Design, Implementation and Applications of the Support Vector Method Learning Algorithm

Mark Oliver Stitson
Royal Holloway and Bedford New College,
University of London

This thesis is submitted in part fulfilment
of the requirements for the degree of Ph.D
from the University of London.

November 20, 1999

To my parents

Abstract

This thesis is devoted to the study of the Support Vector method, a new learning technique. We start with an overview of classical techniques of neural networks, decision tree algorithms and Bayesian approches to machine learning. This is followed by a detailed introduction to the Support Vector method for pattern recognition and regression estimation and the use of kernels to implicitly use high dimensional feature spaces.

We discuss the details of our implementation of the algorithm and the associated problems in modern digital computers.

We show the limitations of the original method and introduce a set of novel heuristics to enable us to process large data sets with the Support Vector machine efficiently.

We present different methods to extend the Support Vector machine from binary pattern recognition to multi-class pattern recognition.

Finally we show experiments using the Support Vector machine. We use a set of benchmark tests and “real-world” applications. From the results we show the the advantages of the Support Vector machine and how to improve its performance.

The original motivation was to create an implementation of the Support Vector machine that would be able to handle large datasets. This has been achieved and and represents the basis for other researchers to investigate more of the properties of the Support Vector machine.

Acknowledgements

My initial thanks go to my supervisors Prof. Alex Gammerman and Prof. Vladimir Vapnik who gave me many ideas for my research.

As well as my supervisors, my advisor Dr. Liz Scott and our postgraduate co-ordinator Dr. Steve Schneider have given me moral support and practical help throughout.

I would like to thank Prof. K. Kulikowski at Rutgers University for enabling me to work with Prof. Vapnik in America for three months.

Special thanks go to Jason Weston, who never questioned my wild ideas, and implemented this massive SVM program with me. I would also like to thank Craig Saunders, Tom Melluish, Chris Watkins and Volodya Vovk for proof reading my thesis.

My thanks go to my friends and colleagues for stimulating conversations particularly over lunch and coffee.

At Royal Holloway: Barry Rising, Pete Burge, Jon Howell, Max van Daalen, Mark Hoyle, Nick Dunkin, Will Flett, Jon Butt, Bob Vickers, Joe Bater, Dave Cohen, Helen Treharne, Scott Smith, John Shawe-Taylor, Malcolm Hole, Siaw Lynn Ng, Victoria Redfern, Dave Springall, Andreas Fuchsberger

At AT&T: Patrice Simmard, Patrick Haffner, Yann LeCun, Leon Bottou

At GMD: Bernhard Schölkopf, Klaus-Robert Müller, Alex Smola

Finally most of all I would like to thank my parents for encouraging me to study at all and buying my first computer.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is devoted to the study of a new learning algorithm, the Support Vector method [71]. I shall present the design and implementation of our Support Vector software and applications of it.

My contribution to the field of Support Vector methods falls into three parts:

- I have designed a large scale Support Vector software library, which has been implemented in our research group.
- I have developed a series of heuristics, which makes it possible to solve large problems efficiently.
- I have applied a set of multi-class classification approaches to Support Vector machines.

These three contributions are underlined by around 500 experiments using our Support Vector machine library.

This chapter gives an overview over the field and presents some existing techniques.

1.1 Learning

The field of machine learning is the art of constructing computer programs that improve with experience.

The Oxford English Dictionary [58] defines learning as the process of acquiring knowledge.

The origins of machine learning can be traced back a long way, but the first real start was made in the 18^{th} and 19^{th} century when the first clock-work robots were built and inspired authors like E. T. A. Hoffmann to write about machines that learnt and even showed feelings [27].

In the 1920s R. A. Fisher developed many powerful statistical methods including one for discriminant analysis. This can be considered as one of the first learning algorithms. These methods were very limited at the time as all calculations had to be done by hand.

With Turing's ideas of intelligent machines with learning programs and the invention of modern binary computers by Zuse in Germany, Flowers in Britain and von Neumann in America, it became possible to implement statistical and computational methods and try out new methods.

Initially the only way to teach a computer to perform a task was to define a complete and correct algorithm and then program it into a computer. Solutions to many problems like image recognition, natural language processing and more complicated medical diagnosis, are hard to describe in words. Even human experts in the field are often not able to give a description in words which solves the problem. No one can describe a complete and correct algorithm to solve these problems. There is no way we can directly teach a computer the solutions or even algorithms to reach a solution. All we can do is to give the computer examples or new facts or some experience and a learning algorithm.

There are different forms of machine learning including

- learning from instruction, e.g. using a system that can learn facts that are presented to it,

- learning by analogy, e.g. changing an existing program slightly to adapt to a new situation,
- supervised learning or learning from examples, e.g. having a teacher that gives classified examples,
- unsupervised learning, e.g. finding facts about the environment by discovery.

We will concentrate on supervised learning in what follows.

1.2 Supervised Learning

In learning from examples there are two aspects: the examples we are given and the learning algorithm we choose. The examples we are given are largely predetermined, so the thing we have to design carefully is the learning algorithm. Although the learning algorithm is largely determined by the examples, most cases today can be reduced examples which contain a set of attributes[1]. This is the current state of the art and the area our research group is active in.

Early attempts at learning from examples include Winston's arches learning program. This program was designed to learn concepts in a world of wooden toy blocks. The program had to learn to distinguish arches from other structures. It was repeatedly shown examples and tried to include the arches it was shown, while trying to exclude other examples from the concept it had learnt so far. This program needed a large set of predefined relationships between blocks and large knowledge of what blocks existed, which made it very inflexible. Many modern algorithms work purely on numbers or discrete sets of attribute-values per attribute. This makes it far easier to write general programs, but puts a larger burden onto the person coding a specific problem. Like the arches program many programs try to minimize the number of mistakes made on the examples they are shown, this process is known as empirical risk minimization and the percentage of mistakes are know as the training error.

Earliest primitive attempts in learning from numeric examples were made in the

[1] This include images and sounds.

1950s using neural networks, when Rosenblatt introduced the perceptron. This single-layer neural network was proved to converge on a solution, if there was one. However it was proven that only a limited number of functions could be learnt by a perceptron. This limitation was only overcome with the introduction of the backpropagation learning algorithms for multi-layer perceptrons by Parker, LeCun and others in the 1980s. During the 1970s Vapnik and Chervonenkis developed their optimal separating hyperplane algorithm which also suffered from the fact that only a limited number of functions could be learnt by it[2]. The Support Vector machine overcomes these limitations. In the 1960s and 1970s decision trees were developed with Breiman and Friedman at the forefront. Probabilistic systems started appearing in the 1970s although probability theory had been long established. Many early probabilistic systems suffered from experts having to estimate many probabilities manually. All four of these methods were developed largely independently, their motivation coming from biology and psychology in the case of neural networks, statistics in the case of the optimal separating hyperplane and probability theory in the case of probabilistic approaches, and sociology in the case of decision trees. These methods only form a tiny fraction of learning algorithms available today, which include fuzzy learning systems, formal grammar learners, production rule systems, techniques that learn graphs and networks, clustering techniques, and many more. We have chosen the above four systems as they are typical systems used in this setting.

1.2.1 Neural Networks

Artificial neural networks are loosely modelled on biological neural networks. Neural networks consist of interconnected neurons, which work in parallel. Each neuron has a a set of inputs (in biology, dendrites) with an associated weight, which multiplies the incoming signal. Finally each neuron has one output (in biology, axon) the signal of which is calculated by applying a typically non-linear activation function to the sum of weighted inputs; this signal can be sent to other neurons or serve as an output of the neural network.

This biological model inspired the perhaps first artificial neural network, the so-

[2]There is however a suggestion of pre-processing the data in a similar way that later is done using kernel functions.

called McCulloch-Pitts neuron, which has the following properties:

- The McCulloch-Pitts neuron has one binary output.
- The McCulloch-Pitts neuron has as many binary inputs as needed.
- McCulloch-Pitts neurons are connected by weighted, directed paths.
- The McCulloch-Pitts neuron has a fixed threshold such that the neuron outputs a 1 when the net input is greater than the threshold, 0 otherwise. This is one of the simplest activation functions.

A learning rule for a single neuron was later proposed by Hebb. Hebb suggested this rule as a model of how biological neurons learnt. The training data has to be in the form of vectors of bipolar data[3] and the classifications or targets for each vector have to be bipolar as well. The neuron has as many inputs as there are attributes in each vector. Initially the weights and the threshold are set to zero. For each example in the training set the vector multiplied by its target is added to the weights and the target is added to the threshold. This training algorithm has several limitations and it cannot learn all functions the neuron can fulfil. It was not until the 1960s that Rosenblatt developed the perceptron learning rule. "Perceptron" is the name given to a type of neural network proposed by Rosenblatt. Perceptrons are very similar to the neural networks constructed from McCulloch-Pitts neurons. The perceptron learning rule has been proven to converge to a solution if such a solution exists. The perceptron rule is similar to the Hebb rule, except that it iterates through the training data until the weights stop changing, it only changes the weights if an example is misclassified and it has a learning rate factor α.

Given a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$ where the $\mathbf{x}_i$ are bipolar vectors and the y_i are bipolar target values, and a learning rate α, which for simplicity can be set to 1, the perceptron learning rule can be formulated as follows:

[3] All values are +1 or -1.

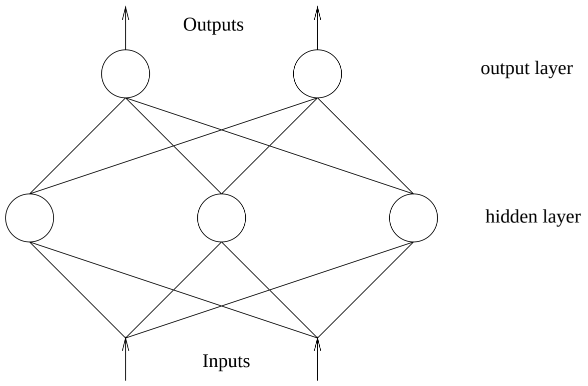

Figure 1.1: Multi-layer Perceptron

```
Initialize weights w and threshold/bias b to zero
Set learning rate α between zero and one
Do
    For each example vector x_i with target y_i
        Calculate the output for the neuron y = sign(b + Σ_i x_i · w_i)
        If the output y does not match the target y_i
            Update the weights w_new = w_old + α y_i x
            Update the bias b_new = b_old + α y_i
Until the weights and bias stop changing
```

The perceptron learning rule can only work for a one neuron network. Other similar algorithms were only capable of training single layer neural networks. A single layer neural network has a very restricted set of functions it can learn, which was one of the main reasons for a decline of interest in neural networks in the 1970s. Interest reemerged in the 1980s when the backpropagation training algorithm was developed independently by David Parker (1985), Yann LeCun (1986) and others to train multilayer feed forward networks. Backpropagation is a very general training algorithm, which in combination with multi-layer perceptrons can learn a wide range of functions. It is still widely used today.

The perceptrons used in multi-layer perceptrons for backpropagation have a continuous sigmoid activation function and are arranged in layers. In figure 1.1 we have two binary inputs and two binary outputs from two neurons in the output layer, the hidden layer contains three neurons. Training is accomplished by feeding each example through the network from the input to the output layer, and then calculating an update for the weights of the output layer and finally propagate this update back through the hidden layers, calculating a new update at each layer. The updates are calculated using the derivative of the activation function and effectively this is a gradient descent method minimizing the training error. Training is carried out until a satisfactory accuracy is achieved, for instance using cross-validation.

Neural networks are often used because they are easy to understand and produce a wide range of functions. They are applicable to many problems and are well established. Once trained they can work very fast, even though training can take a long time. Neural networks can be implemented in hardware, which makes them very fast once trained. Most other learning algorithms cannot be so straightforwardly implemented in hardware which can give neural networks a significant speed advantage.

1.2.2 Decision Trees

There are various algorithms that construct decision trees, including ID3, C4.5 [48], certain MML approaches by Wallace [40], and Bagging decision trees by Breiman [3].

Decision trees consist of decision nodes and leaf nodes. Given an example, a classification is found by traversing the tree starting at the root node. If a leaf node is reached the class associated with the leaf node is returned; if a decision node is encountered a path to follow is chosen on the basis of the example's attributes.

We will describe the method used by C4.5 to construct decision trees in detail.

1.2.3 C4.5

C4.5 creates decision trees where each decision node considers only one attribute.

Given a non-empty set of training examples T, there are two possibilities:

- The set T contains only examples of one class C_i: In this case create a leaf node associated with that class.

- The set T contains examples of k classes: In this case find a test X, based on a single attribute that has one or more mutually exclusive outcomes $\{O_1, \ldots, O_n\}$, which splits the set T into subsets $\{T_1, \ldots, T_n\}$, where T_i contains all examples with outcome O_i in the chosen test. Use this to create a decision node and apply the algorithm recursively to each subset T_i in turn.

If the set of training examples is empty outside information has to be used to determine this node. C4.5 makes this a leaf node with the class assigned as the class most often occurring in the training data.

The problem is to choose the right kind of test. C4.5 evaluates a gain ratio on all possible tests in its domain of tests, the test with the largest gain ratio is then chosen.

The domain of tests usually consists of two types of test, one for discrete attributes and one for continuous attributes.

For discrete attributes a test is created for each attribute. Each of these tests splits the training set T into as many sets as there are attribute values for that attribute. n attribute values will result in the sets $T_1, \ldots, T_n$ being created. To evaluate this test efficiently we can only evaluate the distribution of classes in T and the T_is. If S is a set of examples, let $\text{freq}(C_i, S)$ refer to the number of examples from class C_i in S and let $|S|$ stand for the number of examples in set S.

For each continuous attribute a test is found, thresholding the training set into two sets. The threshold is determined by finding the gain ratio for each possible threshold, chosen as an attribute value from the training set; and choosing the threshold with the largest associated gain ratio.

ID3 the predecessor of C4.5 used a so-called gain criterion which finds its parentage in information theory: The information a message conveys can be measured in bits and is minus the logarithm to the base two of its probability. The probability of an example from a set S being of class C_i is

$$\frac{\text{freq}(C_i, S)}{|S|}$$

and conveys

$$-\log_2\left(\frac{\text{freq}(C_i, S)}{|S|}\right)$$

bits of information.

So the information from this message of indicating class membership or the entropy of a set S can be found by summing the information over the classes in proportion to their frequencies in S:

$$\text{info}(S) = -\sum_{j=1}^{k} \frac{\text{freq}(C_j, S)}{|S|} \times \log_2\left(\frac{\text{freq}(C_j, S)}{|S|}\right) \tag{1.1}$$

bits.

$\text{info}(T)$ therefore measures the average amount of information needed to identify a class of an example in T.

Similarly if a test X is applied, the expected information requirement can be found as the weighted sum of subsets:

$$\text{info}_X(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \text{info}(T_i)$$

The test X that maximizes the gain

$$\text{gain}(X) = \text{info}(T) - \text{info}_X(T)$$

is then selected from all tests X.

It was found that ID3's gain criterion had a bias towards tests with many outcomes. This led to the introduction of normalization for C4.5 in which the gain resulting from many outcomes is adjusted. Analogously to $\text{info}(S)$ (1.1) a normalizer for the number of outcomes is

$$\text{split info}(X) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \log_2\left(\frac{|T_i|}{|T|}\right)$$

This measures the potential information gained by dividing T into n subsets T_i. The information gain measures the information relevant to the classification from using test X.

Now the proportion of information generated by a test X is

$$\text{gain ration}(X) = \text{gain}(X)/\text{split info}(X)$$

To avoid instabilities due to very small split info(X), the algorithm selects only tests where the gain(X) is greater than the average gain over all tests.

1.2.4 Bayesian Techniques

Bayesian approaches are part of probability theory. They learn the probability of outcomes on the basis of previous examples and try to evaluate the probability of outcomes for new examples on the basis of their attributes and previous experience.

The methods described here use conditional probabilities and Bayes' theorem, which will be described here briefly.

In the following A and B will denote events or attribute values and $P(A)$ the probability of event A occurring. Ω is the set of all possible outcomes of which all events are subsets.

Given the definitions of conditional probability of an event and the rule of total probability, Bayes theorem is derived as:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{k=1}^{n} P(A_k)P(B|A_k)} \tag{1.2}$$

The complication with Bayes' theorem arises when there are multiple random variables that take place at the same time. We might want to calculate the probability of A_i occurring given that B_i, C_i, D_i and E_i have occurred ($i \in \{0, 1\}$):

$$P(A_i|B_i \cap C_i \cap D_i \cap E_i) = P(A_i|B_i, C_i, D_i, E_i) \qquad i \in \{0, 1\}$$

We would need measurements for the outcomes $P(A_0)$, $P(A_1)$ and the conditional probabilities

$P(B_0, C_0, D_0, E_0|A_0)$, $P(B_0, C_1, D_0, E_0|A_0)$, $P(B_1, C_0, D_0, E_0|A_0)$,...

In this case there are two measurements needed for $P(A_0)$ and $P(A_1)$ and 32 conditional probabilities. If there are more events the number of measurements needed increases rapidly and becomes infeasible to calculate. Even if the probabilities for A and the conditional probabilities are combined into joint probabilities this still leaves 32 probabilities. The following methods give solutions to this problem.

1.2.4.1 Simple Bayes

The Simple Bayes method as described in [23] uses a drastic measure to simplify the problem of many different events. It simply assumes all events that are conditioned upon are independent.

The above problem resolves itself by assuming the variables B, C and D are independent:

$P(B_0, C_0, D_0, E_0|A_0) = P(B_0|A_0) \times P(C_0|A_0) \times P(D_0|A_0) \times P(E_0|A_0)$,
$P(B_0, C_1, D_0, E_0|A_0) = P(B_0|A_0) \times P(C_1|A_0) \times P(D_0|A_0) \times P(E_0|A_0)$,
etc.

This reduces the number of measurements needed from 34 measurements to two probabilities for A and 16 conditional probabilities. It is easy to show that this method comes into its own when the number of events conditioned upon becomes very large. Once this relatively small number of probabilities is found, usually simply by counting occurrences in the training data, Bayes' theorem can be applied.

In chapter 7.2.6 a medical database is presented with 33 random variables with several possible events each and 9 possible outcomes. For this database the number of conditional probabilities needed to calculate Bayes' formula exactly would be 77×10^{15} plus probabilities for the 9 possible outcomes, using the Simple Bayes method the number of conditional probabilities is 1,215 plus 9 probabilities for the outcomes.

This approach however ignores any interactions between variables and might lead to inconsistent results.

1.2.4.2 G&T

The G&T approach as described in [19] makes use of the observation, that if enough data is available, the outcome of a new set of attribute values is likely to be the same as the most common outcome for that set of attribute values from previous examples. So the probability of all outcomes A_i given the attributes $B_i, C_i, D_i, \ldots$ can be measured by Bayes' formula:

$$P(A_i|B_i, C_i, D_i, \ldots) = \frac{P(B_i, C_i, D_i, \ldots |A_i) \times P(A_i)}{P(B_i, C_i, D_i, \ldots)} = \frac{P(A_i, B_i, C_i, D_i, \ldots)}{P(B_i, C_i, D_i, \ldots)} \tag{1.3}$$

$$= \frac{\text{No. of past examples with } (A_i, B_i, C_i, D_i, \ldots)}{\text{No. of past examples with } (B_i, C_i, D_i, \ldots)} \tag{1.4}$$

The outcome A_i with the highest probability is then selected as the solution.

In practice however there rarely are enough examples for this to work, which is why significant attributes have to be found. For each class a χ^2 test is used to find a significant attribute value, which is then chosen. The training set for that class is then split according to that attribute value and the process is repeated for each split until χ^2 falls below a predetermined threshold. For each class there is a set of relevant symptoms which then can be used in Bayes' theorem to calculate the probability of outcomes for a new set of attributes.

1.2.4.3 Bayesian Networks

Bayesian networks take a slightly different approach than G&T to reduce the number of probabilities needed. Rather than assuming complete independence of all variables as Simple Bayes does, Bayesian belief networks assume independence between subsets of variables.

Bayesian belief networks assume that there are events which only indirectly affect other events. Say in our full probabilistic model event C is conditioned on A and B, a Bayesian belief network might assume that C is conditioned on B and B is conditioned on A such that C is conditionally independent of A given B. There can of course be two or more events influencing one event or one event influencing many other events. These dependencies can be represented by directed acyclic graphs and the probabilities are represented in conditional probability tables.

The structure of the graphs and the values of the probabilities can be determined automatically. However determining the structure of Bayesian belief networks is not trivial.

To use Bayesian belief networks, the outcomes for events which are known are instantiated and the other events' probabilities change and we receive a solution to our problem. Using Bayesian belief networks for classification tasks can be done by looking at the event we are interested in and selecting as the resulting class, the outcome with the highest probability.

A detailed introduction the Bayesian Networks can be found in [44].

1.3 Statistical Learning Theory

All approaches we have looked at so far try to minimize the error on the training set. In the case of neural networks by training until a low error is achieved on the training set, in the case of C4.5 by creating the decision tree according to the training data and in the Bayesian approaches implicitly by deriving the probabilities from the frequencies in the training data. Learning from examples is known as induction and minimising the error on the training examples is known as empirical risk minimisation. Highleyman showed in [26, 25] that teaching machines to be able to perform pattern recognition leads to the problem of minimising the empirical risk in a class of decision rules. Vapnik and Chervonenkis in [73] point out that this can be viewed as a branch of applied statistics, which is why they decided to call the theory "Statistical Learning Theory". Vapnik defines the three problems of pattern recognition, regression estimation and density estimation as the three main learning problems [71]. We will look at pattern recognition and regression estimation in the following and show a formalized setting of those problems.

1.3.1 Pattern Recognition

Given a set of functions

$$\{f_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \Lambda\}, \quad f_{\boldsymbol{\alpha}} : \mathbf{R}^N \to \pm 1 \tag{1.5}$$

and a set of examples, i.e. vectors $\mathbf{x}_i$ of attributes and target values y_i,

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_l, y_l) \in \mathbf{R}^N \times \pm 1, \tag{1.6}$$

each of which is generated from an unknown probability distribution $P(\mathbf{x}, y)$ which reflects the underlying dependency, we want to learn the function $f_{\boldsymbol{\alpha}^0}$ which produces the smallest possible average error on independently drawn examples from the same distribution P. This error is the risk

$$R(\boldsymbol{\alpha}) = \int \frac{1}{2}|f_{\boldsymbol{\alpha}}(\mathbf{x}) - y| dP(\mathbf{x}, y) \tag{1.7}$$

where

$$|f_{\boldsymbol{\alpha}}(\mathbf{x}) - y| \tag{1.8}$$

is the difference between the actual and estimated target values.

As the probability distribution P is unknown, $R(\boldsymbol{\alpha})$ is also unknown, and it is necessary to use an inductive method.

It is possible to minimize the error on the training set, the empirical risk

$$R_{\mathrm{emp}}(\boldsymbol{\alpha}) = \frac{1}{l}\sum_{i=1}^{l}\frac{1}{2}|f_{\boldsymbol{\alpha}}(\mathbf{x}_i) - y_i| \tag{1.9}$$

but this does not guarantee a small actual risk, especially if the number of training examples l is small. A function $f_{\boldsymbol{\alpha}}$ which achieves a small empirical risk on the training set does not necessarily have a small risk on an independent test set. If this is the case the function $f_{\boldsymbol{\alpha}}$ is said to be overfitted to the training data.

Approaches of how to deal with more than two classes are shown in chapter 5.

1.3.2 Regression Estimation

Given a set of functions

$$\{f_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \Lambda\}, \quad f_{\boldsymbol{\alpha}} : \mathbf{R}^N \to \mathbf{R} \tag{1.10}$$

and a set of examples, i.e. vectors $\mathbf{x}_i$ of attributes and target values y_i,

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l) \in \mathbf{R}^N \times \mathbf{R}, \tag{1.11}$$

each of which is generated from an unknown probability distribution $P(\mathbf{x}, y)$ which reflects the underlying dependency, we again want to learn the function $f_{\boldsymbol{\alpha}^0}$ which produces the smallest possible squared or absolute error on independently drawn examples from the same distribution P. This error is the risk either for absolute error

$$R(\boldsymbol{\alpha}) = \int |f_{\boldsymbol{\alpha}}(\mathbf{x}) - y| dP(\mathbf{x}, y) \tag{1.12}$$

or for squared error

$$R(\boldsymbol{\alpha}) = \int (f_{\boldsymbol{\alpha}}(\mathbf{x}) - y)^2 dP(\mathbf{x}, y) \tag{1.13}$$

Again the probability distribution P is unknown, and it is necessary to use an inductive method, such as minimizing the empirical risk

$$R_{\text{emp}}(\boldsymbol{\alpha}) = \frac{1}{l} \sum_{i=1}^{l} |f_{\boldsymbol{\alpha}}(\mathbf{x}_i) - y_i| \tag{1.14}$$

or

$$R_{\text{emp}}(\boldsymbol{\alpha}) = \frac{1}{l} \sum_{i=1}^{l} (f_{\boldsymbol{\alpha}}(\mathbf{x}_i) - y_i)^2 \tag{1.15}$$

but again this does not guarantee a small actual risk so that overfitting and bad generalization can take place.

1.4 Overview

In this chapter we have presented some existing learning algorithms. We have shown basic Neural Networks, C4.5 as an example of decision tree algorithms and Simple Bayes and G&T as two Bayesian methods. Finally we have formalized the pattern recognition and regression estimation problems we will address in this thesis.

The following gives an overview of the structure of the rest of this thesis:

In the next chapter we introduce the Support Vector method for pattern recognition and regression estimation and the use of kernels to implicitly use high dimensional spaces.

In chapter 3 our basic design for the Support Vector machine (SVM) library that was implemented at Royal Holloway is outlined. The object oriented design features are presented and the advantages of this shown. An overview of the C++ class structure is given.

In chapter 4 our heuristics for dealing with the limitations of modern computers are presented. Accuracy and capacity of modern computers are easily put to the limit when learning with Support Vector machines.

In chapter 5 we show various multi-class pattern recognition methods which are used with the Support Vector machine and other binary classifiers.

In chapter 6 we show some problem specific kernels that incorporate prior knowledge into the Support Vector machine.

In chapter 7 the Support Vector machine software is tested on some benchmark data sets and "real-world" data sets and the performance of the Support Vector machine is evaluated.

In chapter 8 we sum up, discussing the usefulness of the Support Vector method and pointing out paths of further research.

Chapter 2

Review of Support Vector Machines

Support Vector machines (SVM) are learning machines for pattern recognition and regression estimation. They are an extension to the linear optimal separating hyperplane method.

Below we consider the basic ideas and solutions for pattern recognition and regression problems. This approach was developed by Vapnik and Chervonenkis in [73, 71, 72].

2.1 The Optimal Separating Hyperplane

Our aim in using separating hyperplanes is to find a $n-1$ dimensional hyperplane in a n-dimensional space that divides the space so that the points

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l), \mathbf{x}_i \in R^n, y_i \in \{\pm 1\}$$

are separated such that all points for which $y_i = +1$ are on one side of the hyperplane and all points for which $y_i = -1$ are on the other side of the hyperplane.

In figure 2.1 it is easy to determine a separating hyperplane like the one in fig-

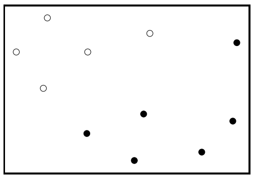

Figure 2.1: Training points

ure 2.2 manually. Throughout this section we will assume that the points are linearly separable as in figure 2.1.

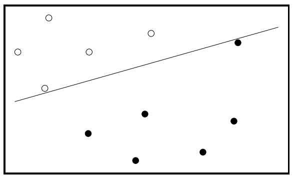

Figure 2.2: Training points with separating hyperplane

In the following we will use the terms inner product or dot product of two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbf{R}^n$ to mean

$$\mathbf{x} \cdot \mathbf{y} = \begin{pmatrix} x_1 \\ \cdots \\ x_n \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ \cdots \\ y_n \end{pmatrix} = x_1 \times y_1 + \cdots + x_n \times y_n$$

and the length of a vector $\mathbf{x}$ to be

$$||\mathbf{x}|| = \sqrt{\mathbf{x} \cdot \mathbf{x}}.$$

The hyperplane $(\mathbf{w} \cdot \mathbf{x}) + b = 0$ in figure 2.2 satisfies the conditions:

$$(\mathbf{w} \cdot \mathbf{x}_i) + b > 0 \text{ if } y_i = 1$$
$$(\mathbf{w} \cdot \mathbf{x}_i) + b < 0 \text{ if } y_i = -1$$

which can be transformed into [1]

[1] possibly multiplying $\mathbf{w}$ and b by some constant factor

$$(\mathbf{w} \cdot \mathbf{x}_i) + b \geq 1 \text{ if } y_i = 1 \tag{2.1}$$

$$(\mathbf{w} \cdot \mathbf{x}_i) + b \leq -1 \text{ if } y_i = -1 \tag{2.2}$$

which is equivalent to

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1, i = 1, \ldots l \tag{2.3}$$

This however is not necessarily a good hyperplane. Intuitively points which are close to each other should be classified the same; however it is possible that some points can be very near the hyperplane and points near them might be the other side of the hyperplane and as such classified differently although intuitively they should be the same class. Therefore it would be preferable to have a hyperplane with a large margin or even the largest possible margin from any of the training points. This hyperplane with the maximum margin is known as the optimal separating hyperplane.

In our example it would look like figure 2.3.

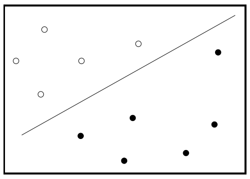

Figure 2.3: Training points with optimal separating hyperplane

To find the optimal separating hyperplane we have to find the hyperplane which satisfies the condition (2.3)[2] and maximizes the minimum distance between the hyperplane and any point in the training data. The distance is:

$$\rho(\mathbf{w}, b) = \min_{\{\mathbf{x}_i | y_i = 1\}} \frac{\mathbf{w} \cdot \mathbf{x}_i + b}{||\mathbf{w}||} - \max_{\{\mathbf{x}_j | y_j = -1\}} \frac{\mathbf{w} \cdot \mathbf{x}_j + b}{||\mathbf{w}||} \tag{2.4}$$

[2]This is true for all separating hyperplanes.

which is the sum of the distances from the nearest points to the separating hyperplane.

From equation (2.4) and equations (2.1) and (2.2) it follows that:

$$\begin{aligned}
\rho(\mathbf{w}, b) &= \min_{\{\mathbf{x}_i | y_i = 1\}} \frac{1}{||\mathbf{w}||} - \max_{\{\mathbf{x}_j | y_j = -1\}} \frac{-1}{||\mathbf{w}||} && (2.5)\\
\Leftrightarrow \ \rho(\mathbf{w}, b) &= \frac{1}{||\mathbf{w}||} - \frac{-1}{||\mathbf{w}||} && (2.6)\\
\Leftrightarrow \ \rho(\mathbf{w}, b) &= \frac{2}{||\mathbf{w}||} && (2.7)\\
\Leftrightarrow \ \rho(\mathbf{w}, b) &= \frac{2}{\sqrt{\mathbf{w} \cdot \mathbf{w}}} && (2.8)
\end{aligned}$$

The optimal separating hyperplane therefore minimizes

$$\frac{1}{2}\mathbf{w} \cdot \mathbf{w}$$

with respect to both the vector $\mathbf{w}$ and the scalar b subject to the constraints (2.1) and (2.2). Therefore the problem of finding the optimal separating hyperplane is transformed into a relatively simple quadratic optimization problem, and in theory can be solved using standard optimization packages.

As the above quadratic optimization problem is hard to solve in practice, another approach is to find the saddle point of the appropriate Lagrange functional:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w} \cdot \mathbf{w} - \sum_{i=1}^{l} \alpha_i \left\{ [(\mathbf{x}_i \cdot \mathbf{w}) + b] y_i - 1 \right\} \qquad (2.9)$$

where α_i are Lagrange multipliers. The Lagrangian has to be minimized with respect to $\mathbf{w}$, b and maximized subject to

$$\alpha_i \geq 0. \qquad (2.10)$$

In the saddle point the solutions $\mathbf{w}_0$, b_0 and $\boldsymbol{\alpha}^0$ should satisfy the following conditions:

$$\begin{aligned} \frac{\partial L(\mathbf{w}_0, b_0, \boldsymbol{\alpha}^0)}{\partial b} &= 0 \qquad (2.11) \\ \Leftrightarrow \quad \textstyle\sum_{i=1}^{l} \alpha_i^0 y_i &= 0 \qquad (2.12) \end{aligned}$$

$$\begin{aligned} \frac{\partial L(\mathbf{w}_0, b_0, \boldsymbol{\alpha}^0)}{\partial \mathbf{w}} &= 0 \qquad (2.13) \\ \Leftrightarrow \quad \mathbf{w}_0 - \textstyle\sum_{i=1}^{l} \alpha_i^0 \mathbf{x}_i y_i &= 0 \qquad (2.14) \end{aligned}$$

This results in the following properties of the optimal separating hyperplane:

1. Constraint on the coefficients α_i^0 from equation (2.12):

$$\sum_{i=1}^{l} \alpha_i^0 y_i = 0, \qquad \alpha_i^0 \geq 0, i = 1, \dots, l \qquad (2.15)$$

2. The vector $\mathbf{w}_0$ is a linear combination of the vectors in the training set (from equation (2.14)):

$$\mathbf{w}_0 = \sum_{i=1}^{l} \alpha_i^0 y_i \mathbf{x}_i, \qquad \alpha_i^0 \geq 0, i = 1, \dots, l \qquad (2.16)$$

Those vectors $\mathbf{x}_i$ for which the corresponding α_i^0 are non-zero are called the Support Vectors and therefore the expansion of $\mathbf{w}_0$ can be written as

$$\mathbf{w}_0 = \sum_{\text{support vectors}} \alpha_i^0 y_i \mathbf{x}_i, \qquad \alpha_i^0 > 0 \qquad (2.17)$$

3. The Kühn-Tucker Theorem states the necessary and sufficient conditions for having found an extreme point of the original problem occurs when either the Lagrangian multipliers $\alpha_i = 0$ or the constraints $\{[(\mathbf{x}_i \cdot \mathbf{w}_0) + b_0]y_i - 1\} =$

0 for all i and the Lagrangian multipliers define a saddle point of the Lagrangian

This means that the separating hyperplane is the optimal separating hyperplane when

$$\alpha_i^0 \{[(\mathbf{x}_i \cdot \mathbf{w}_0) + b_0]y_i - 1\} = 0, \qquad i = 1, \ldots, l \tag{2.18}$$

and $\boldsymbol{\alpha}^0$ defines an extreme point of the Lagrangian.

