_y| = |V_y| + 1$, where $V_y$ is the set of possible values, found in the training set.

Similarly, the number of possible antecedents for a particular numeric attribute, $z$, can then be defined as $|\alpha_z| = 2|V_z| + 1$, where $V_z$ is the set of possible values, found in the training set. The addition of 1 to each of these sizes is used to represent the case when the particular attribute is not used at all.

The size of $C_{AMORI}$ can then be defined as:

$$|C_{AMORI}| = |r| = \prod_{l=1}^{l<x} |\alpha_l|$$

(6.6)

**6.3.2 Metrics for Metric-based Algorithms**

Machine learning evaluation metrics are predominantly used as components in classifier evaluation methods. If instead, they are to be used as embedded metrics there are a number of issues that need to be addressed.

**Required Number of Instances**

First of all, it is important to recognize the fact that some metrics are calculated on a per instance basis, while others summarize the performance over a set of instances. For example, a back-propagated neural network learner processes the data set one instance at a time, which works well in conjunction with the regular inherent error metric.

However, in order to employ metrics such as the area under the ROC curve (AUC), which requires evaluation and ordering of sets of instances, one can only use algorithms that evaluate performance on sets of instances. Consequently, if a back-propagated neural network learner must be used, the choice of embedded metric is restricted to those metrics that can be calculated on a per instance basis and therefore we cannot use AUC.
**Required Types of Information**

Additionally, there exist metrics that require a completely generated classifier for calculation, whereas other metrics can be calculated during the learning process. For example, the Ripper rule learner creates a rule set by generating one rule at a time. When the complete rule set has been generated the algorithm calculates some statistics for each rule and these statistics can be used to give the probability that a certain instance belongs to a certain class.

Consequently, it is important to consider that the embedded metric can be placed at several locations within the algorithm. One approach would be to put it at the generation of each rule or during the pruning of individual rules. Another would be to put it in the optimization stage of the algorithm. Depending on how the metric is integrated, Ripper will provide a different set of information elements for metric calculation.

The statistics and information that can be extracted to be used for calculating metrics also vary between different learning algorithms. For example, the area under the ROC curve metric depends on a ranking of data instances according to the probability that they belong to a certain class. Although many classifiers can provide this probability information, some algorithms can only output classifications.

### 6.4 Experiments

Our main experiment features three instances of AMORI, each using a different embedded metric. The objective with this experiment is to determine if the use of identical evaluation and embedded metrics yields a higher performance in comparison to the use of one particular embedded metric independently of which metric is used for evaluation. In a complementary experiment we also compare the performance of AMORI with some existing rule induction algorithms.

In essence, we want to test the null hypothesis that, for a given evaluation metric, the difference in performance between any of the included algorithms is zero.

Since testing this hypothesis involves the comparison of more than two classifiers on multiple data sets we use the non-parametric Friedman test and the corresponding Nemenyi post hoc test (Demzar, 2006).

The Friedman test is based on the average ranks of each algorithm rather than average performance. The best performing algorithm is awarded the rank of 1, the second best the rank of 2, and so on. In the case of a tie, the average ranks of the tied algorithms are assigned. We can now state the null hypothesis more formally. Given that \( R_j \) represents the average rank of the \( j \)-th out of \( k \) classifiers, we wish to test \( H_0 : R_1 = \cdots = R_k \).

As it has been shown that Friedman’s statistic, \( \chi^2_F \), is too conservative (Iman & Davenport, 1980) we employ the recommended alternative statistic based on the F-distribution, as defined in Equation 6.7 where \( N \) is the number of included data sets and \( k \) is the number of algorithms to compare. Moreover, we use perform hypothesis
testing at $p < 0.05$ and with $k - 1$ and $(k - 1)(N - 1)$ degrees of freedom.

$$F_F = \frac{(N - 1) \chi^2_F}{N (k - 1) - \chi^2_F}$$  \hspace{1cm} (6.7)

If the null hypothesis is rejected, the Nemenyi post hoc test can be used to find out if the performance of two particular classifiers is significantly different. In order for this difference to be significant, the corresponding average ranks of the two classifiers must differ by at least the Critical Difference (CD), which is defined in Equation 6.8, where $q_\alpha$ is the critical value for the two-tailed Nemenyi test at $p < \alpha$.

$$CD = q_\alpha \sqrt{\frac{k (k + 1)}{6N}}$$  \hspace{1cm} (6.8)

We use stratified 10-fold cross-validation tests to measure the performance of each algorithm according to our three selected metrics. The partitioning into folds is carried out by the Weka Evaluation class using a random seed of 1 for each cross-validation test. Thus, given a particular data set, each algorithm has access to identical folds. In essence, this setup also increases the reproducibility of the results.

We now proceed by first presenting the three metrics that are used for the evaluation of all included algorithms and as embedded metrics for AMORI. We then explain the process of data set selection, present the featured data sets, and give a more detailed description of each experiment.

### 6.4.1 Metrics

Many of today’s most frequently applied metrics are used for evaluating the same basic trade-off, consisting of different proportions of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) (Witten & Frank, 2005). The question of which trade-off is more useful to consider varies across different problem domains. For instance, the F-measure (FME) and precision vs. recall are quite common in the area of information retrieval while sensitivity and specificity are more prominent in medical applications. Classification accuracy (ACC), or success rate, has traditionally been the most commonly used metric for evaluating the before mentioned trade-off.

**Classification Accuracy**

The main question that we try to answer in this study is whether it is possible to achieve a performance gain with respect to a certain metric if that particular metric is also used as a component of the algorithmic bias. Thus, the decision on which metrics to incorporate is important. Our first choice is to include classification accuracy as it is still one of the most popular evaluation metrics in machine learning. Coincidentally, it is also frequently used as an inherent metric.
The Area under the ROC Curve

As the first alternative metric, we select the area under the ROC curve metric (AUC), which evaluates the same trade-off as expressed above but in a quite different way compared to the accuracy metric. AUC has become a popular metric for evaluating supervised concept learners because, as opposed to ACC, it does not depend on class distribution or misclassification costs. Many researchers argue that ROC analysis is better than, or at least complements, predictive accuracy estimation as a tool for evaluating classifiers. Although some information is lost when summarizing a ROC curve using this single metric instead of analyzing the actual curve, AUC has still been proven to work quite well as an evaluation metric.

The F-measure

The third and last metric we choose is the F-measure. Again, it tries to capture the same basic trade-off as ACC and AUC do but it is most prominently used in information retrieval. The three selected metrics have a lot in common but what separates them is how they capture the basic trade-off and the preconditions that must be met in order to use them. This makes them good candidates for our particular study.

Conversion to CEF

In order for ACC, AUC, and FME to be used in AMORI they need to be converted to CEF metrics. For the purpose of this particular study, this conversion is effortless. Each metric already has a range of $[0, 1]$ and we do not need to use weights since each CEF instance will only include one metric. However, the abstraction of CEF in AMORI makes it straight-forward to incorporate multiple metrics without changing the algorithm.

6.4.2 Data Sets

The experiments feature 19 data sets from the UCI Machine Learning Repository. A large collection of UCI data sets have been converted to the Weka ARFF file format. We first selected all two-class data sets from this collection and removed sets that featured unsupported attribute types. The compilation of data sets and their statistics are presented in Table 6.5.

In summary, we have performed a systematic selection of data sets on the basis of compatibility with the studied algorithms. Thus, we have not removed or added any data set based on any other criteria or in favor of any algorithm or metric.
6.4.3 Experiment 1

Description

In this experiment we compare three instances of the AMORI algorithm, each using a different embedded metric. From now on, we use the expression, AMORI(\(X\)), to denote a particular AMORI instance that uses \(X\) as its embedded metric. For instance, AMORI(ACC) uses classification accuracy as its embedded metric. We wish to investigate if the use of an arbitrary embedded metric would result in a better performance, according to that particular metric, compared to the use of other embedded metrics.

Note that, in this experiment, we are not interested in comparing different representational biases and their possible impact on performance. Although, in the more general case, it makes sense that some representational biases are more suitable than others when it comes to optimizing different embedded metrics.

For instance, two back-propagated neural networks, each with a different setup of neurons, will both try to optimize the same metric (accuracy), but their performance in doing so could differ greatly.

We divide the experiment into three different cases. In each case we use a different metric for evaluation and calculate the average rank of each AMORI instance according to the performance related to this metric.

Results

In Table 6.1 we present the accuracy (ACC), the F-measure (FME), and area under the ROC curve (AUC) scores for the three included AMORI algorithm instances. Moreover, Table 6.2 shows the average rank for each algorithm and metric combination (a low rank indicates good performance). The null hypothesis was tested separately for each of the three evaluation metrics using the Friedman test (as described in Section 6.4). The test statistic, \(F_f\), is distributed according to the F-distribution with 2 and 36 degrees of freedom. At \(p < 0.05\) the hypothesis was rejected for the AUC and FME metrics since \(F_f > 3.259\). The subsequent Nemenyi post-hoc test revealed that both AMORI(FME) and AMORI(AUC) were significantly better than AMORI(ACC) when evaluated using their respective metrics since the differences in average rank were higher than the critical difference (\(CD = 0.760\)). AMORI(ACC) did perform better than the two other AMORI instances on the ACC metric but the gain was not significant.

We have now seen that we are able to bias the learning towards a specific goal, but how does AMORI perform compared to state-of-the-art rule learners?
<table>
<thead>
<tr>
<th>Data set</th>
<th>AM(ACC)</th>
<th>AM(AUC)</th>
<th>AM(FME)</th>
<th>AM(ACC)</th>
<th>AM(AUC)</th>
<th>AM(FME)</th>
<th>AM(ACC)</th>
<th>AM(AUC)</th>
<th>AM(FME)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACC</td>
<td>AUC</td>
<td>FME</td>
<td>ACC</td>
<td>AUC</td>
<td>FME</td>
<td>ACC</td>
<td>AUC</td>
<td>FME</td>
</tr>
<tr>
<td>backache</td>
<td>0.861</td>
<td>0.500</td>
<td>0.000</td>
<td>0.828</td>
<td>0.632</td>
<td>0.367</td>
<td>0.839</td>
<td>0.554</td>
<td>0.216</td>
</tr>
<tr>
<td>biomed</td>
<td>0.809</td>
<td>0.745</td>
<td>0.661</td>
<td>0.828</td>
<td>0.789</td>
<td>0.731</td>
<td>0.828</td>
<td>0.786</td>
<td>0.727</td>
</tr>
<tr>
<td>breast-cancer</td>
<td>0.731</td>
<td>0.578</td>
<td>0.306</td>
<td>0.720</td>
<td>0.665</td>
<td>0.529</td>
<td>0.720</td>
<td>0.665</td>
<td>0.529</td>
</tr>
<tr>
<td>colic</td>
<td>0.859</td>
<td>0.829</td>
<td>0.789</td>
<td>0.859</td>
<td>0.829</td>
<td>0.789</td>
<td>0.859</td>
<td>0.829</td>
<td>0.789</td>
</tr>
<tr>
<td>credit-a</td>
<td>0.846</td>
<td>0.849</td>
<td>0.835</td>
<td>0.848</td>
<td>0.853</td>
<td>0.840</td>
<td>0.848</td>
<td>0.853</td>
<td>0.840</td>
</tr>
<tr>
<td>credit-g</td>
<td>0.698</td>
<td>0.503</td>
<td>0.032</td>
<td>0.724</td>
<td>0.634</td>
<td>0.471</td>
<td>0.562</td>
<td>0.567</td>
<td>0.443</td>
</tr>
<tr>
<td>cylinder-bands</td>
<td>0.672</td>
<td>0.612</td>
<td>0.366</td>
<td>0.654</td>
<td>0.606</td>
<td>0.425</td>
<td>0.583</td>
<td>0.601</td>
<td>0.592</td>
</tr>
<tr>
<td>diabetes</td>
<td>0.721</td>
<td>0.636</td>
<td>0.470</td>
<td>0.725</td>
<td>0.689</td>
<td>0.592</td>
<td>0.719</td>
<td>0.700</td>
<td>0.613</td>
</tr>
<tr>
<td>haberman</td>
<td>0.745</td>
<td>0.578</td>
<td>0.316</td>
<td>0.722</td>
<td>0.669</td>
<td>0.514</td>
<td>0.719</td>
<td>0.663</td>
<td>0.506</td>
</tr>
<tr>
<td>heart-statlog</td>
<td>0.711</td>
<td>0.695</td>
<td>0.629</td>
<td>0.744</td>
<td>0.732</td>
<td>0.685</td>
<td>0.726</td>
<td>0.719</td>
<td>0.681</td>
</tr>
<tr>
<td>hepatitis</td>
<td>0.813</td>
<td>0.582</td>
<td>0.293</td>
<td>0.774</td>
<td>0.638</td>
<td>0.426</td>
<td>0.800</td>
<td>0.643</td>
<td>0.436</td>
</tr>
<tr>
<td>ionosphere</td>
<td>0.832</td>
<td>0.773</td>
<td>0.706</td>
<td>0.823</td>
<td>0.761</td>
<td>0.687</td>
<td>0.823</td>
<td>0.763</td>
<td>0.690</td>
</tr>
<tr>
<td>kr-vs-kp</td>
<td>0.752</td>
<td>0.741</td>
<td>0.657</td>
<td>0.756</td>
<td>0.746</td>
<td>0.668</td>
<td>0.663</td>
<td>0.677</td>
<td>0.738</td>
</tr>
<tr>
<td>labor</td>
<td>0.842</td>
<td>0.798</td>
<td>0.743</td>
<td>0.825</td>
<td>0.784</td>
<td>0.722</td>
<td>0.789</td>
<td>0.734</td>
<td>0.647</td>
</tr>
<tr>
<td>liver-disorders</td>
<td>0.664</td>
<td>0.616</td>
<td>0.442</td>
<td>0.672</td>
<td>0.633</td>
<td>0.498</td>
<td>0.577</td>
<td>0.587</td>
<td>0.563</td>
</tr>
<tr>
<td>mushroom</td>
<td>0.784</td>
<td>0.776</td>
<td>0.711</td>
<td>0.784</td>
<td>0.776</td>
<td>0.711</td>
<td>0.900</td>
<td>0.897</td>
<td>0.886</td>
</tr>
<tr>
<td>prnn_synth</td>
<td>0.840</td>
<td>0.840</td>
<td>0.839</td>
<td>0.836</td>
<td>0.836</td>
<td>0.835</td>
<td>0.840</td>
<td>0.840</td>
<td>0.841</td>
</tr>
<tr>
<td>schizo</td>
<td>0.559</td>
<td>0.544</td>
<td>0.272</td>
<td>0.582</td>
<td>0.573</td>
<td>0.437</td>
<td>0.488</td>
<td>0.508</td>
<td>0.652</td>
</tr>
<tr>
<td>sick</td>
<td>0.974</td>
<td>0.831</td>
<td>0.760</td>
<td>0.958</td>
<td>0.915</td>
<td>0.717</td>
<td>0.973</td>
<td>0.890</td>
<td>0.781</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.774</strong></td>
<td><strong>0.686</strong></td>
<td><strong>0.517</strong></td>
<td><strong>0.772</strong></td>
<td><strong>0.724</strong></td>
<td><strong>0.613</strong></td>
<td><strong>0.750</strong></td>
<td><strong>0.709</strong></td>
<td><strong>0.641</strong></td>
</tr>
</tbody>
</table>
Table 6.2: Average ranks and Friedman test scores for all the AMORI instances

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Metric</th>
<th>ACC</th>
<th>AUC</th>
<th>FME</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM(ACC)</td>
<td>ACC</td>
<td>1.737</td>
<td>2.474</td>
<td>2.605</td>
</tr>
<tr>
<td>AM(AUC)</td>
<td>AUC</td>
<td>1.921</td>
<td>1.605</td>
<td>1.868</td>
</tr>
<tr>
<td>AM(FME)</td>
<td>FME</td>
<td>2.342</td>
<td>1.921</td>
<td>1.526</td>
</tr>
<tr>
<td>$F_f$</td>
<td></td>
<td>1.917</td>
<td>4.311</td>
<td>7.863</td>
</tr>
</tbody>
</table>

6.4.4 Experiment 2

Description

For the second experiment, we compare AMORI with other existing rule induction algorithms, in order to find out if our metric-based algorithm can be competitive against the state-of-the-art. For this purpose, we choose to compare AMORI with two quite different rule learners; Ripper (Cohen, 1995) and One Rule (Holte, 1993). Since we use the Weka implementations of these algorithms we will onwards use their Weka class names, JRip and OneR, when making references.

