Support Vector Machines:Training and Applications

by

Edgar E. Osuna

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998

Author .....
Department of Electrical Engineering and Computer Science
May 22, 1998

Certified by .....
Federico Girosi
Principal Research Scientist
Thesis Supervisor

Certified by .....
Tomaso A. Poggio
Uncas and Helen Whitaker Professor
Thesis Supervisor

Accepted by .....
Robert M. Freund
Co-director, Operations Research Center

Support Vector Machines:Training and Applications

by

Edgar E. Osuna

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 1998, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

The Support Vector Machine (SVM) is a new and very promising classification and function approximation technique developed by Vapnik and his group at AT&T Bell Labs [16] [27] [107], and can be seen as an approximate implementation of the Structural Risk Minimization (SRM) induction priciple [106].

Since Structural Risk Minimization is an inductive principle that aims at minimizing a bound on the generalization error of a model, rather than minimizing the Mean Square Error over the data set, (as Empirical Risk Minimization methods do) training a SVM to obtain the maximum margin classifier requires a different objective function. This objective function is optimized by solving a large-scale Quadratic Programming Problem with linear and box constraints. The problem is considered non-trivial, since the quadratic matrix is completely dense and its size is square in the number of data points. Therefore, training problems arising in some real applications with large data sets are impossible to load in memory, and cannot be solved using standard nonlinear constrained optimization algorithms.

We present a decomposition algorithm and a computer implementation that can be used to train SVM's over large data sets. The main idea behind the decomposition is the iterative solution of sub-problems and the evaluation of optimality conditions which are used both to detect improving-cost pivots, and also as a stopping criteria.

We also approach other challenges related to the use of the technique, like the reduction of its run-time complexity, and the optimization of parameters that were originally left to the user. The combination of the solutions for these challenges yield an alternative mathematical formulation for SVMs that is characterized by a nonlinear program, and that better approximates the Structural Risk Minimization principle.

As applications of SVM's, we present results in face and people detection systems, as well as time series analysis.

Thesis Supervisor: Federico Girosi
Title: Principal Research Scientist

Thesis Supervisor: Tomaso A. Poggio
Title: Uncas and Helen Whitaker Professor

Acknowledgments

It seems as if writing these few lines will be the next to last thing I will do as a student at MIT (I still need to spend some time with the printer and then rush to the ORC in time for the deadline). Some how it is harder than I expected. But there are good reasons for that. These have been four extraordinary years. During these period not only did I finish this MIT adventure, but I also became a husband and a father.

There are many names that come to me, and many more that I am sure I will forget. So I will relax the non-aggravation constraint and ask for your forgiveness if you do not find your name explicitly mentioned here: it probably should be if you have gotten this far.

I am deeply indebted to my advisors Federico and Tomi. If I am here today doing Support Vector Machines it is thanks to them being a constant source of wisdom and encouragement during the three years as their graduate student. It has been a privilege to work them. It is an honor to become their colleague.

I'm also grateful to my other committee member Rob Freund, not only for his valuable comments which helped improve this work, but also for going beyond his traditional area of expertise.

It would be hard to accomplish this work without the help of the people that keep CBCL and the ORC running: Marypat, Sarah, Laura and Paulette.

The support and friendship of my fellow students at CBCL have made the difference between a fun or a dull day of research. All of you guys, Sayan, Constantine, Massimiliano, Theos, Rif, Nicholas, John, Tony, Vinay and Max have really made my life away from home a little easier.

I want to thank my wife Grisell. With her love, I have never felt alone. You were always there for me through the ups and downs.

Finally, I want to thank my family, especially my parents and sister. Without their endless support and love for me, I would have never achieved my current position.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years Learning Systems have emerged as a practical technology, with successful applications in many fields. The majority of these applications are concerned with problems in pattern recognition and function approximation, and make use of feed-forward network architectures such as multi-layer perceptrons and radial basis function networks. Also in recent years, it has become widely acknowledged that techniques that rely on well-founded theoretical concepts are preferred to others that may involve *ad hoc* procedures, since the former are generally more reliable, predictable and easier to apply.

This thesis studies a new pattern recognition and function approximation technique called a Support Vector Machine. We consider this novel technique interesting not only because of its deep theoretical framework in statistical learning, but also because its application requires the solution of a large-scale linearly constrained quadratic programming problem. We also have found that certain improvements on the technique can be obtained through the formulation and solution of additional nonlinear optimization problems.

In this introduction we present an overview of learning systems, a primer on Support Vector Machines (SVM), a list of challenges we encountered when applying this technique, a brief literature review, the thesis outline, and a list of the main contributions of this work.

1.1 Overview of Learning Systems and Concepts

The main purpose of this section is to offer an overview on Learning Systems, and more specifically, a paradigm within it called *Examples-based Learning* or *Supervised Learning.* We will also describe the function approximation and pattern classification problem within this paradigm of *learning by examples.*

1.1.1 What is a Learning System?

A *learning system* is a computer program that makes decisions based on the accumulated knowledge contained in successfully solved or labeled cases. Unlike an *expert system*, which solves problems using a computer model of human reasoning, a pure learning system can use many different techniques for exploiting available information and the computational power of a computer, regardless of their relation to human cognitive processes. These techniques include many highly mathematical methods, as well as others that simply search systematically over large number of possibilities.

Within this context, we are more interested in learning systems that computationally extract their decision criteria from sets of correctly labeled *examples* without significant human intervention.

1.1.2 The Paradigm of Example-based Learning

Example-based learning is an area of research that has captivated the interest of many technologists, scientists and mathematicians over the last decade. The field deals with models of memory retrieval and techniques for empirically discovering complex relationships in sparse data. The *learning* approach to software development is fundamentally different from the traditional *programmed computing* approach. In *example-based learning*, a programmer *trains* a system to perform an information processing task by providing the system with input features measurements and corresponding output values of the task. This pair of input-output values is called an *example.* The system *learns* the task from the input-output examples the programmer provides by adaptively modifying its state to implement an appropriate mapping.

Example-based learning techniques have been used in many areas ranging from stock market prediction and signal processing to motor control in robots. It is this idea of *training* machines instead of having to actually program them that makes learning especially appealing in areas where general algorithms for dealing with problems are still relatively unavailable.

1.1.3 Example-based Learning and Function Approximation

Learning from examples is a common supervised-learning paradigm that hypothesizes a target concept given a stream of input-output examples that describes the concept. In the domain of real numbers, the task of learning an input-output *concept* from a set of examples is essentially equivalent to approximating a multivariate function that (1) maps input examples onto their respective output values, and (2) reasonably interpolates between output values at regions of the input space where no examples are available [80].

The learning problem can thus be more formally stated as follows: Let $\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be a set of n data points sampled from an unknown multivariate function $f(\mathbf{x})$, possibly in the presence of noise. The task is to recover the function $f(\mathbf{x})$, or at least a reasonable estimate of it, by means of an *approximation function* from a function class $F(\mathbf{w}, \mathbf{x})$, parameterized by the vector $\mathbf{w}$. For a fixed function class F, the problem is then to find the set of parameters $\mathbf{w}$ that best approximates $f(\mathbf{x})$ based on information from the set of data points $\mathcal{D}$.

Clearly, how well one can approximate an unknown target function $f(\mathbf{x})$ depends heavily on the following two factors:

1. **The function class $F(\mathbf{w}, \mathbf{x})$.** Needless to say, it is very important to choose an approximation function class F that can represent the unknown target function f sufficiently well. There would be little point in trying to recover $f(\mathbf{x})$ if the chosen approximation function class $F(\mathbf{w}, \mathbf{x})$ only gives a very poor representation of $f(\mathbf{x})$ even with optimal parameter values. Some popular network-based

function classes with universal approximation properties include *multilayer perceptron* nets [79] [89] and *radial basis function* nets [64] [80]. Other classical methods like splines can also be used. A closely related issue is the *complexity* of the approximation function class, which is traditionally measured by the number of free parameters in $\mathbf{w}$. A more complex function class usually has a better chance of approximating an unknown target function well, but it also requires a larger number of data samples to arrive at a reasonable approximation for predicting unseen data (see [73] for the case of *radial basis function* nets). In Sections 2.1 and 2.2 we will discuss an alternative approach to measuring and controlling this complexity.

2. **The data sample $\mathcal{D}$.** The accuracy of an approximation also depends on the quality of data in $\mathcal{D}$, i.e., the quality of available information about the unknown target function f. A larger data sample charts the output value of f at more input locations, and hence conveys more information about f. It is well known in learning theory that as the complexity of the approximation function class F increases, one must also increase the number of data samples in $\mathcal{D}$ to avoid large approximation errors due to overfitting. The distribution of data samples is another critical aspect of $\mathcal{D}$ that has often been overlooked in example-based learning. Ideally, we want a well-distributed data sample that provides a balanced representation of f. If the learning objective is to closely approximate the unknown target function f at all input locations, then there is little point in collecting a lot of data at one input location while ignoring other input locations. Similarly, there is no point in collecting a lot of data at input locations where the chosen function class F is slowly changing. In the particular case of Support Vector Machines for pattern classification, we will show in section 2.3.2 how points located in the *boundary* between the classes become the keystones in defining the decision surface $F(\mathbf{w}, \mathbf{x})$. A similar geometric interpretation will be given in section 2.4.3 for Support Vector Machines for function approximation.

1.1.4 Pattern Classification as a Function Approximation Problem

One can approach *pattern classification* as learning an approximation function for predicting the class identities of input patterns. Consider a two-way classification task that operates on input domain $\mathcal{X}$ with output classes $\mathcal{W} = \{w_0, w_1\}$. For each input pattern $x \in \mathcal{X}$, let $z_x \in \mathcal{W}$ be the true class label for x. The goal of pattern classification is to construct a functional mapping, $\mathcal{F} : \mathcal{X} \mapsto \mathcal{W}$, such that $\mathcal{F}(x) = z_x$ for all input patterns $x \in \mathcal{X}$.

The classifier $\mathcal{F}$ can be implemented as a real-value target function f that computes the following conditional probability density: $f(x) = P(z_x = w_1|x)$. We have:

$$\mathcal{F}(x) = \begin{cases} w_1 & \text{if } f(x) = P(z_x = w_1|x) > 0.5 \\ w_0 & \text{otherwise} \end{cases}$$

To construct $\mathcal{F}$, we simply learn the conditional probability density function f from examples. In practice, it may not even be necessary to learn the full conditional probability distribution function $f(x) = P(z_x = w_1|x)$. Any regression function that interpolates reasonably between the available data samples should suffice. The training data set $\mathcal{D}$ consists of (x, y) pairs in $\mathcal{X} \times \{-1.0, 1.0\}$. Each $x \in \mathcal{X}$ is an example input pattern in the original problem domain. The corresponding output y value is 1.0 if $z_x = w_1$, and -1.0 if $z_x = w_0$.

By way of example, let us consider the problem of predicting whether the stock market will go up or down over the next three months. Here there are two classes: (1) stock average rises; or (2) stock average declines or remains unchanged. The task is to predict which situation will occur, based on the observation of some current economic indicators. These indicators can be for example: prime interest rate, inflation rate, corporate earnings, stock price-to-earnings ratio,etc.

The sample data could be collected by examining economic records over a period of several years, reflecting varying stock market conditions. The task of the classifier

is to predict future stock market direction, which could be done by building a function approximation of the stock market index.

1.2 A Primer on Support Vector Machines

Support Vector Machines can be seen as an approximate implementation of what Vapnik has defined as Structurai Risk Minimization (SRM), an inductive principle that aims at minimizing a bound on the generalization error of a model, rather than minimizing the Mean Square Error over the particular data set.

In this section we present a primer on Support Vector Machines for pattern classification and regression that merely sketches their properties, but gives the reader an idea of: (1) the basic terminology; (2) the challenges in applying the technique; and (3) the impact of the work contained in this thesis.

It is important to remark that this section has been deliberately kept as short and as simple as possible, and that the technique's deeper theoretical framework will be discussed later in chapter 2 of this thesis.

1.2.1 SVMs for Pattern Classification

Support Vector Machines (SVMs) is a learning technique developed by V. Vapnik and his team at AT&T Bell Laboratories. that can be seen as a new method for training polynomial, neural network, or Radial Basis Functions classifiers. The decision surfaces are found by solving a linearly constrained quadratic programming problem.

In this section we briefly sketch the SVM algorithm and its motivation. A more detailed description of SVM can be found in chapter 2 of this thesis,[107] (chapter 5) and [27].

We start from the simple case of two linearly separable classes. We assume that we have a data set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$ of labeled examples, where $y_i \in \{-1, 1\}$, and we wish to determine, among the infinite number of linear classifiers that separate the data, which one will have the smallest generalization error. Intuitively, a good choice is the hyperplane with the maximum margin between the two classes, where

the margin is defined as the sum of the distances of the hyperplane from the closest point of the two classes (see figure 1-1).

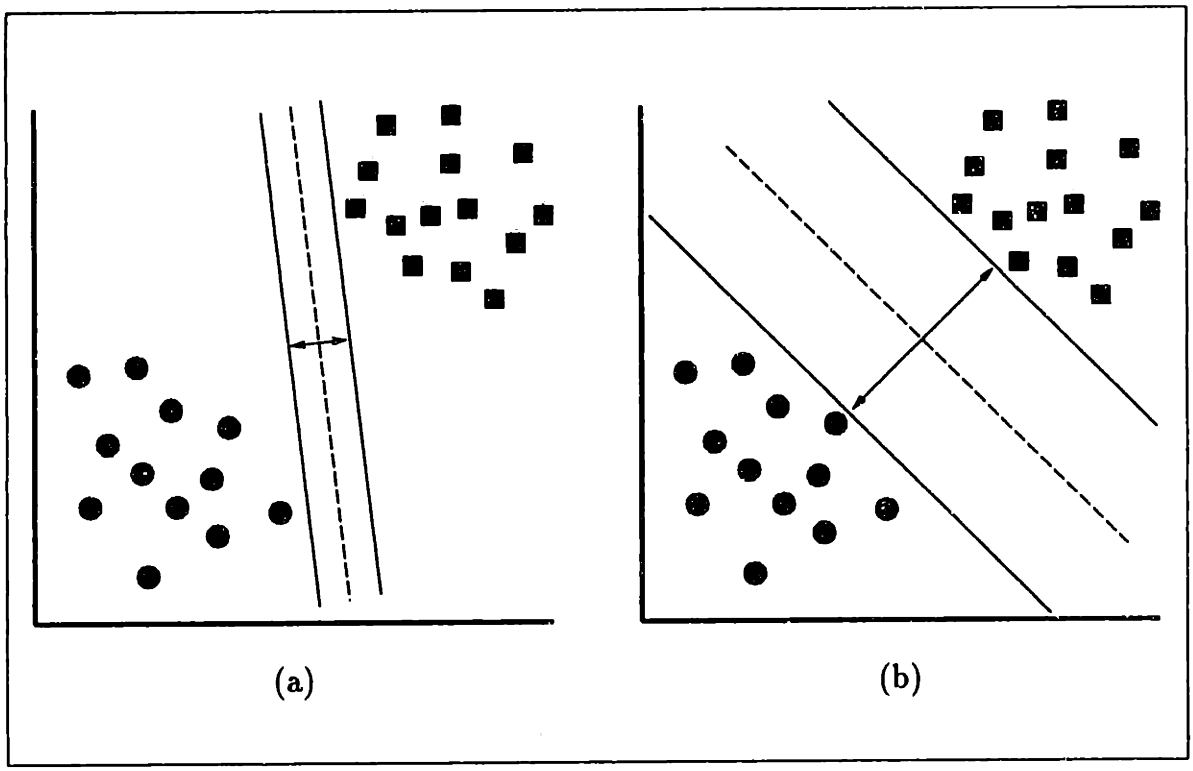


Figure 1-1: (a) A Separating Hyperplane with small margin. (b) A Separating Hyperplane with larger margin. A better generalization capability is expected from (b).

If the two classes are non-separable we can still look for the hyperplane that maximizes the margin and that minimizes a quantity that penalizes misclassification errors. The trade-off between margin and misclassification errors is controlled by a positive constant C that has to be chosen beforehand. In this case it can be shown that the solution to this problem is a linear classifier $f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}^T \mathbf{x}_i + b)$ whose coefficients λ_i (b can later be computed using Λ) are the solution of the following QP problem:

$$\begin{aligned} \text{Minimize} \quad & W(\Lambda) \quad = -\Lambda^T \mathbf{1} + \tfrac{1}{2}\Lambda^T D \Lambda \\ \Lambda & \\ \text{subject to} & \\ & \Lambda^T \mathbf{y} \quad = 0 \\ & \Lambda - C\mathbf{1} \quad \leq \mathbf{0} \\ & -\Lambda \quad \leq \mathbf{0} \end{aligned} \tag{1.1}$$

where $(\Lambda)_i = \lambda_i$, $(\mathbf{1})_i = 1$ and $D_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$. It turns out that in *easy* problems, only a small number of coefficients λ_i are different from zero, and since every coefficient corresponds to a particular data point, this means that the solution is determined by the data points associated with the non-zero coefficients. These data points, called *support vectors*, are the only ones which are relevant for the solution of the problem: all the other data points could be deleted from the data set and the same solution would be obtained. Intuitively, the support vectors are the data points that lie at the border between the two classes. Their number is usually small, and Vapnik showed that it is proportional to the generalization error of the classifier.

Since it is unlikely that any real life problem can actually be solved using a linear classifier, the technique has to be extended in order to allow for non-linear decision surfaces. This is easily done by mapping the original set of variables $\mathbf{x}$ into a higher dimensional *feature space*: $\mathbf{x} \in R^d \Rightarrow \mathbf{z}(\mathbf{x}) \equiv (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x})) \in R^n$ and by formulating the linear classification problem in the feature space. The solution will have the form $f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{\ell} \lambda_i y_i \mathbf{z}^T(\mathbf{x}) \mathbf{z}(\mathbf{x}_i) + b)$, and therefore will be nonlinear in the original input variables. One has to face at this point two problems: (1) the choice of the features $\phi_i(\mathbf{x})$, which should be done in a way that leads to a "rich" class of decision surfaces; (2) the computation of the scalar product $\mathbf{z}^T(\mathbf{x})\mathbf{z}(\mathbf{x}_i)$, which can be computationally prohibitive if the number of features n is very large (for example in the case in which one wants the feature space to span the set of polynomials in d variables the number of features n is exponential in d). A possible solution to this

problems consists in letting n go to infinity and make the following choice:

$$\mathbf{z}(\mathbf{x}) \equiv (\sqrt{\alpha_1}\psi_1(\mathbf{x}), \ldots, \sqrt{\alpha_i}\psi_i(\mathbf{x}), \ldots)$$

where α_i and ψ_i are the eigenvalues and eigenfunctions of an integral operator whose kernel $K(\mathbf{x}, \mathbf{y})$ is a positive definite symmetric function. With this choice the scalar product in the feature space becomes particularly simple because:

$$\mathbf{z}^T(\mathbf{x})\mathbf{z}(\mathbf{y}) = \sum_{i=1}^{\infty} \alpha_i \psi_i(\mathbf{x})\psi_i(\mathbf{y}) = K(\mathbf{x}, \mathbf{y})$$

where the last equality comes from the Mercer-Hilbert-Schmidt theorem for positive definite functions (see [85], pp. 242–246). The QP problem that has to be solved now is exactly the same as in eq. (1.1), with the exception that the matrix D has now elements $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$. As a result of this choice, the SVM classifier has the form: $f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{\ell} \lambda_i y_i K(\mathbf{x}, \mathbf{x}_i) + b)$. In table (1.1) we list some choices of the kernel function proposed by Vapnik: notice how they lead to well-known classifiers, whose decision surfaces are known to have good approximation properties.

Kernel Function	Type of Classifier
$K(\mathbf{x}, \mathbf{x}_i) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2)$	Gaussian RBF
$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T\mathbf{x}_i + 1)^d$	Polynomial of degree d
$K(\mathbf{x}, \mathbf{x}_i) = \tanh(\mathbf{x}^T\mathbf{x}_i - \Theta)$	Multi-Layer Perceptron

Table 1.1: Some possible kernel functions and the type of decision surfaces they define.

1.2.2 SVMs for Regression and Function Approximation

In this section we sketch the ideas behind the Support Vectors Machines (SVM) for regression; a more detailed description can be found in chapter 2 of this thesis, [111] and [107]. In a regression problem we are given a data set $G = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, obtained by sampling, with noise, some unknown function $g(\mathbf{x})$ and we are asked to determine a function f that approximates $g(\mathbf{x})$, based on the knowledge of G. The SVM considers approximating functions of the form:

$$f(\mathbf{x}, \mathbf{c}) = \sum_{i=1}^{D} c_i \phi_i(\mathbf{x}) + b \tag{1.2}$$

where the functions $\{\phi_i(\mathbf{x})\}_{i=1}^{D}$ are called *features*, and b and $\{c_i\}_{i=1}^{D}$ are coefficients that have to be estimated from the data. This form of approximation can be considered as a hyperplane in the D-dimensional feature space defined by the functions $\phi_i(\mathbf{x})$. The dimensionality of the feature space is not necessarily finite, and we will present examples in which it is infinite. The unknown coefficients are estimated by minimizing the following functional:

$$R(\mathbf{c}) = \frac{1}{N} \sum_{i=1}^{N} \mid y_i - f(\mathbf{x}_i, \mathbf{c}) \mid_\epsilon + \lambda \|\mathbf{c}\|^2 \tag{1.3}$$

where λ is a constant and the following *robust* error function has been defined:

$$\mid y_i - f(\mathbf{x}_i, \mathbf{c}) \mid_\epsilon = \begin{cases} 0 & \text{if } \mid y_i - f(\mathbf{x}_i, \mathbf{c}) \mid < \epsilon \\ \mid y_i - f(\mathbf{x}_i, \mathbf{c}) \mid - \epsilon & \text{otherwise.} \end{cases} \tag{1.4}$$

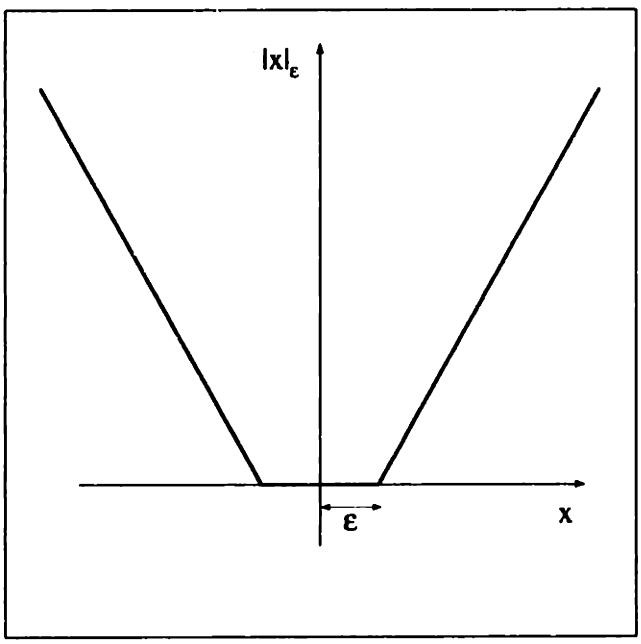


Figure 1-2: Vapnik's ϵ-insensitive cost function $V(x) = |x|_\epsilon$.

Vapnik showed in [107] that the function that minimizes the functional in eq. (1.3) depends on a finite number of parameters, and has the following form:

$$f(\mathbf{x}, \alpha, \alpha^*) = \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) K(\mathbf{x}, \mathbf{x}_i) + b, \tag{1.5}$$

where $\alpha_i^* \alpha_i = 0$, $\alpha_i, \alpha_i^* \geq 0$ $i = 1, \ldots, N$, and $K(\mathbf{x}, \mathbf{y})$ is the so called *kernel* function, and describes the inner product in the D-dimensional feature space:

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{D} \phi_i(\mathbf{x}) \phi_i(\mathbf{y})$$

The interesting fact is that for many choices of the set $\{\phi_i(\mathbf{x})\}_{i=1}^{D}$, including infinite-dimensional sets, the form of K is analytically known and very simple, and the features ϕ_i never need to be computed in practice because the algorithm relies only on computation of scalar products in the feature space. Several choices for the kernel K are available, including Gaussians, tensor product B-splines and trigonometric polynomials. The coefficients α and α^* are obtained by maximizing the following quadratic form:

$$R(\alpha^*, \alpha) = -\epsilon \sum_{i=1}^{N} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{N} y_i (\alpha_i^* - \alpha_i) - \frac{1}{2} \sum_{i,j=1}^{N} (\alpha_i^* - \alpha_i)(\alpha_i^* - \alpha_i) K(\mathbf{x}_i, \mathbf{x}_j), \tag{1.6}$$

subject to the constraints $0 \leq \alpha_i^*, \alpha_i \leq C$ and $\sum_{i=1}^{N} (\alpha_i^* - \alpha_i) = 0$. Due to the nature of this quadratic programming problem, only a small number of coefficients $\alpha_i^* - \alpha_i$ will be different from zero, and the data points associated with them are called *support vectors*. The parameters C and ϵ are two free parameters of the theory, and their choice is left to the user.

1.3 The Challenges of applying Support Vector Machines

When we started working with Support Vector Machines in the summer of 1995, only a few papers (some of them AT&T internal technical reports) existed and were mostly laying down the theoretical framework of the technique. From the applications point

of view, the only reported work was that of Vapnik's group on the Optical Character Recognition (OCR) problem. The technique looked promising, but a lot of research was (and still is) needed in order to apply it and understand its potential as well as its limitations.

In this section we briefly sketch some of the most important challenges that we have found in applying SVMs. The presentation of these challenges should provide a useful baseline for evaluating the contributions presented in this thesis.

1.3.1 Solving the QP

As it was sketched in section 1.2.1 , the decision surface for the pattern classification SVMs can be obtained by solving the QP given by:

$$\begin{aligned} \text{Maximize} \quad & F(\mathbf{\Lambda}) = \mathbf{\Lambda} \cdot \mathbf{1} - \tfrac{1}{2}\mathbf{\Lambda} \cdot D\mathbf{\Lambda} \\ \text{subject to} \quad & \\ & \mathbf{\Lambda} \cdot \mathbf{y} = 0 \\ & \mathbf{\Lambda} \leq C\mathbf{1} \\ & \mathbf{\Lambda} \geq \mathbf{0} \end{aligned}$$

where $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$.

This problem has the following key features or properties:

1. The QP has as many variables λ as data points.
2. The matrix D is symmetric, positive semi-definite and fully dense.
3. In classification problems where the data can be *easily* separated, the number of support vectors (i.e., the number of indices of Λ for which $\lambda_i \neq 0$) is usually very small.
4. In problems where the data cannot be separated *easily* and/or a lot of misclassifications occur, the number of support vectors can be very large.

5. The Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality.

This QP is considered challenging due to the large-scale characteristics it can have in real-life applications. For example, an application like our face detection system that uses 50,000 examples requires at least 25 Gb of RAM just to store the matrix D.

In order to solve this problem, we generate a finite sequence of smaller and more manageable subproblems that use the KKT conditions to assess optimality and directions of improvement.

It can be easily seen that the QP introduced in section 1.2.2 and used for regression SVMs shares similar properties, difficulties and so naturally leads to a similar solution strategy.

1.3.2 Reducing the Run-time complexity of SVMs

As we sketched in section 1.2.1, the classification of a new pattern is based on the sign of $f(\mathbf{x})$ defined as:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

where ℓ is the number of support vectors.

This operation can easily become the bottleneck of any system that performs a massive number of classifications. Examples of this issue arise in face and people detection systems, where SVMs are used as object-nonobject classifiers exhaustively, checks and ZIP code readers, where the system can only afford to spend fractions of a second per check or envelope, etc.

The reduction of the run-time complexity of SVMs can therefore be defined as a series of heuristics or techniques that reduce the computational effort spent in the evaluation or approximation of $f(\mathbf{x})$.

The same issue arises in regression SVMs, where $f(\mathbf{x})$ is similarly defined as:

$$f(\mathbf{x}, \alpha, \alpha^*) = \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) K(\mathbf{x}, \mathbf{x}_i) + b$$

In both pattern classification and regression machines, a speedup can be obtained by:

1. Approximating $f(\mathbf{x})$ as:

   $\hat{f}(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i K(\mathbf{z}_i, \mathbf{x}) + b$

   where $\mathbf{z}_i$ are *synthetic* vectors [1], γ_i are weights, and $\ell' \ll \ell$. The first approach of an algorithm to obtain this approximation was reported by C. Burges in [13], but the procedure is very slow and lacks a principled way for controlling the approximation accuracy.

2. Approximating $f(\mathbf{x})$ as:

   $\hat{f}(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$

   where $\mathbf{x}_i$ is still a support vector with weight γ_i, but $\ell' \ll \ell$. This will be the our first approach and it will be described in detail in chapter 4.

3. Find (when possible) another expansion for $f(\mathbf{x})$ as: $f(\mathbf{x}) = \hat{f} = \sum_{i=1}^{\ell'} \gamma_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$ where $\ell' \ll \ell$. This will be our second approach and it will also be descibed in detail in chapter 4.

Both of these heuristics try to approximate $f(\mathbf{x})$ using the kernel operator so that they can establish a meaningful accuracy comparison through the L_2 norm measured in *feature space*. Therefore, they strongly depend on the mapping:

$\mathbf{x} \in R^d \Rightarrow \mathbf{z}(\mathbf{x}) \equiv (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x})) \in R^n$

where: (1) n can be huge, and (2) not very much is known about the characteristics of the mapping itself. These two properties make this problem very hard to solve.