Substituting equations (2.15) and (2.16) into the Lagrangian (2.9), the following functional is obtained:

$$\begin{aligned} W(\boldsymbol{\alpha}) &= \tfrac{1}{2}\mathbf{w} \cdot \mathbf{w} - \textstyle\sum_{i=1}^{l} \alpha_i \{[(\mathbf{x}_i \cdot \mathbf{w}) + b]y_i - 1\} \\ &= \textstyle\sum_{i=1}^{l} \alpha_i - \tfrac{1}{2} \textstyle\sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \end{aligned} \tag{2.19}$$

This functional has to be maximized in the non-negative quadrant,

$$\alpha_i \geq 0, \qquad i = 1, \ldots, l \tag{2.20}$$

under the constraint

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{2.21}$$

2.1.1 Calculating the class of an unseen example

Once the solution to this problem has been found in form of a vector $\boldsymbol{\alpha}^0 = (\alpha_1^0, \ldots \alpha_l^0)$, an indicator function can be constructed:

$$f_{\boldsymbol{\alpha}^0}(\mathbf{x}) = \operatorname{sign}\Big(\sum_{\text{support vectors}} y_i \alpha_i^0 (\mathbf{x}_i \cdot \mathbf{x}) - b_0 \Big) \tag{2.22}$$

where $\mathbf{x}_i$ are the Support Vectors, α_i^0 are the Lagrange multipliers and b_0 is the threshold value

$$b_0 = \frac{1}{2}[(\mathbf{w}_0 \cdot \mathbf{x}^*(1)) + (\mathbf{w}_0 \cdot \mathbf{x}^*(-1))] \tag{2.23}$$

where $\mathbf{x}^*(1)$ is any Support Vector belonging to the first class and $\mathbf{x}^*(-1)$ is any Support Vector belonging to the second class.

This solution only holds for separable data, but can be modified for non-separable data.

2.1.2 Non-separable data

For linearly non-separable data the constraints (2.1) and (2.2) have to be modified to allow for some slack:

$$(\mathbf{w} \cdot \mathbf{x}_i) + b \geq 1 - \xi_i \text{ if } y_i = 1 \tag{2.24}$$

$$(\mathbf{w} \cdot \mathbf{x}_i) + b \leq -1 + \xi_i \text{ if } y_i = -1 \tag{2.25}$$

where the $\xi_i \geq 0$ are slack variables for each training example and in the ideal case are zero.

In effect some examples are allowed on the wrong side of the margin, which is why this technique is also called Soft Margin Hyperplane technique and the original technique without slack variables is called Hard Margin Hyperplane technique (see [9]).

The aim here must be to not only minimize

$$\frac{1}{2}\mathbf{w} \cdot \mathbf{w}$$

but also to minimize

$$\sum_i \xi_i$$

under the above constraints.

The function to be minimized then is

$$f(x) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \tag{2.26}$$

under the above constraints. C is a factor to weight the two terms against each other and must be set a priori.

The Lagrangian for this is

$$L(\boldsymbol{\alpha}, \xi) = \sum_i \alpha_i - \frac{1}{2}\sum_i y_i y_j \alpha_i \alpha_j \mathbf{x}_i \mathbf{x}_j + \sum_i \xi_i (C - \alpha_i) \tag{2.27}$$

It is derived in appendix B and to solve it (2.19) has to be maximized under the constraints

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, l \tag{2.28}$$

2.2 The Support Vector Machine

Support Vector machines map the input space into a high-dimensional feature space through some non-linear mapping chosen a priori and then construct the optimal separating hyperplane in that feature space. This makes it possible to construct linear decision surfaces in feature space which correspond to non-linear decision surfaces in input space.

Taking the example in figure 2.1 with one extra point as in figure 2.4

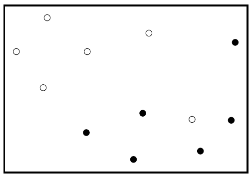

Figure 2.4: Training points

it is obvious that no linear separating hyperplane can be found in input space, yet if we could use a non-linear separating hyperplane, which is equivalent to mapping into a high dimensional feature space and then using a linear separating hyperplane in that feature space, a separation is possible as in figure 2.5.

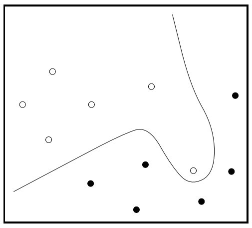

Figure 2.5: Training points with a "non-linear" separating hyperplane

The main problem that arises here is how to handle the problem of mapping all vectors into the feature space. This mapping would generate vectors with high dimensionality entailing large storage and computational expense. This curse of dimensionality however turns out not to be a problem as all we ever need is the dot product of the vectors in feature space.

If we choose a mapping from the original space X to the higher dimensional space Z ($k : X \to Z$):

$$k(\mathbf{x}) = \mathbf{z}$$

where $\mathbf{x} \in X$ and $\mathbf{z} \in Z$ then we would have to maximize

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (k(\mathbf{x}_i) \cdot k(\mathbf{x}_j)) \tag{2.29}$$

to find the Lagrangian multipliers under the constraints from the previous section and calculate the value of the classifier

$$f_{\boldsymbol{\alpha}^0}(\mathbf{x}) = \operatorname{sign}\Big(\sum_{\text{support vectors}} y_i \alpha_i^0 (k(\mathbf{x}_i) \cdot k(\mathbf{x}) - b_0\Big) \tag{2.30}$$

as before.

Note that we do not have to know the function $k(\mathbf{x})$ as long as we know the kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i) \cdot k(\mathbf{x}_j)$.

Example: An commonly used kernel for Support Vector machines is the function

$$K(\mathbf{x}, \mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y}) + 1)^2 \tag{2.31}$$

where $\mathbf{x}$ and $\mathbf{y}$ are in $\mathbf{R}^2$.

It can be easily shown that this function is equivalent to mapping the two dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ into a six dimensional space[3] and then performing a dot product. That is to say that there exists a transformation $k : \mathbf{R}^2 \to \mathbf{R}^6$ such that the inner product in the six dimensional space can be described by $k(\mathbf{x}) \cdot k(\mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y}) + 1)^2$.

Assume that $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ then

[3] This example actually is mapping into a five-dimensional sub-space of a six-dimensional space.

$$
\begin{aligned}
((\mathbf{x}\cdot\mathbf{y})+1)^2 &= (x_1y_1+x_2y_2)^2+2(x_1y_1+x_2y_2)+1 \\
&= (x_1y_1)^2+2(x_1y_1)(x_2y_2)+(x_2y_2)^2+ \\
&\quad 2(x_1y_1)+2(x_2y_2)+1 \\
&= \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_1x_2 \\ 1 \end{pmatrix} \times \begin{pmatrix} y_1^2 \\ y_2^2 \\ \sqrt{2}y_1 \\ \sqrt{2}y_2 \\ \sqrt{2}y_1y_2 \\ 1 \end{pmatrix}
\end{aligned}
\tag{2.32}
$$

$$
\Rightarrow k\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_1x_2 \\ 1 \end{pmatrix}
$$

2.2.1 Non-Separable data

As with the optimal separating hyperplane the non-separable case can be treated very much like the separable case, except that there are further constraints on the α_i:

$$0 \leq \alpha_i \leq C \tag{2.33}$$

2.3 Regression Estimation

In regression estimation our aim is to find a function $y = f_{\boldsymbol{\alpha}}(\mathbf{x})$ which best fits a set of training points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l) \in \mathbf{R}^N \times \mathbf{R}$. We do this by minimizing the error of the function $y = f_{\boldsymbol{\alpha}}(\mathbf{x})$ on the training set. The errors for Support Vector machines could be measured as the absolute error using the loss function

$$L(y, f_{\boldsymbol{\alpha}}(\mathbf{x}) = |f_{\boldsymbol{\alpha}}(\mathbf{x}) - y| \tag{2.34}$$

which gives us a total empirical risk over the training set of

$$R_{\text{emp}}(\boldsymbol{\alpha}) = \frac{1}{l}\sum_{i=1}^{l} |f_{\boldsymbol{\alpha}}(\mathbf{x}_i) - y_i|. \tag{2.35}$$

As Vapnik points out in [72] this error measure requires a perfect fit of $f_{\boldsymbol{\alpha}}(\mathbf{x})$ and we usually consider the more general so-called ϵ-insensitive loss function

$$|y - f_{\boldsymbol{\alpha}}(\mathbf{x})|_\epsilon = \{ \begin{matrix} 0, & \text{if } |y - f_{\boldsymbol{\alpha}}(\mathbf{x})| \leq \epsilon \\ |y - f_{\boldsymbol{\alpha}}(\mathbf{x})| - \epsilon, & \text{otherwise} \end{matrix} \tag{2.36}$$

which gives us the risk

$$R_{\text{emp}}(\boldsymbol{\alpha}) = \frac{1}{l}\sum_{i=1}^{l} |f_{\boldsymbol{\alpha}}(\mathbf{x}_i) - y_i|_\epsilon. \tag{2.37}$$

Function (2.35) is a special case of function (2.36) with $\epsilon = 0$. Setting $\epsilon \neq 0$ allows a change in accuracy of the approximation produced using the loss-function (2.36).

If we use the set of hyperplanes $(\mathbf{w} \cdot \mathbf{x}) + b$ as our set of available functions $f_{\boldsymbol{\alpha}}(\mathbf{x})$ and we are given the training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$ the problem is to minimize the empirical risk

$$R_{\text{emp}}(\mathbf{w}, b) = \frac{1}{l}\sum_{i=1}^{l} |y_i - (\mathbf{w} \cdot \mathbf{x}_i) - b|_\epsilon \tag{2.38}$$

which is equivalent to finding the pair w, b that minimizes the quantity defined by the slack variables $\xi_i, \xi_i^*, i = 1, \ldots, l$

$$F(\xi, \xi^*) = \big(\sum_{i=1}^{l} \xi_i^* + \sum_{i=1}^{l} \xi_i\big) \tag{2.39}$$

under the constraints

$$
\begin{array}{rcll}
(\mathbf{w} \cdot \mathbf{x}_i) + b & \geq & y - \epsilon - \xi_i^* & i = 1, \ldots, l \\
(\mathbf{w} \cdot \mathbf{x}_i) + b & \leq & y + \epsilon + \xi_i & i = 1, \ldots, l \\
\xi_i^* & \geq & 0, & i = 1, \ldots, l \\
\xi_i & \geq & 0, & i = 1, \ldots, l
\end{array}
\tag{2.40}
$$

Note that the parameter $\boldsymbol{\alpha}$ that parameterizes the function $f_{\boldsymbol{\alpha}}(x)$ will be replaced by two parameter vectors, each one associated with one of the slacks associated with a training example.

To solve this problem we have to put some restriction on $\mathbf{w}$ as there are many $\mathbf{w}$s that produce the same hyperplane. Our solution is to use the approach from pattern recognition and minimize $\mathbf{w} \cdot \mathbf{w}$.

This can be justified in the following way. Once ϵ is fixed we want the least complex approximation (to avoid overfitting) that will still approximate the training data well:

$$
\left| \frac{(\mathbf{w} \cdot \mathbf{x}_i) + b}{||\mathbf{w}||} - y_i \right| \leq \epsilon \qquad i = 1, \ldots, l \tag{2.41}
$$

The most complex approximation will be found when $\left| \frac{(\mathbf{w} \cdot \mathbf{x}_i) + b}{||\mathbf{w}||} - y_i \right| = 0$ where $i = 1, \ldots, l$. To find the least complex approximation the worst points $\mathbf{x}_i$ must be distance ϵ from their target values y_i. This is achieved by maximising that distance under the above constraint. The best way to do this is to minimize $||\mathbf{w}||$.

As in the pattern recognition case the solution can be found by finding the saddle point of the Lagrange functional

$$
\begin{aligned}
L(\mathbf{w}, \xi^*, \xi, \boldsymbol{\alpha}^*, \boldsymbol{\alpha}, C^*, \gamma, \gamma^*) = &\sum_{i=1}^{l} (\xi_i^* + \xi_i) \\
&- \sum_{i=1}^{l} \alpha_i [y_i - (\mathbf{w} \cdot \mathbf{x}_i) - b + \epsilon + \xi_i] \\
&- \sum_{i=1}^{l} \alpha_i^* [(\mathbf{w} \cdot \mathbf{x}_i) + b - y_i + \epsilon + \xi_i^*] \\
&- \frac{C^*}{2} (\mathbf{w} \cdot \mathbf{w}) \\
&- \sum_{i=1}^{l} (\gamma_i^* \xi_i^* + \gamma_i \xi_i)
\end{aligned}
\tag{2.42}
$$

This has to be minimized with respect to $\mathbf{w}$, b, ξ_i^* and ξ_i and maximized with respect to the Lagrange multipliers $C^* \geq 0$, $\alpha_i^* \geq 0$, $\alpha_i \geq 0$,$\gamma_i^* \geq 0$ and $\gamma_i \geq 0$ for $i = 1, \ldots, l$. The Lagrange multiplier C^* which is maximized makes sure that $(\mathbf{w} \cdot \mathbf{w})$ is minimized.

As this leads to a complicated optimisation procedure, a simpler approach can be used, if instead of minimising the functional (2.39) or the Lagrangian (2.42) under the appropriate constraints, one minimizes the quadratic optimization problem

$$
\frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C\big(\sum_{i=1}^{l} \xi_i^* + \sum_{i=1}^{l} \xi_i\big) \tag{2.43}
$$

for a given value of C and subject to the constraints (2.40). To find the desired vector $\mathbf{w}$ one has to find the coefficients α_i^* and α_i, such that the quadratic form (derived from the Lagrangian in appendix B.2)

$$
W(\boldsymbol{\alpha}^*, \boldsymbol{\alpha}) = -\epsilon \sum_{i=1}^{l} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{l} y_i (\alpha_i^* - \alpha_i) - \frac{1}{2} \sum_{i,j=1}^{l} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(\mathbf{x}_i \cdot \mathbf{x}_j) \tag{2.44}
$$

is maximized subject to the constraints

$$
\begin{aligned}
\sum_{i=1}^{l} \alpha_i^* &= \sum_{i=1}^{l} \alpha_i \\
0 \leq \alpha_i^* &\leq C, i = 1, \ldots, l \\
0 \leq \alpha_i &\leq C, i = 1, \ldots, l
\end{aligned}
\tag{2.45}
$$

This simplified approach coincides with the full approach when $C = C^*$.

The solution of this problem has some nice properties:

- Generally only a small number of α_i^*s and α_is will be non-zero.
- $\alpha_i^* \neq 0 \Rightarrow \alpha_i = 0$
- $\alpha_i \neq 0 \Rightarrow \alpha_i^* = 0$

2.3.1 Using a kernel in high dimensional feature space

As with the pattern recognition case we can use kernel functions rather than the standard inner product. We can use exactly the same kernel functions we use in the pattern recognition case.

2.3.2 Calculating the value of an unseen example

When all α_i^*s and α_is have been found the function is approximated by

$$
f_{(\boldsymbol{\alpha}^{0*}, \boldsymbol{\alpha}^{0})}(\mathbf{x}) = \Big(\sum_{i=1}^{l} (\alpha_i^* - \alpha_i) K(\mathbf{x}_i, \mathbf{x}) \Big) + b \tag{2.46}
$$

where b is a unique constant which minimizes the error over the training set:

$$
\min_b \sum_{i=1}^{l} \Big(y_i - \Big(\sum_{j=1}^{l} (\alpha_i^* - \alpha_i) K(\mathbf{x}_j, \mathbf{x}_i) \Big) - b \Big)^2 \tag{2.47}
$$

or

$$b_0 = \frac{1}{2} \min_{\mathbf{x}_j | \alpha_j > 0} (-y_j + \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) K(\mathbf{x}_j, \mathbf{x}_i)) + \min_{\mathbf{x}_k | \alpha_k^* > 0} (-y_k + \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) K(\mathbf{x}_k, \mathbf{x}_i)) \tag{2.48}$$

or

if we define

$$b_{0_j} = \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) K(\mathbf{x}_j, \mathbf{x}_i) \tag{2.49}$$

then

$$\min_j (\sum_{k=1}^{l} (| \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) K(\mathbf{x}_k, \mathbf{x}_i) + b_{0_j} - y_k | - \epsilon)) \tag{2.50}$$

will find the best b_{0_j} from our set of possible values.

In all cases $(\mathbf{x}_i, y_i)$, $(\mathbf{x}_j, y_j)$ and $(\mathbf{x}_k, y_k)$ are vectors from the training set.

Formula (2.47) has the disadvantage of being an optimization problem, which might take a long time to solve. Formula (2.48) uses one Support Vector from each side of the hyperplane and is easy to compute, yet only using two vectors might introduce a significant error. The last formula (2.49) is the one we have implemented as it is limited to a relatively small search space and finds the best result in this space. We give different formulae here, because inaccuracies in digital computers can make it necessary to take into account more than one Support Vector. Depending on the accuracy of the computer used one of these might be better than another. In addition one might not want to take into account those Support Vectors the α_i of which are equal to C or which are misclassified.

2.4 Generalization and Overfitting

By controlling the parameter C (or C and ϵ in the regression case) and the choice of kernel it is possible to control the generalisation ability of the Support Vector machine and also affect the complexity of the classifiers produced.

Overfitting can take place when the solutions are too complex, which is usually the case when they have been generated to only produce correct results for the training data and nothing else. This can be detected and controlled using cross-validation. If a classifier overfits on the training data in cross-validation the error rate on the "left out" examples will be high and can serve as an indication of overfitting. In that case usually a too rich set of kernel functions has been chosen and a simpler classifier should be used.

2.5 Summary

In this chapter we have introduced the origins of the Support Vector machine from the original Optimal Separating Hyperplane. We have shown in detail how the Support Vector machine can perform classification on separable data sets. The Soft Margin Support Vector machine has been introduced as a method of dealing with non-separable data as is the case when the data set is contaminated by noise. We have introduced the Support Vector machine for regression estimation and shown how its accuracy can be controlled. Finally we have discusses the issues of overfitting and the generalization ability of Support Vector machines in brief.

Chapter 3

Design and Implementation of the SVM

This chapter will introduce the design and implementation of our Support Vector machine library.

We introduce our basic design decisions, followed by an overview of the main functions of our implementation. We discuss the issues thrown up by our design decisions followed by details of the most important parts of our design.

3.1 Initial design

When writing the first implementation of the Support Vector machine, I soon noticed the need for a unified, modular and object oriented design. I also recognised the need for decomposing larger problems as the optimizers available could not handle these.

From this first implementation, I derived all the functional features needed and created a clean new design, which was implemented initially by myself and later with Jason Weston's help [64]. The implementation has since then grown significantly, but the basic design still remains.

In our implementation we aimed to create a powerful library of classes rather than a single application. This has the advantage that specialist applications can be written on the basis of this library and prototyping is easy.

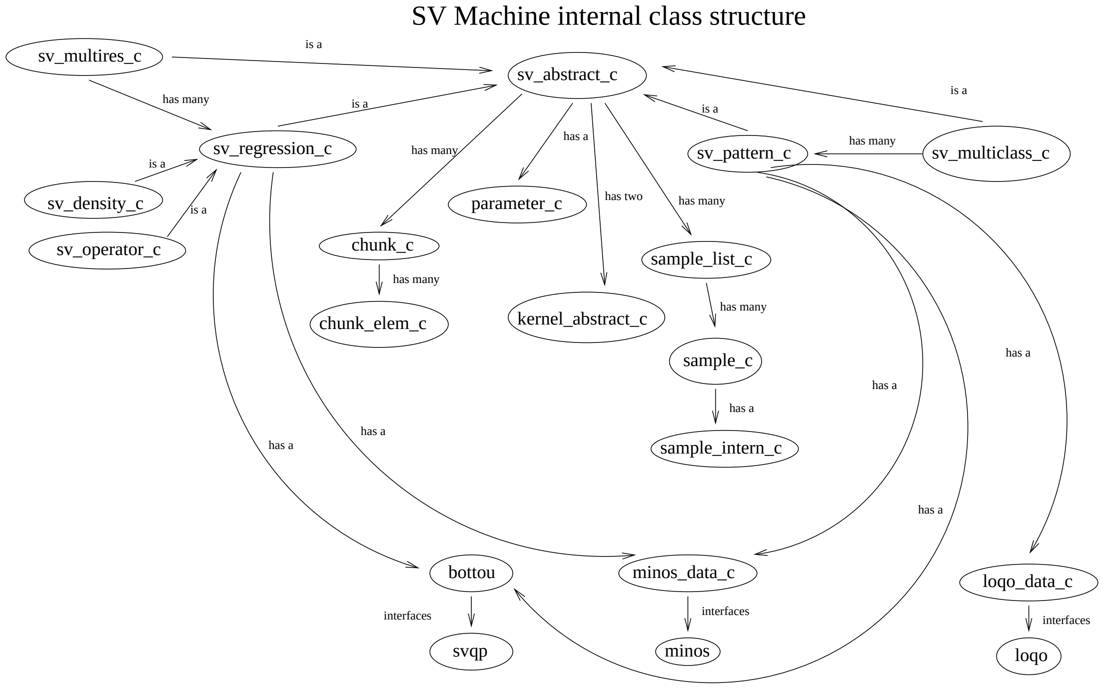

Figure 3.1: Classes for SVM

Figure 3.1 shows a simplified overview of the class structure for the Support Vector machine. In the following I will explain the main parts of this in detail.

3.2 Main functions of a SVM library

A Support Vector machine implementation should fulfil three tasks:

- Take a training set, a kernel function and some additional parameters and produce a classifier.
- Take a test set and give statistics about its members' classification.
- Take a single example and classify it.

Some additional functions are necessary:

- Save its internal state.

- Load a previously saved internal state.

These are necessary to implement check-pointing, so calculations can be stored after each step. In case of a system failure calculations can be resumed at the last internal state saved. They are also necessary for a production system to distribute and reuse classifiers.

These functions require

- storage procedures for examples
- kernel functions and kernel matrices for sets of examples
- parameters for the Support Vector machine
- optimizing routines

3.2.1 Issues

There are many issues of re-usability of code, generality of the implementation and technical necessities:

- Creation of multiple Support Vector machines has to be possible.
- A Support Vector machine should be able to take another Support Vector machine and improve on its performance with new training data, so the training set can be altered and the classifier recalculated.
- It has to be possible to have one copy of the training data and multiple Support Vector machines in memory that use that data. This must not affect the storage of Lagrangian multipliers which have to be stored separately for each Support Vector machine.
- The examples must be stored and referenced efficiently. Copying the examples for every object gives fast access, but a large overhead through copying and memory wastage. Referencing to some global list of examples from every object by storing an index for examples improves on overheads for copying and memory use, but potentially more overhead by searching for

an indexed example in the master list which can be very much larger is introduced. The latter is used in our implementation as memory is finite and small with respect to our task.

- Each Support Vector machine has to be independent of other Support Vector machines in terms of training set, kernel, C value and so forth.

- Any of the parameters can be changed at run-time on a particular instance of an Support Vector machine.

- Code has to be divided sensibly to make it possible to do pattern recognition and regression estimation problems, without replicating code and at the same time keeping the two problems separate.

- Memory use has to be considered to gain speed on virtual memory machines. In particular in the decomposition algorithms introduced later, having look-up tables of the parts of the non-linear objective function that stay constant in each iteration improves speed.

- The data types must be chosen carefully, to give maximum speed, which is vital given the size of the problems. For example dynamic, versus static allocation has to be carefully balanced. A compromise between fast, but unflexible arrays and linked lists for easy insertion and sorting has to be found.

- Wrappers for particular optimization packages or modules have to be written, which makes it possible to use different optimizers by just changing the wrapper and using the same Support Vector code.

- A trade-off between optimized code which is fast and inflexible and less heavily optimized code which is easily adaptable, easily readable, but slower has to be found.

- A data type has to be chosen to be used for attributes of training examples (binary, integer or floating point). For example, a particular problem might be of high dimensionality (130+ dimensions) but have attributes which are only single binary digits. It would waste memory storing each element as a floating point number. However to be as generic as possible IEEE `double` type floating point numbers have been used throughout.

3.2.2 Basic design

From these stipulations, we chose C++ as the language of implementation. C++ was chosen because it can link to most other languages through its parentage of C, it allows an object oriented design which is flexible and easy to maintain, and finally as a compiled language has very good performance.

With the above issues in mind we created our basic design. Firstly there is a generic Support Vector machine which is designed as an abstract factory and implemented as a C++ abstract class.[1] This allowed us to create Support Vector machine classes which inherited the interface and a set of basic and common functions from the generic Support Vector machine class. These Support Vector machine classes were for pattern recognition and regression estimation initially, although our implementation now also includes classes for solving linear operator equations, density estimation, a multi-resolution machine for regression[2] and especially importantly a wrapper machine that implements multi-class pattern recognition by creating a set of independent binary pattern recognition machines.

All of these machines have functions for producing a classifier and classification tasks. We have also added functions giving statistics about the classifier itself, which include measures such as the estimated VC dimension of the classifier and functions for saving and loading the internal state of the machine, that is the classifier produced.

3.2.3 Auxiliary Classes

For the successful implementation of a Support Vector machine, we identified a set of necessary support classes:

- *sample_list_c:* A list of examples for training or classification.
- *sample_c:* A single example from the above list.
- *parameter_c:* A set of parameters used to determine what Support Vector machine is to be used with which kernel and which chunking algorithm is to

[1] Many of the design ideas used in C++ program design can be found in [18]

[2] This takes a previous solution and improves on it, but this will not be covered in this thesis.

be used. This also includes the epsilon accuracy and C parameters amongst others.

- *chunk_c:* A set of examples with their associated alpha values.
- *chunk_elem_c:* A single example with its associated alpha value(s) in the above list.
- *kernel_abstract_c:* An abstract factory, which allows the implementation of kernel functions, and a set of predefined kernel functions from appendix A.
- *optimizer wrappers:* Special classes which provide a unified C++ interface to the different optimization programs used.

These classes and their relationships to each other and the Support Vector machine classes can be seen in figure 3.1 and will be explained in more detail in the following sections.

3.2.3.1 Storage of Examples

The need to store examples for training and testing combined with a modular design means that passing examples around the program must be handled efficiently and flexibly. The most efficient implementations in time and storage space do not allow for much flexibility. For this reason we have opted for an implementation using handles and reference counts. Implementations of this type are memory efficient if the example is used in many different places and also allows for modifications of the example in specific places that do not modify other occurrences of the example. The class used for this is called *sample_c* . This represents a good compromise between memory usage and speed.

Lists of examples are implemented in the same way using handles and reference counts. The lists are based on a parameterised list class, which is used throughout the program. These lists are of type *sample_list_c*.

Using lists of examples is useful as the examples will be used one at a time in the order they occur in. For training the optimizers memory is filled sequentially, even in chunking. For testing each example is taken in turn to be tested against the solution.

Apart from the usual access functions the list class also contains functions for loading, saving and scaling examples.

3.2.3.2 Storage of Support Vectors and their α-values

Support Vectors are recognised as such, because the α-value associated with them after the optimization process is non-zero. This means potential storage has to be allocated to every example to store its α-value. To do this a class has been created that effectively stores an example of type *sample_c* and a IEEE *double* floating point variable for the α-value. This class is called *chunk_elem_c* and is implemented using handles and reference counts.

A list class for these exists so solutions can be saved and loaded. This list is again derived from the parameterized list class and has all the functions of the lists of examples.

3.2.3.3 Parameters

Support Vector machines need a large set of parameters to run. This does not seem obvious from the theory.

The theory specifies a kernel, a box constraint C and in the case of regression an ϵ.

In practice more parameters than that are necessary for a program implementing Support Vector machines. The most important is to select what kind of Support Vector machine should be used. Pattern recognition and regression estimation are the obvious choices, but we have expanded this to also include linear operator machines and meta machines like the multi-class pattern recognition machine, which is assembled from multiple binary pattern recognition machines.

Next the kernel has to be chosen. As it is not easily possible to represent a kernel function as a simple parameter in a language like C++, we have decided to select a kernel function from a list by giving the index into the list and have allowed passing of certain numeric parameters to the kernel functions.[3]

[3] It is possible for the user to write their own kernel functions. Skeleton code is provided for

We also allow the parameters to contain scaling information for the data. This is not strictly necessary for the Support Vector machine, but is useful while experimenting.

Finally the parameters of the decomposition algorithms presented in the next chapter and constants for dealing with numerical inaccuracies are needed.

For the old-style decomposition a number of examples to add for each iteration is needed, for the new-style chunking a fixed working set size and a number of examples to be swapped are needed.

The group of parameters for numerical accuracy contains information as to which optimizer to use, when an example is to be considered a Support Vector, when an example lies within the margin and when the objective function is assumed to have stopped improving. These are generic parameters for the optimization process although some optimizers have more features than this.

Our parameter class also contains a menu driven interface which allows the user to easily set all necessary parameters to run a Support Vector machine. There are also functions for loading, saving and copying parameter objects.

3.2.3.4 Kernels and Kernel-Matrices

Good kernels are vital to Support Vector machines. Not only do they give the power to represent non-linear functions, but they also are one of the most computationally expensive parts of the Support Vector machine as will be exemplified in section 4.1.

Inefficient implementation of kernels will lead to massive performance degradation. We have decided on an object oriented approach, yet allow direct access to internal data structures of the kernel objects to improve speed. This is a compromise which is unavoidable.

Our kernel objects contain the actual kernel function as well as a matrix of kernel values for certain examples. The kernel class always contains the examples the

this, but the user needs some basic understanding of C++ and it is necessary to recompile the code, which is easy.

optimizer is currently working on. In this way the optimizer's evaluation of the objective function is as efficient as possible.

Our kernels are implemented as siblings inherited from one virtual abstract base class. In this way all kernels seem the same to the Support Vector machine. This makes adding new kernels simple and means that prototyping can be done in a matter of minutes.

A kernel object can be passed a list of examples *sample_list_c* or *chunk_c* and calculate the dot product for each pair of examples. It then acts as a lookup table for dot products. It can also calculate a dot product on the fly, which is necessary for test cases.

The data that has to be stored in a kernel object is:

- A list of examples.
- A list or matrix of dot products for those examples.

Kernel objects are usually created for subsets of the training set when using the decomposition methods described in chapter 4, rather than for the whole training set at once for reasons of both speed and memory usage. Several small k^2 matrices are quicker to calculate than one large n^2 matrix, if $k << n$.

In the pattern recognition case, we need to calculate $y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ for each i, j. As the classification for each example (y_i) is constant we can add this into our kernel function and eliminate many of the calculations, hence a matrix storing those vales is created in the pattern recognition case. A virtual function called *calcK* is the actual kernel function for each child of the abstract class, which is called to create the kernel matrix. *calcK* is also necessary for classifying new test vectors after learning has taken place.

If the chunk is over a certain size, a matrix of $K(\mathbf{x}_i, \mathbf{x}_j)$ or $y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ can not be created for each i, j pair, as so much virtual memory is used that the speed of looking up values is slower than calculation on the fly. Instead, the kernel object copies the values it needs to calculate the kernel values into arrays of examples and their target values and uses these to calculate the kernel values on the fly. As arrays are used this should improve performance even if the problem involves page swapping and even though the values are calculated on the fly.

3.2.3.5 Optimizers

Initially we used the free loqo package by Rob Vanderbei [68]. This has severe limitations in that only one optimization problem can be solved per program run. As it is desirable to be able to solve several optimization problems in a program run, either for the decomposition techniques or multi-class techniques introduced later, we decided on using the commercial MINOS package from Stanford University [39] instead. The MINOS package comes with source code in Fortran, making its integration into our software possible due to our use of C++. MINOS is especially designed to handle non-linear optimization tasks over many variables with many constraints. In this context the operations research community regards about 200 variables as a large amount, leading to the necessity of decomposition methods for the Support Vector method problem. We later included optimization packages by Leon Bottou [51] and Alex Smola [51] into our software to enable us to distribute a free version, however both of these do not reach the performance of MINOS, which has been used for all experiments in this thesis. MINOS performs better in terms of speed and accuracy, as well as giving helpful warnings when the optimization process does not produce a correct solution.

