

Master Thesis
Software Engineering
Thesis no: MSE-2003:18
June 2003



Evaluation of classifier performance and the impact of learning algorithm parameters

Niklas Lavesson

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:

Author:

Niklas Lavesson

Niklas.Lavesson@bth.se

University advisor:

Paul Davidsson

Paul.Davidsson@bth.se

Department of Software Engineering and Computer Science

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ABSTRACT

Much research has been done in the fields of classifier performance evaluation and optimization. This work summarizes this research and tries to answer the question if algorithm parameter tuning has more impact on performance than the choice of algorithm. An alternative way of evaluation; a measure function is also demonstrated. This type of evaluation is compared with one of the most accepted methods; the cross-validation test. Experiments, described in this work, show that parameter tuning often has more impact on performance than the actual choice of algorithm and that the measure function could be a complement or an alternative to the standard cross-validation tests.

Keywords: classifier performance, evaluation, optimization

CONTENTS

ABSTRACT	I
CONTENTS	II
ACKNOWLEDGEMENTS	IV
1 INTRODUCTION	1
1.1 MACHINE LEARNING AND CLASSIFIERS	1
1.2 THE CONCEPT OF CLASSIFIER PERFORMANCE	2
1.3 CLASSIFIER COMPARISON.....	3
1.4 GOALS AND AIMS	4
1.5 STRUCTURE OF THE REPORT	4
2 BACKGROUND	5
2.1 CLASSIFIER LEARNING ALGORITHMS.....	5
2.1.1 <i>Back propagation</i>	5
2.1.2 <i>Naïve Bayes</i>	6
2.1.3 <i>C4.5</i>	6
2.1.4 <i>K-Nearest Neighbor</i>	7
2.2 EVALUATION OF CLASSIFIER PERFORMANCE.....	7
2.2.1 <i>Cross-validation</i>	7
2.2.2 <i>ROC curves</i>	8
2.2.3 <i>Measure functions</i>	9
2.2.4 <i>Classifier comparison</i>	10
2.3 METHODS FOR OPTIMIZING PARAMETERS	10
2.3.1 <i>Genetic algorithms</i>	10
2.3.2 <i>Hill climbing</i>	11
2.3.3 <i>Simulated annealing</i>	11
2.4 CONCLUSION	12
2.5 SUMMARY.....	12
3 APPROACH	13
3.1 THE USE OF A MEASURE FUNCTION	13
3.1.1 <i>Similarity evaluation issues</i>	13
3.1.2 <i>Simplicity evaluation issues</i>	13
3.1.3 <i>Approximation for similarity</i>	14
3.1.4 <i>Approximation for simplicity</i>	15
3.2 OPTIMIZING CLASSIFIER PERFORMANCE.....	15
3.2.1 <i>Parameter space sampling</i>	15
3.2.2 <i>Heuristic optimization algorithms</i>	18
3.3 SUMMARY.....	20

4	EXPERIMENTS	21
4.1	OVERALL GOALS OF THE EXPERIMENTS	21
4.1.1	<i>Measure function demonstration</i>	21
4.1.2	<i>Optimization test</i>	21
4.1.3	<i>The impact of parameters</i>	21
4.2	SETUP	22
4.3	CONDUCTION AND RESULTS.....	22
4.3.1	<i>Measure function experiments</i>	22
4.3.2	<i>Optimization</i>	26
4.3.3	<i>Parameter space sampling</i>	27
4.4	SUMMARY.....	30
5	CONCLUSION	31
5.1	THE IMPORTANCE OF METHOD CHOICE.....	31
5.2	THE IMPORTANCE OF PARAMETER TUNING.....	31
5.3	DIFFERENT WAYS OF EVALUATION AND OPTIMIZATION	32
5.4	FUTURE WORK	32
	REFERENCES	33
	FIGURES AND TABLES	35
	MEASURE FUNCTION IMPLEMENTATION.....	36

ACKNOWLEDGEMENTS

I want to say thank you to all of you that have read my work and helped me with useful suggestions. Thanks to Fredrik Wernstedt for introducing me to machine learning and to my supervisor Paul Davidsson for his useful comments and suggestions.

1 INTRODUCTION

This chapter begins by introducing the reader to the field of machine learning and especially the topics concerning classifiers and classifier performance. This is followed by a discussion about classifier comparison and the problems related to this subject. It ends with a presentation of the goals of this report. The machine learning and classifiers overview serves the single purpose of introducing a reader, new to the subject, to the basics of machine learning and classification.

1.1 Machine learning and classifiers

The area of machine learning constitutes a number of paradigms and algorithms for classification and learning, each having its objectives, goals, weaknesses and strengths. An important type of algorithms are those that learn a classifier from examples. A simple and general explanation of these algorithms is that they are used to learn from data to be able to classify instances of data into different categories (classes). Although many of the algorithms are very different in constitution they all have a common interface; they are often configurable and they produce a classifier based on a set of training data.

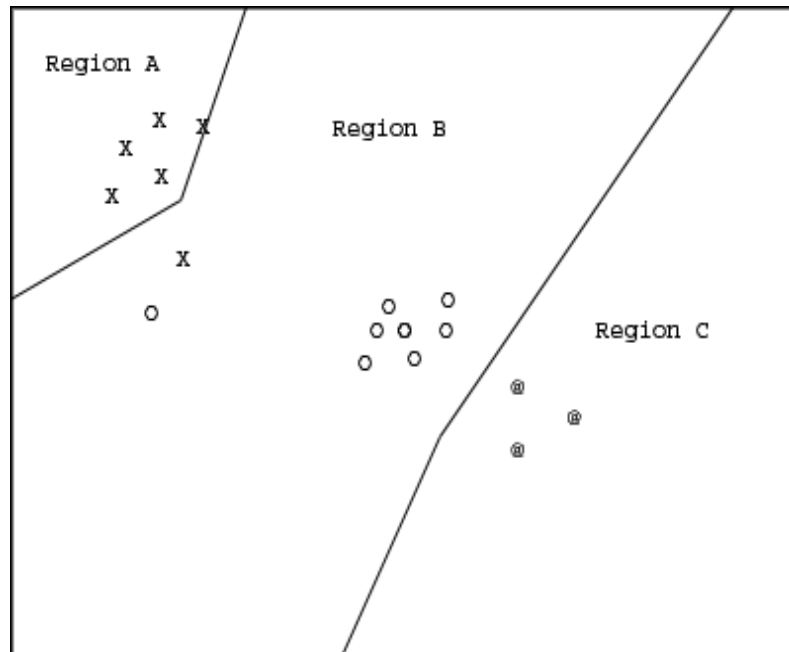


Figure 1. A dataset consisting of 17 instances divided into three different classes has been learned by an algorithm. This is a geometric visualization of the decision borders of a classifier. One of the X instances is classified wrongly as belonging to the B class.

A classifier is built by letting a learning algorithm generalize from a set of data (often referred to as training data). The training data consists of a number of instances. Instances are described by a set of attributes. Thus, a particular instance is described by a set of attribute values.

There exist several types of values and classification types. Attributes can consist of numerical values, Boolean values or other types of values. One of the attributes is often referred to as the target attribute. The target attribute corresponds to the class of the instance. In other words, a classifier should be able to predict the value of the target attribute of an instance, given the values of some or all of the other attributes of the instance. This is true for one type of classification. Other types include concept classification (where the target attribute is a Boolean value; yes/no or true/false) and numerical prediction (where the value of the target attribute is computed from the values of the other parameters).

Figure 1 shows a visualization of the decision borders of a classifier in two dimensions. Think of this picture as a graph in which the two axes represent two numerical attributes e.g. size and weight, contained in a specific database. The X, O and @ symbols represent three different classes and the decision regions of the classifiers are called A, B and C. If the classifier is 100% accurate on the training data all X should lie in the A region, all O should lie in the B region and all @ should lie in the C region. If we were to classify a new, unknown instance of data we would have to know its coordinates (in other words its attribute values). Then we would just place the instance in graph at the correct coordinates and check what region it belongs to.

Sepal length	Sepal width	Petal length	Petal width	Class
5.1	3.5	1.4	0.2	Iris Setosa
4.9	3.0	1.4	0.2	Iris Setosa
4.9	2.4	3.3	1.0	Iris Versicolor
5.9	3.0	5.1	1.8	Iris Virginica

Table 1. Excerpt from Iris database (Anderson, 1935)

Table 1 shows an excerpt from a widely used dataset: the Iris database. The database consists of 150 instances of which four are featured here. There are four attributes or features: sepal length, sepal width, petal length and petal width. All the attributes are numerical. There are three classes of Iris: Setosa, Versicolor and Virginica. One interesting feature (from the learning point of view) of this database is that one of the classes is not linearly separable from the other. What this means is that if you were to lay out the different instances in a four dimensional space, you would not be able to find a plane that separates this class from the other two. In practice, this means that the learning algorithms must be relatively sophisticated in order to be able to learn this database and classify new instances accurately.

1.2 The concept of classifier performance

The performance of a classifier can be measured or estimated in a number of different ways. Which method to use is still subject to research and depends on the type of classification and type of data to be classified. Some of the most common are statistical methods and cross-validation methods. The important question is which classifier attributes to take into account when evaluating the performance of a classifier. One of the most usual attributes to look at is the accuracy of the classification. The accuracy of a classifier is measured by performing a number of classifications and dividing the number of correctly classified instances by the total number of instances. Accuracy can be measured by letting the classifier go through the training data and classify all instances. While the accuracy over the training data can be very high it is often very misleading. Just because the classifier performs well on the training data does not mean that it will perform as well on unknown, new data. In fact, the accuracy measured over the training data is always very optimistic (Mitchell, 1997) and furthermore it does not help in evaluating the generalization capabilities of an algorithm.

Usually the classifier is trained on a set of training data and the accuracy is calculated over a set of test data instead. Other attributes that can be taken into account are computational efficiency and generalization capability. The computational efficiency concerns CPU and memory usage as well as time consumed for training and classification. The generalization capability of a classifier corresponds to how well it will classify unknown instances.

It is often the case that an algorithm has a set of parameters which can be tuned in order to get the best classifier performance given a specific class of problems. However, the optimal values of the parameters of many algorithms, given a specific class of problems, are not always known.

1.3 Classifier comparison

One way to find a good solution for a classifier learning problem is to compare the performance of different classifiers on the same data. A simple comparison could be made by training a number of classifiers on the same data set and comparing their accuracy over the trained data and the time consumed during the training process of each classifier. However, as mentioned above these easy ways of comparison are often not suitable for real world problems. The time consumed during training of a classifier can be interesting but this measure forces the comparison to a certain platform. In this work we will adopt the position taken by Andersson et al. (1999) that there should be a strive to combine a number of performance attributes, biasing towards problem-related areas, into a measure function for classifier performance. Each classifier should then be measured with this function and the results should be compared in order to determine which classifier to use for solving a specific problem.

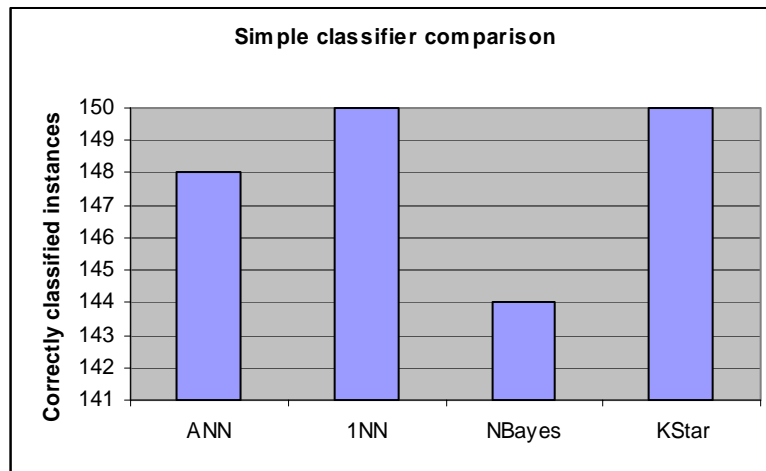


Figure 2. Example of a performance (measured as the number of correctly classified instances) comparison between classifiers learned by four algorithms; Back propagation, 1-Nearest Neighbor, Naïve Bayes and KStar.