[1] Notice that these vectors are not necessarily data points anymore.

1.3.3 Shifts, Scales and other key parameters

As we mentioned in section 1.2.1, non-linear decision surfaces can be obtained by mapping the original set of variables $\mathbf{x}$ into a higher dimensional *feature space*: $\mathbf{x} \in R^d \Rightarrow \mathbf{z}(\mathbf{x}) \equiv (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x})) \in R^n$, and by formulating the linear classification problem in this feature space. This powerful mapping is governed by the kernel operator, and it is not a surprise that not only its *structure*, but also its parameters, can be important in obtaining a *favorable* mapping. Examples of these parameters are:

1. The location of the origin (i.e., shifting), the relative and absolute magnitude of the input vector components (i.e. scaling), and the degree in polynomial kernels.

2. The extent of the localization (i.e., σ) in the Gaussian kernel.

As we explained previously, the problems of (1) huge dimensionality and (2) lack of information about the characteristics of the mapping, makes this issue hard to understand.

Our objective is to study the influence of these parameters, explore some alternatives to optimize them, and characterize their impact on the geometric and theoretical framework of SVMs.

1.4 Brief Literature Review

We now provide a very brief literature review on Support Vector Machines and other related topics. Our discussion here is deliberately kept brief because many of the results mentioned here are discussed in greater detail in corresponding and appropriate chapters of the thesis.

1.4.1 Theoretical Background

The origins of Support Vector Machines can be traced back to the mid-to late-1970's, when books by Vapnik and Chervonenkis [109] and Vapnik [106] described the *Gener-*

alized Portrait algorithm for constructing separating hyperplanes with optimal margin.

Almost 15 years later, Boser,Guyon and Vapnik [8] [44] used Mercer kernels to generalize from linear hyperplanes to nonlinear decision surfaces. Extension to non-separable data-sets would come in 1995 when Cortes and Vapnik introduced slack variables in the separability formulation to allow penalized misclassifications. A comprehensive overview of SVMs and the first description of its regression extension were presented recently by Vapnik in [107].

We consider it relevant to mention that in their 1973 book, Duda and Hart [31] present a slightly different concept of margin that leads to the same optimal margin classifier. Interestingly enough, they also present a similar extension to non-separable data sets, and the use of *potential functions* (see Aizerman *et al.* [1]) as a way to achieve nonlinear decision surfaces. This emphasizes Vapnik's own remarks (see the footnote on page 136 in [107]) that support vectors could have been discovered 30 years ago, since most of its basic ideas had been uncovered by the mid-1960's.

Other recent theoretical results include work done by Pontil and Verri [84], where they discuss some interesting properties of SVMs; by Smola and Schölkopf [96], where they explore kernel-based methods for pattern recognition and function approximation; and by Girosi [43], where he derives SVMs in the framework of regularization theory and shows the equivalence between SVMs and a modified version of Basis Pursuit De-Noising (see Chen [26] and Chen, Donoho and Saunders [25]).

Additional related work has been reported by Schölkopf, Smola and Müller in [93], where they describe a nonlinear form of Principal Component Analysis using the same kernel operators that SVMs; by Bennet and Blue in [3], where they examine a SVM approach to Decision Trees; by Bennet, Bradley, Fayyad and Mangasarian [4] [9] [10] where they study the use of mathematical programming in machine learning ;and by Schölkopf, Burges and Vapnik in [92], where they report on a method of incorporating prior knowledge about transformation invariances by generating artificial or *virtual* examples. This approach is very similar to the work done by Poggio and Vetter in [82], which has been successfully used, for example, in face detection systems by Sung

and Poggio [100], Rowley *et al.* [88], and Osuna *et al.* [78].

1.4.2 Support Vector Machine Training

The first appearance of SVM training can be dated back to [8]. This paper informally suggests the idea of partitioning or *chunking* the data and only considers the separable case.

The first implementation of SVM training was reported internally at AT&T Bell Laboratories in work done by Burges and Vapnik [16]. Their approach used a constrained conjugate gradient adaptation from Moré and Toraldo [66], and incorporated some form of data partitioning that did not guarantee optimality.

More recently, Pontil and Verri [83] have reported an implementation using a transformation to the *Linear Complementarity Problem* (see also page 503 in [2]) that becomes impractical for data sets of more than 1,000 data points.

The group working at Lucent Technologies (Burges, Kaufman and others) has implemented a projected Hessian [11] algorithm with certain heuristics for data partitioning [53] similar to the approach we used in our first decomposition algorithm [77] [78] reported in section 3.3, but not as complete as its later improved version [76] covered in section 3.4.

At this point in time we know that Smola *et al.*, at the Max Planck Institute, and Stitson *et al.*,at Royal Holloway and Bedford College, are implementing our improved version. Companies like Daimler Benz and AT&T have been provided with our complete computer implementation.

1.4.3 Reduction of Run-time complexity

The problem of reducing the run-time effort of SVMs was first approached by Burges in 1996 [13]. His approach, called the *Reduced Set Method*, tries to approximate the decision surface defined by the support vectors using *synthesized* vectors that are not data points anymore. Another reference for this work can be found in [15]. So far this problem has only been approached by Burges, and constitutes an open area of

research.

1.4.4 Existing Applications of Support Vector Machines

The number of reported applications that use SVMs is very limited. The Optical Character Recognition (OCR) problem is perhaps the first real-life application of the technique and was first approached by Vapnik and his co-workers in AT&T Bell Labs in 1992 [8] using small data sets. Further results were reported in 1995 [16] [91] [107] and 1997 [15] [108].The performance obtained using SVMs was very close to the best reported technique by Le Cun *et al.* [28]. Other applications include work done by Schmidt in identification of speakers [90]; and 3D object recognition by Blanz *et al.* [6], and Pontil and Verri [83].

Experimental results using the regression extension are also scarce: Vapnik, Golovich and Smola [111] present results obtained on small *toy* problems; Drucker *et al.* [30] compare it with a committee technique (bagging) based on regression trees and ridge regression in feature space; and more recently[2] Müller *et al.* [70] have used it for time series prediction and compared it with radial basis functions.

1.4.5 Face and People Detection

The problem of face detection has been approached with different techniques in the last few years. These techniques include Neural Networks [12] [88], detection of face features and use of geometrical constraints [114], density estimation of the training data [63], labeled graphs [54] and clustering and distribution-based modeling [100][99].

Out of all these previous works, the results of Sung and Poggio [100][99], and Rowley *et al.* [88] reflect systems with very high detection rates and low false positive detection rates.

There has been a body of work on people detection reported by Tsukiyama and Shirai in 1985 [102], Leung and Yang in 1987 [57][56], Rohr in 1993[87], and Chen and

[2]This work was done after we published in [69] our results on Time Series Analysis described in chapter 6, section 6.3.

Shirai in 1994 [23]. All of these approaches are heavily based on motion and hand crafted models.

1.5 Thesis Outline and Contributions

In Chapter one of the thesis we introduce the paradigm of learning by examples and its relationship with the tasks of pattern classification and function approximation. We also present a primer on Support Vector Machines that merely sketches its properties, but gives the reader an idea of: (1) the basic terminology; (2) the challenges for applying the technique; and (3) the impact of the work contained in this thesis.

In Chapter two we explain in detail the mathematical derivation behind Support Vector Machines for pattern classification and regression. We also provide geometric interpretation and some extensions of these techniques.

In Chapter three we present an active set method for solving the Quadratic Program that defines a Support Vector Machine. Experimental results obtained with our implementation of the algorithm and future directions of research are also provided.

In Chapter four we describe the problem of reducing the execution time required by SVMs when performing classification and estimation, and present a solution to some instances of this problem using Support Vector Regression Machines.

In Chapter five we describe and formulate the problem of finding the radius of the smallest sphere that contains the data points in a high dimensional space that is only characterized by its dot product operator. We also present an active set algorithm and its implementation for solving this problem. Using the radius and a reformulation of the original SVM problem, we offer an alternative mathematical program for Structural Risk Minimization.

In Chapter six we present applications using SVMs that include: face detection, people detection, nonlinear prediction of chaotic time series and image reconstruction.

In Chapter seven we present our conclusions and suggest areas for future research.

The contributions of this thesis are as follows:

1. An active set algorithm for training Support Vector Machines that exploits its geometrical properties, guarantees global optimality, and makes possible the solution of problems with hundreds of thousands of data points that were previously unsolvable.

2. An implementation of this algorithm for both pattern classification and regression problems using MINOS 5.4 [71] [72] as the solver of the subproblems generated by the algorithm. This implementation has been tested under several computer architectures (Sun, Silicon Graphics and IBM PCs) and operating systems (Sun OS, Solaris, Irix and Microsoft Windows NT).

3. A Support Vector Regression Machine solution to the problem of reducing the run-time complexity when performing classification and estimation for the class of problems where many of the non-zero variables λ_i have an active upper bound (i.e., $\lambda_i = C$).

4. A primal reformulation of the SVM training problem which yields the same decision surface obtained by solving the *traditional* QP training problem. This formulation is used in the problem of reducing the run-time complexity when many of the non-zero variables λ_i have an active upper bound (i.e., $\lambda = C$), and is also used in an alternative formulation to Structural Risk Minimization.

5. An active set algorithm for finding the radius of the smallest sphere that contains the data points in feature space (i.e., the high-dimensional space where data points are mapped in nonlinear SVMs). This algorithm also exploits the geometrical properties of the problem, guarantees global optimality, and solves previously unsolvable problems.

6. An implementation of (5) that also uses MINOS 5.4 as the solver for the subproblems generated by the algorithm.

7. An alternative formulation to Structural Risk Minimization that finds not only the support vectors, but also optimizes over some key parameters that drive the behavior of the kernel-based mapping.

8. Applications of SVMs to face detection, people detection and time series analysis. These applications are important since they have been the first to follow the OCR application developed by Vapnik and his co-workers at AT&T Research.

Chapter 2

Support Vector Machines

2.1 Empirical Risk Minimization

In the case of two-class pattern recognition, the task of learning from examples can be formulated in the following way: given a set of decision functions

$$\{f_\lambda(\mathbf{x}) : \lambda \in \Lambda\}, \qquad f_\lambda : \Re^N \to \{-1, 1\}$$

where Λ is a set of abstract parameters, and a set of examples

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell), \qquad \mathbf{x}_i \in \Re^N, y_i \in \{-1, 1\}$$

drawn from an unknown distribution $P(\mathbf{x}, y)$, we want to find a function f_{λ^*} which provides the smallest possible value for the *expected risk*:

$$R(\lambda) = \int |f_\lambda(\mathbf{x}) - y| \, P(\mathbf{x}, y) d\mathbf{x} dy$$

The functions f_λ are usually called *hypothesis*, and the set $\{f_\lambda(\mathbf{x}) : \lambda \in \Lambda\}$ is called the *hypothesis space* and it is denoted by $\mathcal{H}$. The expected risk is therefore a measure of how good an hypothesis is at predicting the correct label y for a point $\mathbf{x}$. The set of functions f_λ could be, for example, a set of Radial Basis Functions, or a

Multi-Layer Perceptron with a certain number of hidden units. In this case, the set Λ is the set of weights of the network. It is important to remark that this space is not binary and in practice it is approximated over $\Re$ to later threshold using the $sign(\cdot)$ operator.

Since the probability distribution $P(\mathbf{x}, y)$ is unknown, we are unable to compute, and therefore to minimize, the expected risk $R(\lambda)$. However, since we have access to a sampling of $P(\mathbf{x}, y)$, we can compute a stochastic approximation of $R(\lambda)$, the so-called *empirical risk*:

$$R_{\text{emp}}(\lambda) = \frac{1}{\ell} \sum_{i=1}^{\ell} |f_\lambda(\mathbf{x}_i) - y_i|$$

Since the law of large numbers guarantees that the empirical risk converges in probability to the expected risk, a common approach consists in minimizing the empirical risk rather than the expected risk. The intuition underlying this approach (the *Empirical Risk Minimization Principle*) is that if R_{emp} converges to R, the minimum of R_{emp} may converge to the minimum of R. If convergence of the minimum of R_{emp} to the minimum of R does not hold, the Empirical Risk Minimization Principle does not allow us to make any inference based on the data set, and it is therefore said to be *not consistent*. As shown by Vapnik and Chervonenkis [112, 110, 106] consistency takes place if and only if convergence in probability of R_{emp} to R is replaced by *uniform* convergence in probability. Vapnik and Chervonenkis [112, 110, 106] showed that a necessary and sufficient condition for consistency of the Empirical Risk Minimization Principle is the finiteness of the *VC-dimension* h of the hypothesis space $\mathcal{H}$. The VC-dimension of the hypothesis space $\mathcal{H}$ (or VC-dimension of the classifier f_λ) is a natural number, possibly infinite, which is, losely speaking, the largest number of data points that can be separated in all possible ways by that set of functions f_λ. The VC-dimension is a measure of the complexity of the set $\mathcal{H}$, and it is often, but not necessarily, proportional to the number of free parameters of the classifier f_λ.

The theory of uniform convergence in probability developed by Vapnik and Chervonenkis also provides bounds on the deviation of the empirical risk from the expected

risk. A typical uniform Vapnik and Chervonenkis bound, which holds with probability $1 - \eta$, has the following form:

$$R(\lambda) \leq R_{\text{emp}}(\lambda) + \sqrt{\frac{h\left(\ln\frac{2l}{h} + 1\right) - \ln\frac{\eta}{4}}{l}} \qquad \forall\lambda \in \Lambda \tag{2.1}$$

where h is the VC-dimension of f_λ. From this bound it is clear that, in order to achieve small expected risk (i.e., good generalization performances), both the empirical risk and the ratio between the VC-dimension and the number of data points has to be small. Since the empirical risk is usually a decreasing function of h, it turns out that, for a given number of data points, there is an optimal value of the VC-dimension. The choice of an appropriate value for h (which in most techniques is controlled by the number of free parameters of the model) is crucial in order to get good performances, especially when the number of data points is small. When using a Multilayer Perceptron or a Radial Basis Functions network, this is equivalent to the problem of finding the appropriate number of hidden units. This problem is known to be difficult, and it is usually solved by some sort of cross-validation technique.

The bound (2.1) suggests that the Empirical Risk Minimization Principle can be replaced by a better induction principle, as we will see in the next section.

2.2 Structural Risk Minimization

The technique of *Structural Risk Minimization* (SRM) developed by Vapnik [106] is an attempt to overcome the problem of choosing an appropriate VC-dimension. It is clear from equation (2.1) that a small value of the empirical risk does not necessarily imply a small value of the expected risk. A different induction principle, called the *Structural Risk Minimization Principle*, has been proposed by Vapnik [106]. The principle is based on the observation that, in order to make the expected risk small, both sides in equation (2.1) should be small. Therefore, both the VC-dimension and the empirical risk should be minimized at the same time.

In order to implement the SRM principle one needs a nested structure of hypoth-

esis spaces

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_n \subset \ldots$$

with the property that $h(n) \leq h(n+1)$ where $h(n)$ is the VC-dimension of the set $\mathcal{H}_n$. Then equation (2.1) suggests that, disregarding logarithmic factors, the following problem should be solved:

$$\min_{\mathcal{H}_n} \left(R_{\text{emp}}[\lambda] + \sqrt{\frac{h(n)}{l}} \right) \tag{2.2}$$

The SRM principle is clearly well founded mathematically, but it can be difficult to implement for the following reasons:

1. The VC-dimension of $\mathcal{H}_n$ could be difficult to compute, and there are only a small number of models for which we know how to compute the VC-dimension.

2. Even assuming that we can compute the VC-dimension of $\mathcal{H}_n$, it is not easy to solve the minimization problem (2.2). In most cases one will have to minimize the empirical risk for every set $\mathcal{H}_n$, and then choose the $\mathcal{H}_n$ that minimizes equation (2.2).

Therefore the implementation of this principle is not easy, because it is not trivial to control the VC-dimension of a learning technique during the training phase. The SVM algorithm achieves this goal, minimizing a bound on the VC-dimension and the number of training errors at the same time. In the next section we discuss this technique in detail, and show how its implementation is related to quadratic programming.

2.3 Support Vector Machines for Pattern Classification

In this section we introduce the SVM classification technique, and show how it leads to the formulation of a QP problem in a number of variables that is equal to the

number of data points.

2.3.1 Mathematical Derivation

In this section we describe the mathematical derivation of the Support Vector Machine (SVM) developed by Vapnik [107]. The technique is introduced in steps: we first consider the simplest case, a linear classifier and a linearly separable problem; then a linear classifier and non-separable problem, and finally a non-linear classifier and non-separable problem, which is the most interesting and useful case.

Linear Classifier and Linearly Separable Problem

In this section we consider the case in which the data set is *linearly separable*, and we wish to find the "best" hyperplane that separates the data. For our purposes, linearly separable means that we can find a pair $(\mathbf{w}, b)$ such that:

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq \quad 1 \qquad \forall \mathbf{x}_i \in \text{Class 1} \tag{2.3}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq \quad -1 \qquad \forall \mathbf{x}_i \in \text{Class 2} \tag{2.4}$$

The hypothesis space in this case is therefore the set of functions given by

$$f_{\mathbf{w},b} = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.5}$$

Notice that if the parameters $\mathbf{w}$ and b are scaled by the same quantity, the decision surface given by (2.5) is unchanged. In order to remove this redundancy, and to make each decision surface correspond to one unique pair $(\mathbf{w}, b)$, the following constraint is imposed:

$$\min_{i=1,\ldots,\ell} |\mathbf{w} \cdot \mathbf{x}_i + b| = 1 \tag{2.6}$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ are the points in the data set. The set of hyperplanes that satisfy

(2.6) are called *Canonical Hyperplanes*. Notice that all linear decision surfaces can be represented by *Canonical Hyperplanes*, and constraint (2.6) is just a normalization, which will prove to be very convenient in the following calculations.

If no further constraints are imposed on the pair $(\mathbf{w}, b)$ the VC-dimension of the *Canonical Hyperplanes* is $N + 1$ [107], that is, the total number of free parameters. In order to be able to apply the Structural Risk Minimization Principle we need to construct sets of hyperplanes of varying VC-dimension, and minimize both the empirical risk (the training classification error) and the VC-dimension at the same time. A structure on the set of canonical hyperplanes is defined by constraining the norm of the vector $\mathbf{w}$. In fact, Vapnik shows that, if we assume that all the points $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ lie in the unit N-dimensional sphere, the set

$$\{f_{\mathbf{w},b} = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \mid \|\mathbf{w}\| \leq A\} \tag{2.7}$$

has a *VC-dimension* h that satisfies the following bound [107] [106]:

$$h \leq \min\{\lceil A^2 \rceil, N\} + 1 \tag{2.8}$$

If the data points lie inside a sphere of radius R, then (2.8) becomes

$$h \leq \min\{\lceil R^2 A^2 \rceil, N\} + 1.$$

The geometrical reason for which bounding the norm of $\mathbf{w}$ constraints the set of canonical hyperplanes is very simple. It can be shown that the distance from a point $\mathbf{x}$ to the hyperplane associated to the pair $(\mathbf{w}, b)$ is:

$$d(\mathbf{x}; \mathbf{w}, b) = \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{\|\mathbf{w}\|} \tag{2.9}$$

According to the normalization (2.6) the distance between the canonical hyperplane $(\mathbf{w}, b)$ and the closest of the data points is simply $\frac{1}{\|\mathbf{w}\|}$. Therefore, if $\|\mathbf{w}\| \leq A$ then the distance of the canonical hyperplane to the closest data point has to be larger than $\frac{1}{A}$. We can then conclude that the constrained set of canonical hyperplanes of

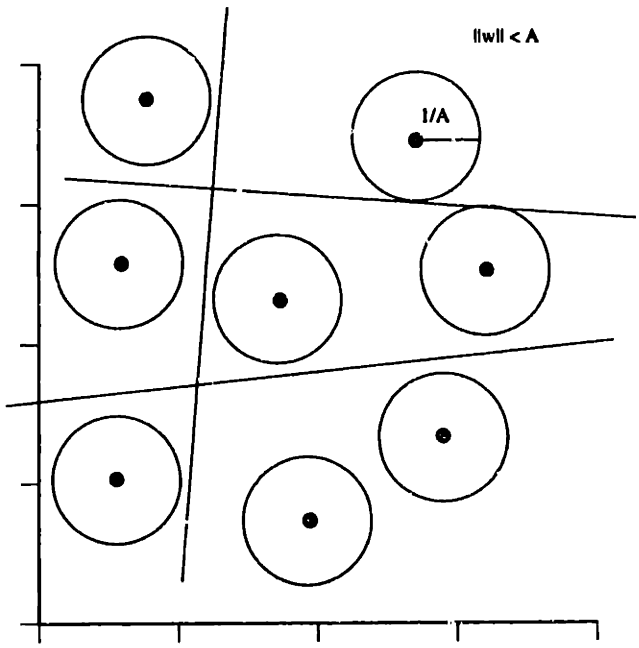


Figure 2-1: Bounding the norm of $\mathbf{w}$ is equivalent to constraining the hyperplanes to remain outside spheres of radius $\frac{1}{A}$ centered around the data points.

equation (2.7) is the set of hyperplanes whose distance from the data points is at least $\frac{1}{A}$. This is equivalent to placing spheres of radius $\frac{1}{A}$ around each data point, and consider only the hyperplanes that do not intersect any of the spheres, as shown in figure (2-1).

If the set of examples is linearly separable, the goal of the SVM is to find, among the *Canonical Hyperplanes* that correctly classify the data, the one with minimum norm, or equivalently minimum $\|\mathbf{w}\|^2$, because keeping this norm small will also keep the *VC-dimension* small. It is interesting to see that minimizing $\|\mathbf{w}\|^2$ (in this case of linear separability) is equivalent to finding the separating hyperplane for which the distance between the two convex hulls (of the two classes of training data), measured along a line perpendicular to the hyperplane, is maximized. In the rest of this paper, this distance will be referred to as the *margin*. Figure (2-2) gives some geometrical interpretation of why better *generalization* is expected from a separating hyperplane with large *margin*.

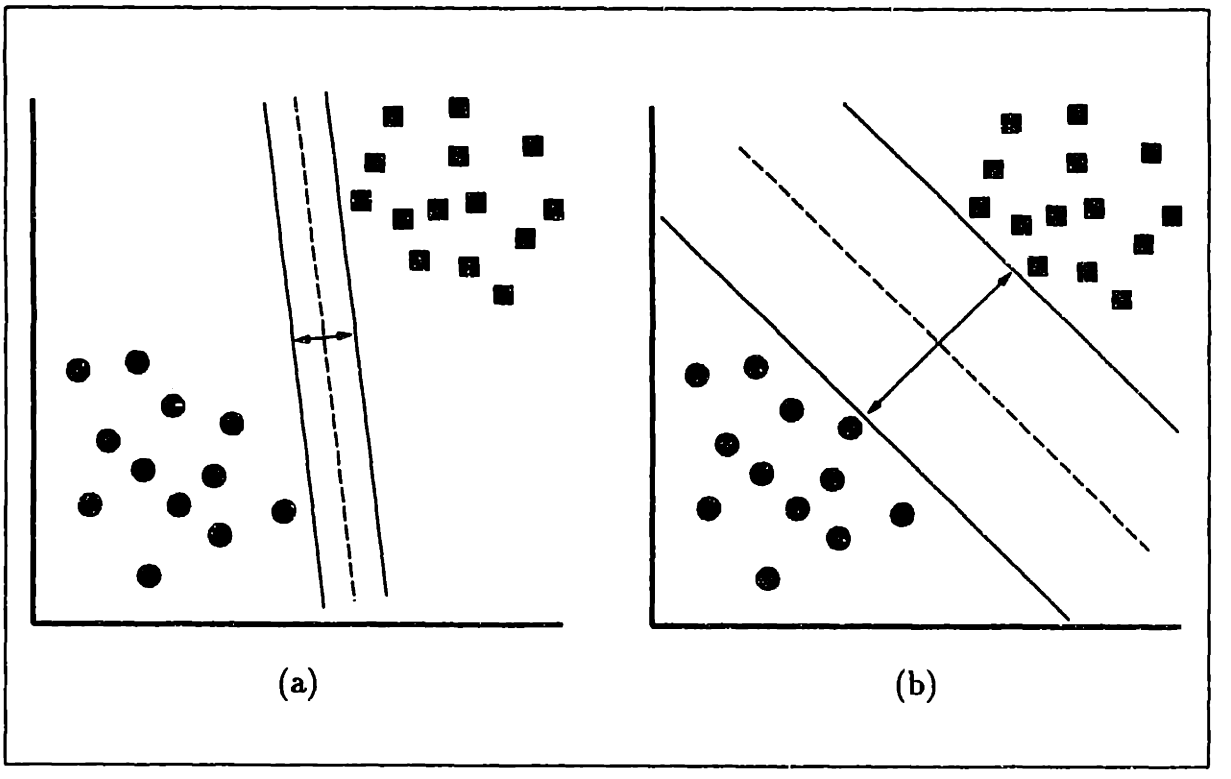


Figure 2-2: (a) A Separating Hyperplane with small *margin*. (b) A Separating Hyperplane with larger *margin*. A better *generalization* capability is expected from (b).

To construct the *maximum margin* or *optimal* separating hyperplane, we need to correctly classify the vectors $\mathbf{x}_i$ of the training set

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell), \qquad \mathbf{x}_i \in \Re^N$$

into two different classes $y_i \in \{-1, 1\}$, using the smallest norm of coefficients. This can be formulated as follows:

$$\begin{aligned} &\text{Minimize} \qquad \Phi(\mathbf{w}) \qquad = \frac{1}{2}\|\mathbf{w}\|^2 \\ &\mathbf{w}, b \\ &\text{subject to} \\ &\qquad\qquad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \;\geq 1 \qquad i = 1 \ldots \ell \end{aligned} \tag{2.10}$$

At this point, this problem can be solved using standard Quadratic Programming (QP) optimization techniques and is not very complex since the dimension of the QP is $N+1$. Since N is the dimension of the input space, this problem is more or less tractable for real applications. Nevertheless, in order to easily explain [1] the extension to nonlinear decision surfaces (which will be described in section 2.3.1), we look at the dual problem, and use the technique of Lagrange multipliers. We construct the Lagrangian

$$L(\mathbf{w}, b, \mathbf{\Lambda}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \lambda_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1], \tag{2.11}$$

where $\mathbf{\Lambda} = (\lambda_1, \ldots, \lambda_\ell)$ is the vector of non-negative Lagrange multipliers corresponding to the constraints in (2.10).

The solution to this optimization problem is determined by a saddle point of this Lagrangian, which has to be minimized with respect to $\mathbf{w}$ and b, and maximized with respect to $\mathbf{\Lambda} \geq 0$. Differentiating (2.11) and setting the gradient equal to zero we obtain:

$$\frac{\partial L(\mathbf{w}, b, \mathbf{\Lambda})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}_i = 0 \tag{2.12}$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{\Lambda})}{\partial b} = \sum_{i=1}^{\ell} \lambda_i y_i = 0 \tag{2.13}$$

Using the superscript $*$ to denote the *optimal* values of the cost function, from equation (2.12) we derive:

$$\mathbf{w}^* = \sum_{i=1}^{\ell} \lambda_i^* y_i \mathbf{x}_i \tag{2.14}$$

which shows that the *optimal* hyperplane solution can be written as a nonnegative linear combination of the training vectors. Notice that only those training vectors $\mathbf{x}_i$

[1] As we will show in Chapter 4 (section 4.5), this step is not vital for the non-linear extension of SVM.

with $\lambda_i > 0$ contribute in the expansion (2.14).

Substituting (2.14) and (2.13) into (2.11) we obtain:

$$F(\mathbf{\Lambda}) = \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2}\|\mathbf{w}^*\|^2 = \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \qquad (2.15)$$

Writing (2.15) in matrix notation, incorporating non-negativity of $\mathbf{\Lambda}$ and constraint (2.13) we obtain the following dual quadratic program:

$$\begin{aligned} \text{Maximize} \quad F(\mathbf{\Lambda}) &= \mathbf{\Lambda} \cdot \mathbf{1} - \frac{1}{2}\mathbf{\Lambda} \cdot D\mathbf{\Lambda} \\ \text{subject to} & \\ \mathbf{\Lambda} \cdot \mathbf{y} &= 0 \\ \mathbf{\Lambda} &\geq 0 \end{aligned}$$

where $\mathbf{y} = (y_1, \ldots, y_\ell)$ and D is a symmetric $\ell \times \ell$ matrix with elements $D_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$.