The rationale for comparing AMORI with these particular algorithms is basically that they represent the state-of-the-art in simple and complex rule inducers, respectively. Complexity-wise, it could be argued that AMORI could be placed between these two algorithms.

OneR

OneR generates rules based on one single attribute. The name of the algorithm is in fact somewhat of a misnomer since it actually generates a set of rules for the selected attribute. OneR ranks attributes according to error rate and generates a classifier using the best attribute. The two major differences between AMORI and OneR are that AMORI can generate rules that operate on more attributes and, when using AMORI, it is possible change the embedded metric, which is the basis for the algorithm to choose one rule over another.

JRip

The JRip algorithm is primarily based on the Incremental Reduced Error Pruning (IREP) algorithm. What separates JRip from IREP are three modifications; the pruning metric and stopping criterion have been replaced with more efficient alternatives, and a post-processing technique has been introduced. As stated before, OneR generates a set of
rules for one particular attribute and AMORI generates one rule that could include sev-
eral attributes. JRip, however, can generate a set of AMORI-like rules. In addition,
neither AMORI nor OneR performs any post-processing.

**Results**

Table 6.4 shows the performance per data set and Table 6.3 shows the average rank of
each algorithm for each of the three evaluation metrics. The null hypothesis was again
tested separately for each of the three metrics using the Friedman test and for the F-
distribution we used 2 and 36 degrees of freedom. The hypothesis was rejected for the
ACC and AUC metrics (with $p < 0.05$). We proceeded with the Nemenyi post hoc test,
which revealed that JRip was significantly better than OneR for both metrics. This result
is not particularly controversial since JRip is often regarded as one of the better rule
based algorithms. However, no other significant differences could be found. The order
according to average rank was identical for all metrics: JRip, AMORI, and OneR.

**6.5 Discussion**

When analyzing the average ranks in Table 6.2, it is clear that each AMORI instance
perform better than the two others if its metric is also used for evaluation. However,
this performance gain was only statistically significant at $p < 0.05$ for AMORI(AUC)
and AMORI(FME) over AMORI(ACC). That is, AMORI(ACC) did not significantly
outperform the other instances on its own metric.

There are several issues that need to be further investigated since they might have
influenced the result (in either way). For example, some embedded metrics might work
better for a particular class of problems. Thus, if there is a similar number of data sets
in favor of each metric this would even out the performance gain for each metric.

Related to the discussion about a particular class of problems, we can observe that
several of the studied data sets seem to favor certain embedded metrics, resulting in top
scores for all three metrics. For instance, observe the use of ACC as an embedded metric
for the labor data set or the use of AUC for the heart-statlog data set.

Finally, the performance gain might be related to the AMORI algorithm itself. More
research into this topic is needed in order to fully understand the impact of the repre-
sentational bias of different algorithms when trying to optimize a particular embedded
metric.

**6.6 Conclusions and Future Work**

The requirements of many real-world data mining applications can be formalized into
application-specific metrics that need to be optimized. Supervised concept learning al-

93
Table 6.3: The Friedman test score for each metric and the average ranks of AMORI, JRip, and OneR

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Metric</th>
<th>ACC</th>
<th>AUC</th>
<th>FME</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMORI</td>
<td>ACC</td>
<td>1.921</td>
<td>1.947</td>
<td>1.895</td>
</tr>
<tr>
<td>JRip</td>
<td>AUC</td>
<td>1.632</td>
<td>1.474</td>
<td>1.684</td>
</tr>
<tr>
<td>OneR</td>
<td>FME</td>
<td>2.447</td>
<td>2.579</td>
<td>2.421</td>
</tr>
<tr>
<td>$F_f$</td>
<td></td>
<td>3.714</td>
<td>7.992</td>
<td>3.029</td>
</tr>
</tbody>
</table>

Table 6.4: Experiment 2 results

<table>
<thead>
<tr>
<th>Data set</th>
<th>ACC</th>
<th>AUC</th>
<th>FME</th>
<th>ACC</th>
<th>AUC</th>
<th>FME</th>
</tr>
</thead>
<tbody>
<tr>
<td>backache</td>
<td>0.856</td>
<td>0.497</td>
<td>0.000</td>
<td>0.828</td>
<td>0.495</td>
<td>0.061</td>
</tr>
<tr>
<td>biomed</td>
<td>0.861</td>
<td>0.827</td>
<td>0.785</td>
<td>0.885</td>
<td>0.895</td>
<td>0.844</td>
</tr>
<tr>
<td>breast-cancer</td>
<td>0.657</td>
<td>0.542</td>
<td>0.310</td>
<td>0.710</td>
<td>0.598</td>
<td>0.428</td>
</tr>
<tr>
<td>colic</td>
<td>0.815</td>
<td>0.814</td>
<td>0.764</td>
<td>0.842</td>
<td>0.823</td>
<td>0.775</td>
</tr>
<tr>
<td>credit-a</td>
<td>0.855</td>
<td>0.862</td>
<td>0.850</td>
<td>0.858</td>
<td>0.874</td>
<td>0.843</td>
</tr>
<tr>
<td>credit-g</td>
<td>0.665</td>
<td>0.532</td>
<td>0.264</td>
<td>0.717</td>
<td>0.593</td>
<td>0.428</td>
</tr>
<tr>
<td>cylinder-bands</td>
<td>0.496</td>
<td>0.548</td>
<td>0.596</td>
<td>0.652</td>
<td>0.666</td>
<td>0.559</td>
</tr>
<tr>
<td>diabetes</td>
<td>0.730</td>
<td>0.667</td>
<td>0.543</td>
<td>0.760</td>
<td>0.739</td>
<td>0.629</td>
</tr>
<tr>
<td>haberman</td>
<td>0.732</td>
<td>0.585</td>
<td>0.349</td>
<td>0.729</td>
<td>0.613</td>
<td>0.443</td>
</tr>
<tr>
<td>heart-statlog</td>
<td>0.711</td>
<td>0.706</td>
<td>0.669</td>
<td>0.789</td>
<td>0.781</td>
<td>0.751</td>
</tr>
<tr>
<td>hepatitis</td>
<td>0.832</td>
<td>0.652</td>
<td>0.458</td>
<td>0.781</td>
<td>0.664</td>
<td>0.393</td>
</tr>
<tr>
<td>ionosphere</td>
<td>0.809</td>
<td>0.785</td>
<td>0.724</td>
<td>0.897</td>
<td>0.900</td>
<td>0.858</td>
</tr>
<tr>
<td>kr-vs-kp</td>
<td>0.665</td>
<td>0.657</td>
<td>0.586</td>
<td>0.992</td>
<td>0.995</td>
<td>0.991</td>
</tr>
<tr>
<td>labor</td>
<td>0.754</td>
<td>0.684</td>
<td>0.563</td>
<td>0.772</td>
<td>0.779</td>
<td>0.667</td>
</tr>
<tr>
<td>liver-disorders</td>
<td>0.548</td>
<td>0.535</td>
<td>0.458</td>
<td>0.646</td>
<td>0.653</td>
<td>0.527</td>
</tr>
<tr>
<td>mushroom</td>
<td>0.985</td>
<td>0.985</td>
<td>0.984</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>prnn_synth</td>
<td>0.832</td>
<td>0.832</td>
<td>0.831</td>
<td>0.844</td>
<td>0.824</td>
<td>0.840</td>
</tr>
<tr>
<td>schizo</td>
<td>0.868</td>
<td>0.869</td>
<td>0.867</td>
<td>0.826</td>
<td>0.877</td>
<td>0.818</td>
</tr>
<tr>
<td>sick</td>
<td>0.964</td>
<td>0.892</td>
<td>0.733</td>
<td>0.982</td>
<td>0.948</td>
<td>0.860</td>
</tr>
<tr>
<td>Average</td>
<td>0.770</td>
<td>0.709</td>
<td>0.597</td>
<td>0.816</td>
<td>0.775</td>
<td>0.669</td>
</tr>
</tbody>
</table>
gorithms are usually designed to maximize a predetermined metric, which we denote the inherent metric. Consequently, these algorithms may not generate classifiers that achieve the best possible performance when evaluated using the application-specific metrics.

In order to address this problem, data mining and machine learning researchers have tried various approaches to modify the algorithmic biases of existing learning algorithms so that they optimize application-specific metrics instead of their inherent metric. However, these solutions are often customized for a particular pair of algorithm and metric. That is, they might increase the performance according to the application-specific metrics for a certain real-world application but they are usually not very generic.

As a consequence, we are interested in studying in more general terms if the performance according to an arbitrary application-specific metric can be boosted by incorporating the metric into the algorithmic bias. For this purpose, we have developed A Metric-based One Rule Inducer (AMORI), for which it is possible to select the embedded metric in a simple manner.

We demonstrate the applicability of our approach by conducting an empirical comparison of the performance of three instances of AMORI, each using a different embedded metric. The example metrics used in this study were classification accuracy (ACC), the area under the ROC curve (AUC), and the F-measure (FME). The results for AMORI, at least for the studied problems, indicate that the embedded metric exchange yielded a performance gain. The gain was statistically significant for AMORI(FME) and AMORI(AUC) over AMORI(ACC) with $p < 0.05$.

Naturally, there are some aspects that need to be further studied. For example, if there is a similar number of data sets in favor of each metric this would even out the performance gain. More research into this topic is needed in order to fully understand the impact of the representational bias of different algorithms when trying to optimize a particular embedded metric.

We also compared AMORI with a rule inducer of similar complexity, called One Rule (OneR), as well as with Ripper (JRip), which is a state-of-the-art rule set inducer with post-processing capabilities. JRip was not able to significantly outperform AMORI on any of the metrics. JRip was significantly better than OneR for the ACC and AUC metrics. AMORI outperformed OneR for all three metrics but the gain was not significant in any of the cases.

We have identified a number of interesting directions for future work. First off, it might be the case that the metrics included in this study are too similar to be used for the purpose of trying to establish whether or not it is beneficial to use identical embedded and evaluation metrics. Consequently, it could be fruitful to select a more diverse set of metrics for similar experiments. Another related direction would be to compare different representational biases using the same embedded metric, thus making it possible to measure the impact of metric exchange for different combinations of representational biases and embedded metrics.

In addition, more research is needed to understand how to select an appropriate em-
bedded metric based on a particular problem. As we observed in this study, some embedded metrics seem to be more suitable than others for certain data sets. A first step in this direction could be to establish objective metrics for some of the widely used data sets, like those from the UCI machine learning repository. These objective metrics could be used as benchmark evaluation metrics when comparing different algorithms but, more importantly for our purposes, they could be used as embedded metrics.

Up until now we have only discussed some specific trade-offs, captured in different ways by established metrics from various areas of research. However, as described in the related work section, quite a few studies advocate the use of more application-specific multi-criteria metrics. It would therefore be interesting to use the multi-criteria functionality of the CEF metric, that is built into AMORI, to be able to trade-off several existing metrics during the learning phase.

In addition, one could investigate if it is possible to convert other existing supervised concept learning algorithms into generic metric-based algorithms by incorporating CEF into their algorithmic biases.

**Acknowledgments**

We would like to thank Fahad Khalid for his ideas about metric-based learning and for valuable discussions in general.
<table>
<thead>
<tr>
<th>Data set</th>
<th>Instances</th>
<th>Attributes</th>
<th>Class1</th>
<th>Class2</th>
<th>#Missing</th>
<th>Missing (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>backache</td>
<td>180</td>
<td>6</td>
<td>27</td>
<td>155</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>biomed</td>
<td>209</td>
<td>7</td>
<td>2</td>
<td>75</td>
<td>134</td>
<td>15</td>
</tr>
<tr>
<td>breast-cancer</td>
<td>286</td>
<td>0</td>
<td>10</td>
<td>201</td>
<td>85</td>
<td>9</td>
</tr>
<tr>
<td>colic</td>
<td>368</td>
<td>7</td>
<td>16</td>
<td>232</td>
<td>136</td>
<td>1927</td>
</tr>
<tr>
<td>credit-a</td>
<td>690</td>
<td>6</td>
<td>10</td>
<td>307</td>
<td>383</td>
<td>67</td>
</tr>
<tr>
<td>credit-g</td>
<td>1000</td>
<td>7</td>
<td>14</td>
<td>700</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>cylinder-bands</td>
<td>540</td>
<td>18</td>
<td>22</td>
<td>228</td>
<td>312</td>
<td>999</td>
</tr>
<tr>
<td>diabetes</td>
<td>768</td>
<td>8</td>
<td>1</td>
<td>500</td>
<td>268</td>
<td>0</td>
</tr>
<tr>
<td>haberman</td>
<td>306</td>
<td>2</td>
<td>2</td>
<td>225</td>
<td>81</td>
<td>0</td>
</tr>
<tr>
<td>heart-statlog</td>
<td>270</td>
<td>13</td>
<td>1</td>
<td>150</td>
<td>120</td>
<td>0</td>
</tr>
<tr>
<td>hepatitis</td>
<td>155</td>
<td>6</td>
<td>14</td>
<td>32</td>
<td>123</td>
<td>167</td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>34</td>
<td>1</td>
<td>126</td>
<td>225</td>
<td>0</td>
</tr>
<tr>
<td>kr-vs-kp</td>
<td>3196</td>
<td>0</td>
<td>37</td>
<td>1669</td>
<td>1527</td>
<td>0</td>
</tr>
<tr>
<td>labor</td>
<td>57</td>
<td>8</td>
<td>9</td>
<td>20</td>
<td>37</td>
<td>326</td>
</tr>
<tr>
<td>liver-disorders</td>
<td>345</td>
<td>6</td>
<td>1</td>
<td>145</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>mushroom</td>
<td>8124</td>
<td>0</td>
<td>23</td>
<td>4208</td>
<td>3916</td>
<td>2480</td>
</tr>
<tr>
<td>prnn-synth</td>
<td>250</td>
<td>2</td>
<td>1</td>
<td>125</td>
<td>125</td>
<td>0</td>
</tr>
<tr>
<td>schizo</td>
<td>340</td>
<td>12</td>
<td>3</td>
<td>177</td>
<td>163</td>
<td>824</td>
</tr>
<tr>
<td>sick</td>
<td>3772</td>
<td>7</td>
<td>23</td>
<td>3541</td>
<td>231</td>
<td>6064</td>
</tr>
</tbody>
</table>
7.1 Introduction

The amount of spyware has increased dramatically due to the high value for marketing companies of the information that is collected. Spyware is designed to collect user information for marketing campaigns without the informed consent of the user. This type of software is commonly spread by bundling, i.e., hiding/packaging, it with popular applications available for free download. A spyware application is typically difficult to remove once it has been installed on a computer system and it can seriously degrade system performance and compromise the privacy of the user (Boldt, Carlsson, & Jacobsson, 2004). This paper presents a novel approach based on data mining, aimed at stopping spyware at the door.