In figure 2 four classifiers are compared; a back propagated neural network (ANN), a nearest neighbor classifier (1NN), a Naïve Bayes classifier and a KStar classifier.

1.4 Goals and aims

The goal of this work is to answer these questions:

- Are measure functions feasible to use in practice?
- Which is more important, the choice of learning algorithm or the configuration of an algorithm?
- Which optimization technique works best for tuning a learning algorithm?

In order to try to answer the first question, a measure function was implemented during the writing of this work. This implementation is based on an example from (Andersson et al., 1999). The benefits and possible drawbacks of the measure function are illustrated with experiments and discussed in the conclusion of this work.

Two known optimization algorithms are also implemented; a genetic algorithm and a standard hill climbing algorithm. A third alternative, parameter space sampling, is proposed. These three techniques are used in experiments to determine which one is most suitable for optimizing learning algorithms.

By investigating the impact that tuning or optimization has on classifier performance and comparing the performance of different classifiers on the same learning data we try to determine which is more important, the choice of algorithm or the configuration of an algorithm.

1.5 Structure of the report

Chapter 2 starts with a background on the topics of learning algorithms, classifier performance and optimization. The different learning algorithms used in the experiments are described and so are the most widely used performance measures and optimization techniques.

Chapter 3 describes the use of a measure function for evaluating classifier performance and describes the implementations of this measure function and three optimization algorithms.

These four implementations are all used in the experiments found in chapter 4. The results of the experiments are discussed in the end of chapter 4 and the last chapter contains the conclusions drawn after performing the experiments.

2 BACKGROUND

In this chapter earlier work on classifiers is presented. This chapter puts our work into context and also provides a brief summary of the different techniques and methods existing today. First the learning algorithms that were used during the experiments on classifier performance is presented.

2.1 Classifier learning algorithms

There are many algorithms available today and there exist many versions of each algorithm. In order to conduct an interesting experiment regarding classifier performance, it would seem appropriate to choose a set of learning algorithms that represent the most used and well known algorithms. The strive have been towards finding algorithms that are different from each other in structure and performance.

2.1.1 Back propagation

Back propagation is an algorithm used for training artificial neural networks. Back propagated neural networks is one of the most well known and oldest learning techniques (Mitchell, 1997). Neural networks consist of a network of neurons. Each neuron is able to both receive stimulus from a number of other neurons and send a reaction to a number of neurons. A neuron can receive different information from each neuron sending but it can only send one type of information at a time. The different inputs of the neuron are weighted and if the sum of all active inputs is higher than a selected threshold, the neuron will output an active signal.

The neural network has a number of inputs and outputs. In a learning context, the inputs represent the instance attribute values and the outputs represent the prediction or target attribute. The network is trained by altering the weights of the inputs to each neuron and this can be accomplished in a number of ways. The most common technique is to use the back propagation algorithm. The network is fed with attribute values of an instance and gives an output. The back propagation algorithm checks if the given output is the same as the known output for that particular instance. If this is not the case a correction is back propagated through the net beginning from the output neurons.

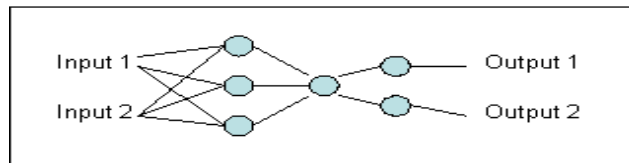


Figure 3. An example of a simple feed forward network with 1 hidden node, 2 inputs and 2 outputs.

One important part of back propagation and neural networks are the many parameters which can be tuned in order to optimize classifier performance. The network can be altered in a number of ways. The number of neurons and the way the different neurons are interconnected are two examples. Back propagation parameters include momentum and learning rate. All these parameters affect the way the network is trained and, possibly, the performance of the learned classifier.

Mitchell (1997) states that, for certain problems such as learning to interpret real-world sensor data, neural networks are among the most effective learning methods currently known but he also states that neural networks typically require long training times. Many researchers have investigated how to optimize different learning aspects or classifier performance of neural networks by introducing different optimization tasks or optimization algorithms, for example: Bebis and Georgiopoulos (1995) and Chalup and Maire (1999).

Artificial neural networks origins back to the 1940's (McCulloch and Pitts 1943, cited by Mitchell 1997) whereas the back propagation algorithm was invented 40 years later (Rumelhart and McClelland 1986; Parker 1985, cited by Mitchell 1997).

2.1.2 Naïve Bayes

The Naïve Bayes algorithm has one major difference compared to the other selected algorithms; it has no tuning parameters. Because of this fact, the classifier will not be included in the parameter tuning experiments. The algorithm is known to perform comparably with decision tree and neural network learning in some domains (Mitchell, 1997). This makes it an interesting algorithm to use in the experiments concerning the evaluation of classifiers using a measure function. Measure-based evaluation of Naïve Bayes can be compared with measure-based evaluation of decision trees and neural networks in order to find out if Naïve Bayes performs comparably with them when measured with a measure function instead of cross-validation.

Naïve Bayes is based on the assumption that the probability of observing a certain conjunction of attributes is just the product of the probabilities of observing the individual attributes. The Bayesian approach to classifying a new instance is therefore to assign the most probable target value to it, given its attributes. The hypothesis is formed simply by counting the frequency of various data combinations within the training examples (Mitchell, 1997). The basic notions of the Bayesian classifiers are discussed in (Duda and Hart 1973, cited by Mitchell, 1997).

2.1.3 C4.5

C4.5 is a decision tree learning algorithm that builds upon the ID3 algorithm. Amongst other enhancements (compared to the ID3 algorithm) the C4.5 algorithm includes different pruning techniques and can handle numerical and missing attribute values. This algorithm is chosen as a representation of the decision tree family of learning algorithms in the experiments conducted for this report. Parameters that can be tuned to achieve different types of classifier performance include pruning choice (or no pruning) and confidence level for pruning. Witten and Frank (2000) states that the induction of decision trees is probably the most extensively studied method of machine learning used in data mining. This statement and the fact that decision tree learners are a part of the experiments conducted on measure functions (Andersson et al.,1999) that are thoroughly discussed in this report justify the selection of C4.5 as one of the algorithms to use in the experiments of this report. Decision tree learners are fast and has been proven to give comparable results to that of neural networks for many problems and they are also generally faster than neural networks (Mitchell, 1997).

Decision tree algorithms learn instances by organizing the attributes of the dataset into rules in a tree structure. Each node in the tree performs a test on a certain attribute and the leaf nodes contain the target attributes. Instances are classified by sorting them down the tree starting from the root until the target attribute is found. As said before C4.5 is adapted from the ID3 algorithm (Quinlan 1986, cited by Mitchell 1997) and is thoroughly discussed by the author of both algorithms in (Quinlan 1993, cited by Mitchell 1997).

2.1.4 K-Nearest Neighbor

K-Nearest neighbor is the most basic type of instance-based learning methods (Mitchell, 1997). In instance-based learning, training instances are stored and a distance function is used to determine which instance of the training set that is closest to a new, unknown instance (Witten and Frank, 2000). This means that the learning time is very fast but the query response time is worse than many other algorithms. Witten and Frank (2000) explains this by stating that, in instance-based learning, all the real work is when the time comes to classify an instance, rather than when the training is processed. Researchers have tried to optimize instance-based algorithms with different methods. For example, Ho et al. (2002) have designed, what they call, an optimal nearest neighbor classifier which uses an intelligent genetic algorithm for optimization of instance space. Instance-based algorithms are often described as simple yet accurate methods for learning. Since the K-Nearest neighbor algorithm is both used in the experiment on measure functions by Andersson et al. (1999) and featured in many articles regarding optimization of learning algorithms their part in the experiments of this report is justified. During the experiments a plain K-nearest neighbor was used (no distance-weighting or other enhancements).

Algorithm	Parameters
Back propagation	Momentum, Learning Rate, Number of hidden layers, Number of hidden neurons
K-Nearest neighbor	Number of neighbors
C4.5	Confidence level for pruning, Pruning method, Instances per leaf, Number of folds for pruning
Naive Bayes	-

Table 2. Learning algorithm parameters used in the experiments contained in this work

In table 2 the different parameters of the learning algorithms, which were used during the experiments contained in this work, are presented. The number of hidden layers and number of hidden neurons parameters are not really part of the back propagation algorithm. These parameters belong to the neural network on which the back propagation operate but are listed as back propagation parameters in order to keep the distinction between classifiers and learning algorithms.

2.2 Evaluation of classifier performance

There are different ways to evaluate a classifier, including different types of cross-validation methods, graphical analysis of lift charts and ROC curves, and the use of measure functions. The choice of which one to use is dependent of many attributes and, according to Mitchell (1997), there is no method that satisfies all the constraints we would like. Adams and Hand (2000) explains that the construction of classifiers often involves a sophisticated design stage, whereas the performance assessment that follows is less sophisticated and sometimes very inadequate.

2.2.1 Cross-validation

Cross-validation is a statistical technique widely used for estimation of classifier performance. The procedure is to divide the available data into a number of folds or partitions. Training is then performed on all except one partition, which is left for testing. According to Witten and Frank (1999) ten fold cross-validation is the standard way of measuring the error rate of a classifier, however they acknowledge that there is substantial variance in an individual ten fold cross-validation. Because of this fact, an average of several ten fold cross-validations should be used.

Among the many variants of cross-validation there are some that occur more frequently in literature and discussions. These are ten fold cross-validation, leave-one-out and the bootstrap.

The difference between these three types of cross-validation lies in the way that data is partitioned. Leave-one-out is equal to n-fold cross-validation, where n stands for the number of instances in the data set. Leave-one-out or n-fold cross-validation is performed by leaving one instance out for testing and training on the other instances. This procedure is then performed until all instances have been left out once.

Bootstrap is based on sampling with replacement. The data set is sampled n times to build a training set of n instances. Some instances will be picked more than one time and the instances that are never picked are used for testing.

Another variant called CVCP (cross-validation strategy for evaluation of classifier performance) was proposed by van der Merwe and Hoffman (2001). The aim for CVCP is to be computationally less expensive compared to the other cross-validation methods. The procedure is to sample one training instance and then use the 1-nearest neighbor rule to determine the instance closest to the sampled instance. The class related to the closest instance is then removed as a representative of the sampled instance. The method is based on an approach presented by Monari and Dreyfus (2000). They have analyzed the effect that the withdrawing of a training instance has on the prediction of the classifier. They have shown, with experiments, that their cross-validation method is computationally less expensive than the conventional leave-one-out method and that this is accomplished by monitoring the influence that each training instance has on the classifier.

2.2.2 ROC curves

ROC stands for receiver operating characteristics. This is a term used in signal detection which characterizes the tradeoff between hit rate and false alarm rate over a noisy channel (Witten and Frank, 1999). ROC curves can be used to study the performance of a classifier over one chosen class. The procedure is to organize all the trained instances in a ranked list and order them by the classifier's predicted probability of classifying them as belonging to the class. A ROC curve is then plotted by starting at the southwest corner of the 2-dimensional graph and iteratively choosing the next instance (beginning at the instance most likely to belong to the class) and plotting it vertically if it is a true sample and horizontally if it is a negative sample. For an explanation of the different categories of samples, refer to table 3. The vertical axis depicts the percentage of true positives (an instance is predicted to belong to the class and it actually does) while the horizontal axis depicts the percentage of false positives (an instance is predicted to belong to the class but it does not).

		Predicted class	
		Yes	No
Actual class	Yes	true positive	false negative
	No	false positive	true negative

Table 3. Outcome of prediction used for creating ROC curves (Witten and Frank, 1999).

ROC curves are a very popular means to determine classifier performance and classifier comparison can be made by comparing the areas under the ROC curves created for each of the classifiers.