Notice that complementary slackness conditions of the form:

$$\lambda_i^* [y_i(\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1] = 0 \qquad i = 1, \ldots, \ell \qquad (2.16)$$

imply that $\lambda_i > 0$ only when constraint (2.10) is active. The vectors for which $\lambda_i > 0$ are called *Support Vectors*. From equation (2.16) it follows that b^* can be computed as:

$$b^* = y_i - \mathbf{w}^* \cdot \mathbf{x}_i$$

for any support vector $\mathbf{x}_i$. By linearity of the dot product and equation (2.14), the decision function (2.5) can then be written as:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{\ell} y_i \lambda_i^* (\mathbf{x} \cdot \mathbf{x}_i) + b^*\right) \qquad (2.17)$$

The Soft Margin Hyperplane: Linearly Non-Separable Case

We now consider the case in which we still seek a linear separating surface, but a separating hyperplane does not exist, so that it is not possible to satisfy all of the constraints in problem (2.10). In order to deal with this case one introduces a new set of variables $\{\xi_i\}_{i=1}^{\ell}$, that measure the amount of violation of the constraints. Then the margin is maximized, paying a penalty proportional to the amount of constraint violations. Formally, one solves the following problem:

$$\begin{aligned} &\text{Minimize} \quad \Phi(\mathbf{w}, \Xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C(\sum_{i=1}^{\ell} \xi_i)^k && (2.18) \\ &\mathbf{w}, b, \Xi \\ &\text{subject to} \\ &\qquad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \qquad i = 1, \ldots, \ell && (2.19) \\ &\qquad \xi_i \geq 0 \qquad i = 1, \ldots, \ell && (2.20) \end{aligned}$$

where C and k are parameters which have to be determined beforehand and define the cost of constraints violation. Other monotonic convex functions of the errors can be defined (see [27] for the more general case). Notice that minimizing the first term in (2.18) amounts to minimizing the *VC-dimension* of the learning machine, thereby minimizing the second term in the bound (2.1). On the other hand, minimizing the second term in (2.18) controls the *empirical risk*, which is the first term in the bound (2.1). This approach, therefore, constitutes a practical implementation of *Structural Risk Minimization* on the given set of functions. In order to solve problem (2.18), we construct the Lagrangian:

$$L(\mathbf{w}, b, \Lambda, \Xi, \Gamma) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \lambda_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{\ell} \gamma_i \xi_i + C(\sum_{i=1}^{\ell} \xi_i)^k, \quad (2.21)$$

where the non-negative multipliers $\Lambda = (\lambda_1, \ldots, \lambda_\ell)$ and $\Gamma = (\gamma_1, \ldots, \gamma_\ell)$ are asso-

ciated with constraints (2.19) and (2.20) respectively. The solution to this problem is determined by the saddle point of this Lagrangian, which has to be minimized with respect to $\mathbf{w}$, $\boldsymbol{\Xi}$ and b, and maximized with respect to $\boldsymbol{\Lambda} \geq 0$ and $\boldsymbol{\Gamma} \geq 0$. Differentiating (2.21) and setting the results equal to zero, we obtain:

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\Lambda}, \boldsymbol{\Xi}, \boldsymbol{\Gamma})}{\partial \mathbf{w}} = (\mathbf{w} - \sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}_i) = 0 \tag{2.22}$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\Lambda}, \boldsymbol{\Xi}, \boldsymbol{\Gamma})}{\partial b} = \sum_{i=1}^{\ell} \lambda_i y_i = 0 \tag{2.23}$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\Lambda}, \boldsymbol{\Xi}, \boldsymbol{\Gamma})}{\partial \boldsymbol{\Xi}} = \begin{cases} kC \left(\sum_{i=1}^{\ell} \xi_i\right)^{k-1} - \lambda_i - \gamma_i = 0 & k > 1 \\ C - \lambda_i - \gamma_i = 0 & k = 1. \end{cases} \tag{2.24}$$

When $k > 1$, by denoting

$$\sum_{i=1}^{\ell} \xi_i = \left(\frac{\delta}{Ck}\right)^{\frac{1}{k-1}}, \tag{2.25}$$

we can rewrite equation (2.24) as:

$$\delta - \lambda_i - \gamma_i = 0. \tag{2.26}$$

From equation (2.22) we obtain:

$$\mathbf{w}^* = \sum_{i=1}^{\ell} \lambda_i^* y_i \mathbf{x}_i \tag{2.27}$$

Substituting (2.27), (2.23) and (2.25) into (2.21) we obtain:

$$F(\boldsymbol{\Lambda}, \delta) = \sum_{i=1}^{\ell} \lambda_i - \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \frac{\delta^{\frac{k}{k-1}}}{(kC)^{\frac{1}{k-1}}} \left(1 - \frac{1}{k}\right) \tag{2.28}$$

Therefore, in order to obtain the *Soft Margin* separating hyperplane we solve:

$$\text{Maximize} \quad F(\Lambda, \delta) \;= \Lambda \cdot \mathbf{1} - \tfrac{1}{2}\Lambda \cdot D\Lambda - \frac{\delta^{\frac{k}{k-1}}}{(kC)^{\frac{1}{k-1}}}\left(1 - \tfrac{1}{k}\right)$$

subject to

$$\begin{aligned} \Lambda \cdot \mathbf{y} \quad &= 0 \\ \Lambda \quad &\leq \delta \mathbf{1} \\ \Lambda \quad &\geq \mathbf{0} \end{aligned} \tag{2.29}$$

where $\mathbf{y} = (y_1, \ldots, y_\ell)$ and D is a symmetric $\ell \times \ell$ matrix with elements $D_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$.

When $k = 1$, that is, penalizing linearly the violations in constraint (2.19), the set of equations (2.29) simplifies to:

$$\text{Maximize} \quad F(\Lambda) \;= \Lambda \cdot \mathbf{1} - \tfrac{1}{2}\Lambda \cdot D\Lambda$$

subject to

$$\begin{aligned} \Lambda \cdot \mathbf{y} \quad &= 0 \\ \Lambda \quad &\leq C\mathbf{1} \\ \Lambda \quad &\geq \mathbf{0} \end{aligned} \tag{2.30}$$

The value $k = 1$ is assumed for the rest of this thesis, since it simplifies the mathematical formulation and has shown very good results in practical applications. By the linearity of the dot product and equation (2.27), the decision function (2.5) can be written as:

$$f(\mathbf{x}) \;=\; \text{sign}\left(\sum_{i=1}^{\ell} y_i \lambda_i^* (\mathbf{x} \cdot \mathbf{x}_i) + b^*\right) \tag{2.31}$$

where $b^* \;=\; y_i - \mathbf{w}^* \cdot \mathbf{x}_i$, for any support vector $\mathbf{x}_i$ such that $0 < \lambda_i < C$ (that is a support vector which is correctly classified). In order to verify this, notice that complementary slackness in the conditions of the form:

$$\lambda_i^*[y_i(\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1 + \xi_i] = 0 \qquad i = 1, \ldots, \ell \tag{2.32}$$

imply that $\lambda_i > 0$ only when constraint (2.19) is active, establishing the need for $\lambda_i > 0$. On the other hand, (2.19) can be active due to $\xi_i > 0$, which is not acceptable since $\mathbf{x}_i$ would be a misclassified point. For $k = 1$ in equation (2.24) we have $\gamma_i = C - \lambda_i$. Since γ_i is the multiplier associated with constraint (2.20), $\gamma_i > 0$ implies $\xi_i = 0$, establishing the sufficiency of $\lambda_i < C$. Notice that this is a sufficient condition, since both γ_i and λ_i could be equal to zero.

Note:

Our calculation above of the threshold value b assumes the existence of some λ_i such that $0 < \lambda_i < C$. We have not found a proof yet of the existence of such λ_i, or conditions under which it does not exist. However, we think this is a very reasonable assumption, because it is equivalent to the assumption that there is at least one support vector which is correctly classified. So far our computational results indicate that this assumption is correct, and we will use it in the rest of this thesis.

Nonlinear Decision Surfaces

Previous sections have only treated linear decision surfaces, which are definitely not appropriate for many tasks. The extension to more complex decision surfaces is conceptually quite simple, and is done by mapping the input variable $\mathbf{x}$ into a higher dimensional *feature space*, and by working with linear classification in that space. More precisely, one maps the input variable $\mathbf{x}$ into a (possibly infinite) vector of "feature" variables:

$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) = (a_1\phi(\mathbf{x}), a_2\phi_2(\mathbf{x}), \ldots, a_n\phi_n(\mathbf{x}), \ldots) \tag{2.33}$$

where $\{a_n\}_{n=1}^{\infty}$ are some real numbers and $\{\phi_n\}_{n=1}^{\infty}$ are some real functions[2]. The

[2]The numbers $\{a_n\}_{n=1}^{\infty}$ are clearly unnecessary, and could be absorbed into the definition of the

Soft Margin version of SVM is then applied, substituting the variable $\mathbf{x}$ with the new "feature vector" $\phi(\mathbf{x})$. Under the mapping (2.33) the solution of a SVM has the form:

$$f(\mathbf{x}) = \text{sign}\left(\phi(\mathbf{x}) \cdot \mathbf{w}^* + b^*\right) \Rightarrow \text{sign}\left(\sum_{i=1}^{\ell} y_i \lambda_i^* \phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i) + b^*\right) \tag{2.34}$$

A key property of the SV machinery is that the only quantities that one needs to compute are scalar products of the form $\phi(\mathbf{x}) \cdot \phi(\mathbf{y})$. It is therefore convenient to introduce the so-called *kernel function* K:

$$K(\mathbf{x}, \mathbf{y}) \equiv \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = \sum_{n=1}^{\infty} a_n^2 \phi_n(\mathbf{x}) \phi_n(\mathbf{y}) \tag{2.35}$$

Using this quantity the solution of a SVM has the form:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{\ell} y_i \lambda_i^* K(\mathbf{x}, \mathbf{x}_i) + b^*\right) \tag{2.36}$$

and the quadratic programming problem (2.30) becomes:

$$\begin{aligned} &\text{Maximize} \quad F(\mathbf{\Lambda}) = \mathbf{\Lambda} \cdot \mathbf{1} - \tfrac{1}{2} \mathbf{\Lambda} \cdot D\mathbf{\Lambda} \\ &\text{subject to} \\ &\qquad \mathbf{\Lambda} \cdot \mathbf{y} = 0 \\ &\qquad \mathbf{\Lambda} \leq C\mathbf{1} \\ &\qquad \mathbf{\Lambda} \geq \mathbf{0} \end{aligned} \tag{2.37}$$

where D is a symmetric, semi-positive definite, $\ell \times \ell$ matrix with elements $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$. Notice that the decision surface (2.36) is now a nonlinear function, given by linear superposition of kernel functions, one for each support vector. The idea of expanding the input space into a feature space is therefore useful only if we find some solution to the following problem starting from the feature space or starting

$\{\phi_n\}_{n=1}^{\infty}$, but we use them here because they make the formulation easier.

from the kernel.

Problem 2.3.1 *Find a set of coefficients* $\{a_n\}_{n=1}^{\infty}$ *and a set of features* $\{\phi_n\}_{n=1}^{\infty}$ *such that:*

1. *the scalar product* $K(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y})$ *is well defined (for example the series converges uniformly);*
2. *the scalar product* $K(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y})$ *is easy to compute as a function of* $\mathbf{x}$ *and* $\mathbf{y}$*;*

In addition to these requirements, we also should require the features ϕ_i to be such that the scalar product $K(\mathbf{x},\mathbf{y})$ defines a class of decision surfaces which is "rich" enough (for example includes some well-known approximation schemes). There are two different approaches to this problem.

Starting from the feature space

One approach consists of choosing carefully a set of features with "good" properties. For example, an obvious choice would be to take as features $\phi_i(\mathbf{x})$ monomials in the variable $\mathbf{x}$ up to a certain degree. Assuming, for simplicity, that we work in a one-dimensional space, one could choose:

$$\boldsymbol{\phi}(x) = (1, x, x^2, \ldots, x^d)$$

where d could be very large, and the coefficients a_i are all equal to one. In this case the decision surface is linear in the components of $\boldsymbol{\phi}$, and therefore a polynomial of degree d in x. This choice is unfortunate, however, because the scalar product

$$\boldsymbol{\phi}(x) \cdot \boldsymbol{\phi}(y) = 1 + xy + (xy)^2 + \ldots (xy)^d$$

is not particularly simple to compute when d is large. However, it is easy to see that, with a careful choice of the parameters a_i that things simplify. In fact, choosing

$$a_n = \binom{d}{n}$$

it is easy to see that

$$\phi(x) \cdot \phi(y) = \sum_{n=0}^{d} \binom{d}{n} (xy)^n = (1 + xy)^d$$

which considerably reduces the computation. A similar result, although with a more complex structure of the coefficients a_n, is true in the multivariable case, where the dimensionality of the feature space grows very quickly with the number of variables. For example, in two variables we can define:

$$\phi(\mathbf{x}) = (1, \sqrt{2}\, x_1, \sqrt{2}\, x_2, x_1^2, x_2^2, \sqrt{2}\, x_1 x_2) \tag{2.38}$$

In this case it is easy to see that:

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^2 \tag{2.39}$$

It is straightforward to extend this example to the d-dimensional case. For example, in three dimensions we have:

$$\phi(\mathbf{x}) = (1, \sqrt{2}\, x_1, \sqrt{2}\, x_2, \sqrt{2}\, x_3, x_1^2, x_2^2, x_3^2, \sqrt{2}\, x_1 x_2, \sqrt{2}\, x_1 x_3, \sqrt{2}\, x_2 x_3)$$

and the scalar product is still of the form of equation (2.39). Still in two dimensions we can use features which are monomials of degree three:

$$\phi(\mathbf{x}) = (1, \sqrt{3}\, x_1, \sqrt{3}\, x_2, \sqrt{3}\, x_1^2, \sqrt{3}\, x_2^2, \sqrt{6} x_1 x_2, \sqrt{3}\, x_1^2 x_2, \sqrt{3}\, x_1 x_2^2, x_1^3, x_2^3)$$

and it can be shown that:

$$K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^3$$

It can also be shown that if the features are monomials of degree less or equal to d, it is always possible to find numbers a_n in such a way that the scalar product is

$$K(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}\cdot\mathbf{y})^d \tag{2.40}$$

In the following we provide a few more examples of how one could choose the features first, and then, with a careful choice of the coefficients a_n, arrive at an analytical expression for the kernel K.

Infinite dimensional feature spaces

We consider one dimensional examples. Multidimensional kernels can be built using tensor products of one-dimensional kernels.

1. Let $x \in [0, \pi$] and let us consider the following feature space:

$$\phi(x) = (\sin(x), \frac{1}{\sqrt{2}}\sin(2x), \frac{1}{\sqrt{3}}\sin(3x), \ldots, \frac{1}{\sqrt{n}}\sin(nx), \ldots)$$

Then

$$K(x,y) = \phi(x)\cdot\phi(x) = \sum_{n=1}^{\infty}\frac{1}{n}\sin(nx)\sin(ny) = \frac{1}{2}\log\left|\frac{\sin\frac{x+y}{2}}{\sin\frac{x-y}{2}}\right|$$

which corresponds to the choice $a_n = \frac{1}{\sqrt{n}}$.

2. Let $x \in [0, 2\pi]$, h a positive number such that $h < 1$, and let us consider the following feature space:

$$\phi(x) = (1, h^{\frac{1}{2}}\sin(x), h^{\frac{1}{2}}\cos(x), h\sin(2x), h\cos(2x), \ldots, h^{\frac{n}{2}}\sin(nx), h^{\frac{n}{2}}\cos(nx), \ldots)$$

Then

$$K(x,y) = \phi(x)\cdot\phi(x) = 1 + \sum_{n=1}^{\infty}h^n\sin(nx)\sin(ny) + \sum_{n=1}^{\infty}h^n\cos(nx)\cos(ny) =$$

$$= \frac{1}{2\pi} \frac{1-h^2}{1-2h\cos(x-y)+h^2}$$

which corresponds to the choice $a_n = h^{\frac{n}{2}}$.

3. In the two examples above we have an infinite but countable number of features. We can also construct cases in which the number of features is infinite and uncountable. Let us consider the following map:

$$\phi(\mathbf{x}) = \left\{ \sqrt{\tilde{G}(\mathbf{s})} e^{i\mathbf{x}\cdot\mathbf{s}} \mid \mathbf{s} \in R^d \right\}$$

where $\tilde{G}(\mathbf{s})$ is the Fourier Transform of a positive definite function, and where we work, for simplicity, with complex features. This corresponds to a kernel

$$K(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = \int_{R^d} d\mathbf{s}\ \tilde{G}(\mathbf{s}) e^{i(\mathbf{x}-\mathbf{y})\cdot\mathbf{s}} = G(\mathbf{x}-\mathbf{y}).$$

which corresponds to a continuum of coefficients $a(\mathbf{s}) = \sqrt{\tilde{G}(\mathbf{s})}$.

Starting from the kernel

Another approach consists of looking for a kernel which is known to have a representation of the form (2.35) for some set of ϕ_i, but whose explicit analytic form may not be known.

In order to find a solution to this problem we need some preliminary facts. Let us call a *positive definite kernel* any function $K(\mathbf{x},\mathbf{y})$ on $\Omega \times \Omega$, with $\Omega \subset R^d$, with the property that:

$$\sum_{i,j=1}^{n} K(\mathbf{x}_i,\mathbf{x}_j) c_i c_j \geq 0 \quad \forall \mathbf{x}_i, \mathbf{x}_j \in \Omega\ , \quad \forall c_i, c_j \in R \tag{2.41}$$

In the following, we will assume that $\Omega \equiv [a,b]^d$. The kernel K defines an integral operator that is known to have a complete system of orthonormal eigenfunctions:

$$\int_\Omega K(\mathbf{x},\mathbf{y}) \phi_n(\mathbf{y})\ d\mathbf{y} = \lambda_n \phi_n(\mathbf{x}) \tag{2.42}$$

In 1976, Stewart [97] reported that, according to a theorem of Mercer from 1909 [62] the following statements are equivalent:

1. The function $K(\mathbf{x}, \mathbf{y})$ is a positive definite kernel;

2. $$\int_{[a,b]^d} K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y})\, d\mathbf{x} d\mathbf{y} \geq 0 \quad \forall g \in C([a,b]^d)$$

3. The eigenvalues λ_n in equation (2.42) are all positive;

4. The series

$$K(\mathbf{x}, \mathbf{y}) = \sum_{n=1}^{\infty} a_n^2 \phi_n(\mathbf{x}) \phi_n(\mathbf{y})$$

(where $a_n^2 = \frac{1}{\lambda_n}$) converges absolutely and uniformly.

This leads to the following:

Statement 2.3.1 *Any feature vector* $\boldsymbol{\phi}(\mathbf{x}) = (a_1\phi(\mathbf{x}), a_2\phi_2(\mathbf{x}), \ldots, a_n\phi_n(\mathbf{x}), \ldots)$ *such that the* $\{a_n\}_{n=1}^{\infty}$ *and the* $\{\phi_n\}_{n=1}^{\infty}$ *are respectively the eigenvalues and the eigenfunctions of a positive definite kernel* $K(\mathbf{x}, \mathbf{y})$ *will solve problem (2.3.1), and the scalar product* $\boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y})$ *has the following simple expression:*

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

A number of observations are in order:

- Vapnik (1995) uses the condition (2) above to characterize the kernels that can be used in a SVM. Definition (2.41) can be used instead, and might be more practical to work with if one has to prove the "admissibility" of a certain kernel.

- There is another result similar to Mercer's result, but is more general. Young (1909) proves that a kernel is positive definite if and only if

$$\int_{\Omega} K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y})\, d\mathbf{x} d\mathbf{y} \geq 0 \quad \forall g \in L_1(\Omega)$$

- The kernels K that can be used to represent a scalar product in the feature space are closely related to the theory of Reproducing Kernel Hilbert Spaces (RKHS) (see appendix A in (Girosi, 1997)[43]). In fact, in 1916 Moore [65] considers a more general setting for positive definite kernels, and replaces Ω in equation (2.41) with any abstract set E. He calls these functions *positive Hermitian matrices* and shows that for any such K one can associate a RKHS.

In table (2.1) we report some commonly used kernels.

Kernel Function	Type of Classifier
$K(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|^2)$	Gaussian RBF
$K(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}\cdot\mathbf{y})^d$	Polynomial of degree d
$K(\mathbf{x},\mathbf{y}) = \tanh(\mathbf{x}\cdot\mathbf{y}-\theta)$ (only for some values of θ)	Multi Layer Perceptron

Table 2.1: Some possible kernel functions and the type of decision surface they define.

2.3.2 Additional Geometrical Interpretation

Just as Figure (2-2) shows why better generalization is expected from maximizing the margin, one should wonder: do the support vectors have any geometrical common characteristic? Are they just scattered points used in a linear combination? It turns out that they are not.

In order to find the optimal decision surface, the support vector training algorithm tries to separate, as best as possible, the *clouds* defined by the data points from both classes.

Particularly, one would expect points closer to the *boundary* between the classes to be more important in the solution than data points that are far away, since the first are harder to classify. These data points, in some sense, help to shape and define better the decision surface than other points. Therefore, the support vectors are from a geometrical point of view *border points.*

A direct consequence of the previous argument delivers another important geometrical and algorithmic property, which is that, usually, the support vector are very

few.

These ideas can be justified algebraically through the optimality conditions derived in section 3.3.1.

Figure (2-3) shows examples of the preceding geometrical interpretations with polynomial and RBF classifiers.

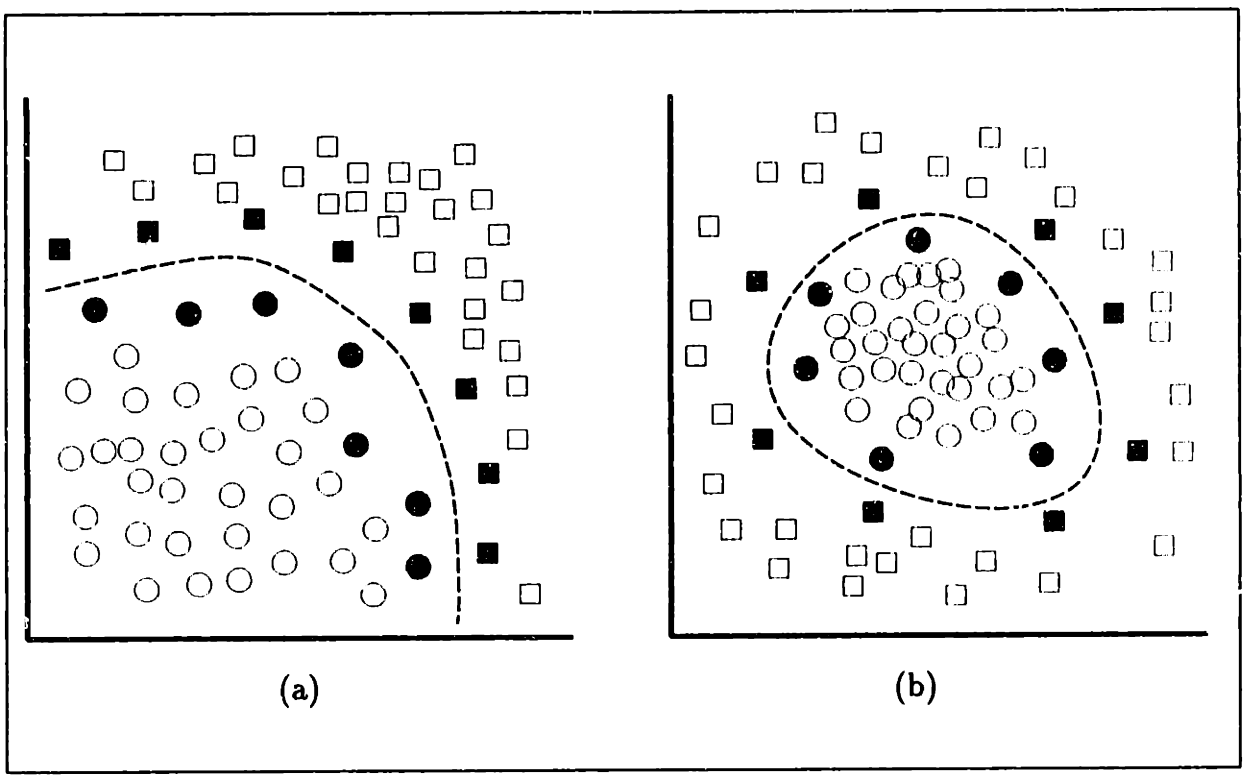


Figure 2-3: Decision Surfaces given in (a) by a polynomial classifier, and in (b) by a RBF, where the Support Vectors are indicated in dark fill. Notice the reduced number of them and their position close to the boundary. In (b), the Support Vectors are the RBF centers.

2.3.3 An Interesting Extension: A *Weighted* SVM

The original formulation of the SVM in the existing literature can be extended to handle two frequent cases in pattern classification and recognition:

- An unequal proportion of data samples between the classes.
- A need to *tilt the balance* or *weight* one class versus the other, which is very frequent when a classification error of one type is more expensive or undesirable than other.

The way to derive this extension is to allow equation (2.37) to be:

$$\begin{aligned} \text{Maximize} \quad & F(\mathbf{\Lambda}) = \mathbf{\Lambda} \cdot \mathbf{1} - \tfrac{1}{2}\mathbf{\Lambda} \cdot D\mathbf{\Lambda} \\ \text{subject to} \quad & \\ & \mathbf{\Lambda} \cdot \mathbf{y} = 0 \\ & \lambda_i \leq C^+\mathbf{1} \qquad \text{for } y_i = +1 \\ & \lambda_i \leq C^-\mathbf{1} \qquad \text{for } y_i = -1 \\ & \mathbf{\Lambda} \geq 0 \end{aligned} \tag{2.43}$$

where $\mathbf{y} = (y_1, \ldots, y_\ell)$, D is a symmetric $\ell \times \ell$ matrix with elements $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, and C^+ and C^- are positive constants.

Equation (2.18) for $k = 1$ now becomes:

$$min \;\; \Phi(\mathbf{w}, \mathbf{\Xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C^+\Big(\sum_{i:y_i=+1} \xi_i\Big) + C^-\Big(\sum_{i:y_i=-1} \xi_i\Big) \tag{2.44}$$

and equations (2.19) and (2.20) remain unchanged.

The quadratic program (2.43) can be interpreted as penalizing with higher penalty (C^+ or C^-) the most undesirable type of error through equation (2.44) . It is also important to notice that this extension has no real impact on the complexity of the problem of finding the optimal vector of multipliers $\mathbf{\Lambda}$, since only the bounding box constraints have changed.

Notice that this extension could be changed even further to allow, for example, higher values of C for highly reliable or valuable data points and lower values for data points of less confidence or value.

2.4 Support Vector Regression Machines

In this section we generalize the Support Vector Machine introduced for pattern classification in section 2.3, to regression and function estimation. The main idea in

this generalization is the use of a new type of *loss-function*, the *ϵ-insensitive* loss-function, whereby controlling ϵ is somewhat equivalent to controlling the margin in the pattern classification case.

In this section we first present some properties of the ϵ-insensitive loss-function and its relation to the Huber [49] robust loss-function. We then show a similar mathematical derivation to the one presented in section 2.3.1 for the pattern classification case, that also yields a linearly constrained convex QP. We end this section with a presentation of some additional insight and geometric interpretation of the technique.

2.4.1 The ϵ-Insensitive Loss-Function

As we stated in section 1.1.3, the function approximation problem can be formally stated as follows: Let $\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \Re^d \times \Re | i = 1, \ldots, \ell\}$ be a set of ℓ data points sampled from an unknown multivariate function $f(\mathbf{x})$, possibly in the presence of noise. The task is to recover the function $f(\mathbf{x})$, or at least a reasonable estimate of it, by means of an *approximation function* from a function class $F(\mathbf{\Lambda}, \mathbf{x})$, parameterized by the vector $\mathbf{\Lambda}$. For a fixed function class F, the problem is then to find the set of parameters $\mathbf{\Lambda}$ that best approximates $f(\mathbf{x})$ based on information from the set of data points $\mathcal{D}$.

Under conditions where $\mathbf{y}$ is the result of measuring a function with normally distributed additive noise ξ the Empirical Risk Minimization (ERM, see section 2.1) principle provides for the loss function

$$L(y, f(\mathbf{\Lambda}, x)) = (y - f(\mathbf{\Lambda}, x))^2 \tag{2.45}$$

the most efficient estimator. However, if the additive noise is generated by other laws, the best approximations are obtained using other loss-functions.

Huber [48] developed in 1964 a theory for finding the best loss-function for the problem under the ERM priciple, using only general information about the noise model.

In particular, Huber showed that if we only know that the density $p(\mathbf{x})$ describing

the noise ξ is a symmetric convex function with second derivatives, then the best approximation for the worst possible density $p(\mathbf{x})$ is given by the loss-function:

$$L(y, f(\Lambda, x)) = |y - f(\Lambda, x)|. \tag{2.46}$$

Under the ERM principle, this loss-function defines the *least-modulus* method and yields the *robust regression* function.

Another loss-function worth mention is the so called Huber loss-function, which is given by:

$$L(y, f(\Lambda, x), c) = \begin{cases} c|y - f(\Lambda, x)| - \frac{c^2}{2} & \text{if} \quad |y - f(\Lambda, x)| > c \\ \frac{1}{2}|y - f(\Lambda, x)|^2 & \text{otherwise.} \end{cases} \tag{2.47}$$

This function penalizes linearly the deviations larger than some parameter c, and quadratically the ones below c. It can be seen that this loss-function is somewhat less sensitive to outliers than (2.45).