Before an optimizer is run it has to be given an objective function to minimize or maximize[4], a set of constraints and a set of variables to optimize over[5]. This has to be set up for every run of the optimizer when using decomposition methods and requires careful data handling. For each optimizer used there is a different set up procedure for each of the Support Vector tasks. This means that although the software has a modular design it is quite a large task to add a new optimizer.

3.3 Summary

We have shown the design of our class library for Support Vector machines. This includes the main problems faced, the issues involved in creating such a library

[4]This function is called many times and has to be programmed to be very efficient.

[5]The constraints contain C and if applicable ϵ is contained in the objective function of the optimizer.

and its most important aspects.

With our implementation we have created a test bed for further investigation of the Support Vector machine. Our implementation allows fast prototyping of specialist systems and allows the evaluation of problems using the Support Vector machine. We have opened up the Support Vector machine to many more researchers and at the moment about 500 institutions are using our software. In chapter 7 we present some experiments using our implementation.

Recently other implementations have become available, but none of them have the capabilities to deal with large data sets and both multi-class pattern recognition and regression estimation. Neither are they available as easy to use libraries for integrating into the users own software. So our implementation still remains the most universal package.

Chapter 4

Decomposition of the SVM Problem

The theory introduced so far lays the foundations for implementing a Support Vector method algorithm on a computer. However the theory does not consider computers explicitly, although without them it could not produce useful results. The theory of the Support Vector method learning algorithm makes assumptions which can not be fulfilled by existing modern digital computers.

The first assumption is the implicit one that computers do not have limited storage. The optimization problem is formulated in a way that requires the problem to be solved in one step. To be efficient the kernel matrix has to be stored in memory, which is impossible for larger problems. Theoretical decomposition algorithms have been developed to deal with this [64] and [41], but heuristics have to be applied to make them converge in a reasonable time. Although in theory the Kühn-Tucker conditions could be checked, we do not check them as they contain equality constraints which are hardly ever met exactly on a computer.

The second assumption is that of near instant solutions. The process of solving a mathematical optimization function like formula (2.9) can take days, if the number of training examples is large. Again if the problem is decomposed into smaller sub-problems solutions can be reached in a reasonable time.

A third assumption made by the basic theory and the decomposition algorithms is the numerical accuracy. Determining which examples are Support Vectors and termination of the decomposition algorithms depends on a level of precision which is not present in modern digital computers.

4.1 Speed and Storage requirements

Optimizers have to evaluate the objective function during the optimization process frequently. For the Support Vector machine this means that the functions (2.19) or (2.44) have to evaluated. To do this efficiently the static parts of the function should be evaluated before the optimizer is started. In pattern recognition this is $y_i y_j K(x_i, x_j)$ and in regression estimation $K(x_i, x_j)$. So there are two main sets of data that have to be available for training a Support Vector machine:

- Training examples
- Kernel matrix

The training examples are needed to calculate the kernel matrix and in the original setting are only needed once. It is useful however to keep them in memory for the decomposition algorithms that follow as they occupy very little memory in comparison to the kernel matrix.

An example often used in the Support Vector machine literature is the USPS hand-written digit database [52]. The database contains approximately 7000 training examples each of which has 256 dimensions in input space.
In the traditional setting the training data would need 7000×256 values. As we use IEEE `double` type floating point variables which use 8 bytes each, this would take approximately 14MB of memory.
As the kernel matrix is symmetric, it can be represented using $\frac{7000^2+7000}{2}$ values in practice. Using IEEE `double` type floating point variables this would require approximately 187MB of memory, which is impractical.

Calculating the kernel matrix on the fly would not be practical either. Assuming 60ns memory which is common in today's computers, accessing the training examples for one evaluation of the objective function would take 7000 examples $\times$ 256 dimensions $\times$ 120ns which is approximately 25 minutes. To this the overhead of calling functions and the necessary 12 billion multiplications have to be added.

Both methods are not practical which in itself necessitates some form of decomposition of the problem. Another observation to make is that most optimizers speed depends on the number of variables optimized over. For problems like the Support Vector machine the time requirements are $O(n^3)$ in the worst case. The solution is to give the optimization routines only a subset of the data at one time. This is known as chunking or decomposition. We use two styles of chunking in our program. Although both yield roughly the same results, it is problem dependent which is faster. However they both outperform the original method in terms of speed and for very large data sets are the only way to handle the problem.

Example: For speed comparisons we will use a pattern recognition example. We extracted the digits 2 and 7 from the USPS database. This gave us a training set with 1376 examples roughly evenly split between the two classes. We measured all timings on a 433MHz Alpha with 512MB of RAM. Using the original Support Vector method the training time was about 37 seconds. The objective function was evaluated 150 times by the optimizer during this process. The kernel matrix needs $\frac{1376^2+1376}{2} \times 256$ multiplications or approximately 242 million for a polynomial kernel.

4.2 Old style chunking

This form of chunking relies on a very simple fact: In the final solution only the Support Vectors have any effect on the objective function of the optimization process. We can therefore use the following iterative process of optimization. Take a chunk of data of a size we can handle, find a solution for the sub-problem, then remove all Non-support Vectors and add a set of new data and solve this new problem; the old alpha values are kept. This process is repeated until optimality is reached. The size of the set which is presented to the optimizer in most cases will stay reasonably small. Unfortunately to my knowledge this method has not been proven to converge. As it is very similar to the next method described, it is highly likely that the convergence will only be provable in the limit. Even if convergence is proven, experiments still show that in some cases due to numerical inaccuracies of modern digital computers this method never converges, unless heuristic stopping conditions are used. This manifests itself in two vectors alternatively becoming Support Vectors and being removed and reintroduced to the working set of vectors.

Example: To show the advantage of this approach we will return to the example used earlier. If we use an initial chunk size of 300, the problem is decomposed into 5 runs of the optimizer. Therefore five kernel matrices have to be calculated. They are all of variable size (300, 110, 52, 46, 45) and need a total of approximately 14 million multiplications. The objective function is evaluated 353 times. The total time needed was approximately 6 seconds. We observe a performance increase of a factor of approximately 6 here.

The heuristics used here are not to add examples randomly, but only to add examples which are on the wrong side of their margin and to finish when no improvement greater than a small number[1] over one iteration is made in the objective function.

```
start with an empty chunk
do
    add n examples violating the constraints to the current chunk
    optimize over the current chunk
    remove non-Support Vectors
while there is a noticeable change of the objective function
```

It should be noted that this style of decomposition is very similar to the one described in [42]. The difference is that our method uses a variable working set size, whereas the method proposed in [42] uses a fixed size working set. The fixed size must be big enough to contain all Support Vectors or the algorithm will fail. This is a problem because the number of Support Vectors can not be determined a priori. Our approach does not suffer this problem.

[1] In theory this should continue until the improvement is zero of course, but as this does not work due to numerical inaccuracies we have introduced a so-called "Objective function zero tolerance", which by default is set to 10^{-7} in the program.

4.3 New style chunking

The newer style chunking introduced by Osuna [41] is a logical extension of the old style chunking. The problem with the old style chunking arises when the number of Support Vectors grows larger than the computer can optimize in one chunk. Osuna suggests to fix the size of the chunk to n and swap out $k \leq n$ vectors, but remembering their alpha values and including them in the objective function calculation, but not the optimization process. Again numerical inaccuracies can make this method oscillate and never converge, so again we have added heuristics to be used as stopping conditions. Heuristics are also needed to decide which examples to keep, remove and add to the chunk at each stage of the process. The user has to determine the chunk size or working set size and the number of vectors to be swapped out in advance.

In this method the savings on storage are greater than in the old style method, but due to the more constrained optimization, speeds can be lower.

Heuristics The heuristics for this method have to be far more elaborate than for the old style method, not only does there have to be a careful evaluation of which examples to add, but also which examples to remove. No heuristics are described in Osuna's paper, but obviously necessary for all but the most trivial problems. The methods used here were developed by ourselves.

The chunking is done in three distinct phases. The first two phases each run until the optimization process shows no further improvement, whereas the last phase simply runs through all data points one final time to remove any major inconsistencies.

```
start with a chunk containing the first n examples
repeat
    do
        optimize over the current chunk
        remove k examples from chunk according to...
            ...the current phase's heuristic
        add k examples to chunk according to current phase's heuristic
    while there is a noticeable change of the objective function
```

```
    move to next phase
until all phases done
```

4.3.1 Phase 1

Initially the first n examples are taken to fill the working set and the optimization problem is solved for that subset. Following that an iterative process is started which continues until the objective function of the optimization process does not improve within a certain tolerance. When there is no more improvement phase 2 is entered.

4.3.1.1 Removing examples from the working set

After the initial optimization the working set is analysed as to which examples should be removed and a preference list of examples is set up. Each example is assigned a score and the examples with the lowest scores are removed. If an example is not a Support Vector, i.e. its associated alpha value is below a threshold[2] a score of -1 is assigned. If an example's associated alpha value is on the C boundary, i.e. hits the box constraint a score of -0.5 is assigned. If the example is a Support Vector within the bounds of the box constraint the change in its alpha value is measured and the difference between its last and current alpha value is given as the score, unless this value is below the threshold for determining Support Vectors, i.e. the alpha value has hardly changed, in which case the score is zero. This swapping out policy is the same for all three phases.

4.3.1.2 Adding examples to the working set

Next the remaining examples in the training set are scored to find suitable examples to fill the gaps left in the working set by removing examples from the last optimization run. In this phase the examples with the lowest score are inserted.

In the pattern recognition case an example receives a zero score if its alpha value lies on the C boundary, i.e. the box constraint, or if it is correctly classified and

[2] This "SV zero threshold" is set to 10^{-16} by default.

does not lie within the margin. If an example is correctly classified but lies within the margin it receives a score equal the negative value of one minus its distance from the hyperplane ($|d| - 1$). If an example is misclassified it receives a score equal to its negative distance from the hyperplane minus one ($-|d| - 1$). This is visualized in figure 4.1.

In the regression case if the absolute difference between the target and the current estimate of a point is greater than epsilon the score is the negative value of that difference. If the difference is less than epsilon the score is 1 over the difference.

After this scoring the relevant examples are swapped. If the set of removable examples contains less than 25% Support Vectors, these Support Vectors are not swapped out, but left. Only if the number of Support Vectors to be swapped out is more than 25% are they swapped out.

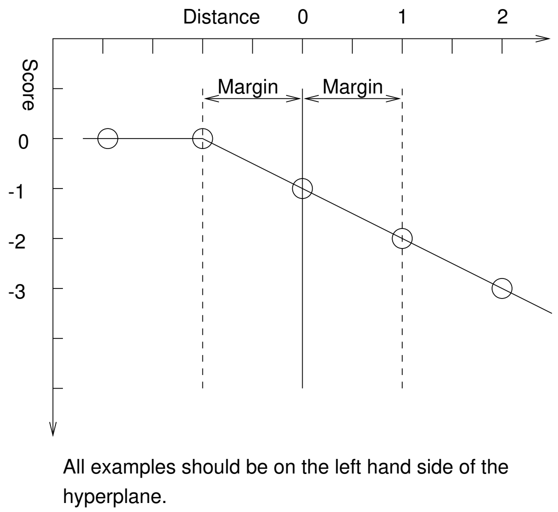

Figure 4.1: Swapping in policy for pattern recognition in phase 1 visualized

4.3.2 Phase 2

The second phase starts when there is no more improvement in the first phase. It is very similar to the first phase. The only difference is the scoring for the swapping in policy.

There are two scores for each example which are multiplied together to form a final score. The examples with a lower score are swapped into the working set first.

In the pattern recognition case the first score takes into account the distance from the margin. If an example is correctly classified and the distance $|d| > 2$ it is given a zero score. If it is correctly classified and the distance $|d| < 2$ it is given a lower score as it seems to be tending to be a Support Vector. A larger distance of the example from the margin gives a lower score. If the distance $|d|$ from the hyperplane lies between 0 and 2 and the example is correctly classified the score is $||d| - 1| - 1$.

If the example is incorrectly classified the score is smaller than before in all cases and exactly $-|d| - 1$.

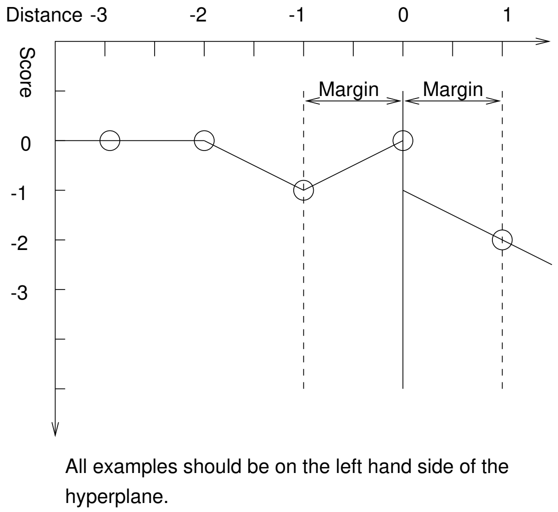

Figure 4.2: First swapping in score for pattern recognition in phase 2 visualized

In the regression estimation case the first score again takes into account the distance from the hyperplane. If the difference between the target and the current estimate of a point is less that epsilon the score is zero, otherwise the score is calculated as how far out the estimate is by taking the negative value of subtracting epsilon from the difference and then dividing by epsilon to get a relative "badness".

For both pattern recognition and regression estimation the second score is assigned according to the alpha size. If the value of alpha is close to 0 or close to C[3] we expect it to hit one of those values very fast and not be very important

[3]When C is set to infinity, it is assumed to be 10^{20} for the purpose of this calculation.

with respect to the optimization process. Hence the second score is $1 - \frac{|\alpha - \frac{C}{2}|}{\frac{C}{2}}$. For the misclassified pattern recognition examples two is subtracted after the two scores are combined.

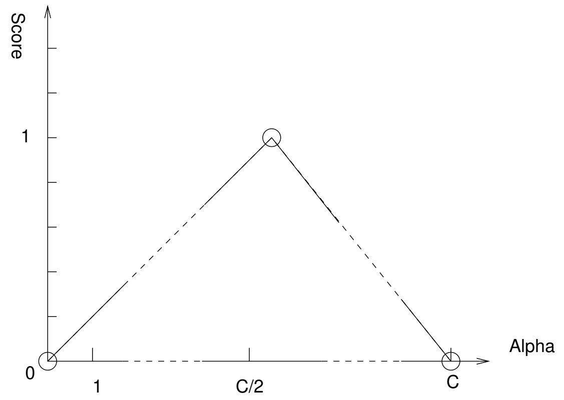

Figure 4.3: Second swapping in score for phase 2 visualized

4.3.3 Phase 3

Finally if phase 2 can make no more improvements phase 3 is entered. In phase 3 all examples are swapped into the chunk in turn exactly once. This allows one final check on the solution.

Example: To demonstrate how well this approach works we return to the USPS digit example. Using an initial working set size of 300 and a swap size of 250, the training took 28 seconds. There were 10 runs of the optimizer each of which required a 300 example kernel matrix. This resulted in approximately 115 million multiplications. The objective function was evaluated 549 times. Chosing a working set size of 100 and a swap size of 50, training took 18 seconds, however there were 32 runs of the optimizer and 1142 evaluations of the objective function, but only 41 million multiplications for the kernel matrix. The performance increase in both cases is not as extreme as with the old-style chunking, yet the advantage of the new style chunking is that it can be applied to problems with far larger numbers of Support Vectors.

4.4 Choice of chunking method

If the size of the training set is below 500 examples it is usually not necessary to use chunking at all, however above that chunking does have significant benefits. The choice of the chunking method is largely dependent on the number of Support Vectors in the final solution. If the number of Support Vectors is small, i.e. can be optimized over in one chunk, the old style method is faster. However if the number of Support Vectors is large the old style method is infeasible. This means that a priori no decision is possible without detailed knowledge about the data. Therefore the new style method is preferable as a general rule.

Method	Time	Multiplications for kernel matrix	Evaluations of objective function	Optimizer runs
Original method	37s	242 mio.	150	1
Old style chunking	6s	14 mio.	353	5
New style chunking 300/250	28s	115 mio.	549	10
New style chunking 100/50	18s	41 mio.	1142	32

Table 4.1: Overview of performances

From the table it is clear that the original method needs a far larger kernel matrix which takes more time to compute than any of the chunking methods, because it requires far more multiplications and associated with that, more memory access and function call overheads.

The evaluation of the objective function is cheaper for smaller chunk sizes, yet it is called more often. The old style chunking has a distinct advantage in this specific example because there were only 45 Support Vectors and that on average it had a small chunk size.

The figures clearly demonstrate that our chunking methods significantly improve training speeds.

Finally the quality of the solution is important. Due to the heuristic stopping conditions an optimal solution is not guaranteed. Although we have observed slightly different alpha values of the Support Vectors for both chunking methods from the non-chunking approach, the actual set of Support Vectors has always been the same. This over a large number of trials has convinced us of the plausibility of the heuristics used.

4.5 Numerical accuracy

We have briefly touched on the numerical accuracy in digital computers which is known to be poor. All modern digital computers have limited accuracy and most can only represent binary numbers internally. This leads to problems with even the most basic numbers. Binary floating point numbers have completely different properties than decimal floating point numbers. The most basic example is that a binary floating point number of finite accuracy can not represent the decimal 0.3. This problem is exacerbated when many operations are performed on a set of numbers. This is a particular problem with decomposition of the Support Vector method as described earlier in this chapter.

In addition to this it is possible for two identical programs to produce different results on different computers. In one example an Alpha machine produced a result with 350 Support Vectors, whereas an Intel machine's result had 450 Support Vectors. This is purely due to the way these computers treat floating point numbers. Alphas can perform operations in hardware on IEEE `double` types, whereas Intel processors have to use two separate operations for IEEE `double` types and concatenate the two results afterwards.

In addition to differences between computers there are algorithmic differences. For our implementation, we use three different optimizers not written by us. All three optimizers produce different solutions to all but the most simple problems. All optimizers have to be told what values to regard as zero, some of the optimizers even have an accuracy setting. This is an additional source of error which is inevitable. The problems directly thrown up by this are:

- When is an example a Support Vector?
- When is an example on or inside the margin of the optimal separating hyperplane?

The decision whether or not an example is a Support Vector affects the decomposition algorithms in what vectors to swap out of the working set. The decision whether an example lies within the margin affects the optimality constraint and whether to swap the example into the working set.

We have had to make these decisions based on our comparisons between the chunking and non-chunking methods and use the following parameters:

- *SV zero threshold:* This determines by how much a vector's alpha value is allowed to deviate from zero before becoming a Support Vector. Usually we choose this as 10^{-16}. This should in theory be chosen on the basis of the magnitude of the kernel values, but the default value lies near enough to the limitations of the optimizers used to make the optimizers limitations more of a restriction than the kernel values. This does of course affect results when it is chosen to be large.

- *Margin threshold:* This determines whether or not a vector lies within the margins of the separating hyperplane. If a vector lies within this distance inside the margin, it is considered to lie outside the margin. This is used in the decomposition techniques. Usually we choose this to be 0.1. Although this value is large it does not influence our final decision about a point, it only affects the decomposition heuristics.

- *Objective zero tolerance:* This is used to determine whether the objective function has changed between different iterations in the decomposition technique. A change less than this value is considered as no change. Usually we choose this to be 10^{-7}.[4]

These values work well on all examples we have seen, but can be changed interactively, if necessary. It might be necessary to change these values for certain problems, certain kernels and certain micro-processors.

[4] The name stems from the fact that a zero change in the objective function of the optimisation process is sought but we allow a certain tolerance.

4.6 Summary

In this chapter we have shown the serious problems of speed, storage and accuracy the basic Support Vector machine faces with large datasets. We have presented two decomposition techniques for dealing with large datasets. We have introduced heuristics to make the proposed decomposition techniques work in practice and we have addressed the issues of numerical accuracy involved therein.

The methods described are implemented in our program, which was the first to combine all of these techniques. E. Osuna [41] had implemented a version of the new style chunking before us, but the implementation was not made public nor were details of it published at the time. The software developed by T. Joachims [30] shortly after us is mainly suited to problems involving sparse vectors, only implements one form of decomposition and is only capable of pattern recognition.

The decomposition approaches and heuristics described here have not been *proven* to converge on a solution. A proof of this kind would have to prove the convergence of a series of optimization sub-problems to the solution of the complete optimization problem. Another approach would be to run a large set of experiments with large data sets from many different domains and evaluate the solutions produced. The experiments should include datasets with small and large numbers of dimensions in input space, evenly and unevenly balanced numbers of examples per class, attributes from the same type or differing types. The selection of datasets should cover the whole range of likely applications. Once the optimal hyperplanes for these datasets have been found it is possible to check them against the Kühn-Tucker conditions and the Slater condition. If these conditions are satisfied the hyperplane is indeed the optimal one which means the optimization problem has been solved by the decomposition technique. In the same way it is of course possible to check a single solution to a specific problem, thus making it possible to check the heuristics for a single case and prove that the solution is correct.

Chapter 5

Multi-class

Traditionally Support Vector machines for pattern recognition are binary classifiers. Many problems however are classification problems of more than two classes.

We will show a range of methods using binary classifiers to classify more than two classes with special emphasis on Support Vector machines.

5.1 n-Class Pattern Recognition

The problem in n-class pattern recognition is to find a function that can classify a data point into one of n-classes. There are formulations of the problem in which a point may be classified into multiple classes at the same time, but this is unusual and not addressed here.

The task is to construct a decision function given a sample of l points from an unknown function

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$$

where $\mathbf{x}_i \in \mathbf{R}^N$ are the data points and $y_i \in \{1, \ldots, n\}$ are the classes of the data points.

5.2 Multi-class SVM

Multi-class classification can be done in three major ways using Support Vector machines: Using regression estimation Support Vector machines, using a set of independent binary classification Support Vector machines with a special voting scheme or using a special Support Vector machine which creates a set of dependent hyperplanes with a voting scheme for classification. Methods that have also been used are density estimation Support Vector machines [77] and transduction methods [50].

5.3 Binary SVM

The Support Vector machine developed at Royal Holloway [51] implements a variety of multi-class schemes based on a set of independent binary Support Vector machines. The methods developed for this are described in this chapter in detail.

5.3.1 Common to all approaches

All of the following multi-class classifiers using binary Support Vector machines have the following in common:

- The multi-class Support Vector machine acts as a wrapper for a set of binary pattern recognition Support Vector machines that provides extra functionality by allowing the use of more than two classes.
- This is achieved by converting the original training set into multiple training sets containing just two new classes each (how the original classes are converted to the new classes depends on the scheme used and may be different for each new training set) and training a pattern recognition Support Vector machine for each training set.
- Classification is done by presenting the new example to all of the Support Vector machines or a subset in turn and applying a voting scheme.

Currently three methods of multi-class classification have been implemented:

- One against all others
- One against one
- Hamming error-correcting codes

Some of these methods can end in a draw between one or more classes using just the binary classifiers. In this case of indecision it is possible to return all possible classes; however in many cases a clear decision for one class is essential and additional information has to be used to distinguish one class from the others. This final decision is only made between the equally likely classes, not all classes, on the basis of the measures mentioned in the next section.

We will use the following example throughout: The original training set has n_1 examples of class c_1, n_2 examples of class c_2, n_3 examples of class c_3 and n_4 examples of class c_4.[1]

5.3.2 Additional measures

Some binary classifiers have a measure, which gives an indication about the accuracy of their classification. This can be used when a decision has to be made between individual binary classifiers and choose one of their decisions. Some classifiers return a quality value from a pre-defined range, which can be seen as a probability. This is the preferable case. The Support Vector machine however has two possible quality measurements; one based on the individual case, and the other that applies to the machine as a whole independently of the individual case:

- The distance of an example from the hyperplane can be seen as one indicator: the further away an example is, the more the Support Vector machine is sure of its decision.

[1] The pictures show just three classes.

This however throws up the problem of chosing a metric in which measure the distance. It is usual to calculate the distance of a point from a hyperplane defined by $(\mathbf{w} \cdot \mathbf{x}) + b = 0$ as

$$\frac{(\mathbf{w} \cdot \mathbf{x}) + b}{||\mathbf{w}||} \tag{5.1}$$

However this gives the absolute distance. Another measure is to calculate the relative distance in terms of the margin. This means distances of more than 1 indicate a high level of certainty about the classification:

$$(\mathbf{w} \cdot \mathbf{x}) + b \tag{5.2}$$

Experimentally we have found the second method to be preferable. In this way all hyperplanes have their Support Vectors at distance 1 for multi-class classification using several binary machines.

- Another approach is to estimate the VC-dimension of a machine [71]: the lower the VC-dimension, the better the generalisation ability of that machine in comparison to other machines on that example and the higher the probability of that machine being right.

5.3.3 One against all others

In this method the Support Vector machines are trained to vote for or against a particular class. It creates as many training sets as there are classes. The training sets all contain the same examples, however in the first training set the first class' classification is changed to 1 and all other classes classification is changed to -1 and so on.

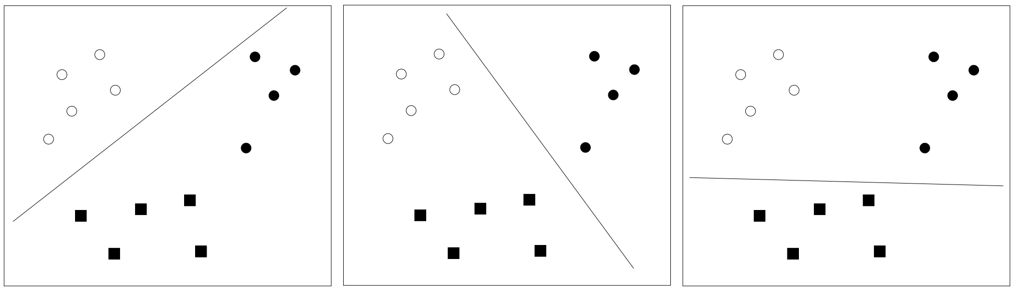

Figure 5.1: one-against-all-others multi-class examples with three classes

Given our example we would create four training sets. All training sets contain $n_1 + n_2 + n_3 + n_4$ examples. In the first Support Vector machine the class label c_1 is replaced by class label 1 and the class labels c_2, c_3 and c_4 are replaced by class label -1; this machine votes for or against class c_1. The second Support Vector machine is given a training set where class label c_2 is replaced by label 1 and class labels c_1, c_3 and c_4 are replaced by label -1; this machine vote for or against class c_2. The other two machines' training sets are created in the same way.

Given an example there are three scenarios of voting:

- There are no votes for any class. In this case the machine classifies the example as un-classifiable. You might randomly assign a class or assign the class with the highest frequency in the training set. Alternatively the class with the smallest distance from the hyperplane might be returned (currently implemented). This corresponds to the least objection to that class.
- Only one machine votes for a class. This class is returned.
- Many machines vote for a class. The class with the highest distance of the example from the hyperplane is returned. Here again there might be better measures. It would for example be possible to return the class the machine of which has the lowest estimated VC-dimension.

This method has the advantage that the number of classifiers is linear with respect to the number of classes.

5.3.4 One against one

In this method Support Vector machines are trained to distinguish between just two classes.

A training set for each combination of two classes (n classes give $(\sum_{i=1}^{n} i) = \frac{n(n-1)}{2}$ training sets). The training sets here usually are different sizes for each machine and each machine decides between two classes.

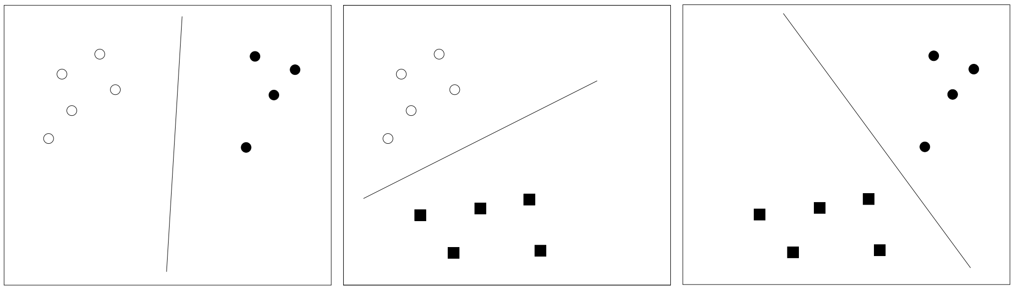

Figure 5.2: one-against-one multi-class examples with three classes

For our example one machine for distinguishing between classes c_1 and c_2 relabelled as 1 and -1 would be created. Then machines for c_1 and c_3, c_1 and c_4, c_2 and c_3, c_2 and c_4, c_3 and c_4.

The voting scheme here counts the votes for each class given by the machines and leads to the following scenarios

- One class has the highest number of votes, so this is returned.
- Many classes have the same highest number of votes. The class with the highest total distance of the example from the hyperplanes in all machines voting for it is returned. Here again there might be better measures. It would be possible to return the class the machine of which has the lowest estimated VC-dimension, or the class with the highest single distance in any machine.

The advantage of this method lies in the small training sets for each machine and less indecisions than in the "one against all others" method. The problem that appears using this method happens when class c_n has more votes than class c_m, but the Support Vector machine deciding between class c_n and class c_m votes for class c_m. We still return class c_n in this case as this is not easily resolved when more than two classes are involved.

5.3.5 Hamming Error Correcting Codes

This method codes all classes using Hamming error correcting codes for correcting one error. There is one code word for each class. One Support Vector machine is trained for each bit in the code. Classification is achieved by running an example through all Support Vector machines and then decoding the class from the Hamming error correcting codes.

The number of Support vector machines needed for a n-class problem is $d + m$, where m is the smallest integer such that $\text{ceil}(\log_2(n)) \leq 2^m - m - 1$ and $d = 2^m - m - 1$. The number of Support Vector machines is always less than the maximum possible number of classes for the Hamming error correcting code selected.

For a description of generating and using Hamming error-correcting codes see [24], pages 39ff.

Using our example we can code the classes c_1, c_2, c_3 and c_4 as $0, 1, 2, 3$. The binary coding of this needs two bits, but a Hamming error correcting code to deal with two bits does not exist. The next highest Hamming error correcting code has to be used, which contains four data bits and needs three parity bits, so the total code has seven bits and we have to create a Support Vector machine for each of these seven bits. As Hamming error-correcting codes do not exist for every number of data bits, this method is only efficient for a certain numbers of classes, e.g. 2, 16, 2048, 67108864 etc. If there are fewer classes in the problem than the code can handle, the machine might return class labels which do not exist. We know of no optimal solution, so we use the option to return the most frequent class in the training set or to allow for indecision.

Other problems with this approach lie in the fact that the binary classifiers sought depend very much on the ordering of the classes. Different orderings and thus different codes can lead to highly different results.

5.4 Other approaches

In this section we look at approaches we have not implemented. These methods require some prior knowledge of the data and require further research before they can be used efficiently. We use the same example as previously.

5.4.1 One against the rest

This method corresponds to using a binary unbalanced decision tree. On the first node the first class is separated from the remaining classes, on the second node the second class is separated from the remaining classes and so on.

Given our example we would create a new training set for the top-level node containing all examples, where the class label c_1 is replaced by class label 1 and the class labels c_2, c_3 and c_4 are replaced by class label -1; this machine separates class c_1 from the rest. The second level node separates class c_2 from class c_3 and c_4, so the training set would contain $n_2 + n_3 + n_4$ examples from the classes c_2, c_3 and c_4, where the class label c_2 is replaced by label 1 and class labels c_3 and c_4 are replaced by label -1. The third machine separates class c_3 from c_4 and contains $n_3 + n_4$ examples from the classes c_3 and c_4, where class label c_3 is replaced by label 1 and class label c_4 is replaced by label -1.

The classification scheme here is very easy. Given a new example it is first decided whether it belongs to class c_1 or the rest, if it does that decision is returned, if not it is decided whether the new example belongs to class c_2 or the rest, a decision is made and so on.

This method has the advantage that the average number of binary classifications needed is usually half the number of classes minus one. The advantage or disadvantage depending on the point of view, is that no indecision is allowed.

The main problem associated with the method is that the order of classes has to be chosen a priori. This means that classes higher up in the decision tree potentially have a higher chance of being selected. Approaches to this include trying to balance the number of examples for each class in the binary classifiers, i.e. the largest class is selected as c_1, the second largest as c_2. On the other hand it might be easier to separate very small sets from large sets, in which case the smallest class is selected as c_1.

5.4.2 Half against half

This method corresponds to using a more balanced decision tree. At the first node the classes are split into two halves, at the second level into quarters and so on.

In our example we might choose the top-level node to split classes c_1 and c_2 from classes c_3 and c_4. At the second level we would split c_1 from c_2 and c_3 from c_4.

The decision here would just follow the tree down to a leaf node and return the appropriate class.

The advantage is the fixed number of classifiers to be evaluated for each example (square root of the number of classes). The disadvantage as with the other tree method is to decide on the splitting method. If very dissimilar classes are grouped together at any stage, classification at that stage would be very hard. One might split the data into roughly equal number of classes for each half or roughly equal number of examples for each half.