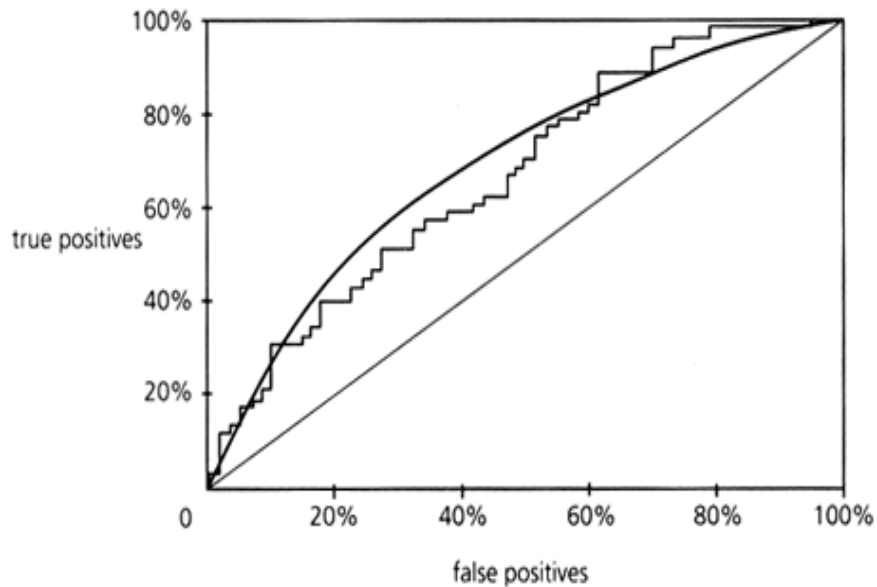


Figure 4. A sample ROC Curve (Witten and Frank, 1999).

In figure 4 a sample ROC curve is shown. The jagged line is created from the a sample of test data. The smooth curve is created by applying cross-validation to reduce sample dependence on the chart. Adams and Hand (2000) explain that ROC curves can only be used for comparison if they do not cross each other at any point. If they do, it means that one classifier is better than the other for some costs, but not all. If conclusions are drawn from such a comparison by summarizing over all costs, they could be completely incorrect.

2.2.3 Measure functions

Andersson et al. (1999) proposed a measure-based performance evaluation and gave an example of a measure function that took three aspects of the classifier into account; subset fit, similarity and simplicity. Subset fit is the most commonly used measure of classifier performance. Subset fit is the measure of correctness in classification of known instances. Andersson et al. (1998) explains that cross-validation can be seen as an average taken on a series of subset fit evaluations. Furthermore they explain that similarity is a good measure of the generalization capabilities of a classifier. Similar instances should be classified similarly. For instance, given a number of clusters of instances from different classes, the decision borders should be centered between the clusters. The simplicity aspect tells us how complex the solution is. One specific example of simplicity can be given regarding the creation of decision trees; the smaller the size of the tree, the more simple the solution is. The different attributes are put together into a measure function and each attribute can be biased in order to reflect the needs for a specific solution.

The measure function given as an example by Andersson et al. (1999) can only be used on data with numerical features since the simplicity and similarity measures are based on decision border lengths and distances between training examples and decision borders. What also needs to be considered is that the complexity of these calculations increases with the number of data attributes. Approximations of similarity and simplicity have been suggested to deal with this increased complexity.

2.2.4 Classifier comparison

Obviously the comparison of classifiers has very much to do with the way chosen to evaluate the performance of the classifiers. Comparisons are made in order to find the most appropriate classifier for a given class of problems. First of all, it must be decided which performance attributes to compare. The data to be learned must also be analyzed in order to decide which type of comparison can be made.

2.3 Methods for optimizing parameters

Many learning algorithms have input parameters that affect their performance. Parameters can be seen as algorithm settings and the optimal setting for a learning algorithm often depends on the problem at hand. Some of the most popular algorithms contain a large number of attributes that can be tuned in order to build a well performing classifier. Since different attributes can have impact on different aspects of classifier performance the logical conclusion of this discussion is that finding the optimal setting for an algorithm, given a specific problem or a specific class of problems, can be very hard or sometimes impossible. There are some existing methods for optimizing different classifiers and most of them work by choosing a set of parameters to affect, an algorithm for optimization and a way to measure how successful the optimization was.

While discussing the importance of network size in artificial neural networks, Bebis and Georgiopoulos (1994) depicts choosing the correct network size (The network size and structure are decided by a number of parameters) for a given problem as something of an art. Furthermore they explain that network size affects generalization capabilities, network complexity and learning time. Thus, in other words, tuning the network size could optimize these three performance attributes of classifier performance.

Next, we are going to look at three different optimization algorithms which can be used to tune one or more parameters of a classifier. While conducting the literature survey it was noticed that genetic algorithms, standard hill climbing and simulated annealing (or variations of these) were among the most frequently used in articles and experiments dealing with classifier performance optimization. Because of this fact, these three types of optimization were selected to be explained in this chapter. This choice has to do with the scope of this thesis. These three optimization techniques are discussed and examples of the usage of some of them in the experiments are shown in order to validate the earlier statement about how optimization algorithms can be used to increase classifier performance.

2.3.1 Genetic algorithms

Genetic algorithms, introduced by (Holland 1962, 1975, cited by Mitchell 1997), can be seen as simulations of the biological evolution. Although several different implementations of genetic algorithms exist, they all share some features. The algorithm begins by creating a pool of random hypotheses called the population. Hypotheses are generally constructed as bit strings and these can be of variable or static lengths. The algorithm works by iteratively evaluating each member of the population. A fitness function is used to calculate the fitness of each hypothesis. A new population is created and the fittest hypotheses are probabilistically chosen and included. Some of the hypotheses are copied into the new population without change while others are used to breed new hypotheses by genetic operations. These genetic operations include different types of crossover and mutation. Mitchell (1997) summarizes genetic algorithms as conducting a randomized, parallel, hill-climbing search for hypotheses that optimize a predefined fitness function.

Genetic algorithms are a popular optimization method and have been used in numerous areas, including the optimization of different classifiers in the field of machine learning. Bebis and Georgiopoulos (1995) have proposed a way to decrease

network size and increase generalization capabilities of a neural network with the help of genetic algorithms and weight elimination. They have compared the generalization capabilities and network size of three approaches, namely back propagation, weight elimination and genetic algorithms in combination with weight elimination. The genetic algorithm approach yielded the best generalization capabilities but came second to the weight elimination approach regarding network size. The original back propagation approach yielded the worst results in both network size and generalization capabilities.

Ho et al. (2002) have proposed a method of designing an optimal nearest neighbor classifier using an intelligent genetic algorithm (IGA). The goals of the optimization are to increase classification accuracy and decrease the sizes of the reference and the feature sets. The IGA uses an intelligent crossover function which includes functionality to economically estimate the contribution of individual genes to a fitness function. This enables the algorithm to choose the best genes from each parent instead of randomly choosing crossover points.

2.3.2 Hill climbing

Hill climbing is also commonly used for optimization. Chalup and Maire (1999) explain that hill climbing is similar to gradient descent in that they both approach a local optimum. Hill climbing, however, does this by randomly selecting step size and direction, whereas gradient descent is using the gradient. Mitchell et al. (1994) compare hill climbing with genetic algorithms with the aim of understanding for which class of problems genetic algorithms outperform hill climbing. They have developed a “Royal Road function” which can be described as a problem having all the necessary features to measure genetic algorithm performance. One of the most serious problems that occurred during the experiments was the problem of crowding or hitchhiking with the genetic algorithm.

Highly fit bit strings have a tendency to spread quickly in the population making it harder for less fit bit strings to evolve (even though they have a similar structure to more fit bit strings and are likely to become good candidates). Since hill climbing only stores two bit strings, one is the best so far and the other one is the next step, this problem does not occur. In reverse, one of the strengths of genetic algorithms is the random beam search which helps the algorithm to avoid getting stuck in local minima (Mitchell, 1997), whereas the hill climbing algorithm transforms smoother from one bit string to another resulting in a greater risk to get stuck in a local minima. Although it should be mentioned that there are ways to avoid or minimize the risk of getting stuck in local minima by using random step size, as described in (Chalup and Maire, 1999).

2.3.3 Simulated annealing

Simulated annealing, an algorithm very similar to standard hill climbing, is based on ideas from statistical physics. It was introduced by Kirkpatrick (1983, cited by Rutenbar 1989). Rutenbar (ibid.) states that [the algorithm] is motivated by an analogy to the statistical mechanics of annealing in solids. Instead of annealing a physical material into a good crystal, a poor solution of an optimization problem is “annealed” into a good solution. The algorithm begins by creating a first solution which consist of a number of parameters with random values. The algorithm then tries to find a configuration of the existing parameters that minimizes a cost function.

To put the algorithm in context of classifier performance optimization, some or all parameters of a classifier could be chosen as parameters for a simulated annealing algorithm with the purpose of optimizing the performance of that given classifier. Rutenbar (ibid.) explains that one issue of using simulated annealing is computation time. He means that annealing algorithms are often slow because of their iterative improvement nature.

However, Chakraborty and Chakraborty (1997) has shown, for two selected problems, that simulated annealing outperforms genetic algorithms by providing solutions with less variance and by requiring fewer computations or function evaluations.

2.4 Conclusion

Much work has been done in the research of classifier performance evaluation, comparison and classifier performance optimization, though the conclusion that can be drawn after conducting the literature survey is that most articles only focus on one optimization technique or one learning algorithm. Instead of explaining how back propagation can be optimized with genetic algorithms or how to measure the performance of a specific algorithm this work will show the importance of performing learning algorithm parameter optimization.

Choosing the “right” optimization algorithm is not crucial either. This work will show that genetic algorithms and hill climbing, although different in structure and execution, can optimize almost equally well when given the same task. All different algorithms include pros and cons but often the most important thing to consider is how to tune their parameters. Furthermore there are often discussions in the literature about which learning algorithm to use given a specific class of problem. We are going to see, with the help of experiments, that the most important matter is not always to choose the right algorithm for the class of problems at hand, but to tune the algorithm that is going to be used to optimize its performance over the class of problems.

One of the most widely used measurements of classifier performance is cross-validation. It has been proven to work well with numerous empirical studies and experiments. The abilities of cross-validation is not questioned but the intention is to give the reader an alternative way of measuring classifier performance. This alternative evaluation is based on an implementation of the measure function discussed in the literature survey.

2.5 Summary

These are the main points of interest in this chapter:

- There are many learning algorithms available. Back propagation, C4.5, Naïve Bayes and K-Nearest neighbor were studied more deeply than other algorithms during the literature survey. The justification for choosing these particular algorithms is that they capture the diversity of learning algorithms in many different aspects and thus making the experiment in this thesis more interesting.
- There are many ways to evaluate classifier performance. Examples include different versions of cross-validation tests, ROC curves and lift-charts. Another method was proposed by Andersson et al. (1999): measure-based classifier performance evaluation. A geometric measure function is calculated by measuring the distance between instances and decision borders as well as measuring the size of the decision borders.
- Classifier comparison should be practiced if there is an interest to find the best classifier given a specific class of problems. When comparing classifiers one or more of the evaluation methods are used.
- Many algorithms do have parameters that can be tuned in order to maximize classifier performance given a specific class of problems. There are many ways to tune or optimize these parameters and examples include genetic algorithms, hill climbing and a version of hill climbing called simulated annealing.

3 APPROACH

3.1 The use of a measure function

Many of today's solutions regarding the evaluation of classifier depend on statistical methods (for example cross-validation). The concept of using a measure function instead is an interesting alternative but is it feasible to use for real-world problems?

Since the use of measure functions for evaluating classifier performance was proposed by Andersson et al. in 1998 and no related work have been published (except the article by Andersson et al. in 1999) the discussion will be based upon their work. We will try to derive the possible negative and positive aspects of using measure functions for evaluation. In order to discuss measure functions it is wise to have an example of one measure function from which we can generalize. The measure function example found in the article by Andersson et al. (ibid.) will be used for this purpose.

As mentioned before this function can be divided into three parts; subset fit, similarity and simplicity. If we are looking at specific algorithms, for example decision trees, the simplicity aspect could be related to the number of nodes of the induced tree. One of the aims of using measure functions, however, is to provide a general solution for classifier evaluation; hence we do not want to lock ourselves to specific algorithms. Anderson et al. (1999) therefore suggest another way of measuring simplicity which they describe as a measure of the total size of the decision borders.