From now on we are going to use the terms bad and good to signify applications that host spyware and legitimate applications, respectively. Distributors of bad software usually try to disguise it as good in an attempt to reach as many users as possible. However, to avoid legal repercussions these distributors are required to mention in the End User License Agreement (EULA) that spyware will indeed be installed. Yet, this information
is given in a way most users find difficult to understand. Even EULAs for legitimate software can be hard to comprehend due to their length and their extensive use of legal terminology (Good et al., 2006).

Consequently, we recognize the need for an efficient method for helping users to distinguish between good and bad software during the installation process. If spyware is detected through such a method, it could assist users in keeping their computers clean from bad software by warning them about the application they are about to install.

7.1.1 Background

The increase of spyware has resulted in vast numbers of users experiencing loss of control over personal information and decreased computer performance (Boldt et al., 2004; A. Weiss, 2005). Anti-virus techniques are used for removing malicious software (such as: computer viruses and worms). Malicious software is undoubtedly illegal in most countries but this is not necessarily true for spyware since it resides in a gray zone between what is considered to be legal and illegal. Thus, the techniques used by spyware can be interpreted as either legal or illegal depending on who is asked; what one individual regards as spyware could be considered a legitimate business application by another.

McFedries (2005) explores the purposes behind spyware as well as its commercial impact. The main conclusion is that spyware is a very good means to gain revenue for online marketing companies, since it provides them with personal user information that can be exploited in order to deliver targeted advertisements.

Zhang (2005) makes visible the urgent need for user education to raise awareness about the spyware threats and the methods available for addressing these threats. This is important, Zhang argues, since most computer users cannot keep up with the rapid development of new spyware that affect their computer systems, personal data, and ultimately their privacy.

Moshchuk, Bragin, Gribble, and Levy (2006) introduce methods for measuring spyware activity on the Internet. In 2005, they performed an automatic analysis of 18 million Internet addresses (URLs) for executable files. Out of the 21,200 applications an astonishing 13.4% were identified as spyware.

Fox (2005) presents a report on how user behavior is affected by the occurrence of spyware in home and work computers\textsuperscript{1} based on performing a telephone interview survey, which featured a nationally representative sample of 2,001 adults living in continental United States telephone households. The report states that “only about one in ten internet users say the current practice of clicking through a user agreement or disclaimer is adequate consent to install adware on a person’s computer”. Townsend (2003) elaborates on how spyware infected applications may violate corporate security policies and

\textsuperscript{1}Fox, S.: Spyware - the thread of unwanted software programs is changing the way people use the Internet, http://www.pewinternet.org/pdfs/PIP_Spyware_Report_July_05.pdf (2005)
procedures\textsuperscript{2}, explaining that spyware could circumvent existing security mechanisms used by the corporations, thus enabling the extraction of sensitive and/or classified information. Moreover, Good et al. (2006) point out the fact that users agree to accept spyware as part of a software bundle as a cost associated with gaining the functionality they desire and demonstrate that interface design can be a significant factor in eliciting informed consent to software installation.

7.1.2 Related Work

In a pilot study (Lavesson, Davidsson, Boldt, & Jacobsson, 2008), we investigated whether it was possible to take advantage of the fact that the installation of bad software is mentioned in the EULA. We addressed this problem by applying supervised concept learning algorithms to classify EULAs of both good and bad applications in order to detect if the associated software hosts spyware. The results indicate that the approach is feasible, however the amount of data was scarce (the data set featured 100 EULAs in total). EULA classification, as a problem, is quite analogous to that of spam classification, i.e., to distinguish between unsolicited commercial emails (spam) and legitimate emails. Much work has been done in the area of spam classification, e.g., using different learning algorithms, such as: rule learners, support vector machines, instance-based learners, decision trees, and stacking (Cohen, 1996; Drucker, Wu, & Vapnik, 1999; Androussopoulos et al., 2000; Carreras & Márquez, 2001; Sakkis et al., 2001). More recently, Koprinska, Poon, Clark, and Chan (2007) investigate the performance of the random forest algorithm for the same type of problem, claiming that it outperforms some of the earlier mentioned algorithms on several problems. Kang, Domeniconi, and Barbara (2005) also study the spam classification problem and applies an unsupervised feature selection algorithm and clustering to classify unlabeled documents. The results from the analysis can be interpreted as follows: the absence of a certain term is a characteristic shared across the emails of a given category; whereas the presence of certain keywords shows a larger variability across emails of a given category.

7.1.3 Scope and Aim

The aim of this paper is to further investigate EULA classification as a means for categorizing software as good (legitimate) or bad (associated with spyware). For this purpose, we have collected a set of 996 EULAs (out of which 9.6\% are associated with software that includes spyware). We will investigate different ways to represent these documents in order to be able to use supervised concept learning as an approach to solve the EULA classification problem.

7.1.4 Outline

The remainder of this paper is organized as follows: we begin by presenting the EULA classification problem in Section 7.2. Next, we describe the data collection process and the representation of data in Section 7.3. We then describe the experiments in Section 7.4. This is followed by a presentation of the results in Section 7.5. In Section 7.6, we analyze the results and outline the design of a spyware prevention tool based on EULA classification. Finally, we draw conclusions and give some pointers to future work in the last section.

7.2 EULA Classification

We want to investigate whether or not it is possible to classify software applications as legitimate (good) or associated with spyware (bad) based on their EULA. Intuitively, we can look upon this problem as a text classification (TC) task, for which there are two main approaches; knowledge engineering and machine learning.

Knowledge engineering systems have been known to often outperform machine learning systems on the TC task, although the gap in performance steadily shrinks (Feldman & Sanger, 2007). The main drawback of the former approach is the amount of skilled labor and expert knowledge required to generate and maintain the knowledge-encoding rules.

In contrast, the latter approach requires only a set of labeled (classified) training instances, which are less costly to produce. As a consequence, most of the recent work on categorization is concentrated on the machine learning approach and our study is no exception. We will now present the TC task more formally with regard to EULA classification.

7.2.1 The EULA Classification Task

Suppose that we have a collection, $I$, of EULAs, each labeled as either good or bad, depending on whether or not it is associated with legitimate software or spyware. The set of all possible classes can thus be defined as $C = \{\text{good, bad}\}$. We would like to approximate the unknown target function, $F : I \times C = \{1, 0\}$. The value of $f(i, c)$ is equal to one if the EULA, $i$, belongs to the class $c$ or zero otherwise.

It is now possible to define a classifier as an approximation function, $M : I \times C = \{1, 0\}$. The objective of the learning task is to generate a classifier that produces results as close to that of $F$ as possible.
7.2.2 Supervised Concept Learning

The machine learning approach to automatically build a classifier by learning the properties of the classes from a set of pre-classified training instances is known as supervised concept learning.

Most supervised concept learning algorithms cannot process text documents in their original form (Feldman & Sanger, 2007), hence they will not be able to process the EULAs. Consequently, we need to preprocess the EULAs by converting them to a manageable representation.

7.2.3 Representation

The choice of representation depends on what one regards as meaningful units of text and the meaningful natural language rules for combining these units, i.e., the problem of lexical and compositional semantics, respectively. The problem of compositional semantics is usually disregarded in TC (Sebastiani, 2002), however, exceptions exist, cf., Denoyer, Zaragoza, and Gallinari (2001). One approach that addresses the problem of lexical semantics is to represent each EULA as a feature vector. We now describe two quite different feature vector representations of EULAs.

The Bag-of-Words Model

The bag-of-words model is a common approach to represent documents as feature vectors (Feldman & Sanger, 2007). In fact, it has been found in several experiments that more sophisticated representations do not yield any significant effectiveness (Sebastiani, 2002). In this model, each word in a document is used as a feature. Thus, a complete document is represented by a vector with one feature for each word that occurs in the document. A collection of documents is then represented by a set of feature vectors and the dimension of the feature space is equal to the number of different words in all of the documents. Some studies have used phrases, rather than individual words, as terms. However, their experimental results have not been as encouraging as those of studies that use words as terms (Sebastiani, 2002).

There are basically two methods for associating weights to features. The simplest is the binary method, which either assigns a value of one if the word is present in the document or a value of zero otherwise. The binary method can be used if the chosen learning algorithm can only handle nominal attributes. However, what appears to be a more common method is to take the frequency of the word into account. We adopt this method and calculate frequencies using the standard Term Frequency - Inverse Document Frequency (TF IDF) scheme (Feldman & Sanger, 2007). The TF IDF function embodies the intuitions that: the more often a term occurs in a document, the more it is representative of its content, and the more documents a term occurs in, the less discriminating it is (Sebastiani, 2002). There exist several versions of TF IDF that differ from
each other in terms of logarithms, normalization or other factors.

We use Weka’s implemented version of TF IDF. Thus, given a word, \( w \), in a EULA, \( d \), the TF IDF weight is calculated as follows:

\[
\text{weight}(w, d) = \text{TermFreq}(w, d) \cdot \log \frac{N}{\text{DocFreq}(w)},
\]

where \( N \) is the total number of EULAs, DocFreq is the number of EULAs containing the word and TermFreq represents the frequency of the word in the particular EULA.

The Meta EULA Model

FaceTime Security Labs\(^3\) has created the EULA Analyzer\(^4\), which is a web-based tool, for assisting people in better understanding the EULAs of software downloadable on the Internet. This tool requires a user to copy the EULA from the pop-up window that appears when a software application is to be installed on a computer. The EULA is analyzed and the results are given as scores according to 10 different metrics. The user is then able to consider the metric scores to make a decision about whether or not to proceed with the software installation. However, EULA Analyzer does not provide a classification of the analyzed EULAs, instead this task is left for the user. Since EULA Analyzer is a proprietary service, its design and inner workings are not publicly available.

Since the objective of EULA Analyzer is to assist people in better understanding EULAs by supplying meta models of the EULAs, we hypothesize that this meta EULA model can be used as an alternative means for representing the EULAs as feature vectors. EULA Analyzer calculates the number of flagged, or suspicious, sentences. In addition, it calculates some basic statistics, such as: the number of characters, words, sentences and the average number of words per sentence. EULA Analyzer also performs more advanced calculations of readability metrics. For example, Flesch score and Gunning-Fog index use word and sentence lengths to indicate how difficult it is to comprehend a certain text, where low scores represent difficult texts (Flesch, 1948). The Flesch grade is calculated based on the same measures but instead gives the number of years of education that is needed to understand the text (Flesch, 1948). Automated readability and Coleman-Liau are two other readability tests that indicate the required level of education needed to comprehend a text (Coleman & Liau, 1975; Smith & Kincaid, 1970). Both of these tests differ from the earlier mentioned metrics by relying on a factor of characters per word instead of syllabus per word.

Our approach is to use the aforementioned EULA Analyzer metrics as attributes for a meta EULA data set. Thus, each EULA is described as a vector of 10 values and it has the same class label as the corresponding bag-of-words instance.

---

\(^{3}\)http://www.facetime.com
\(^{4}\)http://www.spywareguide.com/analyze
7.3 Data Sets

We now describe the data collection and the representation of the data. The collection, and labeling, of EULAs can be carried out in several ways. Our primary objective was to collect a large set of EULAs and at the same time allow for an automatic or semi-automatic labeling of EULA instances.

7.3.1 Data Collection

One of the contributions of this study is the collection of a large data set of labeled EULAs. We strived to collect approximately 1,000 EULAs (compared to 100 as featured in the aforementioned pilot study). The prerequisite was that each application should be easily downloaded from the Internet and present the user with a EULA that can be copied and pasted as ASCII text. We collected the good instances from Download.com\(^5\) and the bad instances from links obtained from SpywareGuide.com\(^6\). Based on the number of applications downloadable from both of these locations we hypothesized that a ratio of 1:10 would serve as a sufficient model of the real world, i.e., that spyware accounts for approximately 10% of all freely available applications. This is not too far from the figure reported by Moshchuk et al. (2006) (13.4%). Thus, we aimed to collect 900 good and 100 bad EULAs. The collection was systematically performed:

- for the good EULAs, we simply downloaded the first 900 applications,
- for the bad EULAs, we followed every obtained link that worked.

In both cases, we omitted applications that did not have a EULA. Additionally, we also omitted applications for which the EULA could not be extracted, e.g., the text could not be copied as ASCII text. Even though this only applies for 0.5% of the downloaded software we note that this (dys)function could possibly be used by the spyware vendors to prevent EULA analysis. For each software application, we collected the associated EULA together with basic information such as: software title, version, and vendor. The result is a collection featuring 900 good and 96 bad instances of real-world EULAs.

7.3.2 Data Representation

The raw data set, which consists of 996 text documents, must be converted to a manageable format. We perform the conversion into two different feature vector representations, as described in Section 7.2.2. We now describe how the two data sets are generated from the raw data set and highlight some characteristics for each of the generated sets of data.

\(^5\)http://www.download.com
\(^6\)http://www.spywareguide.com
Bag-of-words data set

The total number of words, contained in all of the EULAs, is close to 10,000 but experimental evidence suggests that this number can be reduced to around 10% without reducing classifier performance (Feldman & Sanger, 2007). In order to reduce the number of features we convert all characters to lowercase and consider only alphabetic tokens. Furthermore, we remove stop words and words that occur only once (hapax legomena), and store a maximum of 1,000 words per class. Finally, we apply Weka’s iterated Lovins stemmer to be able to store only the stems of the remaining words. The result is a data set with 1,269 numeric features and a nominal target feature that indicates if the EULA is bad or good.

Meta EULA data set

The processing of EULAs through EULA Analyzer is carried out by custom-made Unix shell scripts and Apple Automator programs. This setup allows us to automatically fetch the text from each of the 996 EULAs, paste it into the text field on the EULA Analyzer website, send the text for analysis, and finally retrieve the scores of the 10 metrics that represent the results of the analysis. Using the results according to these metrics, for each EULA, we generate a meta EULA data set and convert it to the Weka ARFF format. Table 7.1 shows the minimum, maximum, and mean value for each EULA Analyzer metric. As can be seen in the table, the Flesch score can be negative. Unfortunately, some algorithms cannot handle negative numeric attributes. For example, Naive Bayes Multinomial, which outperformed all other algorithms in the pilot study of EULA classification, is one such algorithm. Consequently, we opt to remove the Flesch score attribute from the meta EULA data set to resolve this issue. For our data set of 996 instances, the number of flagged sentences attribute values range from 0 to 84 but most EULAs are assigned a value close to zero. In analyzing the distribution of good and bad EULAs across the range of the flagged sentences attribute, we observe that this attribute can achieve a linear separation between a cluster of 25 bad instances and the rest of the EULAs. Approximately 30 additional bad instances can be isolated together with a small number of good instances. In total, this amounts to a separation of approximately 55 bad instances from the rest of the instances. In this case, there are still 41 bad instances classified as good. In analyzing the distributions for the remaining attributes, we do not observe any trivial ways to separate good and bad instances.

