EVALUATION AND ANALYSIS OF SUPERVISED LEARNING ALGORITHMS AND CLASSIFIERS

Niklas Lavesson

Blekinge Institute of Technology
Licentiate Dissertation Series No. 2006:04

School of Engineering

Evaluation and Analysis of Supervised Learning Algorithms and Classifiers

Niklas Lavesson

Blekinge Institute of Technology Licentiate Dissertation Series
No 2006:04
ISSN 1650-2140
ISBN 91-7295-083-8

Evaluation and Analysis of Supervised Learning Algorithms and Classifiers

Niklas Lavesson

Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
SWEDEN

Department of Systems and Software Engineering
School of Engineering
Publisher: Blekinge Institute of Technology
Printed by Kaserntryckeriet, Karlskrona, Sweden 2006
ISBN 91-7295-083-8

Even the recognition of an individual whom we see every day is only possible as the result of an abstract idea of him formed by generalisation from his appearances in the past

- James G. Frazer (in Malinowski & Bronislaw: Argonauts of the Western Pacific, 1922)

Abstract

The fundamental question studied in this thesis is how to evaluate and analyse supervised learning algorithms and classifiers. As a first step, we analyse current evaluation methods. Each method is described and categorised according to a number of properties. One conclusion of the analysis is that performance is often only measured in terms of accuracy, e.g., through cross-validation tests. However, some researchers have questioned the validity of using accuracy as the only performance metric. Also, the number of instances available for evaluation is usually very limited. In order to deal with these issues, measure functions have been suggested as a promising approach. However, a limitation of current measure functions is that they can only handle two-dimensional instance spaces. We present the design and implementation of a generalised multi-dimensional measure function and demonstrate its use through a set of experiments. The results indicate that there are cases for which measure functions may be able to capture aspects of performance that cannot be captured by cross-validation tests. Finally, we investigate the impact of learning algorithm parameter tuning. To accomplish this, we first define two quality attributes (sensitivity and classification performance) as well as two metrics for measuring each of the attributes. Using these metrics, a systematic comparison is made between four learning algorithms on eight data sets. The results indicate that parameter tuning is often more important than the choice of algorithm. Moreover, quantitative support is provided to the assertion that some algorithms are more robust than others with respect to parameter configuration. To sum up, the contributions of this thesis include; the definition and application of a formal framework which enables comparison and deeper understanding of evaluation methods from different fields of research, a survey of current evaluation methods, the implementation and analysis of a multi-dimensional measure function and the definition and analysis of quality attributes used to investigate the impact of learning algorithm parameter tuning.

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Paul Davidsson, for sharing his interesting ideas, knowledge and experience, and for supporting me through out the course of writing this thesis.

I would also like to thank my secondary supervisor, Dr. Stefan Johansson, and the rest of the members of the Distributed and Intelligent Systems Laboratory (DISL) for reviews of my papers and interesting discussions.

This thesis would not have been possible to write without the love and support from my family.

Preface

The thesis is based on the following papers:

I. Niklas Lavesson and Paul Davidsson. An Analysis of Approaches to Evaluate Learning Algorithms and Classifiers. To be submitted for publication.

II. Niklas Lavesson and Paul Davidsson. A Multi-dimensional Measure Function for Classifier Performance[1]. In *2nd IEEE Conference on Intelligent Systems*, pp. 508–513. Varna, Bulgaria, 2004.

III. Niklas Lavesson and Paul Davidsson. Quantifying the Impact of Learning Algorithm Parameter Tuning[2]. Submitted for publication, February 2006.

[1]The version included in this thesis is a modification of the published paper.

[2]An early version of this paper, which only included limited experiment results, was published in the proceedings of the *SAIS-SLSS Workshop on Artificial Intelligence and Learning Systems*, Västerås, Sweden, 2005 (pp. 107–113).

LIST OF FIGURES

LIST OF TABLES

CONTENTS

CHAPTER

ONE

Introduction

This thesis concerns the evaluation of classifiers and learning algorithms. An example of a classification problem is to detect the presence or absence of a particular cardiac disorder, or distinguish between different cardiac disorders based on, e.g., electrocardiogram (ECG) recordings [47]. 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. More generally, given a set of examples, or instances, the learning algorithm generates a classifier that is able to classify instances for which the class membership is unknown. Each instance is described by a set of attributes. There are many learning algorithms available and, depending on the problem at hand, some algorithms may be more suitable than others. In order to make informed choices regarding the appropriate algorithm, or to select between some candidate classifiers, we need evaluation methods. The number of examples available for generating a classifier is usually very limited. Thus, it is important to make use of these known examples in a way that enables the generation of a good classifier as well as an accurate evaluation. Since most algorithms can be configured, or tuned for a particular problem the evaluation process can be used to select algorithm configuration as well. It may also be important to assess the extent to which this type of tuning really affects the performance of a generated classifier.

In this chapter a background is first given to machine learning, and this is followed by an introduction to classifiers, learning algorithms, and evaluation methods. Section 1.2 then presents the motivation for the thesis and introduces the persued research questions and Section 1.3 describes the research methods that have been applied to answer these questions. The next section summarises the contributions of the three papers included in the thesis. At the end of the chapter, conclusions are presented and this is followed by a section that includes some pointers to future work.

1.1 Learning

Machine learning is an interdisciplinary field of research with influences and concepts from, e.g., mathematics, computer science, artificial intelligence, statistics, biology, psychology, economics, control theory, and philosophy. In general terms, machine learning concerns computer programs that *improve their performance at some task through experience* [45], i.e., programs that are able to learn. The machine learning field has contributed with numerous theoretical results, learning paradigms, algorithms, and applications. Among the different applications developed in this field are, e.g., autonomous vehicles that learn to drive on public highways [48], investigations of past environmental conditions and dependencies among co-existent plant species through time [36], life insurance applicant classification [68], handwriting recognition [42], and leak detection in pipelines [14].

Generally, there are three issues that affect the design of computer programs that learn from observations:

- what is the *input*,
- what type of *feedback* is available, and
- how should the solution be *represented*.

Additionally, it is useful to consider how to evaluate learning programs and solutions for different purposes, e.g.:

- ranking of programs or program configurations,
- ranking of solutions for a particular problem, or
- assessment of generalisation performance.

1.1.1 Input

Usually, the input to a learning program consists of a sample of data instances, i.e., a number of observations. Instances are described by a set of attributes, where each attribute can be of a different type, e.g., real, integer, or boolean. By selecting appropriate attributes, instances can be made to describe anything from cars to bank customers to handwritten characters. For instance, the attributes length, width, height, colour, top speed, and horse power could be used to describe cars. Handwritten characters could be described by a set of integer valued attributes, where each value would correspond to the colour of a pixel in an image scanned from an actual handwritten character.

1.1.2 Feedback

It is usually distinguished between three types of feedback for observational learning; supervised, unsupervised and reinforcement [58]. Supervised learners have access to a supervisor, or teacher, that gives the *correct* answers (outputs) to a limited number of questions (inputs). Sometimes, e.g., when analysing real world problems, it could be relevant to assume that this teacher cannot always provide the correct answers, i.e., the possible existence of noise is assumed. The objective is to learn how to generalise from what has been taught, i.e., to give the correct answer to previously unknown questions.

Reinforcement learners get feedback by means of occasional rewards for the actions or decisions they recommend. This type of learners is often used when the mapping from inputs to outputs involves several steps. For example, learning how to land an aircraft involves hundreds, or maybe thousands of steps. Instead of just receiving positive or negative feedback after having tried to land the aircraft (success or failure) it is perhaps more wise to use a reward system that gives small rewards for each correctly executed step, or chain of subsequent steps.

Unsupervised learners do not have access to any outputs, hence the objective is often to find patterns in the available input data. A purely unsupervised learner cannot learn what to do in a certain situation since it is never given information about whether or not an action is correct [58].

Which type of feedback is available depends on the problem at hand. We will here focus on supervised learning problems.

1.1.3 Representation

Depending on the type of output considered for a supervised learning problem (continuous or discrete), the solution is represented by a regression function or a classifier. A regression function is defined by a number of attributes (inputs) and coefficients, and gives a continuous output. During the learning phase, the coefficients are tuned in order to produce the correct output given a particular input. A classifier is a function that gives discrete output, i.e., it assigns a discrete value, often denoted a class, to a particular input.

1.1.4 Evaluation

It is important make a clear distinction between learning algorithms and classifiers. A learning algorithm trains by observing known data, i.e., instances for which there is a correct supervisor response. According to some internal bias, it then generates a classifier. This classifier can then be used to classify new data of the same kind.

One fundamental problem to consider here, is how to find out if the classifications of new data are correct, i.e., how do we measure the generalisation performance of a classifier or the capability of a learning algorithm (the ability to select a classifier that generalises well)?

In general, it can be said that the existing evaluation methods are either empirical or theoretical. Empirical methods evaluate classifiers on portions of known data which have *not* been used for training, while theoretical methods either evaluate classifiers on training data only or combine this evaluation with a theoretic measure of generalisation performance.

Furthermore, we may distinguish between evaluation methods by which metric(s) they use for measuring performance. *Accuracy* (the number of correct classifications) is one of the most common metrics used, along with *error* (the number of incorrect classifications). These metrics are either used alone or combined with other metrics, e.g., some methods combine accuracy with a classifier complexity metric. In these methods, the measured complexity is usually subtracted from the accuracy, in order to reward simple solutions.

Penalising complexity is often justified by the theory of *Ockham's razor* [69] stating that entities should not be multiplied beyond necessity. The common interpretation of this theory is that if we have two equally good solutions we should pick the simplest one. Another type of complexity metric, the VC dimension, is used to measure the ability of an algorithm to shatter instances, i.e., discriminate between instances of different classes. A high value indicates a high algorithm capacity in discriminating between instances of different classes. A high complexity in this respect is often bad since the generated classifier may overfit to the training data. On the other hand, a very low capacity may result in a too general classifier [12]. Other metrics used by classifier evaluation methods include similarity and misclassification cost (e.g., true/false positives).

1.2 Motivation and Research Questions

Evaluation is necessary in order to compare different algorithms or classifiers, but which evaluation method should be used? Several evaluation methods of different origin have been proposed, e.g., from statistics, mathematics, and artificial intelligence. Which evaluation method to use is still an open research question, because evaluating a classifier often depends on the difficult task of measuring generalisation performance, i.e., the performance on new, previously unseen data. For most real-world problems we can only estimate the generalisation performance. Thus, it is important to continue investigating the existing evaluation methods theoretically and empirically in order to determine the characteristics of different estimations and when a certain method should be used. The idea to look at evaluation methods for classifier performance came from reading a theoretical paper on the topic of measure-based evaluation [7]. However, the practical use of this particular method has not been investigated.

To evaluate learning algorithms, we usually evaluate some of the classifiers it can generate. Methods that divide the available data into training and testing partitions a number of times (to generate classifiers from several training sets) have been investigated in many papers. However, learning algorithms like many computer programs can usually be configurated and we argue that it could be important to look at how well an algorithm can be tuned for a specific problem, and not only how well it performs on a particular problem.

The fundamental question that we study is how to measure the performance of supervised learning algorithms and classifiers. Hence, the scope is narrowed from general learning problems to only include learning problems for which:

- the input consists of a sample of instances defined by a set of attributes,
- the correct output for each instance of the sample is available via super-

vised feedback, and

- the goal is to classify previously unseen instances.

The question has then been broken down into a number of research questions:

RQ1 Which methods exist for the evaluation of classifiers and learning algorithms, what are the strengths and weaknesses, which metrics do they use, and how can we categorise the methods?
RQ2 How can evaluation methods, originating from different research disciplines, be described in a uniform way?
RQ3 How does measure-based evaluation work in practise?
RQ4 How can we measure the impact that learning algorithm parameter tuning has on classifier performance?

1.3 Research Methods

The research questions have been approached using different research methods. RQ1 was approached by conducting a *literature review* to find and learn about existing evaluation methods for algorithms and classifiers. After compiling the results from the literature review, an *analysis* was performed on each method to determine the weaknesses and strengths. The evaluation methods have also been categorised according to a number of relevant properties chosen after examining the results from the review and analysis.

The evaluation methods originate from different research disciplines. Consequently, they are described in the literature using different terminology and concepts. RQ2 addresses this issue and the motivation behind this question is that an answer could simplify comparison and presentation of the evaluation methods . A *formal framework* was defined to describe all methods in a uniform and structured way.

Experiments were conducted in order to understand how measure-based evaluation works in practise and to determine how the impact of learning algorithm tuning can be measured.

In the first experiment, addressing RQ3, a measure-based evaluation function was implemented using measure-based evaluation theory. Claims and ideas about measure-based evaluation was empirically strengthened by evaluating different classifiers using the implemented function and comparing some of the results with cross-validation evaluation results.

RQ4 was approached by defining two learning algorithm quality attributes, sensitivity and classification performance, and two metrics to assess each of these attributes. The second experiment then involved cross-validation evaluation of a large number of configurations of a set of common algorithms. The results were used to measure algorithm parameter tuning impact (sensitivity) and classifier accuracy (classification performance).

1.4 Contributions

The contributions of this thesis come from three different papers. RQ1 and RQ2 are addressed in paper I, RQ3 is addressed in paper II, and RQ4 is addressed in paper III.

Paper I includes a literature review of the area of evaluation methods for supervised learning algorithms and classifiers and an analysis of the methods found during the review. The featured evaluation methods are; Vapnik-Chervonenkis Bound (VCB) [70, 71], Minimum Description Length (MDL) [56], Cross-validation (CV) [65], Jackknife (JK) [51], Bootstrap (BS) [20, 35], Structural Risk Minimisation (SRM) [70], Sample Error (SE) [45], Receiver Operating Characteristics Analysis (ROC) [21, 49], Area Under the ROC curve (AUC) [30, 74] and Measure-based example Function (MF) [7].