5.4.3 Regression methods

Regression techniques can be used to perform multi-class classifications. This is done by observing the fact that all discrete classes can be seen as the result of a function which is a function which assigns discrete values to ranges of outputs of a real function. For example given the classes c_1, c_2, c_3 and c_4 it is possible to construct a real function which returns values less than 1.5 if the class is c_1, values between 1.5 and 2.5 if the class is c_2, values between 2.5 and 3.5 if the class is c_3 and values greater than 3.5 if the class is c_4. Classifiers of this type can be constructed using regression Support Vector machines. The problem here is that the classes c_1 and c_4 have a much larger range of real values that map to them. Even extending the ranges for the other classes does not solve this problem.

5.5 Summary

We have shown a set of multi-class classification methods. The two main methods ("one against all others" and "one against one") we have implemented can be used without any knowledge about the specific data. They are efficient in the training time needed as opposed to the methods suggested in [79] while producing similar results.

The methods using Hamming error correcting codes, decision tree semantics and regression have been presented and are a topic of further research.

Chapter 6

Special Kernels

Up to now we have only used generic kernels which are applicable to any vector input space. In this chapter we will investigate kernels which are tailored to the problem in hand.

Tailoring kernels to suit a problem can be done by exploiting prior knowledge of a problem. We will look at three examples of problem specific kernels here.

Experimental results except for the digit kernel appear elsewhere in this thesis.

6.1 Digit recognition

Handwritten digit recognition was the first major application of the Support Vector machine at AT&T. In this problem the task is to recognize handwritten digits from US ZIP codes. The handwritten digits are scanned and scaled to the same size. These images then are converted into training vectors. As the images are two-dimensional and the training vectors only one-dimensional, all spatial information is lost. Worse still if coordinates in the vectors are swapped the value of the kernel function would still be the same, which means that the proximity of pixels in the picture is not accounted for at all.

The images below show one of the digits as it appears in the database (figure 6.1) and the image as a one-dimensional vector (figure 6.2):

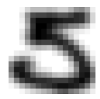

Figure 6.1: Original digit

Figure 6.2: One-dimensional representation of digit

As this approach seems to lose some of the information available, special kernels were developed by B. Schölkopf [52] at AT&T. The idea is to design a kernel that takes regions from the images and calculates the kernel as a combination of these regions.

In a normal polynomial kernel, $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$, products over all combinations of pixels are used. However in the images there are much higher correlations between pixels which are closer to each other. Kernels can be defined to take this into account by not considering all combinations of pixels.

We applied these ideas to the handwritten digits from the US postal service ZIP code database, which are rastered on a 16 by 16 grid[1]. We elected to use non-overlapping squares of size 4 by 4 for our image locality. To calculate our kernel we calculated the dot product for every non-overlapping 4 by 4 region of the image and raised this to the power d_1, then we added the terms together and raised the sum to the power d_2 this gave us a kernel with terms of degree $d_1 \times d_2$:

$$K(\mathbf{x}, \mathbf{y}) = \Big(\sum_{i,j=0}^{3} \big(\sum_{k,l=0}^{3} (\mathbf{x}_{(4j+i)+(4l+k)} \cdot \mathbf{y}_{(4j+i)+(4l+k)}) \big)^{d_1} \Big)^{d_2} \qquad (6.1)$$

where the image is stored as a vector $(\mathbf{x}_0, \ldots, \mathbf{x}_{255})$ and the first 16 elements of the vector are the first row of pixels in the image of the digit etc.

This kernel improves performance from about 4.9% to 4.6%[2] when using a localized kernel with $d_1 = 3$ and $d_2 = 2$ compared to a normal polynomial kernel of

[1] More results on this database using generic kernels can be found in section 7.2.5

[2] The results quoted here differ from the results later in this thesis, due to the use of different hardware.

degree 5.

6.2 ANOVA Decomposition

Support Vector machines using "ANOVA Decomposition" Kernels (SVAD) are a way of creating more specialised versions of some kernels which can be thought of as generic specialisations as they are not task specific.

We demonstrated the feasibility of this approach in [62].

In this section we first describe multiplicative kernels, which form the basis of SVAD and then the SVAD kernels themselves.

6.2.1 Multiplicative Kernels

Kernels, like dot products in $\mathbf{R}^N$, are functions which map from two N-dimensional vectors to a scalar. Multiplicative kernels do this by calculating the product of N identical coordinate-wise one-dimensional kernel functions κ:

$$K(\mathbf{x},\mathbf{y}) = \prod_{i=1}^{N} \kappa(x^i, y^i)$$

where x^i is the i^{th} element of the N-dimensional vector $\mathbf{x}$.

Note that the function $\kappa(x^i, y^i)$ usually is chosen never to be zero, as this would mean certain values in a single coordinate of a vector could make the whole function $K(\mathbf{x},\mathbf{y})$ equal to zero, that is they would dominate the expression.

The pattern recognition classification function takes the form:

$$f(\mathbf{x}) = \operatorname{sign}\Big(\sum_{i=1}^{\ell} y_i \alpha_i \prod_{j=1}^{N} \kappa(x_i^j, x^j) + b\Big) \qquad (6.2)$$

and the regression estimate using these kernels takes the form (2.46):

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) \prod_{j=1}^{N} \kappa(x_i^j, x^j) + b \tag{6.3}$$

where x_i^j is the j^{th} coordinate of $\mathbf{x}_i$, the i^{th} Support Vector and l is the number of Support Vectors.

There is a potential problem with kernels of this type: if N is large and the multiplicands $\kappa(x_i^j, x^j)$ vary widely in size, then for each $\mathbf{x}$ the sum of the right-hand side may be dominated by a few terms only.

Multiplicative kernels include spline kernels, which produce piecewise polynomial approximations of functions. Linear spline kernels with an infinite number of pieces are commonly used and have the form :

$$\kappa(x, y) = \frac{\min(x, y)^3}{3} + \frac{\min(x, y)^2 |x - y|}{2} + 1 + xy \tag{6.4}$$

where $x, y \in \mathbf{R}$ and $x, y \geq 0$. For more information on spline kernels see appendix A.2 or [69].

6.2.2 ANOVA Decomposition

ANOVA (ANalysis Of VAriance) is a statistical technique to analyse interactions between attributes of data. Lower order interactions are preferred as they provide a less complex explanation of the data.

The "ANOVA decomposition" introduced in [72] is motivated by an analogy with ANOVA in classical statistics. The idea is to approximate a function of a vector of N attributes as the sum of functions of smaller numbers of attributes. An order 1 approximation is to use a sum of functions of each attribute separately. An order 2 approximation is to use a sum of functions of all pairs of attributes. In general an order p approximation is to use a sum of functions of all p-tuples of attributes. That is, the p^{th} order approximation f_p is of the form $f_p(\mathbf{x}) = \sum_{1 \leq k_1 < \cdots < k_p \leq N} \psi_{k_1,\ldots,k_p}(x^{k_1}, \ldots, x^{k_p}) + b$ for some functions $\psi_{k_1,\ldots,k_p}$ that are to be found. Not all functions of N variables can be expressed as a p^{th} order approximation with $p < N$.

6.2.2.1 ANOVA Decomposition Kernels

The ideas of multiplicative kernels and ANOVA decomposition are combined to form ANOVA kernels [72]:

$$K_p(\mathbf{x}, \mathbf{y}) = \sum_{1 \leq k_1 < \ldots < k_p \leq N} \kappa(x^{k_1}, y^{k_1}) \times \cdots \times \kappa(x^{k_p}, y^{k_p}) \tag{6.5}$$

where the maximum size p of the ANOVA subsets is chosen a priori.

We will focus on the regression estimate here which from (2.46) then is

$$f_p(\mathbf{x}) = \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) \sum_{1 \leq k_1 < \cdots < k_p \leq N} \kappa(x_i^{k_1}, x^{k_1}) \times \cdots \times \kappa(x_i^{k_p}, x^{k_p}) + b \tag{6.6}$$

This form of regression estimate is the sum of a large number of multiplicative terms, where each term is of some low order p.

This estimate may be better behaved than the estimates (6.2) and (6.3), which are the sums of a smaller number of terms of higher order N. The reason is that if N is large, the terms in (6.2) and (6.3) may vary wildly in absolute size and the sum may be dominated by just a few terms. This means that more Support Vectors may be required to obtain the same level of fit.

Note that it is possible to rewrite the regression estimate in the following way:

$$f_p(\mathbf{x}) = \sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i) \sum_{1 \le k_1 < \cdots < k_p \le N} \kappa(x_i^{k_1}, x^{k_1}) \times \cdots \times \kappa(x_i^{k_p}, x^{k_p}) + b \tag{6.7}$$

$$= \sum_{1 \le k_1 < \cdots < k_p \le N} \sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i)\kappa(x_i^{k_1}, x^{k_1}) \times \cdots \times \kappa(x_i^{k_p}, x^{k_p}) + b \tag{6.8}$$

$$= \sum_{1 \le k_1 < \cdots < k_p \le N} \psi_{k_1,\ldots,k_p}(x^{k_1}, \ldots, x^{k_p}) + b \tag{6.9}$$

where $\psi_{k_1,\ldots,k_p}(x^{k_1}, \ldots, x^{k_p}) = \sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i)\kappa(x_i^{k_1}, x^{k_1}) \times \cdots \times \kappa(x_i^{k_p}, x^{k_p})$.

This formulation shows that the regressor estimated using an "ANOVA decomposition" kernel is indeed a sum of functions of p-tuples of attributes.

6.2.2.2 Algorithm

The "ANOVA decomposition" kernel (6.5) is the sum of a possibly large number of terms, but fortunately it is not necessary to compute this sum term by term.

Vapnik shows the following efficient recurrent procedure to calculate $K_p(\mathbf{x}, \mathbf{y})$ in [72]:

Let $K^s(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n}(\kappa(x^i, y^i))^s$ and $K_0(\mathbf{x}, \mathbf{y}) = 1$. Then

$$K_p(\mathbf{x}, \mathbf{y}) = \frac{1}{p}\sum_{s=1}^{p}(-1)^{s+1}K_{p-s}(\mathbf{x}, \mathbf{y})K^s(\mathbf{x}, \mathbf{y}) \tag{6.10}$$

There are two ways of using ANOVA decomposition to produce kernels of order p. The first method suggested in [72] only includes order p:

$$K(\mathbf{x}, \mathbf{y}) = K_p(\mathbf{x}, \mathbf{y}) \tag{6.11}$$

This method was used to conduct the experiments in section 7.3.

A second possibility, which has not yet been tested, includes order p and all lower orders:

$$K(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{p} K_p(\mathbf{x},\mathbf{y}) \tag{6.12}$$

The regression estimate for this has the form

$$f(\mathbf{x}) = \sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i) \times \tag{6.13}$$

$$\left(\sum_{1\le k_1<\cdots<k_m\le N} \kappa(x_j^{k_1}, x^{k_1}) \times \cdots \times \kappa(x_j^{k_m}, x^{k_m}) + \cdots + \sum_{1\le k_1<n} \kappa(x_j^{k_1}, x^{k_1}) \right) + b$$

6.2.3 Results

We tested this kernel on the Boston housing data from the UCI machine learning repository [6]. This data consists of thirteen real-valued attributes and a real-valued target. We found that the ANOVA splines perform better than normal splines on this data set. Detailed results can be found in section 7.3.

ANOVA decomposition is applicable to many other kernels such as Fourier expansions, Hermite polynomials and Radial Basis Functions and such gives an automatic way of producing further kernels.

6.3 Abdominal Pain Database

From the Digit kernels and ANOVA kernels, we developed some special kernels for the Abdominal Pain database, the results of which are presented in appendix D.4. For the discussion of kernels it shall suffice to point out the characteristics of the data. The kernels shown are applicable to all data of the type described here.

6.3.1 Applicable data

The original database contains discrete attributes which can have zero, one or more attribute values at the same time. Some of this is due to noise, but most of this comes from the fact that the database was collated from a number of sources.

Most learning algorithms cannot cope with data in this form and will only allow each attribute to have exactly one attribute value. A common solution to this, is the one applied here, that every attribute is converted into a number of binary attributes, one for each attribute value. This however loses some of the information contained in the original database. Another reason to convert the data into binary form is that the Support Vector machine assumes its inputs lie in a continuous real valued range. If attribute-values are coded as discrete numbers, some values are closer to others in the Support Vector machine's view, yet we know that the order is unimportant in reality.

6.3.2 Using a kernel to reintroduce structure

Given the data in its binary form and the original information it is possible to construct a kernel, that takes into account the original structure of the data.

6.3.3 First Approach

Assuming that the original data were suitable for use in an Support Vector machine, each original attribute would have the same weight, i.e. one term each, in the kernel function. Given the binary form however attributes with a larger number of attribute values would have more terms in the kernel function and thus more weight on the decision.

As an example imagine a problem with two attributes in the original database. Attribute a_1 can have the values *top, left, right, bottom* which will be code $1, 2, 3, 4$ and attribute a_2 can have values *white* and *black* coded as $1, 2$ and the attributes are stored as vector a. If only one attribute value can occur per attribute there is no problem in using the original form of the data. A simple dot product kernel would be

$$K_a(\mathbf{x}^1, \mathbf{x}^2) = x_1^1 \times x_1^2 + x_2^1 \times x_2^2. \tag{6.14}$$

However if multiple attribute values are allowed binary attributes b_{i_j} have to be generated for each attribute value j of each attribute i. This would lead to four binary attributes b_{1_j} from the original attribute a_1 and only two binary attributes b_{2_j} from the original attribute a_2. This would lead to the kernel

$$K_b(\mathbf{x}^1, \mathbf{x}^2) = x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4} + x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}. \tag{6.15}$$

Clearly there is now a weighting in favour of attribute b_1 which was not present in the original data.

There is a simple solution to this, namely scaling the terms according to their number of attribute values.

The problem becomes worse though when polynomial kernels are used[3].

Assuming a polynomial kernel of the form $(\mathbf{x} \cdot \mathbf{y})^2$ we get

$$K_a(\mathbf{x}^1, \mathbf{x}^2) = (x^1_1 \times x^2_1)^2 + 2(x^1_1 \times x^2_1 \times x^1_2 \times x^2_2) + (x^1_2 \times x^2_2)^2. \tag{6.16}$$

for the original case. On expansion there is one quadratic term for each of the attributes alone and two terms with both attributes. The binary case is different though:

[3]This also applies to other kernels, but can be more easily seen in the case of polynomial kernels.

$$
\begin{aligned}
K_b(\mathbf{x}^1, \mathbf{x}^2) = {} & (x^1_{1_1} \times x^2_{1_1}) \times (x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4}) \\
& + (x^1_{1_1} \times x^2_{1_1}) \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
& + (x^1_{1_2} \times x^2_{1_2}) \times (x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4}) \\
& + (x^1_{1_2} \times x^2_{1_2}) \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
& + x^1_{1_3} \times x^2_{1_3} \times (x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4}) \\
& + x^1_{1_3} \times x^2_{1_3} \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
& + x^1_{1_4} \times x^2_{1_4} \times (x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4}) \\
& + x^1_{1_4} \times x^2_{1_4} \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
& + x^1_{2_1} \times x^2_{2_1} \times (x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4}) \\
& + x^1_{2_1} \times x^2_{2_1} \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
& + x^1_{2_1} \times x^2_{2_1} \times (x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4}) \\
& + x^1_{2_1} \times x^2_{2_1} \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1})
\end{aligned}
\tag{6.17}
$$

From this it can be easily seen that on expansion there would be 16 quadratic terms for $\mathbf{x}^1$, 4 quadratic terms for $\mathbf{x}^2$ and 16 quadratic terms for a combination of both.

For this problem scaling can still be done, but it is hard in general to find a rule for every kernel and data combination of this type, especially if lower order interactions are included.

6.3.4 Second Approach

A second approach to a problem specific kernel is to analyse the transformation to binary attributes in a different way. Using the examples from the first approach it is clear that in the original data the same attribute is combined with itself far less than in the binary case. To counter this we used a kernel in which no terms combine the original attributes with themselves. For our example for a polynomial kernel of degree two this would give:

$$
\begin{aligned}
&K_b(\mathbf{x}^1, \mathbf{x}^2) = \\
&\qquad (x^1_{1_1} \times x^2_{1_1}) \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
&\qquad + (x^1_{1_2} \times x^2_{1_2}) \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
&\qquad + x^1_{1_3} \times x^2_{1_3} \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
&\qquad + x^1_{1_4} \times x^2_{1_4} \times (x^1_{2_1} \times x^2_{2_1} + x^1_{2_1} \times x^2_{2_1}) \\
&\qquad + x^1_{2_1} \times x^2_{2_1} \times (x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4}) \\
&\qquad + x^1_{2_1} \times x^2_{2_1} \times (x^1_{1_1} \times x^2_{1_1} + x^1_{1_2} \times x^2_{1_2} + x^1_{1_3} \times x^2_{1_3} + x^1_{1_4} \times x^2_{1_4})
\end{aligned} \tag{6.18}
$$

This kernel and a similar kernel of degree three have been applied to the medical data, but unfortunately these kernels have not produced results much different from normal kernels. The detailed results for these experiments can be found in appendix D.4.

A deeper investigation into this kind of problem specific kernel should be undertaken as future research because this kind of setting occurs often.

6.4 Summary

In this chapter we have shown some special kernels. The kernel designed for digit recognition draws from additional knowledge about the data. The ANOVA kernel avoids certain terms dominating the Support Vector machine's decision function and can be used as a meta-kernel taking advantage of other kernels' properties while avoiding some of their pitfalls. The kernels proposed for the abdominal pain database show a new direction of research which should be pursued in future work.

Chapter 7

Experiments using the Support Vector machine

In this chapter we give a review of some experiments on benchmark and "real-world" data sets using the Support Vector machine.

The main aim is to investigate the performance of different kernels. While doing this we will also look at other issues on selected data sets. We will look at ways of adjusting some of the other parameters, including certain types of prior knowledge, speed of the Support Vector machine, requirements of memory for different chunking methods, the different multi-class methods and some limitations of the Support Vector machine.

We chose a subset of the UCI/Statlog data [6] for our experiments as there are many published results. In addition to this we have used the USPS handwritten digit database and an acute abdominal pain database.

This should give the reader an insight into the general performance of the Support Vector machine.

There are three distinctly different tasks of the Support Vector machine we will present here:

- Binary Pattern Recognition
- Multi-class Pattern Recognition

- Regression Estimation

Each of these tasks is exemplified by a set of standard benchmarks. Each data set is run on different Support Vector machines and various properties of the Support Vector machines are pointed out. We created parameter files for all experiments using the menu driven interface provided by our Support Vector machine library. This allowed us to run the experiments in batch mode which is very useful when many training and test sets have to be run with the same parameters.

A conclusion is given for each of the three tasks performed by the Support Vector machine after the experiments for that task.

Other sources for results on these data sets include [66] and [67] which summarize a large set of results obtained by many contributors. However, we have not been able to reproduce their results. One learning algorithm that was used in [66] was C4.5. We attempted to reproduce the results that were reportedly obtained by using the Statlog data generation procedure described below and C4.5. Even though we had identical software we could not reproduce their results. The difference between our results and theirs was quite large; for some data sets their results were better, for some data sets our results were better. We cannot explain this differnce, apart from assuming that their method of testing was different from ours and did not follow the Statlog procedures exactly. Given the differing results we did not feel justified in reproducing their results here, as they would not give a meaningful comparison of the different learning algorithms.

7.1 Binary Pattern Recognition

7.1.1 Heart Disease

This database contains 270 patient records, of which 120 have a particular heart disease. Each patient is characterised by 13 symptoms. The database has been extensively used in the Statlog project. We have used the Statlog project's data generation program to generate data sets for 9-fold cross validation. The Statlog data generation program generates nine pairs of training and test sets for 9-fold cross validation. These sets are generated by splitting the data into 9 equal sized generation sets and for the first pair of training and test set using the first of the nine generation sets as the test set and the other eight generation sets as the training set. The other eight pairs are created in the same way. This method of generation depends on the order of the data and certain orderings can occur in which one class only appears in the test set and not in the training set.

The data sets were then tested against different Support Vector machines. We did not bound the Lagrangian variables and used old-style chunking. For spline kernels we scaled the data individually between 0 and 1, for polynomial kernels we scaled the data individually between -1 and 1. This ensured that the kernel and alpha values would stay within a reasonable range.

Cross Validation	Test set size	SV Machine Spline	Poly 4	5	6	7
Total errors	270	67 24.8%	66 24.4%	66 24.4%	66 24.4%	64 23.7%

Table 7.1: Errors on Test set (9×30)

All kernels used here produce similar results (0.9% spread) and take a similar time to run. Results were obtained in a couple of seconds for each kernel. In this case it is possible not to use chunking and using chunking does not improve timings or use of memory. This is because the data set is so small that the overhead of chunking and the size of the program wipe out any advantages in having a smaller optimization problem that is quicker to solve and uses less memory.

7.1.2 Diabetes

This database contains 768 patient records, of whom 500 have diabetes. Each patient is characterised by 8 symptoms. The database has been extensively used in the Statlog project. We have used the Statlog project's data generation program to generate data sets for 12-fold cross validation.

The data sets were then tested against different Support Vector machines. We did not bound the Lagrangian variables and used old-style chunking. For spline kernels we scaled the data individually between 0 and 1, for polynomial kernels we scaled the data individually between -1 and 1.

Cross Validation	Test set size	SV Machine				
		Spline	Poly 4	5	6	7
Total errors	768	241 31.4%	256 33.3%	247 32.2%	240 31.3%	240 31.3%

Table 7.2: Diabetes - Errors on Test set

Again the results for the different kernels are similar (1.9% spread). To get better results we decided to fine tune the performance on the polynomial kernel of degree 6 which was the best so far. We looked at the maximum alpha values for each of the cross-validation runs and decided on C values below these for each cross-validation set. It took four different values of C to come up with an improved error rate of 29.6%.

Comparing the timings, using old-style chunking took about three minutes on a 500MHz Alpha, not using chunking took about three minutes again on the same machine. Using new-style chunking took slightly less time, two and a half minutes. In memory usage the new style chunking used 2.1MB, the old-style chunking used about 2.7MB memory whereas the non-chunking method used about 4.2MB. So although chunking does not necessarily speed the Support Vector machine up, on a heavily loaded machine the use of less memory can give a speed advantage.

7.1.3 Summary

In both experiments the Support Vector machine produces similar results for the different kernels.

Binary Support Vector machines for small data sets have fast training times and the use of chunking is not usually of any advantage, apart from the smaller memory usage.

Leave one out cross-validation could be used to find more appropriate parameters, yet this would entail a large cost of trying different parameters. Even without cross-validation the Support Vector machine's results can be improved by heuristically adjusting the value of C. Further research still has to be done on finding optimal values of the C-bound without heuristics.

7.2 Multi-class Pattern Recognition

For the multi-class experiments we used the same data splits as Weston and Watkins used in [79], rather than the splits suggested in the Statlog project, as we had access to the pre-processed data and this allowed us some comparison as mentioned in chapter 5.

7.2.1 Iris

The Iris plants database has been used since the 1930s as a simple benchmark for multi-class pattern recognition. It was created by R. A. Fisher and is probably the best known database to be found in the pattern recognition literature.

The dataset consists of 150 examples in 3 classes (Iris Setosa, Iris Versicolour, Iris Virginica) of 50 examples. Each example has four attributes: sepal length, sepal width, petal length, and petal width all in cm.

We used this data set with both the "one against all others" and the "one against one" multi-class classification methods with 10-fold cross-validation.

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
Total error	150	5 3.3%	8 5.3%	8 5.3%	8 5.3%	7 4.6%

Table 7.3: Iris - Errors on Test set for "1 against 1"

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
Total error	150	5 3.3%	10 6.6%	8 5.3%	9 6.0%	8 5.3%

Table 7.4: Iris - Errors on Test set for "1 against all others"

As in the binary pattern recognition experiments the difference between the best and worst kernel is small (3.3%). However the spline kernel performs better than all the polynomial kernels. We also observe that the "one against one" method has a better worst case performance than the "one against all others" method.

To improve the results we took into account that it is known that in this data set it is possible to separate one class from the other two linearly. It is possible to use this information in the multi-class Support Vector machine. To reduce the number of parameters we tried this on the "one against one" approach. There are three binary Support Vector machines in this case separating two classes each. Two of these separations are linear, so we can use a simple dot product as the kernel and can leave C at infinity. For the third machine we choose a spline kernel as this gave the best results in the previous experiments on this data. Initially we left C at infinity for this kernel. With this experimental setting we achieved an overall error rate of 3.3% (5 errors), which is no better than using solely spline kernels. However analysing the alpha values of the Support Vector machines we noted that they lie between about 70 and 67,000 for each of the spline kernel Support Vector machines. This range is far too large and the final function is dominated by a few terms. The solution is to restrict the alpha values by a lower C. We decided to set $C = 4000$ and thus reduced the error rate down to 2.0% (3 errors). These errors were occurring only in two of the cross-validation training sets so we tackled these by lowering C even more. We lowered C down to 1500, which left us with errors in just one of the cross-validation training sets, so for this one we lowered C down to 700. These final settings achieved a 0.0% error rate.

This simple test shows that the multi-class Support Vector machine is feasible, used without parameter tuning. However tuning the parameters to optimality can make it more competitive.

For this training set it is interesting to note that the "one-against-one" multi-class approach needs about half a second for training, whereas the "one-against-all-others" approach take just over a second[1]. This is due to the individual binary Support Vector machines having to deal with smaller training sets.

[1] All run on a 200MHz Pentium

7.2.2 Wine

The wine database consists of three types of Italian wine for which 13 constituent parts, found through chemical analysis are recorded. There are 178 different wines from three different cultivars. All attributes are continuous. We used 10-fold cross-validation for this experiment.

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
Total error	250	4 1.6%	9 3.6%	7 2.8%	9 3.6%	11 4.4%

Table 7.5: Errors on Test set for "1 against all others"(10 $\times$ 25)

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
Total error	250	5 2.0%	6 2.4%	6 2.4%	8 3.2%	11 4.4%

Table 7.6: Errors on Test set for "1 against 1"(10 $\times$ 25)

In this experiment we can see that the spline kernel again performs better than the polynomial kernels, even if the improvement is not as marked as with the previous experiment.

In this experiment we observe roughly the same training times as in the previous on and again the "one-against-one" approach is faster.

We have not attempted to find an optimal value for C because we have no additional knowledge about the data which could help, nor is it easy to look at the alpha values and find a suitable one easily. In the case of binary pattern recognition this is possible because there is only one value of C to consider; in the case of multi-class pattern recognition there is one value of C for every binary Support Vector machine. Improving the performance of a single binary machine will have an effect on the other binary machines.

7.2.3 Glass

This database is a forensic database from the Home Office Forensic Science Service in the UK. It contains 214 examples of glass found at crime sites. The examples have 9 real-valued attributes relating to the chemical and physical properties of the glass. The aim is to find the type of glass, which can be one of seven types.

There are in fact 10 attributes in the original database. Unfortunately one of them is an index number which directly identifies the type of glass. Many researchers seem to include this attribute, which makes their results doubtful. Using all ten attributes gave us a very simple decision function, which relied solely on the index number and none of the other attributes, so we removed the index number for the experiments quoted here.

We have used ten-fold cross validation as used in [79], rather than the Statlog recommended 7-fold cross validation. This is also different from the Statlog approach as it randomises the data, whereas the Statlog approach leaves the data in order, which can lead more easily to the problem of the training set not containing any examples of one class as the data is ordered according to class. Using the ordered data often resulted in the training set not containing any of the classes contained in the test set and correspondingly high error rates.

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6	7
Total error	250	74 29.6%	91 36.4%	93 37.2%	90 36.0%	88 35.2%	86 34.4%

Table 7.7: Errors on Test set for "1 against all others"(10×25)

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6	7
Total error	250	77 30.8%	75 30.0%	80 32.0%	82 32.8%	82 32.8%	86 34.4%

Table 7.8: Errors on Test set for "1 against 1"(10×25)

The spline kernel in this experiment again has a better performance than the polynomial kernels (7.6% spread).

7.2.4 Soy

The Soybean database contains 289 examples of soybeans. The soybeans are classified into 17 classes according to 35 attributes which are coded as 208 binary attributes.

Examples with missing attributes have been purged, which explains the slight deviations in the data from the original database.

In the following table not all kernels produced values the optimizer could converge on. When this happened we received no solution from the Support Vector machine. The experiment was run using 10-fold cross-validation.

Cross Validation	Test set size	SV Machine 1-a-1 Poly 3	1-a-1 4	1-a-a Poly 3
Total error	370	9 2.4%	9 2.4%	7 1.9%

Table 7.9: Soy - Errors on Test set

The failure of the Support Vector machine to produce solutions with a number of kernels can be explained by the fact that the ratio of attributes to examples is not favourable. Each class has about 15 examples in the training sets compared to 35 attributes. This ratio makes finding a solution very hard for the optimizer with this data. The kernels a solution was found for correspondingly had the least degree of freedom in the set of possible decision functions.

7.2.5 USPS Handwritten Digits

These experiments aim to create classifiers that can recognise handwritten digits as described in section 6.1.

Experiments were done on the "one against all others", "one against one" and "Hamming error correcting code" methods. The database contains 7291 training points and 2007 test points. Each method was run using a selection of kernels. In all cases a hard-margin classifier was used.

The best results on this database were obtained by P. Simard, Y. LeCun and J. Denker. The analytical tangent distance method obtains an error rate of 2.7% without using any learning [57]. However the tangent distance measure fails on larger digit data sets.

The following table contains the error rates for the test set for the "one against all others" method:

Degree of Polynomial						
1	2	3	4	5	6	7
36.1%	6.4%	6.0%	6.2%	6.6%	6.5%	7.3%

This method is the same as quoted in [71] pages 145-147. In Vapnik's case the parameter C of the Support Vector machines has been adjusted to optimality though.

The following table contains the error rates for the test set for the "one against one" method:

Degree of Polynomial						
1	2	3	4	5	6	7
7.4%	5.2%	4.8%	4.5%	4.4%	4.7%	4.9%

We also applied the "Hamming error correcting code" method. This problem does not have an ideal number of classes for the code used and as such we expect worse results than with the other two methods.

The following table contains the error rates for the test set using the "Hamming error correcting codes" method:

Degree of Polynomial						
1	2	3	4	5	6	7
16.5%	7.4%	6.1%	5.9%	5.9%	5.8%	5.4%

In this experiment no particular polynomial kernel was significantly better than any other kernel. The exception here are the linear simple dot product kernels (polynomials of degree 1). These could not compete with the polynomial kernels. The error rates of the polynomial kernels varied by 2% at most.

This is the only data-set we tested using all three multi-class methods we implemented. We used the "Hamming error correcting code" method here only because error correcting codes with one digit correction capability are far from ideal and the error correcting code implemented just shows that this approach is possible and produces similar results to the other two methods.

This data set is also sufficiently large to measure the run time well. With the polynomial kernel of degree 3 and new-style chunking (400/200) the "one against one" method was the fastest taking 62 minutes[2], the "one against all others" method took 170 minutes and the "Hamming error correcting code" method took 336 minutes. Here the "one against one" method benefitted from the far smaller training sets which contained at most 2200 examples, whereas the other two methods training set always contained all 7291 examples. Although the "one against one" method contained more binary Support Vector machines than the "Hamming" method (10 vs. 7), the "Hamming" method contained harder binary problems to solve. This is not surprising and as pointed out before the ordering of the classes can greatly affect the result. When using old-style chunking with hindsight in choosing the chunk-sizes according to the number of Support Vectors the "one against one" method took 16 minutes, the "one against all others" method took 42 minutes. We did not run the "Hamming" method with the old-style chunking as the number of Support Vectors was too high. These improvements were mainly due to the knowledge of the number of Support Vectors as this knowledge made it possible to estimate how big the chunks had to be.

In terms of memory usage the chunking method did not have any effect, however the multi-class method had a significant effect. In the case of the "one against

[2]Measurements on a 200MHz Pentium

all others" method 41MB of memory were used, the "one against one" method required 156MB of memory and the "Hamming" method used 40MB of memory. For the "one against all others" and "Hamming" methods the training examples can be stored more efficiently than for the "one against one" method because the examples are the same for the first two methods except for the targets. The "one against one" method requires different training sets for each binary machine and there are 45 binary machines. Although the program run time in the case of old-style chunking is three times as long in the case of "one against all others" as opposed to "one against one", the real time required on our machine with 128MB of physical ram was 33 minutes for "one against one" and 45 minutes for "one against all others". In the real world with smaller computers the "one against one" can loose some of its advantage in speed.

7.2.6 The Abdominal Pain Database

This section shows the performance of the Support Vector machine on a "real-world" application in detail. We have gone into far more detail for this data set as it is a "real-world" data set which contains all the problems like missing values, faulty input and an unsuitable format, whereas the other data sets are suitably pre-processed. The data set consists of records of patients with abdominal pain. The records had been collected by Dr. Gunn of the Bangour Hospital in Roxburgh. Dr. Nixon of the Western General Hospital, Edinburgh then passed the data on to Prof. Gammerman of our research group.

De Dombal's paper about computer-aided diagnosis of acute abdominal pain was the first to investigate this field ([28] and [11]). De Dombal used a form of Simple Bayes and this system was very successful in its test setting. It used similar diseases and symptoms as our data set. There have been many studies in this area since, most notably [1] which presents a study involving over 16,000 patients. The domain of acute abdominal pain is well suited to computer-aided diagnosis as there are a small number of possible diagnoses and the symptoms used for initial diagnosis of the diseases are available immediately after the patient is admitted.