Support Vector Regression Machines use a new type of loss-functions, the ϵ-insensitive loss-functions:

$$L(y, f(\Lambda, x)) = L(|y - f(\Lambda, x)|_\epsilon)$$

where:

$$|y - f(\Lambda, x)|_\epsilon = \begin{cases} 0 & \text{if} \quad |y - f(\Lambda, x)| \leq \epsilon \\ |y - f(\Lambda, x)| - \epsilon & \text{otherwise.} \end{cases} \tag{2.48}$$

Examples of these ϵ-insensitive loss-functions are given by:

- Linear ϵ-insensitive:

$$L(y, f(\Lambda, x)) = |y - f(\Lambda, x)|_\epsilon \tag{2.49}$$

- Quadratic ϵ-insensitive:

$$L(y, f(\mathbf{\Lambda}, x)) = |y - f(\mathbf{\Lambda}, x)|_\epsilon^2 \tag{2.50}$$

- Error-counting ϵ-insensitive:

$$L(y, f(\mathbf{\Lambda}, x)) = \begin{cases} 0 & \text{if } |y - f(\mathbf{\Lambda}, x)| \leq \epsilon \\ 1 & \text{otherwise.} \end{cases} \tag{2.51}$$

Using a similar approach as the one depicted in the given examples, one can consider other convex functions $L(y, f(\mathbf{\Lambda}, x))$. However, the Huber loss-function (2.47) and the linear (2.49) and the quadratic (2.50) ϵ-insensitive loss functions lead to quadratic programming problems nearly identical to the one derived for SVM pattern classifiers. The mathematical derivation of these quadratic programs is covered in the next section.

2.4.2 Mathematical Derivation

In this section we take a closer look at the mathematical derivation of the Support Vector Regression Machine, or SVRM. This section however, is not self-contained, as it is assumed that the reader has covered sections 2.1 - 2.3.

The technique is introduced by steps: we first consider approximations by linear functions using a cost function that embodies both complexity and risk minimization. This step is considered for linear and quadratic ϵ-insensitive loss functions, and for the Huber loss function. We then extend the formulation to non-linear functional forms, through the use of kernel functions, as we did for SVM classifiers.

Risk Minimization for linear functions

In this section we consider the approximations of the form

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} + b \tag{2.52}$$

defined so that:

- The complexity of the structure allowed for $f(\mathbf{x}, \mathbf{w})$ is controlled by the norm of $\mathbf{w}$ (i.e. $\|\mathbf{w}\|$), as in the SV classification case.
- A certain empirical risk measure is defined by the use of the linear and quadratic ϵ-insensitive loss functions, and by the Huber loss function.

The reason behind these two points is the simultaneous control over the training error and the complexity of the approximation. This aproach is an approximate implementation of the Structural Risk Minimization (SRM) principle described in section 2.2.

A mathematical formulation that achieves the described goal can be stated as:

$$\begin{aligned}
&\text{Minimize} \quad F(\mathbf{w}, \boldsymbol{\Xi}^*, \boldsymbol{\Xi}) = \tfrac{1}{2}\|\mathbf{w}\|^2 + \tfrac{C}{k}\Big(\sum_{i=1}^{\ell}(\xi_i^*)^k + (\xi_i)^k\Big) \\
&\mathbf{w}, \boldsymbol{\Xi}^*, \boldsymbol{\Xi}, b \\
&\text{subject to} \\
&\qquad y_i - (\mathbf{w}^T\mathbf{x}_i) - b \le \epsilon + \xi_i^* \qquad i = 1 \ldots \ell \qquad (2.53) \\
&\qquad -y_i + (\mathbf{w}^T\mathbf{x}_i) + b \le \epsilon + \xi_i \qquad i = 1 \ldots \ell \\
&\qquad \boldsymbol{\Xi}^*, \boldsymbol{\Xi} \ge \mathbf{0} \\
&\qquad \mathbf{w}, b \quad \textit{free}
\end{aligned}$$

where:

- C is again a positive user-defined constant that controls the tradeoff between functional complexity and training error.
- ϵ is a positive user-defined constant that corresponds to the *insensitive* region of the loss function.
- $k = 1$ gives the linear ϵ-insensitive loss function.

- $k = 2$ gives the quadratic ϵ-insensitive loss function.

By constructing the Lagrangian function:

$$\begin{aligned} L(\mathbf{w}, b, \mathbf{\Lambda}^*, \mathbf{\Lambda}, \mathbf{\Xi}^*, \mathbf{\Xi}, \mathbf{\Gamma}^*, \mathbf{\Gamma}) &= \tfrac{1}{2}\|\mathbf{w}\|^2 + \tfrac{C}{k}(\sum_{i=1}^{\ell}(\xi_i^*)^k + (\xi_i)^k) + \\ &\quad -\sum_{i=1}^{\ell}\lambda_i^*[(\mathbf{w}^T\mathbf{x}_i) + b - y_i + \epsilon + \xi_i^*] + \\ &\quad -\sum_{i=1}^{\ell}\lambda_i[y_i - (\mathbf{w}^T\mathbf{x}_i) - b + \epsilon + \xi_i] - \sum_{i=1}^{\ell}(\gamma_i^*\xi_i^* + \gamma_i\xi_i) \end{aligned} \tag{2.54}$$

and minimizing with respect to $\mathbf{w}, \mathbf{\Xi}, \mathbf{\Xi}^*$ and b, and maximizing with respect to the Lagrange multipliers $\mathbf{\Lambda}, \mathbf{\Lambda}^*, \mathbf{\Gamma}$ and $\mathbf{\Gamma}^*$ one obtains an expansion $\mathbf{w} = \sum_{i=1}^{\ell}(\lambda_i^* - \lambda_i)\mathbf{x}_i$, where $\mathbf{\Lambda}, \mathbf{\Lambda}^*$ are found by solving the following quadratic programming problems:

- Case $k = 1$:

$$\begin{aligned} \text{Maximize} \quad F(\mathbf{\Lambda}, \mathbf{\Lambda}^*) &= -\epsilon\sum_{i=1}^{\ell}(\lambda_i^* + \lambda_i) + \sum_{i=1}^{\ell} y_i(\lambda_i^* - \lambda_i) + \\ \mathbf{\Lambda}, \mathbf{\Lambda}^* &\quad -\tfrac{1}{2}\sum_{i,j=1}^{\ell}(\lambda_i^* - \lambda_i)(\lambda_j^* - \lambda_j)(\mathbf{x}_i^T\mathbf{x}_j) \end{aligned}$$

subject to

$$\begin{aligned} \sum_{i=1}^{\ell}(\lambda_i^* - \lambda_i) &= 0 \\ \mathbf{\Lambda}^*, \mathbf{\Lambda} &\geq \mathbf{0} \\ \mathbf{\Lambda}^*, \mathbf{\Lambda} &\leq C\mathbf{1} \end{aligned} \tag{2.55}$$

- Case $k = 2$:

$$
\begin{aligned}
\text{Maximize} \quad F(\mathbf{\Lambda}, \mathbf{\Lambda}^*) \quad &= \quad -\epsilon \sum_{i=1}^{\ell} (\lambda_i^* + \lambda_i) + \sum_{i=1}^{\ell} y_i (\lambda_i^* - \lambda_i) + \\
\mathbf{\Lambda}, \mathbf{\Lambda}^* \quad & \qquad -\tfrac{1}{2} \sum_{i,j=1}^{\ell} (\lambda_i^* - \lambda_i)(\lambda_j^* - \lambda_j)(\mathbf{x}_i^T \mathbf{x}_j) + \\
& \qquad -\tfrac{1}{2C} \sum_{i=1}^{\ell} ((\lambda_i^*)^2 + (\lambda_i)^2)
\end{aligned}
\tag{2.56}
$$

subject to

$$
\begin{aligned}
\sum_{i=1}^{\ell} (\lambda_i^* - \lambda_i) \quad &= \quad 0 \\
\mathbf{\Lambda}^*, \mathbf{\Lambda} \quad &\geq \quad \mathbf{0} \\
\mathbf{\Lambda}^*, \mathbf{\Lambda} \quad &\leq \quad C\mathbf{1}
\end{aligned}
$$

In order to apply the Huber loss function, we must solve the following mathematical program:

$$
\text{Minimize} \quad F(\mathbf{w}, \mathbf{\Xi}^*, \mathbf{\Xi}) \quad = \tfrac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} (L(\xi_i^*) + L(\xi_i))
$$

$\mathbf{w}, \mathbf{\Xi}^*, \mathbf{\Xi}, b$

subject to

$$
\begin{aligned}
y_i - (\mathbf{w}^T \mathbf{x}_i) - b \quad &\leq \xi_i^* \qquad\qquad i = 1 \ldots \ell \\
-y_i + (\mathbf{w}^T \mathbf{x}_i) + b \quad &\leq \xi_i \qquad\qquad i = 1 \ldots \ell \\
\mathbf{\Xi}^*, \mathbf{\Xi} \quad &\geq \mathbf{0} \\
\mathbf{w}, b \quad &free
\end{aligned}
\tag{2.57}
$$

where:

$$L(\xi) = \begin{cases} q|\xi| - \frac{q^2}{2} & \text{for} |\xi| > q \\ \frac{1}{2}\xi^2 & \text{otherwise.} \end{cases} \tag{2.58}$$

As in the previous cases, after constructing the Lagrangian and optimizing accordingly with respect to its parameters, one obtains the following QP:

$$\begin{aligned} \text{Maximize} \quad F(\mathbf{\Lambda}, \mathbf{\Lambda}^*) \quad &= \quad \sum_{i=1}^{\ell} y_i(\lambda_i^* - \lambda_i) + \\ \mathbf{\Lambda}, \mathbf{\Lambda}^* \qquad &\quad -\tfrac{1}{2}\sum_{i,j=1}^{\ell}(\lambda_i^* - \lambda_i)(\lambda_j^* - \lambda_j)(\mathbf{x}_i^T\mathbf{x}_j) + \\ &\quad -\tfrac{q}{2C}\sum_{i=1}^{\ell}((\lambda_i^*)^2 + (\lambda_i)^2) \end{aligned} \tag{2.59}$$

subject to

$$\begin{aligned} \sum_{i=1}^{\ell}(\lambda_i^* - \lambda_i) &= 0 \\ \mathbf{\Lambda}^*, \mathbf{\Lambda} &\geq \mathbf{0} \\ \mathbf{\Lambda}^*, \mathbf{\Lambda} &\leq C\mathbf{1} \end{aligned}$$

Note: Of the three loss functions presented in this section, we have decided to use (as almost all the other researchers in the field have, too) and further investigate the linear ϵ-insensitive loss function, both because of its simplicity and the very good results that it has shown in preliminary tests and experiments. Therefore, from now on, when we refer to SVRMs, we will be talking about the linear ϵ-insensitive version, unless we state otherwise.

Extending the SVRM to nonlinear functions

As in the pattern classification case, the main idea of this vital extension is the mapping of the input vectors $\mathbf{x}$ into a high dimensional feature space where we then

consider a linear function, as described in the preceeding section. In order to achieve this mapping implicitly, we use (again, as in the pattern classification case) kernel functions that represent the dot product operation in this high dimensional feature space. Therefore, the approximation form is now decribed by:

$$f(\mathbf{x}, \beta) = \sum_{i=1}^{\ell} \beta_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{2.60}$$

where $\beta_i, i = 1, \ldots, \ell$, are coefficients that are found by solving correponding QPs. In the case of the linear ϵ-insensitive loss function, this QP is:

$$\begin{array}{lll} \text{Maximize} \quad F(\mathbf{\Lambda}, \mathbf{\Lambda}^*) & = & -\epsilon \sum_{i=1}^{\ell} (\lambda_i^* + \lambda_i) + \sum_{i=1}^{\ell} y_i (\lambda_i^* - \lambda_i) + \\ \mathbf{\Lambda}, \mathbf{\Lambda}^* & & -\frac{1}{2} \sum_{i,j=1}^{\ell} (\lambda_i^* - \lambda_i)(\lambda_j^* - \lambda_j) K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to} & & \\ \sum_{i=1}^{\ell} (\lambda_i^* - \lambda_i) & & \\ \mathbf{\Lambda}^*, \mathbf{\Lambda} & \geq & \mathbf{0} \\ \mathbf{\Lambda}^*, \mathbf{\Lambda} & \leq & C\mathbf{1} \end{array} \tag{2.61}$$

and the corresponding coefficients in (2.60) are of the form $\beta_i = \lambda_i^* - \lambda_i$.

Similar replacement of the Euclidean dot product $\mathbf{x}_i^T \mathbf{x}_j$ by the kernel dot product $K(\mathbf{x}_i, \mathbf{x}_j)$ can be done in the QPs (2.56) and (2.59).

A short note on Kernels for SVRMs

To build different types of approximations, one has to use different kernels satisfying Mercer's condition. In particular, one can use the same kernels that were described in section 2.3.1 (Table 2.1 on page 56) for pattern classification.

However, regression is more delicate than pattern classification, mainly because the *sign*() operator simplifies somewhat the approximation task. Therefore, the se-

lection of the kernel function to be used is more delicate and must be given special consideration. Some of the kernels that have been used in regression include: Hermite, Chebyshev and Legendre polynomials, semi-local approximations, splines and B-splines, Fourier expansions, Dirichlet kernels, Anova kernels, etc. More detail on these and other kernels, as well as a description on how to build multi-dimensional kernels from one-dimensional kernels, can be found in [108].

2.4.3 Geometrical Interpretation

Just as section 2.3.2 shows a geometrical interpretation for the support vectors in the pattern classification case, one should wonder: do the support vectors in the regression case have any common characteristic? Are they just scattered points used in a linear combination? It turns out that they are not.

In order to find the optimal approximation for the data points, the SVM uses the ϵ-insensitive area to approximately fit them within an *ϵ-tube*. Data points that force an *inflection* in the approximating function, that is, points that are barely fit correctly within the $\pm\epsilon$ range, correspond to support vectors. Also, data points whose approximation lies outside the *ϵ-tube* correspond to support vectors with coefficient $(\lambda_i^* - \lambda_i) = \pm C$.The sign of this coefficient depends on whether the violation is below or above the *ϵ-tube*, respectively.

Figure (2-4) shows an example of this geometrical interpretation. In particular we remark:

- Points A,B,C,E are the support vectors. The other points (like D, for example) have $\lambda_i^* = \lambda_i = 0$, and therefore are not support vectors since their coefficients are $(\lambda_i^* - \lambda_i) = 0$.
- Point A corresponds to a SV with $\xi_i > 0$, $\lambda_i^* = 0$, $\lambda_i = C$, and a coefficient $(\lambda_i^* - \lambda_i) = -C$. Notice that ξ_i is measured from the *ϵ-tube* and not from the observed function value.
- Point B corresponds to a SV with $\xi_i = 0$, $\lambda_i = 0$, and generally (ruling out degenerate cases) $0 < \lambda_i^* < C$.

- Point C corresponds to a SV with $\xi_i = 0$, $\lambda_i^* = 0$, and generally (again, ruling out degenerate cases) $0 < \lambda_i < C$.

- Point E corresponds to a SV with $\xi_i^* > 0$, $\lambda_i = 0$, $\lambda_i^* = C$, and a coefficient $(\lambda_i^* - \lambda_i) = C$.

These ideas can be justified algebraically through the optimality conditions derived in section 3.5.1.

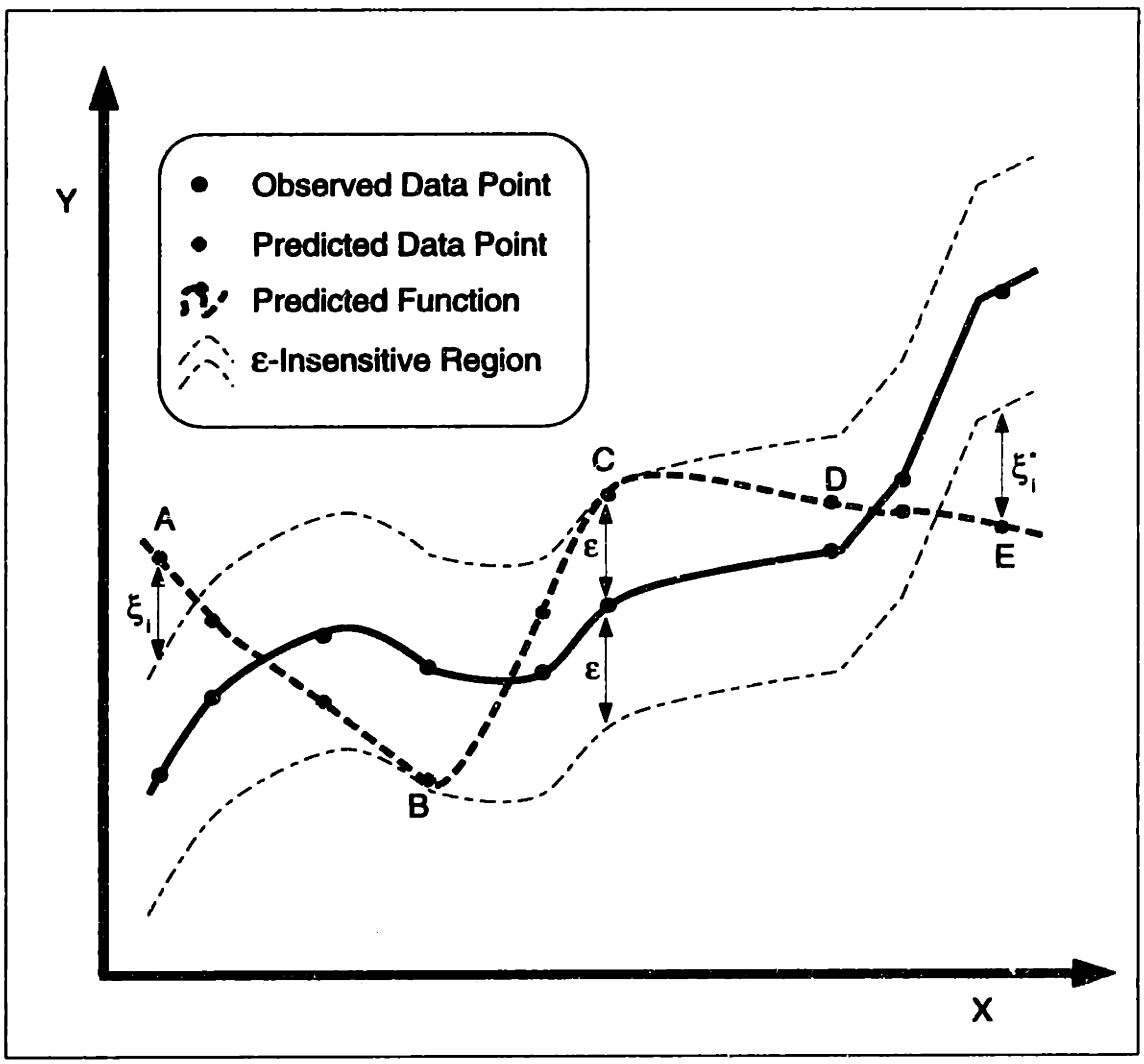


Figure 2-4: Geometrical interpretation of the support vectors in the regression case. Points A,B,C and E are support vectors. Notice how these are points at which the estimated function lies in the boundary of the ϵ-*tube*, or beyond.

Chapter 3

Training a Support Vector Machine

In the case of pattern classification, solving the quadratic program (2.37) (see page 50) determines the desired decision surface given by equation (2.36).

Analogously, the problem of function approximation requires the solution of the QP given by (2.61) (see page 66) and defines the approximation form given by equation (2.60).

This optimization process is referred to as *training a Support Vector Machine*. This chapter covers previous, current, and possible future approaches to solving this problem.

Our presentation in this chapter starts by addressing the training problem associated to pattern classification. We later extend the described approaches to the regression problem. The outline of this chapter is as follows: Section 3.1 is intended to give the reader an idea of the difficulties associated with the problem of training SVMs. Section3.2 deals with approaches to solving *small* training problems, both because they constitute a natural first step, and also because the decomposition algorithms described in sections 3.3, 3.4 and 3.5 iteratively solve *small* subproblems of the type given by (2.37) and (2.61). In section 3.6 we discuss possible inprovements and future research directions.

3.1 Why is this problem Hard?

One important characteristic of (2.37) is that the quadratic form matrix D that appears in the objective function (even though symmetric) is completely dense and with size (i.e., the number of entries) square in the number of data vectors. This fact implies that due to memory and computational constraints, problems with large data sets (above $\approx$ 5,000 samples) cannot be solved without some kind of data and problem decomposition.

3.2 Initial experiments

During the process of this research, the training problem for small data sets was initially approached with three different algorithms and three computer packages: conjugate gradient, Zoutendijk's [117][2] method of feasible directions, (using CPLEX to solve the LP's), GAMS/MINOS (using GAMS as the modeling language and MINOS 5.4 as the solver), and a second-order variant of the reduced gradient method (algorithm implemented in MINOS 5.4). A summary of Zoutendijk's method and the reduced gradient method can be found in Appendix A. Some interesting notes and computational results on these experiments are given below:

Zoutendijk's Method:

One interesting modification that was done to this algorithm in order to help its speed in the computer implementation was to solve the problem several times with an increasing upper bound C. The starting value of C was usually very low, and it was scaled several times until it reached the original value. The solutions were also scaled and used as a starting point for the following iteration.

From a computational point of view, this method behaved a lot better than a *naive* constrained conjugate gradient implementation, both in terms of speed and graceful degradation with the increase of C.

On the other hand, this implementation had serious difficulties in cases where most of the λ_i's were strictly between their bounds. The zigzagging and slow convergence it

presented allowed GAMS/MINOS and MINOS 5.4 to outperform it by several orders of magnitude.

GAMS/MINOS:

GAMS is a modeling language that allows fast description and maintainability of optimization problems. As a language, GAMS *generates* the specified model and calls a user-specified *solver*, depending on the type of problem at hand. In the case of nonlinear programs, MINOS is one of these *solvers*.

The work done with GAMS/MINOS was very important. At the beginning, it offered a verification of the implementation of the conjugate gradient and Zoutendijk's method and a point of comparison in terms of speed and accuracy, but most important, it later pointed to the idea of using MINOS 5.4 directly, without the overhead that GAMS could represent.

Another reason for considering important the work done with GAMS/MINOS was the improvement in the training speed due to a problem reformulation given by:

$$\begin{aligned} \text{Maximize} \quad & F(\mathbf{\Lambda}, \mathbf{\Omega}) = \mathbf{\Lambda} \cdot \mathbf{1} - \tfrac{1}{2}\mathbf{\Lambda} \cdot \mathbf{\Omega} \\ & \mathbf{\Lambda}, \mathbf{\Omega} \\ \text{subject to} \quad & \\ & \mathbf{\Lambda} \cdot \mathbf{y} = 0 \\ & D\mathbf{\Lambda} = \mathbf{\Omega} \\ & \mathbf{\Lambda} \leq C\mathbf{1} \\ & \mathbf{\Lambda} \geq \mathbf{0} \end{aligned} \tag{3.1}$$

Although strange at first sight, this transformation allows a much faster function and gradient evaluation, and was responsible for an important speedup in both steps of the solution (model generation and optimization). This was enough reason to use it as the formulation when using MINOS 5.4.

MINOS 5.4:

MINOS 5.4 solves nonlinear problems with linear constraints using Wolfe's Reduced Gradient algorithm in conjunction with Davidson's quasi-Newton method. Details of its implementation are described by Murtagh and Saunders in [71], in MINOS 5.4 User's Guide [72], and in Gill *et al.* [42]. Bazaraa *et al.* [2] present an overview with some heuristics and comparisons.

Computational Results

In order to compare the relative speed between these methods, two different problems with small data-sets were solved in the same computational environment:

1. Training a SVM with a linear classifier in the Ripley data-set. This data-set consists of 250 samples in two dimensions which are not linearly separable. Table 3.1 shows the following points of comparison:

   - The difference between GAMS/MINOS used in the original problem and in the transformed version (3.1).
   - The performance degradation suffered by the conjugate gradient implementation under the increase of the upper bound C, and on the opposite hand, the negligible effect on GAMS/MINOS (modified) and MINOS 5.4.
   - A considerable advantage in performance by MINOS 5.4.

2. Training a SVM with a third degree polynomial classifier on the Sonar data-set. This data-set consists of 208 samples in 60 dimensions which are not linearly separable, but are polynomially separable. The results of these experiments are shown in Table 3.2 and exhibit the following points of comparison:

   - The difficulty experienced by first-order methods like Zoutendijk's method to converge when the values of the λ_i's are strictly between the bounds.
   - The clear advantage in solving the problem directly with MINOS 5.4, removing the overhead created by GAMS and incorporating the knowledge of the problem into the solution process through, for example, fast and exact gradient evaluation, use of symmetry in the constraint matrix, etc.

- Again, a negligible effect of the upper bound C on the performance, when using MINOS.

An important computational result is the *sub-linear* dependence of the training time with the dimensionality of the input data. In order to show this dependence, Table 3.3 presents the training time for randomly-generated 2,000 data-points problems, with different dimensionality, separability, and upper bound C.

	Methods				
C	Conj. Grad.	Zoutendijk	GAMS/MINOS	GAMS/MINOS Mod.	MINOS 5.4
10	23.9 sec	12.4 sec	906 sec	17.6 sec	1.2 sec
100	184.1 sec	37.9 sec	1068 sec	19.7 sec	1.4 sec
10000	5762.2 sec	161.5 sec	1458 sec	22.6 sec	2.3 sec

Table 3.1: Training time on the Ripley data-set for different methods and upper bound C. GAMS/MINOS Mod corresponds to the reformulated version of the problem.

	Methods		
C	Zoutendijk	GAMS/MINOS Modified	MINOS 5.4
10	4381.2 sec	67.0 sec	3.3 sec
100	N/A	67.1 sec	3.3 sec
10000	N/A	67.1 sec	3.3 sec

Table 3.2: Training time on the Sonar Dataset for different methods and upper bound C.

	Dimension					
	Separable			Non-Separable		
C	4	16	256	4	16	256
10	60.7 sec	106.4 sec	613.5 sec	292.9 sec	476.0 sec	1398.2 sec
100	36.0 sec	69.2 sec	613.7 sec	313.5 sec	541.0 sec	2369.4 sec
10000	21.8 sec	56.2 sec	623.0 sec	327.4 sec	620.6 sec	3764.1 sec

Table 3.3: Training time on a Randomly-generated Dataset for different dimensionality and upper bound C.

3.3 A First Decomposition Approach for Large Database Training

As mentioned before, training a SVM using large data sets (above $\approx$ 5,000 samples) is a very difficult problem to approach without some kind of data or problem decomposition. To give an idea of some memory requirements, an application like the one described later in section 6.1 involves 50,000 training samples, and this amounts to a quadratic form whose matrix D has $2.5 \cdot 10^9$ entries that would need, using an 8-byte floating point representation, 20,000 Megabytes = 20 Gigabytes of memory!

In order to solve the training problem efficiently, we take explicit advantage of the geometric interpretation introduced in Section 2.3.2, in particular, the expectation that the number of support vectors will be very few. If we consider the quadratic programming problem given by (2.37), this expectation translates into having many of the components of $\boldsymbol{\Lambda}$ equal to zero.

In order to decompose the original problem, one can think of solving iteratively the system given by (2.37), but keeping fixed at zero level, those components λ_i associated with data points that are not support vectors, and therefore only optimizing over a reduced set of variables.

To convert the previous description into an algorithm we need to specify:

1. **Optimality Conditions**: These conditions allow us to decide computationally if the problem has been solved optimally at a particular global iteration of the original problem. Section 3.3.1 states and proves optimality conditions for the QP given by (2.37).

2. **Strategy for Improvement**: If a particular solution is not optimal, this strategy defines a way to improve the cost function and is frequently associated with variables that violate optimality conditions. This strategy will be stated in section 3.3.2.

After presenting optimality conditions and a strategy for improving the cost function, section 3.3.3 introduces a decomposition algorithm that can be used to solve

large database training problems, and section 3.3.4 reports some computational results obtained with its implementation.

3.3.1 Optimality Conditions

In order to be consistent with common standard notation for nonlinear optimization problems, the quadratic program (2.37) can be rewritten in minimization form as:

$$\begin{aligned} \text{Minimize} \quad & W(\boldsymbol{\Lambda}) = -\boldsymbol{\Lambda} \cdot \mathbf{1} + \tfrac{1}{2}\boldsymbol{\Lambda} \cdot D\boldsymbol{\Lambda} \\ \boldsymbol{\Lambda} & \\ \text{subject to} & \\ & \boldsymbol{\Lambda} \cdot \mathbf{y} = 0 \qquad (\mu) \\ & \boldsymbol{\Lambda} - C\mathbf{1} \leq \mathbf{0} \qquad (\boldsymbol{\Upsilon}) \\ & -\boldsymbol{\Lambda} \leq \mathbf{0} \qquad (\boldsymbol{\Pi}) \end{aligned} \tag{3.2}$$

where μ, $\boldsymbol{\Upsilon} = (\upsilon_1, \ldots, \upsilon_\ell)$ and $\boldsymbol{\Pi} = (\pi_1, \ldots, \pi_\ell)$ are the associated Karush-Kuhn-Tucker multipliers.

