TAMPERE UNIVERSITY OF TECHNOLOGY

Department of Information Technology

Heikki Orsila

Support Vector Machines with Applications

Master of Science Thesis

Subject approved at Department council meeting on
05.06.2002
Examiner:
Prof. Keijo Ruohonen (TUT)
Prof. Timo D. Hämäläinen (TUT)

Alkulause

Tahdon kiittää diplomityöni ohjaajia Jukka Saarinen ja Mikko Lehtokangas hyvistä neuvoista ja mahdollisuudesta tehdä tätä työtä. Sen lisäksi annan kiitokset tämän työn tarkastajille, Keijo Ruohonen ja Timo D. Hämäläinen, rakentavasta palautteesta.

Haluan myös kiittää perhettäni tuesta opiskelu-urallani. Ilman sopivaa kannustusta tätä ei välttämättä olisi tapahtunut. Lisäksi tahdon kiittää ystäviäni oikeanlaisen kiinnostuksen ylläpitämisestä.

Tampereella 17.05.2004

Heikki Orsila

Opiskelijankatu 4E275

33720 Tampere

heikki.orsila@tut.fi

puh. 040 7325989

Foreword

I want to thank my supervisors Jukka Saarinen and Mikko Lehtokangas for giving me good advice and the opportunity to work on this subject. Also, I thank examiners of this work, Keijo Ruohonen and Timo D. Hämäläinen, for constructive feedback.

I also want to thank my family for support in my studying career. Without appropriate guidance this might not have happened. Additionally I want to thank my friends for keeping me interested in right kind of topics.

Tampere, May 17, 2004

Heikki Orsila

Opiskelijankatu 4E275

33720 Tampere

heikki.orsila@tut.fi

tel. 040 7325989

Contents

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO

Tietotekniikan osasto

Matematiikan laitos ja digitaali- ja tietokonetekniikan laitos

ORSILA, HEIKKI: Support Vector Machines with Applications

Diplomityö, 74 s.

Tarkastajat: Prof. Keijo Ruohonen ja Prof. Timo D. Hämäläinen

Rahoittaja: Tampereen Teknillinen Yliopisto

Toukokuu 2004

Avainsanat: tukivektorikone, neuroverkot, hahmontunnistus

Tukivektorikone (*Support Vector Machine*) on suhteellisen uusi neuroverkkotyyppi, joka lähestyy aihepiiriä tilastollisen riskin minimoimisen näkökulmasta. Mahdollinen hyöty tukivektorikoneesta on parempi yleistämiskyky kuin perinteisillä neuroverkoilla.

Tämä diplomityö käsittelee tukivektorikone-tyyppisiä neuroverkkoja ja niiden sovelluksia tietyissä hahmontunnistustehtävissä. Erityisesti työn tavoitteena on eräältä teollisuusyritykseltä annetun puheentunnistusongelman ratkaiseminen. Lisäksi työn tarkoituksena on arvioida tukivektorikoneiden soveltuvuutta yleisesti hahmontunnistustehtäviin, ja verrata sillä saavutettuja tuloksia muilla neuroverkoilla saavutettuihin tuloksiin.

Ensimmäiseksi tukivektorikoneita sovelletaan keinotekoiseen moniulotteiseen luokittelumalliin, jotta voitaisiin arvioida tukivektorikoneiden kykyä suoriutua suurista ulottuvuusmääristä. Testi osoittaa tukivektorikoneen pääsevän lähelle teoreettisesti mahdollista luokittelukykyä.

Toiseksi tukivektorikoneita sovelletaan ilmakehän mittauksien luokitteluun. Mittauksien luokittelutuloksia verrataan muiden neuroverkkojen vastaaviin tuloksiin

samassa tehtävässä. Tukivektorikoneet havaitaan varsin kilpailukykyiseksi välineeksi tähän tehtävään hyvin pienellä esikäsittelyvaivalla. Lisäksi havaitaan, että tätä tutkimusta varten muokattu versio tukivektorikoneista antaa pienen, mutta havaittavan, parannuksen tähän luokittelutehtävään.

Kolmanneksi tukivektorikoneita sovelletaan puheentunnistusongelmaan. Tälle ongelmalle etsitään ratkaisua, joka käyttää mahdollisimman pientä määrää tukivektoreita. Suuri tukivektoreiden määrä on neuroverkolle laskennallisesti työlästä ja vähentää neuroverkon yleistämiskykyä. Tulokset validoivat teollisuusyrityksen tuotekehityksen saamia tuloksia riippumattomasti. SMO-algoritmin todetaan soveltuvan parhaiten tähän ongelmatyyppiin. Sopivalla harjoitusnäytteiden valinnalla löydetään luokittelija, jonka tukivektoreiden kokonaismäärä on riittävän pieni. Tukivektoreiden kokonaismäärän rajoittamista valitsemalla harjoitusnäytteet verrataan toiseen tukivektorikone-algoritmiin, joka tuottaa ratkaisuja joissa tukivektoreiden määrä on rajoitettu jo ongelman asettelussa, ja todetaan että se ei tuo lisähyötyä tässä sovelluksessa. Ongelman ratkaisuksi suositellaan *polynomi*-tyyppistä tukivektorikonetta.

Lisäksi esitellään algoritmi häviölliselle kuvanpakkaukselle käyttäen tukivektorikoneita. Algoritmi muuntaa lohkoja kuvasta *Lagrange*-kerroinavaruuteen, jossa olisi tarkoitus olla vähemmän informaatiota kuin alkuperäisessä kuvassa. Menetelmää käyttäen pystytään häviöllisen kuvanpakkauksen tarkkuutta säätämään ennakoitavasti. Menetelmän toimivuutta ei kuitenkaan validoitu tässä työssä. Siitä riippumatta se vaikuttaa kiinnostavalta jatkotutkimuksen kohteelta.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Department of Information Technology

Department of Mathematics and Institute of Digital and Computer Systems

ORSILA, HEIKKI: Support Vector Machines with Applications

Master of Science Thesis, 74 pages

Examiner: Prof. Keijo Ruohonen and Prof. Timo D. Hämäläinen

Funding: Tampere University of Technology

May 2004

Keywords: support vector machine, neural network, pattern recognition

The thesis covers Support Vector Machine type Neural Networks and their applications on specific pattern recognition tasks.

Financial support for the work was provided by Tampere University of Technology.

Support Vector Machine is a relatively new Neural Network type approaching the subject area from Structural Risk Minimization principle. The suggested benefit of Support Vector Machine is better generalization ability than with traditional Neural Networks such as Multilayer Perceptrons.

Support Vector Machines were applied to 3 specific cases of pattern recognition tasks. One of the cases uses data from a real speech recognition problem from an industrial company.

Abbreviations

BSP	Binary Space Partitioning
ERM	Empirical Risk Minimization
FTP	File Transfer Protocol (Internet protocol)
LP	Linear Programming
MLP	Multilayer Perceptron
PCA	Principal Component Analysis
QP	Quadratic Programming
RBF	Radial Basis Function
SMO	Sequential Minimal Optimization
SRM	Structural Risk Minimization
SVM	Support Vector Machine
SVT	Support Vector Transformation
VC	Vapnik-Chervonenkis (dimension)

1 Introduction

Machine Learning means computational methods to *know, predict, guess, estimate and explain* issues based on known data and environment.

Computational units that do *Machine Learning* are called *Learning Machines.* An example of *Machine Learning* is a computer program that plays chess against human or computer players, and learns to play better through its experience. A good learning machine should learn from its mistakes against opponents.

Pattern recognition is a concept of *machine learning.* It consists of computational tasks that try to assign task specific data into one of many classes. Pattern recognition task is often based on measurements from empirical experiments. A *learning machine* will try to learn important details of measurements with some assistance or additional data.

An example of the pattern recognition task would be a set of pictures, where some pictures contain an image of a human face and some pictures don't [13]. A learning machine is told which pictures contain a face and which don't. Based on this data, a learning machine would try to decide whether some given picture contains a face or not. This task is non-trivial since formulating properly how people recognize faces is unknown.

Support Vector Machine (SVM) is a class of learning machines used mainly for *pattern recognition* tasks. SVMs became a general interest in *pattern recognition* in early 1990s through successful experiments and interesting theoretical issues. Since then there has been a lot of research on the subject. SVMs have been compared to older methods, such as *Multilayer Perceptron Neural Networks*, and have been found to perform better in many cases.

This thesis looks into use of SVMs in general *pattern recognition* tasks and applications. Part I of this thesis describes theory of SVMs briefly. Part II describes 3 cases of SVM applications. Finally Part III discusses advantages and disadvantages of SVMs and gives conclusions based on earlier parts.

Part I

Theory

2 Statistical Learning Theory

2.1 Introduction

From pattern recognition perspective *Statistical Learning Theory* tries to establish good theoretical frameworks to determine the class of an observation from the environment. The class for an observation may be *good*, *bad*, *0*, or *1*, depending on circumstances and the environment. The decision is based on some *set* of observations and their known classes (the *data)*. This thesis will occasionally refer to *sets*, but they are not used in the same sense as the mathematical concept of set. Here same entries can exist multiple times in the same set so the proper name is *multi-set*. However, unless explicitly specified, the *set* will here refer to a *multi-set*.

In many cases the amount of example *data* is small, and the *feature space* is multi-dimensional. Usually this means there is only a small set of vectors from R^n to base the decision on. This can cause great trouble for a learning machine, especially if the machine lacks some special knowledge of the system being learned. *Feature space* is the mathematical space where the decision is made. Most commonly it is R^n where n is a small number.

Statistical Learning Theory investigates how to learn as much as possible from a small (or large) amount of *data*. Other subjects are how much data is needed to achieve certain level of performance with a learning machine, or what is the maximum knowledge that can be learned from a specific set of data.

2.2 Basics

Assume a pattern prediction task with a multidimensional input space X. Let Y be the output space. Data vectors $\mathbf{x}$ from X are associated with output values y from

Y. The output space of the problem is $Y = \{0, 1\}$, hence consisting of binary values. For example output value of 0 could mean bad, and 1 good.

Given a set of sample data consisting of pairs $\mathbf{z}_i = (\mathbf{x}_i, y_i)$, where $i = 1, 2, ..., l$, find the best possible output value y, according to some model of performance, for any given vector $\mathbf{x}$ in X.

A *teacher* gives the set of right answers for vectors $\mathbf{x}$ in data set $\{\mathbf{z}_i\}$. Assume the teacher knows a function $f(\mathbf{x})$, which can give the best possible, even *correct*, output for all vectors $\mathbf{x}$ in X. Hence $f(\mathbf{x}_i) = y_i$.

The goal is to find a learning machine, a function $F(\mathbf{x}, w)$, which tries to *mimic* the teacher and realise its answers for all vectors $\mathbf{x}$ in X. Besides choosing a suitable function family for F, the parameter vector $\mathbf{w}$ needs to be determined from the *weight space* W (usually R^n). In a case where the learning machine is a *Neural Network*, its weight values are represented by a vector $\mathbf{w}$. In the optimum situation $f = F$, which implies the learning machine has learned everything from the teacher.

From statistical learning theory perspective it is not only interesting what would be the best possible value for $\mathbf{w}$, but what is the right *size* for the structure (*capacity*) of the learning machine (e.g. the dimension of $\mathbf{w}$) in order to achieve good results.

To achieve good results the learning machine must be able to make good predictions for the data it has not seen before (seen in the learning phase). This goal defines how well the learning machine *generalize*s.

Choosing the right *capacity* for the learning machine is very essential in order for it to be successful in learning. *Capacity* of the learning machine means how much data it can learn to classify *correctly*. With too high a capacity the machine learns all data that is presented to it, but may predict unseen data worse than a machine with lower capacity. This phenomena is called *overfitting*.

On the other hand, with too low a capacity for the machine it can't perhaps learn even the easy patterns. A low capacity has the advantage that it is very unlikely to *overfit* the data, so it has potentially a better *generalization* ability.

Choosing the right capacity for a learning machine is hard. In case of Neural Networks the dimension of $\mathbf{w}$ affects the capacity. The higher the dimension higher the capacity.

2.3 Empirical Risk Minimization

To measure how well learning machine does compared to the teacher, a measure is needed to compare two results for the given $\mathbf{x}$ from input space X. The teacher gives one result, and the machine gives the other. A loss function $L(a, b)$ is used to measure differences between the teacher and the machine. One example of loss function L is $L(a, b) = (a - b)^2$, where a and b belong to output space Y. The loss function compares two answers given for an input vector $\mathbf{x}$ that is presented to the teacher and the learning machine. For example, teacher maps $\mathbf{x}$ into a correct answer a, and the learning machine maps $\mathbf{x}$ into its estimate b. A high loss function value indicates big difference between two given answers. The following expression is evaluated to compare the teacher f and the learning machine F:

$$\epsilon = L(f(\mathbf{x}), F(\mathbf{x}, \mathbf{w}))$$

If $\epsilon = 0$ the learning machine F is correct, otherwise it is not totally correct. Depending on the application the machine can be partially correct. In a *binary classification task* the machine is either totally correct or totally wrong. Given a set of vectors $\mathbf{x}$ an empirical probability for correctness can always be constructed. An *Empirical Risk Function* computes an empirical probability of not being correct.

Empirical Risk Minimization Given a set of sample data $(\mathbf{x}_i, y_i)$ where $i = 1, 2, ..., l$ empirical risk function is defined as:

$$R_{emp}(\mathbf{w}) = \frac{1}{l} \sum_{i=1}^{l} L(f(\mathbf{x}_i), F(\mathbf{x}_i, \mathbf{w})) = \frac{1}{l} \sum_{i=1}^{l} L(y_i, F(\mathbf{x}_i, \mathbf{w}))$$

Empirical Risk Minimization (ERM) is to find $\mathbf{w}^* \in W$ which minimizes the empirical risk function $R_{emp}(\mathbf{w})$.

For pattern recognition purposes the following loss function is usually used:

$$L_1(y, F(\mathbf{x}, \mathbf{w})) = \begin{cases} 0, & if\ y = F(\mathbf{x}, \mathbf{w}) \\ 1, & if\ y \neq F(\mathbf{x}, \mathbf{w}) \end{cases}$$

The empirical risk function for this loss function gives an empirical probability of error:

$$R_{emp}(\mathbf{w}) = \frac{1}{l}\sum_{\text{i}=1}^{l} L_1(y_i, F(\mathbf{x}_i, \mathbf{w}))$$

If a learning machine F is trained with sample data $\mathbf{x}_i$, the data set is called *training data*. The empirical risk associated with the training data is called *training error*. Data not shown during training is called *test data*. The associated empirical risk is called *training error*.

An important concept for learning machines is generalization error. It is the risk over the whole input space. The whole input space can only be tested in very special cases, and hence the generalization error must be estimated when needed. *Training error* is often used as an approximation of generalization error.

2.4 The Capacity of the Learning Machine

Assume a data set D containing n different samples (here it is *mathematically correct* to call D a set). Binary labelling of data set D means that each sample in D has a binary label 0 or 1. For a given $\mathbf{w}^*$ the classifier function $F(\mathbf{x}, \mathbf{w}^*)$ gives a specific binary labelling for point $\mathbf{x}$. A learning machine $F(\mathbf{x}, \mathbf{w})$ has a VC (*Vapnik-Chervonenkis*) dimension of at least n, if for any binary labelling of data set D there exists a $\mathbf{w}^*$ such that the classifier function $F(\mathbf{x}, \mathbf{w}^*)$ gives the same binary labelling for each point $\mathbf{x}$ in the data set D. The VC dimension of $F(\mathbf{x}, \mathbf{w})$ is the largest n such that the previous statement holds for some data set D. It is not required that the statement holds for all possible data sets D. It needs only hold for some data set D.