3.1.1 Similarity evaluation issues

To evaluate similarity we need to calculate the distance, d , between each instance and its nearest decision border. If an instance was correctly classified d is positive, otherwise it is negative. As a help for understanding we can model the decision space graphically as described in Anderson et al. (ibid.). The complexity and thus the computational efforts increase when we deal with a greater number of attributes per instance. The Iris database (Anderson, 1935), which is used in the experiments contained in this report, has four numerical attributes and this means that we have to search in four dimensions to locate the closest decision border for each instance. The database consists of 150 instances and this means that we need to perform 150 individual searches through all four dimensions to calculate the similarity measure of the measure function.

3.1.2 Simplicity evaluation issues

The simplicity measure is defined as the lengths of the decision borders. The calculation of simplicity also involves extensive computational effort as described about the similarity calculation above. The lengths of the decision borders are fairly easy to calculate if there are 2 or 3 dimensions. If this is the case there is even a possibility to draw a graph over the decision borders and measure them manually. When dealing with more dimensions, however this is not possible any more. Computational effort is not the biggest issue when working in higher dimensions than 3 though; the complexity of the actual calculation function grows with every dimension. Andersson et al. (1999) suggests that both similarity and simplicity measures should be approximated in some way to decrease computational effort. They give an example of how to approximate the simplicity by using the average number of decision borders crossed by random lines through feature space. The results from such an approximation are shown in the experiments section of this report.

These issues are especially important if the measure function has to be calculated many times during evaluation. Nevertheless the measure function provides us with an interesting approach to measuring classifier performance.

Even though cross-validation can give a prediction of how a certain classifier will cope with new data, it does not provide us with any analysis of the generalization capabilities and decision regions of the classifier. It is important also, to keep in mind that the discussed measure function is just an example. The concept of measure functions proposed by Andersson et al. (ibid.) could be used to come up with new solutions that may have even better analytical aspects. The computational efforts needed for the use of measure functions may also decrease with new variants.

Since the ambition is to give an example of how measure functions could be used in the place of other techniques like cross-validation, a version of the measure function described in Andersson et al. (ibid.) was implemented. In order for the measure function to work properly with more than two dimensions of instance attributes, approximation functions had to be written for the similarity and simplicity calculations.

3.1.3 Approximation for similarity

Similarity is calculated by going through each instance of data and measure the distance from it to the nearest decision border. The approximation lies in the way this search for a decision border is carried out.

For instance, if 2 dimensions are used we need to conduct a search starting from the data instance and moving out from it in a circular manner, checking the area around the instance. After each “circular check” we increase the radius of the search circle and continue this operation until we reach a border. The radius increase can be small in order to get a more exact solution, however the smaller the step the more time will be needed for computation. The ideal way in this case (for maximum correctness) would be to set radius increase close to zero and check every possible position on the circles. If 3 dimensions are used we need to conduct a search starting from the data instance and moving out from it in a spherical manner. Ideally, all possible coordinates in the sphere would have to be checked and the radius increase should be set close to zero. Setting the increase step close to zero is not feasible though.

The size of the increase of the search area can be tuned in order to get good performance and still approximate similarity at a reasonable level of detail. Cartesian coordinates were used instead of polar coordinates when implementing the similarity aspect. This made the search areas rectangular or box shaped instead of circular or spherical. Either way will do, the important step is to choose the size of the search increase step.

The following formula (1) approximates the number of calculations needed for the similarity aspect of the measure function. The following variables must be set: x corresponds to the number of instances, c is a resolution constant and n corresponds to the number of attributes (minus the target attribute). The resolution constant should be set to a value in the interval 200 – 2000. This constant should equal the approximate number of iterations needed when stepping from the instance to the closest border.

$$f(x, c, n) = x * c * (2^{n-1} n^2 + 2^{n-1} 10n) \quad (1)$$

Using (1) we can easily calculate the approximate number of calculations (classifications) needed to measure the performance of a classifier on the Iris database with 150 instances and 4 attributes:

$$f(150, 200, 4) = 150 * 200 * (2^3 4^2 + 2^3 10 * 4)$$

$$f(150, 200, 4) = 13440000$$

3.1.4 Approximation for simplicity

Whereas the similarity approximation methods aims towards reducing the computational effort needed, the main point of the simplicity approximation is to help reduce the complexity of the calculation. As described about the issues of simplicity calculation earlier the authors of the measure function proposal suggested a method of approximating simplicity. The implementation is based on this approximation suggestion. The method starts by computing a set of random line coordinates (in feature space). The lines are then searched by stepping from the beginning to the end and recording each occurrence of a decision border. The simplicity aspect is then approximated as the average number of decision borders crossed when searching a random line in feature space.

The procedure to find decision borders along a line is quite simple; at the start of the line the coordinates are used as attribute values for an instance and the learned classifier then classifies this instance and returns the predicted class. These events are then repeated at each step and if the class value returned has changed it means that we have found a decision border.

There are more questions that need to be answered in order to fully understand the concept of simplicity. One easy explanation of the simplicity aspect is that it can be said to capture how complex the learned classifier is. Two different aspects of simplicity are the lengths of the decision borders and the number of planes or decision regions that a classifier has. The above described approximation of simplicity only takes the number of planes aspect into account

3.2 Optimizing classifier performance

Optimization of classifiers can be done by tuning the parameters of a learning algorithm. Cross-validation or measure functions can be used to evaluate classifier performance. The optimization can be done automatically by optimization algorithms like standard hill-climbing, simulated annealing, genetic algorithms or parameter space sampling or manually by analyzing parameter space sampling charts.

3.2.1 Parameter space sampling

Many algorithms which have parameters passed to them often come with default values assigned to each parameter or at least recommendations of how to tune the parameters in order to get good performance. In other cases empirical studies or reasoning have been used to come up with good values for the different parameters. There are often some parameters that should vary in value depending on the data to be learned. Parameter space sampling is proposed as a general technique for finding good values for the parameters. Parameter space sampling can be used automatically or manually (in the latter case by analyzing parameter space sampling graphs) and can be used with any known classifier that has tunable parameters.

The procedure is to select one or more parameters of a learning algorithm and choose an appropriate set of data to learn from and an evaluation method. An interval is then chosen for each parameter. Next, a simple parameter space sampling algorithm is implemented. This algorithm steps through each parameter interval, calculating the performance and storing the results. The algorithm also keeps track of performance minimum and maximum values.

```

for (parA = minA; parA < maxA; parA += stepsizeA)
  for (parB = minB; parB < maxB; parB += stepsizeB)
    classifier = learning algorithm(parA,parB,dataset)
    performance result = evaluate(classifier,dataset)
    store performance result;
    performance sum += result;
    number of results++;
    if result < min
      min = result;
    else if result > max
      max = result;
average = performance sum / number of results;

```

Figure 5. Pseudo code for a simple parameter space sampling algorithm for two learning algorithm parameters

There are many ways to choose these limits and care should be taken when choosing them since the results that come from using this technique depends largely of the user's choice. A good rule of thumb is to investigate whether or not a certain parameter has a default or preferred value and to consider choosing an interval that includes the preferred value. Of course this rule of thumb is not valid in all cases.

So, why not choose the smallest value of a parameter as min and the largest as max? Well, it is not always possible due to computational effort issues. The performance of the classifier is measured for each possible set of parameter values and the performance can only be measured by training on data and then evaluating the results. Another important matter to consider is the step size of each parameter sampling. Step size affects the amount of samples to be gathered from each parameter interval.

There are different usages depending on the number of parameters (dimensions). If one or two parameters are sampled, the results can be used to draw a 2D or 3D graph, which can be very helpful when analyzing performance variance due to parameter value selection or minimum and maximum performance values (both local and global). Naturally a graph can not be produced if more than two parameters are used but the average, min and max performance values are always computed.

Several parameter space samplings can be performed on the same algorithm, each time sampling different parameters. For instance, sometimes it is valuable to sample two parameters at the same time in order to investigate the relationship between them. If we assume that two parameters are independent of each other, it is perhaps more wise to sample them independently. The computational effort needed for parameter space sampling increases with every parameter added (see for loops in figure 5). There are no limitations regarding the types of parameter values that can be used in parameter space sampling. The technique works for Boolean, integer, float and class type parameters. This statement is validated with experiments found in the next chapter.

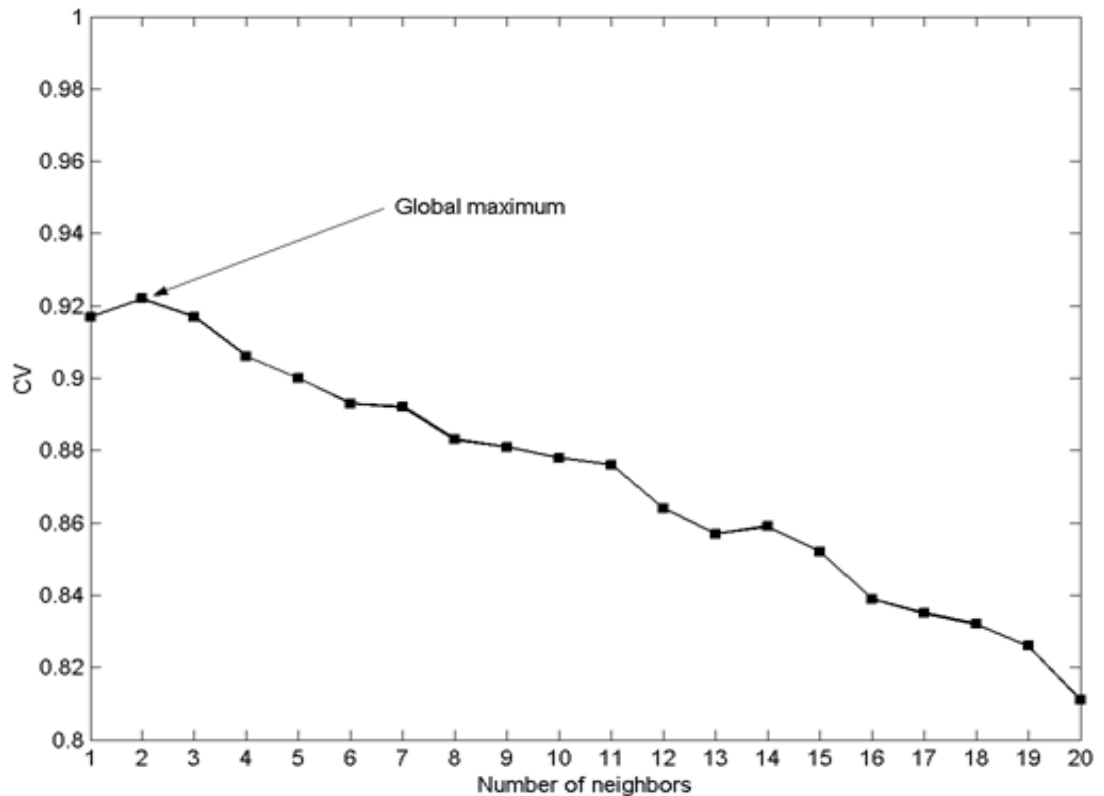


Figure 6. Parameter space sampling performed on K-Nearest Neighbor algorithm. One parameter is sampled (the number of neighbors). Database: Large Soybean database (Michalski and Chilausky, 1980).

An example of using the parameter space sampling technique can be seen in figure 6. The standard value of the chosen parameter (number of neighbors) is 1. Sampling is performed from 1 to 20 and the global maximum can easily be detected at 2. The measure used in this sampling is 10-fold cross validation. In this example the other important parameter of the algorithm, the window size, is set to variable (window size equals the number of instances in the database).

The concept of drawing a graph with two algorithm parameters and a performance result as three axis is not new. This has also been described in the documentation to the YALE environment (Fischer et al., 2002). Except for this documentation I have not found any information or articles related to this kind of parameter/classifier performance visualization. The method of parameter space sampling however has the main focus of searching the algorithm parameter space for good and bad classifier performance.

This can be accomplished without visualization, as has been described earlier, but there is a close relationship between the sampling technique and graphical visualization and they complete each other well in describing how classifier performance is affected by the tuning of parameters. Witten and Frank (2000) explains that one way to find the optimal value of k for the K-nearest neighbor algorithm is to perform several cross-validation tests with different values of k. They state that this procedure often yields excellent predictive performance. This procedure corresponds exactly to that of parameter space sampling.