7.4 Experiments

In this section, we discuss the choice of algorithms and algorithm configurations. In addition, we discuss the metrics we use to evaluate classifier performance and present our experimental approach.
Table 7.1: *EULA Analyzer results for the complete data set*

<table>
<thead>
<tr>
<th>Metric</th>
<th>Min</th>
<th>Mean (Std.Dev)</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flagged Sentences</td>
<td>0</td>
<td>2.14(5.89)</td>
<td>84</td>
</tr>
<tr>
<td>Number of characters</td>
<td>72</td>
<td>7308(6877)</td>
<td>70566</td>
</tr>
<tr>
<td>Number of words</td>
<td>15</td>
<td>1262(1161)</td>
<td>11649</td>
</tr>
<tr>
<td>Number of sentences</td>
<td>1</td>
<td>42.53(34.95)</td>
<td>376</td>
</tr>
<tr>
<td>Average words per sentence</td>
<td>3.75</td>
<td>29.00(8.83)</td>
<td>79.35</td>
</tr>
<tr>
<td>Flesch score</td>
<td>-36.87</td>
<td>24.56(12)</td>
<td>95.87</td>
</tr>
<tr>
<td>Flesch grade</td>
<td>1</td>
<td>17.05(3.54)</td>
<td>37</td>
</tr>
<tr>
<td>Automated readability index</td>
<td>3</td>
<td>19.97(4.68)</td>
<td>47</td>
</tr>
<tr>
<td>Coleman-Liau index</td>
<td>6</td>
<td>19.71(3.20)</td>
<td>29</td>
</tr>
<tr>
<td>Gunning-Fog index</td>
<td>12</td>
<td>43.95(8.77)</td>
<td>93</td>
</tr>
</tbody>
</table>

### 7.4.1 Algorithm Selection and Configuration

We want to determine if it is feasible to apply supervised concept learners to solve the EULA classification problem. In order to investigate the usefulness of different learning techniques we include a diverse population of 17 algorithms from different paradigms, for example: perceptron and kernel functions, instance-based learners, Bayesian learners, decision tree inducers, meta-learners, rule inducers, and so forth. We use original Weka 3.5.7 algorithm implementations and, for most algorithms, we use the default configuration.

The main emphasis in this study is not to find the optimal configuration for each algorithm, that is, the configuration for which the algorithm generates the best performing classifiers. Such an objective would most certainly require extensive parameter tuning. Instead, this study focuses on investigating if it is possible, in general, to distinguish between good and bad EULAs. Consequently, we do not perform any systematic parameter tuning at all.

We alter the configuration for the following algorithms: the number of neighbors ($k$) for IBk is changed in order to distinguish the algorithm from the 1-nearest neighbor (IB1) algorithm. We use $k = 10$ based on standard practice but point out that it is arbitrary. The $k$ parameter can be optimized for a particular problem by systematically performing repeated training and testing runs with different $k$ values. We use SMO as a meta classifier for the Stacking algorithm and include the following algorithms in the ensemble: Naive Bayes Multinomial, SMO, and VotedPerceptron. These three algorithms are selected based on their high performance, as reported in the pilot study. The specification of the key algorithm configurations is provided in Table 7.2. The Weka default configurations (Witten & Frank, 2005) are provided for the sake of reproducibility.
hence they will not be further described.

### 7.4.2 Evaluation of Classifier Performance

#### Classification Accuracy

We need to select relevant evaluation metrics in order to measure classifier performance. Traditionally, the accuracy metric (the number of correct classifications divided by the total number of classifications) has been widely used. However, several issues have been raised against the use of this particular metric as the only means to measure performance (Provost et al., 1998). However, when used in conjunction with other metrics, we believe that accuracy is still a useful metric to consider.

#### The Area Under the ROC Curve

In addition to accuracy, we therefore need to select metrics that are suitable for our particular application. For the purpose of EULA classification, we argue that the cost of misclassification is different for the two classes (good and bad). For example, classifying a bad EULA as good is sometimes far worse than the opposite and this is particularly true if the classification should be the basis for a decision support system that should aid the user in making the decision to install an application or to abort the installation.

If a good EULA is classified as bad, the user might think twice before installing the associated application and instead will try to find alternative applications. If, on the other hand, a bad EULA is classified as good, the user might install an application that contains spyware, believing that the application is legitimate. This is actually worse than not using any EULA classification at all, since the user is under the (false) impression of being informed about the legitimacy of the application.

In order to address this issue, we consider four important metrics; true/false positives and true/false negatives, as documented in Table 7.3. In particular, we will use the Area under the ROC curve (AUC), which is based on the true positives rate (TPR) and the false positives rate (FPR) since it does not depend on equal class distribution or misclassification costs (Witten & Frank, 2005). The calculation of, and motivation for, AUC is described in detail by Fawcett (2003).

#### Multi-criteria Evaluation

In addition to the ACC and AUC metrics we will evaluate each algorithm using the Candidate Evaluation Function (CEF) (Lavesson & Davidsson, 2008). The purpose of CEF is to capture application-specific trade-offs by combining multiple relevant metrics. We shall now demonstrate how multi-criteria metrics can be used as an approach to trade-off some of the important aspects of EULA classification. CEF normalizes the metrics in order to get a uniform output domain and it is also possible to specify explicit
Table 7.2: Learning algorithm configurations

<table>
<thead>
<tr>
<th>Weka Algorithm</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaiveBayes</td>
<td>Kernel estimator: false, Supervised discretization: false</td>
</tr>
<tr>
<td>NaiveBayesMultinomial</td>
<td>N/A</td>
</tr>
<tr>
<td>RBFNetwork&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Ridge: 1.0E-8</td>
</tr>
<tr>
<td>SMO&lt;sup&gt;b&lt;/sup&gt;</td>
<td>Kernel: Polynomial, Complexity: 1.0</td>
</tr>
<tr>
<td>VotedPerceptron</td>
<td>Exponent: 1.0, Max kernel alterations: 10,000</td>
</tr>
<tr>
<td>IBk&lt;sup&gt;c&lt;/sup&gt;</td>
<td>Number of neighbors: 10, Distance weighting: false</td>
</tr>
<tr>
<td>KStar</td>
<td>Missing values treatment: average column entropy curves</td>
</tr>
<tr>
<td>AdaBoostM1</td>
<td>Classifier: DecisionStump</td>
</tr>
<tr>
<td>Bagging</td>
<td>Classifier: REPTree (Pruning: true)</td>
</tr>
<tr>
<td>Stacking</td>
<td>Meta: SMO, Committee: SMO, VotedPerceptron, NaiveBayesMultinomial</td>
</tr>
<tr>
<td>HyperPipes</td>
<td>N/A</td>
</tr>
<tr>
<td>JRip&lt;sup&gt;d&lt;/sup&gt;</td>
<td>Pruning: true, Number of optimizations: 2</td>
</tr>
<tr>
<td>PART</td>
<td>Binary splits: false, Pruning: true (confidence factor: 0.25)</td>
</tr>
<tr>
<td>Ridor</td>
<td>N/A</td>
</tr>
<tr>
<td>DecisionStump</td>
<td>N/A</td>
</tr>
<tr>
<td>J48&lt;sup&gt;e&lt;/sup&gt;</td>
<td>Pruning: Subtree raising, Pruning confidence factor: 0.25</td>
</tr>
<tr>
<td>RandomForest</td>
<td>Number of trees: 10</td>
</tr>
</tbody>
</table>

<sup>a</sup>Radial Basis Function Network  
<sup>b</sup>Support Vector Machines  
<sup>c</sup>K-nearest Neighbor  
<sup>d</sup>Ripper  
<sup>e</sup>C4.5
Table 7.3: Evaluation metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True positives</td>
<td>TP</td>
<td>Good EULAs classified as good.</td>
</tr>
<tr>
<td>False positives</td>
<td>FP</td>
<td>Bad EULAs classified as good.</td>
</tr>
<tr>
<td>True negatives</td>
<td>TN</td>
<td>Bad EULAs classified as bad.</td>
</tr>
<tr>
<td>False negatives</td>
<td>FN</td>
<td>Good EULAs classified as bad.</td>
</tr>
<tr>
<td>True positives rate</td>
<td>TPR</td>
<td>$TPR = \frac{TP}{TP+FP}$</td>
</tr>
<tr>
<td>False positives rate</td>
<td>FPR</td>
<td>$FPR = \frac{FP}{TP+FP}$</td>
</tr>
</tbody>
</table>

weights for each metric to ensure that application-specific trade-offs can be properly represented. CEF itself does not dictate which metrics to use, it merely dictates how metrics are combined. Finally, CEF makes it possible to specify the acceptable range for each metric, pertaining to a particular application.

We define $m_j$ as a metric with index $j$ from an index set, $J$, over the selected set of metrics. Each metric is associated with a weight, $w_j$, and an acceptable range, $r = [b_j^l, b_j^u]$. The lower bound, $b_j^l$, denotes the least desired acceptable score. Similarly, the upper bound, $b_j^u$, denotes the desired score. Note that, in the original CEF definition a metric was normalized according to the best and worst score of that particular metric obtained from the studied set of classifiers. The current normalization uses the lower and upper bound to generate a smooth distribution from 0 (least desired) to 1. CEF is now defined as specified in Equation 7.2.

$$CEF(c, D) = \begin{cases} 0 : & \exists j (\overline{m}_j(c, D) < 0) \\ \sum_{j \in J} w_j \overline{m}_j(c, D) & \text{otherwise} \end{cases}$$

where $\sum_{j \in J} w_j = 1$ and

$$\overline{m}_j(c, D) = \begin{cases} 1 : & \frac{m_j - b_j^l}{b_j^u - b_j^l} > 1 \\ \frac{m_j - b_j^l}{b_j^u - b_j^l} & \text{otherwise} \end{cases}.$$  

To address the fact that a low FPR is more important than a high TPR for our particular application, we define an example CEF metric that combines $m_1 =$TPR with $r_1 = [0.9, 1.0]$ and $m_2 =$FPR with $r_2 = [0.4, 0.0]$. Furthermore, we let $w_1 = 0.2$ and $w_2 = 0.8$ to make $m_2$ four times more important than $m_1$. The weights and ranges are provided as examples to illustrate how CEF can be used to customize the evaluation for our application. However, these properties should be selected by domain experts after careful consideration, preferably using some reliable systematic approach.

110
**Baseline Classifier**

We have reviewed two possible data representations that can be used when applying supervised learning to EULA classification. Besides the comparison between these two representations, we need to quantify the utility of the EULA classification approach compared to the behavior of an average user. In addition, we need to find out if it is even possible to discriminate between bad and good EULAs using the machine learning approach.

Consequently, we argue that it is necessary to compare the performance of the classifiers (generated from both data sets) to that of a baseline classifier. We choose to include Weka’s ZeroR classifier for this purpose. ZeroR classifies instances based on the majority of the class distribution. For our particular case, this means that ZeroR will classify all instances as good. Our model of an average user assumes that the user does not read the EULA but instead continues with the installation. Thus, the behavior of this model is equivalent to that of ZeroR. We use AUC for all statistical comparisons. There are several rationales for this decision:

- the class distribution is skewed, which suggests that AUC is more appropriate than accuracy (Provost et al., 1998),
- the weights and ranges of the CEF metric are provided as examples and thus there is little sense in using the CEF results for statistical tests of significance, and
- no learned classifier should have an AUC less than 0.5 (Fawcett, 2001) and ZeroR always achieves an AUC score of 0.5, which makes ZeroR a simple and meaningful baseline if AUC is used.

ZeroR, which has no predictive power, can thus be used as a baseline to determine if a particular classifier has predictive power, i.e., if it can discriminate between good and bad EULAs. If, for a certain data set, at least one classifier is shown to have predictive power we can also assume more generally that it is possible to discriminate between bad and good EULAs using that data set.

### 7.4.3 Experimental Procedure

Since we have a limited amount of data for training and testing (996 instances), we choose to estimate performance using the average of ten stratified 10-fold cross-validation tests. We use Weka for all experiments and train the 17 learning algorithms on our two data sets.

Moreover, we let $X$ represent the set of performance results obtained from the bag-of-words set. Similarly, $Y$ represents the set of performance results obtained from the meta EULA set. We formulate the following null hypotheses:

- $h_{0}^{x}: x = 0.5$
h_0^y: y = 0.5

h_0^y is tested for each x ∈ X and h_0^y is tested for each y ∈ Y. Both of the main hypotheses are tested using the corrected resampled t-test as recommended by Witten and Frank (2005) for comparisons of two algorithms on the data set and using ten 10-fold cross-validation tests. Thus, we use 99 degrees of freedom and two-sided probability values. We are aware of the issues related to multiple hypothesis testing, cf., Demzar (2006). Thus, since we test 17 hypotheses for each data set, the family-wise error can be elevated. The Bonferroni correction can be applied to maintain the family-wise error but, as Demzar notes, it is overly radical. We will therefore present results using both the regular probability value (p < 0.05) and the Bonferroni corrected value (p < 0.001), which is calculated by dividing the regular p-value by the number of tested hypothesis for each data set.

The third null hypothesis, h_0^z, is that there is no difference in performance between the bag-of-words and the meta EULA generated classifiers. This null hypothesis is tested using the two-sided Wilcoxon signed rank test. In this test, we compare the AUC result for each classifier on the bag-of-words data set to that of the corresponding classifier on the meta EULA data set. We reject h_0^z if the results for one data set are significantly better than those obtained from the other. The rationale for using a non-parametric test instead of, e.g., the paired t-test is that the latter assumes normality, which might be violated in our test (Demzar, 2006). In testing this hypothesis we are aware that the results can only apply for the selected algorithms and configurations.

7.5 Results

We first present the performance results for each data representation approach separately and then review the tested hypotheses. This is followed by an analysis and discussion about both approaches.

7.5.1 Bag-of-words Results

The evaluation results for the bag-of-words data set are presented in Table 7.4. The table features evaluation scores for the 17 learning algorithms, the average learning algorithm score, and score of the baseline classifier. Each algorithm evaluation result is shown using four metrics: classification accuracy (ACC), true positives rate (TPR), false positives rate (FPR), and the Area Under the ROC Curve (AUC).

SMO, Voted Perceptron, and Stacking yield the highest ACC. However, the difference between the best and worst performing supervised algorithm is rather small (.076). The accuracy of Naive Bayes Multinomial is mediocre and the regular Naive Bayes algorithm actually performs worse than the baseline.
With regard to AUC, the performance of the top ACC algorithms is mediocre while Naive Bayes Multinomial is the best performing algorithm. The difference in AUC between the best and worst performing algorithm is quite large (0.350). This difference is much due to the low performance of the KStar algorithm.