There are variations of these methods in the literature and other methods may also exist, but the listed methods appeared most frequently in different studies as well as in the reference literature, cf. [45, 58, 75].

Each method has been categorised according to type, target and metric(s) used. We have identified two types of evaluation methods; theoretical and empirical. Theoretical methods evaluate on the data that was used for training and usually combine this with some theoretical measure of generalisation performance, while empirical methods evaluate on test data. The target has been identified to be either classifier-dependent or classifier-independent algorithm evaluation, or algorithm-dependent or algorithm-independent classifier evaluation. The independent methods work for any algorithm or classifier, while the other methods are dependent of a certain classifier or algorithm. The identified metrics are; accuracy/error, complexity/simplicity, similarity, and cost.

The analysis revealed that the most common metric is accuracy. For instance, CV, BS, JK and SE are all based solely on accuracy. ROC analysis and

AUC use a cost metric, which is measured by recording true and false positives, i.e., instances can be classified correctly (true positive) or incorrectly (false positive) as belonging to a particular class, or they can be classified correctly (true negative) or incorrectly (false negative) as not belonging to the class [75]. This could be compared to methods that only use the accuracy metric, which only make a distinction between true or false, i.e., if the classifier predicted correctly or not. VCB and SRM combine an accuracy metric with a complexity metric. MF combines three weighted metrics; accuracy, simplicity and similarity into one measure for classifier performance. MDL uses a complexity metric based on the theory of Ockham's razor. Using this method, the classifier that needs the least amount of bits to represent a solution is regarded as the best, i.e., the least complex.

The strength, or weakness, of an evaluation method is often related to which metric(s) it uses. Accuracy alone is not a good metric, e.g., because it assumes equal misclassification costs and a known class distribution [50], thus the methods based solely on accuracy may suffer from these inadequate assumptions. Moreover, if accuracy is measured on the training data as an estimate of future performance it will be an optimistically biased estimate [75].

Other than these general conclusions about accuracy-based methods, the literature review helped identifying some characteristics of the different methods, e.g., CV (and JK) has been shown to have low bias and high estimation variance [19], while the opposite holds for bootstrap [8]. ROC and AUC were found to have some limitations, e.g., they can only be used for two-class problems [23] and only certain classifiers can be evaluated [75] (although there exist workarounds for the first limitation, cf. [30]). Regarding MDL, it has been argued that there is no guarantee that it will select the most accurate classifier and, perhaps more importantly, it is very hard to know how to properly calculate MDL for different classifiers, i.e., code the theory for different classifiers [17]. VCB includes elements that need to be explicitly defined for each algorithm to be evaluated. SRM uses VCB, thus it also suffers from this fact. MF is only a theoretical example and no empirical studies have been identified in the literature review. This makes it difficult to compare it with other evaluation methods.

The possibility of describing the current methods using a common framework would be useful since the relations between the methods are not entirely clear due to the fact that they are often described using different terminologies and concepts. One reason for this is that they originate from different research disciplines. We present a formal framework and apply it when describing the different evaluation methods. In this framework, classifiers are defined as mappings between the instance space (the set of all possible instances for a specific problem) and the set of classes. A classifier can thus be described as a complete classification of all possible instances. In turn, the classifier space includes all possible classifiers for a given problem. However, once a learning algorithm and a training set (a subset of the instance space) are chosen, the available classifier space shrinks (this is related to inductive bias, cf. [45]).

In paper II we investigate the practical use of measure-based evaluation. Prior work in the area of measure-based evaluation is theoretical and features a measure-based evaluation function example [7]. However, the algorithms for similarity and simplicity calculation are only defined for two-dimensional instance spaces, i.e., data sets with two attributes. In order to compare MF with other evaluation methods, a generalised version using new algorithms for similarity and simplicty is needed. We present the design and implementation of such a generalised version along with an empirical comparison to CV. The generalised version, which is called the multi-dimensional measure function (MDMF), measures similarity and simplicity on instance spaces of higher dimensions than two. The empirical and theoretical evaluations of MDMF indicate that MDMF may reveal aspects of classifier performance not captured by evaluation methods based solely on accuracy, e.g., it can distinguish between a simple and a complex classifier when they are both equally accurate.

Paper III addresses the question of how to measure the impact that learning algorithm parameter tuning might have on classifier performance. We define two learning algorithm quality attributes (QA) that are used to assess the impact of parameter tuning; classification performance and sensitivity. Two example metrics for each QA are also defined. Average and best performance, in terms of accuracy, are used to assess classification performance. Performance span and variance are used to assess sensitivity. A sensitive algorithm is here defined as an algorithm for which the impact of tuning is large. All four metrics are based on calculating the accuracy-based 10-fold cross-validation test score of a large number of classifiers (using different configurations of one algorithm) on a particular data set. A possible trade-off between the two quality attributes is investigated by analysing the correlation between the values of the four metrics. The conclusion is that there seems to be no trade-off, i.e., a sensitive algorithm does not necessarily have low classification performance or vice versa. The experiment also indicates that parameter tuning is often more important than algorithm selection.

1.5 Conclusions

The question of how to measure the performance of learning algorithms and classifiers has been investigated. This is a complex question with many aspects to consider. The thesis resolves some issues, e.g., by analysing current evaluation methods and the metrics by which they measure performance, and by defining a formal framework used to describe the methods in a uniform and structured way. One conclusion of the analysis is that classifier performance is often measured in terms of classification accuracy, e.g., with cross-validation tests. Some methods were found to be general in the way that they can be used to evaluate any classifier (regardless of which algorithm was used to generate it) or any algorithm (regardless of the structure or representation of the classifiers it generates), while other methods only are applicable to a certain algorithm or representation of the classifier. One out of ten evaluation methods was graphical, i.e., the method does not work like a function returning a performance score as output, but rather the user has to analyse a visualisation of classifier performance.

The applicability of measure-based evaluation for measuring classifier performance has also been investigated and we provide empirical experiment results that strengthen earlier published theoretical arguments for using measure-based evaluation. For instance, the measure-based function implemented for the experiments, was able to distinguish between two classifiers that were similar in terms of accuracy but different in terms of classifier complexity. Since time is often of essence when evaluating, e.g., if the evaluation method is used as a fitness function for a genetic algorithm, we have analysed measure-based evaluation in terms of the time consumed to evaluate different classifiers. The conclusion is that the evaluation of lazy learners is slower than for eager learners, as opposed to cross-validation tests.

Additionally, we have presented a method for measuring the impact that learning algorithm parameter tuning has on classifier performance using quality attributes. The results indicate that parameter tuning is often more important than the choice of algorithm. Quantitative support is provided to the assertion that some algorithms are more robust than others with respect to parameter configuration.

1.6 Future Work

Measure-based evaluation has been proposed as a method for measuring classifier performance [5, 6]. It has been analysed theoretically [7] and practically (paper II), and results indicate that measure-based evaluation may reveal more aspects of classifier performance than methods focusing only on accuracy, e.g., cross-validation tests. However, we intend to perform more extensive experiments to compare measure-based evaluation functions with other evaluation methods, e.g., ROC analysis and bootstrap.

Both the theory behind measure-based evaluation and the measure function implementation are at an early stage of development. The theoretical foundation of measure-based evaluation needs to be further developed and, as a consequence, the measure function design may need to be revised. For instance, the current measure function implementation only supports data sets with real-valued attributes and this has to be remedied to extend the area of application. Moreover, there might be other ways to measure simplicity and similarity, more suitable than those already proposed. Also, since measure-based evaluation is a general concept and is not limited to the three currently used metrics (simplicity, similiarity and subset fit), it is important to investigate the possible inclusion of other metrics.

A related topic, measure-based learning algorithms seems promising to follow up on and thus could be explored in future work. Measure-based algorithms have been introduced using a simple example [7], in which the idea was to create a classifier from training data using a measure-based evaluation function as a criterion for an optimisation algorithm. Starting from an arbitrary classifier, for instance, based on some simple decision planes, the optimisation algorithm incrementally changes the existing planes or introduces new ones. The decision space (the partitioning of the instance space that a classifier represents) is then evaluated by applying the measure function. These changes are repeated until the measure function returns a satisfactory value, i.e., above a given threshold.

The proposed learning algorithm quality attributes and the metrics used to evaluate them need to be investigated further in order to fully determine their applicability. The total number of possible configurations vary between different learning algorithms and the metrics used for measuring the sensitivity and classification accuracy of an algorithm depend on the evaluation of one classifier from each possible configuration of that particular algorithm. Estimates calculated using subsets of configurations have already been presented in our paper, however these estimates vary in accuracy depending on which algorithm is evaluated. Future work on this topic will focus on less biased estimates and different ways to measure sensitivity.

CHAPTER

TWO

Paper I

An Analysis of Approaches to Evaluate Learning Algorithms and Classifiers

Niklas Lavesson, Paul Davidsson
To be submitted for publication

2.1 Introduction

Important questions, often raised in the machine learning community, are how to evaluate learning algorithms and how to assess the predictive performance of classifiers generated by those algorithms, cf. [1, 7, 8, 16, 39, 50].

In supervised learning, the objective is to learn from examples, i.e., generalise from training instances by observing a number of inputs and the correct output [58]. Some supervised learning problems involve learning a classifier. For these problems, the instances belong to different classes and the objective of the learning algorithm is to learn how to classify new, unseen instances correctly. The ability to succeed with this objective is called *generalisation capability*. One example of a supervised classifier learning problem would be that of learning how to classify three different types of Iris plants by observing 150 already classified plants (50 of each type) [4]. The data given in this case, are the petal width, petal length, sepal width, and sepal length of each plant. The type, or class, is one of the following: Iris setosa, Iris versicolor or Iris virginica.

It is often very hard to know if the classifier or learning algorithm is good (or when quantifying; *how good* it is). This leads to the intricate problem of finding out how to properly evaluate algorithms and classifiers. The evaluation is crucial, since it tells us how good a particular algorithm or classifier is for a particular problem, and which of a number of candidate algorithms or classifiers should be selected for a particular problem. It is argued that evaluation is the key to making real progress in application areas like data mining [75] and a more reliable estimation of generalisation quality could result in a more appropriate selection between candidate algorithms and classifiers. One important motivation for conducting the survey is that it could help express the fundamental aspects of the evaluation problem domain using one, preferrably simple and consistent, framework. This is important since the relations between many evaluation methods are unclear due to the use of different terminologies and concepts.

The focus of our analysis lies on the evaluation of classifiers and learning algorithms. Many methods and techniques for evaluation have been brought forth during the last three decades and we argue that there is a need to compile, evaluate and compare these methods in order to be able to further enhance the quality of classifier and learning algorithm performance assessment.

2.1.1 Research Questions

There are a number of questions that need to be answered to get a good picture of the possibilities and drawbacks of current methods for evaluation. The research questions persued in this survey are:

- Which methods exist today for the evaluation of classifiers and learning algorithms, and how can they be categorised?
- What are the weaknesses and strengths of the current evaluation methods?
- How can the different methods be described in a uniform way?
- What type of metric(s) is used by the evaluation methods to measure the performance of classifiers?

2.1.2 Outline

In Section 2.2 a framework used to describe the different methods covered in this survey is presented. Next, in Section 2.3, the featured evaluation methods are described and this is followed by a categorisation of the different methods in Section 2.4. The three last sections include discussions, related work and conclusions.

2.2 Framework

In order to describe the different evaluation approaches consistently, we decided to use a framework of formal definitions. However, no existing framework was fit for all our ideas, thus we had to develop such a framework. Of course, this new framework shares some similarities with existing frameworks or ways to describe supervised learning formally, cf. [45, 58].

Given a particular problem, let I be the set of all possible instances. This set is referred to as the *instance space*. The number of dimensions, n, of this space is equal to the number of attributes, $A = \{a_1, a_2, ..., a_n\}$, defining each instance. Each attribute, a_i, can be assigned a value, $v \in V_i$, where V_i is the set of all possible values for attribute a_i.

$$I = V_1 \times V_2 \times ... \times V_n \quad . \tag{2.1}$$

Instances can be labeled with a category, or class, k, thus dividing the instance space into different groups of instances. Let K be the set of all valid classes of instances from I. A classifier, c, is a mapping, or function, from instances, I, to predicted classes, K [23]:

$$c : I \rightarrow K \quad . \tag{2.2}$$

See Figure 2.1 for a visualisation of two example classifiers. The *classifier space*, C, can now be defined as the set of all possible classifiers, i.e., all possible mappings between I and K. Using the instance and classifier spaces, we now break down algorithm and classifier evaluation into three separate problem domains; 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

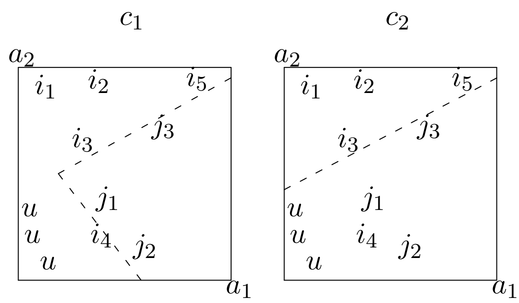

Figure 2.1: *A set of instances from two classes, $K = \{i, j\}$, defined by two attributes, a_1 and a_2, classified by two classifiers, c_1 (left dashed line) and c_2 (right dashed line). The numbered instances are used for training and the instances labeled with u are new, unseen instances which will be classified differently by the two classifiers.*

its relation to C. Last but not least, we formulate the evaluation problems; the evaluation of classifiers and the evaluation of learning algorithms.