Since D is a positive semi-definite matrix (see end of section 2.3.1) and the constraints in (3.2) are linear, the Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality, and they are:

$$
\begin{aligned}
\nabla W(\boldsymbol{\Lambda}) + \boldsymbol{\Upsilon} - \boldsymbol{\Pi} + \mu \mathbf{y} &= \mathbf{0} \\
v_i \cdot (\lambda_i - C) &= 0 \quad i = 1, \ldots, \ell \\
\pi_i \cdot \lambda_i &= 0 \quad i = 1, \ldots, \ell \\
\boldsymbol{\Upsilon} &\geq \mathbf{0} \\
\boldsymbol{\Pi} &\geq \mathbf{0} \\
\boldsymbol{\Lambda} \cdot \mathbf{y} &= 0 \\
\boldsymbol{\Lambda} - C\mathbf{1} &\leq \mathbf{0} \\
-\boldsymbol{\Lambda} &\leq \mathbf{0}
\end{aligned}
\tag{3.3}
$$

In order to derive further algebraic expressions from the optimality conditions (3.3), we assume the existence of some λ_i such that $0 < \lambda_i < C$ (see end of section 2.3.1), and consider the three possible values that each component of $\boldsymbol{\Lambda}$ can have:

1. **Case:** $0 < \lambda_i < C$**:**

   From the first three equations of the KKT conditions we have:

$$
(D\boldsymbol{\Lambda})_i - 1 + \mu y_i = 0 \tag{3.4}
$$

   Noticing that

$$
(D\boldsymbol{\Lambda})_i = \sum_{j=1}^{\ell} \lambda_j y_j y_i K(\mathbf{x}_i, \mathbf{x}_j) \;=\; y_i \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j)
$$

   and that for $0 < \lambda_i < C$,

$$
f(\mathbf{x}_i) \;=\; sign(\sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b) \;=\; \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b \;=\; y_i
$$

we obtain the following:

$$(D\Lambda)_i = \frac{y_i - b}{y_i} = 1 - \frac{b}{y_i} \tag{3.5}$$

By substituting (3.5) into (3.4) we finally obtain that

$$\mu = b \tag{3.6}$$

Therefore, at an optimal solution Λ^*, the value of the multiplier μ is equal to the optimal threshold b^*.

2. **Case:** $\lambda_i = C$:

From the first three equations of the KKT conditions we have:

$$(D\Lambda)_i - 1 + v_i + \mu y_i = 0 \tag{3.7}$$

By defining

$$g(\mathbf{x}_i) = \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b$$

and noticing that

$$(D\Lambda)_i = y_i \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) = y_i g(\mathbf{x}_i) - y_i b$$

equation (3.7) can be written as:

$$y_i g(\mathbf{x}_i) - y_i b - 1 + v_i + \mu y_i = 0$$

By combining $\mu = b$ (derived from case 1) and requiring $v_i \geq 0$ we finally obtain:

$$y_i g(\mathbf{x}_i) \leq 1 \tag{3.8}$$

3. **Case:** $\lambda_i = 0$:

   From the first three equations of the KKT conditions we have:

$$(D\mathbf{\Lambda})_i - 1 - \pi_i + \mu y_i = 0 \tag{3.9}$$

   By applying a similar algebraic manipulation as the one described for case 2, we obtain

$$y_i g(\mathbf{x}_i) \geq 1 \tag{3.10}$$

3.3.2 Strategy for Improvement

In order to incorporate the optimality conditions and the expectation that most λ_i's will be zero into an algorithm, we need to derive a way to improve the objective function value using this information. To do this, let us decompose $\mathbf{\Lambda}$ in two vectors $\mathbf{\Lambda}_B$ and $\mathbf{\Lambda}_N$, where $\mathbf{\Lambda}_N = 0$, and B and N partition the index set, and that the optimality conditions hold in a subproblem defined only for the variables in B. In further sections, the set B will be referred to as the *working set.* Under this decomposition the following statements are clearly true:

- We can replace $\lambda_i = 0$, $i \in B$, with $\lambda_j = 0$, $j \in N$, without changing the cost function or the feasibility of both the subproblem and the original problem.
- After such a replacement, the new subproblem is optimal if and only if $y_j g(\mathbf{x}_j) \geq 1$. This follows from equation (3.10) and the assumption that the subproblem was optimal before the replacement was done.

The previous statements suggest that replacing variables at zero levels in the subproblem, with variables $\lambda_j = 0$, $j \in N$ that violate the optimality condition

$y_j g(\mathbf{x}_j) \geq 1$, yields a subproblem that, when optimized, improves the cost function while maintaining feasibility. The following proposition states this idea formally.

Proposition 3.3.1 *Given an optimal solution of a subproblem defined on B (and assuming as indicated before the existence of some λ_i such that $0 < \lambda_i < C$) , the operation of replacing $\lambda_i = 0$, $i \in B$, with $\lambda_j = 0$, $j \in N$, satisfying $y_j g(\mathbf{x}_j) < 1$ generates a new subproblem that, when optimized, yields a strict improvement of the objective function $W(\mathbf{\Lambda})$.*

Proof: We assume again the existence of λ_p [1] such that $0 < \lambda_p < C$. Let us also assume without loss of generality that $y_p = y_j$ (the proof is analogous if $y_p = -y_j$). Then, there is some $\epsilon > 0$ such that $\lambda_p - \delta > 0$, for $\delta \in (0, \epsilon)$. Notice also that $g(\mathbf{x}_p) = y_p$. Now, consider $\overline{\mathbf{\Lambda}} = \mathbf{\Lambda} + \delta e_j - \delta e_p$, where e_j and e_p are the jth and pth unit vectors, and notice that the pivot operation can be handled implicitly by letting $\delta > 0$ and by holding $\lambda_i = 0$. The new cost function $W(\overline{\mathbf{\Lambda}})$ can be written as:

$$
\begin{aligned}
W(\overline{\mathbf{\Lambda}}) \;=\; & -\overline{\mathbf{\Lambda}} \cdot 1 + \frac{1}{2}\overline{\mathbf{\Lambda}} \cdot D\overline{\mathbf{\Lambda}} \\
& -\mathbf{\Lambda} \cdot 1 + \frac{1}{2}\left[\mathbf{\Lambda} \cdot D\mathbf{\Lambda} + 2\mathbf{\Lambda} \cdot D(\delta e_j - \delta e_p) + (\delta e_j - \delta e_p) \cdot D(\delta e_j - \delta e_p)\right] \\
& W(\mathbf{\Lambda}) + \delta\left[\frac{g(\mathbf{x}_j) - b}{y_j} - 1 + \frac{b}{y_p}\right] + \frac{\delta^2}{2}\left[K(\mathbf{x}_j, \mathbf{x}_j) + K(\mathbf{x}_p, \mathbf{x}_p) - 2y_p y_j K(\mathbf{x}_p, \mathbf{x}_j)\right] \\
& W(\mathbf{\Lambda}) + \delta\left[g(\mathbf{x}_j) y_j - 1\right] + \frac{\delta^2}{2}\left[K(\mathbf{x}_j, \mathbf{x}_j) + K(\mathbf{x}_p, \mathbf{x}_p) - 2y_p y_j K(\mathbf{x}_p, \mathbf{x}_j)\right]
\end{aligned}
$$

Therefore, since $g(\mathbf{x}_j) y_j < 1$, by choosing δ small enough we have $W(\overline{\mathbf{\Lambda}}) < W(\mathbf{\Lambda})$.

q.e.d

[1] Notice that given the problem equality constraint, this also assumes the existence of λ_q such that $0 < \lambda_q < C$ with $y_q = -y_p$

3.3.3 The Algorithm and its Geometric Interpretation

Suppose we can define a fixed-size working set B, such that $|B| \leq \ell$, and it is big enough to contain all support vectors ($\lambda_i > 0$), but small enough such that the computer can handle it and optimize it using some solver. Then the decomposition algorithm can be stated as follows:

1. Arbitrarily chocse $|B|$ points from the data set including both classes.
2. Solve the subproblem defined by the variables in B.
3. While there exists some $j \in N$, such that $g(\mathbf{x}_j)y_j < 1$, where

$$g(\mathbf{x}_j) = \sum_{p=1}^{\ell} \lambda_p y_p K(\mathbf{x}_j, \mathbf{x}_p) + b$$

replace $\lambda_i = 0$, $i \in B$, with $\lambda_j = 0$ and solve the new subproblem.

Notice that this algorithm will strictly improve the objective function at each iteration and therefore will not cycle. Since the objective function is bounded ($W(\mathbf{\Lambda})$ is convex and quadratic, and the feasible region is bounded), the algorithm must converge to the global optimal solution in a finite number of iterations. Figure 3-1 gives a geometric interpretation of the way the decomposition algorithm allows the redefinition of the separating surface by adding points that violate the optimality conditions.

3.3.4 Computational Implementation and Results

We have implemented the decomposition algorithm using the transformed problem defined by equation (3.1) and MINOS 5.4 as the solver.

Notice that the decomposition algorithm is rather flexible about the pivoting strategy, that is, the way it decides which and how many new points to incorporate into the working set B. Our implementation uses two parameters to define the desired strategy:

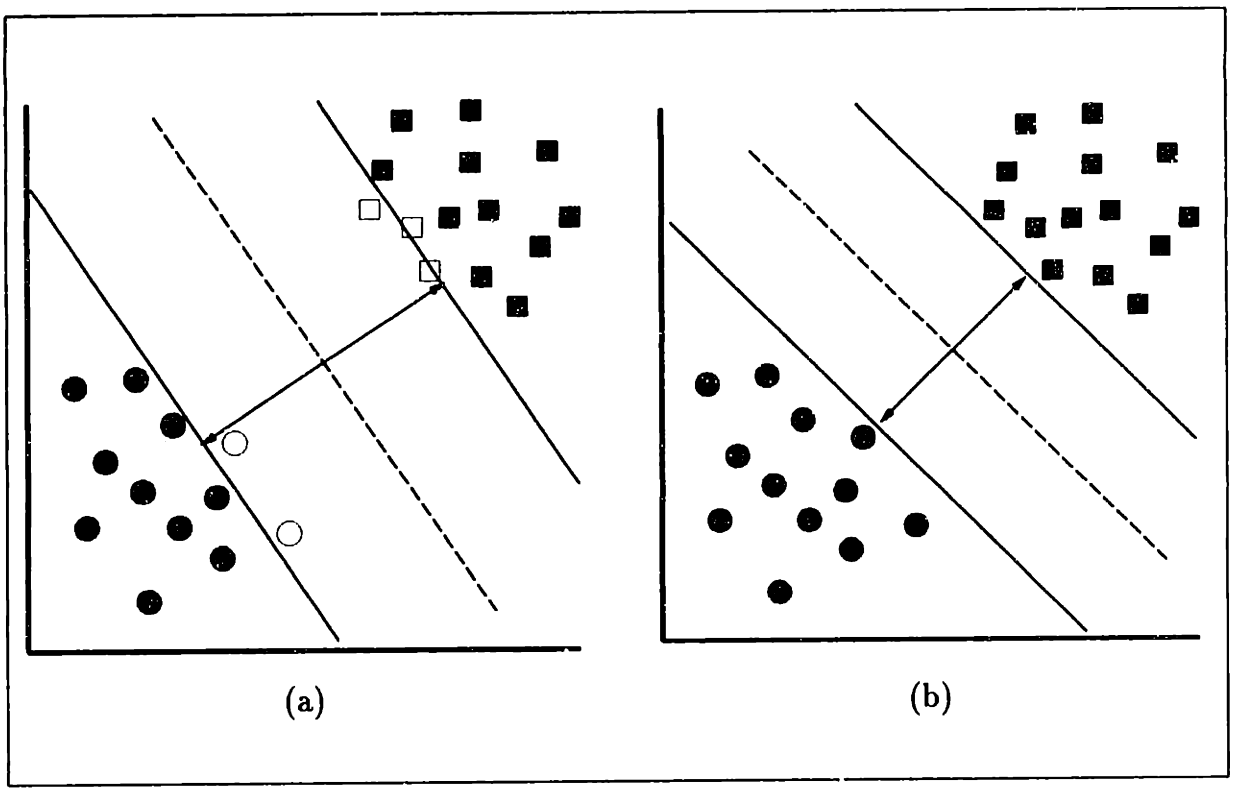


Figure 3-1: (a) A sub-optimal solution where the non-filled points have $\lambda = 0$ but are violating optimality conditions by being inside the ± 1 area. (b) The decision surface gets redefined. Since no points with $\lambda = 0$ are inside the ± 1 area, the solution is optimal. Notice that the size of the margin has decreased, and the *shape* of the decision surface has changed.

- **Lookahead**: this parameter specifies the maximum number of data points the pricing subroutine should use to evaluate optimality conditions (Case 3). If Lookahead data points have been examined without finding a violating one, the subroutine continues until it finds the first one , or until all data points have been examined. In the latter case, global optimality has been obtained.

- **Newlimit**: this parameter limits the number of new points to be incorporated into the working set B.

The computational results that we pr[illegible]nt in this section have been obtained using real data from our Face Detection System, which is described in Section 6.1.

Figure 3-2 shows the training time and the number of support vectors obtained when training the system with 5,000, 10,000, 20,000, 30,000, 40,000, 49,000, and 50,000 data points. We must emphasize that the last 1,000 data points were collected

in the last phase of bootstrapping of the Face Detection System, and therefore make the training process *harder*, since they correspond to errors obtained with a system that was already very accurate. Figure 3-3 shows the relationship between the training time and the number of support vectors, as well as the number of global iterations (the number of times the decomposition algorithm calls the solver). Notice the *smooth* relation between the number of support vectors and the training time, and the jump from 11 to 15 global iterations as we go from 49,000 to 50,000 samples. This increase is responsible for the increase in the training time. The system, using a working set of 1200 variables was able to solve the 50,000 data points problem using only 25Mb of RAM.

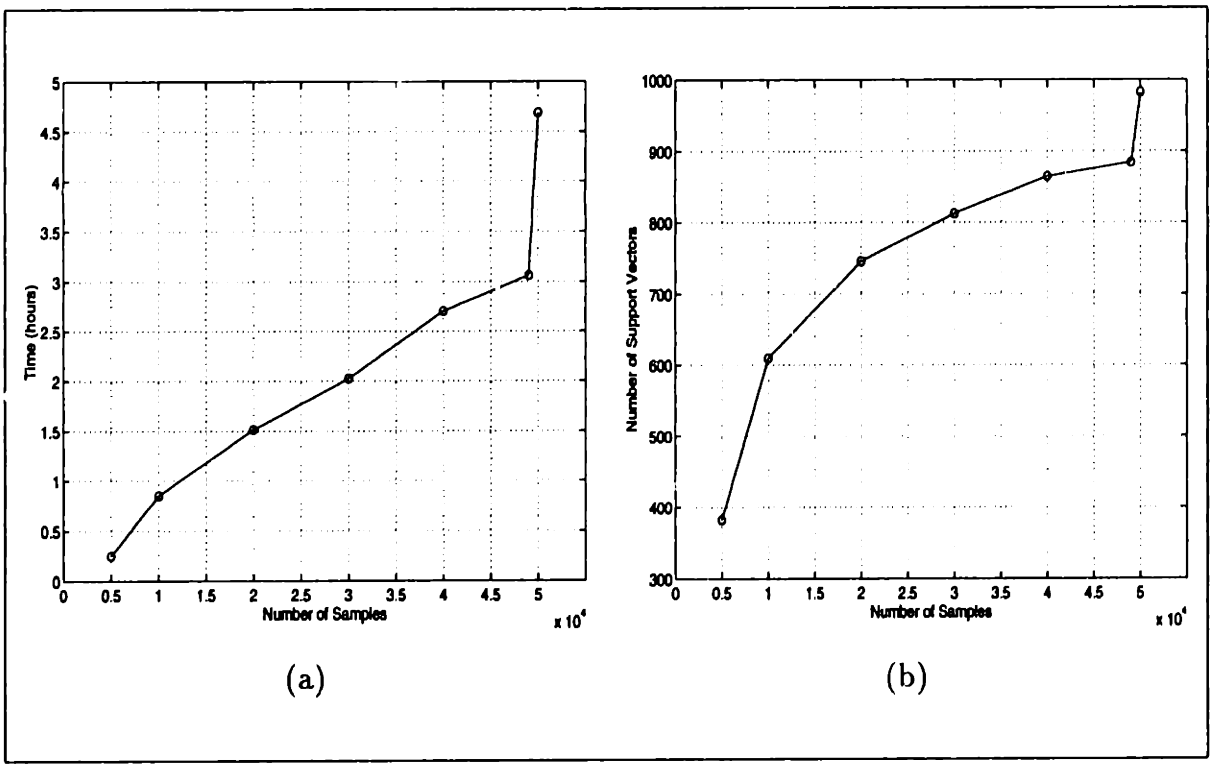


Figure 3-2: (a) Training Time on a SPARCstation-20. (b) Number of Support Vectors obtained after Training

Figure 3-4 shows the effect on the training time due to the parameter Newlimit and the size of the working set, using 20,000 data points. Notice the clear improvement as Newlimit is increased. This improvement suggests that in some way, the faster new violating data points are brought into the working set, the faster the decision surface is defined, and optimality is reached. Notice also that, if the working set is

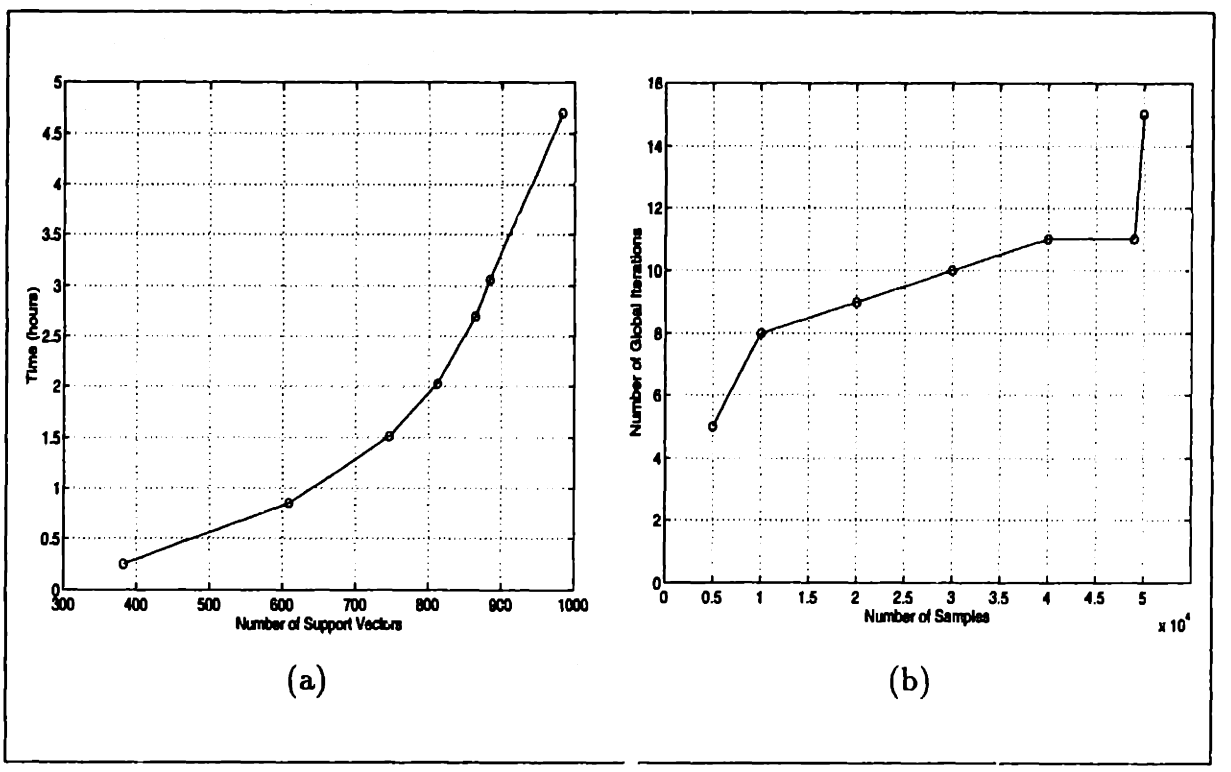


Figure 3-3: (a) Number of Support Vectors versus Training Time on a SPARCstation-20. Notice how the Number of support vectors is a better indicator of the increase in training time than the number of samples alone. (b) Number of global iterations performed by the algorithm. Notice the increase experimented when going from 49,000 to 50,000 samples. This increase in the number of iterations is responsible for the increase in the training time

too small or too big compared to the number of support vectors (746 in the case of 20,000 samples), the training time increases. In the first case, this happens because the algorithm can only incorporate new points slowly, and in the second case, this happens because the solver takes longer to solve the subproblem as the size of the working set increases.

3.3.5 Limitations

So far in the description and implementation of this decomposition algorithm, we have assumed that enough memory is available to solve a working set problem that contains all of the support vectors. However, some applications may require more support vectors than the available memory can manage. How can we overcome this problem? Are there conditions under which this can be done without compromising

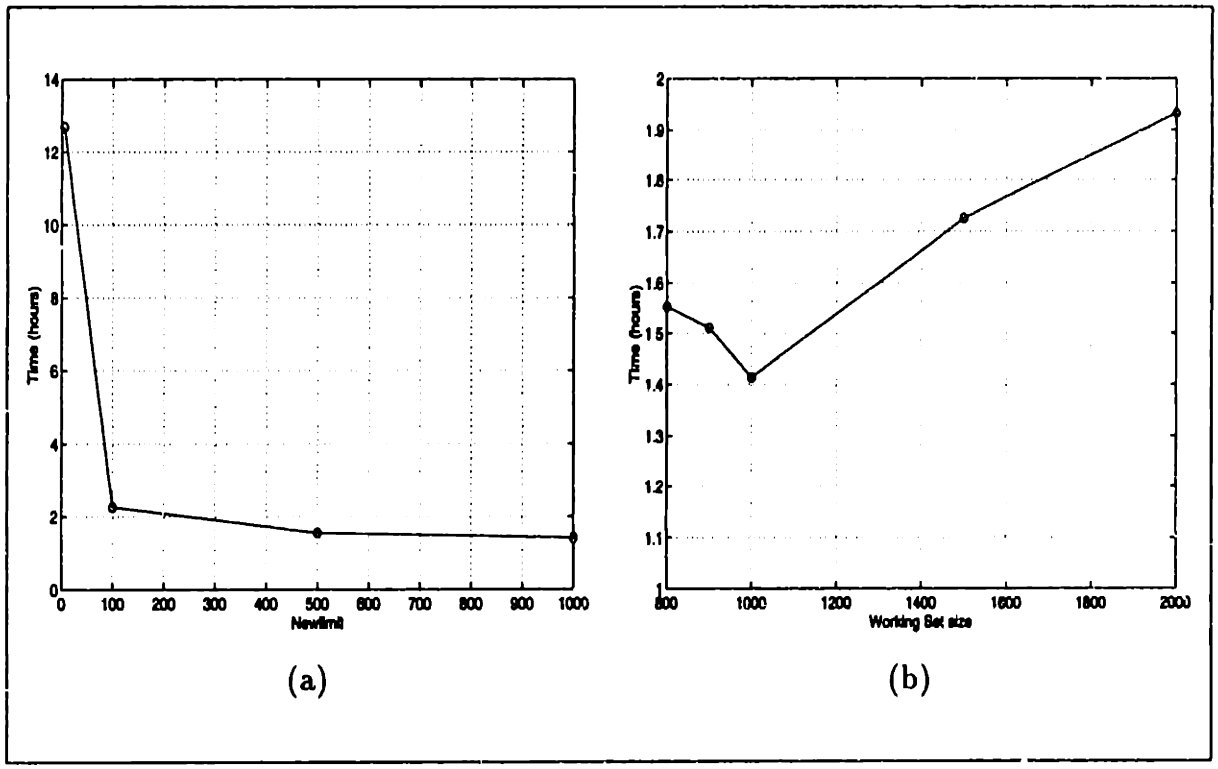


Figure 3-4: (a) Training time for 20,000 samples with different values of Newlimit, using a working set of size 1000 and Lookahead=10,000. (b) Training time for 20,000 samples with different sizes of the working set, using Newlimit=size of the working set, and Lookahead=10,000.

the training time or memory requirements?

A careful inspection of the optimality conditions yields the following observations:

- The optimality condition (3.8) (given for the case where $\lambda_i = C$) implies that any training error, and any data point that lives within the ± 1 margin, will have a coefficient $\lambda_i = C$. This can have severe consequences in this training algorithm since data sets which are non-separable can generate a great number of points with $\lambda_i = C$ and *clog* the working set. To give an idea of the impact of this, consider as an example, a 20,000 data-points financial data set where at best we can obtain 60% correct training. In such an example, the SVM will yield at least 8,000 support vectors, and a working set using at least 512 Mb of RAM!

- The only optimality condition used explicitly in this algorithm is the one stated for $\lambda_i = 0$. In the next section we explore using the other two conditions.

3.4 An Improved Approach

The optimality conditions derived in the previous section are essential in order to devise a decomposition strategy that guarantees that at every iteration the objective function is improved. In order to accomplish this goal we again partition the index set in two sets B and N, where the set B is called the *working set.* Then we decompose $\mathbf{\Lambda}$ in two vectors $\mathbf{\Lambda}_B$ and $\mathbf{\Lambda}_N$, keeping fixed $\mathbf{\Lambda}_N$ (although not necessarily at zero level) and allowing changes only in $\mathbf{\Lambda}_B$, thus defining the following subproblem:

$$\begin{aligned} \text{Minimize} \quad W(\mathbf{\Lambda}_B) \quad &= -\mathbf{\Lambda}_B^T\mathbf{1} + \tfrac{1}{2}\left[\mathbf{\Lambda}_B^T D_{BB}\mathbf{\Lambda}_B + \mathbf{\Lambda}_B^T D_{BN}\mathbf{\Lambda}_N + \right. \\ &\quad \left. + \mathbf{\Lambda}_N^T D_{NB}\mathbf{\Lambda}_B + \mathbf{\Lambda}_N^T D_{NN}\mathbf{\Lambda}_N\right] - \mathbf{\Lambda}_N^T\mathbf{1} \\ \mathbf{\Lambda}_B & \\ \text{subject to} & \end{aligned} \tag{3.11}$$

$$\begin{aligned} \mathbf{\Lambda}_B^T\mathbf{y}_B + \mathbf{\Lambda}_N^T\mathbf{y}_N &= \mathbf{0} \\ \mathbf{\Lambda}_B - C\mathbf{1} &\leq \mathbf{0} \\ -\mathbf{\Lambda}_B &\leq \mathbf{0} \end{aligned}$$

where $(\mathbf{1})_i = 1$, $D_{\alpha\beta}$ is such that $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, with $i \in \alpha, j \in \beta$, and C is a positive constant. Using this decomposition we notice that:

- The terms $-\mathbf{\Lambda}_N^T\mathbf{1} + \frac{1}{2}\mathbf{\Lambda}_N^T D_{NN}\mathbf{\Lambda}_N$ are constant within the defined subproblem.
- Since $K(\mathbf{x}, \mathbf{y})$ is a symmetric kernel, the computation of $\mathbf{\Lambda}_B^T D_{BN}\mathbf{\Lambda}_N + \mathbf{\Lambda}_N^T D_{NB}\mathbf{\Lambda}_B$ can be replaced by $2\mathbf{\Lambda}_B^T\mathbf{q}_{BN}$, where:

$$(\mathbf{q}_{BN})_i = y_i \sum_{j \in N} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \qquad i \in B \tag{3.12}$$

This is a very important simplification, since it allows us to keep the size of the subproblem independent of the number of fixed variables $\mathbf{\Lambda}_N$, which translates

into keeping it also independent of the number of support vectors.

- We can replace *any* λ_i, $i \in B$, with *any* λ_j, $j \in N$ (i.e. there is no restriction on their value), without changing the cost function or the feasibility of both the subproblem and the original problem.
- If the subproblem is optimal before such a replacement, the new subproblem is optimal if and only if λ_j satisfies the Optimality Conditions for the appropriate case (three cases described in section 3.3.1).

The previous statements lead to the following more formal propositions:

Proposition 3.4.1 *("Build down"): moving a variable λ_k from B to N leaves the cost function unchanged, and the solution is feasible in the subproblem.*

Proof: Let $B' = B \setminus \{k\}$ and $N' = N \cup \{k\}$. Then:

$$\begin{aligned}
W(\Lambda_B, \Lambda_N) &= -\sum_{i\in B}\lambda_i - \sum_{i\in N}\lambda_i + \frac{1}{2}\left[\sum_{i,j\in B}\lambda_i\lambda_j D_{ij} + 2\sum_{i\in B}\lambda_i\sum_{j\in N}\lambda_j D_{ij} + \right. \\
&\quad \left. + \sum_{i,j\in N}\lambda_i\lambda_j D_{ij}\right] \\
&= -\sum_{i\in B'}\lambda_i - \lambda_k - \sum_{i\in N}\lambda_i + \frac{1}{2}\left[\sum_{i,j\in B'}\lambda_i\lambda_j D_{ij} + 2\lambda_k\sum_{i\in B'}\lambda_i D_{ik} + \right. \\
&\quad \left. + 2\lambda_k\sum_{j\in N}\lambda_j D_{jk} + 2\sum_{i\in B'}\lambda_i\sum_{j\in N}\lambda_j D_{ij} + \lambda_k^2 D_{kk} + \sum_{i,j\in N}\lambda_i\lambda_j D_{ij}\right] \\
&= -\sum_{i\in B'}\lambda_i - \sum_{i\in N'}\lambda_i + \frac{1}{2}\left[\sum_{i,j\in B'}\lambda_i\lambda_j D_{ij} + 2\sum_{i\in B'}\lambda_i\sum_{j\in N'}\lambda_j D_{ij} + \right. \\
&\quad \left. + \sum_{i,j\in N'}\lambda_i\lambda_j D_{ij}\right] \\
&= W(\Lambda_{B'}, \Lambda_{N'})
\end{aligned}$$

The solution $(\Lambda_{B'}, \Lambda_{N'})$ is feasible in the subproblem since:

$$0 = \Lambda_B^T \mathbf{y}_B + \Lambda_N^T \mathbf{y}_N$$

$$= \mathbf{\Lambda}_{B'}^T \mathbf{y}_{B'} + \lambda_k y_k + \mathbf{\Lambda}_N^T \mathbf{y}_N$$
$$= \mathbf{\Lambda}_{B'}^T \mathbf{y}_{B'} + \mathbf{\Lambda}_{N'}^T \mathbf{y}_{N'}$$

and the bound constraints are always unaffected.**q.e.d.**

Proposition 3.4.2 *("**Build up**"): moving a variable that violates the optimality conditions from N to B gives a strict improvement in the cost function when the subproblem is re-optimized.*

Proof: This is a direct consequence of Proposition 3.4.1 and the fact that the Karush-Kuhn-Tucker conditions are necessary and sufficient for optimality.