The TPR (the rate of correctly classified bad EULAs) is quite high for all algorithms, except the regular Naive Bayes algorithm. Unfortunately the FPR (the rate of bad EULAs classified as good) is also relatively high, even for the best performing algorithms. Among the worst performing algorithms (according to FPR) are the two instance-based learners: KStar and IBk.

### 7.5.2 Meta EULA Results

The evaluation results for the meta EULA data set are presented in Table 7.5 using the same setup as the Table 7.4.

The best performing algorithms are tree and rule learners, as well as meta learners that use tree learners in their ensemble. For ACC, the best performing algorithms are: Bagging, JRip, PART, and J48. The difference between the best and worst performing algorithm, according to ACC, is large (.128). The accuracy of Voted Perceptron is identical to that of ZeroR. Meanwhile, Naive Bayes, Naive Bayes Multinomial, and KStar all perform worse than ZeroR.

In terms of AUC, the top ACC algorithm (Bagging) is the best performing algorithm. The difference in AUC between the best and worst performing algorithm is very large (0.369).

The worst performing algorithms, according to AUC, are: Stacking, SMO, and Voted Perceptron. However, these three algorithms all share perfect TPR scores. The low AUC score is due to their high FPR scores. Similarly to the bag-of-words data set, the lowest FPR scores are achieved by the Bayesian algorithms.

### 7.5.3 Tested Hypotheses

The results from the two first main hypotheses are indicated in Table 7.4 and Table 7.5, respectively. At $p < 0.05$, 15 out 17 algorithms perform significantly better than the baseline on the bag-of-words set. For the meta EULA set, 14 out 17 algorithms perform significantly better than the baseline for the same confidence level. More generally, we can therefore conclude that both data sets can be used for training classifiers that can discriminate between good and bad EULAs.

The two-sided p-value of obtaining our results if $h_0$ holds is .01675, which means that we can reject this null hypothesis. We therefore conclude that the bag-of-words model is statistically significantly better than the meta EULA model, at least for the included algorithms.
<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>ACC</th>
<th>TPR</th>
<th>FPR</th>
<th>AUC</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaiveBayes</td>
<td>0.874</td>
<td>0.881</td>
<td>0.193</td>
<td>0.873</td>
<td>4.204</td>
</tr>
<tr>
<td>NaiveBayesMultinomial</td>
<td>0.939</td>
<td>0.951</td>
<td>0.173</td>
<td>0.926</td>
<td>5.040</td>
</tr>
<tr>
<td>RBFNetwork</td>
<td>0.928</td>
<td>0.950</td>
<td>0.276</td>
<td>0.798</td>
<td>2.708</td>
</tr>
<tr>
<td>SMO</td>
<td>0.950</td>
<td>0.980</td>
<td>0.335</td>
<td>0.822</td>
<td>3.192</td>
</tr>
<tr>
<td>VotedPerceptron</td>
<td>0.949</td>
<td>0.979</td>
<td>0.334</td>
<td>0.827</td>
<td>3.408</td>
</tr>
<tr>
<td>IBk</td>
<td>0.923</td>
<td>0.997</td>
<td>0.768</td>
<td>0.847</td>
<td>3.570</td>
</tr>
<tr>
<td>KStar</td>
<td>0.911</td>
<td>1.000</td>
<td>0.927</td>
<td>0.576</td>
<td>1.053</td>
</tr>
<tr>
<td>AdaBoostM1</td>
<td>0.942</td>
<td>0.975</td>
<td>0.369</td>
<td>0.875</td>
<td>3.909</td>
</tr>
<tr>
<td>Bagging</td>
<td>0.942</td>
<td>0.980</td>
<td>0.416</td>
<td>0.887</td>
<td>4.233</td>
</tr>
<tr>
<td>Stacking</td>
<td>0.948</td>
<td>0.978</td>
<td>0.334</td>
<td>0.822</td>
<td>3.192</td>
</tr>
<tr>
<td>HyperPipes</td>
<td>0.947</td>
<td>0.989</td>
<td>0.446</td>
<td>0.827</td>
<td>3.065</td>
</tr>
<tr>
<td>JRip</td>
<td>0.932</td>
<td>0.964</td>
<td>0.368</td>
<td>0.799</td>
<td>2.982</td>
</tr>
<tr>
<td>PART</td>
<td>0.928</td>
<td>0.963</td>
<td>0.401</td>
<td>0.800</td>
<td>2.726</td>
</tr>
<tr>
<td>Ridor</td>
<td>0.931</td>
<td>0.981</td>
<td>0.537</td>
<td>0.722</td>
<td>2.092</td>
</tr>
<tr>
<td>DecisionStump</td>
<td>0.941</td>
<td>0.971</td>
<td>0.343</td>
<td>0.814</td>
<td>3.435</td>
</tr>
<tr>
<td>J48</td>
<td>0.928</td>
<td>0.968</td>
<td>0.443</td>
<td>0.729</td>
<td>1.846</td>
</tr>
<tr>
<td>RandomForest</td>
<td>0.941</td>
<td>0.992</td>
<td>0.542</td>
<td>0.881</td>
<td>4.198</td>
</tr>
<tr>
<td>ZeroR</td>
<td>0.904</td>
<td>1.000</td>
<td>1.000</td>
<td>0.500</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Table 7.4: Results on the bag-of-words data set
Table 7.5: Results on the meta EULA data set

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Type</th>
<th>ACC</th>
<th>TPR</th>
<th>FPR</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaiveBayes</td>
<td>Bayes</td>
<td>0.892 (0.024)</td>
<td>0.926 (0.025)</td>
<td>0.423 (0.131)</td>
<td>0.851 (0.059)</td>
</tr>
<tr>
<td>NaiveBayesMultinomial</td>
<td>Bayes</td>
<td>0.812 (0.035)</td>
<td>0.828 (0.039)</td>
<td>0.343 (0.131)</td>
<td>0.797 (0.070)</td>
</tr>
<tr>
<td>RBFNetwork</td>
<td>Function</td>
<td>0.919 (0.014)</td>
<td>0.987 (0.013)</td>
<td>0.711 (0.128)</td>
<td>0.846 (0.059)</td>
</tr>
<tr>
<td>SMO</td>
<td>Function</td>
<td>0.921 (0.010)</td>
<td>1.000 (0.000)</td>
<td>0.821 (0.101)</td>
<td>0.590 (0.051)</td>
</tr>
<tr>
<td>VotedPerceptron</td>
<td>Function</td>
<td>0.904 (0.004)</td>
<td>1.000 (0.000)</td>
<td>1.000 (0.000)</td>
<td>0.500 (0.000)</td>
</tr>
<tr>
<td>IBk</td>
<td>Lazy</td>
<td>0.929 (0.016)</td>
<td>0.995 (0.007)</td>
<td>0.686 (0.134)</td>
<td>0.825 (0.071)</td>
</tr>
<tr>
<td>KStar</td>
<td>Lazy</td>
<td>0.900 (0.024)</td>
<td>0.960 (0.021)</td>
<td>0.662 (0.137)</td>
<td>0.732 (0.083)</td>
</tr>
<tr>
<td>AdaBoostM1</td>
<td>Meta</td>
<td>0.934 (0.017)</td>
<td>0.978 (0.018)</td>
<td>0.479 (0.136)</td>
<td>0.809 (0.073)</td>
</tr>
<tr>
<td>Bagging</td>
<td>Meta</td>
<td>0.939 (0.017)</td>
<td>0.984 (0.013)</td>
<td>0.484 (0.139)</td>
<td>0.869 (0.062)</td>
</tr>
<tr>
<td>Stacking</td>
<td>Meta</td>
<td>0.921 (0.010)</td>
<td>1.000 (0.000)</td>
<td>0.821 (0.101)</td>
<td>0.590 (0.051)</td>
</tr>
<tr>
<td>HyperPipes</td>
<td>Misc</td>
<td>0.923 (0.012)</td>
<td>0.997 (0.006)</td>
<td>0.769 (0.110)</td>
<td>0.684 (0.053)</td>
</tr>
<tr>
<td>JRip</td>
<td>Rules</td>
<td>0.938 (0.016)</td>
<td>0.984 (0.014)</td>
<td>0.490 (0.140)</td>
<td>0.747 (0.068)</td>
</tr>
<tr>
<td>PART</td>
<td>Rules</td>
<td>0.937 (0.017)</td>
<td>0.984 (0.013)</td>
<td>0.498 (0.141)</td>
<td>0.820 (0.076)</td>
</tr>
<tr>
<td>Ridor</td>
<td>Rules</td>
<td>0.932 (0.016)</td>
<td>0.984 (0.013)</td>
<td>0.555 (0.163)</td>
<td>0.715 (0.079)</td>
</tr>
<tr>
<td>DecisionStump</td>
<td>Trees</td>
<td>0.934 (0.017)</td>
<td>0.978 (0.018)</td>
<td>0.479 (0.136)</td>
<td>0.749 (0.065)</td>
</tr>
<tr>
<td>J48</td>
<td>Trees</td>
<td>0.936 (0.016)</td>
<td>0.982 (0.014)</td>
<td>0.495 (0.138)</td>
<td>0.742 (0.076)</td>
</tr>
<tr>
<td>RandomForest</td>
<td>Trees</td>
<td>0.929 (0.019)</td>
<td>0.977 (0.017)</td>
<td>0.524 (0.128)</td>
<td>0.807 (0.074)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>0.918 (0.017)</td>
<td>0.973 (0.014)</td>
<td>0.602 (0.123)</td>
<td>0.745 (0.063)</td>
</tr>
<tr>
<td>ZeroR</td>
<td></td>
<td>0.904 (0.004)</td>
<td>1.000 (0.000)</td>
<td>1.000 (0.000)</td>
<td>0.500 (0.000)</td>
</tr>
</tbody>
</table>

*p < 0.05, two-tailed

**p < 0.05, two-tailed, Bonferroni corrected
Table 7.6: CEF Evaluation Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>bag-of-words</th>
<th></th>
<th>meta EULA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m_1$</td>
<td>$m_2$</td>
<td>CEF</td>
<td>$m_1$</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>0.000</td>
<td>0.518</td>
<td>0.000</td>
<td>0.260</td>
</tr>
<tr>
<td>NaiveBayesMultinomial</td>
<td>0.510</td>
<td>0.568</td>
<td>0.556</td>
<td>0.000</td>
</tr>
<tr>
<td>RBFNetwork</td>
<td>0.500</td>
<td>0.310</td>
<td>0.348</td>
<td>0.870</td>
</tr>
<tr>
<td>SMO</td>
<td>0.800</td>
<td>0.163</td>
<td>0.290</td>
<td>1.000</td>
</tr>
<tr>
<td>VotedPerceptron</td>
<td>0.790</td>
<td>0.165</td>
<td>0.290</td>
<td>1.000</td>
</tr>
<tr>
<td>IBk</td>
<td>0.970</td>
<td>0.000</td>
<td>0.000</td>
<td>0.950</td>
</tr>
<tr>
<td>KStar</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.600</td>
</tr>
<tr>
<td>AdaBoostM1</td>
<td>0.750</td>
<td>0.078</td>
<td>0.212</td>
<td>0.780</td>
</tr>
<tr>
<td>Bagging</td>
<td>0.800</td>
<td>0.000</td>
<td>0.000</td>
<td>0.840</td>
</tr>
<tr>
<td>Stacking</td>
<td>0.780</td>
<td>0.165</td>
<td>0.288</td>
<td>1.000</td>
</tr>
<tr>
<td>HyperPipes</td>
<td>0.890</td>
<td>0.000</td>
<td>0.000</td>
<td>0.970</td>
</tr>
<tr>
<td>JRip</td>
<td>0.640</td>
<td>0.080</td>
<td>0.192</td>
<td>0.840</td>
</tr>
<tr>
<td>PART</td>
<td>0.630</td>
<td>0.000</td>
<td>0.000</td>
<td>0.840</td>
</tr>
<tr>
<td>Ridor</td>
<td>0.810</td>
<td>0.000</td>
<td>0.000</td>
<td>0.840</td>
</tr>
<tr>
<td>DecisionStump</td>
<td>0.710</td>
<td>0.143</td>
<td>0.256</td>
<td>0.780</td>
</tr>
<tr>
<td>J48</td>
<td>0.680</td>
<td>0.000</td>
<td>0.000</td>
<td>0.820</td>
</tr>
<tr>
<td>RandomForest</td>
<td>0.920</td>
<td>0.000</td>
<td>0.000</td>
<td>0.770</td>
</tr>
</tbody>
</table>

7.5.4 CEF Results

We now review the results from the CEF evaluation. Table 7.6 shows the CEF score for each algorithm and data set. The scores for the TPR metric ($m_1$) and the FPR metric ($m_2$) have been calculated using the TPR and FPR scores from Table 7.4 and Table 7.5, respectively. As can be observed, several algorithms fail completely, since they do achieve an acceptable score for at least one of the metrics. Eight algorithms manage to achieve a valid CEF score on the bag-of-words data set. However, most decision tree and rule learners fail since they cannot achieve an acceptable FPR. The Bayesian learners are exceptionally strong with regard to FPR but the Naive Bayes algorithm fails to achieve an acceptable TPR. For the meta EULA data set, all algorithms fail. The only algorithm that achieves a good enough FPR is NaiveBayes Multinomial, however it fails because of the unacceptable TPR score.
7.6 Discussion

The experiment raises several interesting issues which will now be discussed. We first bring forth some technical aspects related to the featured algorithms and their performance on EULA classification.

The best performing algorithms for the bag-of-words data set, according to ACC, are predominantly kernel function based. SMO, especially, is known to perform well according to ACC on large text classification data sets (Kibriya, Frank, Pfahringer, & Holmes, 2004). However, in terms of AUC, both Voted Perceptron and SMO are outperformed by Naive Bayes Multinomial and Bagging.

Caruana and Niculescu-Mizil (2006) note that support vector machines are not designed to predict probabilities and explain that their predictions can be calibrated with, e.g.: Platt Scaling or Isotonic Regression. We perform no such calibration and thus this could be related to the poor SMO performance according to AUC.

Several studies have reported on the poor performance of Naive Bayes Multinomial compared to that of, e.g., SMO on text classification tasks. However, Kibriya et al. (2004) argued that the performance, especially in terms of AUC, of Naive Bayes Multinomial can be dramatically improved by applying TF-IDF during preprocessing. Our results on the bag-of-words data set indicate that this claim is valid. Consequently, the obtained results seem to be aligned with related studies of these algorithms for similar domains.

The case is different for the meta EULA data set, which has a small number of attributes compared to the bag-of-words set. Few algorithms seem to perform well on this data set with a notable exception in Bagging.

Bagging seems to be quite suitable for our problem. It is the superior algorithm for both ACC and AUC on the meta EULA data set. In addition, it is the second best algorithm, according to AUC, on the bag-of-words data set and it is positioned in the upper echelon of algorithms for the ACC metric on this data set as well. However, compared to the Bayesian learners it performs poorly with respect to FPR, which is an important metric for this application.