3.2.2 Heuristic optimization algorithms

When good boundaries for the parameters are not known or there is simply not enough time to go through, let's say all combinations of four parameters in a certain interval, heuristic optimization algorithms provide another way to tune the parameters. Genetic algorithms, standard hill climbing and simulated annealing (an algorithm based on hill climbing) were studied in the literature survey. The reason for only discussing these three algorithms is mainly that much research has been made with them concerning classifier performance optimizing. They have all been proven to work for certain classifier optimizations but each has its possible drawbacks too.

Although somewhat different in structure these optimization algorithms perform very similar. Some authors claim that one is better than the other and this could most certainly be true for some cases, but the most important thing to consider, in order getting a good optimization, is not which algorithm to choose but how the parameters of the algorithm are tuned. This argument is validated by experiments with genetic algorithms and standard hill climbing found in the next chapter.

3.2.2.1 A genetic algorithm for classifier performance optimization

I have implemented a genetic algorithm based largely on the prototypical genetic algorithm provided by (Mitchell, 1997). Chromosomes are represented by bit strings so that genetic operations like mutation and cross-over can be carried out easily. Each chromosome represents a set of values for a selected number of learning algorithm parameters. The fitness for a chromosome is calculated by training the chosen learning algorithm after adjusting its parameters according to the values found in that chromosome and measuring its performance with a measure function, a cross-validation test or some other type of measure. The sizes of the chromosomes depend on the specific problem to be solved so they are adjustable. The population size is set to 20. Though it may seem that this is a rather small size it is important to bear in mind that the calculation of fitness that must be made for each individual chromosome in the population is very time consuming, when it comes to classifier performance optimization, since a classifier must be built and evaluated for each calculation.

Cross-over rate is set to a standard value of 95%. In other words, 5% of the chromosomes in a population are copied to a new generation and cross-over is performed on the rest to create new chromosomes. The mutation rate is set to 1% which is higher than the standard value (0.1%). The reason for this is the population size.

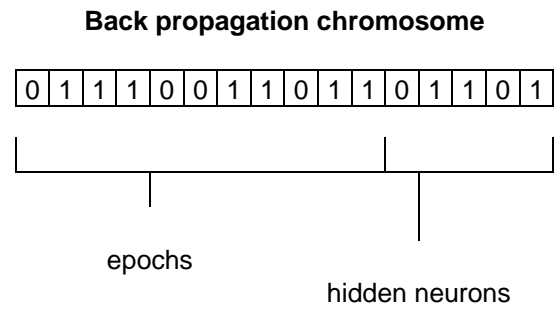


Figure 7. A simple chromosome that can hold two back propagation parameters; the number of epochs to train the network and the number of hidden neurons to use.

Figure 7 shows an example of how to code a chromosome with the values of two learning algorithm parameters. Great care should be taken when choosing the number of bits for each parameter. Fewer bits mean smaller sets of possible values and more bits means larger sets of possible values. If too few bits are used we may limit ourselves to the extent of never finding the optimal solution because it lies outside of the interval. If too many bits are used this may lead to extremely high values of some parameters, which could lead to substantial performance loss. For example, if we code the epoch parameter (from the back propagation chromosome shown in diagram 7) with 16 bits the number of epoch interval will stretch from 0 to 65535. If we, for the sake of argument, have a small database of training examples, the neural network may become very over-fitted if run through 60000 epochs. This is just an example and I do not claim that this is true for all cases of small databases.

The fitness threshold could be adjusted so that the algorithm terminates when a reasonable classifier performance have been found. The fitness threshold is often referred to as the termination criteria.

The roulette wheel method (also known as fitness proportionate) is used for selection of chromosomes for reproduction, cross-over and mutation. Although other methods like rank and tournament selection exist (Mitchell, 1997) these have not been considered. Early runs of the genetic algorithm using the roulette wheel selection have been proven to give satisfactory results and genetic algorithms are not the main focus of this report.

Cross-over is performed with random bits from two parents. A new random bit mask is created for every new generation. Cross-over is then performed by iterating through the random bit mask (which has the same size as the parents) and depending on the bit value found at a specific position, a bit at the same position is copied from the father or the mother to the child. There are many discussions on which cross-over function to choose and the only justification for choosing the random cross-over is the aim to get a diverse population in which new parameter values are created for each generation. It could be argued that the borders between the parameter values contained in the chromosome should be chosen as cross-over points instead of using random cross-over points. No comparison of these two methods of cross-over has been made in this work.

3.2.2.2 A standard hill climbing algorithm for classifier performance optimization

A standard hill climbing algorithm has been implemented as an alternative way of optimizing classifier performance. The procedure is to start with a solution and try to refine it, one step at a time, until a certain termination criteria is met. The implemented version of hill climbing is first given either a set of reasonable values for the parameters to be optimized (for example, values found out to be good, judging by experience or tests) or a set of random values. The algorithm then randomly chooses a parameter to change, a direction (positive or negative) in which to change it and the magnitude of the change. The justification for a random magnitude of change is given by (Chalup and Maire, 1999). Random step sizes can decrease the risk of running into local maxima. The relevant difference between this type of hill climbing and simulated annealing should be obvious. Simulated annealing tolerates large changes at the beginning of the run but the maximum size of change decreases for each “temperature drop” whereas the standard hill climbing allows the same type of changes throughout the run. The hill climbing algorithm gracefully moves from one step to the next and the algorithm only allows changes to “better” steps. Genetic algorithms, on the other hand, can change more drastic and could sometimes go from a good step to one that is worse.

3.3 Summary

These are the main points that have been discussed in this chapter:

- An alternative way of evaluation classifier performance can be achieved with a measure function
- The measure function example described by Andersson et al. (1999) can only be used when instance data consist of no more than two dimensions (attributes). The complexity of the necessary calculations increase greatly with more dimensions
- A measure function based on the example provided by (ibid.) that can handle more dimensions has been implemented as a part of this thesis. In order for this method to work, approximations had to be used for some parts of the measure function calculation.
- A genetic algorithm and a standard hill climbing algorithm have also been implemented to be used for optimization of different learning algorithms in the experiments related to this report.
- Another way to optimize classifier performance has been proposed. Parameter space sampling can be used to tune the parameters of any learning algorithm. The method can be used both to find parameter values for the maximum and minimum classifier performance as well as for providing data for graphical 2D and 3D visualizations of classifier performance with respect to different parameter values.

4 EXPERIMENTS

This chapter starts with a presentation of the overall goals of my experiments and the setup of each individual experiment is then explained. This is followed by a presentation of the environment and applications used for the experiments. The chapter ends with a review of, and discussion about, the results.

4.1 Overall goals of the experiments

The overall goals of the experiments are mainly to show the impact of parameter tuning of classifier learning algorithms and to give example of an alternative way to evaluate classifier performance. Two different optimization algorithms are demonstrated and compared and parameter space sampling is also shown as one way of optimizing.

4.1.1 Measure function demonstration

The measure function demonstration experiments are divided into three separate parts; a comparison between the measure function from the article by Andersson et al. (1999) and my own implementation, a demonstration of how my implementation works for more dimensions than 2 and a comparison between results obtained from cross-validation and results obtained from measure function evaluation.

This demonstration will, in short, show that my implementation gives comparable results to the measure function in the article (*ibid.*), that the geometric measure function concepts works for more dimensions than two, and that the measure function provides an alternative way of evaluation classifier performance although there exist issues concerning the computational effort needed.

4.1.2 Optimization test

The optimization test experiments are divided into two parts; a comparison between genetic algorithms and standard hill climbing, and a test of how well different learning algorithms can be optimized using these two optimization algorithms.

4.1.3 The impact of parameters

The impact of parameters is shown with different optimization tests as well as with the use of parameter space sampling. Another explanation for the impact of parameters is an answer to the question how foolproof the algorithm is. There are different factors to look at. For example, if two parameters are sampled over a reasonable interval, how high is the average performance value and, more importantly, how low is the lowest value? Some parameters affect the correctness of the learned classifier while others affect the complexity of the decision regions or the generalization capabilities. Some parameters affect more than one of these different aspects. Correctness and generalization capabilities (to a certain extent) can be evaluated by sampling the selected parameters using a ten fold cross-validation test as classifier evaluation. Aspects like simplicity and similarity (describing the complexity and decision regions of the classifier) can be evaluated by sampling the selected parameters using a geometric measure function instead of cross-validation. A combination of these two measures can also be used. Often this is not practical though, since both are quite demanding of computational effort.

4.2 Setup

All experiments in this chapter depend on the running of one or more learning algorithm implementations and therefore it was important to find good implementations of these algorithms. During the literature survey a number of environments with collections of learning algorithm implementations were found.

After studying the various environments, one particular environment called WEKA was chosen. Many articles cover the WEKA environment and the authors have also published a book describing how to use it. This book also covers the different algorithms, their possible weaknesses and strengths as well as information about classifier performance and other topics related to machine learning. To summarize, WEKA seems to be an established environment supported by many machine learning authorities. WEKA includes utilities for analyzing classifiers and performing statistical tests but, more importantly, it is also easy for developers to make use of WEKA's classes in their own Java applications. Since the measure function, the parameter space sampling algorithm, the genetic algorithms and the hill climbing algorithm all needed to be able to learn and evaluate classifiers, WEKA was a natural choice of environment. All implementations needed for the experiments were done in Java for seamless connectivity to the WEKA environment.

Microsoft Excel and Math Lab were used to present experiment results and graphs. Math Lab proved to be exceptionally good at creating 3D graphs, whereas Excel often was used to present tables of results and 2D graphs.

4.3 Conduction and results

In this section each experiment is described in detail and the results obtained from the experiment are displayed. A short review of the results can be found in the summary of this chapter and discussions about the results can be found in the conclusion part at the end of the report.

4.3.1 Measure function experiments

The measure function experiments serves two purposes; to examine the feasibility of the measure function as a measurement of classifier performance and to examine how my implementation of the measure function handles more dimensions than two.

The implementation of the measure function computes the different aspects (subset fit, similarity and simplicity) exactly as the function from the article (Andersson et al. 1999) and the same configuration and constants are used. The calculation formula for the measure function and the values used (as well as discussions about how they were chosen) can be found in the mentioned article.

4.3.1.1 Comparison between measure functions

The objectives of this experiment was to validate the implemented measure function by comparing the results obtained from it with the results from that of the measure function in the article by Andersson et al. (1999). The results in this article were obtained by running a number of learning algorithms (with different parameter values) on the Iris database (Anderson, 1935). This database was prepared in exactly the same way for both measure function evaluations. The preparation consisted of normalizing the attribute values and deleting two of the attributes. Only petal width and length attributes were used.

The learning algorithms used were C4.5, back propagation with 30 nodes and 26500 epochs training time, back propagation with 2 nodes and 25000 epochs training time, 1-nearest neighbor and 10-nearest neighbor.

The comparison is divided into four sub comparisons: similarity part 1 (similarity of correctly classified instances), similarity part 2 (similarity of incorrectly classified instances), simplicity (complexity of the decision regions) and the total measure function value. The lower the values of similarity and simplicity the higher the value of the measure function.

Algorithm	Subset fit		Simi1		Simi2		Simp		MF	
	AND	NLA	AND	NLA	AND	NLA	AND	NLA	AND	NLA
C4.5	0.980	0.980	0.682	0.676	0.005	0.008	2.325	2.387	1.260	1.263
Back prop 30n	0.987	0.993	0.599	0.664	0.002	0.000	2.802	4.360	1.215	1.217
Back prop 2n	0.960	0.960	0.703	0.710	0.008	0.008	2.516	2.400	1.245	1.259
1-KNN	1.000	0.993	0.679	0.682	0.000	0.002	3.330	2.700	1.256	1.268
10-KNN	0.967	0.960	0.740	0.736	0.009	0.013	2.697	2.469	1.265	1.273
Correlation	0.943		0.880		0.903		0.245		0.975	

Table 4. Results from the measure function comparison experiment; NLA represents values computed by the implementation in this report, AND represents values from the article by Andersson et al. (1999). Database: Iris (Andersson, 1935) with two attributes; petal length and width.