The data set used by us has been previously used in [19] and [23]. These previous studies concentrate on the Simple Bayes and "Proper Bayes" or G&T approach.

7.2.6.1 Data

The database contains patient records on *Appendicitis, Diverticulitis, Perforated Peptic Ulcers, Non-specific Abdominal Pain, Cholecystitis, Intestinal Obstructions, Renal Colics* and *Dyspepsia*[3]. These are the most common causes of abdominal pain when it is diagnosed. Initial diagnoses are usually made on the basis of external symptoms. The final diagnosis is then made during operation. Any pain that had no apparent cause was classified as *Non-specific Abdominal Pain*.

The database contains the symptoms of patients suffering from abdominal pain and their diagnosis at the end of the treatment period. Every patient was diagnosed[4] into one of eight diagnostic groups or remained undiagnosed. There were 33 symptoms, each of which had a range of attribute values. Some symptoms are not recorded in the database for every patient and for some symptoms many attribute values can be present simultaneously for one particular patient[5]. This setting requires us to pre-process the data for every learning machine we will use.[6]

There are two ways of treating missing attribute values. Either patient records or examples with missing attribute values are removed from the training set, or attributes with missing attribute values are removed from each patient record. Which strategy is chosen for which missing set of attribute values is a choice that benefits greatly from expert advice, but can also be taken depending on the data. If an attribute has missing attribute values for a large number of examples, it is probably best to remove the attribute, whereas if an attribute has missing attribute values for a very small number of examples it is preferable to remove those few examples.

[3] These diseases are numbered 1 to 9 in the following.

[4] This diagnosis usually took place in the operating theatre.

[5] Some of this is due to the data not being recorded in the original paper records, more often though this is due to data input errors. Examples of this include patients who are male and female simultaneously.

[6] For a more detailed description of the data see appendix E.

7.2.6.2 Actual data sets

The original data set was split into a training and a test set by the donors of the data a priori. The training set contains 4387 examples and the test set 2000 examples.

We decided to preprocess the data by converting each of the original 33 attributes into 135 binary attributes. This removed the problem of multiple attribute values per attribute. It does of course introduce additional degrees of freedom within the domain of the function to be learnt.

Once the data was converted into binary valued attributes, we started addressing the problem of missing values. Initially we worked with the whole data set. The results presented in this section were all produced using this data. Then we created two differently preprocessed versions of the data sets, progressively stripping out more examples.

The first stripped version ("strip 1") had all the non-specific abdominal pain examples removed which left 2446 examples in the training set and 1106 examples in the test set. Stripping out examples which fall into "bin-classes" from data sets is commonly done[7].

For a second even more stripped version ("strip 2") we considered additionally removing those examples where a patient was classified as being of no or two genders, or of no, two or more ages. As these values are impossible and only occur in a small number of examples we felt justified in removing them. This however did not present a significant change and we have not included detailed results as they differ only marginally from the previous stripped down dataset as can be seen in table D.13.

The results for the stripped down datasets can be found in appendix D.

7.2.6.3 Test Setting

For each of the three data set versions we ran the Support Vector machine with several parameter sets and kernels.

[7] See [6] and [23]

We decided to run a large selection of Support Vector machines representing commonly used kernels and methods on the datasets. We used polynomial kernels of degrees 3, 4, 5 and 6 as well as spline kernels. Polynomial kernels have the advantage over other kernels like the Radial Basis Function kernel, that there is only one discrete parameter; spline kernels have no parameters at all. In our experience useful solutions are only found with degrees two to seven of the polynomial kernel. We have in our experiments only used degrees three to six. This has limited the number of experiments to a manageable number. The choice of chunking has very little effect on the final solution, which is why we have opted for new style chunking with a maximum chunk size of 500 and a swap size of 200 as this is reasonably fast. In addition to this we ran two of the problem specific kernels on one of the small "strip 1" datasets only, as the problem specific kernels are computationally very expensive. We used both the "1 against 1" (1-a-1) and the "1 against all others" (1-a-a) methods. Scaling was only used when this was necessary for the kernel or the optimizer. This is described in the appropriate experiment.

7.2.6.4 Test results

The following pages contain the an overview of the results from all test runs on the original datasets. The detailed results for all abdominal datasets can be found in appendix D. The reader might want to skip to section 7.2.6.5 and only refer to the more detailed results after having formed a picture from the overview at the end of this chapter.

1-a-a - Poly 3

The first experiment was done using the "1 against all others" multi-class technique using a Vapnik polynomial kernel of degree 3.

Multi-class SVM 1 against all others			
Kernel:	Vapnik Polynomial[8]	Parameters:	Degree 3
Alpha bound C:	∞		
			continued on next page

[8]The kernel names correspond to the names in appendix A

continued from previous page			
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table 7.13: 2000 points - 1-a-a - Poly 3 - Parameters

The total number of errors was 560 out of 2000. Of these errors 213 were due to computer decisions for class 4. As many of the cases of non-specific abdominal pain (class 4) show similar symptoms to the other diseases, the decision function reflects this and classes many cases of other diseases in the test set as class 4. This behaviour can be observed among all the methods we used. In line with this 147 errors were due to classing non-specific abdominal pain as some other disease. Removing class 4 from the sets does indeed reduce the number of errors by nearly that number. Class 4 seems to be the hardest classification sub-task. However the incentive to classify examples of class 4 correctly is very high as it makes up nearly half of the test set and a bad performance on class 4 will greatly reduce the overall accuracy. This is reflected by the fact that the accuracy on class 4 is the highest among the classes.

1-a-a - Poly 4

For polynomial kernels of degree 4, 5 and 6 it is not possible to use the Vapnik Polynomial due to the numerical accuracy of the digital computers used with this multi-class method. From our optimization objective (2.29) it is clear that large kernel values mean small alpha values. If the kernel values become very large and the alpha values very small, the product of the two becomes increasingly less accurate if they have a large number of significant digits. Our Support Vector machine prints appropriate error messages when this happens. In such cases we have to change the kernel.

We use the so-called “Full Polynomial” kernel from appendix A.1, which allows us to scale down the kernel values and make the optimization process feasible again. As our 135 attributes lie between 0 and 1 the minimum normal dot product is 0 and the maximum 135. As 135^4 is too large a number, we can scale the normal dot product contained in the kernel down before raising to the power:

$$(\frac{(\mathbf{x} \cdot \mathbf{y})}{135} + 1)^4$$

The same is done in the case of polynomial kernels of degree 5 and 6.

Multi-class SVM 1 against all others			
Kernel:	Full Polynomial	Parameters:	Deg. 4, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table 7.14: 2000 points - 1-a-a - Poly 4 - Parameters

The total number of errors was 564. Of these again a total of 211 were decisions where examples were misclassified as class 4 and 161 were decisions where class 4 was misclassified as some other class. When class 4 is stripped out 314 errors are removed. It is interesting to note here that the accuracies per class are distributed similarly as with the previous kernel.

1-a-a - Poly 5

Multi-class SVM 1 against all others			
Kernel:	Full Polynomial	Parameters:	Deg. 5, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table 7.15: 2000 points - 1-a-a - Poly 5 - Parameters

The total number of errors for this kernel was 549. Of these 230 were misclassified examples classified as class 4 and 144 were misclassified examples of class 4. We again have class 4 as the most accurately classified class and a similar distribution of per class accuracies as with the previous kernels.

1-a-a - Poly 6

Multi-class SVM 1 against all others			
Kernel:	Full Polynomial	Parameters:	Deg. 6, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table 7.16: 2000 points - 1-a-a - Poly 6 - Parameters

The total number of errors was 541. Here we have 226 wrong decisions for class 4 and 141 decisions which should have been class 4. We observe a similar distribution of errors per class and it seems that the polynomial kernels all give similar results in general and it is simply a task of choosing the best from a discrete set of kernels.

1-a-a Spline

Multi-class SVM 1 against all others			
Kernel:	Splines	Parameters:	-
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	global	Range:	0.0 to 0.5

Table 7.17: 2000 points - 1-a-a - Spline - Parameters

The total number of errors was 529. Of these 94 were due to class 4 examples being misclassified and 268 to other examples being classed as class 4. It is interesting to note that although class 4 was again the most accurately diagnosed class, the percentage of correct diagnoses is far higher than with the polynomial kernels. The classification accuracy for the other classes is lower though.

1-a-1 - Poly 3

The following Support Vector machines were run using the “one against one” multi-class method. This method uses a larger number of binary Support Vector machines each of which is easier to train than the binary Support Vector machines in the “one against all others” approach used in the previous experiments.

Multi-class SVM 1 against 1			
Kernel:	Vapnik Polynomial	Parameters:	Degree 3
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table 7.18: 2000 points - 1-a-1 - Poly 3 - Parameters

The total number of errors here was 554. We observe 157 errors of misclassified class 4 and 223 errors of misclassification into class 4.

1-a-1 - Poly 4

Multi-class SVM 1 against 1			
Kernel:	Vapnik Polynomial	Parameters:	Degree 4
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table 7.19: 2000 points - 1-a-1 - Poly 4 - Parameters

The total number of errors was 544. There were 141 errors which should have been classified as class 4 and 246 which should not have been classified as class 4.

1-a-1 - Poly 5

Multi-class SVM 1 against 1			
Kernel:	Full Polynomial	Parameters:	Deg. 5, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table 7.20: 2000 points - 1-a-1 - Poly 5 - Parameters

The total number of errors was 559. Out of these 156 were examples of class 4 which were misclassified and 226 of other classes which were classified as class 4.

1-a-1 Poly 6

Multi-class SVM 1 against 1			
Kernel:	Full Polynomial	Parameters:	Deg. 6, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table 7.21: 2000 points - 1-a-1 - Poly 6 - Parameters

The total number of errors was 541. The number of errors due to classification not as class 4 is 146 and due to classification as class 4 212.

In all of the “1-against-1” experiments with the polynomial kernels we observe similar distributions of error among the classes, with class 4 being the best diagnosed.

1-a-1 - Spline

Multi-class SVM 1 against 1			
Kernel:	Splines	Parameters:	-
Alpha bound C:	∞		
			continued on next page

continued from previous page			
Chunking:	new style	Sizes:	500/200
Scaling:	global	Range:	0.0 to 0.5

Table 7.22: 2000 points - 1-a-1 - Spline - Parameters

The total number of errors was 553. Of these 109 were due to wrong decisions for examples of class 4 and 278 decision as class 4. We observe the same behaviour as with the previous spline kernel in that although class 4 is far more accurately classified than with the polynomial kernels, the other classes' accuracy drops in comparison.

7.2.6.5 Analysis of Abdominal Pain

Table 7.23 shows an overview of the different kernels' performance. The Support Vector machines all produce similar results for all kernels (1.7% spread). Both multi-class methods produced results in the same ranges with the "one against all others" methods slightly ahead.

We have observed that large classes like class 4 dominate the classifiers. It is most important for the crude form of measurement of total accuracy to classify the majority of examples correctly, rather than to achieve a good performance over all classes.

To optimize the performance each binary classifier can be set up with a different kernel, which might be more appropriate, as well as adjusting C to improve the generalization ability.

Method	1-a-a	1-a-1
SV Poly 3	28.0%	27.7%
SV Poly 4	28.2%	27.2%
SV Poly 5	27.5%	28.0%
SV Poly 6	27.1%	27.1%
SV Spline	26.5%	27.7%

Table 7.23: Overview of error rates on Abdominal Pain

Analysis of class performance We will now look in detail at the performance of the Support Vector machine on a per class basis for the abdominal pain problem in detail.

The graph 7.1 shows the per-class performance of some of the Support Vector machines against each other.

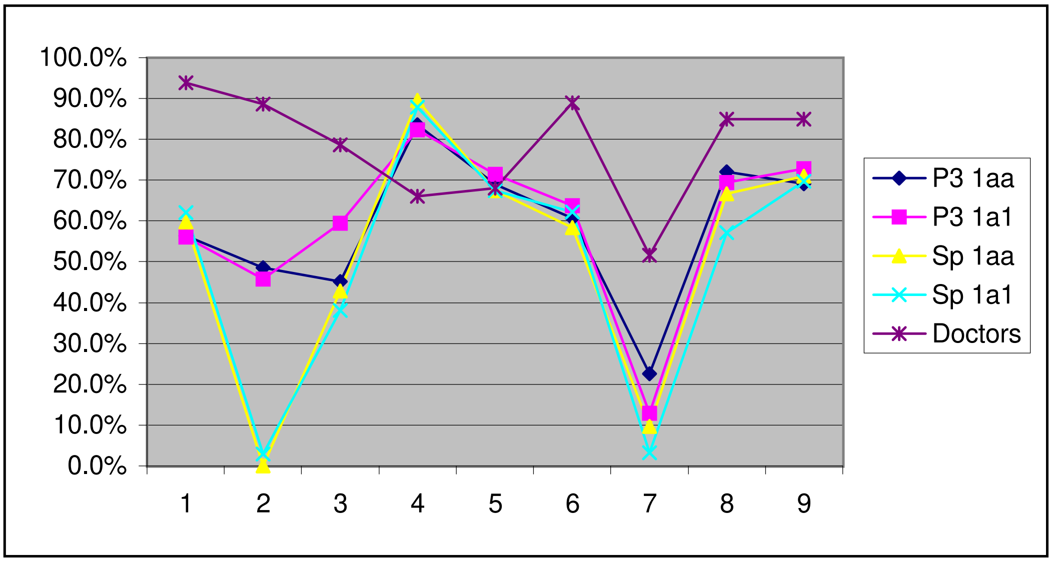

Figure 7.1: Per class performance

Class 4 is the largest group in the data. It makes up 44.2% of the training set and 44.7% of the test set. This class is not a well defined diagnostic group like the other eight classes, but a sink for undiagnosed diseases. This is reflected by the fact that the doctors performed badly on this group. All Support Vector machines in contrast performed extremely well on this group and better than the doctors in all cases. The table also demonstrates that due to the number of examples in class 4, misclassifying examples as class 4 does not have to degrade the overall performance much. Class 4 is the most important class to achieve a high performance on, for the purpose of measuring performance, yet this might not be an ideal measure. Errors involving class 4 also make up a large proportion of the errors overall, larger in fact than its proportion of the test set.

Algorithm	Total errors	Class 4 wrong	Other classes as class 4
SVM P3 1aa	560	147	213
SVM P4 1aa	564	161	211

continued on next page

continued from previous page			
Algorithm	Total errors	Class 4 wrong	Other classes as class 4
SVM P5 1aa	549	144	230
SVM P6 1aa	541	141	226
SVM Sp 1aa	529	94	268
SVM P3 1a1	554	157	223
SVM P4 1a1	544	141	246
SVM P5 1a1	559	156	226
SVM P6 1a1	541	146	212
SVM Sp 1a1	553	109	278

Table 7.24: Overview of Errors involving class 4

Class 7 is badly diagnosed by the doctors as well as by the algorithms. This is mainly due to the fact that vital symptoms were mis-recorded in the data sets. In addition to this there are very few examples (1.5% in the training set and 1.6% in the test set). This makes the likelihood of overfitting for this class greater.

Class 2 is badly diagnosed by the two Spline Support Vector machines. It represents 2.5% of the training set and 1.8% of the test set. We assume that this is due to overfitting.

An overview of the results on individual diseases can be seen in tables 7.25 and 7.26. We have produced bar graphs of these tables in appendix D.1.

These initial observations led us to try to use the best overall Support Vector machine for "one against all others" and change the kernels that classified class 2 and 7 against the other classes, to the best Support Vector machine's kernel for that class. The best Support Vector machine for class 2 was the one using the polynomial of degree 4 and the best for class 7 was the one using the polynomial of degree 5. Using these settings we indeed improve the performances on those two classes as can be seen in table D.11. However although the performance on two classes improved, the overall performance decreased as all binary classifiers are dependent. This is mainly due to the performance on class 4 decreasing. Although class 8 and 9 had a few more errors, the fact that class 4 dominates the test set, means that the overall performance is hugely affected by it. Looking at the percentages class 4 lost 2.0% errors in contrast to all other classes together gaining 1.1%. The overall performance dropped by 0.2%.

As a second experiment we tried using the best classifier's kernel for every class (see table D.12). We observed the same behaviour as before, in that the performance on class 4 drops by 6.4%, but the performance on all other classes increases by 2.5% and the overall performance drops by 1.1% against the best single kernel classifier.

These two experiments show that, any improvements which are targeted solely at single binary classifiers within the multi-class Support Vector machine do not necessarily improve the overall performance. However we can see that direct targeting can improve the performance on the classifiers targeted specifically.

	P3 1aa	P4 1aa	P5 1aa	P6 1aa	Sp 1aa	Doctors	Mix 1aa	Mix2 1aa
1	56.4%	53.7%	53.7%	55.6%	59.8%	93.8%	60.6%	58.3%
2	48.6%	51.4%	48.6%	42.9%	0.0%	88.6%	57.1%	54.3%
3	45.2%	42.9%	50.0%	57.1%	42.9%	78.6%	45.2%	47.6%
4	83.6%	82.0%	83.9%	84.2%	89.5%	66.1%	87.5%	83.2%
5	69.0%	71.0%	72.0%	71.5%	67.5%	68.0%	68.0%	67.0%
6	60.6%	65.4%	62.2%	65.4%	58.3%	89.0%	58.3%	61.4%
7	22.6%	22.6%	25.8%	12.9%	9.7%	51.6%	19.4%	16.1%
8	72.1%	70.1%	70.7%	68.0%	66.7%	85.0%	63.3%	69.4%
9	69.1%	72.8%	71.3%	72.8%	70.9%	84.9%	67.5%	71.7%
Avg.	58.6%	59.1%	59.8%	58.9%	51.7%	78.4%	58.5%	58.8%

Table 7.25: Percentages of success per class ("one against all others")

	P3 1a1	P4 1a1	P5 1a1	P6 1a1	Sp 1a1	Doctors
1	56.0%	54.4%	55.2%	55.6%	62.1%	93.8%
2	45.7%	31.4%	40.0%	45.7%	2.9%	88.6%
3	59.5%	54.8%	59.5%	57.1%	38.1%	78.6%
4	82.4%	84.2%	82.6%	83.7%	87.8%	66.1%
5	71.5%	73.0%	70.0%	71.5%	67.5%	68.0%
6	63.8%	67.7%	64.6%	64.6%	62.2%	89.0%
7	12.9%	9.7%	12.9%	19.4%	3.2%	51.6%
8	69.4%	70.1%	68.0%	71.4%	57.1%	85.0%
9	72.8%	71.7%	73.6%	72.1%	69.8%	84.9%
Avg.	59.3%	57.4%	58.5%	60.1%	50.1%	78.4%

Table 7.26: Percentages of success per class ("one against one")

7.2.6.6 Some comparisons

For the abdominal data set we compared our result to those obtained by A. Gammerman. From the tables below we can see that the Support Vector machine can compete with other algorithms even without fine tuning.

Disease	Final diagnosis	Doctors' diagnosis	Computer diagnoses Simple Bayes	SVM 1aa Spline	G&T	CART	ID3	NN	CPRS
1 APP	259	243	192	155	121	91	97	165	145
2 DIV	35	31	22	0	9	0	2	26	9
3 PPU	42	33	32	18	21	21	15	28	25
4 NAP	894	591	669	800	753	744	766	751	714
5 CHO	200	136	152	135	121	133	122	165	139
6 INO	127	113	92	74	55	64	58	64	84
7 PAN	31	16	7	3	1	0	0	1	2
8 RCO	147	125	117	98	73	74	79	102	84
9 DYS	265	225	192	188	148	163	166	33	165
Total	2000	1513	1475	1471	1302	1293	1305	1335	1367
Accuracy (in %)	100	75.7	73.8	73.6	65.1	64.7	65.3	66.8	68.4

Table 7.27: Overview of success rates on Abdominal Pain

Disease	Final diagnosis	Doctors' diagnosis	Computer diagnoses Simple Bayes	SVM 1aa Spline	G&T	CART	ID3	NN	CPRS
1 APP	100	93.8	74.1	59.8	46.7	35.1	37.5	63.7	56.0
2 DIV	100	88.6	62.9	0.0	25.7	0.0	5.7	74.3	25.7
3 PPU	100	78.6	76.2	40.9	47.7	47.7	34.1	63.6	56.8
4 NAP	100	66.1	74.8	89.5	84.2	83.2	85.7	84.0	79.9
5 CHO	100	68.0	76.0	67.5	60.5	66.5	61.0	82.5	69.5
6 INO	100	89.0	72.4	58.3	43.3	50.4	45.7	50.4	66.1
7 PAN	100	51.6	22.6	9.7	3.2	0.0	0.0	3.2	6.5
8 RCO	100	85.0	79.6	66.7	49.7	50.3	53.7	69.4	57.1
9 DYS	100	84.9	72.5	70.9	55.8	61.5	62.6	12.5	62.3
Total	100	75.7	73.8	73.6	65.1	64.7	65.3	66.8	68.4

Table 7.28: Overview of success percentages on Abdominal Pain

In these tables we have included the best Support Vector machine results. The best Support Vector machine performs on par the Simple Bayes method, whereas the other approaches are significantly worse. This strengthens the necessity of further research in the area of tuning parameters and find more exact methods than the ad hoc heuristics used by us for tuning parameters which do seem to point in the right direction.

However, the performance of the Simple Bayes method is misleading. The Simple Bayes method makes the assumption that all attributes are independent of each other. This is clearly not the case and other research [23] has shown that given more examples the Simple Bayes method can become inconsistent, whereas most other methods improve with more training data. In the abdominal pain database there are only two examples with identical attributes. These examples also have identical classifications. If the classifications were not identical, this would lead to inconsistencies in the Simple Bayes method, whereas the Soft Margin Support Vector machine does not suffer from this. The Simple Bayes method can also suffer inconsistencies from examples that are not identical because it simply counts the occurrences of attributes independently.

As our performance was lower than the best previous result by 0.2% we tried using an automated heuristic approach to find an optimal value for C. The ad hoc heuristic we used was the adjustment of C as will be described for the regression experiments later. This heuristic tries to find an optimal C by slowly lowering it. We found that after three months of continously running the algorithm had not even found an optimal C for the first binary machine. This is due to the fact that when C is lowered the training time becomes longer and there are more Support Vectors and the problem may even become impossible to solve.

7.2.7 Summary

At the beginning of this chapter we aimed to investigate the differing performance of the kernels we used. Although for each experiment one kernel did produce the best result there was no clear trend to be seen. In four of the multi-class experiments the spline kernel produced the best result. Only in two experiments a polynomial kernel produced the best result. In the binary experiments however the polynomial kernels always produced the best results.
Examining the feature space and the space the classification function occupy does give some clues to why this happens. The space the classification function occupies is determined by the number of Support Vectors. The Support Vectors span the space the classification function lives in and the number of Support Vectors represents an upper limit on the dimensionality of the space spaned. In all experiments presented here the dimensionality of the feature space is larger

than the number of Support Vectors. This means the feature space is never fully used and it is possible that for all kernels similar subspaces are used by the classification function. Further investigation is needed here, which should also investigate examples in which the results vary widely between different kernels. At this stage we cannot draw any definite conclusions, however in the experiments we have run the spline kernels have always given reasonable results which leads us to recommend the use of spline kernels as a starting point for any experiment.

We have seen that prior knowledge can improve results as in the Iris data set especially in combination the fine tuning the parameter C. However fine tuning C is not trivial, as improving one of the binary classifiers has a knock on effect on the others (Wine, Abdominal Pain). In theory it is possible to use leave one out cross-validation to find a good value of C, in practice however this can take too long if the training set is too big, as many possibilities have to be tested. We have observed that the error rate for a single kernel can not be seen as a smooth monotonic function of C.

We have looked at the way chunking affects the speed and memory usage of the Support Vector machine. There are rules when using chunking and which type of chunking to use, and although they depend on the number of Support Vectors in the solution they still give a good guideline. If the number of examples is below about 500 no chunking is usually necessary although this can be larger in some cases. If there are more than about 500 examples, some form of chunking has to be used. The old-style chunking always keeps all Support Vectors in the chunk, which means that it becomes impractical to use when there are more than about 500 Support Vectors. The new-style chunking does not keep all Support Vectors in the current chunk which makes it appropriate when there are more than about 500 Support Vectors. The new-style method however always uses a fixed number of examples in its optimization task making it slower when there are in fact fewer Support Vectors than the chunk can hold.

In general the choice of multi-class method did not affect the results significantly. However we have observed a clear advantage in speed when processing large data sets when using the "one against one" method. The "one against all others method" is slower in most cases. The very experimental "Hamming" method is the slowest of all though, which can be easily explained when the binary classifiers' training sets are observed (USPS). In the Diabetes data set we observed that although chunking does not have any speed advantage as the data set is relatively small, we obtained far better memory usage, which can be beneficial in multi-tasking systems.

We have seen that the data has to be in a suitable format (Glass, Abdominal Pain) and this can affect the performance. Some data cannot be processed with the Support Vector machine or many other algorithms in its initial form.

Some of the limits of Support Vector machine implementations have become clear. As we are working with finite accuracy machines, we have to carefully control the ranges all values lie in. In the case of the Soy data set the optimizer could not solve the problem set and in the case of Abdominal Pain the kernels produced values which were not suited to the problem. However we can detect these problems and we can solve them, so that the Support Vector machine can still be used.

For all the experiments we have made use of our implementations user interface, which allowed us the change the parameters of the Support Vector machine easily and make small adjustments quickly, so that different kernels and C values could be tested efficiently.

In general we have shown that the Support Vector machine is a viable and useful algorithm.

7.3 Regression Estimation

We will start by looking at a set of toy examples for regression using the Support Vector machine.

This is followed by a "real-world" application of Support Vector regression where we pick up our investigation of special kernels again.

7.3.1 Simple Regression Examples

To show a simple example without noise we have chosen the so-called Mexican Hat function:

$$y = \frac{sin(||\mathbf{x}||)}{||\mathbf{x}||} \tag{7.1}$$

where $\mathbf{x} \in \mathbf{R}^n$ and $y \in \mathbf{R}$.

This function was used to generate equidistant data points in one and two dimensional space in the range $[-10:10]$.

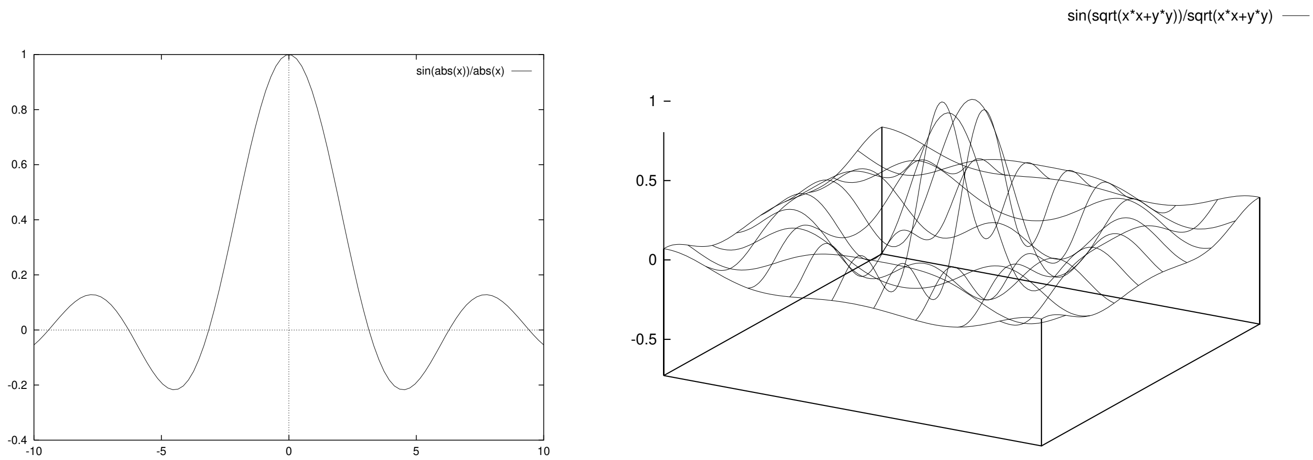

Figure 7.2: Mexican Hat function

We then ran the Support Vector machine using spline kernels with different accuracies ϵ in one-dimensional input space using 100 training points.

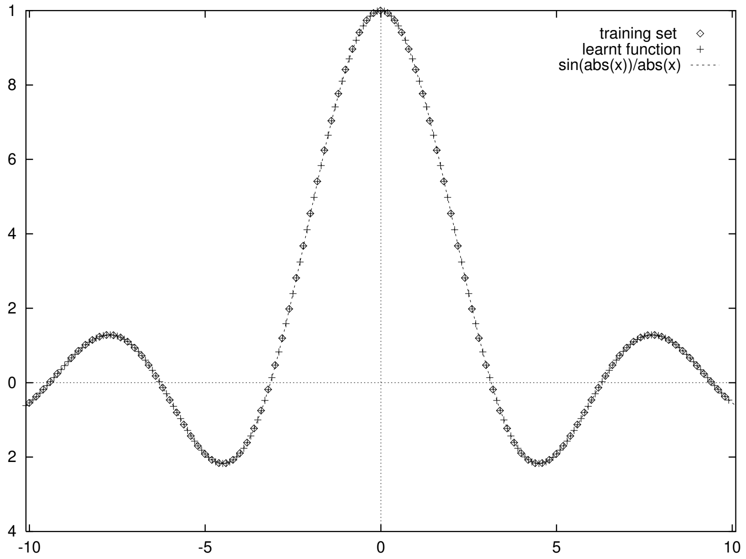

Figure 7.3: Mexican Hat - $\epsilon = 0$

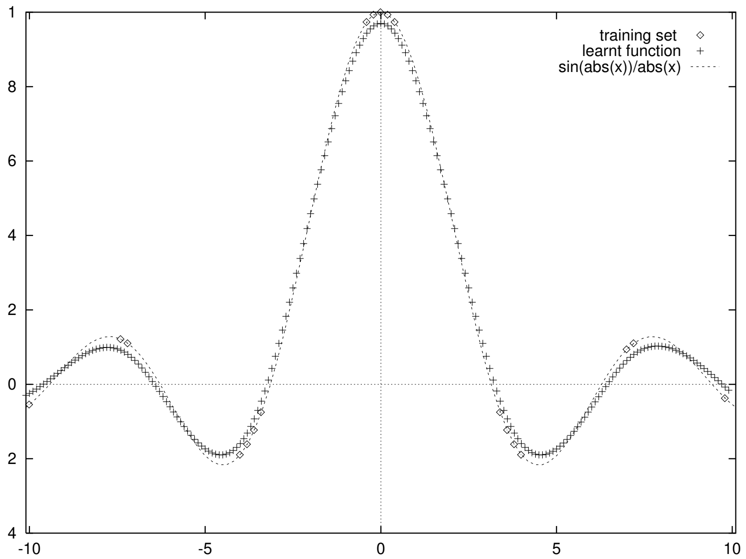

Figure 7.4: Mexican Hat - $\epsilon = 0.03$

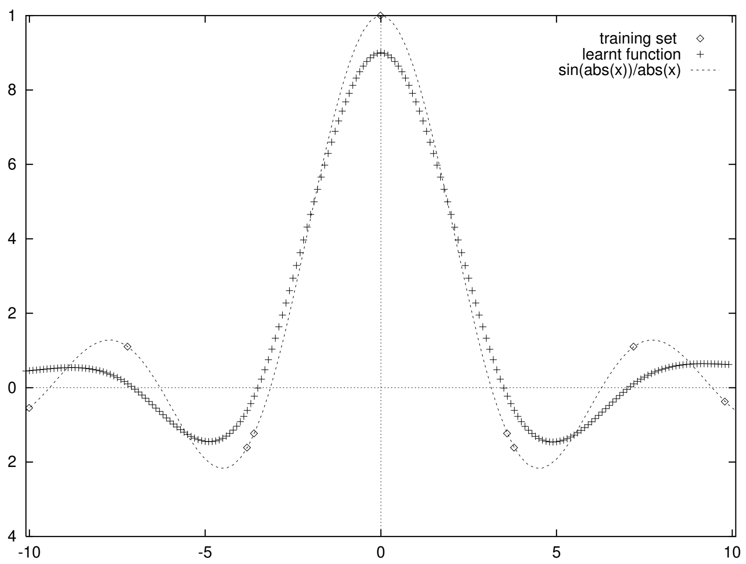

Figure 7.5: Mexican Hat - $\epsilon = 0.1$

ϵ	Number of SVs
0	100
0.01	35
0.03	19
0.05	12
0.1	9
0.2	7

Table 7.29: Mexican Hat - Support Vectors vs. ϵ

Table 7.29 shows clearly that with increasing ϵ, i.e. decreasing accuracy, the number of Support Vectors needed to approximate a function decreases.

Figures 7.3, 7.4 and 7.5 show how far the approximation deviates from the actual function.

As a second example we chose to keep the accuracy fixed and vary the number of training points. This example uses a two dimensional input space, which gives more opportunity to increase the number of training examples.