2.2.1 The Classification Problem

We now formulate the classification problem; given a non-classified instance, $i \in I$, assign a class, $k \in K$, to i. Given that we have a classifier, c, this can be expressed as:

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

An instance, i, with an assigned class, k, is defined as the *classified instance* i^k. A classified set of instances is a set, T, of pairs, such that each pair consists of an instance, $i \in I$, and an assigned class, $k \in K$, ordered by instance index j:

$$T = \{< i_j, k_j > | i_j \in I, k_j \in K, j = 1 \ldots n\} \quad . \tag{2.4}$$

It should be noted that, in the strict mathematical sense, a set can not contain duplicates, however in practise a classified set used for training can [75]. Noise can also be a factor in real world training sets, e.g., there could exist two identical instances, i_1 and i_2, where $i_1^k \neq i_2^k$.

2.2.2 The Learning Problem

Using the definitions provided above, the learning problem can now be formulated as follows; given a classified set of instances, T, select the most appropriate classifier from C [1]. 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 [45].

Without inductive bias, or *prior assumptions* about the learning target, a learner has no rational basis for classifying new, unseen instances [45]. Inductive bias can be viewed as a rule or method that causes an algorithm to choose one classifier over another [44]. However, averaged over all possible learning tasks, the bias is equally both a positive and a negative force [62].

Inductive biases can be divided into two categories; *representational* and *procedural* (algorithmic). Easily explained, the representational bias defines which parts of C that the learning algorithm considers and the procedural bias determines the order of traversal when searching for a classifier in that subspace [28].

In order to solve a learning problem, a learning algorithm, a, capable of selecting a classifier, c, from a subset of classifier space ($c \in C_a$), must be chosen. The representational bias of a affects which classifiers will be in this subset and the procedural bias of a affects the classifier selection process. This means that the inductive bias of a learning algorithm narrows down C in order to find a suitable classifier in a feasible amount of time.

Related to inductive bias are the *No Free Lunch* (NFL) theorem for supervised learning[2] [77], e.g., stating 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, the algorithms will perform equally well averaged over all possible tasks of learning.

However, in most real world cases, the interest lies in finding out how well one or more algorithms perform on one particular problem or a limited set of

[1] This is sometimes called model selection. This term, originating from statistics, is frequently but also ambigously 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" [39], "a mechanism for [..] selecting a hypothesis among a set of candidate hypotheses based on some pre-specified quality measure" [55]. The different definitions of model selection have been discussed by many researchers, cf. [16].

[2] NFL theorems are also available for other areas, e.g., search [78] and optimisation [79].

problems. Thus, the inductive biases of learning algorithms often do play an important role. The NFL theorems and their implications are explained, in an intuitive way, by Ho [32].

For some learning problems, the existence of a true classifier for I is assumed (a classifier that classifies all instances of I correctly). Under the assumption that such a true classifier (c^*) exists, we say that the learning problem is *realisable* if $c^* \in C_a$ and *unrealisable* if $c^* \notin C_a$ [58]. In some work, an algorithm with a particular configuration, a, for which $c^* \in C_a$, is known as a faithful model and if $c^* \notin C_a$ then a is a non-faithful model, e.g., it is argued that a model is said to be faithful if the learning target can be expressed by the model [66].

2.2.3 The Evaluation Problems

Having formulated the learning and classification problems, we now formulate the evaluation problems. Given an algorithm or a classifier, determine how appropriate it is for a particular problem.

The Classifier Evaluation Problem

The classifier evaluation problem is formulated as follows; given a classifier, $c \in C$, and a classified set of instances, T, assign a value describing the classification performance, v, of classifier c with respect to T. We may also define an algorithm-independent classifer evaluation function, e, that for each $c \in C$ and T assigns a value, v:

$$e(c, T) = v \ . \tag{2.5}$$

Similarly, the algorithm-dependent classifier evaluation problem can be formulated as follows; Given an algorithm, a, a classifier, $c \in C_a$, and a classified set of instances, T, assign a value describing the classification performance, v, of classifier c with respect to T. The algorithm-dependent classifier evaluation function, e^ϵ, is then defined so that, for each c and T it assigns a value, v:

$$e^\epsilon(c, T) = v \ . \tag{2.6}$$

A classifier evaluation function uses one or several metric(s) to measure performance. Equations 2.7 and 2.8 show two common, and closely related metrics for this purpose. We have chosen to call these metrics *accuracy* and *error* respectively but many different names are used in the literature, e.g., success rate and error rate [75].

T is sometimes divided into two subsets; the *training set*, T_s, and the *test set*, T_t. Empirical evaluation methods measure performance by using T_s for training, i.e., generating/selecting a classifier, and T_t for evaluation. Theoretical evaluation methods measure the performance on T_s and use the result in combination with some theoretical estimate of generalisation performance.

If the evaluation is performed using T_s the accuracy and error metrics are sometimes referred to as subset fit [7] and training/resubstitution error [75] respectively. The training error/accuracy metrics are usually poor estimates of the performance on new, unseen instances [75]. Another term, sample error [45] is applied when instances used for evaluation are assumed to be independently and randomly drawn from an unknown probability distribution. If the evaluation is performed using T_t the error metric is sometimes referred to as the *holdout error* [18], or *empirical error*.

The error (or accuracy) is often calculated using a *0-1 loss* function, in which a comparison is made between the predicted class, $c(i)$, and the known, or *correct*, classification, i^k, of the same instance from T:

$$\text{Accuracy:} \quad \beta(c,T) = 1 - \alpha(c,T) \quad \text{where} \tag{2.7}$$

$$\text{Error:} \quad \alpha(c,T) = \frac{1}{|T|}\sum_{i\in T}\delta\left(i^k, c(i)\right) \quad \text{and} \tag{2.8}$$

$$\text{0-1 loss:} \quad \delta\left(i^k, c(i)\right) = \begin{cases} 1 & i^k \neq c(i) \\ 0 & \end{cases} . \tag{2.9}$$

Another metric, risk, is defined as the probability that a classifier, c, misclassifies an instance, i ($P(i^k \neq c(i))$). As explained in Section 2.2.2, some learning (and evaluation) methods assume the existence of c^*. Under this assumption, it is possible to define the *true risk* as the *probability of misclassifying a single randomly drawn instance from I*. Under the same assumption, it is possible to define true error as the fraction of misclassified instances on I. It is argued that, if the error is calculated on a large enough T for a particular confidence level, the true error would lie within its confidence interval [45].

Two key issues concerning the use of accuracy or error calculation on some T to estimate the true error are *estimation bias* and *estimation variance* [45]. If evaluation using accuracy is performed on the same instances that were used for training ($T_s \cap T_t \neq \oslash$), 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 ($T_s \cap T_t = \oslash$). Even if the true error is estimated from an unbiased set of test instances the estimation can still vary from the true error, depending on the size and makeup of the test set. If T_t is small the expected variance will be large [45].

Another relationship between bias and variance has to do with a well-known concept in statistics, the bias-variance trade-off (BV), which concerns the relationship with classifier complexity and performance on new, unseen instances. According to BV, a classifier of high complexity scores well on training data but poorly on test data (low bias, high variance) as opposed to a simple classifier, which can score well on test data but perhaps not as good as a complex classifier would do on training data (high bias, low variance). Relating this to classifiers, it can be argued that a too complex classifier has a tendency to overfit the data. Simply put, there could exist a trade-off between estimation of more parameters (bias reduction) and accurately estimating the parameters (variance reduction) [40]. The parameters of a classifier are used to express the partitioning of I. In contrast, Domingos questions that a complex classifier automatically leads to overfitting and less accuracy and argues that the success of classifier ensembles shows 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 [17]. 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 [61].

The Algorithm Evaluation Problem

The algorithm evaluation problem is formulated as follows:

Given a learning algorithm (with a particular parameter configuration), a, and a classified set, T, assign a value describing the generalisation capability, g, of a with respect to T.

We may define an algorithm evaluation function, e, that for each a and T assigns a generalisation capability value, g:

$$e(a, T) = g \quad . \tag{2.10}$$

Some methods can only evaluate algorithms that generate classifiers of a certain type. We define a classifier-dependent algorithm evaluation function, e^{ϵ}, that for each a and T assigns a generalisation capability value, g:

$$e^{\epsilon}(a, T) = g \quad . \tag{2.11}$$

The generalisation capability of a learner affects how well the classifier it selects is able to predict the classes of new, unseen instances. However, the question of how to best measure generalisation capability is still open.

After having formulated the classification, learning, and evaluation problems, we are now ready to describe the evaluation methods featured in this study.

2.3 Evaluation Methods

The evaluation methods have been divided into four groups, according to which evaluation problem they solve;

- algorithm-dependent classifier evaluation (Definition 2.6).
- algorithm-independent classifier evaluation (Definition 2.5).
- classifier-independent algorithm evaluation (Definition 2.10).
- classifier-dependent algorithm evaluation (Definition 2.11).

2.3.1 Algorithm-dependent Classifier Evaluation

Algorithm-dependent classifier evaluation methods depend on;

- a particular algorithm, or
- an algorithm with a particular configuration.

The Vapnik-Chervonenkis (VC) bound is a classifier evaluation method based on a theoretical measure of algorithm capacity (the VC dimension) and the training error (commonly called the empirical risk) of a classifier selected by that algorithm [12].

The Minimum description length principle (MDL) is a theoretical measure of classifier complexity. Both evaluation methods depend on calculations specific to a particular algorithm or classifier. The VC dimension must be defined and calculated for different algorithms and MDL must be defined for different classifiers.

Vapnik-Chervonenkis Bound

The Vapnik-Chervonenkis (VC) dimension [70, 71], for a particular algorithm a, is a measure of its capacity. It is argued that the VC dimension and the training error of a classifier selected by a can be combined to estimate the error on new, unseen instances [45]. The VC dimension has been defined for different learning algorithms since it depends on the inductive bias, which often differs between algorithms. The VC dimension of an algorithm a, is related to the ability of its selectable classifiers to shatter instances [45], 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 [70].

Relating to our framework we say that the VC dimension is a measure of the generalisation capability, g, of algorithm a and the empirical risk (error calculated on training data) is a measure of the classification capability, v, of classifier $c \in C_a$. The combination, called the VC bound (VCB), is argued to be an upper bound of the expected risk (actual, or true risk), i.e. an estimate of the error on test data, or new, unseen instances [12, 70].

Since calculation of the VC dimension depends on a particular a it has to be explicitly calculated for each a. According to our terminology the VC bound is a measure of both generalisation capability and classification performance:

$$e^{\epsilon}(a, c, T) = e(a, T) + e(c, T) \tag{2.12}$$

A good classifier has high classification performance (low empirical risk/training error) and high generalisation capability (low capacity) [12].

Minimum Description Length Principle

Minimum Description Length Principle (MDL) [56] is an approximate measure of Kolmogorov Complexity [41], which defines simplicity as “the length of the shortest program for a universal Turing machine that correctly reproduces the observerved data” [58]. MDL is based on the insight that any regularity in the data can be used to compress the data, i.e., to describe it using fewer symbols than the number of symbols needed to describe the data literally [75]. MDL is claimed to embody a form of *Ockham’s Razor* [69] as well as protecting against *overfitting*. There exist several tutorials on the theoretical background and practical use of MDL, cf. [29]. It is argued that there is no guarantee that MDL will choose the most accurate classifier [17]. Furthermore, some researchers note that it is hard to use MDL in practise 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 [75]. One practical problem is how to calculate the length or size of the theory of classifiers in order to be able to perform a fair comparison of different theories. As an example, the decision tree learning algorithm ID3 [52] can be said to implement a type of MDL since it tries to create a minimal tree (classifier) that still can express the whole theory (learning target). MDL could be defined as:

$$e^{\epsilon}(c) = \text{sizeof}(c) \tag{2.13}$$

2.3.2 Classifier-independent Algorithm Evaluation

There exist a number of empirical accuracy-based methods for algorithm evaluation, e.g., cross-validation (CV) [65], jackknife (JK) [51] and bootstrap (BS) [20, 35]. JK is a special case of CV [18] and will only be briefly mentioned in that context.

Cross-validation

A CV test is prepared by partitioning T into n number of subsets (T_1, T_2, ..., 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* [75]. 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.

Algorithm 1 describes the general cross-validation evaluation method, in which β is the accuracy function (see Equation 2.7). The algorithm can be used, e.g., with $n = |T|$ for leave-one-out (jackknife) or $n = 10$ for 10-fold cross-validation. If stratification is desired, the $SplitIntoFolds$ function has to be written with this in mind.

Algorithm 1 Cross-validation.

Require: T, a classified set
Require: n, the number of folds
Require: a, a learning algorithm
  $F = SplitIntoFolds(T, n)$
  $v \leftarrow 0$
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      if $j \neq i$ **then**
        $Train(a, F[j])$
      end if
    end for
    $c \leftarrow SelectedClassifier(a)$
    $v \leftarrow v + \beta(c, F[i])$
  end for
  $g \leftarrow v/n$

Bootstrap

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 standard CV, the same instance can be selected/sampled several times). In one particular version, *0.632 bootstrap* [75], 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 [39]. It has been has shown that leave-one-out is almost unbiased, but it has a high estimation variance, leading to unreliable estimates [19]. 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 [8].

2.3.3 Classifier-dependent Algorithm Evaluation

This type of methods depends on a certain hierarchy, or structure, of the classifiers that need to be evaluated, in order to evaluate an algorithm.