As can be seen in table 4 the correlation is very high between the similarity values and result values of the different measure functions but simplicity has a low correlation. Taking a look at the actual simplicity figures for each algorithm reveals that the algorithm that gained the highest simplicity value for one measure function came on second place when measured with the other function. Except for these two algorithms, the rank order of best algorithm correlates well between the different measure functions.

In the next experiment, the measure function results from the article by Andersson et al. (ibid.) were compared with the results obtained from the implemented measure function when using all four attributes of the Iris database (Anderson, 1935).

Algorithm	Subset fit		Simi1		Simi2		Simp		MF	
	AND	NLA	AND	NLA	AND	NLA	AND	NLA	AND	NLA
C4.5	0.980	0.980	0.682	0.763	0.005	0.013	2.325	2.019	1.260	1.317
Back prop 30n	0.987	1.000	0.599	0.699	0.002	0.000	2.802	1.941	1.215	1.301
Back prop 2n	0.960	0.987	0.703	0.726	0.008	0.003	2.516	1.239	1.245	1.320
1-KNN	1.000	1.000	0.679	0.692	0.000	0.000	3.330	2.517	1.256	1.283
10-KNN	0.967	0.967	0.740	0.723	0.009	0.008	2.697	2.096	1.265	1.279
Correlation	0.684		0.369		0.530		0.655		-0.254	

Table 5. Results from the measure function comparison experiment; NLA represents values computed by the implementation in this report, AND represents values from the article by Andersson et al. (1999). Database: Iris (Andersson, 1935).

Looking at table 5 and comparing with table 4 we see that the correlation is lower for simi1, simi2 and remarkably low for mf. The simp function correlation is higher than in table 4. Now we have to ask ourselves, should classifiers built on two and four attributes respectively produce similar results? The answer is, not necessarily. The subtraction or addition of one or more attribute greatly affects the learned classifier. This is shown in table 6 were the results from two cross-validation tests are compared.

Algorithm/Properties	CV4	CV2
Pruned tree	0.940	0.940
Backprop, 30 nodes	0.960	0.940
Backprop, 2 nodes	0.967	0.960
1-nearest neighbor	0.953	0.947
10-nearest neighbor	0.953	0.960

Correlation 0.51

Table 6. Comparison between cross-validation test results by learned classifiers using the Iris database (Anderson, 1935) with 4 and 2 attributes respectively.

The results from the two cross-validation tests differs considerably although not as much as the results shown in table 5. This may be an indication that the subtraction or addition of attributes have a greater impact on measure function results than on cross-validation test results.

4.3.1.2 The usage of measure functions

The experiment describing the usage of a measure function as an evaluation method have been carried out in order to try to highlight some of the features of the measure function.

A number of classifiers have been compared using the measure function and this is the result of the experiment.

Algorithm	Subset fit	Simi1	Simi2	Simp	Result
C4.5	0.980	0.763	0.013	2.144	1.314
Naive Bayes	0.960	0.733	0.018	3.898	1.238
Back 30	1.000	0.699	0.000	1.931	1.301
Back 2	0.987	0.722	0.003	1.335	1.316
1-nearest	1.000	0.692	0.000	2.604	1.281
10-nearest	0.967	0.723	0.008	2.043	1.281

Table 7. A demonstration of the usage of the measure function for analyzing classifiers.

There are many interesting aspects of the experiment results shown in table 7. First there are two comparisons between the same algorithms with different parameters. Two versions of back propagated neural networks are evaluated; one over fitted (Back 30) and one that is not (Back 2). From the results we can read that Back 30 managed a higher accuracy over the training data (Subset fit) and placed its decision border slightly closer to the instances than Back 2. The complexity of Back 30's decision borders is higher than that of Back 2. Consequently, Back 2 scores the highest overall result by providing a less complex solution. Exactly the same patterns can be spotted in the comparison between 1-nearest and 10-nearest. Unfortunately the difference in simplicity is not as clear as in the article by Andersson et al. (1999). This results in equal scores for the two nearest neighbor learned classifiers.

Naïve Bayes was said to give comparable results to that of decision trees and neural networks. When evaluated with the measure function we can see that it performs poorly. It has the longest distances from incorrectly classified instances to decision borders and the highest complexity. Consequently the Naïve Bayes classifier has the worst performance in this experiment.

The C4.5 learned decision tree performance is excellent in this experiment, outrun only by Back 2 (back propagated neural network with 2 neurons in 1 hidden layer, trained for 20000 epochs). The ranking of the algorithms was calculated using the measure function and the Iris database (Anderson, 1935) as the training set.

4.3.1.3 Comparison of CV and measure function results

In this part experiments are carried out on all the algorithms (and their configurations) contained in the examples by Andersson et al. (1999) except the ID3 algorithm. The reason for not including the ID3 algorithm is that the ID3 implementation in WEKA does not support numerical attributes. The first experiment compares the measure function values from the article with 10-fold cross-validation test values obtained from WEKA when running the same algorithms with the same configurations as in the text.

Algorithm/Properties	MF results	CV results
Pruned tree	1.260	0.940
Backprop, 30 nodes	1.215	0.940
Backprop, 2 nodes	1.245	0.960
1-nearest neighbor	1.256	0.947
10-nearest neighbor	1.265	0.960
Correlation		0.41

Table 8. Correlation between measure function results and cross-validation test results.

The correlation value tells us that the two methods of evaluating classifiers are quite similar (that is, they give similar results), but if we were to rank the best algorithms in two different lists (one ordered by measure function value and the other ordered by cross-validation value) the list would only slightly be alike. The first and the last places would be the same in both lists but in between there would be differences. My conclusion of this experiment is that it further validates the usage of a measure function as an alternative form of evaluation. If there would be no correlation with the widely used cross-validation it would be hard to recognize measure functions as a proper means to evaluate classifiers. If the correlation was too high this would mean that the two methods give more or less equal results. If that was the case there would be no need for an alternative method. It is important also, to consider the possibility that the correlation maybe would get higher if the measure function was configured differently. As mentioned before, the measure function implementation is configured the same way as the measure function from the article.

In the second experiment the simplicity part of the measure function has been taken away in order to investigate the impact that simplicity has on the correlation between measure functions and cross-validation tests.

Algorithm/Properties	MF results	CV results
Pruned tree	1.324	0.940
Backprop, 30 nodes	1.288	0.940
Backprop, 2 nodes	1.316	0.960
1-nearest neighbor	1.340	0.947
10-nearest neighbor	1.342	0.960
Correlation		0.46

Table 9. Correlation between measure function results (without simplicity factor) and cross-validation test results.

As can be seen in table 9 the simplicity factor has little impact on the correlation between the measure function values and those from the cross-validation tests. The experiment shows that, to a small extent, the simplicity factor decreases the correlation between results of the two methods.

As a last comparison of cross-validation tests and measure functions one experiment was run in order to measure the time consumed during evaluation of different classifiers. This is the result of the experiment:

Algorithm	CV Time (sec)	MF Time (sec)
C4.5	0,13	19,38
Back prop 30 nodes	2509,53	606,36
Back prop 2 nodes	218,42	79,99
1-Nearest neighbor	0,06	1818,19
10-Nearest neighbor	0,05	2168,47

Table 10. Time consumed during calculation of 10-fold cross-validation and measure function. Iris database (Anderson, 1935)

As can be seen in table 10 the cross-validation tests are generally quite fast except for the test on back propagation with 30 nodes. Reviewing the same table it seems that the measure function can be calculated quite fast except for the nearest neighbor algorithms. Some conclusions were drawn after running the time experiments. First the fact was considered that cross-validations works by training the algorithms over and over again and testing them with different test sets, whereas the measure function only needs to train each algorithm once. The opposite can be said about queries (a query is the same as the classification of an instance). The measure function has to query the classifier a large number of times in order to produce the simplicity and, especially, the similarity parts. Thus it would be logical if slow learning algorithms took a longer time to cross-validate than fast learners. It would also be equally logical if algorithms that answer queries slowly took longer time to evaluate with a measure function than those algorithms that answer queries fast.

From these conclusions we can deduce that query answering time is the most important factor for measure function calculation time and that learning time is the most important factor for cross-validation test time.

The C4.5 algorithm is a fast learner and answers queries quite fast too. We also know that back propagation algorithms provide rather slow learning but answer queries fast and that the nearest neighbor algorithm learns very fast but the classifier answers slowly. The different aspects of learning and answering queries for each algorithm mentioned in the literature survey combined with my conclusions about the abilities of the measure function and the cross-validation test explains the results shown in table 10.

4.3.2 Optimization

4.3.2.1 GA and HC comparison

In the next experiment, genetic algorithms and hill climbing were compared with respect to their chance of success and the time consumed during optimization. The Iris database (Anderson, 1935) was used as training and testing source and 10-fold cross-validation as a means to measure classifier performance. The number of calculations (number of cross-validation tests) was limited to a maximum of 5000 for each optimization algorithm. Since the genetic algorithms use a population size of 20 this means that 20 calculations was made each generation. This results in a maximum limit of 250 generations for the genetic algorithms.

The ending criteria for both optimization algorithms was that the learned classifier would find the, previously known, optimal value of 0.980 on the cross-validation test.

Algorithm	Chance of success	Mean error	Mean calculations	Mean error
GA	95%	2.2%	736	69
HC	85%	3.6%	1867	125

Table 11. Comparison of hill climbing and genetic algorithms.

As can be seen in table 11 both algorithms have similar chances of succeeding with the optimization task but genetic algorithms produce a slightly higher result and needs less than half the calculations compared with the hill climbing technique. What differs considerable between the two algorithms is the mean number of calculations needed to succeed in finding the optimal configuration. The number of calculations has a major impact on time consumed and genetic algorithms can therefore be said to execute twice as fast as the hill climbing algorithm.

100 runs were performed in order to get fair values of the chance of success and mean calculations for each algorithm.

4.3.2.2 GA and HC optimization example

In this experiment, the back propagation algorithm is optimized by tuning two parameters in order to get the optimal score on a 10-fold cross-validation test. The experiment is run on the Iris database (Anderson, 1935) and the two chosen parameters of back propagation are the number of epochs to train and the number of hidden neurons. WEKA's default values for these parameters are 500 and 3 respectively. When running with these default parameter values the back propagated neural network managed to score 0.973 on the cross-validation test.

Optimization	End epochs	End neurons	End CV	Improvement
GA	89	3	0.980	0.72%
HC	208	2	0.980	0.72%

Table 12. Optimization of back propagation with genetic algorithms and hill climbing.

Both standard hill climbing and genetic algorithms managed to find parameter values that made the learned classifier score the optimal value on the cross-validation test. The maximum value has been found by running a number of optimizations and parameter space samplings over several parameters of the back propagation algorithm and 0.980 is the highest score achieved on the Iris database. Clearly a lower value of the number of epochs made the network less over-fitted. The hill climbing optimization method also found a solution with 2 neurons instead of the default 3 (for WEKA).

4.3.3 Parameter space sampling

As described earlier parameter space sampling can be used to investigate the impact that different parameters have on classifier performance. The best, worst and average performance are calculated by stepping through different parameter configurations, training the selected learning algorithm and then calculating performance with one of the evaluation methods (for example cross-validation tests or measure functions). Judging by the difference between the best and the worst solution as well as the average it is possible to make some conclusions on how important a certain parameter is and how closely related it is to the training and test data.

4.3.3.1 K-Nearest neighbor sampling

The objective of this experiment was to investigate how the choice of neighbors impacts the learned classifier. A reasonable interval between 1 and 20 was chosen for this particular experiment. The run on the Iris database showed that a 2.1% increase of performance can be gained by using 15 nearest neighbors instead of 1. The run on the Soybean database (Michalski and Chilausky, 1980) showed that an 11.6% increase of performance can be gained by using 2 nearest neighbors (compared to the results obtained from the worst k of 19). The default number of neighbors for WEKA is 1.

Database	Best 10-CV	Best K	Worst CV	Worst K	Average CV
Iris	0.973	15	0.953	1	0.964
Soybean	0.922	2	0.826	19	0.875

Table 13. Parameter space sampling of K-Nearest neighbor. The sample interval was 1 - 20. 10-fold cross-validation was used as performance measure.