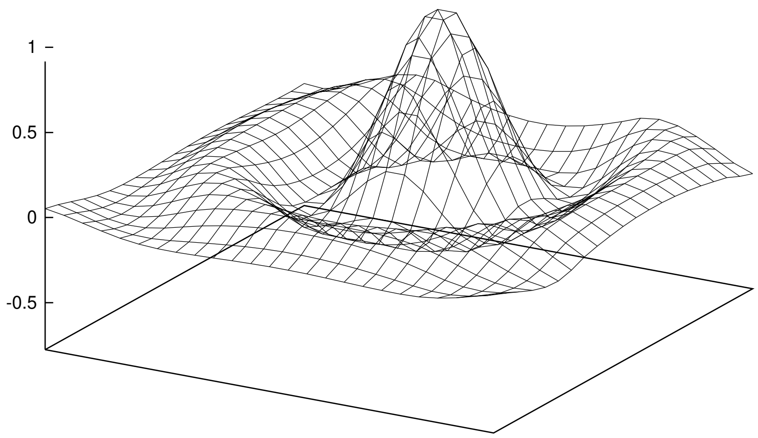

Figure 7.6: Mexican Hat 2d - $\epsilon = 0.03$

Training set size	Number of SVs
400	100
2025	201
7921	267

Table 7.30: Mexican Hat 2d - Training examples vs. Support Vectors

Table 7.30 shows that although the number of training examples increases very fast the number of Support Vectors does not. This means that more training examples do not necessarily result in a more complex solution.

These toy examples show just a few general properties of the regression Support Vector machine. We will look at a practical application in the next section.

7.3.2 Boston Housing

The following experiments were conducted using the Boston housing data from the UCI machine learning repository [6]. This has been frequently used as a benchmark data set for non-linear regression estimation.

The data set consists of 506 cases. Each case consists of 13 socio-economic and environmental factors - such as crime rate, nitric oxide concentration, distance to employment centres and age of a property - determining the median house price in a certain area of Boston. The prices lie between \$5,000 and \$50,000 to the nearest \$100. The data set has been used frequently to evaluate regression estimation techniques, for example Bagging [3] and Polynomial kernel Support Vector machines [14].

Breiman in [3] uses his Bagging technique on regression trees. This technique aggregates a set of unstable predictors, generated from subsets of a training set, to form a single more stable predictor with a lower error rate. [14] compare Support Vector machine using polynomial kernels to Breiman's Bagging.

7.3.2.1 Method

Following [3] and [14] the data is split into 481 training points and 25 test points randomly 100 times. This gives 100 independent trials on which an average squared error rate is measured.

The ANOVA spline kernel, the spline kernel (6.4) and a polynomial kernel $K(\mathbf{x}, \mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y}) + 1)^d$ similar to [14] are tested here. These results also include the variance of the average squared error for each trial over the hundred trials, as this gives an indication of the stability of the method.

On each trial the training points were split into a training set of 401 points and a validation set of 80 points. The validation set was used to determine the Support Vector machine parameters ϵ from (2.36) and C from (2.43), and the kernels' parameters (order of the ANOVA decomposition or degree of polynomial d).

The input data was scaled linearly for all kernels, the target values were not scaled. There was no other pre-processing, even though several attributes appeared to be suitable for non-linear re-scaling. Spline kernels require all data to be positive (see appendix A.2), so for both spline kernels all input values were scaled to lie between 0 and 1. Polynomial kernels generally perform better if the data is centred around 0, so for the polynomial kernel all input values were scaled to lie between -1 and 1.

To find the optimal parameters for the Support Vector machine, a set of Support Vector machines was created for each trial for each of the three types of kernels and for each type of kernel the Support Vector machine from that set with the lowest error on the validation set was selected. The error rate of the selected Support Vector machine was then measured on the test set.

For each kernel the tolerance ϵ is set to \$1,000, \$2,000, \$3,000, \$4,000 and \$5,000 in turn.

For the ANOVA kernel the set of Support Vector machines is created by choosing the orders 2,4,6,8,10 and 13 in turn with each of the above tolerances ϵ. C is optimized as described below for each of these 30 Support Vector machines.

For the linear spline kernel a Support Vector machine is created for each ϵ above and C is optimized as described below for each of the 5 Support Vector machines.

For the polynomial kernel, Support Vector machines with kernels of degree $d = 4$ and $d = 5$ are created for each of the above ϵ and C is optimized for each of the 10 Support Vector machines as described below.

C was optimized as follows. The error rate on the validation set computed as a function of C is approximately a convex, but non-smooth function. A heuristic method similar to golden section search was used to find the minimum of this function and thus the optimum value of C. There is no guarantee that this works, but it is a useful heuristic, which achieves very good results at the cost of being a lengthy and time consuming process.

7.3.2.2 Results

The error rates are calculated for each trial as the average square error over the test sets:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 \qquad (7.2)$$

where $n = 25$ is the number of cases in the test set of each trial and the unit for $f(\mathbf{x}_i)$ and y_i is \$1,000, i.e. a deviation of 4 corresponds to \$4,000 and the unit of the average squared error is $(\$1,000)^2$.

The average and variance of (7.2) is then calculated over all 100 trials:

Kernel	Avg. Square Error	Variance
Bagging [3]	11.7	
Polynomial	8.28	24.02
Splines	7.87	12.67
ANOVA Splines	7.56	8.70

Table 7.31: Error rates on the test set

The experiments produced better results for the Support Vector machines than for the Bagging results obtained in [3]. Among the three different types of Support Vector machine kernel the ANOVA splines performed slightly better than the other kernels; as did the variance of the error rates on the test set, but neither of these effects is statistically significant[9].

7.3.2.3 Summary

We have shown some general properties of the regression estimation Support Vector machine at the start of this section.

A Support Vector machine using ANOVA decomposition kernels gave a better result than a Support Vector machine using other kernels on the Boston housing data and performed more reliably. The accuracy on all kernels is not bad though compared to the value of ϵ. In this examples we cannot recommend a single kernel as the best in general, but again the spline kernel seems attractive as it needs no parameters.

The low variance observed also indicates a slightly more stable method for the ANOVA kernel than with the other kernels. The Support Vector machine also yields results which are better than previously reported results with other regression methods on the same data.

[9] More detailed results can be found in [61].

7.4 Choosing Parameters for SVMs

The Support Vector machine software has many parameters. This section gives some general rules for using the software and how to detect and partially solve problems in using the software. It applies to any Support Vector machine implementation and the advice given here was gained from running not only the over 500 experiments presented in this thesis, but also a much larger number of experiments while writing and testing the software.

7.4.1 Kernels

From the kernels in appendix A, the polynomial kernel, the spline kernel and the radial basis function kernel are the most widely used. In this thesis we have only used the polynomial kernel and the spline kernel. The radial basis function kernel is harder to use as it has a real valued parameter, whereas the polynomial has a integer parameter and the spline kernel has no parameter at all. We usually observe very good results from the spline kernel and use this as a starting point for most of our experiments.

7.4.2 Data

When using any kernel it is important to analyse the output from the kernel function to see what the extreme values are for the given data set. If the ranges of values are too large and the alpha values are too small, numerical inaccuracies might make the problem impossible to solve.

In the case of the spline kernel we usually scale the data to lie between 0 and 1. In this way the output values from the kernel function stay within a reasonable range. In the case of polynomial kernels we found that scaling the data between -1 and 1 and then using the scale factor in the kernel to bring the output of the kernel function down to the -1 to 1 range brought the best results.

In some cases we have had to scale the data for spline kernels between 0.25 and 0.75 to keep the output for higher dimensional input spaces within reasonable limits.

Most errors involving the scaling of data or kernels producing too wide output ranges are picked up by the optimizer and flagged up as errors.

7.4.3 Value of C or the Box Constraint

If we are interested in data that are not error free we have to adjust the box constraint determined by the parameter C. C is a bound on the Lagrange multipliers α. To find a suitable value of C it is therefore possible to find an upper limit for C by running the Support Vector machine without any constraint on the Lagrange multipliers and using the largest value of the Lagrange multipliers as an upper limit on C, i.e. if C is set to that value the solution will not change. Analysing the α values now enables us to adjust C. If we know that there are approximately 10% errors in the training data, we can adjust C downwards to cut out that many support vectors. A more straightforward approach is to look at the outliers among the α values. Often there will be a few very small values and a few very large values, whereas most of the Support Vectors' α values will be in a small range somewhere in the middle. These outliers are usually vectors which are misclassified. It is desirable to remove these outliers by adjusting C. The outliers at the lower end are not removed in this way, as they contribute only a negligible amount to the solution anyway.

7.4.4 Epsilon for Regression

The value of epsilon determines the level of accuracy of the approximated function. It relies entirely on the target values in the training set. If epsilon is larger than the range of the target values we cannot expect a good result. If epsilon is zero, we can expect overfitting. Epsilon must therefore be chosen to reflect the data in some way. Choosing epsilon to be a certain accuracy does of course only guarantee that accuracy on the training set; often to achieve a certain accuracy overall, we need to choose a slightly smaller epsilon.

Say we have a problem like the Boston housing problem discussed earlier. In this problem the target values range from 5 to 50. These values represent $5000 to $50000. From these values we can deduct a fixed accuracy needed either as a percentage or as an absolute value; both are possible using our software. In house prices like these we would not want a deviation in our prediction of more than 10% or if we use an absolute value say \$3000. To achieve the accuracy in the absolute case we could set epsilon to 3. This will give us an error of 3 or less on all the training data. However this does not affect the test data directly, although it does affect the test data through the fact that the training and test data should come from the same source. Often to achieve the required accuracy on the test data, a higher accuracy has to be choosen on the training data. For the Boston housing problem this could translate to using an epsilon value of 2 to try and achieve an overall accuracy of 4, i.e. \$4000.

7.4.5 Settings for Chunking

The basic limit to the Support Vector machine is given by the optimizers used. The optimizers become infeasibly slow when they have to optimize over 500 or more training vectors. Thus if the training set contains more than about 500 vectors chunking becomes necessary. The choice of chunking depends on the number of Support Vectors in the solution. If the number of Support Vectors in the solution is less than 500 old-style chunking is more efficient than new-style chunking. However new-style chunking will work regardless of the number of Support Vectors in the solution. One rule of thumb is that very often only 10% of the training vectors become Support Vectors and base the decision for which style of chunking to use on that. Another possibility is to initially run the problem with the old-style method and observe the first few iterations to see how the number of Support Vectors develops and then deciding whether to continue with the old-style method or to switch to the new-style method.

7.4.6 Parameter Selection Heuristics

The parameter selection heuristics are meant as a guide to potential users of the software. I have crystallised these out from running many experiments over the course of my research. This advice may look daunting but is in fact very easy to follow in practice.

These heuristics are a starting point for further research into the choice of parameters, which is important for implementing production systems.

Chapter 8

Conclusions

In this thesis we have looked at pattern recognition and regression estimation in machine learning using the Support Vector machine.

Initially we looked at the problem formulation for pattern recognition and regression estimation. We have looked at a range of methods including decision tree algorithms and Bayesian approaches. After this background research we turned to Support Vector machines and studied their underlying theory. From this I have developed an implementation which I tested on small benchmark datasets.

1. I have shown the basic design of our specific Support Vector machine program. I have shown the structure and object oriented design features. The program is modular and versatile in terms of software engineering, as well as implementing a range of Support Vector machine features. The software is capable of pattern recognition and regression estimation.

2. To use larger datasets, I developed a set of heuristics to overcome the limitations of modern digital computers. The main problems with modern digital computers are limited memory and limited speed. I have developed heuristics for two approaches which allow decomposition of the Support Vector machine algorithm. Both of these approaches use less storage and are faster than the original formulation for large datasets.

3. I have also investigated some problem specific kernels. We have used the Support Vector machine on the "Boston housing problem". This is a hard

regression benchmark with noisy data. We achieved extremely good results and improved further on these results by using problem specific "ANOVA decomposition" kernels.

4. I have tackled multi-class classification problems by using combinations of binary pattern recognition Support Vector machines. I have described different possible approaches to this and implemented three of them. I have described how they fit into the design of our implementation. The implementation has been tested with some small benchmark data sets, the US Postal service ZIP code database of handwritten digits and a medical database on abdominal pain.

5. In this I have shown how different kernels behave. I have analysed speed and memory usage and presented rules on how to improve these. I have investigated the use of the parameter C in the Support Vector machine and given heuristics on how to tune it. I have compared different multi-class approaches and discussed some of their merits and failings. And finally I have shown how to deal with the practical implications of numerical accuracy in the Support Vector machine implementation.

This research makes it possible to address a number of questions in future research:

1. Can we choose a kernel on the basis of the data a priori?
2. How can we construct problem specific kernels in a structured way?
3. How can we best adjust the parameter C in the Support Vector machine without using heuristics?

These questions can be investigated and possible solutions can be evaluated using our Support Vector machine implementation.

I have contributed a number of small but important improvements to the field of Support Vector machines. The most useful lasting result from the research for this PhD thesis is the software implemented. The software which was only possible due to my research and the research of my collegues, has opened the field to many more researchers in related areas. Without the software which is based on my research, far fewer researchers would have found out about Support Vector machines let alone used them. At the start of our research the Support Vector machine community numbered just a few researchers world-wide. Today there are hundreds who are actively participating in the Support Vector research. They range from academia to industry world-wide. Many of them use our software without which few large preoblems could be attempted. On the basis of our Support Vector machine implementation a JAVA applet has been developed. This web site receives over 400 hits per day and over 100 researchers download our software per month.

I hope that the Support Vector method will be a topic of research for many years to come. Many of the Machine Learning conferences now include Support Vector workshops and a number of papers I have read now use the Support Vector method as an established benchmark which to test new ideas against. After appearing in the 1960s the Support Vector method is now well established and our research group has had an active part in this.

Appendix A

SVM Kernels

A.1 Standard Kernels

The numbers on the following kernels refer to the indexes used in the software.

- 1. The simple dot product:

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$$

- 2. The simple polynomial kernel:

$$K(\mathbf{x}, \mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y}) + 1)^d$$

  known as "Vapnik Polynomial" or alternatively[1]

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$$

  where d is a user defined integer.

  Taken from [71].

- 3. Vovk's real polynomial:

[1]The determinant of the hessian can become zero, which makes solving the Lagrangian impossible in this alternative kernel, which is why it is seldom used.

$$K(\mathbf{x}, \mathbf{y}) = \frac{1 - (\mathbf{x} \cdot \mathbf{y})^d}{1 - (\mathbf{x} \cdot \mathbf{y})}$$

where d is user defined and where $-1 < (\mathbf{x} \cdot \mathbf{y}) < 1$.

From private communications with V. Vovk.

- 4. Vovk's real infinite polynomial:

$$K(\mathbf{x}, \mathbf{y}) = \frac{1}{1 - (\mathbf{x} \cdot \mathbf{y})}$$

where $-1 < (\mathbf{x} \cdot \mathbf{y}) < 1$.

From private communications with V. Vovk.

- 5. Radial Basis function:

$$K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma ||\mathbf{x} - \mathbf{y}||^2)$$

where γ is user defined.

Taken from [71].

- 6. Two layer neural network:

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\frac{b(\mathbf{x} \cdot \mathbf{y})}{1} - c)$$

where b and c are user defined.

Taken from [71].

- 7. Linear splines with an infinite number of points:

How these come about can be found in section A.2.

For the one-dimensional case:

$$K_k(x_i, x_j) = 1 + x_i x_j + x_i x_j \min(x_i, x_j) - \frac{x_i + x_j}{2} (\min(x_i x_j))^2 + \frac{(\min(x_i, x_j))^3}{3}$$

where $x_i \geq 0$ and $x_j \geq 0$

For the multi-dimensional case

$$K(\mathbf{x}, \mathbf{y}) = \prod_{k=1}^{n} K_k(x^k, y^k)$$

Taken from [69].

- 8. Full polynomial kernel:

$$K(\mathbf{x}, \mathbf{y}) = \left(\frac{\mathbf{x} \cdot \mathbf{y}}{a} + b\right)^d \tag{A.1}$$

where a, b and d are user defined.

From [71] and generalized.

- 9. Regularized Fourier (weaker mode regularization)

For the one-dimensional case:

$$K_k(x_i, x_j) = \frac{\pi}{2\gamma} \frac{\cosh \frac{\pi - |x_i - x_j|}{\gamma}}{\sinh \frac{\pi}{\gamma}}$$

where $0 \leq |x_i - x_j| \leq 2\pi$ and γ is user defined.

For the multi-dimensional case

$$K(\mathbf{x}, \mathbf{y}) = \prod_{k=1}^{n} K_k(x^k, y^k)$$

From [69] and [72].

- 10. Semi Local Kernel

$$K(\mathbf{x}, \mathbf{y}) = [(\mathbf{x} \cdot \mathbf{y}) + 1]^d \exp(-||\mathbf{x} - \mathbf{y}||^2 \sigma^2)$$

where d and σ are user defined and weight between global and local approximation.

From private communications with V. Vapnik.

- 11. Regularized Fourier (stronger mode regularization)

For the one-dimensional case:

$$K_k(x_i, x_j) = \frac{1 - \gamma^2}{2(1 - 2\gamma \cos(x_i - x_j) + \gamma^2)}$$

where $0 \leq |x_i - x_j| \leq 2\pi$ and γ is user defined.

For the multi-dimensional case

$$K(\mathbf{x}, \mathbf{y}) = \prod_{k=1}^{n} K_k(x^k, y^k)$$

From [69] and [72].

- 17. Anova 1

$$K(\mathbf{x}, \mathbf{y}) = (\sum_{k=1}^{n} \exp(-\gamma(x^k - y^k)^2))^d$$

where the degree d and γ are user defined.

From private communications with V. Vapnik.

- Another interesting function, which might work is

$$K(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{l} (\mathbf{x} - \mathbf{t}_k)(\mathbf{y} - \mathbf{t}_k) \tag{A.2}$$

where t_ks are from the training data. This however is not a function determined a priori and thus is not strictly a proper dot product or kernel until the training data is available.

From private communications with V. Vapnik.

A.2 Spline kernels

This section explains how spline kernels correspond to spline functions.

Splines are piecewise polynomial functions. The pieces are polynomial functions of a certain degree d. The number of pieces $l+1$ is fixed by a given set of $l+1$ knot points $t_0, \ldots, t_l$

$$\text{spline}_d(x) = \sum_{r=0}^{l} \gamma_r (x - t_r)_+^d + \sum_{r=0}^{d} \gamma_r^* x^r$$

If $\gamma_r = (y - t_r)_+^d$ and $\gamma_r^* = y^r$, then $\text{spline}_d(x)$ can be re-written as

$$f(x) = \sum_{i=1}^{n} (\alpha_i^* - \alpha_i) \kappa_d(x, x_i)$$

where

$$\kappa_d(x, y) = \sum_{r=0}^{l} (x - t_r)_+^d (y - t_r)_+^d + \sum_{r=0}^{d} x^r y^r$$

and $\kappa(x, y)$ can be considered as an inner product.

In general, a sum of terms of this type is needed, as it is not possible to match $\gamma_0, \ldots, \gamma_l$ and $\gamma_0^*, \ldots, \gamma_l^*$ in the original spline using a single value of y.

This has been generalised to deal with an infinite number of nodes on the interval $[0, a]$, $0 < a \leq \infty$ [69][2]:

$$\text{spline}_d(x) = \int_0^a a(t)(x-t)_+^d dt + \sum_{r=0}^{d} \gamma_r^* x^r$$

where the unknown values γ_i^*, $i = 0, \ldots, d$, and the unknown function $a(t)$ define the expansion. This leads to the following kernel, which effectively uses the Support Vectors as knot points:

$$\kappa_d(x,y) = \int_0^{min(x,y)} (x-t)_+^d (y-t)_+^d dt + \sum_{r=0}^{d} x^r y^r$$

This is equivalent to

$$\kappa_d(x,y) = \sum_{r=0}^{d} \frac{\binom{d}{r}}{2d-r+1} \min(x,y)^{2d-r+1} |x-y|^r + \sum_{r=0}^{d} x^r y^r$$

where $\binom{d}{r} = \frac{d!}{r!(d-r)!}$.

The experiments in section (7.3) use the linear case $(d = 1)$:

$$\kappa_1(x,y) = \frac{\min(x,y)^3}{3} + \frac{\min(x,y)^2 |x-y|}{2} + 1 + xy \qquad \text{(A.3)}$$

[2]This kernel requires all values to be positive, as the knot points are all positive.

Appendix B

Lagrangians

Lagrangian functionals are a way of expressing non-linear optimization problems over convex functions with non-linear constraints as non-linear optimization problems with a set of simple linear constraints.

B.1 Pattern Recognition

Pattern recognition Support Vector machines optimize the following convex functions:

$$\min(\frac{1}{2}||\mathbf{w}||^2) \tag{B.1}$$

and

$$\min(C\sum_i \xi_i) \tag{B.2}$$

under the constraints:

$$y_i[\mathbf{w} \cdot \mathbf{x}_i + b] \geq 1 - \xi_i \tag{B.3}$$

$$\xi_i \geq 0, \qquad i = 1, \dots, l \tag{B.4}$$

Lagrangians need a single convex function to optimize over, which is

$$f(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \tag{B.5}$$

This has to be minimized under the constraints

$$f_i(\mathbf{w}) = -\xi_i - y_i[\mathbf{w} \cdot \mathbf{x}_i + b] + 1 \leq 0 \tag{B.6}$$

$$\xi_i \geq 0, \qquad i = 1, \dots, l \tag{B.7}$$

As the second set of constraints already is a set of simple linear constraints, we do not replace it by Lagrangian multipliers, we only replace the first set of constraints by Lagrangian multipliers:

$$L(\mathbf{w}, \alpha, \xi) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i - \sum_i \alpha_i(\xi_i + y_i[\mathbf{w} \cdot \mathbf{x}_i + b] - 1) \tag{B.8}$$

under the constraints

$$\xi_i \geq 0, \quad \alpha_i \geq 0 \qquad i = 1, \dots, l \tag{B.9}$$

There are now $2l$ Lagrangian multipliers (ξ_is and α_is). We can simplify this by observing the following properties in the saddle point:

$$\frac{\delta L(\mathbf{w}, \alpha, \xi)}{\delta b} = 0 \tag{B.10}$$

and

$$\frac{\delta L(\mathbf{w}, \alpha, \xi)}{\delta w} = 0 \tag{B.11}$$

This is equivalent to

$$\sum_i \alpha_i y_i = 0 \tag{B.12}$$

and

$$w = \sum_i y_i \alpha_i \mathbf{x}_i \tag{B.13}$$

Substituting these leads to the following Lagrangian:

$$L(\mathbf{w}, \alpha, \xi) = \sum_i \alpha_i - \frac{1}{2} \sum_i y_i y_j \alpha_i \alpha_j \mathbf{x}_i \mathbf{x}_j + \sum_i \xi_i (C - \alpha_i) \tag{B.14}$$

As the term $\sum_i \xi_i(C - \alpha_i)$ has to be minimized over ξ_i and in fact is 0 in the saddle point (no error ξ_i or $\alpha_i = C$), ξ_i has to be zero if $\alpha_i < C$ and ξ_i would have to be infinity if $\alpha_i \geq C$. As no ξ_i actually can be infinity, this term can be dropped, given the additional constraint

$$\alpha_i \leq C \qquad , i = 1, \dots, l \tag{B.15}$$

That is the ξ_i become irrelevant to the solution, if the α_i are bounded between 0 and C:

$$L(\mathbf{w}, \alpha) = \frac{1}{2} ||\mathbf{w}||^2 - \sum_i \alpha_i y_i [\mathbf{w} \cdot \mathbf{x}_i + b] - 1 \tag{B.16}$$

under the constraints

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, l \tag{B.17}$$

or

$$L(\mathbf{w}, \alpha, \xi) = \sum_i \alpha_i - \frac{1}{2} \sum_i y_i y_j \alpha_i \alpha_j \mathbf{x}_i \mathbf{x}_j \tag{B.18}$$

under the constraints

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, l \tag{B.19}$$

B.2 Regression Estimation

Regression estimation Support Vector machines optimize the following functions:

$$\min(\frac{1}{2}||\mathbf{w}||^2) \tag{B.20}$$

$$\min(C(\sum_i \xi_i^* + \sum_i \xi_i) \tag{B.21}$$

under the constraints:

$$\begin{array}{lcll} (\mathbf{w} \cdot \mathbf{x}_i) + b & \geq & y_i - \epsilon - \xi_i^* & i = 1, \ldots, l \\ (\mathbf{w} \cdot \mathbf{x}_i) + b & \leq & y_i + \epsilon + \xi_i & i = 1, \ldots, l \\ \xi_i^* & \geq & 0, & i = 1, \ldots, l \\ \xi_i & \geq & 0, & i = 1, \ldots, l \end{array} \tag{B.22}$$

Our convex function is

$$f(\mathbf{w}) = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C\big(\sum_{i=1}^{l} \xi_i^* + \sum_{i=1}^{l} \xi_i\big) \tag{B.23}$$

which has to be minimized under the constraints

$$\begin{array}{rcll} f_i^*(\mathbf{w}) = -(\mathbf{w} \cdot \mathbf{x}_i) - b + y_i - \epsilon - \xi_i^* & \leq & 0 & i = 1, \dots, l \\ f_i(\mathbf{w}) = (\mathbf{w} \cdot \mathbf{x}_i) + b - y_i - \epsilon - \xi_i & \leq & 0 & i = 1, \dots, l \\ \xi_i^* & \geq & 0, & i = 1, \dots, l \\ \xi_i & \geq & 0, & i = 1, \dots, l \end{array} \tag{B.24}$$

As the second set of constraints already is a set of simple linear constraints, we do not replace it by Lagrangian multipliers and only replace the first set of constraints by Lagrangian multipliers:

$$\begin{array}{rr} L(\mathbf{w}, \alpha, \alpha^*, \xi, \xi^*) = & \frac{1}{2}||\mathbf{w}||^2 + C(\sum_i \xi_i + \sum_i \xi_i^*) \\ & + \sum_i \alpha_i^*[-(\mathbf{w} \cdot \mathbf{x}_i) - b + y_i - \epsilon - \xi_i^*] \\ & + \sum_i \alpha_i[(\mathbf{w} \cdot \mathbf{x}_i) + b - y_i - \epsilon - \xi_i] \end{array} \tag{B.25}$$

or

$$\begin{array}{rr} L(\mathbf{w}, \alpha, \alpha_*, \xi, \xi^*) = & \frac{1}{2}||\mathbf{w}||^2 - \sum_i(\alpha_i^* - \alpha_i)[(\mathbf{w} \cdot \mathbf{x}_i) + b - y_i] - \epsilon \sum_i(\alpha_i^* + \alpha_i) \\ & + \sum_i \xi_i(C - \alpha_i) + \sum_i \xi_i^*(C - \alpha_i^*) \end{array} \tag{B.26}$$

under the constraints

$$\xi_i \geq 0, \quad \xi_i^* \geq 0, \quad \alpha_i \geq 0, \quad \alpha_i^* \geq 0 \qquad i = 1, \dots, l \tag{B.27}$$

There are now $4l$ Lagrangian multipliers (ξ_is, ξ_i^*s, α_is and α_i^*s). We can simplify this by observing the following properties in the saddle point:

$$\frac{\delta L(\mathbf{w}, \alpha, \alpha^*, \xi, \xi^*)}{\delta b} = 0 \tag{B.28}$$

and

$$\frac{\delta L(\mathbf{w}, \alpha, \alpha^*, \xi, \xi^*)}{\delta w} = 0 \tag{B.29}$$

This is equivalent to

$$\sum_i \alpha_i = \sum_i \alpha_i^* \tag{B.30}$$

and

$$\mathbf{w} = \sum_i (\alpha_i^* - \alpha_i)\mathbf{x}_i \tag{B.31}$$

Substituting these leads to the following Lagrangian:

$$\begin{aligned} L(\mathbf{w}, \alpha, \alpha_*, \xi, \xi^*) = \quad & -\tfrac{1}{2}\sum_i \sum_j (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(\mathbf{x}_j \cdot \mathbf{x}_i) \\ & + \sum_i (\alpha_i^* - \alpha_i) y_i - \epsilon \sum_i (\alpha_i^* + \alpha_i) \\ & + \sum_i \xi_i (C - \alpha_i) + \sum_i \xi_i^* (C - \alpha_i^*) \end{aligned} \tag{B.32}$$

As the term $\sum_i \xi_i (C - \alpha_i)$ has to be minimized over ξ_i and in fact is 0 in the saddle point (no error ξ_i or $\alpha_i = C$), ξ_i has to be zero if $\alpha_i < C$ and ξ_i would have to be infinity if $\alpha_i \geq C$. As no ξ_i actually can be infinity, this term can be dropped, given the additional constraint

$$\alpha_i \leq C \qquad , i = 1, \ldots, l \tag{B.33}$$

The same holds for ξ^* and the ξ_i and ξ_i^* become irrelevant to the solution, if the α_i and α_i^* are bounded between 0 and C:

$$L(\mathbf{w}, \alpha, \alpha_*) = -\frac{1}{2}\sum_i \sum_j (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(\mathbf{x}_j \cdot \mathbf{x}_i) + \sum_i (\alpha_i^* - \alpha_i) y_i - \epsilon \sum_i (\alpha_i^* + \alpha_i) \tag{B.34}$$

under the constraints

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, l \tag{B.35}$$

$$0 \leq \alpha_i^* \leq C \qquad i = 1, \ldots, l \tag{B.36}$$

Appendix C

Detailed Experimental Results

This appendix contains detailed results for the experiments presented in chapter 7.

The tables contain results for each individual cross-validation experiment and a total. The cross-validation sets are generated as described in section 7.1.

C.1 Heart Disease

Cross	Test set	SV Machine				
Validation	size	Spline	Poly 4	5	6	7
1	30	9	9	8	8	8
2	30	7	6	5	5	4
3	30	7	7	8	8	9
4	30	11	10	9	8	7
5	30	6	4	5	4	4
6	30	6	5	6	5	5
7	30	9	12	13	13	13
8	30	6	6	5	8	7
9	30	6	7	7	7	7
Total	270	67	66	66	66	64
		24.8%	24.4%	24.4%	24.4%	23.7%

continued on next page

continued from previous page						
Cross	Test set	SV Machine				
Validation	size	Spline	Poly 4	5	6	7
Avg.		7.4	7.3	7.3	7.3	7.1
Std.		1.7	2.4	2.4	2.5	2.6

Table C.1: Errors on Test set (9×30)

C.2 Diabetes

Cross	Test set	SV Machine				
Validation	size	Spline	Poly 4	5	6	7
1	64	22	18	18	17	16
2	64	19	19	18	18	18
3	64	23	21	19	17	18
4	64	20	24	22	22	22
5	64	20	22	23	23	22
6	64	18	19	19	18	20
7	64	23	21	19	18	18
8	64	16	23	22	20	19
9	64	20	19	18	20	19
10	64	20	25	23	22	23
11	64	23	24	25	27	26
12	64	17	21	21	18	19
Total	768	241	256	247	240	240
		31.4%	33.3%	32.2%	31.3%	31.3%
Avg.		20.1	21.3	20.6	20.0	20.0
Std.		2.3	2.2	2.3	2.9	2.6

Table C.2: Errors on Test set (12×64)

C.3 Iris

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
1	15	0	0	0	0	0
2	15	0	0	0	0	0
3	15	0	0	0	0	0
4	15	0	0	0	0	0
5	15	1	2	2	2	2
6	15	1	2	2	2	2
7	15	1	0	1	1	1
8	15	0	1	0	0	0
9	15	2	2	2	2	2
10	15	0	1	1	1	0
Total	150	5	8	8	8	7
		3.3%	5.3%	5.3%	5.3%	4.6%
Avg.		0.5	0.8	0.8	0.8	0.7
Std.		0.7	0.9	0.9	0.9	0.9

Table C.3: Errors on Test set for "1 against 1"(10 × 15)

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
1	15	0	0	0	0	0
2	15	0	1	1	1	1
3	15	0	0	0	0	0
4	15	0	1	0	0	0
5	15	1	2	2	2	2
6	15	1	2	2	2	2
7	15	1	0	0	1	1
8	15	0	1	0	0	0
9	15	2	2	2	2	2
10	15	0	1	1	1	0
Total	150	5	10	8	9	8
		3.3%	6.6%	5.3%	6.0%	5.3%

continued on next page

continued from previous page

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
Avg.		0.5	1.0	0.8	0.9	0.8
Std.		0.7	0.8	0.9	0.8	0.9

Table C.4: Errors on Test set for “1 against the rest” (10×15)

C.4 Wine

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
1	25	0	1	0	1	2
2	25	0	0	0	0	0
3	25	1	1	1	1	1
4	25	1	3	3	3	3
5	25	1	2	1	1	1
6	25	0	0	0	1	1
7	25	0	0	0	0	1
8	25	1	0	0	0	0
9	25	0	1	1	1	1
10	25	0	1	1	1	1
Total	250	4	9	7	9	11
		1.6%	3.6%	2.8%	3.6%	4.4%
Avg.		0.4	0.9	0.7	0.9	1.1
Std.		0.5	0.9	0.9	0.8	0.8

Table C.5: Errors on Test set for “1 against all others” (10×25)

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
1	25	0	0	0	1	2

continued on next page

continued from previous page

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
2	25	1	0	0	0	0
3	25	0	1	1	1	1
4	25	2	2	2	2	3
5	25	0	1	1	1	1
6	25	0	0	0	1	1
7	25	0	0	0	0	1
8	25	2	0	0	0	0
9	25	0	1	1	1	1
10	25	0	1	1	1	1
Total	250	5	6	6	8	11
		2.0%	2.4%	2.4%	3.2%	4.4%
Avg.		0.5	0.6	0.6	0.8	1.1
Std.		0.8	0.7	0.7	0.6	0.8

Table C.6: Errors on Test set for "1 against 1"(10×25)

C.5 Glass

Cross Validation	Test set size	SV Machine Spline	Poly 4	5	6	7
1	25	6	8	7	7	8
2	25	8	10	9	9	9
3	25	9	11	11	10	10
4	25	8	10	9	9	9
5	25	8	8	8	8	7
6	25	8	10	10	10	10
7	25	8	9	9	9	9
8	25	6	12	12	11	9
9	25	7	7	7	7	7
10	25	6	8	8	8	10
Total	250	74	93	90	88	86

continued on next page

continued from previous page

Cross Validation	Test set size	SV Machine Spline	Poly 4	5	6	7
		29.6%	37.2%	36.0%	35.2%	34.4%
Avg.		7.4	9.3	9.0	8.8	8.6
Std.		1.0	1.5	1.5	1.2	1.1

Table C.7: Errors on Test set for "1 against all others"(10×25)

Cross Validation	Test set size	SV Machine Spline	Poly 4	5	6	7
1	25	6	6	6	6	6
2	25	7	8	7	7	8
3	25	8	9	9	10	10
4	25	14	12	14	13	13
5	25	8	6	6	6	7
6	25	6	9	9	9	11
7	25	8	9	10	10	10
8	25	7	6	6	6	6
9	25	8	7	7	7	7
10	25	5	8	8	8	8
Total	250	77	80	82	82	86
		30.8%	32.0%	32.8%	32.8%	34.4%
Avg.		7.7	8.0	8.2	8.2	8.6
Std.		2.3	1.8	2.4	2.2	2.2

Table C.8: Errors on Test set for "1 against 1"(10×25)

C.6 Soy

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
1	37	-	0	0	-	-
2	37	-	1	1	-	-
3	37	-	2	2	-	-
4	37	-	0	0	-	-
5	37	-	1	1	-	-
6	37	-	1	1	-	-
7	37	-	0	0	-	-
8	37	-	1	1	-	-
9	37	-	2	2	-	-
10	37	-	1	1	-	-
Total	370	-	9 2.4%	9 2.4%	-	-
Avg. Std.			0.9 0.7	0.9 0.7		

Table C.9: Errors on Test set for "1 against 1"(10 × 15)

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
1	37	-	0	-	-	-
2	37	-	0	-	-	-
3	37	-	2	-	-	-
4	37	-	0	-	-	-
5	37	-	1	-	-	-
6	37	-	1	-	-	-
7	37	-	0	-	-	-
8	37	-	1	-	-	-
9	37	-	1	-	-	-
10	37	-	1	-	-	-
Total	370	-	7 1.9%	-	-	-

continued on next page

continued from previous page

Cross Validation	Test set size	SV Machine Spline	Poly 3	4	5	6
Avg.			0.7			
Std.			0.6			

Table C.10: Errors on Test set for "1 against all others"(10×15)

Appendix D

Detailed Abdominal Results

This appendix contains the detailed results from the tests on the abdominal pain database. For every learning machine of parameters for a learning machine there is a cross-table of errors. In these tables the columns represent the actual classes and the rows represent the computer decisions. This means that all examples in column one should have been classified as class 1. The examples in row one, column one were correctly classified as class 1; the examples in row two, column one should have been classified as class 1, but were classified as class 2.