Example:

Assume a family of classifier functions $R^2 \to \{0, 1\}$:

$$\left\{\varphi(\mathbf{w}^T\mathbf{x} + b) \mid \mathbf{w} \in R^2, b \in R\right\}$$

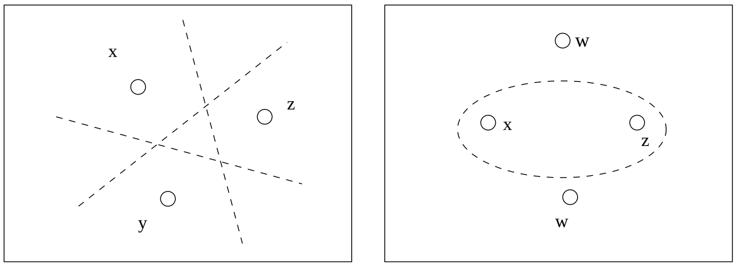

Figure 1: On the left there are 3 points in R^2 which can be classified. On the right there are 4 points in R^2 which can not be classified totally correctly

where $\mathbf{w}$ denotes weight vector and b bias. φ is an activation defined as:

$$\varphi(t) = \begin{cases} 1, \; t \geq 0 \\ 0, \; t < 0 \end{cases}$$

Given 3 separate points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ in R^2 with arbitrary binary labellings, the given family of classifier functions can classify each point correctly, provided that the 3 points are not on the same line. This is shown in the left picture of Figure 1.

Therefore the VC dimension of the function family is at least 3. The VC dimension is 3 since with 4 or more different points they could not have arbitrary binary labellings and be classified all correctly with a line. This can be seen in the right picture of Figure 1. No $(\mathbf{w}, b)$ exists such that it can separate $\{\mathbf{x}, \mathbf{z}\}$ from $\{\mathbf{y}, \mathbf{w}\}$.

More generally, classifier function family

$$\left\{\varphi(\mathbf{w}^T\mathbf{x} + b) \mid \mathbf{w} \in R^m, b \in R\right\}$$

has VC dimension of $m + 1$.

In practice VC dimension relates to dimension of free parameters ($(\mathbf{w}, b)$ in the previous example) so that higher dimension usually implies higher VC dimension. In MLP (*multilayer perceptron*) neural networks with sigmoid activation function

$$\varphi(t) = \frac{1}{1 + \exp(-t)}$$

the VC dimension is proportional to n_w^2, where n_w is the total number of free parameters in the network.

In theory, VC dimension of family of classifier functions can be *infinite* with just one free parameter. This is true because arbitrary amount of information can be encoded into just one real number. For example, pick a function

$$f(x, c) = sgn(c_{r(x)})$$

where c is a free parameter that is presented in binary form as

$$c = \pm \cdots c_1 c_0 . c_{-1} c_{-2} \cdots$$

where $c_i \in \{0, 1\}$ and $i \in Z$. Function $r(x) : R \to Z$ rounds a number x in R to the nearest integer in Z.

Now it is possible to choose arbitrary number of points belonging to Z, and choose digits of c so that they match arbitrarily given binary labels for those points in Z. Thus the VC dimension of this set of classifier functions is *infinite*.

2.5 Structural Risk Minimization

The goal of the learning machine is to minimize generalization error on the underlying task. In practice this means to minimize the empirical error on data that has not been seen in the training process. With a small amount of training data it is easy to overfit a learning model to the available data and thus have lesser generalization ability.

Vapnik [25] investigated theoretical aspects that link the generalization ability to the VC dimension h of a learning machine and the amount of training data N available. An upper bound for generalization error is:

$$v_{gene} < v_{guaranteed} = v_{train} + e_1(N, h, v_{train})$$

where $e_1(N, h, v_{train})$ is the *confidence interval* which depends on the accepted risk for misclassification. Details can be found from [25, 9, 8]. v_{gene}, $v_{guaranteed}$ and v_{train} are generalization, guaranteed and training errors respectively. Vapnik's suggestion

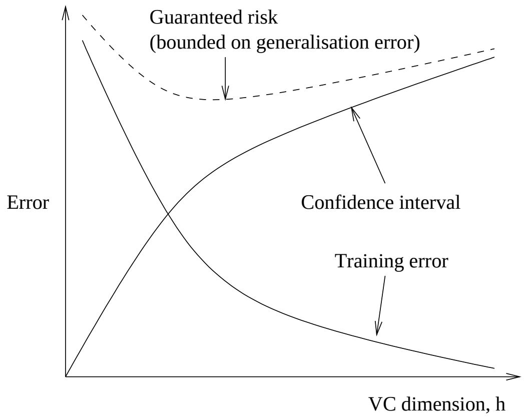

Figure 2: Relationship between training error, confidence interval, and guaranteed risk

was to minimize guaranteed error instead of generalization error, because generalization error is usually unknown. Minimizing the guaranteed error minimizes the upper bound of generalization error. Generally speaking structural risk minimization methods such as SVMs try to find the minimum guaranteed error. Figure 2 shows relationship between training error, confidence interval, and guaranteed error.

Assume there is a collection of families ξ_k of classifier functions representing different learning machines. Each family ξ_k is a class of learning machines with a specific weight space size.

$$\xi_k = \{F(\mathbf{x}, \mathbf{w}) \mid \mathbf{w} \in W_k\},\ k = 1, ..., n$$

where the families are in increasing weight space order:

$$\xi_1 \subset \xi_2 \subset \ldots \subset \xi_{n-1} \subset \xi_n$$

VC dimensions h_i of families ξ_i satisfy

$$h_1 \leq h_2 \leq \ldots \leq h_{n-1} \leq h_n$$

Structural Risk Minimization With these definitions *Structural Risk Minimization* (SRM) can be described with following procedure:

1. Train all n learning machines (ξ_k) with the same training data.
2. The learning machine, associated with a classifier function family ξ_k, which has the lowest guaranteed risk is considered the best choice.

Learning machine families should be constructed in increasing VC dimension order. Adding hidden neurons to a simple MLP neural network is an example of this.

Knowing the guaranteed risk is the hardest part of this approach. It is not possible without knowing the exact VC dimension of each learning machine, because the upper bound for generalization error depends on it. Therefore the guaranteed risk must be estimated.

Usually the guaranteed risk is estimated with an empirical risk based on test data. The test data must not belong to training data that was used in training. Usually only a tenth of test data is needed compared to training data to gain accurate estimates of guaranteed risk.

2.6 Different Approaches to SRM

Basically two approaches for SRM are used.

1. First choose capacity (VC dimension) for the learning machine, then minimize the quaranteed risk.
2. First choose quaranteed risk, and then minimize capacity for the learning machine.

Traditional Multilayer Perceptron neural networks employ the first approach, but *Support Vector Machines* use the second approach.

3 Linear Learning Machines

3.1 Introduction

This section introduces *Linear Learning Machines*, a basic concept of pattern recognition theory. Generally speaking *Linear Learning Machines* form a decision boundary between two classes in a feature space. This knowledge is later used to inspect mechanisms of SVMs. The idea of SVMs is to find an optimal decision boundary between two classes in an implicit feature space. The feature space is implicit since it may not be known explicitly.

3.2 Linear Classification

Assume two classes of points in R^n with l points together. Let those points be defined as $\mathbf{x}_i$ where $i = 1, \ldots, l$. *Linear Classification* is using an *affine* function $f(\mathbf{x})$ to classify point $\mathbf{x}$ into one of two classes, either class 1 or class 2. If $f(\mathbf{x}) \geq 0$, $\mathbf{x}$ is assigned to class 1, otherwise class 2.

Affine function f has presentation $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, where $\mathbf{w} \in R^n$. $\mathbf{w}$ is often referred to as the *weight*, and b as the *bias term.*

Linear Learning Machines *Learning Machines* that do *Linear Classification* are called *Linear Learning Machines.*

3.3 Linear Separation

Assume each point $\mathbf{x}_i$ is associated with y_i, where $y_i = 1$ when $\mathbf{x}_i$ belongs to class 1, and $y_i = -1$ when $\mathbf{x}_i$ belongs to class 2. These are later called positive and negative classes respectively.

Equation $\mathbf{w}^T\mathbf{x} + b = 0$ defines the set of points $\mathbf{x}$ belonging to the hyperplane. Hyperplane has two parameters that define it: $\mathbf{w} \in R^n$ and $b \in R$. Here hyperplanes are named with respect to their parameters as a pair $(\mathbf{w}, b)$.

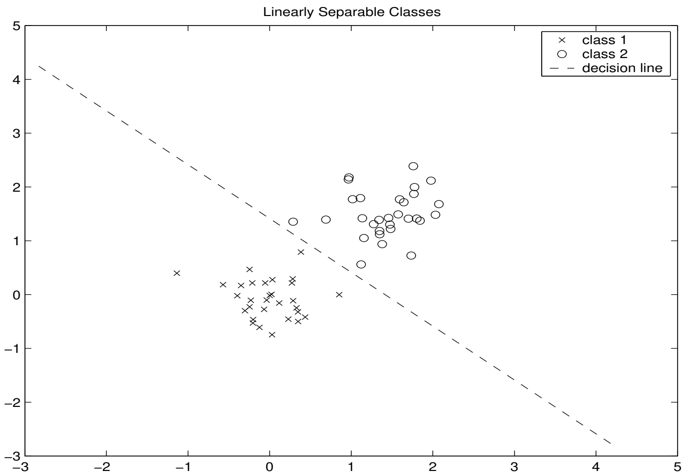

Figure 3: Two linearly separable classes

These points of two classes are *linearly* separable if there exists a hyperplane $(\mathbf{w}, b)$ such that

$$\mathbf{w}^T\mathbf{x}_i + b > 0 \text{ when } \mathbf{x}_i \text{ belongs to class 1 } (y_i = 1)$$
and
$$\mathbf{w}^T\mathbf{x}_i + b < 0 \text{ when } \mathbf{x}_i \text{ belongs to class 2 } (y_i = -1)$$

That is

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0 \text{ for all } \mathbf{x}_i$$

Figure 3 shows an example of linearly separable classes.

Functional margin of a point $\mathbf{x}$ with respect to hyperplane $(\mathbf{w}, b)$ is defined as $\mathbf{w}^T\mathbf{x}+b$.

The distance of a point $\mathbf{x}$ from the hyperplane $(\mathbf{w}, b)$ can be computed with:

$$\delta = \frac{|\mathbf{w}^T\mathbf{x}+b|}{||\mathbf{w}||}$$

δ is called the *geometrical margin* of point $\mathbf{x}$ with respect to the hyperplane.

3.4 Maximal Marginal Classifier

Assume point $\mathbf{x}_m$ belonging to class 1 with geometrical margin δ_m is the closest point in it's class to the hyperplane and respectively point $\mathbf{x}_n$ belonging to class 2 with geometrical margin δ_n is the closest point in it's class to the same hyperplane. Hyperplane $(\mathbf{w}, b)$ is called optimal separator or maximal marginal classifier if $\delta_m = \delta_n$. Notice that if hyperplane $(\mathbf{w}, b)$ separates two classes, then so does $(c\mathbf{w}, cb)$ for all c > 0. Thus c may be chosen so that optimal separator hyperplane $(c\mathbf{w}, cb)$ has functional margin of ± 1 for $\mathbf{x}_m$ and $\mathbf{x}_n$. Assume $(\mathbf{w}, b)$ has that property.

Now the *total geometrical margin* for optimal hyperplane $(\mathbf{w}, b)$ is:

$$\delta_m + \delta_n = 2\delta_m = 2\frac{\left|\mathbf{w}^T\mathbf{x}_m + b\right|}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

When functional margins are restricted to as described then minimizing the *Euclidean norm* of hyperplane normal gives the maximal geometrical margin. Notice that for all points $\mathbf{x}_i$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ is satisfied.

Maximal marginal classifier problem can now be formulated as a minimization problem of the following form.

Minimize

$$\frac{1}{2}\|\mathbf{w}\|^2$$

with respect to $(\mathbf{w}, b)$ when satisfying boundary conditions

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for } i = 1, \ldots, l$$

Solution of this problem is a hyperplane $(\mathbf{w}^*, b^*)$. Then the *optimal* decision function for classifying points into one of two classes is

$$f(\mathbf{x}) = \mathbf{w}^{*T}\mathbf{x} + b^*$$

Point $\mathbf{x}$ is classified as follows:

- classified to class 1, if $f(\mathbf{x}) > 0$
- classified to class 2 otherwise

This problem is investigated in more detail in Section 4, Optimization Theory.

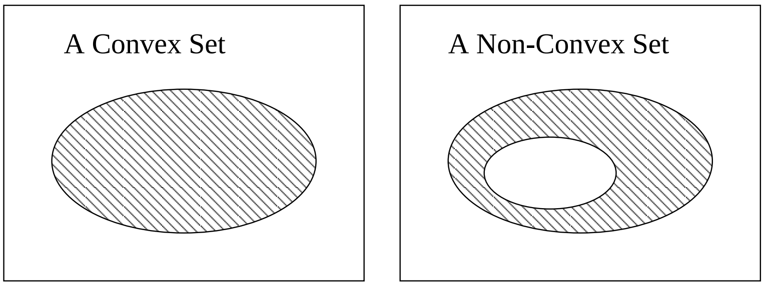

Figure 4: An example of a convex set at left, and a non-convex set at right.

4 Optimization Theory

4.1 Introduction

Some basic optimization theory is presented in this chapter. These concepts will be essential to understanding how SVM based machine learning works. First the concept of *convex functions* is introduced, which is very relevant to SVM learning and mathematical optimization theory in general. Then properties and examples of convex optimization are presented with suitable applications in focus.

4.2 Convex Functions

Convex Set Set $E \subset R^n$ is *convex* if $t\mathbf{x} + (1-t)\mathbf{y} \in E$ for all $\mathbf{x}, \mathbf{y} \in E$ and $t \in [0, 1]$. Figure 4 shows an example of a convex and a non-convex set.

Convex Functions Function $f : E \to R$ is a *convex function*, if

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y})$$

for all $\mathbf{x} \neq \mathbf{y} \in E$ and $t \in (0, 1)$. An example of a convex and a non-convex function is shown in Figure 5.

Examples

Here are a few examples of *convex functions*, which will be later used in SVM context.

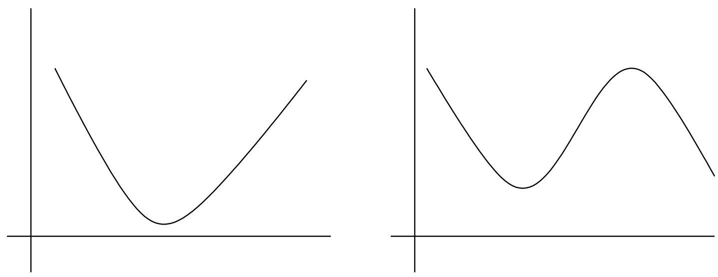

Figure 5: An example of a convex function at left, and a non-convex function at right.