Structural Risk Minimisation

Structural Risk Minimisation (SRM) [70] is an algorithm evaluation (and, perhaps more importantly, a classifier selection) method based on the VC dimension (explained in Section 2.3.1). One algorithm, a, may have different C_a depending on the configuration, e.g., the number of neurons per layer for a multi-layer perceptron. Classifiers are organised into nested subsets, such that each subset is a subset of the structure of the next, i.e., the first subset includes the possible classifiers with 2 neurons in 1 layer, the next subset includes the possible classifiers with 3 neurons in 1 layer, etc.

A series of classifiers are trained and one is selected from each subset with the goal of minimising the empirical risk, i.e., the training error. The classifier with the minimal sum of empirical risk and VC confidence (the lowest VC bound) is chosen (as the best classifier). If a too complex classifier is used the VC confidence will be high and the implications of this is that even if the empirical risk is low the number of errors on the test set can still be high, i.e., the problem of overfitting arises. It is argued that overfitting is avoided by using classifiers with low VC dimension, but if a classifier with a too low VC dimension is used it will be difficult to approximate the training data [70]. This is related to the bias-variance trade-off discussed in Section 2.2.2.

The use of SRM for evaluation is limited to algorithms, for which it is possible to create nested subsets, as explained above. A tutorial on Support Vector Machines (a learner that uses SRM for classifier selection), which also covers VC dimension and the SRM process is given by Burges [12].

2.3.4 Algorithm-independent Classifier Evaluation

The following methods do not evaluate any classifier or algorithm properties. Only the performance of the particular classifier to be evaluated is used, thus the evaluation is carried out in the same manner regardless of classifier or algorithm.

Table 2.1: *Example evaluation using SE and CV. A neural network, consisting of* 1 *layer of* k *neurons (trained with the back-propagation algorithm [57] for* 500 *epochs), is evaluated using SE (on training data) and CV on a data set including* 150 *instances and a discrete target. As the error is calculated instead of accuracy for both evaluation methods, the lowest result is always the best. SE is more optimistic than CV for* $k = \{3, 5, 7\}$. *Even the upper bound of a 95% confidence interval (*C^+*) is more optimistic than CV in two cases out of four.*

k	M	SE	C^+	CV
1	150	0.0333	0.0621	0.0333
3	150	0.0133	0.0317	0.0267
5	150	0.0133	0.0317	0.0333
7	150	0.0133	0.0317	0.0400

Sample Error

Sample error (SE) is the most basic algorithm-independent method for estimating the performance of a learned classifier. It has already been explained in Section 2.2. An example evaluation using cross-validation and sample error is shown in Table 2.1.

Receiver Operating Characteristics Analysis

Receiver Operating Characteristics (ROC) analysis was first used in signal detection theory [21] and later introduced to the machine learning community [49]. It is a graphical technique that can be used to evaluate classifiers. 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

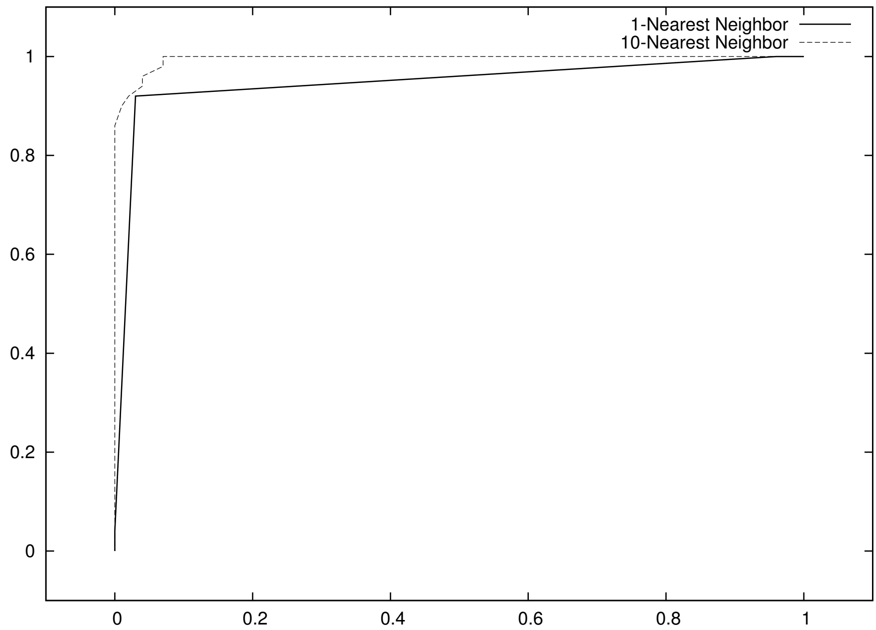

Figure 2.2: *ROC plots of two K-nearest neighbor classifiers on the Iris data set, for* $k = 1$ *and* $k = 10$ *respectively.*

that they belong to a particular class[3]. Most classifiers can only provide information about the predicted class of an instance, not the probability that an instance belongs to a particular class. The ROC curve is then plotted by starting from origo, reading from the ranking list (starting at the highest ranked 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 horisontal axis shows the percentage of false positives [75]. See figures 2.2 and 2.3 for examples of ROC plots.

[3]It is necessary to calculate the probability of an instance belonging to a particular class. We say that, given a non-classified instance, $i \in I$, and a particular class, $k \in K$, assign the probability that i belongs to k, e.g., $c(i, k) = P(c(i) = k)$.

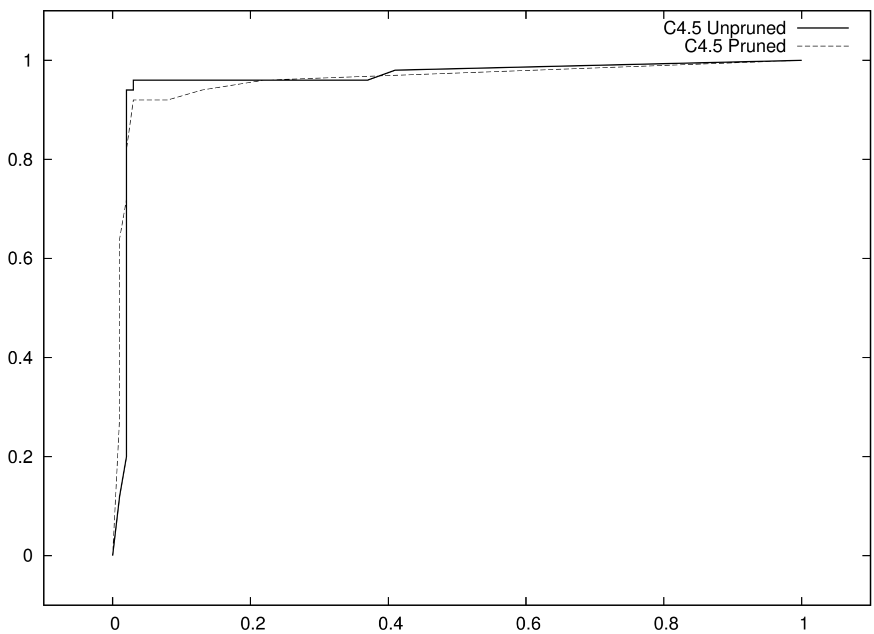

Figure 2.3: *ROC plots of one pruned and one unpruned C4.5 decision tree classifier on the Iris data set.*

Area Under the ROC Curve

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. [30, 74]. As with ROC plots, the standard AUC can only be used for problems where $|K| = 2$, however it has been generalised to work for problems where $|K| \geq 2$ [30].

Comparisons have been made between ROC/AUC and other evaluation methods [49, 50]. In one such study [50] it was argued that accuracy-based evaluation of machine learning algorithms cannot be justified, since accuracy maximisation assumes that the class distribution is known for I but often for benchmark data sets it is 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. Furthermore, it has been argued that the additional features of ROC make up for the added setup and usage time (in comparison to single-number methods) [50]. There exist several introductory texts on ROC analysis, cf. [75] for a brief explanation or [23] for a tutorial on the usage of ROC analysis for machine learning research, including algorithms for plotting ROC curves and calculating AUC.

Measure-based Evaluation

Some studies make use of the geometric/measurable qualities of C and I. One such study proposes geometric strategies for detecting overfitting and performing robust classifier selection using simple metric-based intuitions [63].

The concept of measure functions for generalisation performance has been suggested, providing an alternative way of selecting and evaluating classifiers, and it allows for the definition of learning problems as computational problems. Measure-based evaluation provides a way to evaluate certain qualitative properties of classifiers, unlike most commonly used methods, e.g., CV, which only performs a quantitative evaluation by means of accuracy calculation [6, 7].

An example measure function (MF) based on the theory of measure-based evaluation has been suggested [7] and it is this example that is featured in the categorisation of evaluation methods in this study (see Section 2.4). Three criteria are combined into one numeric function. *Subset fit* is accuracy calculated on the training set, *similarity* is calculated by averaging the distance from each instance to its closest decision border, and *simplicity* is calculated by measuring the total length of the decision borders. Each criteria can be 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 2.14 describes the example measure function. a_0, a_1 and a_2 are the weights of subset fit, i.e., accuracy (see Equation 2.7), similarity (*simi*) and simplicity (*simp*) respectively. k_1 and k_2 are the weights of the similarity measure for correctly classified instances (*simi*$_1$) and incorrectly classified instances (*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 *simp* is subtracted from the subset fit and similarity measures.

$$\mathrm{mf} = a_0\beta(c,T) + a_1\left(k_1\mathrm{simi}_1 + k_2\mathrm{simi}_2\right) - a_2\mathrm{simp}\ . \qquad (2.14)$$

2.4 Categorisation

In order to get an overview of the evaluation methods they have been categorised in a more structured manner by the introduction of a number of selected properties.

2.4.1 Properties

- **Type:** Empirical (E), Theoretical (T).

  If training and testing are performed on the same data, the evaluation is theoretical, otherwise it is empirical.

- **Target:** Algorithms (A), Classifiers (C),
  Classifiers from a particular classifier space (C_n),
  Algorithms, whose classifier spaces can be arranged into nested subsets (A_n)

- **Metric(s):** Accuracy/Error/Subset fit (A), Complexity/Simplicity (C), Cost (M), Similarity (S).

2.4.2 Discussion

The categorisation is presented in Table 2.2. CV (as well as the special case, JK), and BS share many similarities and have been proven to be mathematically related. CV has been categorised as a resampling method based on the fact that it is a numerical procedure to assess the loss of a classifier and that it uses data for both training and testing in order to determine the true behavior [55], hence BS and JK are also qualified to belong to the resampling method group since they include these properties too. CV and BS differ in the way they create training and test sets. SRM and VCB are strongly related since SRM uses VCB. ROC and AUC are also strongly related since AUC is the area under the ROC curve. Like SE, both AUC and ROC can be calculated using either training or test data. Measure-based evaluation is a general concept and MF refers to the example measure function as described by Andersson et al. [7].

Table 2.2: *Categorisation of evaluation methods.*

Method	Type	Target	Metric
Sample Error (SE)	T,E	C	A
Cross-validation/Jackknife (CV/JK)	E	A	A
Bootstrap (BS)	E	A	A
Measure-based Evaluation (MF)	T	C	A,C,S
Structural Risk Minimisation (SRM)	T	A_n	A,C
Vapnik-Chervonenkis Bound (VCB)	T	C_n	A,C
ROC Plots (ROC)	T,E	A,C	M
Area Under the ROC Curve (AUC)	T,E	A,C	M
Minimum Description Length Principle (MDL)	T	C_n	C

2.5 Related Work

The STATLOG project [37] involved evaluating a large number of algorithms on 20 data sets using three objectively measurable criteria; accuracy, misclassification cost, and time consumed to produce results. However, the focus was on the comparison of learning algorithms rather than evaluation methods.

Meta-learning focuses on the exploitation of knowledge about learning that enables understanding, and improvement of performance, of learning algorithms [26]. Performance assessment is a component of many meta-learning methods, e.g., self-adaptive learners (cf. [72]).

Much research has been carried out on information-theoretic criteria for evaluation, e.g., AIC [3], BIC [54, 64], SIC [66] and NIC [46] (several comparisons exist, cf. [67]). However, these criteria are used to evaluate regression functions rather than classifiers, i.e., they are used when the learning target is continuous.

A number of studies focus on the statistical validity of experimental comparisons of classifiers, cf. [60, 16], however they seldom discuss the methods used, e.g., cross-validation, but rather the experimental setup.

2.6 Conclusions

This analysis has covered the evaluation methods currently used for algorithms and classifiers within the area of supervised learning. Each method has been analysed, described and categorised according to type (theoretical or empirical), target (algorithms, classifiers, etc.), and metric(s) (accuracy/error, complexity, cost, and similarity). Several comparisons have been made between similar methods. One contribution of this work is that it features a formal framework that has been defined in order to describe the different evaluation methods using the same terminology. Since the methods originate from different research fields and thus are explained using different terminologies, this was an important step towards making the differences and similarities between the methods explicit.

One conclusion is that accuracy is the most widely used metric and that many evaluation methods use accuracy as the only metric for measuring classifier performance. Accuracy can be measured either on training or testing data and studies have shown earlier that accuracy alone is not a good estimate of future classifier performance if it is measured on the training data (the estimate will be optimistically biased). Even if accuracy is measured on testing data (not used for training) it can be a bad estimate of future performance since it assumes that the class distribution is known.