3.4.1 The Algorithm

Using the results of the previous sections we are now ready to formulate our decomposition algorithm:

1. Arbitrarily choose $|B|$ points from the data set including both classes.
2. Solve the subproblem defined by the variables in B.
3. While there exists some $j \in N$, such that:
   - $\lambda_j = 0$ and $g(\mathbf{x}_j)y_j < 1$
   - $\lambda_j = C$ and $g(\mathbf{x}_j)y_j > 1$
   - $0 < \lambda_j < C$ and $g(\mathbf{x}_j)y_j \neq 1$,

   replace any λ_i, $i \in B$, with λ_j and solve the new subproblem given by:

$$
\begin{aligned}
\text{Minimize} \quad & W(\boldsymbol{\Lambda}_B) && = -\boldsymbol{\Lambda}_B^T \mathbf{1} + \tfrac{1}{2}\boldsymbol{\Lambda}_B^T D_{BB} \boldsymbol{\Lambda}_B + \boldsymbol{\Lambda}_B^T \mathbf{q}_{BN} \\
& \boldsymbol{\Lambda}_B & & \\
\text{subject to} & & & \\
& & \boldsymbol{\Lambda}_B^T \mathbf{y}_B + \boldsymbol{\Lambda}_N^T \mathbf{y}_N &= 0 \\
& & \boldsymbol{\Lambda}_B - C\mathbf{1} &\leq \mathbf{0} \\
& & -\boldsymbol{\Lambda}_B &\leq \mathbf{0}
\end{aligned}
\tag{3.13}
$$

where:

$$
(\mathbf{q}_{BN})_i = y_i \sum_{j \in N} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \qquad i \in B \tag{3.14}
$$

Notice that we have omitted the constant term $-\boldsymbol{\Lambda}_N^T \mathbf{1} + \frac{1}{2}\boldsymbol{\Lambda}_N^T D_{NN} \boldsymbol{\Lambda}_N$ in the cost function, and that according to Proposition 3.4.2, this algorithm will strictly improve the objective function at each iteration and therefore will not cycle. Since the objective function is bounded ($W(\boldsymbol{\Lambda})$ is convex quadratic and the feasible region is bounded), the algorithm must converge to the global optimal solution in a finite number of iterations.

3.4.2 Computational Results

We have implemented the decomposition algorithm using MINOS 5.4 [71] as the solver of the sub-problems. We tested our technique on a problem known for being "difficult": a foreign exchange rate time series that was used in the 1992 Santa Fe Institute Time Series Competition, in which we looked at the sign of the change of the time series, rather than its value. We considered data sets of increasing sizes, up to 110,000 points, obtaining up to 100,000 support vectors. Figure 3-5 shows the relationship between training times, number of data points and number of support

vectors in our experiments. The training time on a SUN Sparc 20 with 128 Mb of RAM ranged from 3 hours for 10,000 support vectors to 48 hours for 40,000 support vectors. The results that we obtain are comparable to the results reported in [116] using a Neural Networks approach, where generalization errors around 53% were reported. The purpose of this experiment was not to benchmark SVM's on this specific problem, but to show that its use in a problem with as many as 100,000 support vectors is computationally tractable.

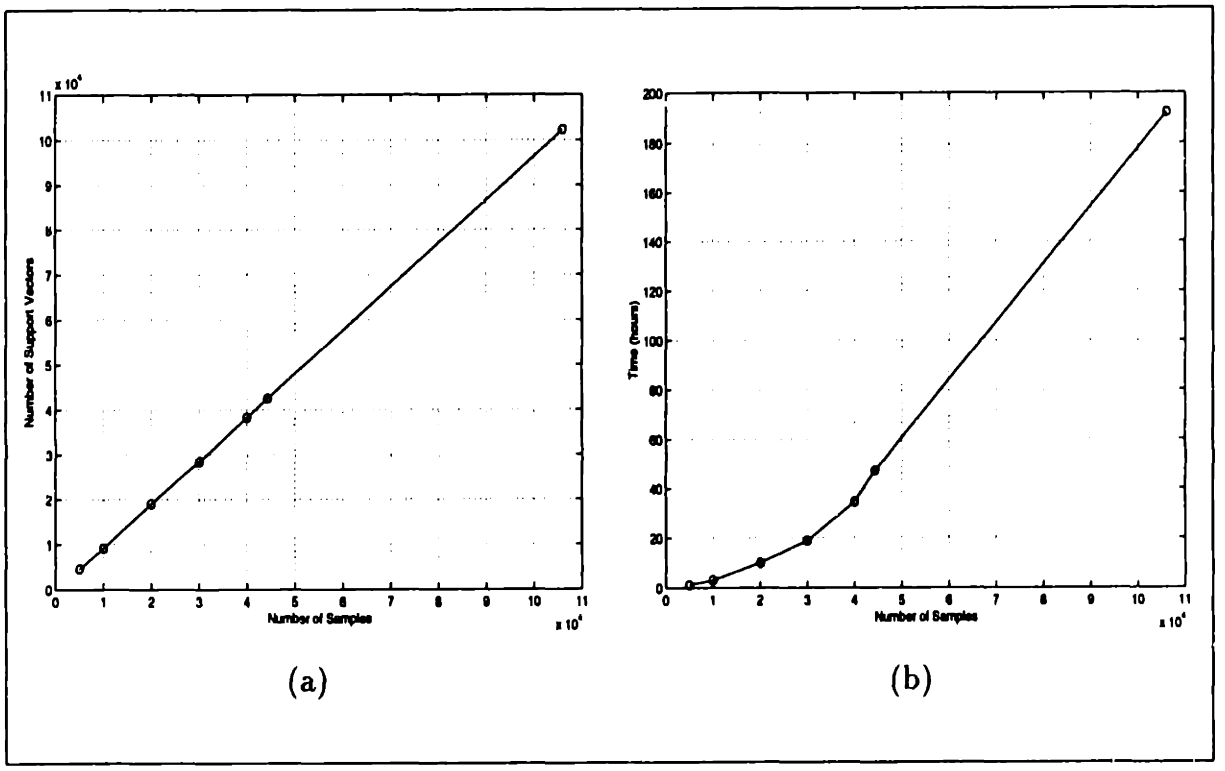


Figure 3-5: (a)Number of support vectors Vs. number of data points. (b) Training time Vs. number of data points.

3.5 Extension to Support Vector Regression

We have so far discussed the approach taken in order to train a Support Vector Machine for pattern classification. In this section we extend the decomposition algorithm described in section 3.4.1 to the problem of training a SVM for regression and function approximation. In order to accomplish this task, we first derive the optimality conditions for the QP formulation given in (2.61). We then present the algorithm

and some computational results of its implementation.

3.5.1 Optimality Conditions

In order to be consistent with common standard notation for nonlinear optimization problems, the quadratic program (2.61) can be rewritten in minimization form as:

$$\begin{aligned}
\text{Minimize} \quad & F(\boldsymbol{\Lambda}, \boldsymbol{\Lambda}^*) = \epsilon \sum_{i=1}^{\ell} (\lambda_i^* + \lambda_i) - \sum_{i=1}^{\ell} y_i (\lambda_i^* - \lambda_i) + \\
\boldsymbol{\Lambda}, \boldsymbol{\Lambda}^* \quad & + \tfrac{1}{2} \sum_{i,j=1}^{\ell} (\lambda_i^* - \lambda_i)(\lambda_j^* - \lambda_j) K(\mathbf{x}_i, \mathbf{x}_j)
\end{aligned}$$

subject to

$$\begin{aligned}
& \sum_{i=1}^{\ell} (\lambda_i^* - \lambda_i) = 0 && (\mu) \\
& -\boldsymbol{\Lambda}^* \leq \mathbf{0} && (\boldsymbol{\Upsilon}^*) \\
& -\boldsymbol{\Lambda} \leq \mathbf{0} && (\boldsymbol{\Upsilon}) \\
& \boldsymbol{\Lambda}^* - C\mathbf{1} \leq \mathbf{0} && (\boldsymbol{\Pi}^*) \\
& \boldsymbol{\Lambda} - C\mathbf{1} \leq \mathbf{0} && (\boldsymbol{\Pi})
\end{aligned} \tag{3.15}$$

where μ, $\boldsymbol{\Upsilon}^* = (\upsilon_1^*, \ldots, \upsilon_\ell^*)$, $\boldsymbol{\Upsilon} = (\upsilon_1, \ldots, \upsilon_\ell)$, $\boldsymbol{\Pi}^* = (\pi_1^*, \ldots, \pi_\ell^*)$ and $\boldsymbol{\Pi} = (\pi_1, \ldots, \pi_\ell)$ are the associated Karush-Kuhn-Tucker multipliers.

Since the cost function is convex (as in the pattern classification case) and the constraints are linear, the Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality, and they are:

$$
\begin{aligned}
&\epsilon - y_i + \sum_{j=1}^{\ell}(\lambda_j^* - \lambda_j)K(\mathbf{x}_i, \mathbf{x}_j) - \upsilon_i^* + \pi_i^* + \mu && = 0 \quad i = 1, \ldots, \ell \\
&\epsilon + y_i - \sum_{j=1}^{\ell}(\lambda_j^* - \lambda_j)K(\mathbf{x}_i, \mathbf{x}_j) - \upsilon_i + \pi_i - \mu && = 0 \quad i = 1, \ldots, \ell \\
&\pi_i^* \cdot (\lambda_i^* - C) && = 0 \quad i = 1, \ldots, \ell \\
&\upsilon_i^* \cdot \lambda_i^* && = 0 \quad i = 1, \ldots, \ell \\
&\pi_i \cdot (\lambda_i - C) && = 0 \quad i = 1, \ldots, \ell \\
&\upsilon_i \cdot \lambda_i && = 0 \quad i = 1, \ldots, \ell \\
&\boldsymbol{\Upsilon}^*, \boldsymbol{\Upsilon}, \boldsymbol{\Pi}^*, \boldsymbol{\Pi} && \geq \mathbf{0} \\
&\sum_{i=1}^{\ell}(\lambda_i^* - \lambda_i) && = 0 \\
&\boldsymbol{\Lambda}^*, \boldsymbol{\Lambda} && \geq \mathbf{0} \\
&\boldsymbol{\Lambda}^*, \boldsymbol{\Lambda} && \leq C\mathbf{1}
\end{aligned}
\tag{3.16}
$$

In order to simplify our presentation, we first demonstrate the following proposition:

Proposition 3.5.1 *If $\epsilon > 0$, then an optimal solution $(\boldsymbol{\Lambda}^*, \boldsymbol{\Lambda})$ satisfies $\lambda_i^* \cdot \lambda_i = 0$, for $i = 1, \ldots, \ell$.*

Proof: *Suppose we have an optimal feasible solution $(\boldsymbol{\Lambda}^*, \boldsymbol{\Lambda})$ such that $\lambda_i^* \cdot \lambda_i > 0$ for some i. Then:*

$$(\overline{\boldsymbol{\Lambda}^*}, \overline{\boldsymbol{\Lambda}}) = (\boldsymbol{\Lambda}^*, \boldsymbol{\Lambda}) - min(\lambda_i^*, \lambda_i) \cdot (\mathbf{e}_i, \mathbf{e}_i)$$

where $\mathbf{e}_i$ is the ith unit vector, is also feasible. Moreover,it can be verified that:

$$F(\overline{\boldsymbol{\Lambda}^*}, \overline{\boldsymbol{\Lambda}}) = F(\boldsymbol{\Lambda}^*, \boldsymbol{\Lambda}) - \epsilon \cdot min(\lambda_i^*, \lambda_i)$$

and therefore $F(\overline{\boldsymbol{\Lambda}^*}, \overline{\boldsymbol{\Lambda}^*}) < F(\boldsymbol{\Lambda}^*, \boldsymbol{\Lambda})$. This contradicts the optimality of $(\boldsymbol{\Lambda}^*, \boldsymbol{\Lambda})$.

q.e.d

In order to derive further algebraic expressions from the optimality conditions (3.16), we assume the existence of some λ_i^* and λ_i such that $0 < \lambda_i^* < C$ and $0 < \lambda_i < C$, and consider the possible values that they can have. Since $\lambda_i^* \cdot \lambda_i = 0$, for $i = 1, \ldots, \ell$, we assume a zero value for the corresponding variable when it applies. The possible values are as follows:

1. **Case** $0 < \lambda_i^* < C$:

   From the first, third and fourth KKT conditions we have:

   $$\epsilon - y_i + \sum_{j=1}^{\ell}(\lambda_j^* - \lambda_j)K(\mathbf{x}_i, \mathbf{x}_j) + \mu = 0$$

   Since the desired approximation form (see section 2.4.2) is described by:

   $$f(\mathbf{x}, \beta) = \sum_{i=1}^{\ell} \beta_i K(\mathbf{x}_i, \mathbf{x}) + b$$

   we have that for $b = \mu$ and $\beta_i = \lambda_i^* - \lambda, i = 1, \ldots, \ell$:

   $$\epsilon - y_i + f(\mathbf{x}, \beta) = 0$$

   which can geometrically be seen as having an approximation value at point $\mathbf{x}_i$ equal to $y_i - \epsilon$, that is, right at the **lower** wall of the *epsilon-tube* described in section 2.4.3 (see also Figure 2-4).

2. **Case** $0 < \lambda_i < C$:

   From the second, fifth and sixth KKT conditions we have:

   $$\epsilon + y_i - \sum_{j=1}^{\ell}(\lambda_j^* - \lambda_j)K(\mathbf{x}_i, \mathbf{x}_j) - \mu = 0$$

   As in the previous case,we have that for $b = \mu$ and $\beta_i = \lambda_i^* - \lambda, i = 1, \ldots, \ell$:

   $$\epsilon + y_i - f(\mathbf{x}, \beta) = 0$$

which can geometrically be seen as having an approximation value at ,int $\mathbf{x}_i$ equal to $y_i + \epsilon$, that is, right at the **upper** wall of the *epsilon-tube.*

3. **Case** $\lambda_i^* = C$:

   From the first and fourth KKT conditions we have:

$$\epsilon - y_i + \sum_{j=1}^{\ell}(\lambda_j^* - \lambda_j)K(\mathbf{x}_i, \mathbf{x}_j) + \pi_i^* + \mu = 0$$

   As in the previous case,we have that for $b = \mu$ and $\beta_i = \lambda_i^* - \lambda, i = 1, \ldots, \ell$:

$$\epsilon - y_i + f(\mathbf{x}, \beta) + \pi_i^* = 0$$

   which can geometrically be seen as defining π_i^* as the difference between the **lower** wall of the *epsilon-tube* at data point $\mathbf{x}_i$ and the approximation at $\mathbf{x}_i$ given by $f(\mathbf{x}_i, \beta)$.

   In order to satisfy the non-negativity condition of the multiplier, we then must have that:

$$\pi_i^* = y_i - \epsilon - f(\mathbf{x}_i, \beta) \geq 0$$

   and thus conclude that $\lambda_i^* = C$ can geometrically be seen been as having the approximation at $\mathbf{x}_i$, given by $f(\mathbf{x}_i, \beta)$, exactly or below the **lower** wall of the *epsilon-tube.*

4. **Case** $\lambda_i = C$:

   From the second and sixth KKT conditions we have:

$$\epsilon + y_i - \sum_{j=1}^{\ell}(\lambda_j^* - \lambda_j)K(\mathbf{x}_i, \mathbf{x}_j) + \pi_i - \mu = 0$$

   As in the previous case,we have that for $b = \mu$ and $\beta_i = \lambda_i^* - \lambda, i = 1, \ldots, \ell$:

$$\epsilon + y_i - f(\mathbf{x}, \beta) + \pi_i = 0$$

which can geometrically be seen as defining π_i as the difference between the approximation at $\mathbf{x}_i$ given by $f(\mathbf{x}_i, \beta)$ and the **upper** wall of the *epsilon-tube* at data point $\mathbf{x}_i$.

In order to satisfy the non-negativity condition of the multiplier, we then must have that:

$$\pi_i = f(\mathbf{x}_i, \beta) - y_i - \epsilon \geq 0$$

and thus conclude that $\lambda_i = C$ can geometrically be seen been as having the approximation at $\mathbf{x}_i$, given by $f(\mathbf{x}_i, \beta)$, exactly or above the **upper** wall of the *epsilon-tube.*

5. **Case** $\lambda_i^* = 0, \lambda_i = 0$:

   From the first and third KKT conditions we have:

$$\epsilon - y_i + \sum_{j=1}^{\ell}(\lambda_j^* - \lambda_j)K(\mathbf{x}_i, \mathbf{x}_j) - v_i^* + \mu = 0$$

   Similarly, from the second and fourth KKT condition we have:

$$\epsilon + y_i - \sum_{j=1}^{\ell}(\lambda_j^* - \lambda_j)K(\mathbf{x}_i, \mathbf{x}_j) + v_i - \mu = 0$$

   Using the same definition for $f(\mathbf{x}_i, \beta)$ as before, we have that:

$$\epsilon - y_i + f(\mathbf{x}_i, \beta) - v_i^* = 0$$

   and

$$\epsilon + y_i - f(\mathbf{x}_i, \beta) - v_i = 0$$

   By requiring the non-negativity of the multipliers v_i^* and v_i, we have:

$$v_i^* = f(\mathbf{x}_i, \beta) + \epsilon - y_i \geq 0$$

which can be seen as requiring the function approximation at $\mathbf{x}_i$ to be at or **above** the lower wall of the *epsilon-tube.*

Similarly, the condition :

$$v_i = \epsilon + y_i - f(\mathbf{x}_i, \beta) \geq 0$$

requires the approximation to be at or **below** the upper wall of the *epsilon-tube.*

Therefore, combining these two conditions we can conclude that $\lambda_i^* = \lambda_i = 0$ implies that the function approximation at point $\mathbf{x}_i$ lies at or **within** the *epsilon-tube.*

3.5.2 The Algorithm

In order to derive a training algorithm, we again partition the index set into two sets B and N, where the set B is called the *working set.* Then we decompose $(\mathbf{\Lambda}^*, \mathbf{\Lambda})$ into corresponding vectors $(\mathbf{\Lambda}_\mathbf{B}^*, \mathbf{\Lambda}_\mathbf{B})$ and $(\mathbf{\Lambda}_\mathbf{N}^*, \mathbf{\Lambda}_\mathbf{N})$, with the idea of keeping fixed $(\mathbf{\Lambda}_\mathbf{N}^*, \mathbf{\Lambda}_\mathbf{N})$ and allowing changes only $(\mathbf{\Lambda}_\mathbf{B}^*, \mathbf{\Lambda}_\mathbf{B})$. Using this partition, we can formulate the decomposition algorithm as:

1. Arbitrarily choose $|B|$ points from the data set.
2. Solve the subproblem defined by the variables in B.
3. While there exists some $j \in N$, such that:
   - $\lambda_j^* = 0$, $\lambda_j = 0$ and $|g(\mathbf{x}_j) - y_j| > \epsilon$
   - $(\lambda_j^* = C$ or $\lambda_j = C)$, and $|g(\mathbf{x}_j) - y_j| < \epsilon$
   - $(0 < \lambda_j^* < C$ or $0 < \lambda_j < C)$, and $|g(\mathbf{x}_j) - y_j| \neq \epsilon$,

   with

$$g(\mathbf{x}_j) = \sum_{i=1}^{\ell} (\lambda_i^* - \lambda_i) K(\mathbf{x}_i, \mathbf{x}_j) + b,$$

replace any pair (λ_i^*, λ_i), $i \in B$, with (λ_j^*, λ_j) and solve the new subproblem given by:

$$\begin{aligned}
\text{Minimize} \quad & F(\mathbf{\Lambda^*}_B, \mathbf{\Lambda}_B) && = \epsilon(\mathbf{\Lambda^*}_B + \mathbf{\Lambda}_B)^T\mathbf{1} - (\mathbf{\Lambda^*}_B - \mathbf{\Lambda}_B)^T(\mathbf{y}_B - \mathbf{q}_{BN}) + \\
\mathbf{\Lambda^*}_B, \mathbf{\Lambda}_B & && + \tfrac{1}{2}(\mathbf{\Lambda^*}_B - \mathbf{\Lambda}_B)^T D_{BB}(\mathbf{\Lambda^*}_B - \mathbf{\Lambda}_B)
\end{aligned}$$

subject to

$$\begin{aligned}
(\mathbf{\Lambda^*}_B - \mathbf{\Lambda}_B)^T\mathbf{1} &= -(\mathbf{\Lambda^*}_N - \mathbf{\Lambda}_N)^T\mathbf{1} \\
\mathbf{\Lambda^*}_B, \mathbf{\Lambda}_B &\leq C\mathbf{1} \\
\mathbf{\Lambda^*}_B, \mathbf{\Lambda}_B &\geq \mathbf{0}
\end{aligned} \tag{3.17}$$

where $\mathbf{1}$ is a vector of ones, $D_{\alpha\beta}$ is such that $D_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, with $i \in \alpha, j \in \beta$, and

$$(\mathbf{q}_{BN})_i = \sum_{j \in N} (\lambda_j^* - \lambda_j) K(\mathbf{x}_i, \mathbf{x}_j) \qquad i \in B$$

3.5.3 Computational Results

We have implemented this decomposition algorithm using MINOS 5.4 as the solver of the sub-problems. We tested our technique on an image representation/reconstruction problem. In particular, we used the traditional image of *Leena* at different resolutions, having as data points the grey-level values at its (x, y) pixel positions. Figure 3-6 shows an input image of Leena and its corresponding SV reconstruction.

Figure 3-7 shows the relationship between training times, number of data points and number of support vectors in our experiments. The training time on a SUN Sparc 20 with 128 Mb of RAM [2] ranged from 1.5 minutes for 428 support vectors (30×30

[2]The same computer used for the computer implementation and experiments of the SV classification training algorithms

(a) (b)

Figure 3-6: (a) Original Leena image at a 128 pixel resolution. (b) Support Vector reconstruction using $\epsilon = 20$ and a Gaussian kernel with $\sigma = 2$.

image = 900 data points) to 4.7 hours for 4341 support vectors ((128 × 128 image = 16,384 data points). These results were obtained using ϵ=20, C=500, a working set size of 900, and a Gaussian kernel with σ=2.

Two interesting by-products of these experiments are worth mentioning:

1. Figure 3-8(a) shows how the number of support vectors diminishes as ϵ increases: as the ϵ-tolerance becomes *larger*, fewer coefficients are needed in order to build the desired approximation.

2. Figure 3-8(b) shows the relationship between training time and the size of the working set. Notice that if the working set is too small or too big compared to the number of support vectors (860 in this case, for a 50 × 50 image = 2,500 data points), the training time increases. In the first case, this happens because the algorithm can only incorporate *violating* points slowly, and in the second case, because the solver takes longer to solve the corresponding sub-problem as the size of the working set increases.

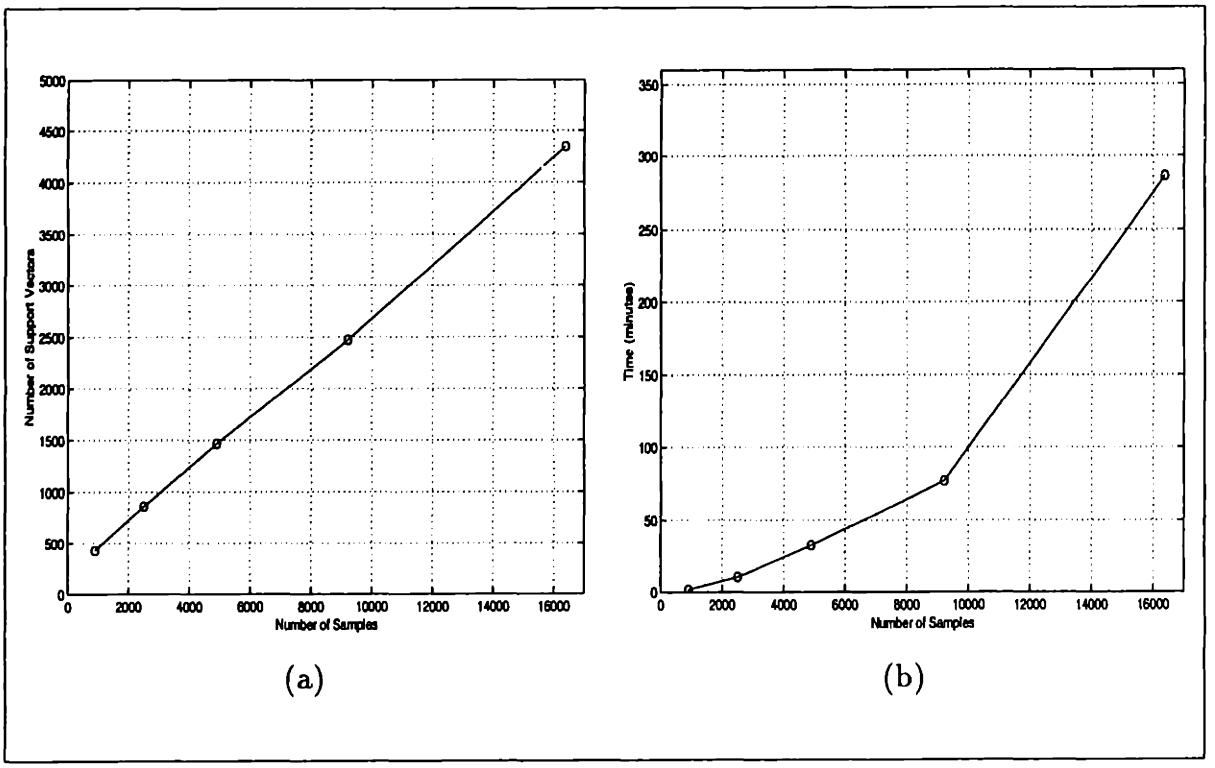


Figure 3-7: (a)Number of support vectors Vs. number of data points. (b) Training time Vs. number of data points.

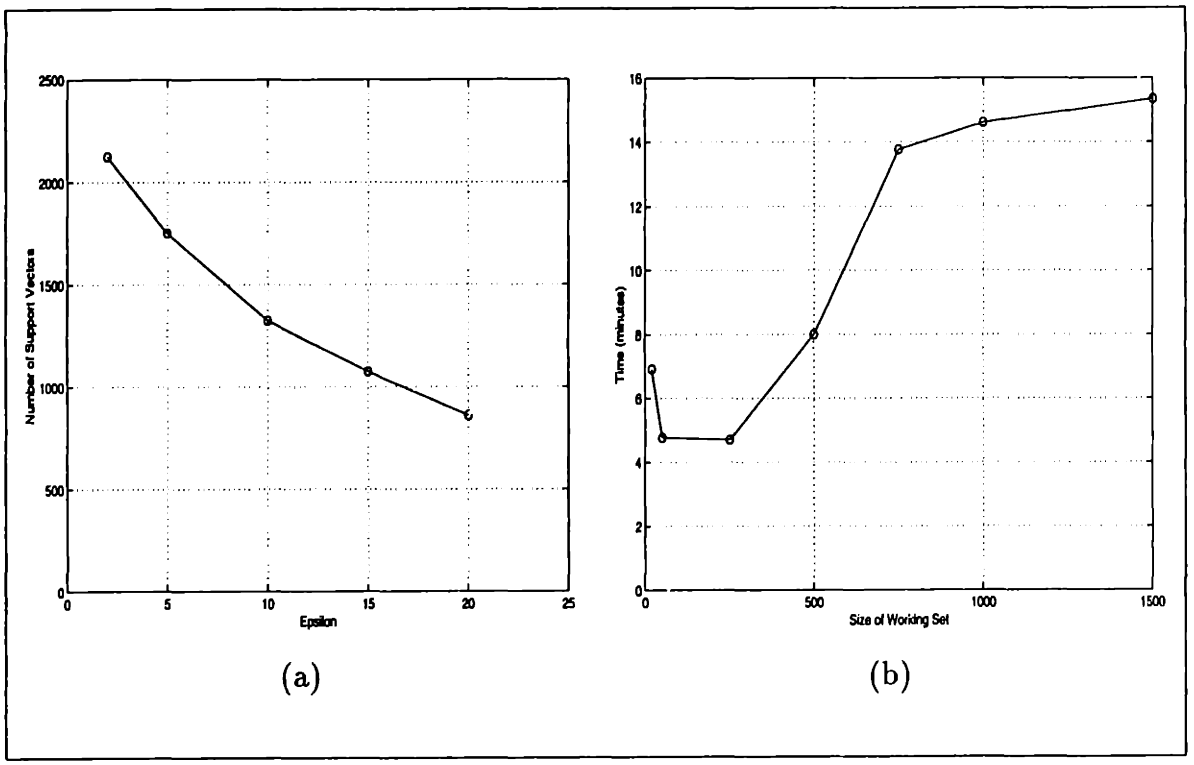


Figure 3-8: (a) Number of support vectors Vs. Epsilon. (b) Training time Vs. Size of the working set. These results were obtained using a 50×50 image (2,500 data points), $\epsilon = 20$ and a Gaussian kernel with $\sigma = 2$.