Affine Functions

Function $f : R^n \to R$ is *affine*, if it has the form $f(\mathbf{x}) = \mathbf{b}^T\mathbf{x} + c$ where $\mathbf{b} \in R^n$ and $c \in R$. Affine functions are convex since

$f(t\mathbf{x} + (1-t)\mathbf{y}) = \mathbf{b}^T(t\mathbf{x} + (1-t)\mathbf{y}) + c = t(\mathbf{b}^T\mathbf{x}) + (1-t)(\mathbf{b}^T\mathbf{y}) + tc + (1-t)c$

$= t(\mathbf{b}^T\mathbf{x} + c) + (1-t)(\mathbf{b}^T\mathbf{y} + c) = tf(\mathbf{x}) + (1-t)f(\mathbf{y})$

Quadratic Functions

Function $f(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}$, where A is positive semidefinite matrix, is a convex function. The following proof is taken from [11] p.46.

Let $\mathbf{x}, \mathbf{y} \in R^n$ and $t \in (0, 1)$. Now

$$0 \le f(\mathbf{x} - \mathbf{y}) = \mathbf{x}^T A\mathbf{x} - \mathbf{x}^T A\mathbf{y} - \mathbf{y}^T A\mathbf{x} + \mathbf{y}^T A\mathbf{y},$$

since A is positive semidefinite. So

$$\mathbf{x}^T A\mathbf{y} + \mathbf{y}^T A\mathbf{x} \le \mathbf{x}^T A\mathbf{x} + \mathbf{y}^T A\mathbf{y}$$

Thus

$$\begin{aligned}
&f(t\mathbf{x} + (1-t)\mathbf{y}) = (t\mathbf{x} + (1-t)\mathbf{y})^T A(t\mathbf{x} + (1-t)\mathbf{y}) \\
&= (t\mathbf{x}^T A + (1-t)\mathbf{y}^T A)(t\mathbf{x} + (1-t)\mathbf{y}) \\
&= t^2\mathbf{x}^T A\mathbf{x} + t(1-t)\mathbf{x}^T A\mathbf{y} + t(1-t)\mathbf{y}^T A\mathbf{x} + (1-t)^2\mathbf{y}^T A\mathbf{y} \\
&= t^2\mathbf{x}^T A\mathbf{x} + t(1-t)[\mathbf{x}^T A\mathbf{y} + \mathbf{y}^T A\mathbf{x}] + (1-t)^2\mathbf{y}^T A\mathbf{y} \\
&\le t^2\mathbf{x}^T A\mathbf{x} + t(1-t)[\mathbf{x}^T A\mathbf{x} + \mathbf{y}^T A\mathbf{y}] + (1-t)^2\mathbf{y}^T A\mathbf{y}
\end{aligned}$$

$$= t^2\mathbf{x}^T A\mathbf{x} - t^2\mathbf{x}^T A\mathbf{x} + t\mathbf{x}^T A\mathbf{x} + t\mathbf{y}^T A\mathbf{y} - t^2\mathbf{y}^T A\mathbf{y} + (1-t)^2\mathbf{y}^T A\mathbf{y}$$
$$= t\mathbf{x}^T A\mathbf{x} + (t - t^2 + (1-t)^2)\mathbf{y}^T A\mathbf{y}$$
$$= t\mathbf{x}^T A\mathbf{x} + (t - t^2 + 1 - 2t + t^2)\mathbf{y}^T A\mathbf{y} = t\mathbf{x}^T A\mathbf{x} + (1-t)\mathbf{y}^T A\mathbf{y}$$
$$= tf(\mathbf{x}) + (1-t)f(\mathbf{y})$$

Composite Functions

If $g_i(\mathbf{x})$, where $i = 1, ..., n$, are convex functions, and $c_i > 0$ for all i, then

$$f(\mathbf{x}) = \textstyle\sum_{i=1}^{n} c_i g_i(\mathbf{x})$$

is convex too.

4.3 Theory

Some basic optimization theory is presented here to support optimization in SVM context. Especially optimization of quadratic functions with affine *boundary constraints* is needed with SVM learning.

4.3.1 Primal Problem

Minimize $f(\mathbf{w})$ with respect to $\mathbf{w}$, satisfying following conditions (*boundary constraints*):

$$g_i(\mathbf{w}) \leq 0,\ i = 1, ..., m \text{ (inequality constraints)}$$
$$h_j(\mathbf{w}) = 0,\ j = 1, ..., p \text{ (equality constraints)}$$

Functions $f : R^n \to R$, $g_i : R^n \to R$ and $h_j : R^n \to R$ are continuous and differentiable in R^n (*well behaved* in other words). Constraints can be written in vector form as follows:

$$\mathbf{g}(\mathbf{w}) \leq 0$$
$$\mathbf{h}(\mathbf{w}) = 0$$

The *feasible region* of the problem is that set F for which all constraints are satisfied.

$$F = \{\mathbf{w} \in R^n \mid \mathbf{g}(\mathbf{w}) \leq 0, \mathbf{h}(\mathbf{w}) = 0\}$$

It may not be trivial to find any point from the *feasible region* F for a given problem even when such points exist.

When constraints are *linear functions* the optimization problem is called *linear programming* (LP) problem. When they are *quadratic functions* the problem is called *quadratic programming problem* (QP).

For a given feasible solution $\mathbf{w}^*$ inequality constraint $g_i(\mathbf{w}^*) \leq 0$ is said to be *active* if $g_i(\mathbf{w}^*) = 0$, otherwise it is *inactive.*

4.3.2 Convex Optimum Theorem

If $f \in C^1(R)$ is convex and $\mathbf{w}^*$ satisfies $\nabla f(\mathbf{w}^*) = 0$ in unconstrained optimization problem, then $\mathbf{w}^*$ is the global optimum point.

Proof Choose any point $\mathbf{u} \neq \mathbf{w}^*$. Since $\mathbf{w}^*$ is local minimum, that is $\nabla f(\mathbf{w}^*) = 0$, there exists a value t close enough to 1 such that

$$f(\mathbf{w}^*) \leq f(t\mathbf{w}^* + (1-t)\mathbf{u})$$

And since f is convex

$$f(t\mathbf{w}^* + (1-t)\mathbf{u}) \leq tf(\mathbf{w}^*) + (1-t)f(\mathbf{u})$$

And so it must be that

$$\begin{aligned}
&f(\mathbf{w}^*) \leq tf(\mathbf{w}^*) + (1-t)f(\mathbf{u}) \\
&(1-t)f(\mathbf{w}^*) \leq (1-t)f(\mathbf{u}) \\
&f(\mathbf{w}^*) \leq f(\mathbf{u})
\end{aligned}$$

So $\mathbf{w}^*$ is a global minimum, which is not necessarily *unique.* See also [11] p.45-46 (section 3.2.2) or [8] p.81 (section 5.2) for proof.

4.3.3 Minimum Maximum Existence Theorem

If the feasible region F is *closed* and *bounded*, and f is *continuous*, then f has a *minimum* and a *maximum* in F. Later on C-SVM optimization problem will have guaranteed minimum based on this theorem.

Proof See one of many mathematical analysis books, such as [15] section 4.15.

4.3.4 Kuhn-Tucker Theorem

Let $\mathbf{w}^* \in F$ be a feasible solution to a given *primal problem*. The necessary condition (**Kuhn-Tucker** conditions) for $\mathbf{w}^*$ to be the minimum point is that there exist $\lambda_i \geq 0, \mu_j \in R$ such that

$$
\begin{array}{l}
\nabla f(\mathbf{w}^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\mathbf{w}^*) + \sum_{j=1}^{p} \mu_j \nabla h_j(\mathbf{w}^*) = 0 \\
\lambda_i g_i(\mathbf{w}^*) = 0 \text{ for all } i = 1, ..., m \\
g_i(\mathbf{w}^*) \leq 0 \text{ and } h_j(\mathbf{w}^*) = 0 \text{ for all } i = 1, ..., m \text{ and } j = 1, ..., p
\end{array}
$$

Proof See [1].

Points satisfying these conditions are called *Kuhn-Tucker points.* Kuhn-Tucker Theorem is central to many practical continuous optimization problems.

4.3.5 Lagrangian Function

Define $\lambda = [\lambda_1, \lambda_2, ..., \lambda_m]^T$ and $\mu = [\mu_1, \mu_2, ..., \mu_p]^T$

The *Lagrangian function* related to the previous Kuhn-Tucker Conditions (4.3.4) is

$$L(\mathbf{w}, \lambda, \mu) = f(\mathbf{w}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{w}) + \sum_{j=1}^{p} \mu_j h_j(\mathbf{w})$$

or

$$L(\mathbf{w}, \lambda, \mu) = f(\mathbf{w}) + \mathbf{g}(\mathbf{w})^T \lambda + \mathbf{h}(\mathbf{w})^T \mu$$

Kuhn-Tucker Conditions in Theorem 4.3.4 can be written as

$$
\begin{array}{l}
\frac{\partial L(\mathbf{w}, \lambda, \mu)}{\partial \mathbf{w}} = \nabla f(\mathbf{w}) + \mathbf{g}'(\mathbf{w})^T \lambda + \mathbf{h}'(\mathbf{w})^T \mu = 0 \\
\lambda_i g_i(\mathbf{w}^*) = 0 \text{ for all } i = 1, ..., m \\
g_i(\mathbf{w}^*) \leq 0 \text{ and } h_j(\mathbf{w}^*) = 0 \text{ for all } i = 1, ..., m \text{ and } j = 1, ..., p
\end{array}
$$

4.3.6 Sufficient Conditions for Kuhn-Tucker Theorem

Kuhn-Tucker Conditions presented in Theorem 4.3.4 are sufficient, if functions g_i are *convex*, h_j are *affine* and there exists such $\mathbf{w}$ that $g_i(\mathbf{w}) < 0$ for all i and $h_j(\mathbf{w}) = 0$ for all j.

Proof See [1].

This theorem is used in Section 4.3.8 to solve a *maximal margin classifier* problem.

4.3.7 Stronger Sufficient Conditions for Kuhn-Tucker Theorem

Kuhn-Tucker Conditions presented in Theorem 4.3.4 are sufficient, if f is *convex* and has a continuous first derivative, and the optimization domain is convex, and functions g_i and h_j are *affine*, and there exists such $\mathbf{w}$ that $g_i(\mathbf{w}) \leq 0$ for all i and $h_j(\mathbf{w}) = 0$ for all j.

Proof See [1].

This theorem is used in Section 5.3 to solve one of the core problems.

4.3.8 Maximal Marginal Classifier Solution

The *maximal marginal classifier* was formulated in Section 3.4, and now the optimization framework is applied to solve it. The solution here leads to a QP problem. The problem restated is:

Minimize

$$\frac{1}{2}\left\|\mathbf{w}\right\|^2$$

with respect to $(\mathbf{w}, b)$ when satisfying boundary conditions

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for } i = 1, \ldots, l$$

It is assumed here that an optimal solution for the optimization problem exists, that is, classes to be separated are *linearly separable.*

Objective function $f(\mathbf{w})$, which should be minimized, and l inequality contraint functions $g_i(\mathbf{w})$ are defined as follows:

$$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$g_i(\mathbf{w}) = 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0 \text{ for all } i = 1, ..., l$$

Now the *Lagrangian function* is

$$L(\mathbf{w}, b, \alpha) = f(\mathbf{w}) + \sum_{i=1}^{l} \alpha_i g_i(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{l} \alpha_i - \sum_{i=1}^{l} \alpha_i y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

Theorem 4.3.6 shows important properties of the optimum point $\mathbf{w}^*$, because f is a *convex function,* and constraint functions g_i are *affine*, and such $\mathbf{w}$ exists that $g_i(\mathbf{w}) < 0$ for all i because classes are linearly separable. Hence there are the following necessary and sufficient requirements for $\mathbf{w}^*$ to be an optimum with b^*, α^*:

$$\begin{aligned}
\frac{\partial L(\mathbf{w}^*, b^*, \alpha^*)}{\partial \mathbf{w}} &= 0, \\
\frac{\partial L(\mathbf{w}^*, b^*, \alpha^*)}{\partial b} &= 0, \\
\alpha_i^* g_i(\mathbf{w}^*) &= 0, \quad i = 1, ..., l \\
g_i(\mathbf{w}^*) &\leq 0, \quad i = 1, ..., l \\
\alpha_i^* &\geq 0, \quad i = 1, ..., l
\end{aligned}$$

Then

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum \alpha_i y_i \mathbf{x}_i = 0 \text{ or } \mathbf{w} = \sum a_i y_i \mathbf{x}_i$$

and

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = -\sum \alpha_i y_i = 0$$

Define $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_l]^T$, $\mathbf{y} = [y_1, y_2, ..., y_l]^T$, diagonal matrix $Y = diag(y_1, y_2, ..., y_l)$, $\alpha = [\alpha_1, \alpha_2, ..., \alpha_l]^T$ and $1 = [1, 1, ..., 1]^T \in R^l$.

So, previous optimality conditions become

$$\mathbf{w} = X^T\mathbf{y}\alpha \text{ and } 1^T\alpha = 0$$

Substituting these back to Lagrangian gives

$$
\begin{aligned}
&L(\mathbf{w}, b, \alpha) = \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + \textstyle\sum_{i=1}^{l} \alpha_i - \sum_{i=1}^{l} \alpha_i y_i(\mathbf{w}^T\mathbf{x}_i + b) \\
&= \tfrac{1}{2}(X^TY\alpha)^T(X^TY\alpha) + 1^T\alpha - \Sigma_{i=1}^{l}[(X^TY\alpha)^T\alpha_i y_i\mathbf{x}_i + \alpha_i y_i b] \\
&= \tfrac{1}{2}(X^TY\alpha)^T(X^TY\alpha) + 1^T\alpha - (X^TY\alpha)^T\Sigma_{i=1}^{l}\alpha_i y_i\mathbf{x}_i - \Sigma_{i=1}^{l}\alpha_i y_i b \\
&= \tfrac{1}{2}\alpha^TYXX^TY\alpha + 1^T\alpha - \alpha^TYX\Sigma_{i=1}^{l}\alpha_i y_i\mathbf{x}_i - b\Sigma_{i=1}^{l}\alpha_i y_i \\
&= \tfrac{1}{2}\alpha^TYXX^TY\alpha + 1^T\alpha - \alpha^TYX(X^TY\alpha) - b * 0 \\
&= -\tfrac{1}{2}\alpha^TYXX^TY\alpha + 1^T\alpha
\end{aligned}
$$

Which gives a quadratic programming problem, which solves the optimization task:

Minimize

$$-\tfrac{1}{2}\alpha^TYXX^TY\alpha + 1^T\alpha$$

with respect to α, when

$$\alpha \geq 0$$

If α^* is the optimal solution, then b^* can be solved. Choose any α_i such that $\alpha_i > 0$. From necessary and sufficient conditions of the optimization problem it must hold that

$$\alpha_i^* g_i(\mathbf{w}^*) = 0$$

Since $\alpha_i^* > 0$ it must be that $g_i(\mathbf{w}^*) = 0$. Hence

$$g_i(\mathbf{w}^*) = 1 - y_i(\mathbf{w}^{*^T}\mathbf{x}_i + b^*) = 0$$

which implies

$$b^* = y_i - \mathbf{w}^{*^T}\mathbf{x}_i = y_i - \alpha^{*^T}YX\mathbf{x}_i$$

4.3.9 Example: Minimal Sphere Bounding Problem

Assume l points $\mathbf{x}_i$ in R^n and the goal is to find a sphere centered at $\mathbf{x}^*$ with minimal radius r such that it contains all points $\mathbf{x}_i$. It must be assumed here that not all points $\mathbf{x}_i$ are the same, otherwise the problem is trivial.