The default configuration of Bagging uses REP trees as base classifiers. In analyzing the REP trees generated from the complete meta EULA set we observe that they seem to be very diverse in structure but most are rather complex in terms of tree size (most of them use more than 30 nodes). The aggregated Bagging classifier can thus be described as a quite complex model. This seems to be aligned with the essential property of Bagging as Breiman (1996) concludes: if perturbing the learning set can cause significant changes in the constructed classifier, then Bagging can improve performance. Consequently, the REP trees learned from the different bootstrap generated sub sets of data are diverse and this may contribute to the overall good performance.

As stated earlier, most algorithms perform poorly on the meta EULA set and this is especially true for Voted Perceptron and SMO. The former performs identically to that
of ZeroR and SMO is only slightly better. Moreover, all algorithms perform poorly according to FPR on the meta EULA set, which suggests that it is difficult to find structural patterns in data compiled from the EULA Analyzer metrics.

It is not always trivial to determine what triggers a certain classification, mainly due to the opaque nature of most of the well-performing classifiers. However, the rule based classifiers share a substantial number of words used for classification, as can be seen in Table 7.7.

Table 7.8 shows the corresponding rules generated from the meta EULA set. The classifier representations for the latter data set are more simple than those obtained from the bag-of-words data set, mainly for two reasons: (i) the bag-of-words data set is more complex in terms of the number of attributes and (ii) the information gain of the meta EULA attributes seem to be low in general except for flagged sentences. The consequence of the second point is that most tree and rule learners build very simple classifiers, based solely on the flagged sentence attribute.

We analyze the rules and trees generated from the bag-of-words data set in terms of which word stems are associated with a classification of EULAs as bad. Not surprisingly, some frequently used stems are: search, upgrad, web, uninst, opportun, and adver. These words stems can easily be associated with typical concepts that would be mentioned by spyware vendors. In summary, the bag-of-words model seems to be more suitable as a basis for EULA classification. However, even for this data set, many supervised learners achieve FPR scores too high to be acceptable in a real-world decision support system. We hypothesize that a larger collection of bad instances along with careful parameter tuning of the algorithms might contribute to decrease this rate.

7.6.1 A Novel Tool for Spyware Prevention

Assuming that the EULA classification approach can make distinctions between good and bad software, we would like to use the approach to help users make informed decisions about the software they download. We therefore suggest that the EULA classification method outlined in this paper can be incorporated in a spyware prevention tool.

This tool could presumably be designed as a middleware that operates between the operating system and the application installer software. Furthermore, we suggest that it could be executed as a background process that identifies and analyzes EULAs as they appear on the screen during an installation. To accomplish this automatic identification, the tool can plant a hook into the operating system function for displaying dialogues.

Based on the classification of the EULA, the tool can provide the user with a recommendation about whether to install the application or not. This allows the tool to assist users in making informed decisions about the installation of software without forcing them to read (and understand) the lengthy and intricate EULAs. For example, if the tool classifies the EULA as bad, the user can take appropriate actions against it, e.g., by disagreeing with the EULA and exiting the installation process.
Table 7.7: Rule-based classifiers generated using the complete bag-of-words data set

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRip</td>
<td>(search $\geq 1.67$) AND (upgrad $\geq 1.01$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>(web $\geq 2.10$) AND (uninst $\geq 1.42$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>(opportun $\geq 1.89$) AND (adver $\geq 1.34$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>ELSE good</td>
</tr>
<tr>
<td>PART</td>
<td>(visit $\leq 1.48$) AND (search $\leq 0$) AND (opportun $\leq 0$) AND ad $\leq 1.19$) AND (graph $\leq 0$) AND (definit $\leq 0$) AND (view $\leq 0$) THEN good</td>
</tr>
<tr>
<td></td>
<td>(contextu $&gt; 0$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>(redirect $\leq 0$) AND (panel $&gt; 0$) AND (fraud $\leq 0$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>(redirect $\leq 0$) AND (cook $\leq 0$) AND (vi $\leq 1.50$) AND (timel $\leq 0$) AND (patch $\leq 0$) AND (pc $\leq 0$) AND (tot $\leq 0$) THEN good</td>
</tr>
<tr>
<td></td>
<td>(redirect $\leq 0$) AND (zip $\leq 0$) AND (accru $\leq 0$) AND (explicit $\leq 1.61$) AND (adver $\leq 3.10$) AND (detect $\leq 0$) AND (asser $\leq 0$) AND (injunct $\leq 0$) AND (encour $\leq 0$) AND (born $\leq 0$) THEN good</td>
</tr>
<tr>
<td></td>
<td>(charg $\leq 0.99$) AND (rever $\leq 0.21$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>(tort $\leq 0$) AND (pop $\leq 0$) THEN good</td>
</tr>
<tr>
<td></td>
<td>ELSE bad</td>
</tr>
<tr>
<td>Ridor</td>
<td>(search $&gt; 0.84$) AND (upgrad $&gt; 0.50$) AND (vulgar $\leq 1.24$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>(inform $&gt; 0.93$) AND (download $&gt; 1.88$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>(pol $&gt; 0.49$) AND (visit $&gt; 1.91$) AND (whichever $\leq 0.93$) THEN bad</td>
</tr>
<tr>
<td></td>
<td>ELSE good</td>
</tr>
</tbody>
</table>
Table 7.8: Rule-based classifiers generated using the complete meta EULA data set

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRip</td>
<td>(\text{flagged_sentences} \geq 10) \text{ THEN bad}</td>
</tr>
<tr>
<td></td>
<td>ELSE good</td>
</tr>
<tr>
<td>PART</td>
<td>(\text{flagged_sentences} \leq 9) \text{ AND } (\text{flagged_sentences} \leq 2) \text{ AND } (\text{automated_readability_index} \leq 20) \text{ THEN good}</td>
</tr>
<tr>
<td></td>
<td>(\text{flagged_sentences} \leq 9) \text{ THEN good}</td>
</tr>
<tr>
<td></td>
<td>ELSE bad</td>
</tr>
<tr>
<td>Ridor</td>
<td>(\text{flagged_sentences} &gt; 9.5) \text{ THEN bad}</td>
</tr>
<tr>
<td></td>
<td>ELSE good</td>
</tr>
</tbody>
</table>

It should be noted that any tool based on our method should not be used in isolation, but rather as a complement to other approaches, e.g., anti-spyware software. In addition, tree and rule learners generate classifiers that can be used for visualizing the decision process. Despite the fact that this category of learners does not seem to be the most appropriate at solving the classification task, their visualizations could increase the number of correct classifications since they may allow the user to make a decision based on more information than what is provided by the opaque learners.

We outline basic requirements for the imagined prevention tool as follows: first, we need to make sure that the tool is accurate in its classifications since this is the main functionality. The tool should essentially be able to detect all, or a very high quantity of, bad software but it is also desirable that it manages to classify good software correctly. Furthermore, we need the tool to respond rather quickly when an application presents a EULA. However, the actual training phase could be performed on a central server and the generated classifier(s) could then be downloaded by the tool periodically. Thus, there are no specific requirements related to classifier training time. Finally, it is desirable that the tool can visualize what element(s) in the EULA triggered a certain classification.

7.6.2 Potential Problems

It could be argued that, if the prevention tool is made available, the spyware authors would tailor their EULA around it. We believe that this argument does not hold since, in order to avoid legal repercussions, the spyware authors are in most countries required to mention in the EULA that spyware will be installed. We exploit this fact and use it against the spyware distributors.

Another argument against the tool is that there are already quite sophisticated tools
for prevention and removal of spyware (e.g.: AdAware\(^7\)). However, the idea is not to create a replacement for such products. Essentially, the spyware prevention tool should work as a complement that could be used to detect spyware that has not yet been classified by anti-spyware software.

### 7.7 Conclusions and Future Work

We have investigated the relationship between the contents of End User License Agreements (EULAs) and the legitimacy of the associated software applications. For this purpose, we collected a data set that features 996 EULA instances of legitimate (good) and spyware associated (bad) software. This is a text classification task and we argue that supervised learning is a suitable approach to the problem. Since most supervised learning algorithms cannot handle unstructured text input, we had to convert the data set to a manageable format. We therefore opted to use the bag-of-words model, in which text documents are converted to feature vectors. We compared this model to a meta EULA model that describes each EULA using several text analysis metrics.

We applied 17 supervised concept learning algorithms from several algorithm categories, such as: kernel functions, instance-based learners, tree and rule inducers, and Bayesian learners. The main objective was to investigate the possibility to classify software as good or bad by training classifiers on the associated EULAs.

For both data models, the experimental results show that a majority of the featured algorithms significantly outperformed a baseline classifier based on majority class decision. However, the results indicate that the bag-of-words model was more suitable than the meta EULA model, at least for the studied algorithms.

The results support our hypothesis that EULAs can indeed be used as a basis for classifying the corresponding software as good or bad. Based on this, we conclude that it would be quite possible to use the EULA classification method in a spyware prevention tool that classifies the EULA when it is shown to the user during an application installation. The result from such an analysis gives the user a recommendation about the legitimacy of the application before the installation continues. There are several directions for future work. For example, we intend to:

- select metrics, weights, and ranges for CEF in a more informed manner, e.g., by interviewing experts and potential users,

- develop a spyware prevention tool, based on EULA classification, that can help users to make informed decisions about the software they install,

- investigate metric-based algorithms and other approaches to optimize the CEF metric, including to minimize the false positives rate.

\(^7\)http://www.lavasoft.com
• merge the meta EULA and the bag-of-words data sets to find out if the classification performance can be improved by having access to both the term frequencies and the meta information,

• compare the contents of the flagged sentences with the words discovered for EULAs classified as bad in the bag-of-words.
8.1 Introduction

We consider supervised concept learning, where the task is to infer general descriptions of concepts, i.e., classifiers, given examples with known class labels. Common metrics for evaluating classifiers are, e.g., accuracy and the area under the ROC curve. However, it is argued that, for a particular application, some evaluation metrics are more appropriate than others (Caruana & Niculescu-Mizil, 2006). In addition, certain application-specific trade-offs can only be captured by evaluating multiple metrics. Consequently, we recognize the need for a structured approach to evaluation in which more care is taken to consider the context and requirements of the application at hand.

Drummond (2006) argues that the evaluation metric must represent something we care about and notes that one difficulty is the diversity of the people who must be considered in this judgment (e.g., the researcher, the research community, and application end users). The conclusion is that it is unreasonable to expect that all these concerns can be captured by one single metric. In previous work (Lavesson & Davidsson, 2008) we
addressed this issue using an approach, which we denote Generic Multi-Criteria (GMC) evaluation.

We defined the Candidate Evaluation Function (CEF), which is a GMC metric that can combine any number of arbitrary metrics. CEF is applicable for a broad range of applications since it only specifies how to combine metrics, not which metrics to combine. However, when focusing on a particular application, we are still faced with the intricate problem of selecting appropriate metrics to include in CEF. To address this problem, we investigate the possibility of developing a framework for metric selection.

In Section 8.2 we discuss various aspects of application-oriented evaluation and describe the metric selection problem. This is followed by a review of related work in Section 8.3. We continue by suggesting a framework for metric selection in Section 8.4. Based on this framework, we give an example of how to approach the metric selection problem in Section 8.5. Finally, we draw conclusions and give pointers to future work in the last section.

8.2 Application-oriented Evaluation

Evaluation is a prerequisite to determine the capability of a particular learning algorithm or classifier and to rank different algorithms according to their suitability with respect to some problem. Arguably, the most frequently applied evaluation method has been to estimate the predictive accuracy, i.e., the accuracy of a classifier when classifying data that was not used for learning the classifier. Predictive accuracy has been estimated using statistical methods like cross-validation (Stone, 1974), which work by systematically partitioning the known data into training and testing sets. For example, the $n$-fold cross-validation test partitions the data into $n-1$ folds for training and 1 fold for testing. A classifier is then generated from the training folds and evaluated on the testing fold. Classifier generation and evaluation are repeated until each fold has been used for testing once. The estimation of predictive accuracy is obtained by calculating the average accuracy of the $n$ tests. Serious concerns have been raised against the validity of such estimations, for example, that the accuracy metric assumes equal class distribution and misclassification cost, which are unlikely to be true in real world applications (Provost et al., 1998).

8.2.1 No Metric Is Superior for All Problems

The no-free-lunch theorems (Wolpert, 1995) stipulate that no algorithm is superior for all problems. We argue that a relaxed analogy can be made to evaluation metrics in the sense that no single metric is superior for all problems, i.e., that all metrics are equally good taken over all possible problems. However, it has been argued that some evaluation metrics are more appropriate than others, given a particular type of problem (Caruana
& Niculescu-Mizil, 2006). For example, it might be more important to have a low false positives rate than a high accuracy.

### 8.2.2 Some Metrics Are More Suitable Than Others

With regard to alternative performance metrics, for example, the Area under the ROC Curve (AUC) has been shown to be a more suitable metric than accuracy for several problems mainly because it is independent of class distribution and misclassification costs. In addition, it has an appealing statistical property in that the AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance (Fawcett, 2001).

One possible problem with the AUC metric is that it is derived from a ROC curve, which is a two-dimensional depiction of classifier performance. If two classifiers are compared and one of them achieves a higher AUC score than the other, it is still possible that the highest scoring classifier performs worse than the other in a specific region of ROC space. Thus, AUC can be more or less reliable depending on the situation. Moreover, there are other metrics that are designed for, and might therefore be more applicable to, certain types of problems. For example, the F-measure (Rijsbergen, 1979) measures the effectiveness of document retrieval and it is used predominantly in the area of information retrieval. The F-measure, AUC, and accuracy have one thing in common; essentially they are all classification performance metrics.

### 8.2.3 One Criterion Does Not Fit All Purposes

There are often space and time requirements that prohibit the use of many popular techniques in real-world applications, for example, when test sets are extremely large or where storage space or computational power is severely limited (Bucila, Caruana, & Niculescu-Mizil, 2006). Thus, in order to select appropriate classifiers or algorithms for such applications we need also to evaluate performance in terms of space and time. We make a distinction between criteria and metrics in the sense that the latter are used to evaluate the former. Classification, space, and time are all sub criteria of the performance criterion but it makes sense to discuss them explicitly as distinct criteria because the term performance is frequently used as a synonym for one of them (classification performance).

If the goal is to discover new knowledge to be used in human decision making there might be completely different criteria to consider in addition to performance. For example, comprehensibility (Dehuri & Mall, 2006) relates to how simple it is for us to understand the classifier and its decision process.

In addition, a crucial aspect of data mining is that the discovered knowledge should be somehow interesting. This is related to the interestingness criterion (Freitas, 1998), which has to do with the unexpectedness, usefulness and novelty of the discovered
We have now highlighted some examples of criteria that can be important to evaluate but we believe there are many more to consider for various domains. As a consequence, we argue that evaluation criteria should be selected on the basis of the problem at hand. Moreover, it seems plausible that one should first decide on what criteria to evaluate before selecting metrics. This might sound obvious, but many studies seem to assume that performance is the only suitable criterion to evaluate.