3.5.4 An Improved Version

The training algorithm for SV regression given in section 3.5.2 was developed following closely the spirit of its SV classification counterpart given in section 3.4. However, one interesting property of the optimal solution for SV regression was not taken into account in its formulation: The algorithm does not take advantage of Proposition 3.5.1 which states that if $\epsilon > 0$, then an optimal solution (Λ^*, Λ) satisfies $\lambda_i^* \cdot \lambda_i = 0$, for $i = 1, \ldots, \ell$. This means that after the solution of every sub-problem, at least half of variables involved (remember that we have two variables per data point) will be at zero level. There are many advantages in taking this proposition into account:

1. The storage needs for solvers that require as an explicit parameter the complete quadratic form (e.g. LSSOL [38] and CPLEX 4.0) is drastically reduced by 75%.

2. Even if the explicit quadratic form is not required by the solver, memory needed for basis storage is also saved.

3. Gradient and function evaluation, as well as line search procedures are at least two times faster.

4. More data points can simultaneously be considered within the same sub-problem size.

The implementation of this improvement can be considered as a minor modification of our current version.

3.6 Improving the Training of SVM: Future Directions

The algorithms described in the previous sections suggest two main areas where improvements can be made through future research. These two areas are:

1. **The Solver**: The second-order variant of the reduced gradient method implemented by MINOS 5.4 has given very good results so far in terms of accuracy, robustness and performance. However, this method is a general nonlinear optimization method that is not designed in particular for quadratic programs, and in the case of SVM's, is not designed in particular for the special characteristics of the problem. Having as a reference the experience obtained with MINOS 5.4, new approaches to a *tailored* solver through, for example, projected Newton [5] or interior point methods [20], should be attempted. Other QP-solvers available in the market like LSSOL [38], LOQO [104][105], CPLEX (version 4.0 or above), SQOPT 5.0 [39] [41] and QPOPT 1.0 [39] [40] should be investigated and compared with the solutions achieved using MINOS. Moré and Wright [67] present a comprehensive optimization software guide which contains contact information for these solvers.

   At this point it is not clear whether the same type of algorithm is appropriate for all stages of the solution process. To be more specific, it could happen that an algorithm performs well with few non-zero variables at early stages, and then is outperformed by others when the number of non-zero variables reaches some threshold. In particular, we learned that the number of non-zero variables that satisfy $0 < \lambda_i < C$ has an important effect on the performance of the solver.

2. **The Pivoting Strategy**: This area offers great potential for improvements. The improvements are based on some qualitative characteristics of the training process that have been observed:

   - During the execution of the algorithm, as much as 40% of the computational effort is dedicated to the evaluation of the optimality conditions. At final stages, it is common to have all the data points evaluated, yet only to collect very few of them to incorporate them into the working set.
   - Only a small portion of the input data is ever brought into the working set (about 16% in the case of the face detection application).

- Out of the samples that ever go into the working set, about 30% of them enter and exit this set at least once. These vectors are responsible for the first characteristic mentioned above.

Possible future strategies that exploit these characteristics are:

- Keep a list or file with all or part of the input vectors that have exited the working set. At the pricing stage, when the algorithm computes the optimality conditions, evaluate these data points before other data points to determine the entering vectors. This strategy is analogous to one sometimes used in the revised simplex method where the algorithm keeps track of the basic variables that have become non-basic. In the case of training of SVM's, the geometric interpretation of this heuristic is to think that if a point was a support vector at some iteration, it was more or less close to the boundary between the classes, and as this boundary is refined or *fine-tuned*, it is possible for it to switch from active to non-active several times. This heuristic could be combined with main memory and cache management policies used in computer operating systems.
- During the pricing stage, instead of bringing into the working set the first k points that violate optimality conditions, we could try to determine r violated data points, with $r > k$ and choose from these the k *most violated* points. This is done under the geometric idea that the *most violated* points help in defining the decision surface faster and therefore save time in future iterations.
- These last two approaches can be combined by keeping track not only of the points exiting the working set, but also of the remaining violating data points as well.

Chapter 4

Reducing the Run-time complexity of SVM's

As we sketched in section 1.2.1, the classification of a new pattern is based on the sign of $f(\mathbf{x})$ defined as:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{4.1}$$

where ℓ is the number of support vectors.

The reduction of the run-time complexity of SVMs can therefore be defined as a series of heuristics or techniques that reduce the computational effort spent in the evaluation or approximation of $f(\mathbf{x})$. The same issue arises in regression SVMs, where $f(\mathbf{x})$ is similarly defined as:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} (\lambda_i^* - \lambda_i) K(\mathbf{x}_i, \mathbf{x}) + b \tag{4.2}$$

In this section we focus on pattern classification with the understanding that the same approaches can be used in regression as well.

The outline of this chapter is as follows: in section 4.1 we introduce the motivation of this problem and comment on possible approaches to its solution; in section 4.2 we summarize the previous work in this topic; in section 4.3 we give an exact description of the properties of the problem we are approaching; in section 4.4 we present a first

approach in solving this problem using Support Vector Regression; in section 4.5 we offer a second approach to the solution which involves the reformulation of the training problem; in section 4.6 we present experimental results; and in section 4.7 we comment on the limitations of our two suggested approaches.

4.1 Motivation and Statement of the Problem

The operations described in (4.1) and (4.2) can easily become the bottleneck of any system that performs a massive number of classifications or function evaluations. Examples of this issue arise in our face (see section 6.1 and [78]) and people (see section 6.2 and [75]) detection systems, where SVMs are used as object-nonobject classifiers exhaustively; in checks and ZIP code readers, where the system can only afford to spend fractions of a second per check or envelope, etc.

In both pattern classification and regression machines, a speedup can be obtained by:

1. Approximating the solution $f(\mathbf{x})$ by:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i K(\mathbf{z}_i, \mathbf{x}) + b$$

   where $\mathbf{z}_i$ are *synthetic* vectors, which are not necessarily data points anymore, γ_i are weights, and $\ell' \ll \ell$. This approach, which for radial kernels K resembles the technique of "moving centers" [64, 81], has been pursued by C. Burges [13], but the procedure is slow, hard to implement, and lacks a principled way for controlling the approximation accuracy.

2. Approximating the solution $f(\mathbf{x})$ by:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

   where $\mathbf{x}_i$ is still a support vector with weight γ_i, but $\ell' \ll \ell$. This will be our first approach and it will be described in detail in section 4.4.

3. If possible, rewriting the solution $f(\mathbf{x})$ as:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

where $\mathbf{x}_i$ are still data points (but not necessarily support vectors according to the traditional definition) with weight γ_i, but $\ell' \ll \ell$. Our hope in this approach is to find an alternative more *efficient* representation of the separating hyperplane in problems were its expansion is not unique. This will be our second approach and it will be described in detail in section 4.5.

All of these heuristics try to approximate $f(\mathbf{x})$ using the kernel operator so that they can establish a meaningful accuracy comparison through the L_2 norm measured in *feature space*. Therefore, they strongly depend on the mapping:

$\mathbf{x} \in R^d \Rightarrow \Phi(\mathbf{x}) \equiv (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x})) \in R^n$

where: (1) n can be huge, and (2) not very much is known about the characteristics of the mapping itself. These two properties make this problem very hard to approach.

4.2 Previous Work: The Reduced Set Method

As we mentioned in section 4.1, one approach to speedup the test-phase of an SVM is to approximate $f(\mathbf{x})$ by:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i K(\mathbf{z}_i, \mathbf{x}) + b$$

where $\mathbf{z}_i$ are *synthetic* vectors, γ_i are weights, and $\ell' \ll \ell$. The purpose of this section is to describe this technique due to Burges [13], called the *Reduced Set Method*, which is the only previous work in this area.

In order to approximate $f(\mathbf{x})$ with the suggested form, Burges uses the kernel properties that define the dot product in feature space to calculate and minimize the difference between the true decision surface $\mathbf{w}$ and its approximation given by $\hat{\mathbf{w}}$. More formally:

- $\mathbf{w} = \sum_{i=1}^{\ell} \lambda_i y_i \Phi(\mathbf{x}_i)$ is the hyperplane in feature space obtained during the SVM training, and it defines $f(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b = \sum_{i=1}^{\ell} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$.

- $\hat{\mathbf{w}} = \sum_{i=1}^{\ell'} \gamma_i \Phi(\mathbf{z}_i)$ is the approximating hyperplane in feature space that we wish to obtain, and it defines $\hat{f}(\mathbf{x}) = \hat{\mathbf{w}}^T \Phi(\mathbf{x}) + b = \sum_{i=1}^{\ell'} \gamma_i K(\mathbf{z}_i, \mathbf{x}) + b$.

- The approximation error ρ can be defined as:

$$\begin{aligned} \|\mathbf{w} - \hat{\mathbf{w}}\|_2^2 &= (\mathbf{w} - \hat{\mathbf{w}})^T (\mathbf{w} - \hat{\mathbf{w}}) \\ &= \mathbf{w}^T \mathbf{w} + \hat{\mathbf{w}}^T \hat{\mathbf{w}} - 2\mathbf{w}^T \hat{\mathbf{w}} \\ &= \sum_{i,j=1}^{\ell} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i,j=1}^{\ell'} \gamma_i \gamma_j K(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{i=1}^{\ell} \sum_{j=1}^{\ell'} \lambda_i y_i \gamma_j K(\mathbf{x}_i, \mathbf{z}_j) \end{aligned}$$

- By minimizing ρ with respect to the vectors z_i's (and its components) and the weights γ_i, we can obtain, for a fixed ℓ', the desired approximation.

The Reduced Set Method can then be stated as an unconstrained minimization of ρ with respect to the synthetic vectors and the corresponding weights. This minimization is not easy to perform because of the non-smooth and non-convex characteristics of the surface and the rapid increase in the number of free variables in real life applications. Burges showed that the use of one particular class of kernel, the second degree homogeneous polynomial, allows a closed form solution to this minimization problem. Next we describe this solution in more detail, since it is used in our face and people detection applications. Our intention is to give some idea of how this particular case behaves. More detail can be found in [13].

Homogeneous Quadratic Polynomials:

We consider the case when $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^2$, for $\mathbf{x}_i, \mathbf{x}_j \in \Re^n$. To simplify the presentation, we start with the simplest case, that is, when $\ell' = 1$. If we introduce the symmetric matrix S, where:

$$S_{\mu\nu} = \sum_{i=1}^{\ell} \lambda_i y_i (\mathbf{x}_i)_\mu (\mathbf{x}_i)_\nu \qquad \text{for } \mu, \nu = 1 \ldots n$$

one can show that to minimize ρ, $\mathbf{z}$ must satisfy:

$$S_{\mu\nu}\mathbf{z}_{\nu} = \gamma\|\mathbf{z}\|^2\mathbf{z}_{\mu}$$

and that ρ^2 reduces to:

$$\rho^2 = \text{Trace}(S^2) - \gamma^2\|\mathbf{z}\|^4.$$

From this expression we obtain that the largest reduction in ρ results when we select $\mathbf{z}$ to be the eigenvector of S with the largest sized absolute eigenvalue, and scale $\mathbf{z}$ so that $\gamma\|\mathbf{z}\|^2$ is that eigenvalue. The factor γ allows the use of negative eigenvalues,and can be chosen accordingly such that $\gamma \in \{-1, 1\}$.

By extending this idea to more vectors (i.e. $\ell' > 1$), one can show that the approximation becomes exact (i.e. $\rho = 0$) when $\ell' = n$, and that ρ^2 is reduced by:

$$\rho^2 = \text{Trace}(S^2) - \sum_{i=i}^{\ell'} \|\mathbf{z}\|^4$$

where $\mathbf{z}_i$ are the eigenvectors of S. In practice, one selects the top ℓ' eigenvectors sorted by the absolute value of their corresponding eigenvalue.

We applied this technique to two different databases:

- A set of 1969 support vectors obtained from training our 29 feature vector from the people detection database (see section 6.2).
- A set of 964 support vectors obtained from training our 283 pixel vector from the face detection database (see section 6.1).

As we have stated earlier, exact *compression* to 29 and 283, respectively, can be achieved without any loss or system degradation.

Results obtained when applying this technique can be seen in Figures 4-1 and 4-2.

Two interesting issues arise from our experiments:

1. Figure 4-1 shows that in order to get a satisfactory approximation, we must use nearly all of the eigenvectors. This can be contrasted with Figure 4-2, which shows that an equally satisfactory approximation can be obtained when using

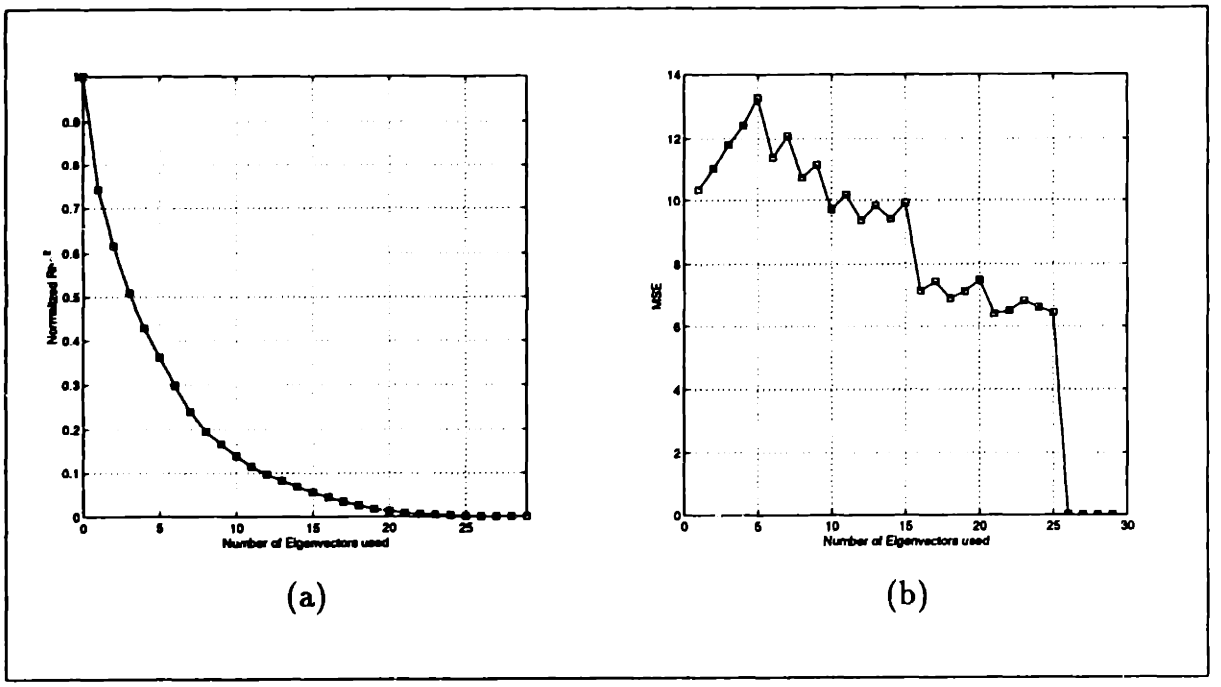


Figure 4-1: The Reduced Set Method applied to the people detection database. (a) Normalized ρ^2 as a function of ℓ' (i.e., the number of eigenvectors used in the expansion). (b) MSE from the approximation as a function of ℓ', evaluated over the $\ell = 1969$ support vectors originally obtained during training.

just 30% of the eigenvectors. This is an interesting observation that seems to suggest that the ratio ℓ'/n is somewhat dependent on the amount of useful information coded in the input vector. In these particular experiments, the results suggest that nearly all of the input components are relevant for the people/non-people discrimination task, whereas not even half are relevant for the face/non-face discrimination task.

2. Figures 4-1(b) and 4-2(b) show that the MSE, computed as:

$$MSE = \frac{\sum_{i=1}^{\ell}(f(\mathbf{x}_i) - \hat{f}(\mathbf{x}_i))^2}{\ell}$$

is not monotonically decreasing as a function of ℓ'. Since the sign of the classification is on the sign of $f(\mathbf{x})$, these plots can be used to define a hierarchical classification procedure. This procedure, for example, may approximate $f(\mathbf{x})$ using the first ℓ_1 eigenvectors, and continue using up to ℓ_2 (with $\ell_1 < \ell_2$) eigenvectors if $\hat{f}(\mathbf{x})$ falls within some range of *low confidence.*

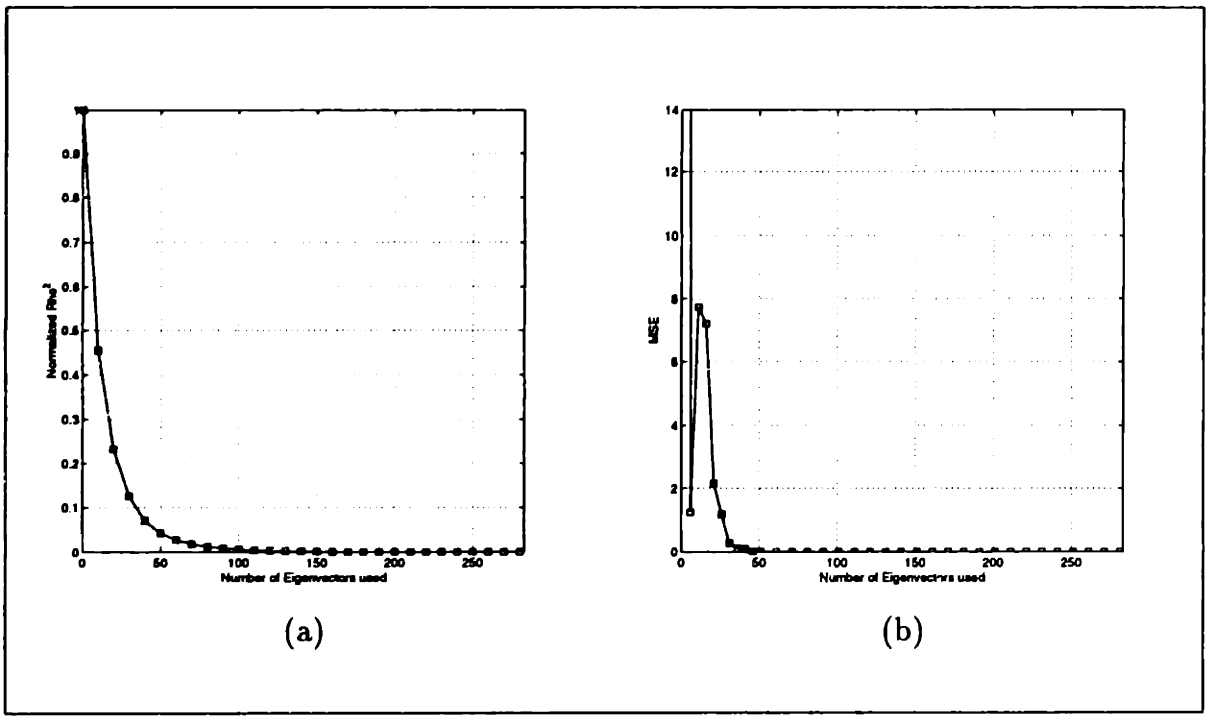


Figure 4-2: The Reduced Set Method applied to the face detection database. (a) Normalized ρ^2 as a function of ℓ' (i.e. the number of eigenvectors used in the expansion). (b) MSE from the approximation as a function of ℓ', evaluated over the $\ell = 964$ support vectors originally obtained during training.

4.3 The class of problems we approach

Before characterizing explicitly the kind of problems we are trying to solve, we want to present the following example: Training the Ripley data set [1] (250 datapoints, 2 dimensions, not linearly separable) yields roughly 90 support vectors (actually 89 for C=100). This means that the separating hyperplane $\mathbf{w} = \sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}_i$ is defined using 90 non-zero coefficients. Figure 4-3 presents the data set, the support vectors and the separating hyperplane. This representation for $\mathbf{w}$ is strikingly inefficient, since we are using a linear combination of 90 vectors to define a vector $\mathbf{w}$ in $\Re^2$.

Although the example given is a simple linear classifier in $\Re^2$, it is important to remark that similar behavior will be encountered in other examples involving nonlinear decision surfaces and input spaces with higher dimensionality.

Why is this happening?

[1] Available at ftp://markov.stats.ox.ac.uk/pub/neural/papers/

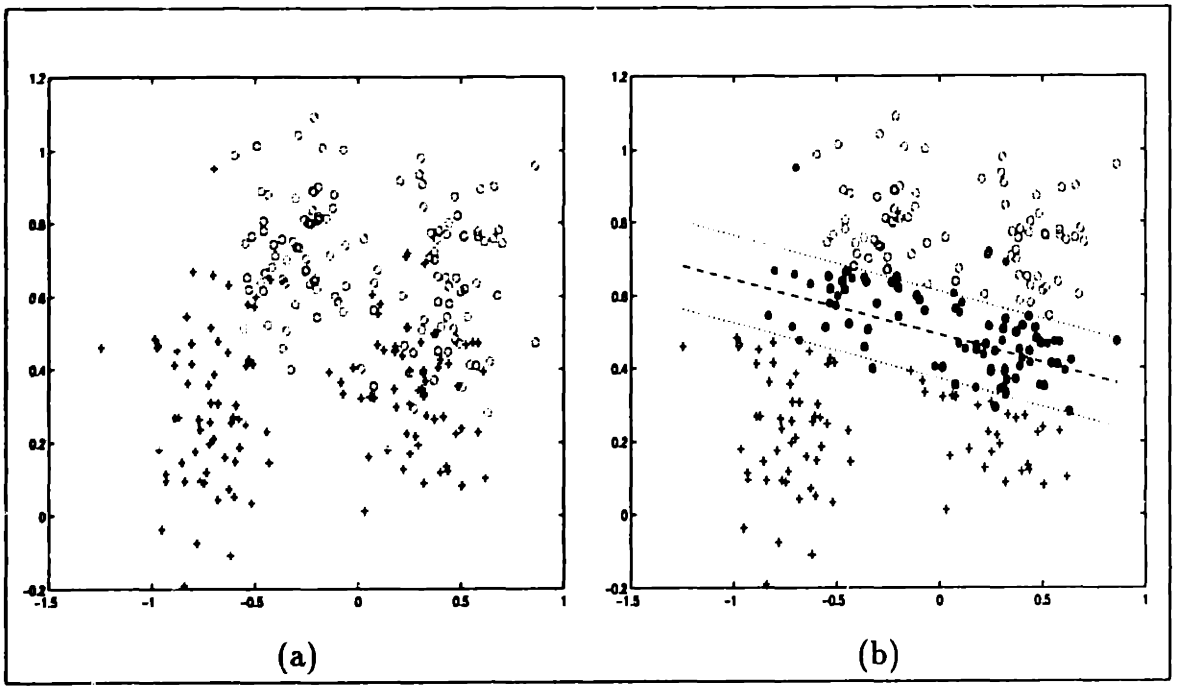


Figure 4-3: (a) The Ripley data set. (b) The optimal separating hyperplane and its support vectors. The dotted lines above and below the hyperplane depict the $\pm$ 1 range around the saparating surface.

1. The Karush-Kuhn-Tucker optimality conditions of the training problem mandate for any misclassified training point [2] to have $\lambda_i = C$.

2. Any non-zero λ_i contributes in the expansion of $\mathbf{w}$, and cannot be removed from the training set without affecting the solution.

3. Since the data set is non-separable, all of the training errors are present in the expansion.

Moreover, desperate attempts like:

- removing known errors from the data-set before training,
- removing errors from the expansion after training, or
- training, removing the errors from the obtained set of support vectors, and retraining,

[2] Also points falling within the margin, but correctly classified will have $\lambda_i = C$.

will not, in general, give the same separating hyperplane, since some of these errors are actually needed in the expansion of $\mathbf{w}$ and also need to be taken into account when penalizing non-separability. Therefore, we want to approach problems in which the number of support vectors with $\lambda_i = C$ is *excessively* high. This is a problem that appears in particular when the data set is non-separable, and becomes more relevant with an increase in noise, degree of non-separability, and size of the training set.

4.4 First approach: Using SVRM

It has been recently shown [43] that under certain conditions Support Vector Regression Machines (SVRM) are equivalent to Basis Pursuit De-Noising (BP) [25][26]. BP is an instance of a *sparse* approximation technique in which a function is reconstructed using the least possible number of basis functions chosen from a large set, which is called a *dictionary.* Having said that, we can think of building a *sparse* approximation of $f(\mathbf{x})$ using as dictionary the implicit mappings given by the kernel function evaluated at the data points [3].

Given a desired ϵ-accuracy, we are interested in drawing the least number of basis functions in order to build our approximation.

Which Kernel do we use?

At this point we can use any of the valid kernels for SV regression and still obtain a *sparse* approximation. However, by using the same kernel that was used during the initial training, we obtain as byproduct three interesting properties:

1. ϵ-accuracy can be arbitrarily small and a perfect approximation can be achieved, provided that the parameter C is large enough. This is an important property if we consider the approximation quality to be more important than speed.

2. The error measure $\|\mathbf{w} - \hat{\mathbf{w}}\|_2^2$ (where $\mathbf{w}$ is the separating hyperplane in feature space and $\hat{\mathbf{w}}$ is its approximation) can be computed using the kernel function.

[3]The datapoints are in this case the support vectors obtained during the training of the SV classifier.

3. Since the kernel used initially during training has already built a linear hyperplane in feature space, using the same kernel takes advantage of that same linearization.

The Algorithm:

The formal algorithm can be stated as:

1. Train the SV classifier using the kernel and parameters of one's choice. This step defines $f(\mathbf{x}) = \sum_{i=1}^{\ell} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$.

2. Run SVRM on the data points defined by $(\mathbf{x}, f(\mathbf{x}))$; where $\mathbf{x}$ is a support vector obtained in step 1. Use in this step a high value for the parameter C, the same kernel used in (1), and the desired ϵ-accuracy.

The results obtained with this approach are shown in section 4.6.

4.5 Second approach: Reformulating the Training Problem

Traditionally, the flow of the mathematical derivation for the training problem in SVMs (see section 2.3.1 and [107]) has gone from the easiest problem of linearly separable data sets and linear decision surfaces, to non-linear decision surfaces using kernel functions. During this derivation, a Lagrangian function is introduced as a way to solve the problem and to show how the separating hyperplane $\mathbf{w}$ can be written as a linear combination using Lagrange multipliers as coefficients. This step has somehow (and erroneously) suggested to several researchers that solving a primal version without explicitly incorporating the mapping $\mathbf{x} \in R^d \Rightarrow \mathbf{z}(\mathbf{x}) \equiv (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x})) \in R^n$ is impossible. One can easily show that the Wolfe dual of the training problem currently in use corresponds to:

$$\text{Minimize} \quad F(\mathbf{\Lambda}, b, \mathbf{\Xi}) = \frac{1}{2}\mathbf{\Lambda}^T Q \mathbf{\Lambda} + C\sum_{i=1}^{n}\xi_i$$

subject to

$$\begin{aligned} y_i(\sum_{j=1}^{n}\lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b) &\geq 1 - \xi_i & i &= 1 \ldots n \\ \xi_i, \lambda_i &\geq 0 & i &= 1 \ldots n \\ b & \quad free \end{aligned} \tag{4.3}$$

where $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$

This problem formulation, which from now on we refer to as the *primal reformulation* has the same initial structure as the original primal formulation, but incorporates the kernel mapping implicitly through the kernel function, and therefore, works also for defining non-linear decision surfaces. Moreover, its interpretation is natural and clear:

- $\mathbf{\Lambda}^T Q \mathbf{\Lambda}$ can be shown to be proportional to the inverse of the margin in feature space (and therefore we want to minimize it);
- The constraint $y_i(\sum_{j=1}^{n}\lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b) \geq 1 - \xi_i$ models how well the data point $\mathbf{x}_i$ is classified.
- ξ_i captures for data point i, its *degree of separability*, and *pays* a linear penalty C in the cost function.

One can show that the minimum of this problem gives the same separating surface as in the classical approach, but probably with a different expansion for $\mathbf{w}$. We say *probably*, since the problems we are approaching are characterized by the fact that the expansion of $\mathbf{w}$ is not unique, and by the observation that the current SVM training gives us an expansion where the number of coefficients is absurdly large.

It is not guaranteed that the new representation of the hyperplane is more sparse than the classical solution. However, experimentally, this approach works very well

for two reasons: first, the coefficients are not *forced* to the upper bound for misclassifications, since they are not Lagrange multipliers anymore; and second, starting with $\mathbf{\Lambda} = \mathbf{0}$ as initial solution helps in keeping at 0 level unnecessary points in the expansion.

4.5.1 Possible Improvements

A clear advantage of this primal reformulation, when compared to the training problem typically solved , is that we can include in the cost function certain attractor terms in order to benefit certain types of expansions. For example:

- We can include a small penalty of the form $\sum_{i=1}^{l} |\lambda_i|$, which has been used before in other techniques to enforce sparse representations [25][26][43][74].
- We can include a small penalty for using points that do not meet the separability constraints exactly. This will cause the set of coefficients to be a subset of the support vectors obtained through the current training problem. Since the coefficients are not Lagrange multipliers anymore, the values can be drastically different.
- We can include a small penalty for using errors, etc.