The optimization problem reduces to:

Minimize

r^2

with respect to $(\mathbf{x}, r)$ such that following inequality constraints are satisfied

$\|\mathbf{x}_i - \mathbf{x}\|^2 \le r^2$ for $i = 1, ..., l$
$r \ge 0$

Now it is assumed that the solution exists without any *proof*. Hence it can be deduced that the solution exists at some Kuhn-Tucker Point (necessary condition due to Kuhn-Tucker Theorem 4.3.4).

Problem restated is:

Minimize

$$f(\mathbf{x}, r) = r^2$$

when

$$\mathbf{g}_i(\mathbf{x}, r) = \mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}^T \mathbf{x}_i + \mathbf{x}^T \mathbf{x} - r^2 \le 0 \text{ for } i = 1, ..., l$$
$$-r \le 0$$

The *Lagrangian function* is

$$L(\mathbf{x}, r, \alpha, \gamma) = r^2 + \sum_{i=1}^{l} \alpha_i (\mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}^T \mathbf{x}_i + \mathbf{x}^T \mathbf{x} - r^2) + \gamma(-r)$$

It is *known* that for optimal solution $r > 0$ so Lagrange multiplier γ is zero for the optimal solution. The last term is dropped from the Lagrangian function, because it is zero.

From Necessary Kuhn-Tucker Conditions for optimal solution it must be that

$$\frac{\partial L(\mathbf{x},r,\alpha,\gamma)}{\partial r} = 2r - 2r \sum \alpha_i = 0 \text{ that is } 2r(1 - \sum \alpha_i) = 0$$

Since $r > 0$ for optimal solution, it must be that $\sum \alpha_i = 1$.

Also $\frac{\partial L(\mathbf{x},r,\alpha,\gamma)}{\partial \mathbf{x}} = \sum \alpha_i (-2\mathbf{x}_i + 2\mathbf{x}) = 2 \sum \alpha_i \mathbf{x} - 2 \sum \alpha_i \mathbf{x}_i = 2\mathbf{x} \sum \alpha_i - 2 \sum \alpha_i \mathbf{x}_i = 0$. Since $\sum \alpha_i = 1$, it follows that $2\mathbf{x} \sum \alpha_i = 2\mathbf{x} = 2 \sum \alpha_i \mathbf{x}_i$. And hence

$$\mathbf{x} = \sum \alpha_i \mathbf{x}_i$$

Define (as in the previous example) $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_l]^T$, $\alpha = [\alpha_1, \alpha_2, ..., \alpha_l]^T$, $\mathbf{c} = [\mathbf{x}_1^T \mathbf{x}_1, \mathbf{x}_2^T \mathbf{x}_2, ..., \mathbf{x}_l^T \mathbf{x}_l]^T$ and $1 = [1, 1, ..., 1]^T \in R^l$.

Previous optimality conditions, necessary Kuhn-Tucker conditions, become

$$\mathbf{x} = X^T\alpha$$
$$1^T\alpha = 1$$

Now assuming $(\mathbf{x}, r)$ is optimal:

$$L(\mathbf{x}, r, \alpha, \gamma) = r^2 + \sum_{i=1}^{l} \alpha_i(\mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{x}^T\mathbf{x}_i + \mathbf{x}^T\mathbf{x} - r^2)$$
$$= r^2 - r^2 \sum \alpha_i + \sum_{i=1}^{l} \alpha_i(\mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{x}^T\mathbf{x}_i + \mathbf{x}^T\mathbf{x})$$
$$= \sum_{i=1}^{l} \alpha_i\mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{x}^T \sum \alpha_i\mathbf{x}_i + \mathbf{x}^T\mathbf{x} \sum \alpha_i$$
$$= \mathbf{c}^T\alpha - 2\mathbf{x}^T\mathbf{x} + \mathbf{x}^T\mathbf{x}$$
$$= \mathbf{c}^T\alpha - \mathbf{x}^T\mathbf{x}$$
$$= \mathbf{c}^T\alpha - (X^T\alpha)^T(X^T\alpha)$$
$$= \mathbf{c}^T\alpha - \alpha^T XX^T\alpha$$

which becomes a QP problem:

Maximize

$$\mathbf{c}^T\alpha - \alpha^T XX^T\alpha$$

with respect to α when

$$\alpha \geq 0$$
$$1^T\alpha = 1$$

5 Support Vector Machines

5.1 Introduction

This section introduces common *Support Vector Machine* (SVM) theory and methods as well as an applied variant of the SVM classifier (5.4) with special applications in mind. SVMs are compared to other learning machines.

5.2 SVM Learning Theory

SVM system is theoretically different from many existing learning methods like MLPs with back-propagation. SVM training is an SRM method. It has an SRM method built directly into the learning process. MLP training can also be categorized into SRM methods if the size of the MLP network is tuned appropriately. Differences between ERM and SRM were discussed in Section 2.5.

An SRM method has two parts:

- minimize the machine construction (learning machine capacity)
- minimize the *empirical risk* (Section 2.3)

The order of these two parts is not fixed. Usually both are not optimized simultaneously. Instead, the other factor is fixed while the other is being optimized. MLPs and SVMs differ in this.

In MLP training the machine construction (learning capacity) is chosen first, and the *empirical risk* is minimized afterwards. In SVM training the *empirical risk* is fixed before training, and the machine construction (learning capacity) is minimized afterwards. In other words, both implement an SRM method but in complementary ways.

5.3 C-SVM

C-SVM, or *soft margin classifier* SVM, is a binary classifier, which was introduced in 1995 by Cortes and Vapnik [6]. C-SVM was a major breakthrough in applicability of SVMs.

C-SVM maps samples from input space into one of two classes. Those two classes are -1 and $+1$. Unlike maximal margin classifier in Section 3.4, C-SVM allows data sets to be linearly non-separable. The predecessor of C-SVM was *optimal margin classifier* by Boser, Guyon and Vapnik [2]. It was limited to linearly separable data sets.

5.3.1 C-SVM problem formulation in R^n

C-SVM is similar to *maximal margin classifier* in 3.4 except that it allows classes to be linearly non-separable. A cost is assigned for breaking linear separability. The problem is:

Minimize

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i$$

with respect to $(\mathbf{w}, b)$ and ξ when satisfying boundary conditions:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

for $i = 1, \ldots, l$

A high level interpretation of this problem formulation is a maximal margin classifier which tolerates crossing the decision boundary $(\mathbf{w}, b)$ with a cost factor of C. If $\xi_i = 0$ for a given vector $\mathbf{x}_i$ then that vector does not cross the boundary because for that vector $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ is satisfied. If $0 < \xi_i < 1$ the vector is classified correctly but it is so near the decision boundary that a cost of $C\xi_i$ is assigned. If $\xi_i \geq 1$ the vector is classified erroneously. A cost associated with this case is $C\xi_i$. The higher the ξ_i more costly it is.

This minimization problem has a compromise between separating the two classes as well as possible (minimize term $\frac{1}{2}\|\mathbf{w}\|^2$) and having the tolerance cost (minimize term $C\sum_{i=1}^{l}\xi_i$) as low as possible. Choosing the C value sets the level of compromise.

5.3.2 C-SVM problem solution in R^n

Assume that a solution for the optimization problem exists. A rigorous mathematical proof is not important here, but notice Theorem 4.3.7. The objective function is *convex*, and it has a continuous derivative, and the optimization domain is *convex*, and inequality constraints are *affine*. Based on this theorem Kuhn-Tucker conditions can be used to transform the problem into a Quadratic Programming problem. The optimization task reformulated is:

$$f(\mathbf{w}, b, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i$$
$$g_i^1(\mathbf{w}, b, \xi_i) = 1 - \xi_i - y_i(\mathbf{w}^T\mathbf{x}_i + b)$$
$$g_i^2(\xi_i) = -\xi_i$$

for $i = 1, \ldots, l$.

Now the *Lagrangian function* is

$L(\mathbf{w}, b, \xi, \alpha, \beta) = f(\mathbf{w}) + \sum_{i=1}^{l}\alpha_i g_i^1 + \sum_{i=1}^{l}\beta_i g_i^2$

$L(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\Sigma_{i=1}^{l}\xi_i + \sum_{i=1}^{l}\alpha_i(1 - \xi_i - y_i(\mathbf{w}^T\mathbf{x}_i + b)) + \sum_{i=1}^{l}\beta_i(-\xi_i)$

Define $1 = [1, 1, ..., 1]^T$.

Now find the necessary (and sufficient) conditions for the optimum:

$$\frac{\partial L(\mathbf{w},b,\xi,\alpha,\beta)}{\partial \mathbf{w}} = \mathbf{w} - \sum \alpha_i y_i \mathbf{x}_i = 0, \text{ or } \mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$$

and

$$\frac{\partial L(\mathbf{w},b,\xi,\alpha,\beta)}{\partial b} = -\sum \alpha_i y_i = 0$$

and

$$\frac{\partial L(\mathbf{w},b,\xi,\alpha,\beta)}{\partial \xi} = C1 - \alpha - \beta = 0, \text{ or } \alpha + \beta = C1$$

Define

$$
\begin{aligned}
&X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_l]^T \\
&\mathbf{y} = [y_1, y_2, ..., y_l]^T \\
&Y = diag(y_1, y_2, ..., y_l) \\
&\alpha = [\alpha_1, \alpha_2, ..., \alpha_l]^T \\
&\beta = [\beta_1, \beta_2, ..., \beta_l] \\
&\xi = [\xi_1, \xi_2, ..., \xi_l]
\end{aligned}
$$

Using definitions of X and Y previous optimality conditions become

$$
\begin{aligned}
&\mathbf{w} = X^T Y \alpha \\
&1^T \alpha = 0 \\
&\alpha + \beta = C1
\end{aligned}
$$

Substituting these back to Lagrangian gives

$$
\begin{aligned}
&L(\mathbf{w}, b, \xi, \alpha, \beta) \\
&= \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\Sigma_{i=1}^{l}\xi_i + \sum_{i=1}^{l}\alpha_i(1 - \xi_i - y_i(\mathbf{w}^T\mathbf{x}_i + b)) + \sum_{i=1}^{l}\beta_i(-\xi_i) \\
&= \tfrac{1}{2}(X^TY\alpha)^T(X^TY\alpha) + C1^T\xi + -(\alpha+\beta)^T\xi + 1^T\alpha + \sum_{i=1}^{l} -\alpha_i y_i((X^TY\alpha)^T\mathbf{x}_i + b) \\
&= \tfrac{1}{2}(X^TY\alpha)^T(X^TY\alpha) + (C1^T - (\alpha+\beta)^T)\xi + 1^T\alpha - \sum_{i=1}^{l}(X^TY\alpha)^T\alpha_i y_i \mathbf{x}_i - b\sum_{i=1}^{l}\alpha_i y_i \\
&= \tfrac{1}{2}(X^TY\alpha)^T(X^TY\alpha) + 0^T\xi + 1^T\alpha - (X^TY\alpha)^T\sum_{i=1}^{l}\alpha_i y_i \mathbf{x}_i - b * 0 \\
&= \tfrac{1}{2}(X^TY\alpha)^T(X^TY\alpha) + 1^T\alpha - (X^TY\alpha)^T\mathbf{w} \\
&= \tfrac{1}{2}(X^TY\alpha)^T(X^TY\alpha) + 1^T\alpha - (X^TY\alpha)^T(X^TY\alpha) \\
&= -\tfrac{1}{2}(X^TY\alpha)^T(X^TY\alpha) + 1^T\alpha \\
&= -\tfrac{1}{2}\alpha^T Y X X^T Y \alpha + 1^T\alpha
\end{aligned}
$$

Which is surprisingly the same cost function as with maximal margin classifier. However, bounds for the optimizing problem are different.

Lagrangian theory tells that $\alpha \geq 0$ and $\beta \geq 0$. Combine this with the optimum condition $\alpha + \beta = C1$, and it must be that $\alpha \leq C1$.

Also, instead of minimizing the cost function Lagrange theory tells to maximize the Lagrangian function. This is supported by the observation that the minimum for the cost function (quadratic convex function) does not exist since it is $-\infty$. Therefore the optimization problem is

Minimize

$$\frac{1}{2}\alpha^T YXX^T Y\alpha - 1^T\alpha$$

with respect to α when satisfying boundary constraints

$$0 \leq \alpha \leq C1$$
$$\mathbf{y}^T\alpha = 0$$

Theorem 4.3.3 shows that this optimization problem has a minimum, because constraints form a closed space. Now it follows from Kuhn-Tucker optimality conditions that the optimum weight vector $\mathbf{w}_o$ is:

$$\mathbf{w}_o = X^T Y\alpha$$

To have a complete linear classifier for R^n the b constant is also needed. The b constant can always be determined from *active* boundary conditions. Consider the optimum point $\mathbf{w}_o$ and active boundary constraints that satisfy $0 < \alpha_i < C$. The equality from boundary constraints optimum conditions is $\alpha_i + \beta_i = C$. Thus $\beta_i > 0$ because $\alpha_i < C$ for all such active boundary constraints, and hence it must be that $\xi_i = 0$ because β_i was a Lagrange multiplier for ξ_i. Hence $g_i^1(\mathbf{w}, b, \xi_i) = 1 - \xi_i - y_i(\mathbf{w}_o^T\mathbf{x}_i + b) = 1 - y_i(\mathbf{w}_o^T\mathbf{x}_i + b)$ which implies means that $g_i^1(\mathbf{w}, b, \xi_i) = 0$ because the constraint was active. Then $0 = 1 - y_i(\mathbf{w}_o^T\mathbf{x}_i + b)$ which leads to:

$$b_o = y_i - \mathbf{w}_o^T\mathbf{x}_i$$

If numerical precision is in doubt, b_o can be averaged among all active boundary constraints satisfying $0 < \alpha_i < C$.

The resulting classifier is:

$$\mathbf{w}_o^T\mathbf{x} + b_o = \alpha^T Y^T X\mathbf{x} + b_o = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b_o$$

For the *optimal* classifier in R^n, $\mathbf{w}_o^T\mathbf{x} + b_o > 0$ implies $\mathbf{x}$ belongs to positive class $+1$, otherwise negative class -1.

5.3.3 Kernel C-SVM

Generally used C-SVMs differ from the previously shown method by doing the classification in an implicit feature space determined by a kernel function $K(\mathbf{x}, \mathbf{y})$. The kernel function K maps two vectors from R^n into some, perhaps unknown, high dimensional feature space and computes an *inner-product* between $\mathbf{x}$ and $\mathbf{y}$ in that space.

Ordinary feature map function $\phi(\mathbf{x})$ transports $\mathbf{x}$ into another feature space more suitable for classification. For example in common pattern recognition $\phi(\mathbf{x}) = FFT(\mathbf{x})$ can be used as a feature map to extract frequency components from a time window $\mathbf{x}$. This is desirable of course only when frequency components have relevant information for classification.

The kernel function $K(\mathbf{x}, \mathbf{y}) = < \phi(\mathbf{x}), \phi(\mathbf{y}) >$ must satisfy special properties so that the implicit feature space exists. The *inner-product* $< \cdot, \cdot >$ in the implicit feature space has the form

$$< \phi(\mathbf{x}), \phi(\mathbf{y}) > = \Sigma \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{y})$$

where $\lambda_i \geq 0$. The feature space can be infinite dimensional, which is often the case with SVMs. Function ϕ is not explicitly needed as long as kernel function K satisfies the conditions set in *Mercer theorem* [7].