4.3.3.2 C4.5 sampling

The objectives of this experiment was to investigate how the choice of number of folds used for reduced error pruning impacts the learned classifier. Reduced error pruning is one method of pruning a C4.5 decision tree. One algorithm parameter is used to decide how many folds that should be used for pruning. The default value for WEKA is 3 folds and the lowest possible number is 2. The run on the Iris database showed that a 2.2% increase of performance can be gained by using the standard number of folds instead of 5. The run on the Soybean database showed that a 1.9% increase of performance can be gained by using 8 folds instead of 2.

As also shown in the K-nearest neighbor sampling (table 13) above the choice of parameter value is often closely related to the training data. This further justifies the need for parameter tuning instead of using the default values or the same values for different kinds of data.

Database	Best 10-CV	Best N Folds	Worst CV	Worst N Folds	Average CV
Iris	0.940	3	0.920	5	0.932
Soybean	0.893	8	0.876	2	0.885

Table 14. Parameter space sampling of C4.5. The sample interval was 2 – 10. 10-fold cross-validation was used as performance measure. The variable that was sampled was the number of folds used for reduced error pruning.

4.3.3.3 Back propagation sampling

In this experiment three parameters of the back propagation was sampled. Momentum and learning rate was sampled in five steps between 0.1 and 0.9. The number of epochs parameter was sampled between 100 and 500. The experiment was conducted using the Iris database. WEKA's default values are: momentum 0.2, learning rate 0.3 and number of epochs 500.

	CV	Momentum	Learning rate	Number of epochs
Best	0.980	0.1	0.3	100
Worst	0.933	0.9	0.9	110

Table 15. Parameter space sampling of back propagation.

The experiment shows two features of the parameter space sampling; the ability to find the most suitable value of one or more parameters, three in this case, as well as the ability to show the distance between the best and worst result values in a specific interval.

4.3.3.4 Comparison of sampled algorithms

In figure 8 and 9 the lowest and highest cross-validation test scores from the parameter space sampling experiments are visualized with box-whisker charts. In figure 8 we can see that C4.5 learned classifier will always score less than the K-nearest neighbor classifier but the neural network classifier (learned by back propagation) could score less than the worst C4.5 classifier as well as better than the best K-nearest neighbor classifier.

In figure 9 we can see that, depending on the parameter configuration, either one of the algorithms can produce the best classifier.

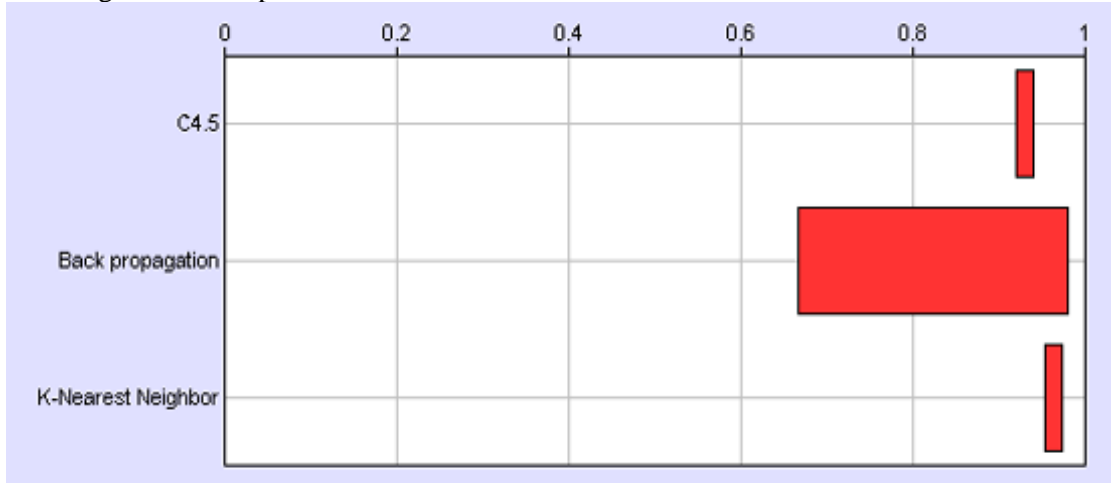


Figure 8. The lowest (left) and highest (right) cross-validation test results for the three algorithms subject to parameter space sampling. Database: Iris (Anderson, 1935)

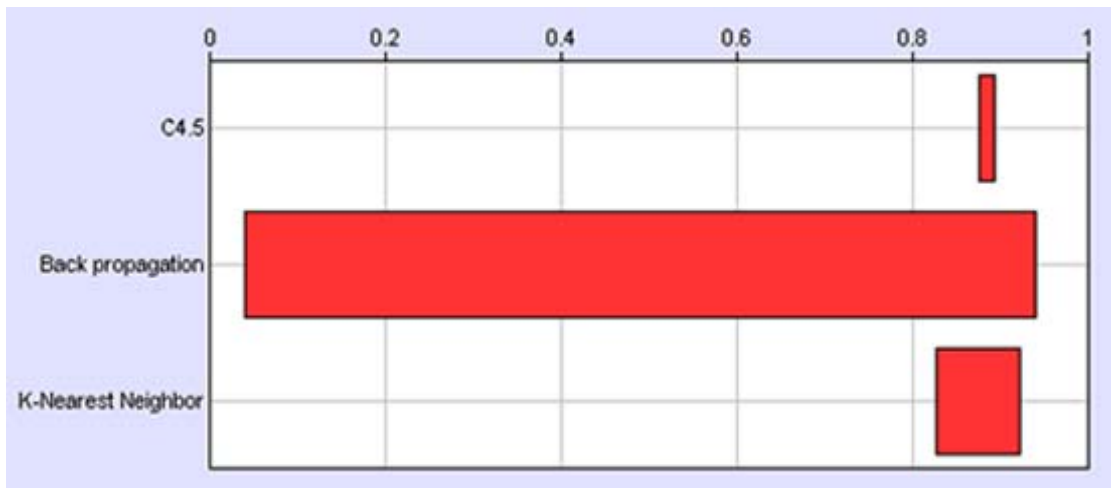


Figure 9. The lowest (left) and highest (right) cross-validation test results for the three algorithms subject to parameter space sampling. Database: Soybean (Michalski and Chilausky, 1980)

4.4 Summary

- The measure function has been shown to be an alternative to methods such as cross-validation tests. Issues concerning complexity of calculation and computational efforts exist with the implementation of the measure function and with the concept of measure functions as a whole but the method provides interesting analytical views of classifier performance.
- Cross-validation tests are generally faster than measure functions, especially when used on databases with many attributes.
- It is possible to optimize almost any algorithm with tunable parameters. The optimization can be done with optimization algorithms such as hill climbing or manually with parameter space sampling.
- Experiments show that my implementation of the measure function does not correlate satisfactory with the measure function from the article by Andersson et al. (1999) regarding performance results. It correlates high enough to bring fourth the features of a measure function; it can, for example, show the difference in complexity of a over-fitted classifier and one that is not.

5 CONCLUSION

There are many articles covering the evaluation and optimization of classifiers but almost all seem to cover just one optimization or evaluation technique often telling the reader that this particular method is the best. Perhaps one issue here is scope. It is not possible to cover many methods in detail in one small article. One part of this work can be seen as an attempt to get a wider perspective on evaluation and optimization by summarizing and presenting the different methods in both domains. The other part should be seen as a presentation and evaluation of an alternative measure of classifier performance. The implemented Java version of the measure function works very similar to that of the example found in (Andersson et al., 1999). Although there are issues with the implementation (mostly concerning computational effort) it produced interesting results in the experiments.

5.1 The importance of method choice

As can be seen in many experiments contained in this work there is often more difference in the results produced by the same algorithm with different parameter values than there is between two different algorithms or methods. This would mean that method choice is not so important. However, some algorithms are more foolproof than others. Experiments show that artificial neural networks produce higher average performance than most other algorithms. The lowest performance results of that algorithm are very high compared to the other algorithms too. The choice of method or algorithm should also depend on the class of problems. To conclude, one needs to consider more than method choice when trying to solve a machine learning problem. Many seem to focus on one particular algorithm and try to show that is better than other algorithms when, in fact, it can perform both better and worse than other algorithms depending on how well its parameters are tuned.

5.2 The importance of parameter tuning

The parameter configuration of a learning algorithm is often decided using prior knowledge or by using default values of a specific environment such as WEKA. The experiments contained in this work show that results produced by an algorithm (with one configuration) can differ greatly on different sets of data. The parameters can be tuned to produce good results on different sets of data. For example, the nearest neighbor algorithm was run on two different databases and the best performance was obtained by choosing 2 and 15 neighbors respectively. Another example can be found in a thesis by Hagelbäck and Svensson (2003). They have built a system to locate the transmembrane domains in a membrane protein sequence using a custom nearest neighbor algorithm. After evaluating and tuning of the k (number of neighbors) parameter they increased classifier performance with 2.1% up to 90.8% when using 21 nearest neighbors instead of 1. Cross-validation tests was used to evaluate classifier performance.

Clearly there is often a connection between the parameter values of an algorithm and the data set for which the classifier is built. Consequently one should not make use of default values of parameters and omit optimization. One important thing to remember however is that not all parameters are critical to the performance of a classifier. Some parameters may only affect training time while others have a major impact on accuracy and predictability.

5.3 Different ways of evaluation and optimization

Cross-validation tests are among the most frequently used and popular means of classifier performance evaluation. Although it is important to consider the fact that there may be other aspects of classifier performance that cross-validation tests do not reveal. The measure function featured in my work has been shown, in experiments, to reveal differences in decision space partitioning, classifier complexity and generalization capabilities. The conclusion after running the experiments is that the implementation of the measure function needs to be faster and, most importantly, must provide a simplicity calculation that is more stable.

There are many ways of optimizing a learning algorithm. Three methods are examined with experiments in this work. The genetic algorithm and the standard hill climbing both managed to find optimal configurations but the genetic algorithm found the optimal configuration in a smaller number of calculations. The parameter space sampling method was shown, with experiments, to be a feasible alternative.

5.4 Future work

When working on this thesis a lot of suggestions on future work has come up. There are certainly a number of experiments that would be interesting to run and many implementations should be examined and enhanced in order to become computationally less demanding to run. This is a list of the suggestions for future work in the area.

- The implementation of the measure function could be reworked. The calculation of similarity must, in some way, be remade so that it requires less computational effort. This is especially important when dealing with databases that contain many attributes.
- The calculation of simplicity is made by an approximation method as suggested in the article by Andersson et al. (1999). The correlation between the results of the implementation of the simplicity aspect presented in this work and the simplicity results from the article is not very high. In the article, the lengths of the decision borders was calculated in order to obtain a measure of the complexity of the learned classifier. The approximation works by calculating random lines across the decision space and checking, along these lines, how many times decision borders are encountered. Which method is most appropriate for calculating simplicity? Are there other alternatives?
- The simplicity implementation generates slightly different test results between runs. This is in fact a product of the random elements of the implementation and should be reworked in order to make the simplicity aspect more stable.
- A detailed comparison between simulated annealing, standard hill climbing, genetic algorithms and other optimization algorithms could be made to further investigate the similarities and differences of them.
- A prediction experiment involving the cross-validation test method and the measure function could be done. A number of learning algorithms should first be trained and evaluated with the two different methods and then new, unknown instances should be introduced and classified by the learned classifiers in order to investigate if their ability match the score they received by the different performance measures.