Another conclusion is that most methods focus on either cost or accuracy (e.g., ROC, AUC, SE, CV, BS, and JK), while others have a strong focus towards the complexity of the solution (e.g., MDL, AIC, BIC, SIC, and NIC). Measure-based evaluation puts the decision of what to focus on the user since it uses adjustable weighted metrics. For instance, one measure-based example function, MF, uses similarity, simplicity, and subset fit as weighted metrics. This way the evaluation is tailored to a specific learning problem. However, other methods that combine metrics exist, e.g., SRM and VCB which combine the accuracy metric with a measure of complexity (the size of the classifier space available to a specific learning algorithm configuration).

Some methods can be used for any classifier or algorithm (e.g., CV, BS, JK, MF, SE), while others depend on the representation of classifiers or have to be explicitly defined for different algorithms (e.g., ROC, AUC, MDL, VCB, and SRM).

Finally, it is concluded that the formal framework makes it easy to describe the learning, classification, and evaluation problems, as well as explaining key concepts of learning theory like inductive bias, bias and variance, and the no free lunch theorems.

Acknowledgements

The data sets used for example evaluations have been downloaded from *UCI Machine learning repository* [47].

CHAPTER
THREE

Paper II

A Multi-dimensional Measure Function
for Classifier Performance

Niklas Lavesson, Paul Davidsson
Published at the 2nd IEEE conference on Intelligent Systems
Varna, Bulgaria, June 2004

3.1 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 [45]. 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 [1]. Sometimes the wrong evaluation method for the problem at hand is chosen and sometimes 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 [65], bootstrap [20] and holdout [18]. There are some alternative evaluation methods that take into account more than just accuracy, e.g., ROC plots and lift charts [75]. Moreover, Bebis defined a fitness function consisting of one generalisation term and one network size term for use with a genetic algorithm when optimising neural networks [9]. 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 [5, 7]. 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 3.4 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.

3.2 Measure-based Evaluation

According to [7] a measure function assigns a value describing how good the classifier is at generalising from the data set, for each possible combination of data set and classifier. They also argue that most popular learning algorithms try to maximise 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 [7] builds upon all three heuristics. There is often a trade-off between the three heuristics and the measure function reveals this trade-off. The measure

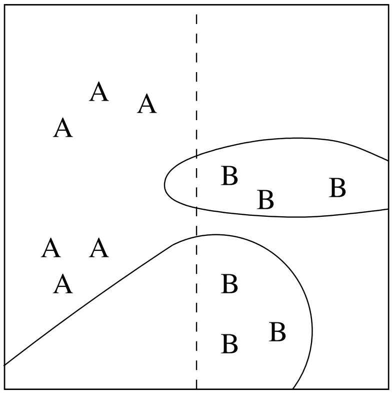

Figure 3.1: *Two-dimensional decision spaces of two classifiers, one illustrated by the dashed line, and the other illustrated by the curves. The letters indicate the positions of instances from two categories (A and B).*

function helps in analysing the learned classifier and the partitioning of the decision space. For example, consider the two classifiers illustrated in Figure 3.1. 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 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 classifiy 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 [7] one often used heuristic of similarity is that the decision borders should be centred between clusters of instances belonging to different classes (e.g., Support Vector Machines try to accomplish this by finding the maximum margin hyperplane, cf. [75]). 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. [69]). According to [7] 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, of the classifiers.

3.3 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 10-fold, n-fold, and bootstrap CV [75].

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 stands for the number of instances in the data set, is sometimes referred to as jackknife [18] or leave-one-out [75]. 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 generalisation 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 10-fold CV introduces a source of correlation since one uses examples for training in one trial and for testing in another [43].

3.4 A Multi-dimensional Measure Function

We will now present an implementation of the example measure function suggested by [7], 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 [75], 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 3.4). From now on we refer to the implementation of the measure function as MDMF (Multi-Dimensional Measure Function) and to the Iris data set [4], used for all experiments and examples in this paper, as IRIS.

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

3.4.1 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 3.2. 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 3.1 and Equation 3.2.

$$1 - \frac{1}{2^{bd}} \ . \tag{3.1}$$

$$\frac{1}{2^{bd}} - 1 \ . \tag{3.2}$$

The normalisation 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 3.1 and Equation 3.2 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. Algorithms 3 and 4 describe the calculation of similiarity in detail. For pseudo code on a higher abstraction level, see Algorithm 2. We will now explain the basic functionality of the algorithms used for the calculation of similarity. Each instance, x, is classified, p, and the result 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 selected 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

Algorithm 2 MDMF Similarity calculation.

```
for all instances do
  MoveToInstancePos()
  SDist ← 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 ∗ 2/10 do
        MoveInExcludedDimension(pos)
        if NewPrediction != OldPrediction then
          CalculateDistance()
          Break()
        end if
      end for
    end for
  end while
end for
```

Algorithm 3 MDMF Similarity calculation.

Require: I (A set of instances)
Require: A (The set of attributes of I)
Require: c (A classifier)

```
for all x ∈ I do
  p ← PredictedClass(x, c)
  a ← ActualClass(x, I)
  if a = p then
    Correct ← true
  else
    p ← a
    Correct ← false
  end if
  SDist ← 0
  while a = p do
    SDist ← SDist + RadiusIncrease
    if SDist > 1 then
      break
    end if
    step ← (SDist * 2)/10
    for mainAttr = 0 to |A| do
      for z = 0 to Power(2, |A| − 1) do
        Search()
      end for
    end for
  end while
  if Correct then
    ContributePosValue(dist)
  else
    ContributeNegValue(dist)
  end if
end for
```

Algorithm 4 Similarity search function.

```
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
```

differs from the classification 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 Euclidian 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 3.1 or Equation 3.2. 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 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.

IRIS has four numerical 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 150 individual searches through all four dimensions to calculate the similarity aspect of MDMF. The number of search steps is currently set to 10 but this can be changed very easily, e.g., a higher value would result in a more detailed but slower search.

3.4.2 Simplicity

It has been suggested that simplicity can be measured by calculating the total size of the decision borders [7]. 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. The following way to approximate the decision border size has been proposed [7]: 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

Table 3.1: *MDMF configuration.*

Weight	Affects	Value
a_0	Subset fit	1.0
a_1	Similarity	1.0
a_2	Simplicity	0.1
k_1	Simi_1	0.5
k_2	Simi_2	0.5
Detail	Affects	Value
square size	Simplicity	0.01
radius size	Similarity	0.0005

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.

3.4.3 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.

3.4.4 A Weighted Measure Function

MDMF was implemented in Java using the WEKA machine learning workbench [75]. WEKA contains a collection of machine learning algorithms as well as tools for classification, visualisation, 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.

$$\text{MDMF} = a_0\text{Subsetfit} + a_1\left(k_1\text{Simi}_1 + k_2\text{Simi}_2\right) - a_3\text{Simp}\ . \tag{3.3}$$

It is now possible to describe the measure function with Equation 3.3. Except for the tuning of metric weights, the level of detail for the simplicity and similarity calculations can alsobe 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.

Table 3.2: *Comparison between 10CV and MDMF.*

Algorithm	10CV	MDMF
J48	0.953	1.086
J48-P	0.947	1.101

Table 3.3: *Measure-based evaluation of C4.5.*

Algorithm	Subset	Similarity	Simp	MDMF
J48	0.980	0.756	2.724	1.086
J48-P	0.967	0.720	2.253	1.101

The MDMF configuration used in the example and the experiments of this article is detailed in Table 3.1. 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 [53], 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 3.2. 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 3.3. 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 visualisations of the decision spaces of these two decision trees can be seen in Figure 3.2. 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 excluded by setting their corre-

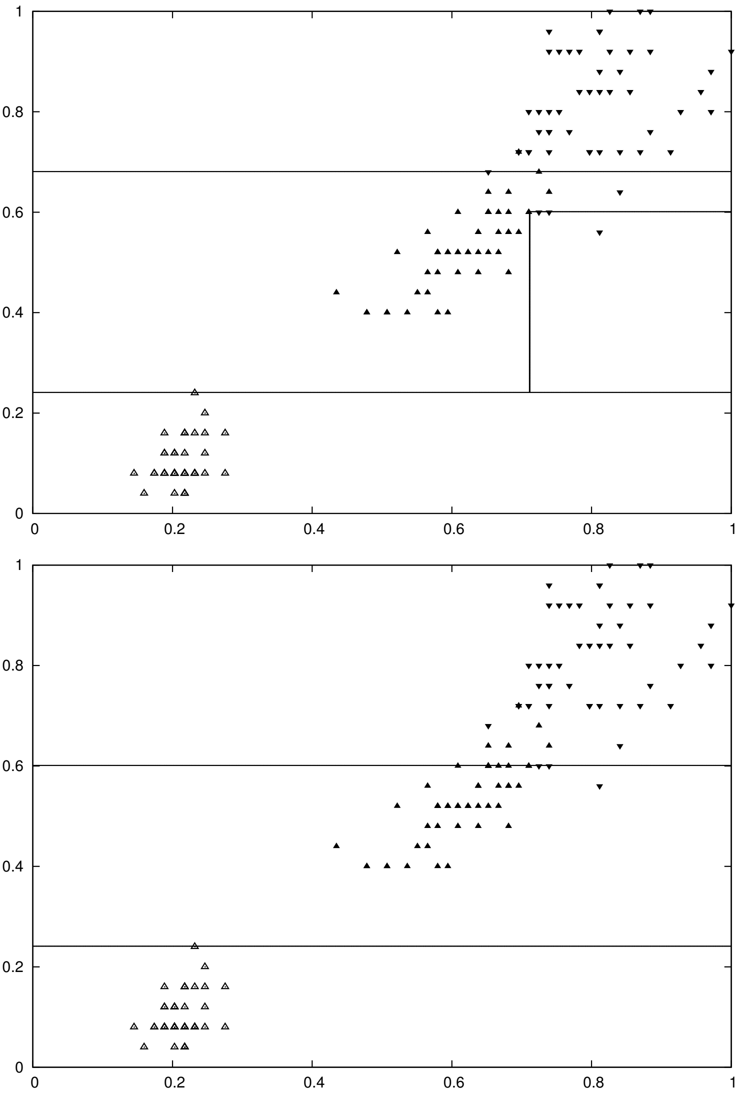

Figure 3.2: *Visualisations of two-dimensional decision spaces. The J48 (top) and J48-P (bottom) classifiers have been produced by the J48 algorithm.*

Table 3.4: *Measure-based evaluation of 10 classifiers.*

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

sponding weight to zero. The default configuration, discussed in Section 3.4, is chosen so that each part contributes about equally much to the evaluation result.

3.5 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 3.4 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

[1]The name of this class has been been changed to *MultilayerPerceptron* in WEKA after the publication of this paper.

been used to produce five different neural network classifiers (denoted BPx-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 comparably with decision tree and neural network classifiers in some domains [45]. 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 3.2 shows this difference visually (for two selected dimensions). The same pattern applies to the nearest neighbor classifiers. Figure 3.3 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 3.4 and comparing it to the decision spaces of the other classifiers.

When analysing the neural network classifier results we can see that the simplest classifier is the one generated with 3 nodes and a training time of 20000 epochs. When comparing with another neural network classifier (with the same size but trained for 32000 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 *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 0.99 when calculated from the two different time measure columns presented in Table 3.5. 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

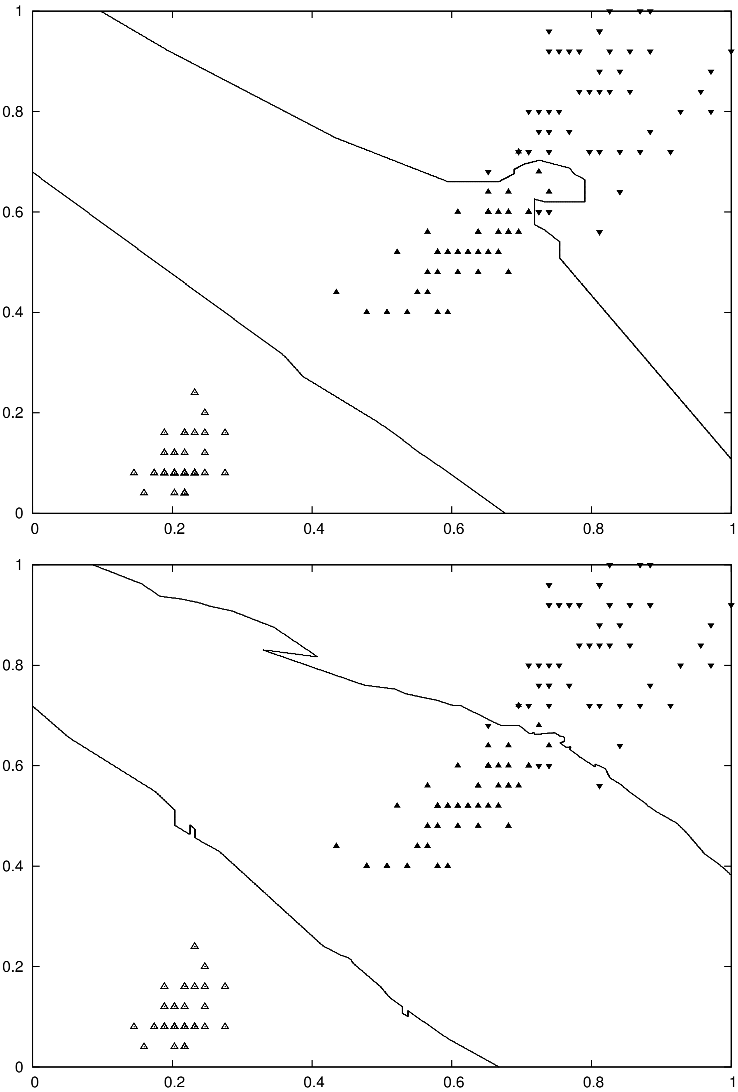

Figure 3.3: *Visualisation of a two-dimensional decision space. The classifiers IB1 (top) and IB10 (bottom) have been generated by the IBk algorithm.*

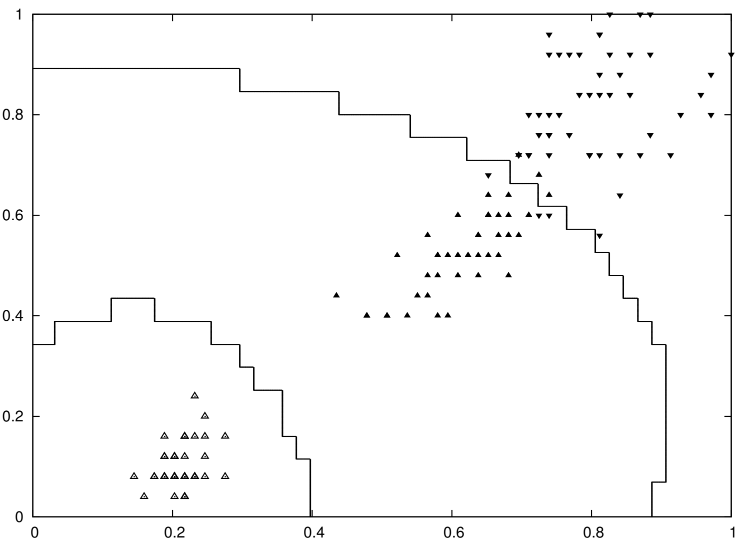

Figure 3.4: *Visualisation of a two-dimensional decision space. The NBayes classifier have been generated by the NaiveBayes algorithm.*

Table 3.5: *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.*

Algorithm	10 grid	100 grid
J48	19 ms	353 ms
IB1	1738 ms	25905 ms
IB10	2103 ms	27501 ms
BP2	57 ms	850 ms
BP3	67 ms	943 ms
NBayes	83 ms	1148 ms

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 10-fold CV, for which 10 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 affect 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.