Mercer Theorem Let X be a compact subset of R^n. Let K be a continuous symmetric function ($K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x})$) such that the integral operator $T_K : L_2(X) \rightarrow L_2(X)$,

$$(T_K f)(\cdot) = \textstyle\int_X K(\cdot, \mathbf{x}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z}$$

is positive, that is

$$\textstyle\int_{X \times X} K(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0,$$

for all $f \in L_2(X)$. Then we can expand $K(\mathbf{x}, \mathbf{z})$ in a uniformly convergent series (on $X \times X$) in terms of T_K's eigen-functions $\phi_j \in L_2(X)$, normalized in such a way that $\|\phi_j\|_{L_2} = 1$, and positive associated eigenvalues $\lambda_j \geq 0$:

Table 1: Summary of Kernel functions

Kernel type	Kernel function	Parameters and comments
Polynomial kernel	$(\mathbf{x}^T\mathbf{y}+1)^p$	Positive integer p
Radial-basis function kernel	$\exp(-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2)$	Standard deviation σ
Two-layer perceptron	$\tanh(\beta_0\mathbf{x}^T\mathbf{y}+\beta_1)$	Mercer not satisfied for all β_0 and β_1

$$K(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x})\phi_j(\mathbf{z})$$

The previous derivation of SVM learning algorithm using QP depends only on *inner-products* of vectors $\mathbf{x}_i$ in feature space R^n. This extends to all implicit feature spaces induced by kernel function K satisfying the *Mercer theorem.* The objective function for QP is

$$\frac{1}{2}\alpha^T YXX^T Y\alpha - 1^T\alpha$$

Matrix XX^T is the same as matrix $M = (K(\mathbf{x}_i,\mathbf{x}_j))_{u,j=1}^{l}$ when $K(\mathbf{x},\mathbf{y}) = \mathbf{x}^T\mathbf{y}$. Hence the objective function is

$$\frac{1}{2}\alpha^T YMY\alpha - 1^T\alpha$$

The *Mercer* theorem is satisfied if K is symmetric ($K(\mathbf{x},\mathbf{y}) = K(\mathbf{y},\mathbf{x})$ for all $\mathbf{x}$ and $\mathbf{y}$) and if $K \geq 0$ (*positive semidefinite*). [8] p.33 contains a proof of this sufficient condition.

Some common kernel functions K used are depicted in table 1. Choosing the kernel means choosing the implicit feature space used for the pattern classifier. Polynomial kernel of dimension p can learn all p dimensional polynomial features, whereas radial-basis function kernel is a highly local classifier centered around the other kernel function parameter. Where polynomial kernel has a finite dimensional feature space, the Implicit feature space of the radial-basis function kernel is infinite dimensional.

With the implicit feature space some mechanism is needed to classify arbitrary vectors from the input space. The classifier $\mathbf{w}_o^T\mathbf{x}+b$ is evaluated to determine the class of vector $\mathbf{x}$ as follows:

$$\mathbf{w}_o^T\mathbf{x}+b = (X^TY\alpha)^T\mathbf{x}+b = \alpha^TYX\mathbf{x}+b = \alpha^TY\begin{bmatrix} K(\mathbf{x}_1,\mathbf{x}) \\ \ldots \\ K(\mathbf{x}_l,\mathbf{x}) \end{bmatrix} + b$$

Still, a kernel method is needed to determine constant b. The formula $b_o = y_i - \mathbf{w}_o^T \mathbf{x}_i$, where i is an index of a support vector with $0 < \alpha_i < C$, can easily be adapted into the kernel function case:

$$b_o = y_i - \mathbf{w}_o^T \mathbf{x}_i = y_i - \alpha^T Y \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_i) \\ \dots \\ K(\mathbf{x}_l, \mathbf{x}_i) \end{bmatrix}$$

Now the kernel case optimization problem is:

Minimize

$$\tfrac{1}{2}\alpha^T Y K Y \alpha - 1^T \alpha$$

with respect to α with boundary constraints

$$0 \leq \alpha \leq C1$$
$$\mathbf{y^T} \alpha = 0$$

This is the most important SVM specific optimization problem for practical applications. There are several methods discussed later to solve it.

5.4 C^n-SVM

C-SVM optimization problem in Section 5.3 tries to minimize

$$\tfrac{1}{2}\mathbf{w}^T \mathbf{w} + C \sum \xi_i$$
$$+ \textit{ boundary constraints shown before}$$

In a binary classification task it may be preferred to decrease risk of misclassification of some class. For example the positive class may be more important to classify correctly than the negative class. C-SVM is not suitable for this since it assigns constant loss (C) for all training samples. C^n-SVM is a modified method to overcome this limitation.

The different approach is to give a separate loss constant for each training sample. C^n-SVM optimization problem tries to minimize

$\frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum C_i\xi_i$, where $C_i > 0$
$+$ *boundary constraints shown before*

C_i may be chosen to be higher for a class of interest, hence tolerate less loss for that class.

Deriving the optimization solution for this problem is not harder than for C-SVM, and it goes the same way, so the result of derivation is only stated here:

Minimize

$$\frac{1}{2}\alpha^T YXX^T Y\alpha - 1^T\alpha$$

with respect to α with boundary constraints

$$0 \le \alpha_i \le C_i \text{ for } i = 1, ..., l$$
$$\mathbf{y^T}\alpha = 0$$

Notice that only boundary constraints of α_i have changed in the *quadratic programming* task. Hence this is very similar to C-SVM. Kernel function case is as before.

5.5 ν-SVM

A new method called ν-support vector classification [16] allows to control the number of support vectors resulting from training. An implementation of this algorithm can be obtained from a website [5] authored by Chih-Chung Chang and Chih-Jen Lin. Theoretical aspects of this SVM type are beyond the scope of this thesis.

5.6 Function Approximation with SVM

SVMs are extendible to *function approximation* applications as well. With MLP networks function approximation is taken as granted, while SVM was originally aimed at classification. Vapnik, Golowich and Smola showed first how to apply SVM to function approximation in [24].

5.7 Application Methods

5.7.1 Parameter Estimation

Estimating parameters, such as C or any kernel parameters, for SVM is generally a brute force task. Iterating over training and verification phases of the given problem is currently the only known method to find out good parameters. Methods for determining free parameters have been investigated without significant results.

5.7.2 Quadratic Programming

Since SVM training was originally formulated as a quadratic programming task earlier publications have used this approach. However, QP is not well suited for big data sets since the memory requirement is $O(n^2)$ with respect to training samples n. The kernel matrix K needs memory on order n^2. When $n = 10000$ this yields almost 800 Megabytes of memory as 64-bit *floating-point* values. QP optimization scales to n of several thousands, but not higher than that.

5.7.3 Sample Decomposition

Sample Decomposition (chunking) can use quadratic programming as a tool. It is not a full solution to the problem. It needs help of other training algorithms to be used.

The brief idea of Sample Decomposition [12] is to start with a small subset of the training set. Train the subset. Drop all the non-support vectors from the subset, and see which samples of the rest of the big training set give false classifications. Insert samples with false classifications into the subset and start over again by training the new set. Iterating this process works for tens of thousands of samples, since only a small amount, say 10%, of the full training set is used at any step of the training process. However, this process does not solve the problem that memory consumption is still $O(n^2)$ with respect to number of active support vectors n.

This approach shows the benefit of SVMs. All the training data that was not relevant (not belonging to the support vector set) could have been discarded before training and the same optimal result would still have been reached.

Experiments such as the face recognition application in [13] show that Sample Decomposition is a viable tool for training in practice. However, *Sequential Minimal Optimization* (SMO) algorithm in 5.7.4 has proven to be still a lot better. Sample Decomposition scaled up feasible training set sizes by a factor of 10 when it was invented, but SMO did even better.

5.7.4 Sequential Minimal Optimization

Sequential Minimal Optimization (SMO) was a major breakthrough in SVM training. The idea was originally implemented by Platt [14].

SMO splits QP problems into series of smallest possible QP subproblems. SMO has memory complexity of $O(n)$ with respect to training set size n. This allows SMO algorithm to scale to very large QP problems. However, for the reasons of practical efficiency SMO algorithms use sample cache that is proportional to something between linear and quadratic memory complexity. SMO optimization is a kind of discrete optimization algorithm. It iterates a simple process of changing two samples of the training set at a time to approach the global minimum. Time complexity of SMO is probably non-polynomial, but it can stop the iteration process at any given time and get a valid answer for the given task. Experimental results show that SMO is very useful in practice, and it converges fast into an acceptable result. The SMO implementation used in the work of this thesis can be found from [4, 5].

The brief idea of SMO is to start with QP task with all Lagrange multipliers zero. Then see which 2 samples associated with Lagrange multipliers violate the boundary constraints most. Set those 2 samples as support vectors and fix Lagrange multipliers of those two samples so that equality constraint of the QP problem is preserved and cost function decreases. Iterate this process as long as acceptable result is reached.

A great benefit of SMO algorithm is that it can be used without quadratic programming libraries. Reimplementation of QP libraries is a big task, but reimplementation of SMO is not. Of course this is not a theoretical benefit, but a practical benefit anyway. Practice is important in adoption of pattern recognition techniques. Most SVM techniques used apply SMO.

Part II

Practice

6 Empirical Tests

6.1 Dimensionality Test

6.1.1 Problem

SVM infrastructure is promising in the way that it tries to avoid hardships of high dimensional input space training data by doing training indirectly in an implicit (sometimes explicitly known) feature space. The optimization problem complexity depends on the amount of training data rather than the input space dimension. In practice effectiveness of this approach has to be verified empirically. This is tested with two high dimensional normally distributed random data with the same covariance matrices $C = \delta^2 I$. Dimensionality of data is varied to see how it affects learning ability of the SVM.

6.1.2 Consideration

SVM training can be modelled as a quadratic programming optimization problem in a space which has the dimension of the number of training data. This is different from traditional MLP networks since they are optimized in a space which has the dimension related to the number of input space dimensions. With MLP networks it's important to reduce the dimension of training space by cutting unnecessary connections in the network, that is, setting as many weights to zero as possible. A high amount of training is not an obstacle but an advantage when best possible generalization performance is targeted with MLPs. Trained SVM has the evaluation function:

$$y(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

where n is the amount of training data used.

With SVM it's also important to reduce the number of supports vectors, that is to have as many weights α_i set to zero as possible, to make the function better *behaved* (to have supposedly better generalization ability).

When input space dimension grows MLP networks increase in weight space size. With SVMs it doesn't happen directly from input space dimension, but from data that is used for training. The network complexity is linked to training data complexity. SVM tries to create a high dimensional boundary surface to the implicit feature space to separate between classes. Complexity of the boundary surface is related primarily to training data but also to input space dimension. Increasing input space dimension makes the boundary surface more complex, and hence causes more weights (more support vectors).

Testing separation of two d dimensional normally distributed classes with same covariance matrix has the advantage that the optimal solution is known analytically, and so the empirical SVM solution can be compared the optimal solution in Bayesian statistics sense. Optimal classifier actually reduces to decision function $d(\mathbf{x}) = \mathbf{a}^{\mathbf{T}}\mathbf{x} + b$ where $\mathbf{a}$ is d dimensional vector and b is a bias term. If $d(\mathbf{x}) > 0$, $\mathbf{x}$ belongs to class 1, otherwise it belongs to class 2.

6.1.3 Test Data

Assume two classes of data in input space R^d. Both classes are multinormally distributed in the d dimensional input space with following parameters.

- class 1:

  $\mu_1 = 0$ and $\Sigma = I_{d \times d}$

  Sample $\mathbf{x}_i \sim N(\mu_1, \Sigma)$

- class 2:

  $\mu_2 = \frac{2}{\sqrt{d}} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$

  Sample $\mathbf{x}_i \sim N(\mu_2, \Sigma)$

Hence class means have Euclidean distance of 2 regardless of the dimension d. From properties of normal distributions it follows that each dimension is *independently* but identically distributed because the covariance matrix is a diagonal matrix.[1]Assume random samples are drawn from probability distribution $p(\mathbf{x})$. According to Bayesian statistics vector $\mathbf{x}$ should be classified to class i if $p(class\, i|\mathbf{x}) > p(class\, j|\mathbf{x})$ for all $j \neq i$. Now

$$p(class\, i|\mathbf{x}) = \frac{p(\mathbf{x}|class\, i)p(class\, i)}{p(\mathbf{x})}$$

where $p(class\, i)$ means the probability that random sample is drawn from class i. In this case $p(class\, i) = 0.5$ for all i.

Thus sample $\mathbf{x}$ should be classified to class 1 if

$$p(\mathbf{x}|class\, 1) > p(\mathbf{x}|class\, 2)$$

Multinormal distribution gives

$$\begin{aligned}
&p(\mathbf{x}|class\, 1) = ((2\pi)^d det(\Sigma))^{-0.5} \exp(\tfrac{-1}{2}(\mathbf{x}-\mu_1)^T\Sigma^{-1}(\mathbf{x}-\mu_1)) \\
&= (2\pi)^{-d/2} \exp(\tfrac{-1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{x}) \\
&p(\mathbf{x}|class\, 2) = ((2\pi)^d det(\Sigma))^{-0.5} \exp(\tfrac{-1}{2}(\mathbf{x}-\mu_2)^T\Sigma^{-1}(\mathbf{x}-\mu_2)) \\
&= (2\pi)^{-d/2} \exp(\tfrac{-1}{2}(\mathbf{x}-\mu_2)^T(\mathbf{x}-\mu_2))
\end{aligned}$$

With little effort it results into

$$\begin{aligned}
&p(\mathbf{x}|class\, 1) > p(\mathbf{x}|class\, 2) \\
&\Leftrightarrow \mathbf{x}^{\mathbf{T}}(-\mu_2) + \tfrac{1}{2}\mu_2^T\mu_2 > 0
\end{aligned}$$

This shows that the optimal decision surface lies halfway between $\mu_1 = 0$ and μ_2 with a hyperplane normal $\mu_1 - \mu_2$. An analogy can be drawn to 1 dimensional classifying problem between two distributions:

$$N_1(0,1) \text{ and } N_2(2,1)$$

Expected error of optimal classification is

$$P_{expectederror} = p(class\, 1)\int_{R\setminus\Omega_1} p(\mathbf{x}|class\, 1)dx + p(class\, 2)\int_{R\setminus\Omega_2} p(\mathbf{x}|class\, 2)dx$$

[1]Zero correlation between two variables in multi normal distribution implies statistical independence. A diagonal covariance matrix shows that correlations between all variables are zero.

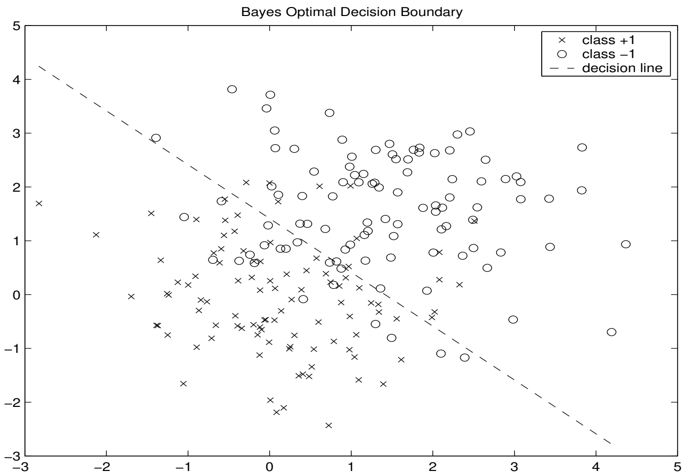

Figure 6: Optimal Bayes Classification

where Ω_i means the area where sample $\mathbf{x}$ should be classified to class i, so that it is consistent with Bayes classification. Now it's obvious that $\Omega_1 = \{x|x < 1\}$ and $\Omega_2 = \{x|x > 1\}$.