D.1 Per class error rates on the original dataset

This section contains graphs, which show the success rates of various algorithms on the original abdominal pain database.

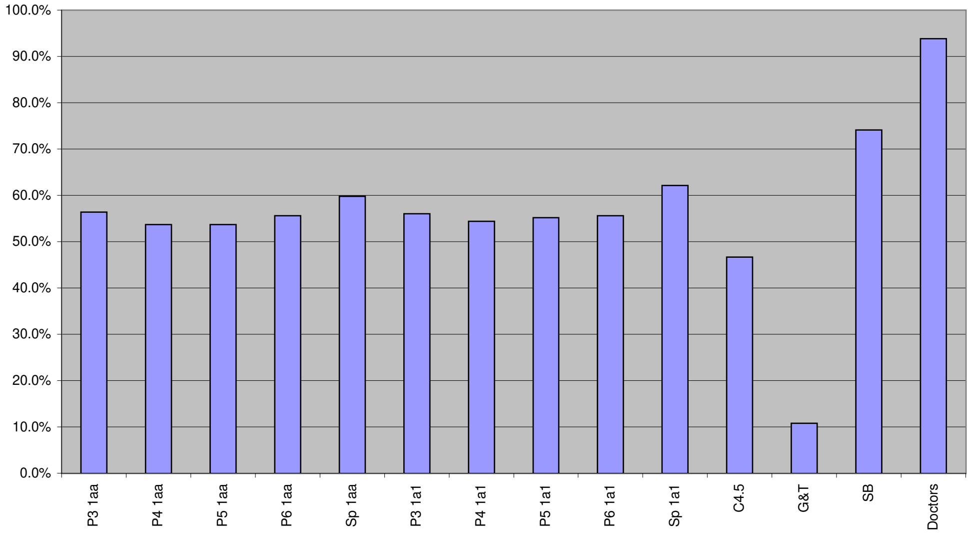

Figure D.1: Success rates on class 1

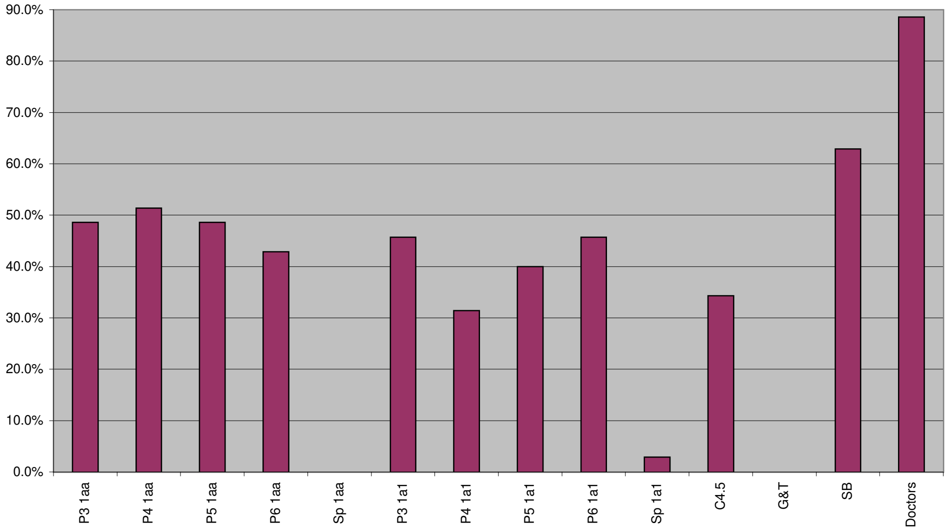

Figure D.2: Success rates on class 2

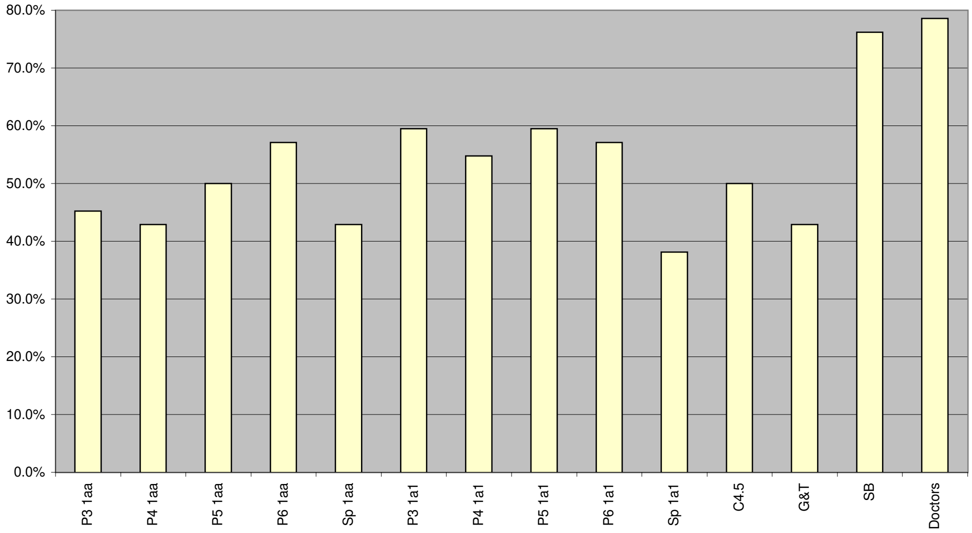

Figure D.3: Success rates on class 3

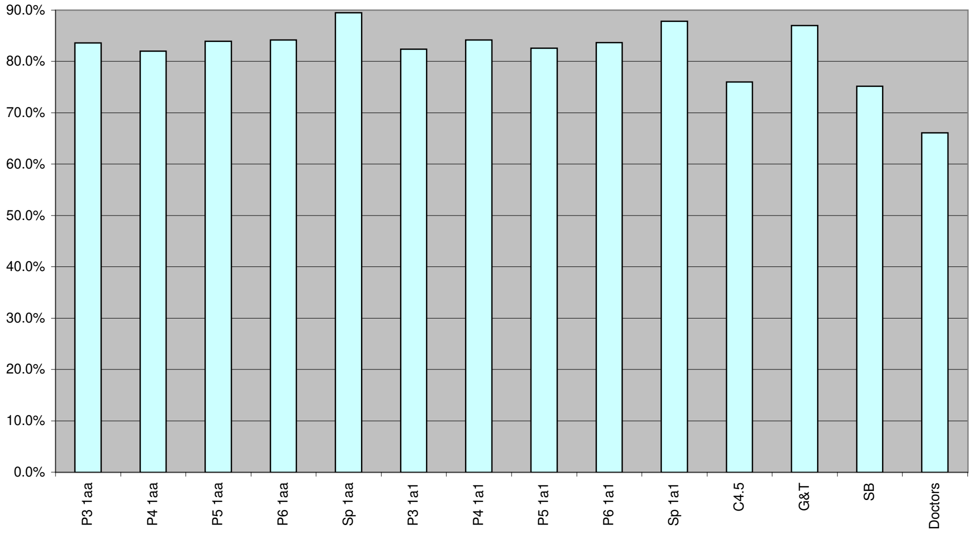

Figure D.4: Success rates on class 4

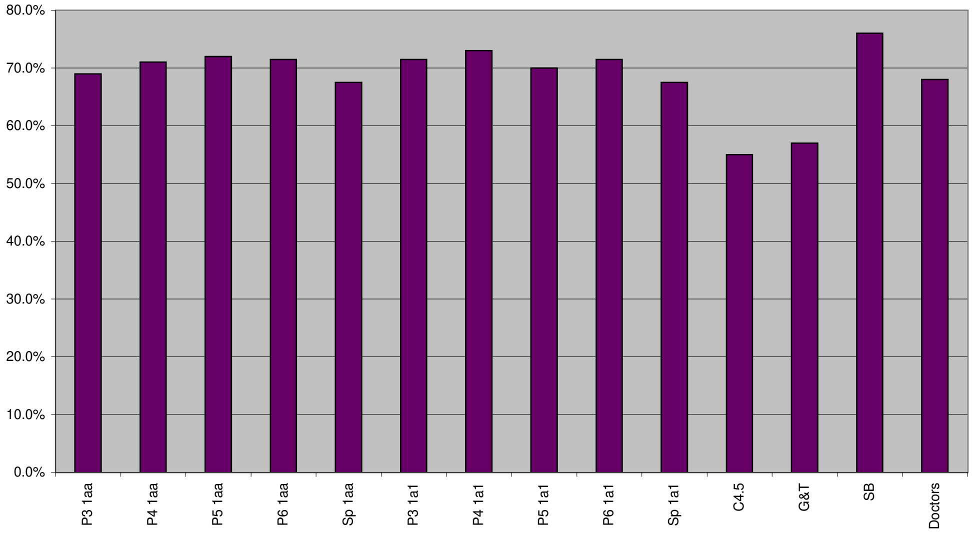

Figure D.5: Success rates on class 5

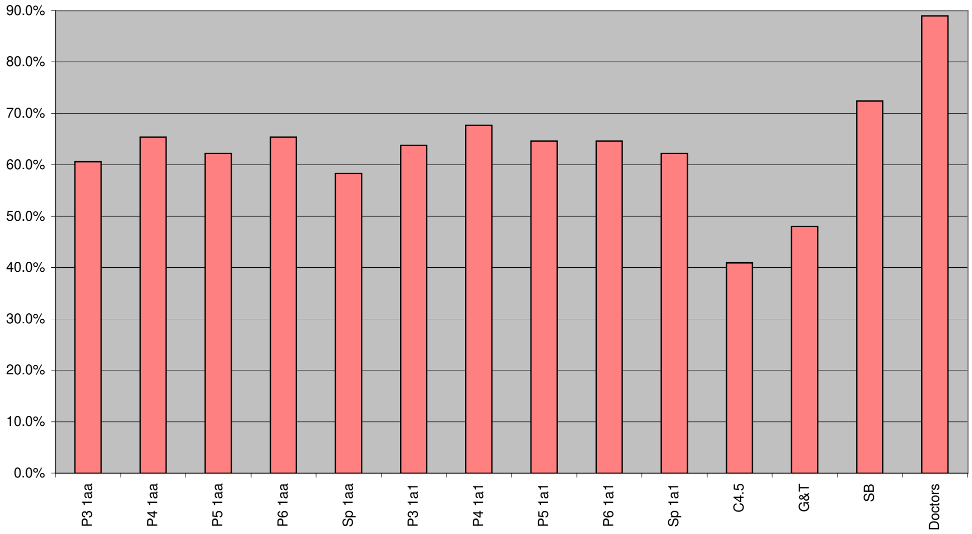

Figure D.6: Success rates on class 6

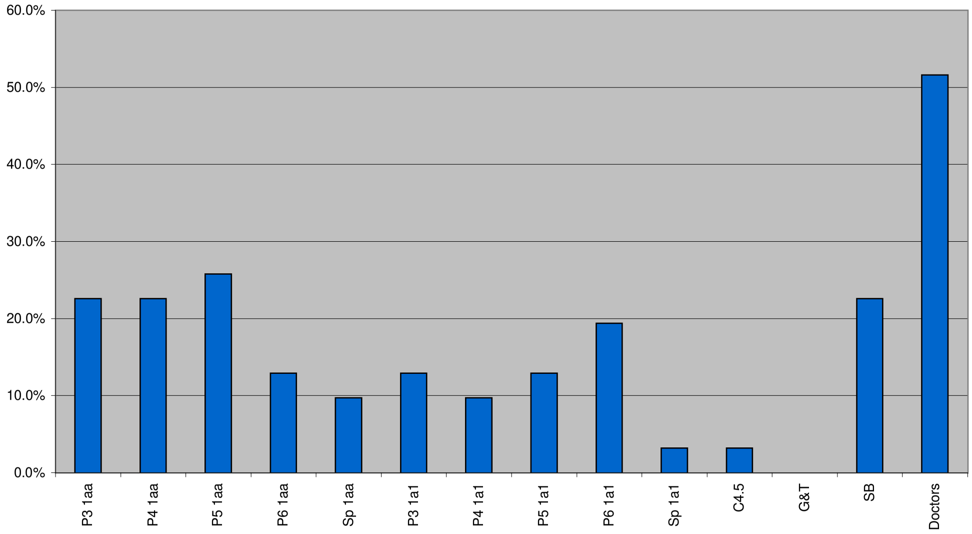

Figure D.7: Success rates on class 7

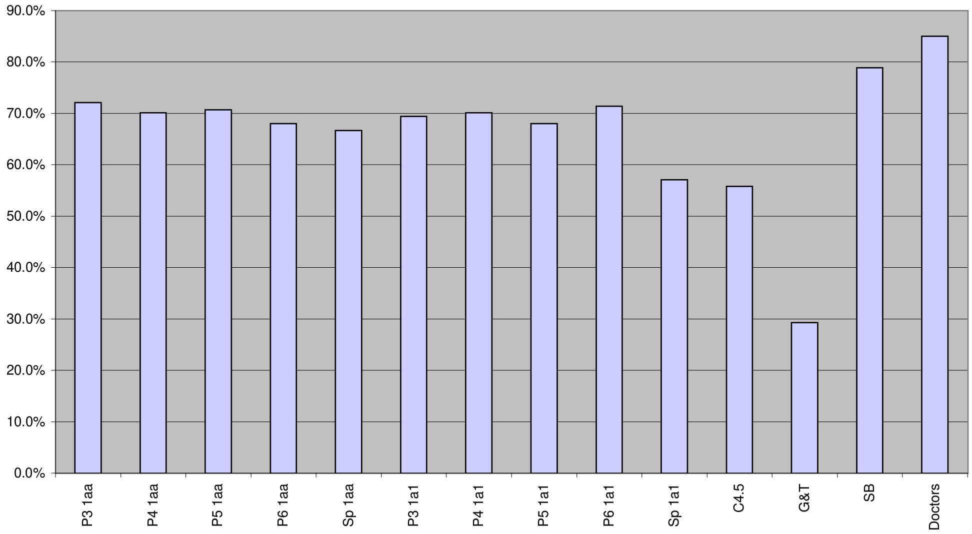

Figure D.8: Success rates on class 8

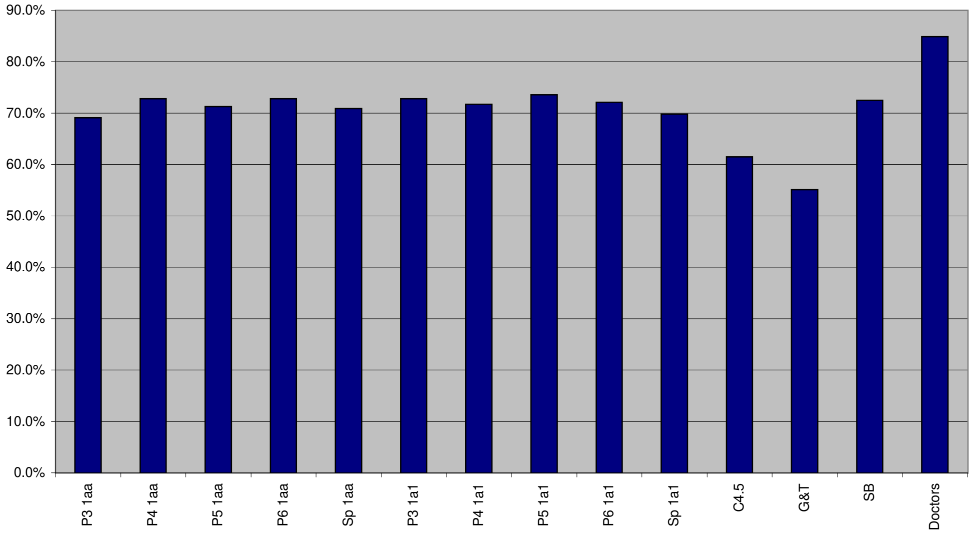

Figure D.9: Success rates on class 9

D.2 Original Dataset

The results here are the ones quoted in section 7.2.6.

D.2.1 1-a-a - Poly 3

Class	1	2	3	4	5	6	7	8	9	Total
1	146	0	6	71	3	6	0	5	2	239
2	2	17	0	6	0	6	0	5	3	39
3	3	2	19	2	2	1	1	0	3	33
4	95	12	5	747	18	25	4	26	28	960
5	3	0	2	13	138	3	6	2	26	193
6	4	3	2	18	10	77	2	0	10	126
7	1	0	2	0	3	1	7	0	8	22
8	0	0	1	12	4	2	0	106	2	127
9	5	1	5	25	22	6	11	3	183	261
Total	259	35	42	894	200	127	31	147	265	2000

Table D.1: 2000 points - 1-a-a - Poly 3

D.2.2 1-a-a - Poly 4

Class	1	2	3	4	5	6	7	8	9	Total
1	139	0	6	70	4	5	0	5	3	232
2	3	18	0	7	1	6	0	5	2	42
3	2	2	18	1	1	1	2	0	4	31
4	103	11	6	733	15	20	1	26	29	944
5	2	0	2	15	142	3	5	4	19	192
6	4	3	1	23	8	83	1	0	7	130
7	1	0	0	2	3	0	7	0	6	19
8	0	0	1	19	2	2	1	103	2	130
9	5	1	8	24	24	7	14	4	193	280
Total	259	35	42	894	200	127	31	147	265	2000

Table D.2: 2000 points - 1-a-a - Poly 4

D.2.3 1-a-a - Poly 5

Class	1	2	3	4	5	6	7	8	9	Total
1	139	0	7	61	3	5	0	5	2	222
2	3	17	0	8	0	3	0	5	2	38
3	3	2	21	0	2	1	1	0	3	33
4	101	12	4	750	16	29	1	25	32	970
5	1	0	2	14	144	3	6	4	23	197
6	4	3	1	19	9	79	1	0	8	124
7	1	0	0	1	2	0	8	0	4	16
8	0	0	1	14	3	2	1	104	2	127
9	7	1	6	27	21	5	13	4	189	273
Total	259	35	42	894	200	127	31	147	265	2000

Table D.3: 2000 points - 1-a-a - Poly 5

D.2.4 1-a-a - Poly 6

Class	1	2	3	4	5	6	7	8	9	Total
1	144	0	3	65	2	4	0	3	1	222
2	1	15	0	7	1	5	0	1	1	31
3	2	2	24	0	1	1	1	0	3	34
4	101	12	4	753	11	25	3	36	34	979
5	1	0	2	12	143	2	6	3	21	190
6	2	4	3	21	10	83	1	1	8	133
7	1	0	0	0	3	0	4	0	3	11
8	0	0	1	14	1	0	0	100	1	117
9	7	2	5	22	28	7	16	3	193	283
Total	259	35	42	894	200	127	31	147	265	2000

Table D.4: 2000 points - 1-a-a - Poly 6

D.2.5 1-a-a - Spline

Class	1	2	3	4	5	6	7	8	9	Total
1	155	0	8	47	9	8	2	5	3	237
2	0	0	0	0	0	0	0	0	0	0
3	3	2	18	0	1	1	2	0	2	29
4	91	27	6	800	17	35	7	39	46	1068
5	0	0	2	11	135	1	4	2	21	176
6	3	4	2	10	9	74	0	0	4	106
7	0	0	0	0	1	0	3	0	0	4
8	0	0	1	7	1	1	0	98	1	109
9	7	2	5	19	27	7	13	3	188	271
Total	259	35	42	894	200	127	31	147	265	2000

Table D.5: 2000 points - 1-a-a - Spline

D.2.6 1-a-1 - Poly 3

Class	1	2	3	4	5	6	7	8	9	Total
1	145	0	3	78	1	3	0	4	1	235
2	2	16	0	9	1	5	0	1	1	35
3	3	2	25	0	1	1	1	0	4	37
4	99	12	4	737	12	27	2	34	33	960
5	1	0	2	13	143	2	7	2	22	192
6	2	3	2	19	9	81	2	1	7	126
7	0	0	0	0	2	0	4	0	3	9
8	0	0	1	15	1	0	0	102	1	120
9	7	2	5	23	30	8	15	3	193	286
Total	259	35	42	894	200	127	31	147	265	2000

Table D.6: 2000 points - 1-a-1 - Poly 3

D.2.7 1-a-1 - Poly 4

Class	1	2	3	4	5	6	7	8	9	Total
1	141	0	2	76	1	2	0	3	1	226
2	1	11	0	5	0	3	0	1	1	22
3	3	2	23	0	1	1	1	0	2	33
4	104	15	7	753	18	28	4	33	37	999
5	0	0	2	13	146	1	5	3	22	192
6	2	4	2	17	10	86	2	1	8	132
7	0	0	0	0	1	0	3	0	3	7
8	0	0	1	11	1	0	0	103	1	117
9	8	3	5	19	22	6	16	3	190	272
Total	259	35	42	894	200	127	31	147	265	2000

Table D.7: 2000 points - 1-a-1 - Poly 4

D.2.8 1-a-1 - Poly 5

Class	1	2	3	4	5	6	7	8	9	Total
1	143	0	3	75	2	4	0	3	1	231
2	2	14	0	10	1	6	0	1	1	35
3	2	2	25	0	1	1	2	0	4	37
4	100	15	3	738	13	25	3	35	32	964
5	1	0	2	13	140	2	6	3	21	188
6	3	3	3	21	10	82	1	1	7	131
7	1	0	0	0	3	0	4	0	3	11
8	0	0	1	14	1	0	0	100	1	117
9	7	1	5	23	29	7	15	4	195	286
Total	259	35	42	894	200	127	31	147	265	2000

Table D.8: 2000 points - 1-a-1 - Poly 5

D.2.9 1-a-1 - Poly 6

Class	1	2	3	4	5	6	7	8	9	Total
1	144	0	4	63	2	6	0	5	3	227
2	2	16	0	7	0	4	0	4	3	36
3	3	2	24	0	2	1	1	0	4	37
4	98	12	3	748	16	26	1	26	30	960
5	1	0	2	13	143	1	6	4	21	191
6	3	4	1	19	8	82	1	0	6	124
7	1	0	0	2	3	0	6	0	4	16
8	0	0	1	16	2	2	1	105	3	130
9	7	1	7	26	24	5	15	3	191	279
Total	259	35	42	894	200	127	31	147	265	2000

Table D.9: 2000 points - 1-a-1 - Poly 6

D.2.10 1-a-1 - Spline

Class	1	2	3	4	5	6	7	8	9	Total
1	161	3	10	62	10	5	4	5	8	268
2	0	1	0	0	0	0	0	0	0	1
3	2	1	16	0	1	0	1	0	1	22
4	87	26	7	785	21	34	4	53	46	1063
5	0	0	2	10	135	1	4	2	20	174
6	2	3	3	15	9	79	2	0	3	116
7	0	0	0	0	0	0	1	0	1	2
8	0	0	0	7	0	0	0	84	1	92
9	7	1	4	15	24	8	15	3	185	262
Total	259	35	42	894	200	127	31	147	265	2000

Table D.10: 2000 points - 1-a-1 - Spline

D.2.11 1-a-a - Mix 1

Class	1	2	3	4	5	6	7	8	9	Total
1	157	0	7	49	6	6	2	5	3	0
2	6	20	1	7	1	9	1	6	2	0
3	3	2	19	0	1	0	2	0	3	0
4	84	9	5	782	12	23	3	36	44	0
5	0	0	2	11	136	1	4	3	24	0
6	2	2	2	14	9	74	1	0	5	0
7	0	0	0	2	8	4	6	1	4	0
8	0	0	1	8	1	1	1	93	1	0
9	7	2	5	21	26	9	11	3	179	0
Total	259	35	42	894	200	127	31	147	265	2000

Table D.11: 2000 points - Mix 1

The number of errors is 534 corresponds to an error rate of 26.7%.

D.2.12 1-a-a - Mix 2

Class	1	2	3	4	5	6	7	8	9	Total
1	151	0	3	48	3	1	0	4	3	0
2	4	19	0	6	1	8	0	4	2	0
3	3	2	20	0	1	1	2	1	4	0
4	84	10	3	744	8	19	1	28	26	0
5	2	0	4	14	134	3	5	5	22	0
6	7	2	2	19	9	78	1	0	11	0
7	1	0	0	2	7	2	5	0	5	0
8	0	0	1	22	2	1	1	102	2	0
9	7	2	9	39	35	14	16	3	190	0
Total	259	35	42	894	200	127	31	147	265	2000

Table D.12: 2000 points - Mix 2

The number of errors is 557 corresponds to an error rate of 27.9%.

D.3 Stripped down Datasets

The following tables contain the detailed results on the first stripped down version of the abdominal pain dataset. For the second stripped down version we have only given the end result.

Each Support Vector machine experiment is presented with the parameters used.

A summary of all results is given in table D.13.

Method	Dataset					
	Original		Strip 1		Strip 2	
	1-a-a	1-a-1	1-a-a	1-a-1	1-a-a	1-a-1
SV Poly 3	28.0%	27.7%	22.6%	21.3%	22.6%	21.5%
SV Poly 4	28.2%	27.2%	22.6%	*20.8%*	22.4%	21.5%
SV Poly 5	27.5%	28.0%	21.8%	21.4%	21.8%	21.7%
SV Poly 6	27.1%	27.1%	21.3%	20.9%	21.5%	21.3%
SV Spline	26.5%	27.7%	21.5%	24.8%	*20.9%*	24.5%
SV Special 2				23.1%		
SV Special 3				21.6%		

Table D.13: Overview of error rates on Abdominal Pain

D.4 Test results Strip 1

SV Vapnik Poly 3

Multi-class SVM 1 against all others			
Kernel:	Vapnik Polynomial	Parameters:	Degree 3
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.14: 1106 points - 1-a-a - Poly 3 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	227	3	6	3	9	0	6	4	258
2	3	23	0	2	8	0	5	2	43
3	6	2	21	3	2	2	0	4	40
5	1	1	4	150	4	5	4	25	194
6	9	3	2	9	94	1	3	13	134
7	1	0	0	4	1	8	0	6	20
8	0	0	1	2	2	1	124	2	132
9	12	3	8	27	7	14	5	209	285
Total	259	35	42	200	127	31	147	265	1106

Table D.15: 1106 points - 1-a-a - Poly 3

The total number of errors was 250.

SV Vapnik Poly 4

Multi-class SVM 1 against all others			
Kernel:	Vapnik Polynomial	Parameters:	Degree 4
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.16: 1106 points - 1-a-a - Poly 4 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	232	2	5	3	10	0	5	6	263
2	1	21	0	2	7	0	5	4	40
3	6	2	23	3	2	4	0	4	44
5	0	0	1	144	2	4	3	25	179
6	4	7	2	10	92	1	1	6	123
7	0	0	0	3	0	3	0	1	7
8	1	0	1	4	4	1	127	5	143
9	15	3	10	31	10	18	6	214	307

continued on next page

continued from previous page									
Class	1	2	3	5	6	7	8	9	Total
Total	259	35	42	200	127	31	147	265	1106

Table D.17: 1106 points - 1-a-a - Poly 4

The total number of errors was 250.

SV Vapnik Poly 5

Multi-class SVM 1 against all others			
Kernel:	Full Polynomial	Parameters:	Deg. 5, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.18: 1106 points - 1-a-a - Poly 5 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	229	2	4	4	9	0	6	4	258
2	2	23	0	2	7	0	5	2	41
3	5	3	23	2	2	2	0	5	42
5	1	0	4	151	3	5	4	24	192
6	9	4	2	9	94	1	3	10	132
7	1	0	0	4	2	8	0	5	20
8	1	0	1	2	2	1	125	3	135
9	11	3	8	26	8	14	4	212	286
Total	259	35	42	200	127	31	147	265	1106

Table D.19: 1106 points - 1-a-a - Poly 5

The total number of errors was 241.

SV Vapnik Poly 6

Multi-class SVM 1 against 1			
Kernel:	Full Polynomial	Parameters:	Deg. 6, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.20: 1106 points - 1-a-a - Poly 6 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	229	3	5	3	9	0	6	3	258
2	2	25	0	2	5	0	5	2	41
3	6	2	26	2	1	3	0	5	45
5	1	0	3	150	3	5	4	24	190
6	9	3	1	10	96	1	3	11	134
7	0	0	1	4	2	8	0	6	21
8	0	0	1	2	2	1	125	3	134
9	12	2	5	27	9	13	4	211	283
Total	259	35	42	200	127	31	147	265	1106

Table D.21: 1106 points - 1-a-a - Poly 6

The total number of errors was 236.

SV Spline

Multi-class SVM 1 against all others			
Kernel:	Splines	Parameters:	-
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	global	Range:	0.0 to 0.5

Table D.22: 1106 points - 1-a-a - Spline - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	233	5	4	4	9	2	9	7	273
2	1	16	0	1	2	0	3	2	25
3	3	1	26	2	0	1	0	3	36
5	0	0	3	149	2	5	2	27	188
6	9	9	4	11	98	1	2	6	140
7	1	0	0	2	0	4	0	0	7
8	0	0	1	1	2	1	124	2	131
9	12	4	4	30	14	17	7	218	306
Total	259	35	42	200	127	31	147	265	1106

Table D.23: 1106 points - 1-a-a - Spline

The total number of errors was 238.

SV Vapnik Poly 3

Multi-class SVM 1-against-1			
Kernel:	Vapnik Polynomial	Parameters:	Degree 3
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.24: 1106 points - 1-a-1 - Poly 3 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	230	3	5	3	7	0	5	2	255
2	3	22	0	2	9	0	3	3	42
3	3	2	25	1	2	1	0	3	37
5	1	0	3	151	3	7	4	23	192
6	7	5	3	10	95	2	4	11	137
7	0	0	0	2	0	4	0	3	9
8	3	0	1	2	0	2	126	3	137
9	12	3	5	29	11	15	5	217	297

continued on next page

continued from previous page									
Class	1	2	3	5	6	7	8	9	Total
Total	259	35	42	200	127	31	147	265	1106

Table D.25: 1106 points - 1-a-1 - Poly 3

The total number of errors was 236.