3.6 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 3.2 and a practical experiment in Section 3.4, 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 normalised data sets with numerical 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 2). 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 optimisation methods such as genetic algorithms [27, 33] and simulated annealing [38] to produce classifiers optimised for a specific class of problems. Simple approaches of this type has been carried out using accuracy measures as fitness functions or thresholds [13, 31]. We plan to develop a public web site 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 realtime.

CHAPTER

FOUR

Paper III

Quantifying the Impact of Learning Algorithm Parameter Tuning

Niklas Lavesson, Paul Davidsson

Submitted for publication, February 2006

4.1 Introduction

It has been argued that many of the well-known learning algorithms often generate classifiers of comparable quality [45, 73, 75]. However, it is not always made clear what assumptions are made in such comparisons. For instance, the type or method of evaluation is not mentioned nor which configurations of the included algorithms are used and how these were determined.

Within the application domains of induction algorithms, the goals may vary significantly. For some problems, a robust learning algorithm may be needed, capable of generating a reasonably good classifier with minimum parameter tuning. For other problems it could be more important to choose a learning algorithm capable of producing 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 on classifier performance of the choice of algorithm compared to the configuration of algorithms. 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 [62, 77]. 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 algorithm 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.

4.2 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 practise. 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, but seldom precisely defined. In particular, the relation to the possible configurations of the algorithm is unclear. Characteristic for this attribute, however, is that a high value is almost always desired. The average performance metric and its estimate are defined as:

$$p_1(C_a, D) = \frac{\sum_{x \in C_a} f(x, D)}{|C_a|} \tag{4.1}$$

$$\hat{p}_1 = p_1(Q_a, D) = \frac{\sum_{x \in Q_a} f(x, D)}{|Q_a|} \quad . \tag{4.2}$$

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

$$p_2 = max_{x \in C_a} f(x, D) \tag{4.3}$$

$$\hat{p}_2 = max_{x \in Q_a} f(x, D) \quad . \tag{4.4}$$

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 statistical variance 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:

$$s_1(C_a, D) = \frac{\sum_{x \in C_a} \left(f\left(x, D\right) - p_1\left(C_a, D\right)\right)^2}{|C_a|} \tag{4.5}$$

$$\hat{s}_1 = s_1\left(Q_a, D\right) = \frac{\sum_{x \in Q_a} \left(f\left(x, D\right) - \hat{p}_1\left(Q_a, D\right)\right)^2}{|Q_a| - 1} \quad . \tag{4.6}$$

The metric s_2 uses the range, i.e., the difference between the highest and lowest score, to capture the degree of impact. We label s_2 the performance span and define this metric and its estimate as:

$$s_2 = p_2 - min_{x \in C_a} f(x, D) \tag{4.7}$$

$$\hat{s}_2 = \hat{p}_2 - min_{x \in Q_a} f(x, D) \quad . \tag{4.8}$$

4.3 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 [75], which includes implementations of all needed algorithms. Some algorithms covered in this study include additional parameters that are not present in the WEKA implementations.

4.3.1 Featured Algorithms

The four algorithms; *Back-Propagation* (BP) [57], *K-Nearest Neighbor* (KNN) [2], *Support Vector Machines* (SVM) [15], and *C4.5* [53] were selected for this study. The motivation is that 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. [45, 58].

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 (called MultilayerPerceptron in WEKA) and many researchers have suggested configuration guidelines for different problems [9].

A subset of the BP algorithm parameters and the network structure were selected for this particular study; the number of epochs, hidden layers and neurons in each hidden layer. The justification for choosing these three parameters is that they all are relative to the data set on which the algorithm operates; hence they should be optimised 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 [45, 75]. The WEKA default value for training time is 500 epochs and the number of hidden neurons is defined by n:

$$n = (\alpha + \beta)/2 \ , \tag{4.9}$$

where α is the number of classes and β is the number of attributes for a particular data set. Table 4.1 describes the configurations of BP used in the experiment. The epochs and layers parameters both have static intervals whereas 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 α=2 and β=3 then n=2 which would make the lower limit n-6=2-6=-4 if it was not restricted.

Table 4.1: *BP parameter configurations.*

Parameter	Setting(s)
Training epochs	[500...27500] step size: 3000
Hidden neurons	[n-6...n+8] step size: 2
Hidden layers	[1...5] step size: 1

Regarding the K-Nearest Neighbor algorithm (IBk in WEKA), the best parameter value of k can be difficult to know beforehand and a number of methods for finding an optimal value, given a specific data set, have been proposed. The WEKA default value of k is 1. The KNN algorithm can also be configured to weight instances differently and two techniques for weighting included in the WEKA implementation are inverse-distance weighting (IDW) and similarity weighting (SW). Equal weighting (no weighting) is the default technique. IDW lets the weights be high at zero-distance from a new instance and decreases the weights as the distance increases. The configurations included in the experiments are made up of combinations of different number of neighbors and weighting techniques. The KNN parameters are shown in Table 4.2.

The C4.5 algorithm induces decision tree classifiers and an implementation, called J48, is included WEKA. J48 can be used to generate both pruned and unpruned trees. 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

Table 4.2: *KNN parameter configurations.*

Parameter	Setting(s)
Neighbors	[1...40] step size: 1
Weighting	{equal, inverse-distance, similarity}

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 summarised as follows. Pruning

Table 4.3: *C4.5 parameter configurations.*

Parameter	Setting(s)
Confidence threshold	[0.02...0.5] step size: 0.02
Minimum instances per leaf	[1...4] step size: 1
Folds for pruning	[2...5] step size: 1
Pruning	{no, yes}
Reduced-error pruning	{no, yes}
Subtree raising	{no, yes}

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 4.3.

For Support Vector Machines, 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 [34]. The SVM parameters are shown in Table 4.4.

Table 4.4: *SVM parameter configurations.*

Parameter	Setting(s)
Complexity	[0.1...1.0] step size: 0.1
Kernel function	{polynomial, radial basis}
Gamma	[0.005...0.060] step size: 0.005
Lower order terms	{true, false}
Exponent	[0.5...2.5] step size: 0.5

4.3.2 Procedure

Eight data sets from *UCI machine learning repository* were used in the experiments [47]. The number of instances (I), classes and attributes (A) of each data set are shown in Table 4.5. 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

Table 4.5: *Data set overview.*

Data set	Instances (I)	Classes	Attributes (A)	$I * A$
Breast-cancer	286	2	9	2574
Contact-lenses	24	3	4	96
Iris	150	3	4	600
Labor	57	2	16	912
Lymph	148	4	19	2812
Soybean	683	19	35	23905
Weather	14	2	4	56
Zoo	101	7	18	1818

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 performance 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 guarantees a non-stratified sample [75].

4.4 Results

The classification performance and sensitivity of each algorithm for each data set, measured according to the four metrics, are presented in Table 4.6. In addition, figures 4.1 to 4.8 contain box plots 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 quartile values.

We see that both the highest mean, $\hat{p}_1$, (Table 4.6) and the highest median (figures 4.1 to 4.8) 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 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. The correlation is quite low, for instance the correlation coefficient between $\hat{p}_1$ and $\hat{s}_1$ is 0.01. In fact, $\hat{p}_1$ and $\hat{s}_2$ are slightly negatively correlated (-0.21), which suggests that a robust algorithm has high average performance. The two classification performance metrics are positively correlated (0.73), indicating that a learning algorithm with a good average performance also tends to have a high best performance. However, they are not perfectly correlated which suggests that they measure different aspects of classification performance. There is also a similarly large positive correlation between the two sensitivity metrics (0.71). 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 five 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 the majority of the data sets, followed by C4.5 and SVM.

In six out of eight data sets, the performance span ($\hat{s}_2$) 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 could indicate that BP and SVM are harder to tune. In 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 no correlation between sensitivity and performance.

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.

4.5 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 as well as 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 less sensitive 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 level of impact that different parameters have on classifier performance. For one, 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. Secondly, there is a degree of randomness connected

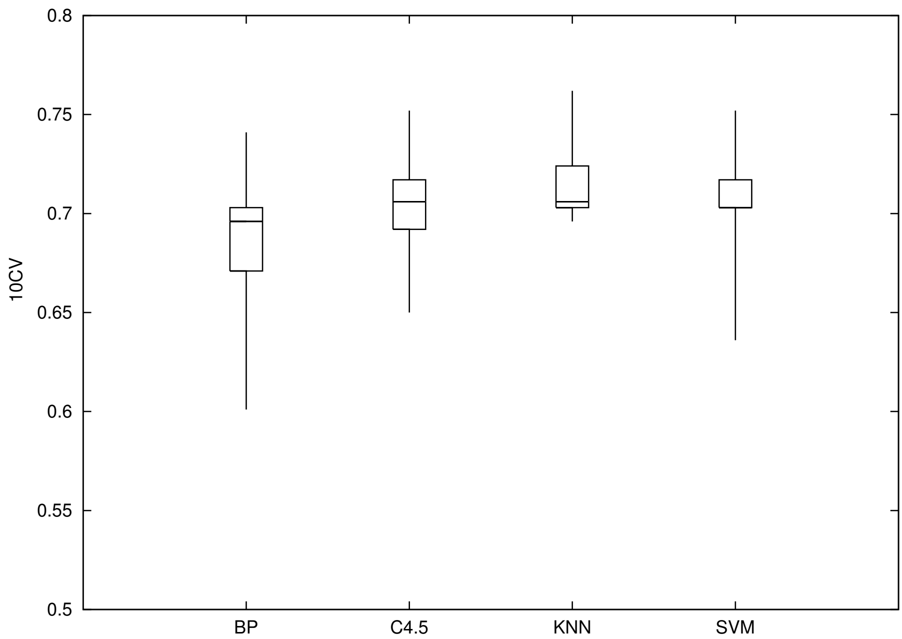

Figure 4.1: *Box plots of the breast-cancer data set.*

to the evaluation. For example, the weights of a neural network are initialised 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 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

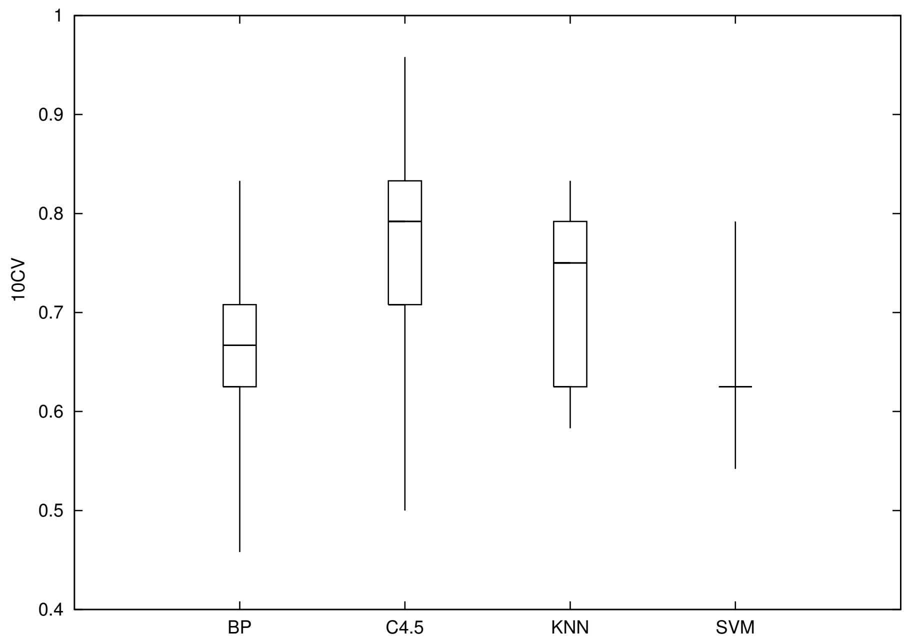

Figure 4.2: *Box plots of the contact-lenses data set.*

would be to apply the *Analytic Hierarchy Process* (AHP) [59], 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.