By computing appropriate cumulative distributive functions for these normal distributions the expected error is

$$P_{expectederror} = 0.5 * P_{N_1}(x \geq 1) + 0.5 * P_{N_2}(x \leq 1) = 0.1587$$

It is not possible to classify any better than this error rate. The SVM should go as close to this as possible. Figure 6 shows optimal Bayes classifier result in 2 dimensions.

6.1.4 Simulation

In simulations there were $n = 2048$ training samples with 1024 samples of each class. To verify classification accuracy there were $v = 2000$ verification samples with 1000 samples of each class.

Two types of kernels were tried:

- Radial Basis Function (RBF) kernel:

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

Table 2: Classification Results

Dimensions / Kernel	RBF	Polynomial $k = 1$	Polynomial $k = 2$
32	0.1540 (968)	-	-
64	0.1540 (947)	0.1586 (772)	0.2630 (1219)
128	0.1645 (835)	0.1580 (818)	0.2745 (1800)
256	-	0.1625 (2048)	0.3075(2036)

- Polynomial kernel with $k = 1, 2$:

  $K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^{\mathbf{T}}\mathbf{y})^k$

And the following dimensions: $d \in \{32, 64, 128, 256\}$.

For each kernel and dimension combination proper SVM parameters were found with brute force search and manual inspection. These parameters include:

- C, the tolerance parameter which describes how quadratic programming task tolerates unwanted classification of samples
- γ (gamma), the RBF kernel parameter

Software used for simulation was Cawley's SVM toolbox [4].

Table 2 shows verification errors. Values in brackets are numbers of support vectors.

6.1.5 Analysis

These numbers show that it is possible to achieve good classifying accuracy for high dimensional data when lots of support vectors can be used. Error rates should be compared to the theoretical error rate calculated in Section 6.1.3 (0.1587).

RBF kernel got close to theoretical error rate with 32, 64 and 128 dimensions. The optimization failed badly with 256 dimensions.

Polynomial kernel with $k = 1$ scaled up to 256 dimensions having the error rate close to the theoretical error rate.

Test runs showed that with dimension less than 256, a classifying error less than 17% was easily achieved. However, in some cases it took a few runs to reach that accuracy.

When more or less optimal kernel parameters were found, training results did not vary more than a few percent on different runs. The RBF and polynomial (when $k = 1$) kernel achieved more or less the same results near to optimal theoretical accuracy.

What is surprising is that the polynomial kernel with $k = 2$ did so badly although *in theory* it contains the same feature space as kernel with $k = 1$. The most likely explanation for the $k = 2$ case is that it had too much structure for the optimization algorithm to handle it properly on this data set. Similar phenomena has been observed with other classification problems as well. The SMO algorithm used to solve the classification problem is only a heuristic to solve the quadratic programming problem *better* and more *efficiently*. It doesn't guarantee success. In any case classification was successful with the kernel in case $k = 1$. Both the RBF and the case $k = 2$ kernels are more complex than the kernel in case $k = 1$, so it would be chosen as the prime candidate for a classifier. Choosing a simple kernel function seems also intuitively wise, because the dimension of the feature space is low.

With dimension of 256 or higher problems started to arise. Learning algorithm was not able to finish the task in many cases. As the dimension grows it is expected that the decision boundary becomes more complex. The learning algorithm gets increasingly harder tasks, and at some stage it will fail to deliver the desired performance. With this setup the limit was at around 256 dimensions without further tweaking of algorithm, data or parameters. With this type of data it is expected that the number of vectors that describe the decision boundary at the critical zone grows exponentially with respect to the dimension d. The critical zone of multinormal distribution is a *hypersurface* of a *hypersphere*, and hence exponentially related to the dimension d. Therefore increasing dimension makes the application rapidly infeasible.

Keeping the number of support vectors as low as possible is important when the dimension grows, otherwise the optimization may fail. Some methods are suggested here in order to have less support vectors.

1. Better selection of training data. It is possible to have support vectors on the decision boundary that are useless in the sense that they do not produce better boundary.

2. Use less training data (related to previous).

3. Find better kernel (and better parameter values).

4. Use better algorithm to solve the SVM problem. Straightforward quadratic programming with Matlab's quadprog function is not a good choice, and therefore Cawley's SVM toolbox was chosen. It is based on Platt's SMO algorithm [14], which is designed for SVM pattern classifier tasks. Quadprog in Matlab is more general purpose tool.

6.2 Ionosphere Data

6.2.1 Problem

To assess how useful SVMs are they should be compared to other Neural Network types. John Hopkin's University Ionosphere Database was used and compared to independent classification results achieved with various learning machines.

6.2.2 Background

The data investigated is from a radar system in Goose Bay, Labrador. 16 high-frequency antennas were used to observe *structure* in the ionosphere. The data set has "*good*" and "*bad*" radar returns, where "*good*" returns show evidence of structure in the ionosphere, while "*bad*" returns do not. Each observation in the data set has 34 attributes and is classified as good or bad. The goal is to predict the class of an observation from attributes using only a subset of available observations.

The data for the test was acquired from UCI Machine Learning web site [23] from the FTP archive directory [22]. The *ionosphere.names* file on the directory describes some previous experiments on the data.

6.2.3 Data

Relevant information about the data:

- a binary classification problem
- 351 samples
- input space 34 dimensional real valued data (R^{34})

As described in the ionosphere database information file training and verification data were chosen as follows:

- training data consisted of 200 samples: 50% positive and 50% negative (100 positive and 100 negative)

Table 3: Classification Accuracy Results

Kernel	Mean	Median	Best	Worst
RBF	95.5%	95.4%	98.7%	91.4%
Polynomial (2)	93.1%	93.4%	96.7%	88.1%

- verification data consisted of 151 samples: 125 positive and 26 negative

No pre-processing for the data was done to avoid twisting results in favour of SVMs. A fair comparison was the goal.

6.2.4 Simulation

Training and verification data were chosen randomly for each simulation run. Mean, median, best and the worst results are given for simulation runs.

Following types of kernels were tried:

- Radial Basis Function (RBF) kernel:

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

- Polynomial kernels (degrees 1, 2, 3, 4 and 5):

  $K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^{\mathbf{T}}\mathbf{y})^k$

The simulation was run using Cawley's SVM toolbox [4]. C^n-SVM testing was done with Matlab's QP solver.

SVM parameters for training were found with brute force techniques.

6.2.5 C-SVM Results

Classification accuracy results are shown in Table 3.

Results from the table show that RBF kernel did somewhat better than polynomial kernels. Polynomial kernel of degree 2 was chosen to represent all polynomial kernels because it was the best of all degrees. This is very different than dimensionality test

Table 4: Comparative Results

Classifier	Accuracy
Non-Linear Perceptron	92%
Nearest Neighbour	92.1%
Quinlan's C4	94%
SVM	95.4%
Backpropagation MLP	96%
Aha & Kibler: IB3	96.7%

results in Table 2 where polynomial kernel of degree 2 did very badly with the same SVM software.

Classification accuracy for positive samples was much higher than for negative samples. This is consistent with [21], and implies that negative samples have more complex patterns, because there was equal amount of negative and positive samples for training.

6.2.6 C^n-SVM Results

Similar runs were done with C^n-SVM. Results were comparable, but since the error rate was already quite low with the common case, it was not possible to improve results much. However, tuning C_i in favour of another class gave 1.6% median advantage for classifying positive class correctly compared to the common case, where all C_i are equal. The mean advantage was only 0.8% for classifying positive class correctly. When looking at both classes the overall advantage with C_i tuning was approximately 0.6%.

6.2.7 Discussion

The results were comparable to independent tests using Neural Networks with backpropagation.

Comparative results are shown in table 4.

Sigillito, Wing, Hutton and Baker [21] investigated the data by using backpropagation and the perceptron training algorithm. They achieved an average accuracy of 96% with backpropagation. Aha & Kibler used IB3 system to achieve 96.7% accuracy. These are in average better than the SVMs tried.

The backpropagation MLP had an average accuracy of 96% (precision not known) which 0.5% higher than RBF kernel average (95.5%). The worst case result for RBF kernel was rather bad looking (91.4% accuracy). This is probably a failure of SMO algorithm rather than a failure of C-SVM. However, it may have been only bad luck when choosing the training data randomly. Choosing a *bad* set of features can be disastrous for any classifier.

The mild success of using C^n-SVM was a positive surprise. Further experiments with C^n-SVM are needed. There are lots of applications where the accuracy of one class is more important than the accuracy of other classes.

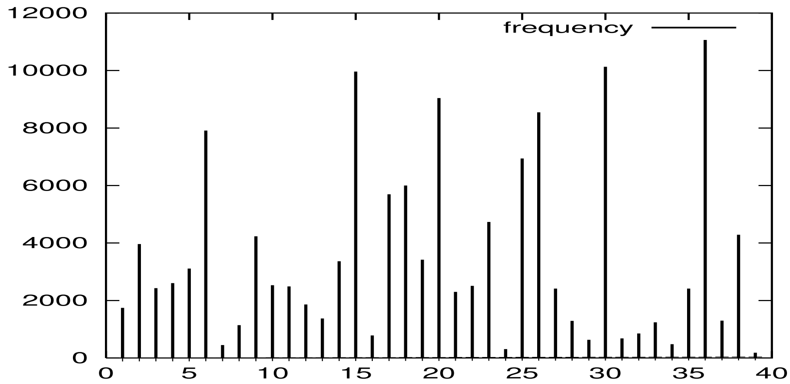

Figure 7: Comparative Frequencies of 39 Classes

6.3 Phoneme Recognition

6.3.1 Problem

Speech Recognition presents great challenges for the machine learning community. It has been widely studied with many approaches. Nowadays practical solutions are appearing on the general market, and thus there are commercial interests to pursue research on this area.

The problem here is induced by an independent party providing experimental data on a real speech recognition problem. The objective in this thesis was to verify and improve on their results.

6.3.2 Data

The data given contains 39 classes. Each class has a number of 4 dimensional feature vectors which are labeled as positive or negative samples (values $+1$ and -1 respectively). The binary classification machine is given a feature vector and a class number i ($i = 1, 2, ..., 39$). The machine should determine whether or not the given vector is a *positive* sample of the class i based on the empirical data.

Figure 7 presents amount of samples in each class.

For some classes there were less than 500 samples for training and verification. These cases were handled differently.

Table 5: Kernels used for Phoneme Recognition

Polynomial kernel	$K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T\mathbf{y})^d$
RBF kernel	$K(\mathbf{x}, \mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}$

Given a sample distribution for any of the 39 classes, there do not seem to be clear regions of negative or positive samples, but negative and positive samples are mixed together in the same region of space, at least in the euclidian sense of geometry. Figures 8, 9 and 10 show class 1 distribution of negative (marked with $\times$) and positive (marked with $+$) samples projected from six different dimension pairs. Only 400 hundred samples were chosen from the class 1 data. More samples would make the figures less informative.

All six different projections show great overlapping in negative and positive sample regions. This makes accurate classifying hard.

6.3.3 Simulation

An SVM was trained for each class to recognize positive samples. A part of the data for each class was used to train the associated SVM and the rest of the data was used to verify accuracy of the binary classifier.

The goal was to classify each class as well as possible with less than 130 *support vectors*. And preferably estimate how many support vectors would be needed to classify the data as well as possible within the limits of the data.

Table 5 shows kernels chosen for SVM training. Polynomial and RBF kernel were chosen based on tests with many different kernels. Other kernels, like the linear kernel, were tried but they did not result in any improvement over polynomial and RBF kernels.

Best estimated kernel parameters were found by brute force techniques. In the case of polynomial kernels the polynomial degree was in the range of $[1, 5]$ and C in the

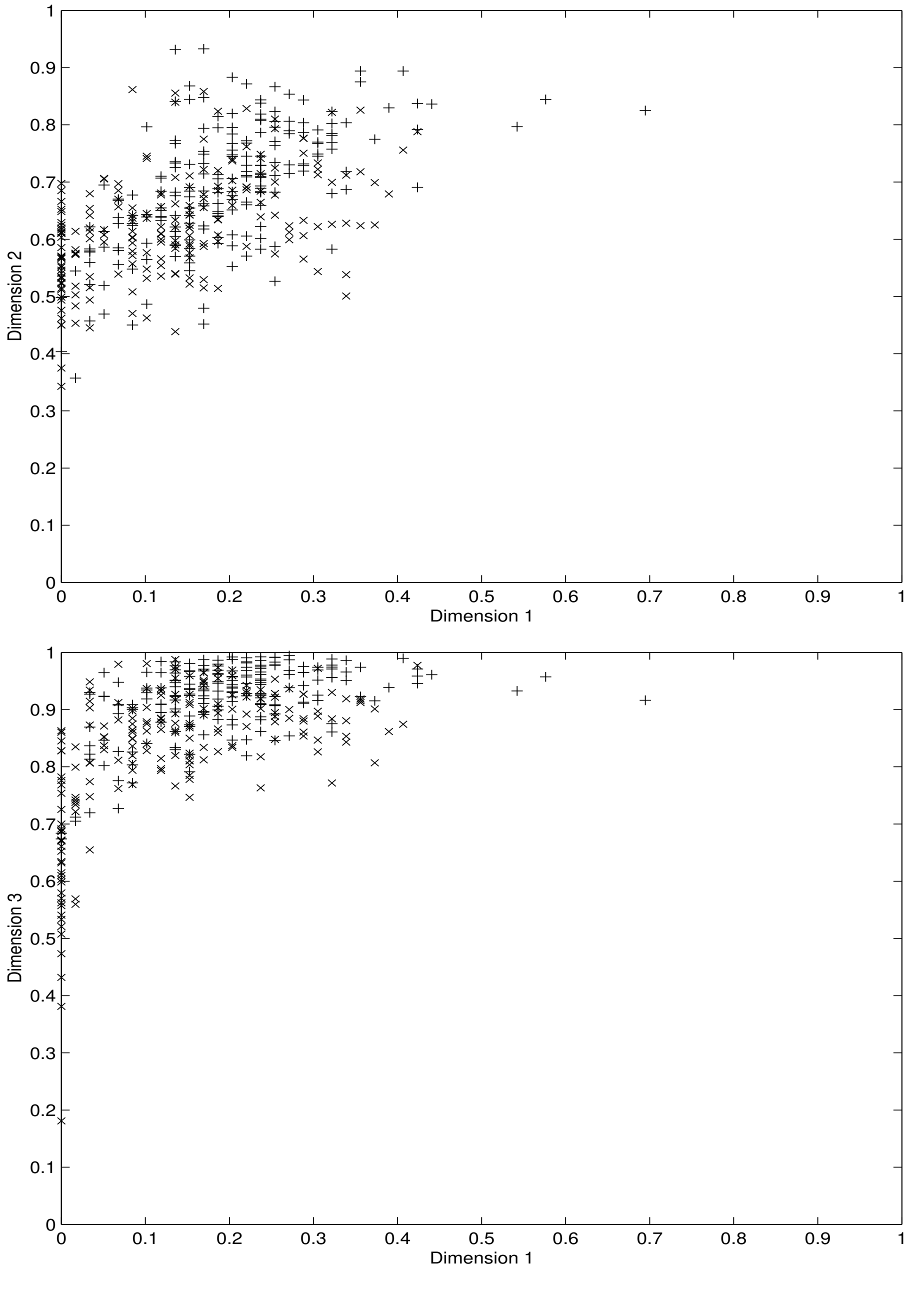

Figure 8: Class 1 distribution from 2 different projections: 1 vs. 2 and 1 vs. 3. Negative samples are marked with $\times$, and positive samples are marked with $+$.