8.2.4 Mapping Criteria to Metrics

Freitas (1998) notes that interestingness has both an objective (data-driven) and a subjective (user-driven) aspect. We argue that these two objectives together with the mere complexity of the concept of interestingness suggest that this criterion can be measured using a host of metrics. Coincidentally, there are also several ways to measure comprehensibility. For example Dehuri and Mall (2006) measures the comprehensibility of rule-based classifiers by counting the number of rules and conditions in the rules. While the size of rules intuitively has something to do with the level of difficulty in interpreting them, other studies would associate such a metric with another criterion; complexity, cf., Andersson et al. (1999). Additionally, criteria such as interpretability and explainability are often used as synonyms for comprehensibility but sometimes their descriptions reveal slight differences. This would suggest that there is a need for; (i) universal machine learning definitions of relevant evaluation criteria and (ii) systematic approaches to map criteria to metrics.

8.2.5 Multiple Criteria and Multiple Metrics

The requirements of real-world applications often imply that a suitable trade-off between several important criteria is desired (Joachims, 2005). For example, Dehuri and Mall (2006) note that the accuracy and comprehensibility of rules are often in conflict with each other. Thus, depending on the application, the evaluation may involve trading off multiple criteria. Since metrics are used to evaluate criteria, this means that an approach is needed to trade off multiple metrics for these cases. Certainly, there exist defined metrics for many general trade-offs, e.g., the Area under the ROC Curve is used to capture the trade-off between hit rate and false alarm rate (Witten & Frank, 2005). However, for more specific situations it is not trivial to find a reasonable multi-criteria metric.

8.2.6 Generic Multi-criteria Metrics

We believe that there is a fundamental problem with some proposed multi-criteria metrics in that they are usually based on a static set of metrics. For example, Caruana and
Niculescu-Mizil (2004) propose the SAR metric, which combines Squared Error, Accuracy, and the Area Under the ROC metrics. This metric can of course be valid for a certain application, however it helps us little if we need to compare other criteria as well. Thus, we need a standardized way to trade off multiple metrics but we would like the actual choice of which metrics to combine to be decided based on the particular application studied. We therefore suggest the use of Generic Multi-Criteria (GMC) metrics, which dictate how to integrate metrics but do not specify what metrics to include.

It could be argued that there is no need for GMC metrics and that the integration could instead be tailor-made for the application at hand. However, we argue that the benefits from using a GMC metric is that different instances of such a metric could be more easily compared and refined across studies and applications. In addition, the use of a GMC metric shifts the focus from integration issues to application issues.

The GMC metric approach also simplifies the development and use of what we refer to as metric-based learning algorithms, i.e., algorithms that take a metric as input and try to learn a classifier with optimal performance with respect to this metric. A metric-based algorithm that is based on a GMC metric can thus be tailored for a particular application just by selecting relevant metrics and specifying trade-offs. Other learning algorithms have been specifically customized to optimize a certain metric, cf., Ferri et al. (2002); Herschtal and Raskutti (2004); Musicant et al. (2003); Joachims (2005). Thus, they cannot be made to optimize other metrics in a simple manner.

### 8.2.7 The Metric Selection Problem

We have discussed several aspects of application-oriented evaluation. It is important to understand the characteristics, requirements, restrictions, goals, and the overall context of the application in order to select relevant evaluation metrics and define meaningful trade-offs. Unfortunately, these aspects are often covered in an ad-hoc manner and some of them might not be covered at all.

### 8.3 Related Work

A number of studies address user-oriented and application-oriented evaluation of classifiers and learning algorithms. Nakhaeizadeh and Schnabl (1997) present a multi-criteria metric for learning algorithm evaluation and define the efficiency of an algorithm as its weighted positive properties (e.g., understandability) divided by its weighted negative properties (e.g., computation time). The metric weights for the efficiency metric are decided by an optimization algorithm in order to calculate objective weights. The problem with this approach is that these objective weights might not contribute to a realistic representation of the studied application. Moreover, the authors do not present any approaches to select metrics based on a particular application.
In a follow-up paper, Nakhaeizadeh and Schnabl (1998) suggest an approach to measure qualitative properties, such as understandability, with the efficiency metric. In addition, they introduce personalized algorithm evaluation, which is basically a collection of approaches to specify user preferences about restrictions and acceptable ranges on different metrics. The personalized evaluation approaches enable the user to influence the weights to reflect, e.g., that understandability is more important than accuracy. However, one of the main problems is that the user preferences are defined as linear equalities of inputs and outputs, which is not very intuitive. The paper features some discussion on how to quantify qualitative metrics, however, no information is given about how to select relevant metrics.

Soares et al. (2000) present an approach for user-oriented evaluation of learning algorithms. They define a multi-criteria metric, called the Simple and Intuitive Measure, which can be used to evaluate several metrics. The paper gives ideas on how to specify application restrictions by using bounds, however, similarly to the papers about efficiency, there is no description on how to select metrics to include in the evaluation. Kalousis and Theoharis (1999) describe an intelligent assistant for classifier selection. The assistant extracts data set characteristics combines this information with the performance results of different classifiers. Thus, when a new problem is studied, the assistant can give advice on the most suitable classifier. The main problem is that the performance data is limited (only accuracy and computation time are obtained). Secondly, there is no discussion about how these particular metrics were chosen.

Several studies suggest that quality criteria should be decided on the basis of the application at hand, cf., Fayyad, Piatetsky-Shapiro, and Smyth (1996); Francisci and Collard (2003); Freitas (2006). Although these studies give examples of what criteria can be useful for certain domains, there exist to our knowledge no systematic approaches to select metrics based on a particular supervised concept learning application.

### 8.4 A Framework for Metric Selection

We argue that the different aspects covered in Section 8.2 need to be considered when developing systematic approaches for selecting and evaluating application-specific metrics. We will now suggest a framework for metric selection that encompasses all the covered aspects. The framework uses concepts from the area of software engineering that enable a simple, structured path from application description to the choice of relevant metrics.

The software engineering community has long since focused on identifying and describing evaluation criteria related to both the software development process and the actual software product, cf., McCall, Richards, and Walters (1977); McCall (1994). The area of software engineering has also contributed a myriad of metrics for evaluating different criteria. Software engineers use the term Quality Attribute (QA) to refer to a
goal or an important criterion. In fact, there is an ISO standard\textsuperscript{1} that describes six key characteristics, or main quality attributes, under which the rest of the quality attributes, or sub criteria, can be organized. The machine learning community could benefit from introducing concepts and methods from software engineering.

The framework describes what we argue are necessary steps to take in application-oriented evaluation. These steps are shown in Figure 8.1. There are obviously many ways to perform these steps and the sequence to follow may vary slightly across different solutions. We will now go through each step and in the process highlight important considerations and give some suggestions on how the steps can be taken.

8.4.1 Identification of Quality Attributes

\textit{Without goals, and plans to reach them, you are like a ship that has set sail with no destination} – \textbf{Fitzhugh Dodson}

The first step is to describe the application by identifying which quality attributes are important. There are several definitions available but a quality attribute is typically regarded as a characterization of an application goal. Some examples of quality attributes include: \textit{understandability}, \textit{accuracy}, \textit{usability}, \textit{robustness}, and \textit{responsiveness}. Quality attributes are often organized hierarchically in taxonomies and are typically related by

\textsuperscript{1}ISO/IEC FDIS 9126-1:2000 (E)
sub type - super type relationships. For example, accuracy can be organized under either efficiency or performance. We argue that the concept of quality attributes enables a clear and simple description of important application characteristics. Another benefit of using this concept is that we can assess the importance of quality attributes by applying systematic prioritization methods from software engineering and other areas.

As we perceive it, the step of identifying quality attributes can be broken down into two sub steps, i.e., identification of; (i) generic quality attributes and (ii) specific quality attributes.

**Generic Quality Attributes**

Various taxonomies of quality attributes exist, e.g., the aforementioned ISO standard includes six main characteristics: functionality, reliability, usability, efficiency, maintainability, and portability.

Functionality is the essential purpose of any application. It is expressed as a totality of essential functions that the application provides. These functions are either present or absent, whereas the remaining characteristics are only present to a certain degree.

Reliability defines the capability of the application to work as expected under defined conditions for defined periods of time. Usability refers to how easy it is to use different functions. Efficiency is concerned with the resources used when providing the required functionality (e.g.: disk space, memory, and so forth).

There are often trade-offs between the characteristics. For example, the usability of an application is influenced by the performance (an instance-based learner might be slow in performing classifications, thus rendering a system less useable but on the other hand a fast classifier might not be as accurate which might also impact the usability negatively). Portability refers to how well the application can adopt to changes in its environment. Finally, maintainability refers to the ability to identify and fix a fault within a software component. Thus, it does not seem to be applicable in any obvious way to machine learning evaluation.

The six characteristics, or quality attributes, are further divided into sub attributes. For example: efficiency is broken down into time and space behaviors. Some of the quality attributes featured in the ISO standard might not be applicable to learning or data mining applications, e.g., maintainability and portability. We therefore suggest the development of a generic quality attribute taxonomy especially targeted toward machine learning and data mining applications. Such a taxonomy can then be browsed through to identify attributes that explain what the application should do and how. More specifically, we are interested in expressing what actions the application should be able to execute and what kind of behavior we regard as unacceptable or commendable. As a starting point, we can regard the previously discussed evaluation criteria from Section 8.2.3 as candidate quality attributes. Hopefully, they could serve as a basis for further discussions and research.
Specific Quality Attributes

We would argue that it is impossible to create a taxonomy that encompasses all important quality attributes across different application domains. Thus, when the generic attributes have been selected it is important to identify relevant attributes that are more specific to the studied application. A possible approach to perform this step is to formalize the characteristics or goals of the application and extract attribute-like bits of information from the results.

8.4.2 Prioritization of Quality Attributes

After identifying the quality attributes it is important to prioritize them, i.e., to assess the relative importance of each quality attribute. One rationale for performing this step before selecting metrics is that we can avoid spending time on identifying metrics for low-priority quality attributes that we may not even include in the final evaluation. However, the main objective of prioritization is to elicit information that can be used to determine metric weights. Ideally, we want to optimize our application for all relevant quality attributes but we argue that this is practically impossible since any given application has trade-offs. Thus, changing one quality attribute often forces a positive or negative change in another quality attribute. If we consider important trade-offs of quality attributes, by prioritizing them, it is plausible to assume that the trade-offs can be incorporated more easily into the multi-criteria evaluation. Thus, quality attribute prioritization can be viewed as part of a systematic approach to determine metric weights for a multi-criteria metric. This is important since multi-criteria metrics have been criticized based on the fact that the setting of weights for the included metrics is often done ad hoc (cf., Freitas, 2004).

8.4.3 Selection of Metrics

In the context of supervised concept learning, metrics are predominantly used to evaluate classifiers and learning algorithms even though recent work has put more attention to the fact that metrics also constitute an integral part of the algorithmic bias of learners, i.e., they can also be used as learning metrics.

Most of the current evaluation metrics are used to evaluate performance. As we mentioned earlier, some studies do focus on other aspects, such as: complexity, understandability, and so forth. Still, to our knowledge, the notion that several qualities do exist, and that there can be trade-offs between different qualities, is seldom discussed in the machine learning and data mining communities.

Metrics are discussed in terms of their merit, e.g., of measuring the generalization performance of an algorithm. Conversely, we argue that there is more to understand about metrics for learning algorithms and classifiers. A more diverse set of metrics, and
a more formalized categorization of these metrics, would arguably benefit both machine learning and application areas such as data mining.

The software engineering area has contributed with a number of categorizations of metrics, cf., Oman and Pfleeger (1996). We believe that some of these categorizations could be used in, and would benefit, the machine learning and data mining communities. We therefore give some brief examples of categorizations that could be relevant.

For example, metrics can be labeled as objective or subjective, where the latter involves human judgment. Most of the metrics used in concept learning today (e.g.: accuracy, AUC, training time) are objective. However, more and more emphasis is put on measuring usefulness and understandability. A classifier that distinguishes between Spam and legitimate emails does not need to be understandable; we are content if the classifier quietly processes our emails. On the contrary, we might expect from a classifier used for supporting medical diagnosis that it can visualize the decision process.

In addition, metrics can either be direct or indirect. A direct metric does not involve measuring any other metrics. Examples of such metrics, in the machine learning context, are: the number of true/false positives (TP/FP) and the number of true/false negatives (TN/FN). In contrast, an indirect metric always involves measuring at least two other metrics. A good example is the true positives rate \( \text{TPR} = \frac{TP}{TP + FP} \). The question is if this categorization is relevant for machine learning. Suppose that we would like to use a particular metric, which happens to depend on the calculation of two additional metrics. In turn, these metrics might require information that we are unable to produce for the studied problem. Thus, it would be beneficial if we can clearly judge from a categorization of metrics that some metrics are dependent of the calculation of other metrics.

Other categorizations include (Solingen & Berghout, 1999): explicit vs. derived metrics, absolute vs. relative metrics, and dynamic vs. static metrics. Some of the above mentioned categorizations apply more to machine learning than others. Moreover, it might be possible to identify additional categorizations that are specific to machine learning.

The main idea we wish to convey is that, by categorizing evaluation metrics, we add systemacy to the process of evaluation and simplify the application-specific selection of metrics. Essentially, we recognize the relationship between quality attributes (goals) and metrics (quantifications of the achievements of goals) that exist in software engineering. We argue that this relationship can be exploited to serve our needs as well. Thus, we suggest an extension of the aforementioned quality attribute taxonomy by adding evaluation metrics (as terminal nodes).

### 8.4.4 Metric Weights and Acceptable Ranges

Each metric should be assigned a weight, indicating the relative importance of the quality attribute it evaluates. In many multi-criteria metric settings, this kind of weight is often
set ad hoc. However, we argue that the prioritization step enables a more systematic setting of weights.

The acceptable range for an evaluation metric is in itself an interesting aspect to consider and a valuable asset for capturing trade-offs. An acceptable range has a lower bound, which indicates the least desired metric score, and an upper bound, which indicates the actual desired score. Metric scores that are better than the desired score are not counted as better than the desired scores. The justification for using a range is that, in doing so, we can more easily tune the evaluation towards capturing a specific trade-off. Suppose, for example, that we compare the performance of two classifiers using metrics for classification accuracy and classification time (the amount of time needed to classify some set of instances). Let us assume that the desired classification time is 500 ms and that the acceptable range for accuracy is 80% - 90%. If both classifiers manage to achieve at least the desired time but their accuracy scores reside somewhere in the acceptable range, the less accurate classifier might still be regarded as the best classifier if it achieves a better time score than the more accurate classifier. By removing any improvement over the desired time score we can make a more fair comparison.

The acceptable range can be static for some metrics, whereas for other metrics it is more dynamic, i.e., dependent of the context or problem. Essentially, this means that the dynamic ranges must be decided based on the application at hand. For example, the classification accuracy metric depends on the class distribution. For some data sets, an accuracy score of 90% would be quite impressive. On the other hand, for a data set in which 90% of the instances belong to the same class this score could be utterly worthless.

Moreover, the use of one bound (e.g.: the lower bound or tolerance threshold) is actually just a special case of the more general range representation. It is achieved by adjusting the upper bound so that it is equal to the optimal score.