4.6 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

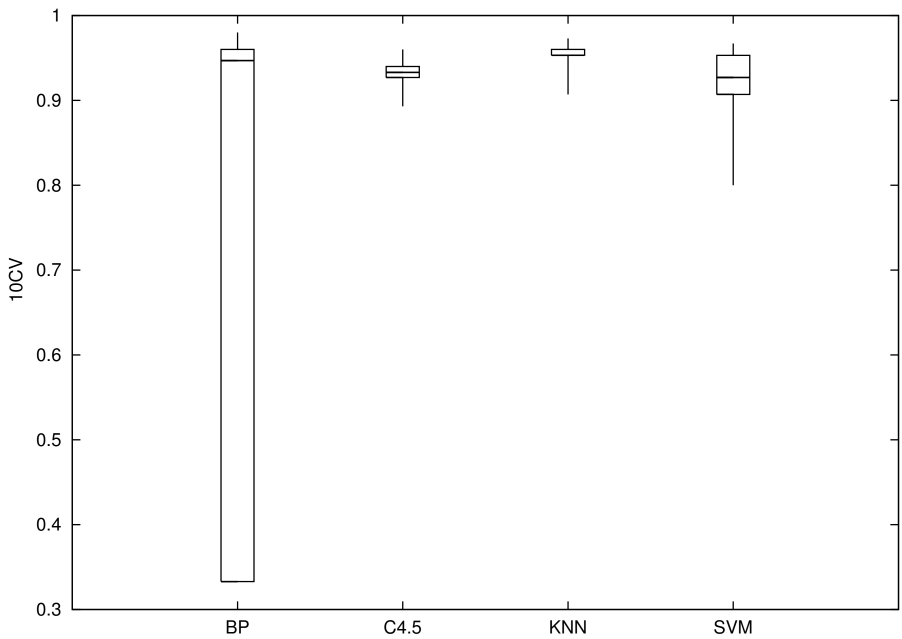

Figure 4.3: *Box plots of the iris data set.*

to analyse and interpret prior experiments and data set characteristics in order to improve the quality of learning [25]. Many researchers have recognised the fact that algorithm optimisation 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. cascade-correlation for automatically creating and training neural networks [22]. Other enhancing methods, such as *bagging*, *boosting* and *stacking*, combine several classifiers to increase performance, cf. [11, 24, 75, 76]. 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 instance, classifiers can be evaluated using ROC curves [50], measure functions [7] or holdout [18] while algorithms are commonly evaluated using cross-

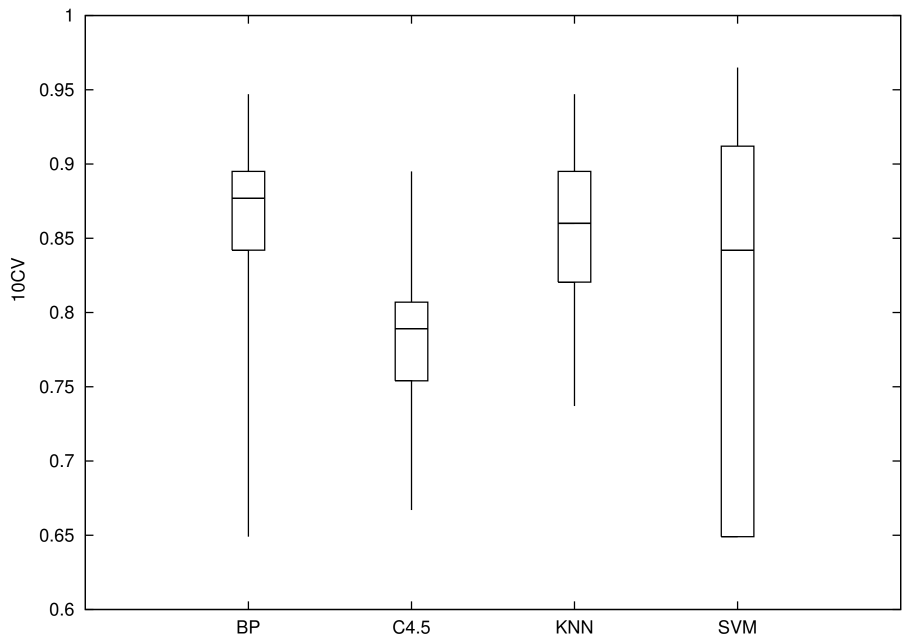

Figure 4.4: *Box plots of the labor data set.*

validation [65] or bootstrap [20]. However, these methods only evaluate a particular parameter configuration of an algorithm, whereas the quality attributes presented in this work are used to make a more general evaluation of the algorithm.

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 [10]. 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.

4.7 Conclusions and Future Work

We have studied the impact of learning algorithm optimisation by means of parameter tuning and argue that some algorithms are more robust to parameter

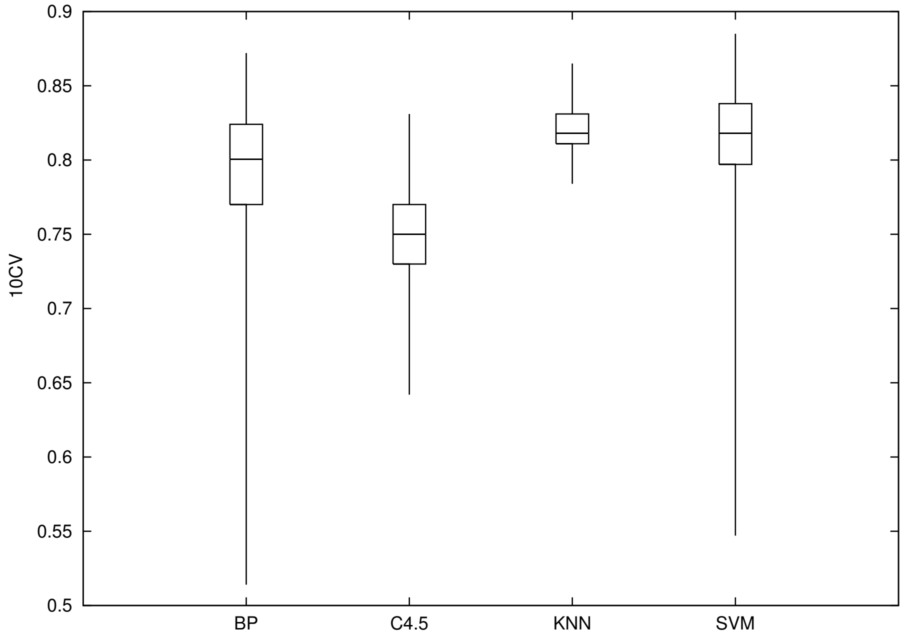

Figure 4.5: *Box plots of the lymph data set.*

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 show that these metrics complete each other by revealing different aspects of sensitivity and classification 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

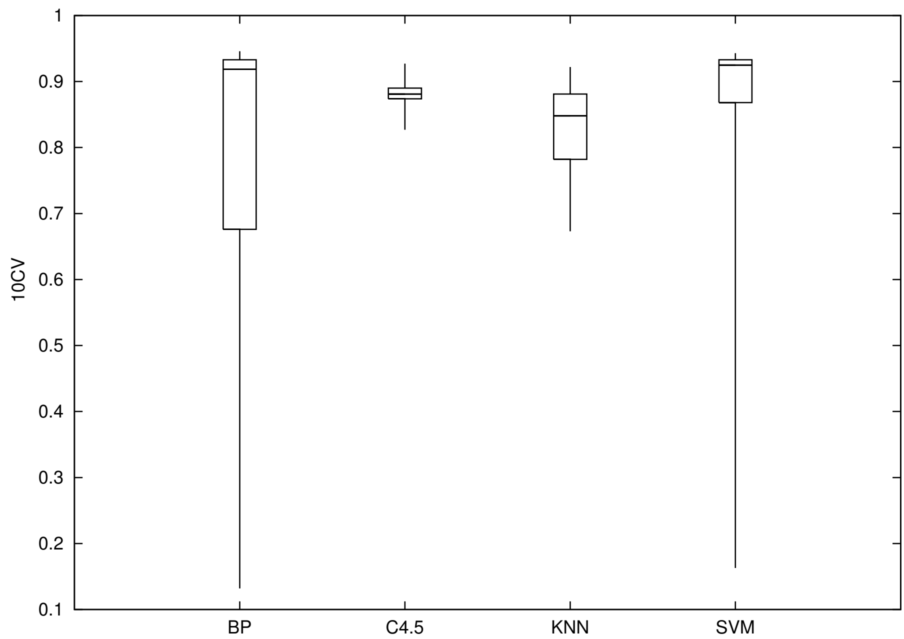

Figure 4.6: *Box plots of the soybean data set.*

others. Moreover, the results suggest that no trade-off exists between sensitivity and classification performace.

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 optimisation 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.

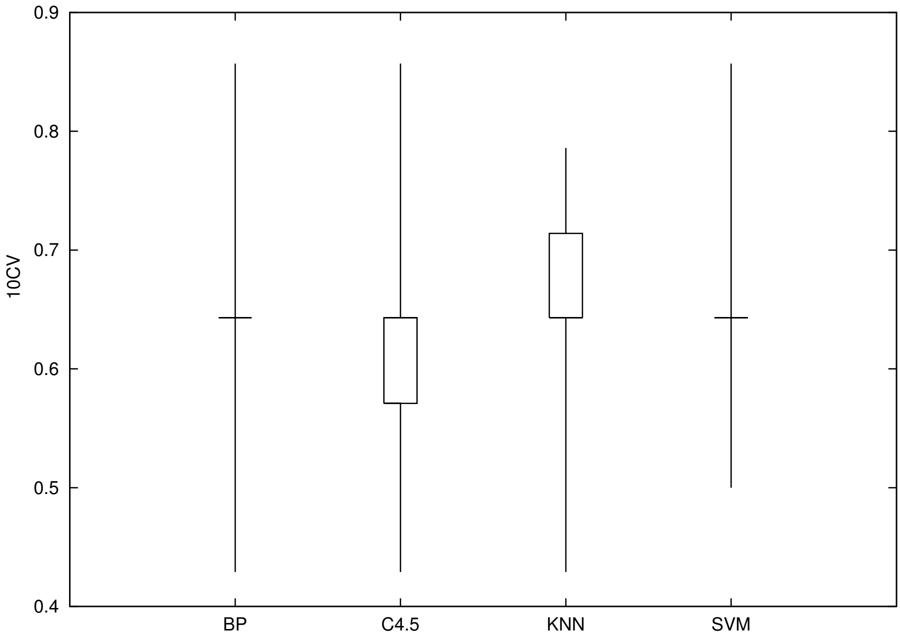

Figure 4.7: *Box plots of the weather data set.*

Table 4.6: *Experiment results.*

Algorithm	Data set	Performance		Sensitivity	
		$\hat{p}_1$	$\hat{p}_2$	$\hat{s}_1$	$\hat{s}_2$
SVM	Breast-cancer	0.71	0.75	0.00039	0.12
	Contact-lenses	0.64	0.79	0.00166	0.29
	Iris	0.92	0.97	0.00109	0.17
	Labor	0.81	0.98	0.01479	0.33
	Lymph	0.78	0.89	0.00975	0,34
	Soybean	0.86	0.94	0.01922	0.78
	Weather	0.63	0.86	0.00402	0.43
	Zoo	0.78	0.97	0.03469	0,56
C4.5	Breast-cancer	0.70	0.75	0.00030	0.10
	Contact-lenses	0.76	0.92	0.00760	0.42
	Iris	0.94	0.96	0.00012	0.07
	Labor	0.78	0.91	0.00209	0.26
	Lymph	0.75	0.83	0.00080	0.19
	Soybean	0.88	0.93	0.00019	0.10
	Weather	0.61	0.93	0.00262	0.57
	Zoo	0.89	0.96	0.00060	0.14
BP	Breast-cancer	0.69	0.74	0.00058	0.14
	Contact-Lenses	0.67	0.83	0.00293	0.29
	Iris	0.78	0.98	0.07609	0.65
	Labor	0.83	0.95	0.01068	0.30
	Lymph	0.75	0.87	0.01267	0.36
	Soybean	0.81	0.95	0.02549	0.81
	Weather	0.64	0.79	0.00421	0.36
	Zoo	0.76	0.97	0.04521	0.57
KNN	Breast-cancer	0.71	0.76	0.00022	0.07
	Contact-lenses	0.64	0.83	0.00289	0.25
	Iris	0.96	0.97	0.00004	0.03
	Labor	0.81	0.97	0.00791	0.32
	Lymph	0.82	0.87	0.00024	0.08
	Soybean	0.83	0.92	0.00418	0.25
	Weather	0.64	0.79	0.00253	0.36
	Zoo	0.92	0.96	0.00057	0.08

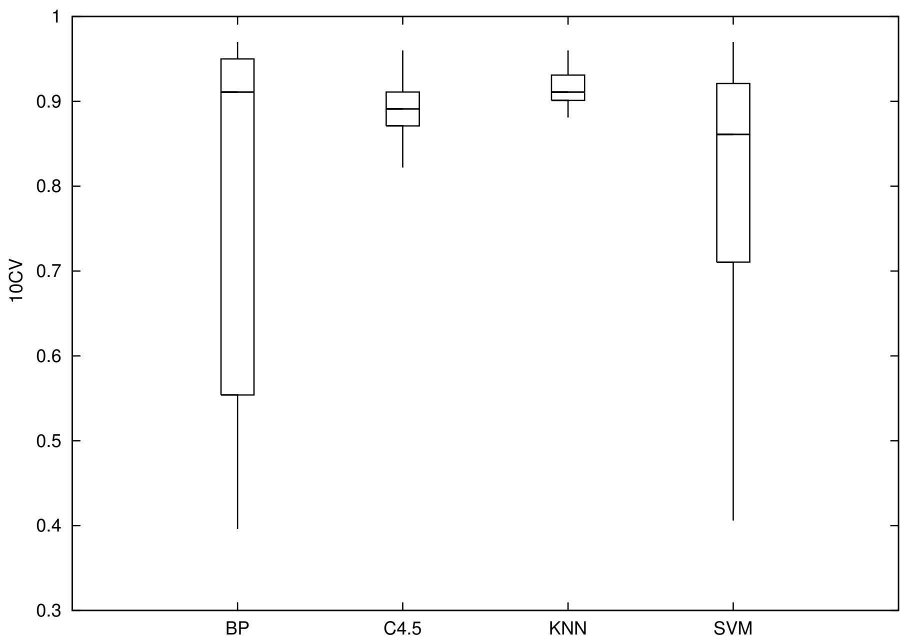

Figure 4.8: *Box plots of the zoo data set.*

BIBLIOGRAPHY

[1] N. M. Adams and D. J. Hand. Improving the Practice of Classifier Performance Assessment. *Neural Computation*, 12(2):305–312, 2000.