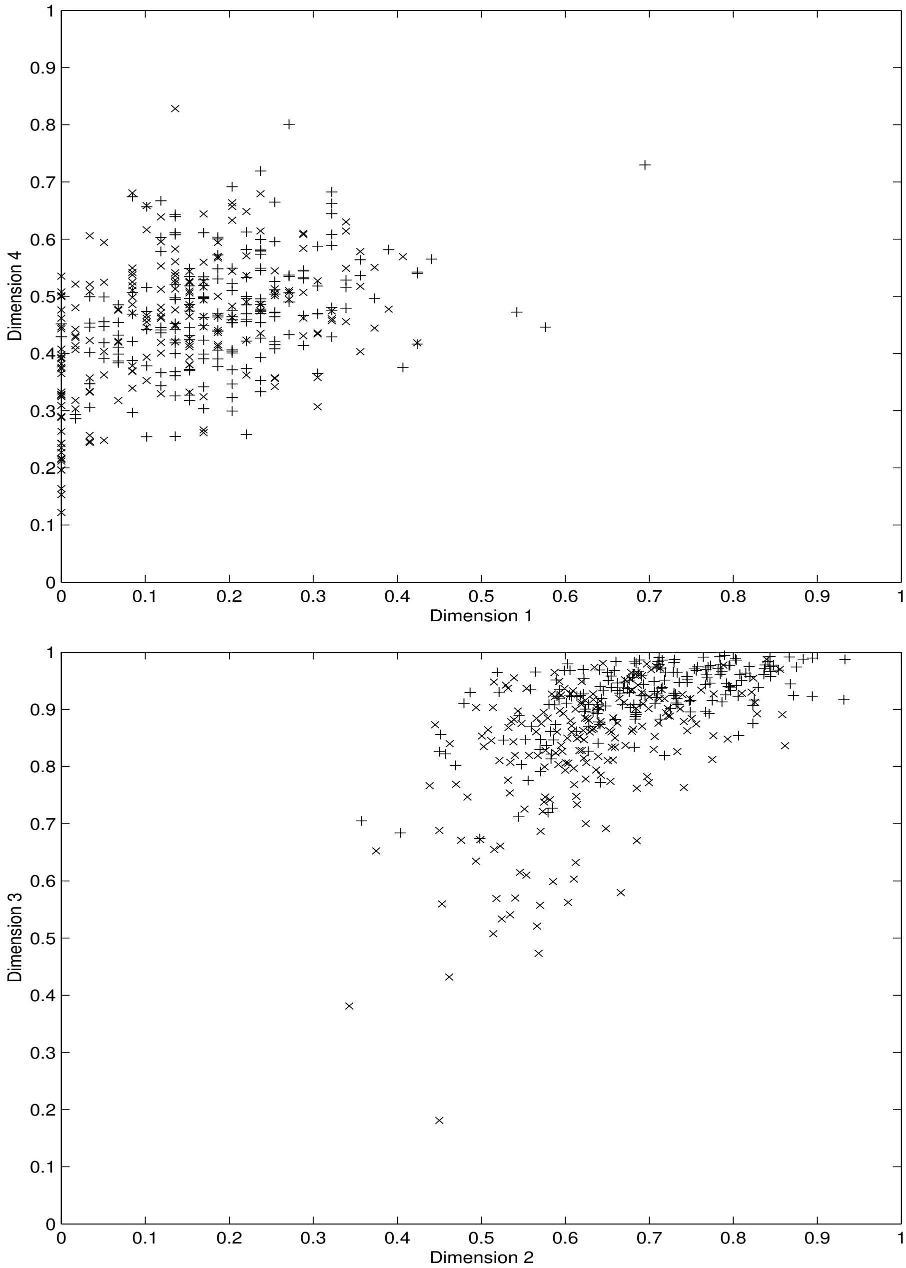

Figure 9: Class 1 distribution from 2 different projections: 1 vs. 4 and 2 vs. 3. Negative samples are marked with ×, and positive samples are marked with +.

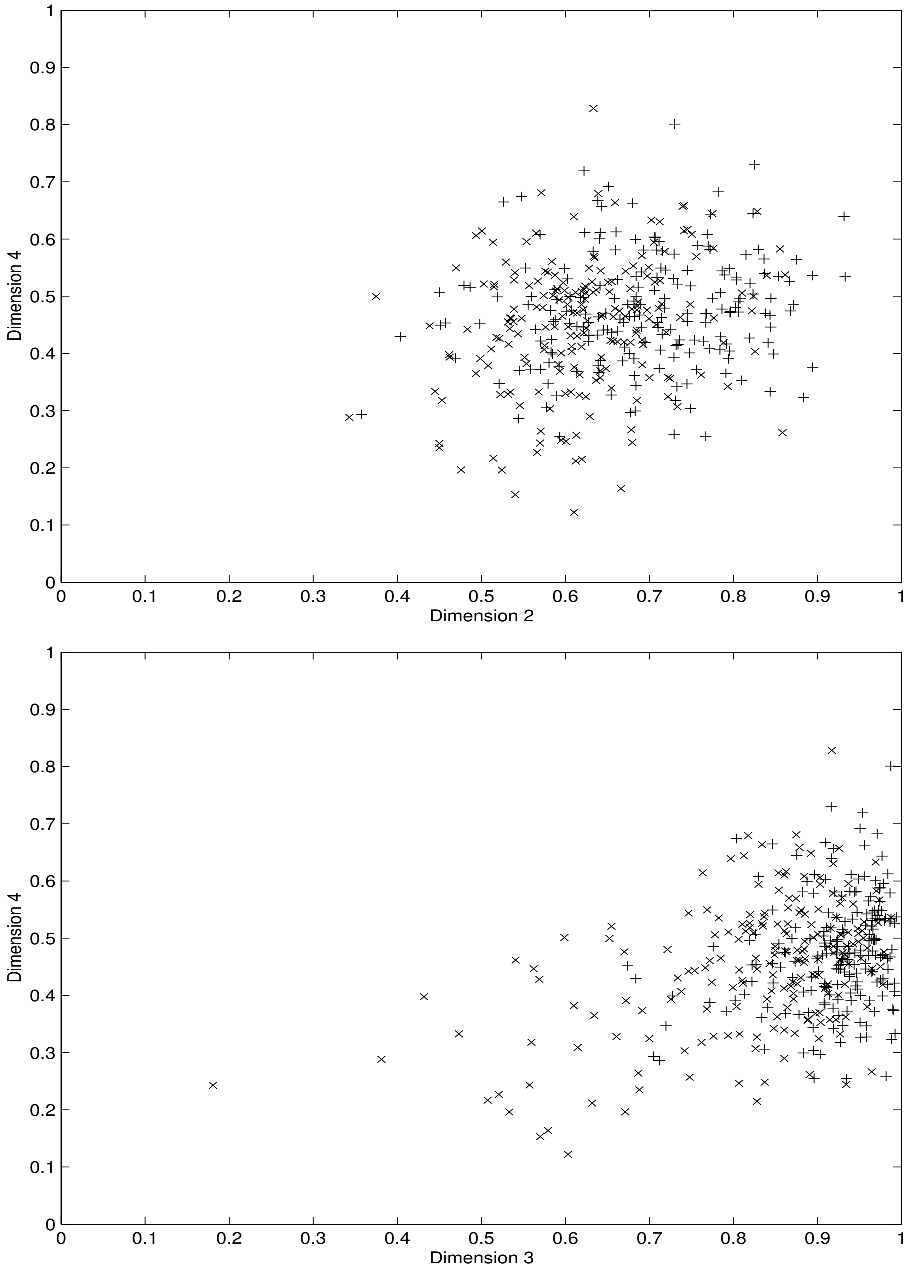

Figure 10: Class 1 distribution from 2 different projections: 2 vs. 4 and 3 vs. 4. Negative samples are marked with $\times$, and positive samples are marked with $+$.

range of $[0.01, 20]$. For RBF kernels γ parameter was in the range of $[0.1, 1000]$ and C in the range of $[0.1, 20]$. These ranges were determined by empirical testing.

Requirement was to have less than 130 support vectors for all classes. This requirement is tricky, because the number of support vectors resulting from training process is the number of active boundary constraints placed on the *quadratic programming problem*. Generally speaking it is very hard to control the upper bound limit for the amount of active boundary constraints of the optimization problem.

However, a new method called ν-support vector classification [16] allows to control the number of support vectors resulting from training. It was tried, but it did not give significant practical advantage over traditional support vector classification in this case.

The easy and bad solution to control the number of support vectors is to set the number of training samples less than or equal to the given upper bound limit for support vectors.

Our solution was to carry out optimization many times, each time with a different number of training samples. Given an upper limit for the number of support vectors, the best case in the series of training was chosen.

6.3.4 Results

Polynomial Kernel

These results reflect training of all classes, except classes 24 and 39, which had too few training samples.

Figure 11 shows how classifying error rate decreases when upper limit for support vectors is increased. Various classes were chosen for this figure based on result interest. Class 1 was told be especially important, while class 4 is hard. Other classes try to represent the task as well as possible.

Error rate of 1 (all wrong) should be interpreted so that for that number of support vectors regression was not possible because of the limit of support vectors. For example in class 4 more than 140 support vectors was needed to make any sensible results.

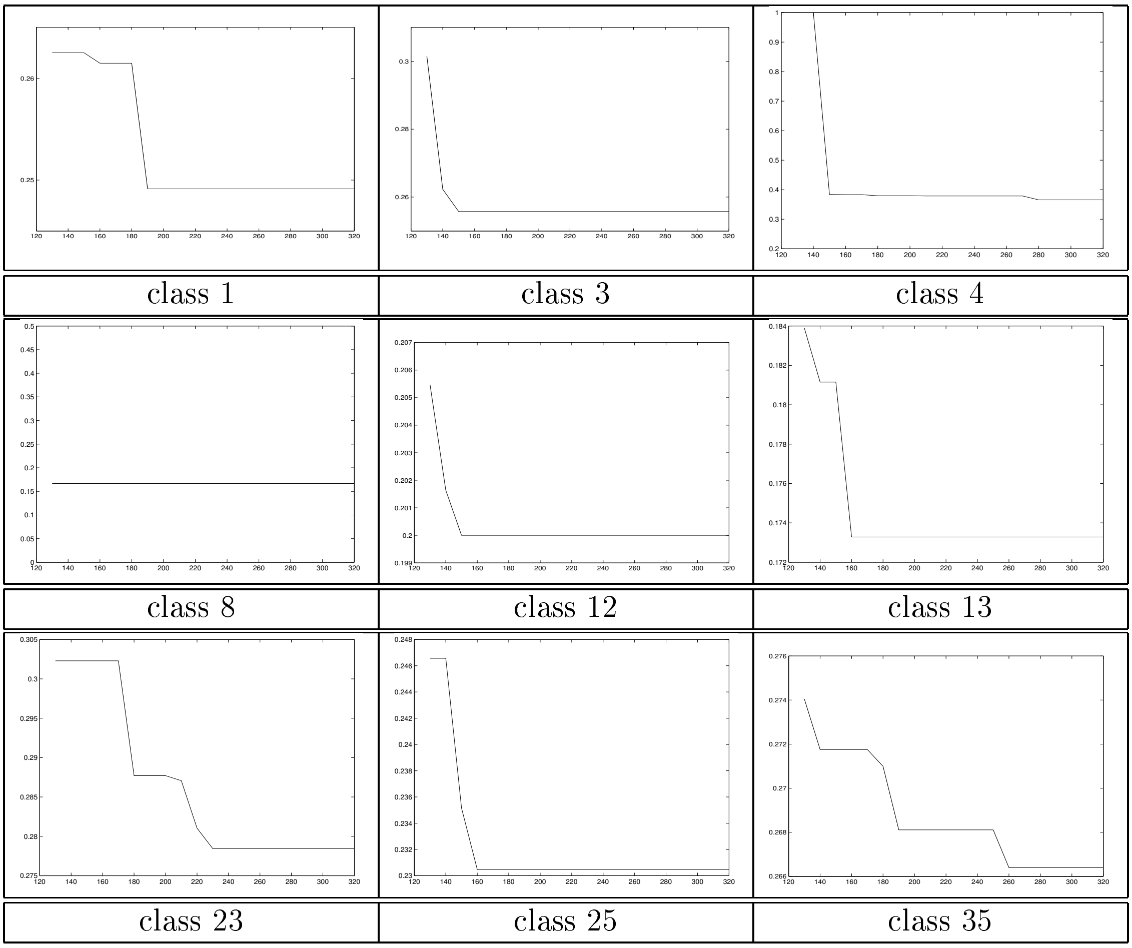

Figure 11: Polynomial error rate on various classes

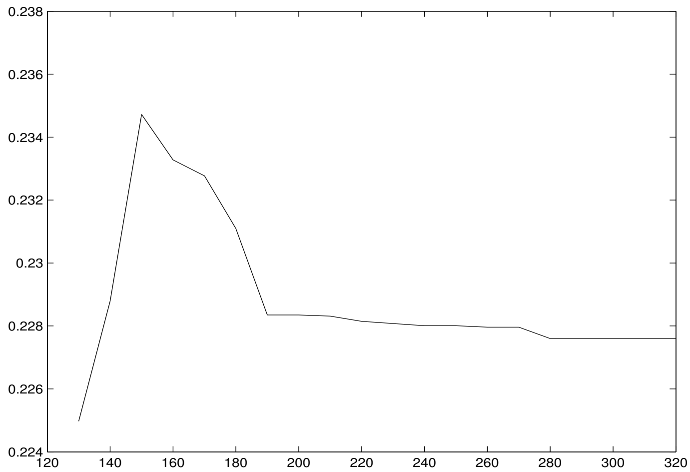

Figure 12: Polynomial average error of all classes with respect to support vector upper limit

Table 6: Summary of polynomial error averages with support vector limits

n. classes	32	35	37	37
error / n. svs	0.2250 / 130	0.2288 / 140	0.2284 / 190	0.2276 / 320

Figure 12 shows average error of all classes with respect to the support vector upper limit. No more than the limit amount of support vectors were allowed for classifiers.

Notice the increase of error rate when support vectors are increased. This is the result of new classes taken into account. The average was counted over all classes for a given number of support vectors. However, those classes were discarded for the average which couldn't be trained for the given number of support vectors. Some classes were much harder than others and couldn't be trained for less than 190 support vectors. When there are more than 190 support vectors their effect can be seen in the error rate average. The effect is increase in error rate because they are hard.

Table 6 summarizes classifying error. *n.classes* refers to the number of classes affecting the error, and *n.svs* is the maximum number of support vectors for the given error. Two classes (24 and 39) were left out because there was insufficient training data for them.