### 8.4.5 Multi-criteria Evaluation

As we have previously mentioned, multi-criteria evaluation is a prerequisite for capturing trade-offs between different quality attributes. Whether or not it is suitable to compute a single quantity from (weighted) metrics rather than, e.g., using graphical visualization techniques depends on the problem. In this paper we focus on multi-criteria metrics, i.e., single quantity multi-criteria evaluation. In our framework, the multi-criteria evaluation is the final step. The multi-criteria metric takes as input a set of metrics. Each metric is associated with a weight that corresponds to the relative importance of the metric (and the quality attribute it evaluates) and an acceptable range. The multi-criteria metric then outputs a numeric evaluation score.
8.5 An Example Approach

We have suggested a framework for metric selection. The aim with this framework is to concretize what we believe to be important steps to consider in application-oriented evaluation. Moreover, the framework also indicates a possible sequence for performing the steps. However, there are no standard approaches to any of them. Thus, a natural question that arises is how to address the steps in practice. We will therefore present an example approach, based on a combination of software engineering methods and the Candidate Evaluation Function.

8.5.1 Identification of Quality Attributes and Metrics

The Goal/Question/Metric method (GQM) (Solingen & Berghout, 1999) is an approach to software metrics, which is widely used in the software engineering community. The original ideas for the paradigm came from the need to solve a practical software engineering problem, namely, how to decide what needs to be measured in order to achieve the goals.

The GQM approach was intended to measure traditional software systems. In fact, the method is used to elicit relevant metrics from defined corporate, division and project goals. Hence, the approach cannot be directly applied to identify suitable quality attributes and metrics for machine learning applications. However, GQM has been tailored to evaluate how well goals are achieved for other applications, for example, web-based systems (Khan, 2008). We believe that the core ideas of GQM could very well be suitable for application-oriented evaluation of learning algorithms and classifiers. We represent goals with quality attributes, e.g., a goal for an application might be understandability. Furthermore, our main purpose is to study processes but rather to evaluate algorithms or classifiers.

The GQM Measurement Model

The GQM approach provides a framework that involves three steps, or levels. First, at the conceptual level, we define the goals of the application. Next, at the operational level, we formulate a set of questions that are used to characterize the achievement of a specific goal. Finally, at the quantitative level, we try to associate a set of metrics with every question in order to answer it in a measurable way. The result of GQM can be visualized as depicted in Figure 8.2.

Identifying Goals

The standard GQM method uses templates as a structured means to specify goals. The template uses five facets of information: object, purpose, focus, viewpoint, and context. However, we follow the GQM adaption by Khan (2008) and specify goals simply by...
Figure 8.2: A tree-structured visualization of GQM in terms of derived goals, questions, and metrics

referring to quality attributes. In order to identify relevant quality attributes we may use a taxonomy, as described earlier. For the purpose of discussion, we give some example goals (a parenthesized term indicates a sub quality attribute):

\[ G_1 \] (classification) performance

\[ G_2 \] (time) efficiency

\[ G_3 \] understandability

**Formulating Questions**

Once the goals (quality attributes) have been identified, the next step is to associate a set of questions to each goal. The whole idea here is to ask questions for which the answer is the result(s) from a (set of) metric(s). We provide one example question for each goal specified in Section 8.5.1:

\[ Q_1 \] What is the accuracy on the test set?

\[ Q_2 \] How long does it take to generate a classifier?

\[ Q_3 \] How understandable is the classifier?
Identifying Metrics

When we have associated relevant questions to each goal, we next try to identify metrics that give the answer to each question. Either we know directly from the question which metric to use or we can traverse a quality attribute taxonomy with metrics as terminal nodes. If we cannot find suitable metrics in existence, we can try to define new metrics. For example, in order to answer \( Q_3 \) we might define an application-specific metric. For each asked question, we give one example metric that can answer the question:

- \( M_1 \) accuracy (success rate)
- \( M_2 \) training time
- \( M_3 \) qualitative rating of simplicity of representation

8.5.2 Prioritization of Quality Attributes

The prioritization or assessment of the importance of the quality attributes is not done by GQM. We believe that this step is always subjective to some extent but there are methods that introduce a certain level of systemacy. One regularly used approach in the area of software engineering is pairwise comparison, which was introduced by Thurstone (1927). It refers to any process of comparing entities in pairs to judge which of each pair is preferred.

The approach was introduced to software engineering as part of the Analytic Hierarchy Process (AHP) by Saaty (1980). AHP has since been revised in different ways. In particular, a number of approaches to pairwise comparison have been suggested. We will not discuss the differences between these approaches in this paper, nor will we discuss the analytic hierarchy process in greater depth. Rather, we will present the general idea of pairwise comparison.

In this form of comparison, the main objective of comparing several alternatives (in our case; quality attributes), is decomposed into a set of sub comparisons. The alternatives are compared to each other in pairs and so if we want to compare three alternatives we have to perform three paired comparisons. More generally, for \( n \) alternatives, we would need to perform \( n(n - 1)/2 \) paired comparisons. For example, if we want to compare alternatives time efficiency (A), classification performance (B), and understandability (C), we would have to perform the following comparisons; A versus B, B versus C, and A versus C.

Each comparison is conducted by rating the preference for, or the subjective opinion on the importance of, one alternative in relation to the other alternative. For qualitative data such as preference or subjective opinions, we need an ordinal scale, e.g., between 1 and 3 for each alternative. In AHP, it is common to use the numbers 9, 7, 5, 3, and 1. However, according to Saaty (1980) the actual choice of numbers is of low importance. We label each number with a description that indicates different levels of importance or
Table 8.1: A pairwise comparison of three alternatives

<table>
<thead>
<tr>
<th>Alternative</th>
<th>Very strong</th>
<th>Strong</th>
<th>Equal</th>
<th>Strong</th>
<th>Very strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

preference, for example: very strong, strong, equal, and so forth. Thus, in our example we assign equal importance to 1 and very strong importance to 3.

We stress that the descriptions are what matters. Thus, a rating of 9 is not four steps better than a rating of 5. Table 8.1 shows an example comparison of three alternatives. After a paired comparison of the different alternatives it is possible to calculate a comparison matrix. This matrix can then be used to calculate priorities for each alternative. This means that, if we compare quality attributes, we will end up with a prioritized list of quality attributes. The descriptions of the exact calculation steps from ratings to priority vector are extensive. Thus, it does not serve any purpose to describe them here since we are not working with any real data.

8.5.3 Multi-criteria Evaluation

The main purpose of the Candidate Evaluation Function (CEF) is the integration of an arbitrary number of existing metrics in order to get a single quantity. Additionally, CEF normalizes each included metric, on the basis of its application-dependent acceptable range, in order to get a uniform output domain.

It is also possible for the user to specify explicit weights for each metric to ensure that trade-offs important for the application at hand can be properly represented. CEF itself does not dictate which metrics should be used; it merely dictates how metrics are combined.

Let \( c \) be a candidate classifier and \( D \) a data set. We then define \( m_i \) as a metric with index \( i \) from an index set, \( I \), over the selected set of metrics. Each metric is associated with a weight, \( w_i \), and an acceptable range, \( r = [b_{l_i}^i, b_{u_i}^i] \). The lower bound, \( b_{l_i}^i \), denotes the least desired acceptable score. Similarly, the upper bound, \( b_{u_i}^i \), denotes the desired score. Note that, in the original CEF definition a metric was normalized according to the best and worst score of that particular metric obtained from the studied set of classifiers. The current normalization uses the lower and upper bound to generate a smooth
distribution from 0 (least desired) to 1. CEF is then defined as specified in Equation 8.1.

$$CEF(c, D) = \begin{cases} 0 : \exists i (\overline{m}_i(c, D) < 0) \\ \sum_{i \in I} w_i \overline{m}_i(c, D) \quad \text{otherwise} \end{cases}$$

where $$\sum_{i \in I} w_i = 1$$ and

$$\overline{m}_i(c, D) = \begin{cases} 1 : \frac{m_i - b_{i'}}{b_{i'} - b_{i}} > 1 \\ \frac{m_i - b_{i'}}{b_{i'} - b_{i}} \quad \text{otherwise} \end{cases}.$$ (8.1)

An attractive feature of the priorities calculated from the pairwise comparison is that they sum up to 1. Thus, the priorities for each alternative, or quality attribute, can be used as metric weights for CEF. However, as discussed in Section 8.4.4, the acceptable ranges of metrics do not only vary across applications but also between metrics. For example, the acceptable range for AUC in general would seem to be between 0.5 and 1.0, since 0.5 denotes no predictive power and 1.0 denotes a perfect score (Fawcett, 2001), whereas for accuracy the perfect score would also be 1.0 but the generic lower bound depends on the class distribution (Caruana & Niculescu-Mizil, 2006). For both of these metrics, the ranges can be narrowed in different ways to suit application requirements. We recognize the need to empirically validate CEF. However, we argue that the definition above is sufficient for demonstrating some of the more important concepts of GMC evaluation methods. In particular, we argue that CEF has the capabilities needed to evaluate candidates based on the information obtained through GQM and AHP.

8.6 Conclusions and Future Work

There are arguably important aspects to consider when evaluating candidate learning algorithms or classifiers for a certain application: (i) some metrics are more suitable than others for evaluating an important criterion for the studied problem, (ii) there are often trade-offs between different criteria, (iii) there is not always a defined mapping between different criteria and the metrics that evaluate them, and (iv) multi-criteria metrics can capture application-specific trade-offs but the weighting of metrics is often ad hoc.

To address these different aspects, we use the concept of quality attributes from software engineering. A quality attribute can be viewed as a characterization of an important application goal. Thus, by identifying relevant quality attributes we can describe the application goals and assess how well candidate algorithms or classifiers achieve these goals.

We suggest a framework for metric selection that takes all the aforementioned aspects into consideration. The framework describes a set of steps that can be followed, more or less sequentially, to select relevant metrics and metric weights for a particular
application: (i) identify its quality attributes, (ii) prioritize these attributes, (iii) select suitable metrics for each attribute, and (iv) specify metric weights based on the prioritization.

We give an example approach to metric selection, based on the framework, which makes use of common software engineering methods. In this approach, the Goal/Question/Metric (GQM) method is used to elicit appropriate metrics, given a set of quality attributes. Pairwise comparison is suggested as an approach to prioritize the identified quality attributes. This prioritization leads to a priority vector, which can be used for specifying the weight of each metric. Finally, we use a multi-criteria metric, called the Candidate Evaluation Function, for the purpose of evaluating candidate classifiers or learning algorithms based on the identified metrics.

Our conclusion is that, by using the concept of quality attributes, we can easily adopt methods for identification, prioritization, and metric selection from the area of software engineering. One step in the GQM approach is to ask questions about application goals (quality attributes) in order to determine appropriate quantifications of the achievement of each goal (metrics). What could be considered as an attractive side-effect of asking these questions is that they can represent explicitly defined research questions. Consequently, if researchers publish the questions and the quality attributes they address, we hypothesize that it will be more clear what properties of learning are actually studied. In addition, other researchers might pose the same questions but evaluate a different set of algorithms. In such cases, it would seem plausible that comparisons across studies will be simpler to perform.

For future work, we suggest the development of a machine learning oriented quality attribute taxonomy to facilitate the mapping from typical machine learning and data mining goals to relevant evaluation metrics. Another interesting direction would be to more deeply investigate methods for organizing, prioritizing, and selecting relevant quality attributes (e.g., the Analytic Hierarchy Process). The concept of acceptable ranges should be investigated further to enable a more systematic configuration of metric ranges on the basis of the application.

Finally, we believe that it would be possible to develop a software-based tool for application-oriented evaluation. This tool should preferably be based on some software library of machine learning algorithms, e.g., the Weka machine learning workbench (Witten & Frank, 2005), to enable the evaluation of common algorithms and the processing of standard formatted data sets.

Acknowledgments

We would like to thank Dr. Mikael Svahnberg and Dr. Tony Gorschek from Blekinge Institute of Technology for their ideas and valuable suggestions.
REFERENCES


the Vapnik-Chervonenkis dimension. ACM, 36(4), 929–965.
nordic workshop on secure it systems (pp. 23–30). Espoo, Finland: Helsinki Uni-
versity of Technology.
detection using end user license agreements. In 2nd international conference on
Research, 2, 499–526.
Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of
machine learning algorithms. Pattern Recognition, 30(7), 1145–1159.
ACM SIGKDD international conference on knowledge discovery and data mining.
New York City, NY, USA: ACM Press.
Data Mining and Knowledge Discovery, 2(2), 121–167.
R. Mitkov, G. Angelova, K. Bontcheva, N. Nicolov, & N. Nikolov (Eds.), Euro-
pean conference on recent advances in natural language processing (pp. 58–64).
Bulgaria: Tzigov Chark.
analysis of supervised learning performance criteria. In 10th ACM SIGKDD inter-
national conference on knowledge discovery and data mining (pp. 69–78). New
York City, NY, USA: ACM Press.
learning algorithms. In 23rd international conference on machine learning (pp.
training. In 1999 IEEE congress on evolutionary computation (Vol. 3, pp. 1999–
parameters for support vector machines. Machine Learning, 46, 131–159.
machine learning (pp. 115–123). San Francisco, CA, USA: Morgan Kaufmann
Publishers.
that Classify E-Mail). Amsterdam, the Netherlands: IOS Press.
Coleman, M., & Liau, T. L. (1975). A computer readability formula designed for


puting Surveys, 34(1), 1–47.


ABSTRACT
A classifier is a piece of software that is able to categorize objects for which the class is unknown. The task of automatically generating classifiers by generalizing from examples is an important problem in many practical applications. This problem is often referred to as supervised concept learning, and has been shown to be relevant in e.g. medical diagnosis, speech and handwriting recognition, stock market analysis, and other data mining applications.

The main purpose of this thesis is to analyze current approaches to evaluate classifiers as well as supervised concept learners and to explore possible improvements in terms of alternative or complementary approaches. In particular, we investigate the metric-based approach to evaluation as well as how it can be used when learning.

Any supervised concept learning algorithm can be viewed as trying to generate a classifier that optimizes a specific, often implicit, metric (this is sometimes also referred to as the inductive bias of the algorithm). In addition, different metrics are suitable for different learning tasks, i.e., the requirements vary between application domains. The idea of metric-based learning is to both make the metric explicit and let it be defined by the user based on the learning task at hand.

The thesis contains seven studies, each with its own focus and scope. First, we present an analysis of current evaluation methods and contribute with a formalization of the problems of learning, classification and evaluation. We then present two quality attributes, sensitivity and classification performance, that can be used to evaluate learning algorithms. To demonstrate their usefulness, two metrics for these attributes are defined and used to quantify the impact of parameter tuning and the overall performance. Next, we refine an approach to multi-criteria classifier evaluation, based on the combination of three metrics and present algorithms for calculating these metrics. In the fourth study, we present a new method for multi-criteria evaluation, which is generic in the sense that it only dictates how to combine metrics. The actual choice of metrics is application-specific. The fifth study investigates whether or not the performance according to an arbitrary application-specific metric can be boosted by using that metric as the one that the learning algorithm aims to optimize.

The subsequent study presents a novel data mining application for preventing spyware by classifying End User License Agreements. A number of state-of-the-art learning algorithms are compared using the generic multi-criteria method. Finally, in the last study we describe how methods from the area of software engineering can be used to solve the problem of selecting relevant evaluation metrics for the application at hand.