SV Vapnik Poly 4

Multi-class SVM 1 against 1			
Kernel:	Vapnik Polynomial	Parameters:	Degree 4
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.26: 1106 points - 1-a-1 - Poly 4 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	232	3	4	3	6	0	5	1	254
2	2	20	0	2	6	0	3	3	36
3	3	2	26	1	1	1	0	3	37
5	1	0	4	148	2	7	4	23	189
6	8	7	2	10	101	3	4	10	145
7	0	0	0	1	0	3	0	3	7
8	2	0	1	2	0	2	126	2	135
9	11	3	5	33	11	15	5	220	303
Total	259	35	42	200	127	31	147	265	1106

Table D.27: 1106 points - 1-a-1 - Poly 4

The total number of errors was 230.

SV Vapnik Poly 5

Multi-class SVM 1 against 1			
Kernel:	Full Polynomial	Parameters:	Deg. 5, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.28: 1106 points - 1-a-1 - Poly 5 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	232	3	5	3	7	0	5	1	256
2	3	22	0	2	9	0	4	3	43
3	2	2	25	1	3	2	0	5	40
5	1	0	3	150	3	6	3	24	190
6	7	5	3	11	96	3	4	12	141
7	1	0	0	2	0	4	0	3	10
8	2	0	1	2	0	2	125	2	134
9	11	3	5	29	9	14	6	215	292
Total	259	35	42	200	127	31	147	265	1106

Table D.29: 1106 points - 1-a-1 - Poly 5

The total number of errors was 237.

SV Vapnik Poly 6

Multi-class SVM 1 against 1			
Kernel:	Full Polynomial	Parameters:	Deg. 6, Scale Factor 135, Threshold 1
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.30: 1106 points - 1-a-1 - Poly 6 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	232	2	5	3	6	0	5	2	255
2	2	22	0	2	8	0	4	3	41
3	2	2	25	1	2	1	0	4	37
5	1	0	3	152	3	5	3	26	193
6	6	6	3	11	99	2	4	10	141
7	1	0	0	2	0	4	0	3	10
8	3	0	1	2	0	2	126	2	136
9	12	3	5	27	9	17	5	215	293
Total	259	35	42	200	127	31	147	265	1106

Table D.31: 1106 points - 1-a-1 - Poly 6

The total number of errors was 231.

SV Spline

Multi-class SVM 1 against 1			
Kernel:	Splines	Parameters:	-
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	global	Range:	0.0 to 0.5

Table D.32: 1106 points - 1-a-1 - Spline - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	230	6	10	13	16	3	11	12	301
2	0	12	0	0	2	0	3	0	17
3	2	1	16	1	0	1	0	1	22
5	0	0	2	143	2	4	5	22	178
6	9	11	5	11	94	1	3	7	141
7	0	0	0	0	0	1	0	1	2
8	3	0	1	1	2	1	116	2	126
9	15	5	8	31	11	20	9	220	319

continued on next page

continued from previous page									
Class	1	2	3	5	6	7	8	9	Total
Total	259	35	42	200	127	31	147	265	1106

Table D.33: 1106 points - 1-a-1 - Spline

The total number of errors was 274.

SV Special Kernel Deg 2

This is using the special kernel from 6.3.4. Basically this is Vapnik's polynomial kernel of degree two, where no two values of the same attribute are ever multiplied together. "Same" here means of the same original attribute out of 33.

Multi-class SVM 1 against 1			
Kernel:	see text	Parameters:	Degree 2
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.34: 1106 points - 1-a-1 - Special 2 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	229	2	4	3	6	0	6	3	253
2	3	23	0	2	9	0	5	3	45
3	2	2	24	2	3	2	0	5	40
5	1	0	3	147	3	5	4	28	191
6	10	5	4	10	95	3	4	12	143
7	1	0	0	3	1	4	0	5	14
8	2	0	1	2	1	2	122	2	132
9	11	3	6	31	9	15	6	207	288
Total	259	35	42	200	127	31	147	265	1106

Table D.35: 1106 points - 1-a-1 - Special 2

The total number of errors was 255.

SV Special Kernel Deg 3

Same as previous kernel but of degree 3.

Multi-class SVM 1-against-1			
Kernel:	see text	Parameters:	Degree 3
Alpha bound C:	∞		
Chunking:	new style	Sizes:	500/200
Scaling:	-	Range:	-

Table D.36: 1106 points - 1-a-1 - Special 3 - Parameters

Class	1	2	3	5	6	7	8	9	Total
1	230	3	4	3	6	0	5	2	253
2	2	22	0	2	8	0	4	3	41
3	2	2	26	1	1	1	0	4	37
5	1	0	4	150	3	7	4	28	197
6	10	5	2	10	99	3	4	12	145
7	1	0	0	2	0	4	0	3	10
8	2	0	1	2	0	2	125	2	134
9	11	3	5	30	10	14	5	211	289
Total	259	35	42	200	127	31	147	265	1106

Table D.37: 1106 points - 1-a-1 - Special 3

The total number of errors was 239.

Appendix E

Details of the Abdominal Pain Database

This appendix give a detailed overview of the symptoms and diseases occurring in the abdominal pain database described in chapter 7.2.6.

E.1 Symptoms

For every one of the 33 symptoms there is a list of possible attributes and the number of the corresponding binary attribute, cf. section 6.3.

In the following table the attribute value "missing" refers to the number of cases for which this attribute is missing; this is not an attribute value that appears in the data. Also note that as multiple attribute values can be present for the same symptom for a single case.

No.	Symptom	Value		Cases	
		Name	No.	Train Set	Test Set
1	Sex	male	1	2137	989
		female	2	2240	1009
		missing		11	2
2	Age	0-9	3	194	77
		10-19	4	1119	553
		20-29	5	870	389

continued on next page

continued from previous page

No.	Symptom	Value		Cases	
		Name	No.	Train Set	Test Set
		30-39	6	626	254
		40-49	7	450	216
		50-59	8	362	163
		60-69	9	313	143
		70+	10	453	208
		missing		16	4
3	Pain-site onset	right upper quadrant	11	294	152
		left upper quadrant	12	112	35
		right lower quadrant	13	690	336
		left lower quadrant	14	261	119
		upper half	15	388	155
		lower half	16	542	240
		right half	17	189	106
		left half	18	147	65
		central	19	971	466
		general	20	163	52
		right loin	21	183	82
		left loin	22	163	54
		epigastric	23	666	303
		missing		55	28
4	Pain-site present	right upper quadrant	24	294	164
		left upper quadrant	25	89	29
		right lower quadrant	26	1232	533
		left lower quadrant	27	260	114
		upper half	28	334	137
		lower half	29	498	227
		right half	30	257	126
		left half	31	137	66
		central	32	534	264
		general	33	190	70
		right loin	34	213	85
		left loin	35	173	68
		epigastric	36	554	249
		pain settled	37	273	141
		missing		61	31
5	Aggravating factors	movement	38	1593	702
		coughing	39	838	381
		inspiration	40	457	230
		food	41	279	124

continued on next page

continued from previous page

No.	Symptom	Value Name	Value No.	Cases Train Set	Cases Test Set
		other	42	374	176
		nil	43	1915	860
		missing		35	20
6	Relieving factors	lying still	44	1116	486
		vomiting	45	295	142
		antacids	46	105	50
		milk or food	47	81	47
		other	48	518	252
		nil	49	2421	1086
		missing		48	26
7	Progress of pain	getting better	50	996	457
		no change	51	1473	674
		getting worse	52	1810	805
		missing		109	65
8	Duration of pain	under 12 hours	53	1680	764
		12-24 hours	54	729	321
		24-48 hours	55	494	236
		over 48 hours	56	1373	637
		missing		111	42
9	Type of pain	steady	57	2086	1018
		intermittent	58	1243	528
		colicky	59	1425	616
		sharp	60	1093	432
		missing		73	25
10	Severity of pain	moderate	61	1923	939
		severe	62	2401	1015
		missing		67	47
11	Nausea	present	63	3124	1420
		not present	64	1243	567
		missing		21	13
12	Vomiting	present	65	2392	1121
		not present	66	1976	872
		missing		20	8
13	Anorexia	present	67	2740	1212
		normal appetite	68	1609	764
		missing		39	25
14	Indigestion	history of dyspepsia	69	1047	468
		no history of dyspepsia	70	3251	1503
		missing		89	30

continued on next page

continued from previous page					
No.	Symptom	Value		Cases	
		Name	No.	Train Set	Test Set
15	Jaundice	history of jaundice	71	127	64
		no history of jaundice	72	4165	1896
		missing		95	40
16	Bowel habit	no change	73	2887	1315
		constipated	74	843	367
		diarrhoea	75	623	299
		blood	76	69	30
		mucus	77	30	23
		missing		41	19
17	Micturition	normal	78	3466	1562
		frequent	79	551	249
		dysuria	80	311	137
		haematuria	81	67	28
		dark urine	82	138	89
		missing		72	39
18	Previous pain	similar pain before	83	2037	945
		no similar pain before	84	2318	1039
		missing		33	16
19	Previous surgery	yes	85	1160	538
		no	86	3073	1401
		missing		156	62
20	Drugs	being taken	87	1764	843
		not being taken	88	2360	1056
		missing		263	101
21	Mood	normal	89	1840	865
		distressed	90	648	291
		anxious	91	934	364
		missing		1017	498
22	Colour	normal	92	2500	1163
		pale	93	876	389
		flushed	94	744	320
		jaundiced	95	65	37
		cyanosed	96	19	5
		missing		194	89
23	Abdominal movements	normal	97	3499	1582
		poor or nil	98	363	140
		visible peristalsis	99	22	14
		missing		504	264
24	Abdominal scar	present	100	1016	472
					continued on next page

continued from previous page

No.	Symptom	Value Name	Value No.	Cases Train Set	Cases Test Set
		absent	101	2526	1233
		missing		846	295
25	Abdominal distension	present	102	367	154
		absent	103	3920	1794
		missing		100	52
26	Site of tenderness	right upper quadrant	104	389	224
		left upper quadrant	105	96	38
		right lower quadrant	106	1407	615
		left lower quadrant	107	383	156
		upper half	108	254	116
		lower half	109	384	181
		right half	110	335	164
		left half	111	177	83
		central	112	248	110
		general	113	252	109
		right loin	114	170	83
		left loin	115	136	59
		epigastric	116	582	255
		none	117	499	235
		missing		25	10
27	Rebound	present	118	948	416
		absent	119	3337	1535
		missing		105	49
28	Guarding	present	120	1317	523
		absent	121	3039	1457
		missing		34	20
29	Rigidity	present	122	197	87
		absent	123	4147	1897
		missing		43	17
30	Abdominal masses	present	124	287	130
		absent	125	4016	1834
		missing		85	36
31	Murphy's Test	positive	126	284	173
		negative	127	3932	1739
		missing		172	88
32	Bowel sounds	normal	128	3468	1579
		decreased or absent	129	354	153
		increased	130	502	238
		missing		64	30

continued on next page

continued from previous page					
No.	Symptom	Value		Cases	
		Name	No.	Train Set	Test Set
33	Rectal examination	tender left side	131	90	41
		tender right side	132	459	192
		generally tender	133	351	158
		mass felt	134	60	26
		normal	135	2985	1408
		missing		480	191

Table E.1: Symptoms

Table E.2 lists whether the attribute must be present because all cases are covered by the attributes, but may have multiple attribute values. Table E.4 lists the attributes which are not required, but may take at most one value and table E.3 lists the attributes which have to have exactly one attribute value.

No.	Symptom	Examples violating	
		Train Set	Test Set
5	Aggravating factors	35	20
6	Relieving factors	48	26

Table E.2: Required attributes - multiple values

In table E.3 there are two kinds of error. The first number in each addition refers to examples with multiple values and the second number refers to examples with missing values.

No.	Symptom	Examples violating	
		Train Set	Test Set
1	Sex	1+11	0+2
2	Age	16+16	7+4
7	Progress of pain	1+109	1+65
11	Nausea	1+21	0+13
12	Vomiting	1+20	1+8
14	Indigestion	0+89	1+30
15	Jaundice	0+95	0+40

continued on next page

continued from previous page			
No.	Symptom	Examples violating	
		Train Set	Test Set
18	Previous pain	1+33	0+16
19	Previous surgery	2+156	1+62
20	Drugs	0+263	0+101
24	Abdominal scar	1+846	0+295
25	Abdominal distension	0+100	0+52
27	Rebound	3+105	0+49
28	Guarding	3+34	0+20
29	Rigidity	0+43	1+17
30	Abdominal masses	1+85	0+36
32	Bowel sounds	1+64	0+30

Table E.3: Required attributes - single values

No.	Symptom	Examples violating	
		Train Set	Test Set
8	Duration of pain	0	0
13	Anorexia	1	1
23	Abdominal movements	1	0
31	Murphy's test	1	0

Table E.4: Not-required attributes - single values

E.2 Diseases

This section contains on overview of the abdominal diseases in the database. Table E.5 lists how many examples there are and how many have missing values which are required as described in the previous section.

No.	Class	Examples		Examples with missing values	
		Train Set	Test Set	Train Set	Test Set
1	Appendicitis	585	259	306	133
2	Diverticulitis	108	35	54	16
3	Perforated peptic ulcer	88	42	47	19
4	Non-specific abdominal pain	1941	894	912	398
5	Cholisititis	372	200	185	104
6	Intestinal obstruction	290	127	158	70
7	Pancreatitis	65	31	36	20
8	Renal colic	326	147	159	75
9	Dyspepsia	612	265	298	110
	Total	4387	2000	2145	945

Table E.5: Abdominal diseases

E.3 Bigtable

This is the table used by the Simple Bayes algorithm to work out its probabilities on the original dataset. It also shows a more detailed distribution of the attribute values per class.

Binary attribute	Class 1	2	3	4	5	6	7	8	9	Total
1	335	42	53	793	97	132	43	250	392	2137
2	250	66	35	1143	274	158	22	75	217	2240
3	31	0	0	162	0	1	0	0	0	194
4	273	1	6	772	2	7	0	8	50	1119
5	123	2	9	445	57	16	4	49	165	870
6	74	2	11	246	53	28	18	75	119	626
7	29	9	10	144	37	35	12	83	91	450
8	25	17	17	66	58	44	11	59	65	362
9	19	26	20	20	62	56	9	40	61	313
10	10	51	15	84	100	106	10	13	64	453
11	9	0	10	52	166	6	6	5	40	294
12	2	1	5	45	9	5	3	7	35	112

continued on next page

continued from previous page

Binary attribute	Class 1	2	3	4	5	6	7	8	9	Total
13	149	5	4	458	7	25	0	37	5	690
14	8	33	1	165	2	10	0	39	3	261
15	32	5	21	107	54	34	24	4	107	388
16	63	46	6	334	3	59	5	14	12	542
17	19	1	3	96	16	9	1	30	14	189
18	5	8	1	74	0	11	1	42	5	147
19	266	8	8	505	17	97	7	3	60	971
20	34	5	10	71	2	25	1	1	14	163
21	4	0	2	52	11	0	0	109	5	183
22	2	2	0	42	3	2	0	109	3	163
23	28	0	30	78	125	22	20	0	363	666
24	2	0	9	43	177	5	5	4	49	294
25	1	0	2	41	4	4	2	6	29	89
26	436	7	4	680	7	34	2	58	4	1232
27	3	36	0	155	1	10	0	51	4	260
28	13	4	21	70	58	32	23	1	112	334
29	49	38	7	325	3	50	6	10	10	498
30	30	0	5	119	26	15	1	43	18	257
31	3	5	0	69	1	10	1	42	6	137
32	66	5	6	301	9	86	3	9	49	534
33	19	7	24	70	10	40	5	1	14	190
34	7	0	3	71	17	1	0	109	5	213
35	0	2	0	50	3	1	1	113	3	173
36	14	1	19	58	92	15	18	0	337	554
37	2	9	0	170	26	8	3	9	46	273
38	380	34	61	752	102	66	22	44	132	1593
39	218	19	38	357	60	33	17	18	78	838
40	94	5	31	174	67	11	14	15	46	457
41	15	3	2	74	47	13	6	3	116	279
42	42	8	10	165	27	29	10	21	62	374
43	120	54	12	837	169	175	25	247	276	1915
44	249	18	21	584	69	36	11	26	102	1116
45	33	1	3	95	28	45	2	11	77	295
46	1	0	2	11	15	5	1	0	70	105
47	2	1	5	8	5	4	0	0	56	81
48	54	20	9	216	48	30	7	65	69	518
49	263	66	50	1066	221	177	44	231	303	2421
50	69	24	8	548	81	35	7	81	143	996
51	171	39	34	655	127	103	20	100	224	1473

continued on next page

continued from previous page

Binary attribute	Class 1	2	3	4	5	6	7	8	9	Total
52	339	42	44	682	154	147	38	138	226	1810
53	182	15	66	732	137	77	28	219	224	1680
54	162	21	4	317	56	53	9	39	68	729
55	124	12	3	213	45	30	4	13	50	494
56	115	52	14	632	125	115	22	51	247	1373
57	346	46	63	786	209	104	47	177	308	2086
58	103	31	9	646	84	84	9	71	206	1243
59	178	42	24	638	99	175	12	133	124	1425
60	148	13	33	485	87	45	21	90	171	1093
61	245	51	10	1089	102	111	7	36	272	1923
62	333	55	77	816	266	176	57	289	332	2401
63	474	68	64	1208	305	239	50	253	463	3124
64	108	39	24	724	67	48	13	72	148	1243
65	401	49	41	802	257	220	47	214	361	2392
66	182	58	47	1131	113	68	17	111	249	1976
67	477	61	56	1050	276	217	49	161	393	2740
68	105	47	30	878	93	68	15	158	215	1609
69	90	21	45	251	158	68	18	49	347	1047
70	482	85	43	1650	206	210	44	273	258	3251
71	8	2	2	26	58	9	5	4	13	127
72	562	102	85	1881	308	271	58	312	586	4165
73	411	33	54	1314	259	106	34	269	407	2887
74	86	47	27	307	74	142	15	38	107	843
75	87	23	7	311	35	41	14	21	84	623
76	0	15	5	21	2	6	1	1	18	69
77	0	7	0	11	3	2	2	0	5	30
78	478	75	78	1592	265	232	52	166	528	3466
79	70	21	8	230	43	29	3	94	53	551
80	42	9	0	142	14	20	1	60	23	311
81	1	1	0	6	5	0	1	52	1	67
82	12	1	0	16	62	5	9	20	13	138
83	179	49	23	816	249	119	46	159	397	2037
84	400	58	65	1117	120	166	19	165	208	2318
85	39	47	28	368	146	181	28	102	221	1160
86	534	58	58	1525	210	91	35	209	353	3073
87	136	70	51	581	251	150	44	151	330	1764
88	426	32	32	1251	103	116	13	154	233	2360
89	197	42	13	1023	112	98	15	95	245	1840
90	96	17	39	163	67	68	20	95	83	648

continued on next page

continued from previous page

Binary attribute	Class 1	2	3	4	5	6	7	8	9	Total
91	126	28	16	354	90	57	17	75	171	934
92	219	60	29	1248	178	143	24	194	405	2500
93	114	27	45	248	88	104	24	99	127	876
94	226	15	5	357	39	19	12	24	47	744
95	3	0	1	1	50	6	3	0	1	65
96	1	2	5	1	5	2	1	0	2	19
97	423	85	25	1648	289	194	41	284	510	3499
98	88	10	55	67	30	52	17	10	34	363
99	1	0	0	3	1	16	1	0	0	22
100	33	39	24	333	130	147	26	87	197	1016
101	441	47	45	1237	196	81	28	157	294	2526
102	24	27	9	85	22	165	8	10	17	367
103	546	78	78	1813	341	122	56	308	578	3920
104	8	0	8	58	236	11	8	5	55	389
105	0	2	2	43	3	5	1	9	31	96
106	484	14	3	784	16	48	2	42	14	1407
107	10	44	0	251	4	23	1	34	16	383
108	2	2	17	54	35	21	27	0	96	254
109	49	20	3	251	2	44	4	5	6	384
110	49	1	7	156	29	22	2	39	30	335
111	4	7	0	95	1	15	3	42	10	177
112	17	4	1	153	6	44	5	2	16	248
113	25	14	46	58	14	62	6	4	23	252
114	9	0	2	71	12	0	0	74	2	170
115	1	1	0	43	2	0	0	85	4	136
116	8	2	15	82	98	15	25	3	334	582
117	2	11	1	277	19	32	1	73	83	499
118	373	25	42	312	57	53	15	13	58	948
119	197	77	42	1599	305	226	47	310	534	3337
120	359	31	73	400	149	74	40	59	132	1317
121	222	74	15	1532	221	213	24	264	474	3039
122	37	3	57	35	18	10	5	8	24	197
123	543	103	29	1890	352	276	58	313	583	4147
124	15	11	1	124	41	63	3	14	15	287
125	558	93	85	1799	322	214	57	307	581	4016
126	12	0	8	25	186	5	7	4	37	284
127	543	99	74	1842	174	270	55	318	557	3932
128	473	76	24	1641	304	110	43	298	499	3468
129	73	15	61	55	33	50	15	16	36	354

continued on next page

continued from previous page										
Binary attribute	Class 1	2	3	4	5	6	7	8	9	Total
130	29	17	1	221	27	127	7	8	65	502
131	20	6	0	52	2	3	1	4	2	90
132	205	1	1	223	6	6	0	8	9	459
133	77	19	9	181	7	16	2	7	33	351
134	5	5	1	28	6	10	1	3	1	60
135	259	70	57	1271	311	201	52	260	504	2985
Total	585	108	88	1941	372	290	65	326	612	4387

Table E.6: Abdominal diseases

Bibliography

[1] I. D. Adams, M. Chan, P. C. Clifford, W. M. Cooke, V. Dallos, F. T. de Dombal, M. H. Edwards, D. M. Hancock, D. J. Hewett, N. McIntyre, P. G. Sommerville, D. J. Spiegelhalter, J. Wellwood, and D. H. Wilson. Computer aided diagnosis of acute abdominal pain: amulticentre study. *British Medical Journal*, 293:800–804, September 1986.

[2] K. P. Bennett and J. A. Blue. A support vector machine approach to decision trees. From K. P. Bennets web page.

[3] L. Breiman. Bagging predictors. Technical Report 421, Department of Statistics, UC Berkeley, 1994.

[4] Chris J. C. Burges and Bernhard Schölkopf. Improving the accuracy and speed of support vector machines. *Neural Information Processing Systems 9*, 1997.

[5] Chris J. C. Burges and Bernhard Schölkopf. Improving the accuracy and speed of support vector machines. In M Mozer, M. Jordan, and T. Petsche, editors, *Neural Information Processing Systems*, volume 9, Cambridge, MA, 1997. MIT Press.

[6] E. Keogh C. Blake and C.J. Merz. UCI - repository of machine learning databases, 1998.

[7] Vladimir Cherkassky and Filip Mulier. *Learning from Data.* Wiley-Interscience, New York, 1998.

[8] Nello Christianini, Colin Campbell, and John Shawe-Taylor. Dynamically adapting kernels in support vector machines. NeuroCOLT Technical Report Series NC2-TR-1998-017, Royal Holloway, University of London, Egham, UK, May 1998.

[9] Corrina Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, June 1995.

[10] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume I. Springer Verlag, Berlin, first English edition, 1953.

[11] F. T. de Dombal, D. J. Leaper, J. R. Staniland, A. P. McCann, and Jane C. Horrocks. Computer-aided diagnosis of acute abdominal pain. *British Medical Journal*, 2:9–13, April 1972.

[12] Thomas Dietterich, Wolfgang Maass, Hans Ulrich Simon, and Richard Sutton, editors. *Theory and Practice of Machine Learning.* Schloss Dagstuhl, Internationales Begegnungs- und Forschungszentrum für Informatik, January 1997. Dagstuhl-Seminar-Report: 163.

[13] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence*, 2:263–286, 1995.

[14] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, Cambridge, MA, 1997. MIT Press.

[15] Laurene Fausett. *Fundamentals of Neural Networks.* Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[16] Jerome H. Friedman. Another approach to polychotomous classification. From author's web page, October 1996.

[17] Stephen Fry. *Making History.* Hutchinson, 1996.

[18] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Massachusetts, 1994.

[19] A. Gammerman and A. R. Thatcher. Bayesian inference in an expert system without assuming independence. Technical Report 88/11, Computer Science Department, Heriot-Watt University, Edinburgh, December 1988.

[20] A. Gammerman, V. Vapnik, and V. Vovk. Transduction in Pattern Recognition. *Submitted for publication to Int. Joint Conference on Artificial Intelligence*, 1997.

[21] Professor Alex Gammerman. Machine Learning: Progress and Prospects. Avaiable from the Department, Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, TW20 0EX, UK, December 1996.

[22] A.J. Gammermann and W.J. Crabbe. Computational models of probabalistic reasoning in expert systems: Cprs a causal probabalistic reasoning system. Technical Report No.87/16, Heriot-Watt University, Department of Computer Science, 1987.

[23] Yiqun Gu. *A Bayesian system for Computer Aided Diagnosis without assuming Conditional Independence*. PhD thesis, Department of Computer Science, Heriott-Watt University, Edinburgh, May 1992.

[24] Richard W. Hamming. *Coding and Information Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[25] W. H. Highleyman. The design and analysis of pattern recognition experiments. *Bell Systems Technical Journal*, 41:723–744, 1962.

[26] W. H. Highleyman. Linear decision functions with applications to pattern recognition. In *Proc. IRE No. 6*, 1962.

[27] E. T. A. Hoffmann. *Dichtungen und Schriften sowie Briefe und Tagebücher - Gesamtausgabe in fünfzehn Bänden*. Erich Lichtenstein, Weimar, 1924.

[28] Jane C. Horrocks, A. P. McCann, J. R. Staniland, D. J. Leaper, and F. T. de Dombal. Computer-aided diagnosis: Description of an adaptable system, and operational experience with 2,034 cases. *British Medical Journal*, 2:5–9, April 1972.

[29] Alan Hutchinson. *Algorithmic Learning*. Clarendon Press, Oxford, 1994.

[30] T. Joachims. *Advances in Kernel Methods - Support Vector Learning*, chapter Making Large-Scale Support Vector Machine Learning Practical, pages 169–184. MIT Press, Cambridge, MA, 1999.

[31] Ulrich Kreßel and Ingo Graf. Paarweise Klassifikation mit Support Vektoren. submitted to DAGM - Mustererkennung 1997, 1997.

[32] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society (B)*, 50:157–224, 1988.

[33] S. B. Lippman. *C++ primer.* Addison-Wesley, Reading, Massachusets, second edition, 1991.

[34] George F. Luger and William A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* Benjamin / Cummings, Redwood City, California, second edition, 1992.

[35] Zhiyuan Luo. *A Probabalistc Reasoning and Learning System Based On Bayesian Belief Networks.* PhD thesis, Heriot-Watt University, Department of Computer Science, May 1992.

[36] Jochen Marly, editor. *ComputerR.* Reihe: "Beck-Texte im dtv". dtv / C. H. Beck, München, May 1994. ISBN 3 423 055626.

[37] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning.* tioga publishing company, Palo Alto, California, 1983.

[38] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, Singapore, international edition, 1997.

[39] Bruce A. Murtagh and Michael A. Saunders. Minos 5.4 User's Guide. Technical Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA, February 1995.

[40] Jonathan J. Oliver and David Hand. Introduction to minimum encoding inference. Technical Report TR 4-94, Dept. of Statistics, Open University, Walton Hall, UK, 11 1994. also TR 205 from Dept. of Computer Science, Monash University, Australia.

[41] Edgar Osuna, Robert Freund, and Frederico Girosi. An improved training algorithm for support vector machines. In *Proceedings of IEEE NNSP'97*, Amelia Island, FL, September 1997.

[42] Edgar Osuna, Robert Freund, and Frederico Girosi. Support vector machines: Training and applications. Technical Report AI Memo No. 1602, CBCL Paper No. 144, MIT Artificial Intelligence Laboratory, March 1997.

[43] J. Pearl and J. H. Kim. A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of IJCAI-83*, 1983.

[44] Judea Pearl. *Probabalistic Reasoning in intelligent systems: Networks of plausible inference.* Morgan Kaufmann, San Mateo, 1988.

[45] John C. Platt. Using analytical qp and sparseness to speed training of support vector machines. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1999. MIT Press.

[46] Karl R. Popper. *The Logic of Scientific Discovery.* Hutchinson, London, 1959.

[47] Lutz Prechtel. Proben 1 - a set of neural network benchmark problems and benchmarking results. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, September 1994.

[48] J. Ross Quinlan. *C4.5 - Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[49] Elaine Rich and Kevin Knight. *Artificial Intelligence.* McGraw-Hill, Singapore, 2^{nd} international edition, 1991.

[50] C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proceedings of the 15th International Conference on Machine Learning (ICML '98)*, 1998.

[51] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola. Support vector machine reference manual. Technical Report CSD-TR-98-03, Royal Holloway, University of London, Egham, UK, 1998.

[52] B. Schölkopf. *Support Vector Learning.* R. Oldenbourg Verlag, Munich, 1997.

[53] Bernhard Schölkopf, Chris Burges, and Vladimir Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *First International Conference on Knowledge Discovery & Data Mining*, pages 252–257, Menlo Park, CA, 1995. AAAI Press.

[54] Bernhard Schölkopf, Patrice Simard, Alex Smola, and Vladimir Vapnik. Prior knowledge in support vector kernels. From authors' web page.

[55] John Self. *Artificial Intelligence and Human Learning*. Chapman and Hall, London, 1988.

[56] L. Shepp, Y. Vardi, and L. Kaufman. A statistical model for Positron Emission Tomography. *J. Amer. Stat. Assoc.*, 80(389):8–37, 1985.

[57] P. Y. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems 5*, pages 50–58, Cambridge, MA, 1993. MIT Press.

[58] J. A. Simpson and E. S. C. Weiner, editors. *Oxford English Dictionary*, volume VIII. Clarendon Press, Oxford, second edition, 1989.

[59] J. A. Simpson and E. S. C. Weiner, editors. *Oxford English Dictionary*, volume I. Clarendon Press, Oxford, second edition, 1989.

[60] M. O. Stitson. XPress - Probabalistic Reasoning Exper System Shell under X-Windows. Technical Report CS382, Royal Holloway, University of London, Egham, UK, March 1995.

[61] M. O. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, and J. Weston. Support vector regression with ANOVA decomposition kernels. Technical Report CSD-TR-97-22, Royal Holloway, University of London, 1997.

[62] M. O. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, and J. Weston. *Advances in Kernel Methods - Support Vector Learning*, chapter Support Vector Regression with ANOVA Decomposition Kernels, pages 285–291. MIT Press, Cambridge, MA, 1998.

[63] M. O. Stitson and J. A. E. Weston. Implementational issues of support vector machines. Technical Report CSD-TR-96-18, Royal Holloway, University of London, Egham, UK, December 1996.

[64] M. O. Stitson, J. A. E. Weston, A. Gammerman, V. Vovk, and V. Vapnik. Theory of support vector machines. Technical Report CSD-TR-96-17, Royal Holloway, University of London, Egham, UK, December 1996.

[65] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, third edition, 1997.

[66] Henry Tirri, Petri Kontkanen, and Petri Myllymäki. Probabilistic instance-based learning. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 507–515, San Francisco, CA, 1996. Morgan Kaufmann Publishers.

[67] Multi-Pass Instance-Based Learning Algorithm MPIL, 1997.

[68] Robert J. Vanderbei. Loqo User's Manual - Version 2.21. Technical Report SOR-96-07, Princeton University, Princeton, NJ, June 1996.

[69] V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 281–287, Cambridge, MA, 1997. MIT Press.

[70] Vladimir N. Vapnik. *Estimation of Dependencies Based on Empirical Data.* Springer, New York, 1982.

[71] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[72] Vladimir N. Vapnik. *Statistical Learning Theory.* J. Wiley, New York, 1998.

[73] Vladimir N. Vapnik and A. J. Chervonenkis. *Theorie der Zeichenerkennung.* Akademie-Verlag, Berlin, 1979. Translated from the Russian original (1974).

[74] Grace Wahba. Support vector machines, reproducing kernel hilbert spaces and the randomized gacv. Technical Report 984, Department of Statistics, University of Wisconsin, Madison, WI, November 1997.

[75] Ing. Rudolf Walther, editor. *Wörterbuch der Technik Englisch-Deutsch.* Cornelsen Verlag Schwann-Giradet, Düsseldorf, fifth edition, 1986.

[76] Ing. Rudolf Walther, editor. *Wörterbuch der Technik Deutsch-Englisch.* Cornelsen Verlag Schwann-Giradet, Düsseldorf, fourth edition, 1988.

[77] J. Weston, A. Gammerman, M. O. Stitson, V. Vapnik, V. Vovk, and C. Watkins. Density Estimation using SV Machines. Technical Report CSD-TR-97-23, Royal Holloway, University of London, 1997.

[78] J. Weston, M. O. Stitson, V. Vovk, A. Gammerman, and V. Vapnik. Learning by Support Vector Machine and Transduction. In *Causal Models and Statistical Learning*, Uxbridge, UK, March 1997. UNICOM Seminars.

[79] J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Royal Holloway, University of London, Egham, UK, 1998.