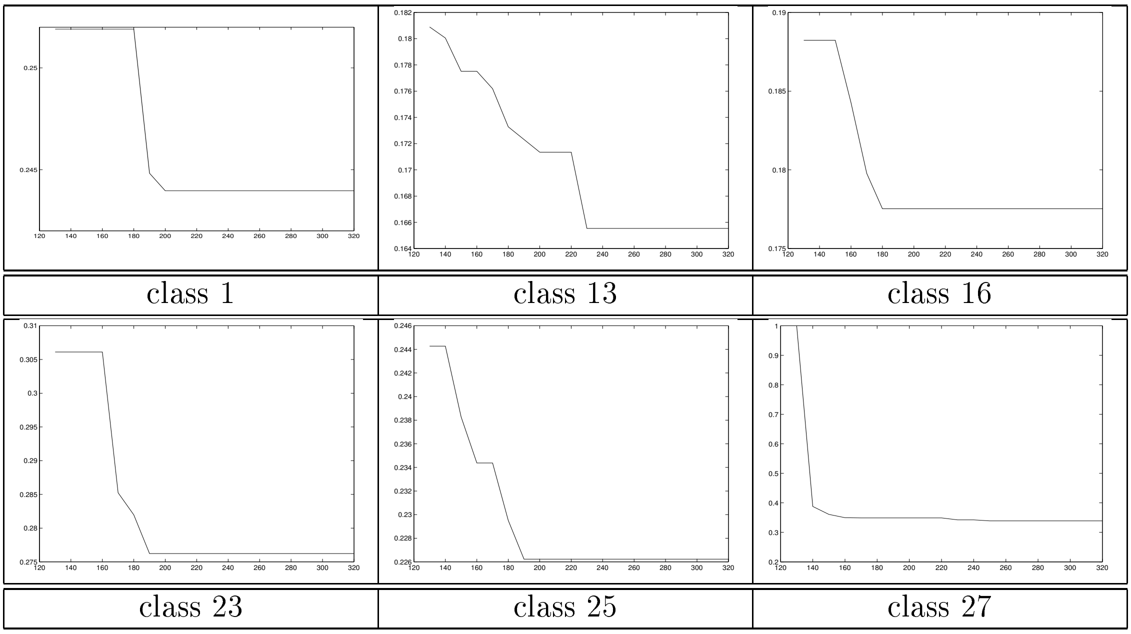

Figure 13: RBF error rate of various classes

RBF kernel

Figure 13 shows how classifying error rate decreases when upper limit for support vectors is increased. This figure is similar to the polynomial kernel case, except that different classes were interesting.

Figure 14 shows average error for all classes plotted against upper limit of support vectors.

As with polynomial kernel, the average error first increases with respect to support vectors, since new classes are taken into account. Finally, average error reaches 0.2267 where it is decreasing very slowly.

Table 7 summarizes classifying error. *n.classes* refers to number of classes affecting the error, and *n.svs* is the maximum number of support vectors for the given error. Two classes (24 and 39) were left out because there was insufficient training data for them.

6.3.5 Discussion

Kernels

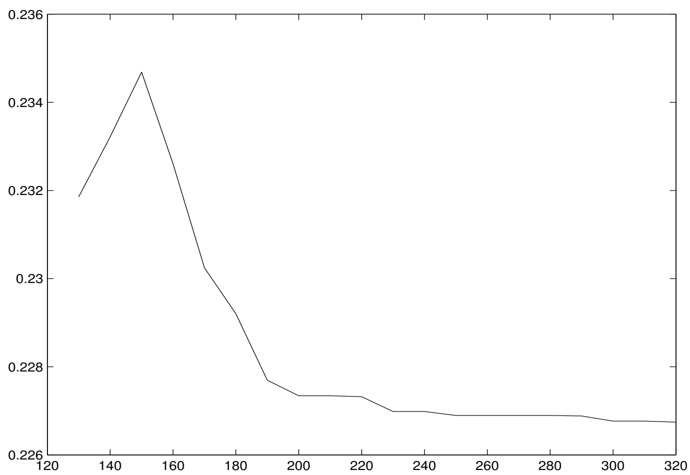

Figure 14: RBF average error of all classes with respect to support vector upper limit

Table 7: Summary on RBF error averages with support vector limits

n. classes	35	36	37	37
error / n. svs	0.2319 / 130	0.2332 / 140	0.2277 / 190	0.2267 / 320

Many kernels were tried to train the SVM, but no kernel was significantly better than the others. Table 8 shows kernels tried with the data.

Since there was no significant advantage on any kernel, polynomial and RBF kernels were chosen for simulations, because they are common and intuitive. What is surprising is that feature spaces of these kernels are very different from each other, but results were close to each other. For example, polynomial kernel has finite dimensional feature space whereas RBF kernel has infinite dimensional. For each kernel function the number of support vectors was also close to that of the others. It was

Table 8: Kernels functions tested with phoneme recognition

Kernel name	Kernel function
Linear	$K(\mathbf{x},\mathbf{y}) = \mathbf{x}^T\mathbf{y}$
Polynomial	$K(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}^T\mathbf{y})^d$
RBF	$K(\mathbf{x},\mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}$
Hyperbolic Sine	$K(\mathbf{x},\mathbf{y}) = asinh(c\mathbf{x}^T\mathbf{y}+d)$

also observed that support vectors were often same for different kernels. This is probably the consequence of feature map from input space to feature space being a continuous function.

To improve the classifying accuracy different kernels can be used for different classes. Based on the experience gained it can give 1% overall increase in performance on this data set.

Complexity

Previously presented figures for class 1 show great overlap for positive and negative samples. This is true for all the classes. On the other hand, results show that increasing the number of support vectors beyond certain point doesn't give better generalization ability. For example, see the figure of class 16 with RBF kernel. The error rate does not drop significantly after 180 support vectors, and neither does average error rate over all classes. Actually, no class does more than a few percents better with 140 support vectors compared to 320 support vectors. The problem with SVM optimizing is to select the right vectors from the training set. If the training set is large it is difficult. Running many SVM many times on a big set can and will give better results depending on the chance that *right* training samples are chosen as support vectors. This probably means that without clever pre-processing it's not possible to classify better. But it can also mean that the training machines used were not as good as they could be.

Optimization Algorithms

2 different SMO training machines [4, 5] were tried to optimize SVMs, and they gave similar results. Matlab's general purpose quadratic programming function did not give results nearly as good as SMO training did. SMO training was faster and consumed less memory. Minos quadratic programming solver was not tried since its license is too restrictive for continuing academic use, and it would be expensive. However, many publications have reported using it.

Pre-processing

Principal Component Analysis (PCA) pre-processing did not show any improvement on these Gaussian-like distributions.

Also, subpartitioning of space into multiple regions (in this case BSP or *binary space partitioning*) along principal axes of class distributions, and training different SVM for each region did not improve results. This would also have increased total number of support vectors which is against the goal.

Simulations show evidence that more features (dimensions) are needed to improve results significantly. This is based on the observation that error rate does not improve when the number of support vectors grows, and on the observation that positive and negative samples are highly overlapped in the same region of space.

Usually classes distributed like in Figures 8, 9 and 10 can't be classified better by using clever pre-processing methods, and good solutions existing are unknown. Gaussian like distributions are uncertain, and they will remain so. Measuring features that are well separated is essential in these cases. Hence no good method for pre-processing the data set was found.

Generally speech recognition pattern classification tends to be hard since the measured data is highly *context dependent.*

Time Complexity

For embedded system applications polynomial kernel classification gives comparable results to RBF kernel, but is much cheaper in computational resources. The exponential function for RBF kernel is quite costly. Polynomial kernel is also trivial in implementation, even on machine language level.

Part III

Discussion and Conclusions

7 Discussion

7.1 Higher Dimensions

SVM system is definitely interesting because the training in itself is invariant of the input space dimension. This makes it a great tool for challenging pattern recognition tasks in very high dimensional input spaces. For example, SVM has had applications with gene sequences [3]. A quote from the publication:

> Our experiments show the benefits of classifying genes using support vector machines trained on DNA microarray expression data. We begin with comparison of SVMs versus four non-SVM methods and show that SVMs provide superior performance.
>
> ...
>
> One significant benefit offered by SVM is scalability. The number of support vectors selected by the SVM is usually small, even for very large training sets, and the resulting SVM is consequently an efficient classifier. In this work, training a radial basis SVM using two-thirds of the data set (1645 examples) takes an average of 89.6 CPU seconds on a DEC Alpha 4100 workstation running at 466 MHz. This resulting machine contains only 216 support vectors on average. Thus, classifying a new gene requires comparisons with only approximately 13.1% of the training set.

SVM makes it possible to classify in very high dimensional feature spaces, since only dot-products of the feature space (that can be infinite dimensional) are used. This is convenient for image recognition, for example. One sample can be one picture.

7.2 Pattern Recognition

Support Vector Machines have been successfully applied to various pattern recognition tasks. Some widely discussed applications have been recognition of hand-written digits [19], text categorization [10], gene sequence analyzing [3] and face detection (spotting human faces from a picture) [13].

Schölkopf [17] and others [18] showed that *support vectors* of a data set represent a good subset of information based on which recognition can be successful. This implies that *support vectors* characterize the essential information in a given data set. Furthermore usually only a small subset of training data is support vectors.

Invariance on pattern data can be applied by directly transforming *support vectors* instead of transforming all training data, as shown in [20]. As there is significantly less support vectors as there is other data, this may help adding invariance with less effort.

These and other experiments make SVMs an interesting research and application area.

7.3 Radial Basis Function Neural Networks

Using RBF kernel with SVM provides a way to achieve similar goals as RBF Neural Networks. SVM can be applied to do full classification, or it can be used only to choose data centers for the RBF network. SVM networks have been compared to RBF networks in [19].

7.4 Support Vector Transformation for Image Compression

During the work on this thesis a new way of using SVMs was briefly thought and experimented. One potential application for SVMs would be image compression using SVM function approximation.

Following pseudo-algorithm describes the process:

1. Convert the image to small blocks.

2. Transform the blocks into Lagrange-multiplier space (Support Vector Transformation).

3. Compress each block with some lossless compression algorithm.

In phase 1 image is divided into fixed size blocks, 16×16 for example.

In phase 2 each block is approximated with an SVM. This is called Support Vector Transformation, or SVT. This is simply thinking an image block as a function. Pixel positions in R^2 are mapped with the function into pixel brightness values. Support vector function is used to approximate image block function.

Lagrange multipliers are the result from the quadratic programming task on support vector function approximation. Since the grid can be set constant for each transformed image block, the only essential information needed to be stored for each block is all its Lagrange multipliers. These values are said to be in Lagrange-multiplier space.

In phase 3 Lagrange multipliers are compressed with some lossless compression algorithm. If only a small subset of pixels in blocks are support vectors, meaning that they have non-zero Lagrange-multipliers, the compression will result in a small size.

Also, SVM function approximation gives the ability to control the maximum error of the approximative compressed image. For example for 256 colour (8-bit) grayscale image and 16-bit color display (5 bits for red, 6 bits for green and 5 bits for blue per pixel), error tolerance could be set to $1/32$. This would certainly lose information from the picture, but for the given display device it wouldn't make a difference. Tuning error tolerance allows in theory to control compression ratio for the image. Also, the original image can be rounded to suitable accuracy before SVT.

Although this method was tried for lossy image compression there are no results available at the moment due to optimization problems.

8 Conclusions

SVMs have been used successfully in many applications. They have been especially useful in high dimensional pattern recognition tasks, such as gene sequence analyzing. Hard problems that MLPs have not been able to resolve may be retried with SVMs with hope for at least slight improvement in accuracy. However, SVM is still just another neural network. The most important factor in pattern recognition is always choosing good features by pre-processing.

The ionosphere data set test showed that it is easy to have competitive results with SVMs compared to MLPs. C^n-SVM was tested on the ionosphere data, which gave a slight increase in accuracy. The improvement was not significant, but showed a way to tune an SVM to fit the application.

Phoneme recognition results showed given data was extremely hard to classify correctly. Huge overlap in class distributions allowed only 22% percent average accuracy in classification. Some classes were significantly harder than others. It was suggested that new features are needed in order to increase classifying accuracy significantly. Some heuristics were given to improve training with current features, but no significant improvement can be expected.

Looking at SVMs from practical and theoretical perspective, it is possible that in the future SVMs will be as important learning machines as MLPs are presently. The author believes that in-built SRM methods, such as that featured in SVM, will overcome MLPs eventually.

Part IV

Related Material

References

[1] Bazaraa, M. S. & Shetty, C. M. Nonlinear programming: Theory and Algorithms. John Wiley & Sons, 1979.

[2] Boser, B. E. & Guyon, I. M. & Vapnik V. N. A Training Algorithm for Optimal Margin Classifiers. Fifth Annual Workshop on Computational Learning Theory. ACM Press, Pittsburgh, 1992.

[3] Brown, M. P. S. & Grundy, W. N. & Lin, D. & Cristianini, N. & Sugnet, C. & Ares, M. Jr. & Haussler, D. Support Vector Machine Classification of Microarray Gene Expression Data. UCSC-CRL 99-09, Department of Computer Science, University of California Santa Cruz, Santa Cruz, CA, 1999.

[4] Cawley, G. MATLAB Support Vector Machine Toolbox. `http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox`

[5] Chang, C. & Lin, C. LIBSVM - A Library for Support Vector Machines, 2002. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[6] Cortes, C. & Vapnik, V. N. Support Vector Networks. Machine Learning, 1995. Volume 20, Number 3. pp. 273-297.

[7] Courant, R. & Hilbert, D. Methods for Mathematical Physics, J. Wiley, New York, 1953.

[8] Cristianini, N. & Shawe-Taylor, J. An Introduction to Support Vector Machines (and other kernel-based learning methods), 1st edition. Cambridge University Press, 2000.

[9] Haykin, S. Neural Networks - A Comprehensive Foundation, 2nd edition. Prentice Hall, 1998.

[10] Joachims, T. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Proceedings of ECML-98, 10th European Conference on Machine Learning, 1997.

[11] Kaleva, O. Matemaattinen Optimointi 1. Tampere University of Technology course material, 2001.

[12] Lin, C. Stopping Criteria of Decomposition Methods for Support Vector Machines: A Theoretical Justification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2001.

[13] Osuna, E. & Freund, R. & Girosi, F. Support Vector Training And Applications. AIM-1602, 1997.

[14] Platt, J. C. Fast training of support vector machines using sequential minimal optimization. Advances in Kernel Methods - Support Vector Learning. MIT Press, 1999. Chapter 12. pp. 185-208.

[15] Rudin, W. Principles of Mathematical Analysis. McGraw-Hill Book Company, 1976.

[16] Schölkopf, B. & Smola, A. & Williamson, R. C. & Bartlett, P. L. New Support Vector Algorithms. Neural Computation, 2000. Volume 12. pp. 1207-1245.

[17] Schölkopf, B. Support Vector Learning. Doctoral Dissertation. Informatik der Technischen Universität Berlin, 1997.

[18] Schölkopf, B. & Burges, C. & Vapnik, V. N. Extracting Support Data for a Given Task. Proceedings, First International Conference on Knowledge Discovery & Data Mining. AAAI Press, Manlo Park, CA, 1995. pp. 252-257.

[19] Schölkopf, B. & Sung, K. & Burges, C. & Girosi, F. & Niyogi, P. & Poggio, T. & Vapnik, V. N. Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers. IEEE Transactions on Signal Processing, 1997. Volume 45, Number 11. pp. 2758-2765.

[20] Schölkopf, B. & Burges, C. & Vapnik, V. N. Incorporating Invariances in Support Vector Learning Machines. Artificial Neural Networks, ICANN'96, Springer Lecture Notes in Computer Science, 1996. Volume 1112. pp. 47-52.

[21] Sigillito, V. G. & Wing, S. P. & Hutton, L. V. & Baker, K. B. Classification of Radar Returns from the Ionosphere Using Neural Networks. Johns Hopkins APL Technical Digest, 1989. Volume 10. pp. 262-266.

[22] The Ionosphere Data, `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/ionosphere/`

[23] UCI Machine Learning web site: `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[24] Vapnik, V. N. & Golowich, S. E. & Smola, A. Support Vector Method for Function Approximation, Regression Estimation and Signal Processing, 1996.

[25] Vapnik, V. N. The Nature of Statistical Learning Theory, 2nd edition. Springer Verlag, 1999.