[2] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based Learning Algorithms. *Machine Learning*, 6:37–66, 1991.

[3] H. Akaike. Information Theory and an Extension of the Maximum Likelihood Principle. In B. N. Petrov and F. Caski, editors, *2nd International Symposium on Information Theory*, Budapest, 1973.

[4] E. Anderson. The Irises of the Gaspé Peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

[5] A. Andersson, P. Davidsson, and J. Lindén. Measuring Generalization Quality. Technical Report LU-CS-TR 98-202, Department of Computer Science, Lund University, Lund, Sweden, 1998.

[6] A. Andersson, P. Davidsson, and J. Lindén. Model Selection Using Measure Functions. In *10th European Conference on Machine Learning 1998 Workshop on Upgrading Learning to the Meta-Level: Model Selection and Data Transformation*, pages 54–65, Chemnitz, Germany, 1998.

[7] A. Andersson, P. Davidsson, and J. Lindén. Measure-based Classifier Performance Evaluation. *Pattern Recognition Letters*, 20(11-13):1165–1173, 1999.

[8] T. L. Bailey and C. Elkan. Estimating the Accuracy of Learned Concepts. In *13th International Joint Conference on Artificial Intelligence*, pages 895–901, Chambéry, France, 1993.

[9] G. Bebis and M. Georgiopoulos. Improving Generalization by Using Genetic Algorithms to Determine the Neural Network Size. In *Southcon 95*, pages 392–397, Fort Lauderdale, FL, USA, 1995.

[10] O. Bousquet and A. Elisseeff. Stability and Generalization. *Machine Learning Research*, 2:499–526, 2002.

[11] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[12] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[13] S. Chalup and F. Maire. A Study on Hill Climbing Algorithms for Neural Network Training. In *1999 Congress on Evolutionary Computation*, volume 3, pages 1999–2021, Washington, DC, USA, 1999.

[14] H. Chen, H. Ye, C. Lv, and H. Su. Application of Support Vector Machine Learning to Leak Detection and Location in Pipelines. In *21st IEEE Conference on Instrumentation and Measurement Technology*, volume 3, pages 2273–2277. IEEE, 2004.

[15] C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[16] T. G. Dietterich. Approximate Statistical Test For Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[17] P. Domingos. The Role of Occam's Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.

[18] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, USA, 2nd edition, 2000.

[19] B. Efron. Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation. *American Statistical Association*, 78(382):316–330, 1983.

[20] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Monographs on Statistics & Applied Probability. Chapman & Hall, 1993.

[21] J. P. Egan. *Signal Detection Theory and ROC Analysis*. Cognition and Perception. Academic Press, London, 1975.

[22] S. E. Fahlman and C. Lebiere. The Cascade-Correlation Learning Architecture. *Advances in Neural Information Processing Systems*, 2:524–532, 1990.

[23] T. Fawcett. ROC Graphs – Notes and Practical Considerations for Data Mining Researchers. Technical Report HPL-2003-4, Intelligent Enterprise Technologies Laboratories, HP Laboratories, Palo Alto, 2003.

[24] Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. In *13th International Conference on Machine Learning*, pages 148–156, 1996.

[25] C. Giraud-Carrier and J. Keller. *Dealing With the Data Flood: Mining Data, Text and Multimedia*, chapter Meta Learning. STT/Beweton, The Hague, Netherlands, 2002.

[26] C. Giraud-Carrier, R. Vilalta, and P. Brazdil. Introduction to the Special Issue on Meta-learning. *Machine Learning*, pages 187–193, 2004.

[27] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, USA, 1989.

[28] D. F. Gordon and M. Desjardins. Evaluation and Selection of Biases in Machine Learning. *Machine Learning*, 20:5–22, 1995.

[29] P. Grünwald. A Tutorial Introduction to the Minimum Description Length Principle. In P. Grünwald, I. J. Myung, and M. Pitt, editors, *Advances in Minimum Description Length–Theory and Applications*. MIT Press, 2005.

[30] D. J. Hand and R. J. Till. A Simple Generalisation of the Area under the ROC Curve for Multiple Class Classification Problems. *Machine Learning*, 45:171–186, 2001.

[31] S.-Y. Ho, C.-C. Liu, and S. Liu. Design of An Optimal Nearest Neighbor Classifier Using an Intelligent Genetic Algorithm. *Pattern Recognition Letters*, 23(13):1495–1503, 2002.

[32] Y. C. Ho and D. L. Pepyne. Simple Explanation of the No Free Lunch Theorem and Its Implications. *Optimization Theory and Applications*, 115(3):549–570, 2002.

[33] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[34] T. Howley and M. G. Madden. The Genetic Evolution of Kernels for Support Vector Machine Classifiers. In *15th Irish Conference on Artificial Intelligence*, 2004.

[35] A. K. Jain, R. C. Dubes, and C.-C. Chen. Bootstrap Techniques for Error Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):628–623, 1987.

[36] M. Jeraj, S. Džeroski, L. Todorovski, and M. Debeljak. Application of Machine Learning Methods to Palaeoecological Data. *Ecological Modelling*, 191(1):159–169, 2006.

[37] R. D. King, C. Feng, and A. Sutherland. STATLOG: Comparison of Classification Algorithms on Large Real-world Problems. *Applied Artificial Intelligence*, 9(3):259–287, 1995.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[39] R. Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *14th International Conference on Artificial Intelligence*, pages 1137–1145, Montréal, Québec, Canada, 1995. Morgan Kaufmann.

[40] R. Kohavi and G. H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[41] A. N. Kolmogorov. Three Approaches to the Quantitative Definition of Information. *Problems in Information Transmission*, 1(1):1–7, 1965.

[42] E. M. Kussul and L. M. Kasatkina. Neural Network System for Continuous Hand-written Words Recognition. In *International Joint Conference on Neural Networks*, volume 4, pages 2855–2858. IEEE, 1999.

[43] M. A. Maloof. On Machine Learning, ROC Analysis, and Statistical Tests of Significance. In *16th International Conference on Pattern Recognition*, volume 2, pages 204–207, Los Alamitos, CA, USA, 2002. IEEE.

[44] T. M. Mitchell. The Need for Biases in Learning Generalizations. *Readings in Machine Learning*, pages 184–191, 1980.

[45] T. M. Mitchell. *Machine Learning*. Computer Science Series. McGraw-Hill, Singapore, international edition, 1997.

[46] N. Murata, S. Yoshizawa, and S. Amari. Network Information Criterion – Determining the Number of Hidden Units for an Artificial Neural Network Model. *IEEE Transactions on Neural Networks*, 5(6):865–872, 1994.

[47] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI Repository of Machine Learning Databases, 1998.

[48] D. A. Pomerleau. ALVINN: An Autonomous Land Vehicle in a Neural Network. Technical Report CMU-CS-89-107, Carnegie Mellon University, Pittsburgh, PA, USA, 1989.

[49] F. Provost and T. Fawcett. Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions. In *3rd International Conference on Knowledge Discovery and Data Mining*, Huntington Beach, CA, USA, 1997. AAAI Press.

[50] F. Provost, T. Fawcett, and R. Kohavi. The Case Against Accuracy Estimation for Comparing Induction Algorithms. In *15th International Conference on Machine Learning*, pages 445–453, Madison, WI, USA, 1998. Morgan Kaufmann.

[51] M. H. Quenouille. Notes on Bias in Estimation. *Biometrika*, 61:353–360, 1956.

[52] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[53] J. R. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, 1993.

[54] A. E. Raftery. Choosing Models for Cross-classifications. *American Sociological Review*, 51:145–146, 1986.

[55] J. Ratsaby, R. Meir, and V. Maiorov. Towards Robust Model Selection Using Estimation and Approximation Error Bounds. In *Computational Learing Theory*, pages 57–67, 1996.

[56] J. Rissanen. Modeling by Shortest Data Description. *Automatica*, 14:465–471, 1978.

[57] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. *Parallel Distributed Processing*, 1:318–362, 1986.

[58] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2nd edition, 2003.

[59] T. L. Saaty and L. G. Vargas. *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Kluwer Academic Publisher, 2001.

[60] S. L. Salzberg. On Comparing Classifiers: A Critique of Current Research and Methods. *Data Mining and Knowledge Discovery*, 1:1–12, 1999.

[61] C. Schaffer. Overfitting Avoidance as Bias. *Machine Learning*, 10(2):153–178, 1993.

[62] C. Schaffer. A Conservation Law for Generalization Performance. In *11th International Conference on Machine Learning*, pages 259–265. Morgan Kaufmann, 1994.

[63] D. Schuurmans. A New Metric-Based Approach to Model Selection. In *14th National Conference on Artificial Intelligence*, Providence, RI, USA, 1997.

[64] G. Schwartz. Estimating the Dimension of a Model. *Annals of Statistics*, 6(2):461–464, 1978.

[65] M. Stone. Cross-validatory Choice and Assessement of Statistical Predictions. *Royal Statistical Society*, 36:111–147, 1974.

[66] M. Sugiyama and H. Ogawa. Subspace Information Criterion for Model Selection. *Neural Computation*, 13(8):1863–1890, 2001.

[67] M. Sugiyama and H. Ogawa. Theoretical and Experimental Evaluation of the Subspace Information Criterion. *Machine Learning*, 48:25–50, 2002.

[68] Y. Tan and G. Zhang. The Application of Machine Learning Algorithm in Underwriting Process. In *International Conference on Machine Learning and Cybernetics*, pages 3523–3527. IEEE, 2005.

[69] S. C. Tornay. *Ockham: Studies and Selections*. Open Court Publishing Company, La Salle, IL, USA, 1938.

[70] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY, USA, 1995.

[71] V. Vapnik and A. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and Its Applications*, 16:264–280, 1971.

[72] R. Vilalta and Y. Drissi. A Perspective View and Survey of Meta Learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[73] W. Weideman, M. Manry, Y. Hung-Chun, and G. Wei. Comparisons of a Neural Network and a Nearest-neighbor Classifier via the Numeric Handprint Recognition Problem. *IEEE Transactions on Neural Networks*, 6(6):1524–1530, 1995.

[74] G. M. Weiss and F. Provost. The Effect of Class Distribution on Classifier Learning: an Empirical Study. Technical Report ML-TR-44, Department of Computer Science, Rutgers University, 2001.

[75] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.

[76] D. H. Wolpert. Stacked Generalization. *Neural Networks*, 5:241–259, 1992.

[77] D. H. Wolpert. The Supervised Learning No Free Lunch Theorems. Technical report, NASA Ames Research Center, Moffett Field, CA, USA, 2001.

[78] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Search. Technical Report SFI-TR 95-02-010, Santa Fe Institute, Santa Fe NM, USA, 1995.

[79] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

ABSTRACT

The fundamental question studied in this thesis is how to evaluate and analyse supervised learning algorithms and classifiers. As a first step, we analyse current evaluation methods. Each method is described and categorised according to a number of properties. One conclusion of the analysis is that performance is often only measured in terms of accuracy, e.g., through cross-validation tests. However, some researchers have questioned the validity of using accuracy as the only performance metric. Also, the number of instances available for evaluation is usually very limited. In order to deal with these issues, measure functions have been suggested as a promising approach. However, a limitation of current measure functions is that they can only handle two-dimensional instance spaces. We present the design and implementation of a generalised multi-dimensional measure function and demonstrate its use through a set of experiments. The results indicate that there are cases for which measure functions may be able to capture aspects of performance that cannot be captured by cross-validation tests. Finally, we investigate the impact of learning algorithm parameter tuning. To accomplish this, we first define two quality attributes (sensitivity and classification performance) as well as two metrics for measuring each of the attributes. Using these metrics, a systematic comparison is made between four learning algorithms on eight data sets. The results indicate that parameter tuning is often more important than the choice of algorithm. Moreover, quantitative support is provided to the assertion that some algorithms are more robust than others with respect to parameter configuration. To sum up, the contributions of this thesis include; the definition and application of a formal framework which enables comparison and deeper understanding of evaluation methods from different fields of research, a survey of current evaluation methods, the implementation and analysis of a multi-dimensional measure function and the definition and analysis of quality attributes used to investigate the impact of learning algorithm parameter tuning.

ISSN 1650-2140
ISBN 91-7295-083-8

2006:04