REFERENCES

- Adams, N. M. and Hand, D. J. (2000) 'Improving the Practice of Classifier Performance Assessment', *Neural Computation*, volume: 12 issue: 2, MIT Press, pp. 305-312
- Andersson, A., Davidsson, P. and Lindén, J. (1999) 'Measure-based classifier performance evaluation', *Pattern Recognition Letters*, volume: 20 issue: 11-13, North-Holland, Elsevier, pp. 1165-1173
- Andersson, A., Davidsson, P. and Lindén, J. (1998) 'Measuring generalization quality', Technical Report LU-CS-TR: 98-202. Department of Computer Science, Lund University, Lund, Sweden
- Andersson, E. (1935) 'The irises of the Gaspé Peninsula', *Bulletin of the American Iris Society* 59, pp. 2-5
- Bebis, G. and Georgiopoulos, M. (1995) 'Improving generalization by using genetic algorithms to determine the neural network size' in *Southcon 95*, Fort Lauderdale, Florida, pp. 392-397
- Bebis, G. and Georgiopoulos, M. (1994) 'Feed-forward neural networks', *IEEE Potentials*, volume: 13 issue: 4, pp. 27-31
- Chakraborty, M., Chakraborty, U.K. (1997) 'Applying genetic algorithm and simulated annealing to a combinatorial optimization problem' in proceedings of the 1997 International Conference on Information, Communications and Signal Processing, 9-12 Sep, ICICS, volume: 2, pp. 929 -933
- Chalup, S., Maire, F. (1999) 'A study on hill climbing algorithms for neural network training' in proceedings of the 1999 Congress on Evolutionary Computation, CEC99, volume: 3, pp. 1999-2021
- Fischer, S., Klinkenberg, R., Mierswa, I., and Ritthoff, O. (2002) 'Yale: Yet Another Learning Environment – Tutorial', CI-136/02, Collaborative Research Center 531, University of Dortmund, ISSN 1433-3325
- Hagelbäck, J. and Svensson, K. (2003) 'Locating transmembrane domains in protein sequences', Master Thesis, MCS-2003:07 / MSE-2003:10, Blekinge Institute of Technology, Ronneby, Sweden
- Ho, S.-Y., Liu, C.-C., Liu, S. (2002) 'Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm', *Pattern Recognition Letters*, volume: 23 issue: 13, North-Holland, Elsevier, pp. 1495-1503
- Michalski, R.S. and Chilausky, R.L. (1980) 'Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis', *International Journal of Policy Analysis and Information Systems*, volume: 4, No. 2

- Mitchell, M., Holland, J.H. and Forrest, S. (1994) 'When will a Genetic Algorithm Outperform Hill Climbing' in Cowan, J.D., Tesauro, G., and Alspector, J. (editors) *Advances in Neural Information Processing Systems*, volume: 6, 1994, Morgan Kaufmann Publishers, ISBN 1558603603, pp. 51-58
- Mitchell, T. M. (1997), *Machine Learning*, International Edition, McGraw-Hill Book Co, Singapore, ISBN 0-07-042807-7
- Monari, G. and Dreyfus, G. (2000) 'Withdrawing an example from the training set: an analytical estimation of its effect on a non-linear parameterised model', *Neurocomputing*, volume: 35 issue: 1-4, Elsevier, pp. 195-201
- Rutenbar, R.A. (1989) 'Simulated annealing algorithms: an overview', *IEEE Circuits and Devices Magazine*, 1989, volume: 5 issue: 1, IEEE, pp. 19-26
- van der Merwe, N.T. and Hoffman, A.J. (2001) 'Developing an efficient cross validation strategy to determine classifier performance (CVCP)' in proceedings of International Joint Conference on Neural Networks, IJCNN '01, Volume: 3, pp. 1663-1668
- Witten, I. H. and Frank, E. (1999), *Data Mining: practical machine learning tools and techniques with Java implementations*, Academic Press, Morgan Kaufmann Publishers, ISBN: 1-55860-552-5

FIGURES AND TABLES

- **Figure 1.** A database consisting of 17 instances divided into three different classes has been learned by an algorithm and this is a geometric visualization of the decision borders of the created classifier.
- **Figure 2.** Example of a performance (measured as the number of correctly classified instances) comparison between four learning algorithms.
- **Figure 3.** An example of a simple feed forward network with 1 hidden node, 2 inputs and 2 outputs.
- **Figure 4.** A sample ROC curve.
- **Figure 5.** Pseudo code for a simple parameter space sampling algorithm.
- **Figure 6.** Parameter space sampling performed on K-Nearest Neighbor algorithm.
- **Figure 7.** A simple chromosome that can hold two back propagation parameters; the number of epochs to train the network and the number of hidden neurons to use.
- **Figure 8.** The lowest (left) and highest (right) cross-validation test results for the three algorithms subject to parameter space sampling. Database: Iris (Anderson, 1935)
- **Figure 9.** The lowest (left) and highest (right) cross-validation test results for the three algorithms subject to parameter space sampling. Database: Soybean (Michalski and Chilausky, 1980)

- **Table 1.** A small excerpt from the Iris database (Anderson, 1935)
- **Table 2.** Learning algorithm parameters
- **Table 3.** Outcome of prediction used when creating ROC curves.
- **Table 4.** Results from the measure function comparison experiment.
- **Table 5.** Results from the measure function comparison experiment.
- **Table 6.** Comparison between cross-validation test results of two and four attribute learned classifiers.
- **Table 7.** A demonstration of the usage of the measure function.
- **Table 8.** Correlation between measure function results and cross-validation test results.
- **Table 9.** Correlation between measure function results.
- **Table 10.** Time consumed during calculation of 10-fold cross-validation and measure function.
- **Table 11.** Comparison of hill climbing and genetic algorithms.
- **Table 12.** Optimization of back propagation with genetic algorithms (first row) and hill climbing.
- **Table 13.** Parameter space sampling of K-Nearest neighbor.
- **Table 14.** Parameter space sampling of C4.5.
- **Table 15.** Parameter space sampling of back propagation.

MEASURE FUNCTION IMPLEMENTATION

```
/* Measure function implementation version 1.0
   Design and code by Niklas Lavesson, 2003

   Based on the measure function found in:
   Andersson, A., Davidsson, P. and Lindén, J. (1999) 'Measure-
   based classifier performance evaluation', Pattern
   Recognition Letters, volume: 20 issue: 11-13, North-Holland,
   Elsevier, pp. 1165-1173
*/

// Import classes from the WEKA environment needed for building
// and evaluating classifiers.

import weka.classifiers.*;
import weka.core.*;

import java.util.Enumeration;

public class MF {

    private double a0, a1, a2, k1, k2,
                  subsetFit, simi1, simi2,
                  similarity, simplicity,
                  measure;
    private Evaluation e;

    // Constructor for the MF class. For information on
    // the parameters a0, a1, a2, k1 and k2 please refer
    // to the article mentioned in the class comment.

    public MF(double a0, double a1, double a2,
              double k1, double k2) {

        this.a0 = a0;
        this.a1 = a1;
        this.a2 = a2;
        this.k1 = k1;
        this.k2 = k2;
    }

    // Method for evaluating a learned classifier's
    // performance on a specific set of data. For
    // information on the b parameter, please refer to
    // the article mentioned in the class comment.

    public double measure(Classifier cl, Instances data,
                          double b) {

        simi1 = 0;
        simi2 = 0;
        e = null;

        try {
            e = new Evaluation(data);
            e.evaluateModel(cl, data);
        } catch (Exception ex) {
            System.out.println(ex);
            System.exit(0);
        }

        double R = e.correct();
        double Cx = e.numInstances();
        getDistanceToBorder(cl, data, b);
        subsetFit = R / Cx;
        simi1 = simi1 / Cx;
        simi2 = simi2 / Cx;
    }
}
```

```

        similarity = k1 * simil + k2 * simi2;
        simplicity = calculateSimplicity(cl,
            data.firstInstance(), 800);
        measure = a0 * subsetFit + a1 * similarity - a2 *
            simplicity;

        return measure;
    }

    public double getSubsetFit() {
        return subsetFit;
    }

    private double calculateSimplicity(Classifier cl,
        Instance data, int numLines) {

        int numAttr = data.numAttributes() - 1;
        double si = 0.0, nc = 0.0, c = 0.0;
        int sa = 0;
        boolean ready = false;
        double[] start = new double[numAttr],
            stop = new double[numAttr],
            step = new double[numAttr],
            ia = new double[numAttr];

        for (int j = 0; j < ia.length; j++) {
            ia[j] = data.value(j);
        }

        for (int i = 0; i < numLines; i++) {
            sa = (int)(Math.random() * numAttr);
            start[sa] = 0.0;
            stop[sa] = 1.0;
            for (int j = 0; j < numAttr; j++) {
                if (j != sa) {
                    start[j] = Math.random() * (1.0/2.0);
                    stop[j] = Math.random() * (1.0/2.0) +
                        (1.0/2.0);
                }
                step[j] = (stop[j] - start[j])/100;
            }

            ready = false;
            c = -1;
            while (!ready) {
                for (int k = 0; k < numAttr; k++) {
                    data.setValue(k, start[k]);
                    start[k] += step[k];
                    if (start[k] >= stop[k]) ready = true;
                }
                try {
                    nc = cl.classifyInstance(data);
                } catch (Exception ex) {
                    System.out.println(ex);
                    System.exit(0);
                }
                if (c == -1) c = nc;
                else if (c != nc) {
                    c = nc;
                    si += 1.2;
                }
            }
        }
        for (int j = 0; j < ia.length; j++) {
            data.setValue(j, ia[j]);
        }
        si /= (double)numLines;
        return si;
    }
}

```

```

public void normalize(Instances data, double[] norm) {
    Instance i = null;
    for (Enumeration e = data.enumerateInstances();
         e.hasMoreElements();) {
        i = (Instance)e.nextElement();
        for (int j = 0; j < i.numAttributes() - 1; j++) {
            i.setValue(j,i.value(j)/norm[j]);
        }
    }
}

private void getDistanceToBorder(Classifier cl,
                                Instances data, double b) {
    Instance i = null;
    double[] ia = new double[data.numAttributes() - 1];
    double[] vardist = new double[ia.length];
    double c = 0.0,nc = 0.0, inc = 0.0;
    double part = 0.0;
    double incs = 0.0005;
    double dist = 0.0;
    int type = 0;
    int calc = (int)Math.pow(2,ia.length-1);
    int[] placeVal = {1,-1};
    int varChoice = 0;

    for (Enumeration e = data.enumerateInstances();
         e.hasMoreElements() ;) {
        i = (Instance)e.nextElement();
        for (int j = 0; j < ia.length; j++) {
            ia[j] = i.value(j);
        }
        c = i.classValue();
        type = 1;
        nc = classifyInstance(cl,i);
        if (nc != c) {
            type = -1;
            c = nc;
        }
        inc = 0.0;

        while (nc == c) {
            inc += incs;
            if (inc > 1) break;
            part = (inc * 2.0) / 10.0;
            for (int mainVar = 0; mainVar < ia.length;
                 mainVar++) {
                for (int cnt = 0; cnt < calc; cnt++) {
                    varChoice = 0;
                    for (int subVar = 0; subVar < ia.length;
                         subVar++) {
                        if (mainVar != subVar) {
                            vardist[subVar] =
                                placeVal[(cnt >> varChoice) & 1] * inc;
                            i.setValue(subVar,ia[subVar]+vardist[subVar]);
                            varChoice++;
                        }
                    }
                    for (double var = -inc; var < inc;
                         var+=part) {
                        vardist[mainVar] = var;
                        i.setValue(mainVar,ia[mainVar]+var);
                        nc = classifyInstance(cl,i);
                        if (nc != c) break;
                    }
                    if (nc != c) break;
                }
            }
        }
    }
}

```

```

        if (nc != c) break;
    }
}
dist = dist(vardist);
if (type == -1) simi2 +=
    ((1.0/Math.pow(2.0,b*(type*dist))) - 1.0);
else simi1 += (1.0 -
    (1.0/Math.pow(2.0,b*(type*dist))));

for (int j = 0; j < ia.length; j++) {
    i.setValue(j,ia[j]);
}
}
}

private double classifyInstance(Classifier cl, Instance i) {
    try {
        return cl.classifyInstance(i);
    } catch (Exception ex) {
        System.out.println(ex);
        System.exit(0);
    }
    return -1;
}

private double dist(double[] vars) {
    double sum = 0.0;
    for (int i = 0; i < vars.length; i++) {
        sum += Math.pow(vars[i],2);
    }
    return Math.sqrt(sum);
}

public double getSimi1() {
    return simi1;
}

public double getSimi2() {
    return simi2;
}

public double getSimilarity() {
    return similarity;
}

public double getSimplicity() {
    return simplicity;
}
}

```