We want to remark that these penalties should be *small* so that the essence of the cost function is altered in a minimal way, that is, these small penalties are just coding a *preference* among all the possible linear combinations. A scheme in which one gradually reduces these penalties to zero is also possible.

The results obtained with this approach are shown in section 4.6.

4.6 Experimental Results

Three datasets were selected for our experiments:

- *skin* corresponds to pixel examples of normalized red and green values, where the task is skin/non-skin classification [4].

- *electrons* corresponds to measurements taken after the collision of electrons, and the task is the classification of the outcome [5].

- *ripley* is a synthetic two-class problem generated by Ripley [86](page 11) where each class has a bimodal distribution [6].

Two different runs were performed with each data set,where each run corresponds to a different kernel and/or parameter setting. The problems selected, the information regarding the kernel parameters, the number of support vectors and training performance is given in table 4.1. The training performance is presented to give the reader an idea of the separability of the data.

Database	Dim	# Data points	Kernel	C	Tr. Perf.	# SVs	$\|w\|$
skin (1)	2	1600	pol 2	100	91.94	587	12,669.36
skin (2)	2	1600	pol 5	100	95.81	227	58,005.72
electrons (1)	8	2000	pol 2	100	89.20	611	2,082.35
electrons (2)	8	2000	rbf σ =2	100	91.60	554	10,210.16
ripley (1)	2	250	linear	100	86.40	89	71.80
ripley (2)	2	250	rbf σ =1	100	89.60	77	643.11

Table 4.1: Problems selected for our experiments.

The results obtained with the SV regression approach can be found in table 4.2. Notice that reduction percentages are above 50% in all runs except electrons (2). Notice also the relationship between the ϵ-approximation quality and the number of vectors obtained.

The results obtained with the primal reformulation are also shown in table 4.2. As it was the case with the SV regression approach, reduction percentages are above 50% in all runs except electrons (2).

[4]This data set is available at ftp://ftp.ai.mit.edu/pub/cbcl/skin.zip.

[5]This data set was provided by A. Verri and M. Pontil, and is available at ftp://ftp.ai.mit.edu/pub/cbcl/electrons.zip.

[6]This data set is available at ftp://markov.stats.ox.ac.uk/pub/neural/papers/.

Database	Regression						Primal Ref.	
	$\epsilon = 10^{-2}$			$\epsilon = 10^{-6}$				
	$\frac{\|w-\tilde{w}\|}{\|w\|}$	# vec.	Red. %	$\frac{\|w-\tilde{w}\|}{\|w\|}$	# vec.	Red. %	# vec.	Red. %
skin (1)	1.4×10^{-3}	11	98.13	1.6×10^{-10}	25	95.74	6	98.98
skin (2)	1.2×10^{-3}	10	95.59	6.1×10^{-7}	75	66.96	57	74.89
electrons (1)	6.1×10^{-3}	40	93.45	2.9×10^{-9}	47	92.30	44	92.80
electrons (2)	6.6×10^{-3}	297	46.38	5.1×10^{-7}	554	0.0	521	5.96
ripley (1)	2.6×10^{-5}	3	96.63	2.5×10^{-12}	3	96.63	2	97.75
ripley (2)	4.6×10^{-3}	14	76.82	2.4×10^{-7}	33	57.14	34	55.84

Table 4.2: Results obtained using the SV regression and primal reformulation approaches. The column "# vec." reflects the number of vectors that are now present in the expansion. This number corresponds with a percentage reduction indicated in the column "Red. %". The performance of both approaches in corresponding test-sets were the same as with the original SV formulation.

4.7 Limitations and Final Remarks

Although we think that our solutions have been successful for the kind of problem we decided to approach, there are two clear limitations that must be discussed:

1. Neither solution will reduce the run-time complexity of the classifier when all or most of the coefficients are strictly between the bounds (i.e. $0 < \lambda_i < C$). This situation tends to occur when the data is highly (if not totally) separable, and worsens as the dimensionality of the feature space grows. Up to date, the best we can do in cases like this is to use the reduced set method of Burges [13].

2. Although a delayed column generation algorithm can be devised for decomposing and solving our primal reformulation, memory limitations make it prohibitive for large data sets (beyond 10,000). Alternatives to this problem are still an open area of research.

We feel that both approaches have attacked and solved the problem of an excessive number of support vectors with active upper bound (i.e. $\lambda = C$). The SV regression approach is useful since it can be applied as an *after-training* filter, it is not limited by memory requirements, and it offers accuracy control of the approximation through the use of the parameter ϵ. The primal reformulation offers a *one-step*

training approach that is not an approximation, but in fact gives exactly the same hyperplane obtained by training the current QP problem. Together with the possible improvements suggested in section 4.5.1, this technique can be made to enforce certain properties of the points used in the linear combination, and in fact, can be used to provide a subset of the original support vectors without any loss of accuracy or approximation error. This primal reformulation has a very natural structure and therefore is suitable for improvements in the technique.

As a last comment, we must remark that throughout this section we have only considered reducing the run-time complexity of the SV classifier. The proposed methods can easily be adapted to reducing the complexity of SV regression machines as well.

Chapter 5

VC-Bound Computation and Parameterized Kernels

5.1 Motivation

The purpose of this chapter is to study in detail how to improve the way we use the VC-Bound (see chapter 2, equation (2.1)) given by:

$$R(\lambda) \leq R_{\text{emp}}(\lambda) + \sqrt{\frac{h\left(\ln\frac{2l}{h}+1\right) - \ln\frac{\eta}{4}}{l}} \qquad \forall\lambda \in \Lambda \tag{5.1}$$

where h is the VC-dimension, and the bound on h (see chapter 2, equation (2.8)) is given by:

$$h \leq \min\{\lceil R^2A^2\rceil, N\} + 1 \tag{5.2}$$

These two bounds are at the heart of the Support Vector Machine. The first one motivates the idea of not just controlling how well we fit our data, but also of controlling the complexity of the functions we use, i.e., the VC-dimension h.

The second bound relates the VC-dimension h with the *margin*, and therefore justifies the idea of maximizing the margin as a way to control the upper bound on h, and consequently, the complexity measure on bound (5.1).

The first issue we should point out is that the bound on h (5.2) depends not only on A (the margin), but also on R: The radius of the smallest sphere that contains all the data points. This means that finding a way to compute R is important in order to calculate the bound on h, and as a consequence, to estimate the VC-bound (5.1). This computation is particularly difficult to achieve when using non-linear SVM's, since the kernel is the only source of information available to describe how the high-dimensional mapping has been performed. As we will show in section 5.2, R can be computed by solving a QP which is very similar in structure and difficulty to the one solved for SVM training. Also in that section, we formulate a decomposition algorithm for solving this problem, and report some computational results of its computer implementation.

A second issue we should point out is that SVM's **only** concentrate on minimizing A, that is, once the kernel and its parameters have been selected, R is a constant and A is the only variable quantity we can therefore consider. The question is: can one select a *better* set of kernel parameters (degree of polynomial, sigma of Gaussian, etc.) or a better data representation (scaling, location of origin, etc) so that the bound on h is smaller and better generalization is expected?

The last issue we want to point out is that since the kernel parameters influence R, then a tighter VC-bound can most likely be obtained if instead of minimizing just A (as SVM's currently do), we concentrate on minimizing RA. In this way we can not only find the SVM decision surface, but also the optimal one in the sense of having a tighter VC-bound. In section 5.3 we propose a non-linearly constrained problem that models the desired solution. Some preliminary results obtained on toy problems are also discussed in that section.

5.2 Computing the Radius R

As we defined in the previous section, R corresponds to the radius of the smallest sphere that contains the data points. In the case where the SVM is performing a linear separation on a set of data points, R is computed in the input space. However,

when a non-linear kernel is used to generate a non-linear SVM, the radius R has to be computed in what we have defined as *feature space*, since it is in that space where the SVM is building the linear decision surface.

This section concentrates on the formulation and solution of the problem of finding R. The outline is as follows: in section 5.2.1 we explore the connection between the problem at hand and that of *facility location*, giving also a brief literature review on several papers in the topic. In section 5.2.2 we formulate the problem as a convex QP, in section 5.2.3 we develop a decomposition algorithm for finding R in large data sets, in section 5.2.4 we consider its geometric interpretation, and in section 5.2.5 we describe computational results on its computer implementation.

5.2.1 Previous Work

The minimum covering sphere problem, with many applications in location theory [1], is that of finding the sphere of smallest radius which encloses a set of points in $\Re^n$. This can be stated more formally as:

$$\min_{p} \max_{i=1,\ldots,m} distance(p, p_i)$$

where:

$p_i \in \Re^n \quad i = 1, \ldots, m$ are given points, and

p is a variable point, center of the sphere.

The literature involving facility location problems is abundant. In this section we briefly report on some papers, giving additional atention to those directly related to our specific purpose, and therefore considered more relevant.

The work done by Elzinga and Hearn [33] is probably one of the most relevant for the problem at hand. Their contribution in the cited paper can be summarized as follows:

[1] The connection between the problem of finding the radius R and location theory was pointed out by Professor Robert Freund during the presentation of the thesis proposal. Most of the relevant results had been obtained before this happenned, but we are grateful to Professor Freund for pointing it out, as it has justified and somewhat enhanced our approach.

- Formulate the primal problem for n-dimensional points and Euclidean distance.
- Formulate the dual and show that Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality.
- Show that the center p can be written as a convex combination of data points. By Carathéodory's [2] theorem, this means that at most $n + 1$ data points are needed.
- Formulate s simplex-like algorithm for its solution using $n + 2$ variables at a time.

In additional work, Elzinga and Hearn [32] extend their approach to include rectilinear distance (which is more appropiate for urban environments), and using geometrical arguments, justify an algorithm that is well suited for solving problems in $\Re^2$. The complexity of this algorithm is analyzed by Drezner and Shelah in [29].

More contribution from these authors include the solution of the problem of finding the smallest sphere enclosing a convex polyhedron [35] when the extreme points are unknown. Their solution is achieved by systematically generating larger enclosing spheres and is more efficient than enumerating and using all the extreme points, since their number is potentially very large.

Hearn and Vijay [47], constraining the problem to points in $\Re^2$, extend Elzinga and Hearn's [33] approach to a more general *weighted* distance version of the problem. More importantly, they report computational experiments where they compare heuristics and some of the algorithms known by then. In particular they discuss the importance of a good initial solution using work done by Chakraborty and Chaudhuri [22]. A very related problem involving the weighted sum of distances is also approached by Kuhn in [55].

Facility location theory is also very rich in papers that deal with multifacility location, being in some cases a valid generalization of the single one.

Using rectilinear distances and approaching the cost function as the sum of distances (no allocation or assignment considered), Wesolowsky [113] formulates the

problem as a parametric linear program. A later paper by Elzinga and Hearn [34] simplifies his formulation to non-parametric LP. Morris [68] later described a simplification of the same problem than can be solved in closed form.

Using Euclidean distances and still aproaching the problem as the sum of distances, Love [59] formulates the problem in $\Re^3$ and explores the convexity and duality properties of the problem, Francis and Cabot [37] explore more of its properties in $\Re^2$, while Calamai *et al.* [17][18] formulate algorithms to solve it in $\Re^n$. Extensions to deal with other distances (ℓ_p with $1 < p < \infty$) can be found in work done by Calamai and Conn [19].

A more interesting approach to the multifacility problem not using the sum of distances but a true minimax criteria is explored by Elzinga *et al.* in [36]. In this paper, they basically extend their previus work reported in [33], study the primal formulation of the problem in $\Re^n$, and solve its dual.

Chen [24], along the same line as Elzinga *et al.* in [36], gives an excellent explanation about the computational differences between the single and the multifacility instances, and about where non-differentiability arises. More recently, Bongartz *et al.* [7] extend the work to other distances (ℓ_p with $1 < p < \infty$).

5.2.2 Problem Formulation

In this section we describe the mathematical formulation for computing the radius R. The formulation is derived by steps: we first consider the simplest case, which is finding R in input space (this corresponds to using a linear kernel); then we extend this formulation to the general case of non-linear kernels.

Finding R in input space

Given P, a set of m points $\mathbf{x}_i \in \Re^n$, the problem of finding R can be formulated as:

$$\min_{a} \max_{i=1,\ldots,m} distance(\mathbf{a}, \mathbf{x}_i)$$

where:

$\mathbf{x}_i \in \Re^n \quad i = 1, \ldots, m$ are given points,

$\mathbf{a} \in \Re^n$ is a variable point, center of the sphere, and

$$R = \max_{i=1,\ldots,m} distance(\mathbf{a}^*, \mathbf{x}_i)$$

When we are working with Euclidean distances, we can rewrite $distance(\mathbf{a}, \mathbf{x}_i)$ as:

$$distance(\mathbf{a}, \mathbf{x}_i) = \sqrt{(\mathbf{a} - \mathbf{x}_i)^T(\mathbf{a} - \mathbf{x}_i)}$$

By letting r be the square of the radius R centered at $\mathbf{a}$, we have the following equivalent formulation:

$$\begin{array}{l} \text{Minimize} \quad r \\ \\ \quad \mathbf{a}, r \\ \\ \text{subject to} \end{array} \tag{5.3}$$

$$(\mathbf{a} - \mathbf{x}_i)^T(\mathbf{a} - \mathbf{x}_i) \leq r \quad i = 1, \ldots, m$$

This problem will be referred to as the primal problem.

We construct the Lagrangian:

$$\begin{aligned} L(\mathbf{a}, r, \mathbf{\Lambda}) &= r + \textstyle\sum_{i=1}^m \lambda_i[(\mathbf{a} - \mathbf{x}_i)^T(\mathbf{a} - \mathbf{x}_i) - r] \\ &= r + \textstyle\sum_{i=1}^m \lambda_i[\mathbf{x}_i^T\mathbf{x}_i + \mathbf{a}^T\mathbf{a} - 2\mathbf{x}_i^T\mathbf{a} - r] \end{aligned} \tag{5.4}$$

where $\mathbf{\Lambda} = (\lambda_1, \ldots, \lambda_m)$ is the vector of non-negative Lagrange multipliers corresponding to the constraints in (5.3).

The solution to this optimization problem is determined by a saddle point of this Lagrangian, which has to be minimized with respect to $\mathbf{a}$ and r, and maximized with respect to $\mathbf{\Lambda} \geq 0$. Differentiating (5.4) and setting the results equal to zero we obtain:

$$\frac{\partial L(\mathbf{a}, r, \mathbf{\Lambda})}{\partial r} = 1 - \sum_{i=1}^{m} \lambda_i = 0 \tag{5.5}$$

$$\frac{\partial L(\mathbf{a}, r, \mathbf{\Lambda})}{\partial \mathbf{a}} = \mathbf{a} \sum_{i=1}^{m} \lambda_i - \sum_{i=1}^{m} \lambda_i \mathbf{x}_i = 0 \tag{5.6}$$

Using the superscript $*$ to denote the *optimal* values of the cost function and substituting (5.5) in (5.6) we obtain:

$$\mathbf{a}^* = \sum_{i=1}^{m} \lambda_i^* \mathbf{x}_i \tag{5.7}$$

which combined with (5.5) shows that the center $\mathbf{a}$ can be written as a convex combination of data points. As we mentioned before, this was also pointed out by Elzinga and Hearn in [33].

Substituting (5.5) and (5.7) in (5.4) we obtain:

$$\begin{aligned} F(\mathbf{\Lambda}) &= r + \textstyle\sum_{i=1}^{m} \lambda_i [\mathbf{x}_i^T \mathbf{x}_i + \mathbf{a}^T \mathbf{a} - 2\mathbf{x}_i^T \mathbf{a}] - r \sum_{i=1}^{m} \lambda_i \\ &= \textstyle\sum_{i=1}^{m} \lambda_i [\mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \sum_{j=1}^{m} \lambda_j \mathbf{x}_j] + \mathbf{a}^T \mathbf{a} \sum_{i=1}^{m} \lambda_i \\ &= \textstyle\sum_{i=1}^{m} \lambda_i \mathbf{x}_i^T \mathbf{x}_i - 2 \sum_{i,j=1}^{m} \lambda_i \lambda_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i,j=1}^{m} \lambda_i \lambda_j \mathbf{x}_i^T \mathbf{x}_j \\ &= \textstyle\sum_{i=1}^{m} \lambda_i \mathbf{x}_i^T \mathbf{x}_i - \sum_{i,j=1}^{m} \lambda_i \lambda_j \mathbf{x}_i^T \mathbf{x}_j \end{aligned} \tag{5.8}$$

By incorporating constraint (5.5) and the non-negativity of $\mathbf{\Lambda}$ we get the following dual quadratic program:

$$\begin{aligned} &\text{Maximize} \quad F(\mathbf{\Lambda}) = \textstyle\sum_{i=1}^{m} \lambda_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{\Lambda}^T Q \mathbf{\Lambda} \\ &\quad \mathbf{\Lambda} \\ &\text{subject to} \\ &\qquad \mathbf{\Lambda}^T \mathbf{1} = 1 \\ &\qquad \mathbf{\Lambda} \geq 0 \end{aligned} \tag{5.9}$$

where Q is a symmetric $m \times m$ matrix with elements $Q_{ij} = \mathbf{x}_i^T \mathbf{x}_j$.

This problem will be referred to as the dual problem, and its derivation also

appears in [33].

Notice that complementary slackness conditions of the form:

$$\lambda_i^*[(\mathbf{a}^* - \mathbf{x}_i)^T(\mathbf{a}^* - \mathbf{x}_i) - r^*] = 0 \qquad i = 1, \ldots, m \tag{5.10}$$

imply that $\lambda_i > 0$ only when the constraint in (5.3) is active. The data points for which $\lambda_i > 0$ are therefore on the *surface* of the sphere.

The radius R can thus be computed as:

$$R = \sqrt{(\mathbf{a}^* - \mathbf{x}_i)^T(\mathbf{a}^* - \mathbf{x}_i)}$$

for any $\mathbf{x}_i$ such that $\lambda_i > 0$. By substituting $\mathbf{a}^*$ with equation (5.7) we obtain:

$$R = \sqrt{\mathbf{x}_i^T\mathbf{x}_i - 2\sum_{j=1}^{m}\lambda_j\mathbf{x}_j^T\mathbf{x}_i + \sum_{k,j=1}^{m}\lambda_k\lambda_j\mathbf{x}_k^T\mathbf{x}_j}$$

By algebraic manipulation (using the fact that $\sum_{i=1}^{m}\lambda_i r_i = r$) we can obtain the expected duality result:

$$R^2 = F(\mathbf{\Lambda}^*)$$

where $\mathbf{\Lambda}^*$ is the optimal solution of (5.9).

Finding R in feature space

As we did in chapter 2 (section 2.3.1) we notice that by replacing the dot product operation in the expressions derived for finding R in input space, with the kernel operator, we obtain the QP:

$$\begin{aligned} \text{Maximize} \quad & F(\mathbf{\Lambda}) = \sum_{i=1}^{m} \lambda_i K(\mathbf{x}_i, \mathbf{x}_i) - \mathbf{\Lambda}^T Q \mathbf{\Lambda} \\ & \mathbf{\Lambda} \\ \text{subject to} \quad & \\ & \mathbf{\Lambda}^T \mathbf{1} = 1 \\ & \mathbf{\Lambda} \geq 0 \end{aligned} \tag{5.11}$$

where Q is a symmetric $m \times m$ matrix with elements $Q_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$.

Similarly, the radius R can now be computed as:

$$R = \sqrt{K(\mathbf{x}_i, \mathbf{x}_i) - 2\sum_{j=1}^{m} \lambda_j K(\mathbf{x}_j, \mathbf{x}_i) + \sum_{k,j=1}^{m} \lambda_k \lambda_j K(\mathbf{x}_k, \mathbf{x}_j)} \tag{5.12}$$

for any $\mathbf{x}_i$ such that $\lambda_i > 0$.

In the next section, we focus on an algorithm devised to solve the QP given by (5.11).

5.2.3 A Decomposition Algorithm for large data-sets

As in the case of training a SVM using large data sets (above $\approx$ 5,000 samples), this is a very difficult problem to approach without some kind of decomposition. To give an idea of some memory requirements, an application like the one described later in section 6.1 involves 50,000 training samples, and this amounts to a quadratic form whose matrix Q has $2.5 \cdot 10^9$ entries that would need, using an 8-byte floating point representation, 20,000 Megabytes = 20 Gigabytes of memory!

In order to solve this problem efficiently, we take explicit advantage of the geometric interpretation mentioned in Section 5.2.2, which states that only points on the *surface* of the sphere have a corresponding Lagrange multiplier $\lambda_i > 0$. If we consider the quadratic programming problem given by (5.11), this interpretation translates into having many of the components of $\mathbf{\Lambda}$ equal to zero.

In order to decompose the original problem, one can think of solving iteratively

the system given by (5.11), but keeping fixed at zero level those components λ_i associated with data points that are not on the *surface* of the sphere, and therefore only optimizing over a reduced set of variables.

Before we formulate the decomposition algorithm we first examine the optimality conditions.

Optimality Conditions

As in previous chapters, we rewrite the quadratic program (5.11) in minimization form as:

$$\begin{aligned} \text{Minimize} \quad & F(\boldsymbol{\Lambda}) = -\sum_{i=1}^{m} \lambda_i K(\mathbf{x}_i, \mathbf{x}_i) + \boldsymbol{\Lambda}^T Q \boldsymbol{\Lambda} \\ \boldsymbol{\Lambda} & \\ \text{subject to} & \\ & \boldsymbol{\Lambda}^T \mathbf{1} = 1 \qquad (\mu) \\ & -\boldsymbol{\Lambda} \leq \mathbf{0} \qquad (\boldsymbol{\Pi}) \end{aligned} \tag{5.13}$$

where μ and $\boldsymbol{\Pi} = (\pi_1, \ldots, \pi_m)$ are the associated Karush-Kuhn-Tucker multipliers.

Since Q is a positive semi-definite matrix (see end of section 2.3.1) and the constraints in (5.13) are linear, the Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality, and they are:

$$\begin{aligned} \nabla F(\boldsymbol{\Lambda}) - \boldsymbol{\Pi} + \mu \mathbf{1} &= \mathbf{0} \\ \pi_i \cdot \lambda_i &= 0 \quad i = 1, \ldots, m \\ \boldsymbol{\Pi}, \boldsymbol{\Lambda} &\geq \mathbf{0} \\ \boldsymbol{\Lambda}^T \mathbf{1} &= 0 \end{aligned} \tag{5.14}$$

We consider the two possible values that each component of $\boldsymbol{\Lambda}$ can have:

1. **Case:** $\lambda_i > 0$: From the first two equations of the KKT conditions we have:

$$2(Q\Lambda)_i - K(\mathbf{x}_i, \mathbf{x}_i) + \mu = 0 \tag{5.15}$$

which translates into:

$$\mu = K(\mathbf{x}_i, \mathbf{x}_i) - 2\sum_{j=1}^{m} \lambda_j K(\mathbf{x}_i, \mathbf{x}_j)$$

2. **Case:** $\lambda_i = 0$: From the first KKT condition we have:

$$\pi_i = 2(Q\Lambda)_i - K(\mathbf{x}_i, \mathbf{x}_i) + \mu$$

In order to enforce $\pi_i \geq 0$, we require that the above expression is non-negative. To do this, we use a point $\mathbf{x}_k$ such that $\lambda_k > 0$ and use the first case derived before. This translates into:

$$\pi_i = 2\sum_{j=1}^{m} \lambda_j [K(\mathbf{x}_i, \mathbf{x}_j) - K(\mathbf{x}_k, \mathbf{x}_j)] + K(\mathbf{x}_k, \mathbf{x}_k) - K(\mathbf{x}_i, \mathbf{x}_i)$$

A careful inspection of the formula for R given in equation (5.12) reveals that:

$$\pi_i = R^2 - distance^2(\mathbf{x}_i, a)$$

which geometrically can be seen as a confirmation that points in the *interior* of the sphere have $\lambda_i = 0$.

The Algorithm:

We again define a fixed-size working set B, such that $|B| \leq m$, and $|B|$ is large enough to contain all the data points for which $(\lambda_i > 0)$, but small enough such that the computer can handle it and optimize it using some solver. We also define N, such that it contains the data points for which $(\lambda_j = 0)$. Then the decomposition

algorithm can be stated as follows:

1. Arbitrarily choose $|B|$ points from the data set.
2. Solve the subproblem defined by the variables in B.
3. While there exists some $j \in N$, such that $\pi_j < 0$ (π_j defined as above) replace any $\lambda_i = 0$, $i \in B$, with $\lambda_j = 0$ and solve the new subproblem.

Notice that this algorithm will strictly improve the objective function at each iteration and therefore will not cycle. Since the radius R is finite and weak duality implies $F(\mathbf{\Lambda}) \leq R$, then the algorithm must converge to the global optimal solution in a finite number of iterations.

5.2.4 Geometric Interpretation of the Algorithm

Figure 5-1 shows an example of the execution of the algorithm. In (a), we see the algorithm being run with the black and gray points in the working set B, and the clear points in N. After the first iteration, we obtain two sets of zero (gray points) and non-zero (black points) coefficients. The black points correspond to *surface* points.

In a second iteration (b), we replace the gray points with points outside the sphere and re-optimize. After every iteration, the sphere grows in size, and the algorithm continues until all the points satisfy the optimality conditions.

To summarize, the two main geometrical ideas behind this algorithm are:

1. Points in N that violate optimality conditions correspond to points located outside the *current* sphere.
2. Only points in the surface of the sphere end up with non-zero coefficients (λ_i).

5.2.5 Experimental Results

The computational results that we present in this section have been obtained using real data from our Face Detection System, which is described in Section 6.1.

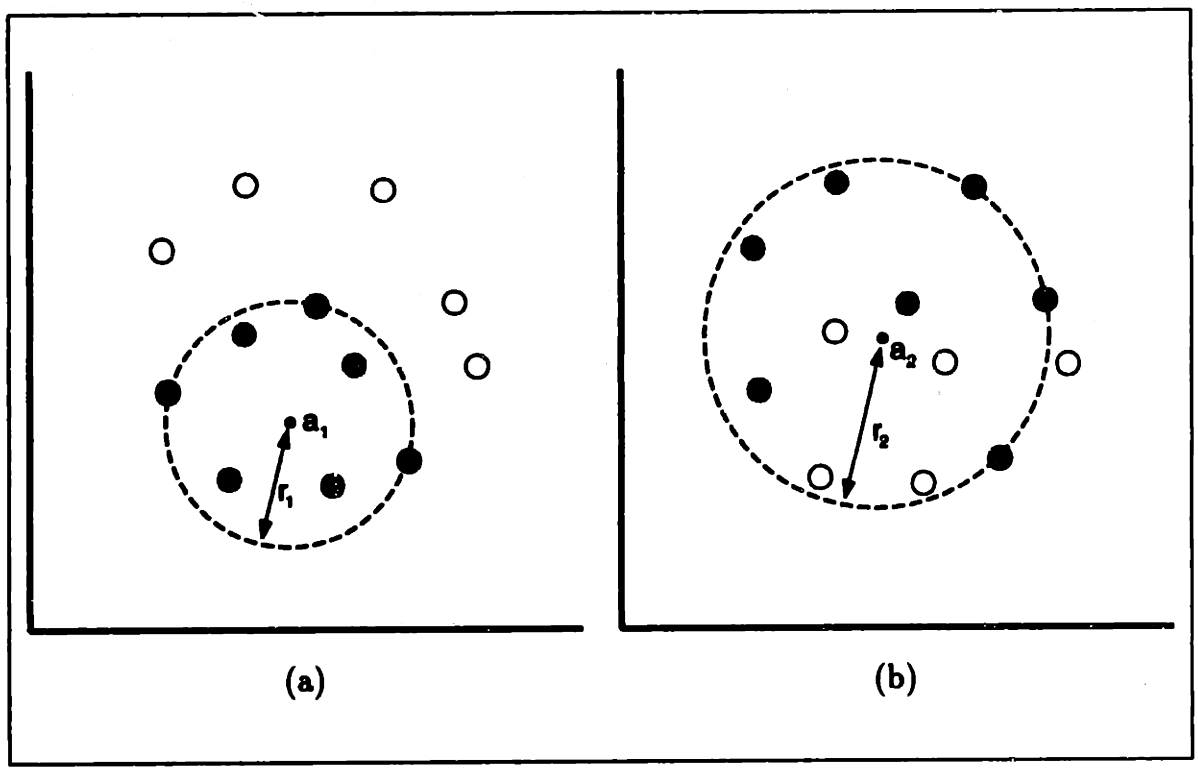


Figure 5-1: (a)First iteration of the algorithm starting with a random set of points and finding the sphere that contains them. (b) Points that are left outside of the sphere (clear color in (a)) replace in the working set points located in the *interior* of the sphere (gray color in (a)). This continues until optimality.

Figure 5-2 shows the execution time as a function of the number of data points considered and the size of the working set. In particular, we observe a linear increase in the execution time as a function of the size of the working set. However, we remind the reader that since we do not know how many *surface* points will appear in the solution (nor how many we will encounter during the solution process), estimating a *small-enough* working set size is at the moment very empirical.

Figure 5-3 shows the size of the radius R and the number of *surface* points as a function of the number of points considered. An interesting result to remark: This data is of dimension 283 and we are using a 2nd-degree homogeneous polynomial. This means that the dimensionality of feature space is $\approx$ 80,000, which is noticeable larger than 32, which is the number of *surface* points found when considering 50,000 data points. This interesting result should be explored further, since it might be uncovering important properties of the high-dimensional *feature space.*

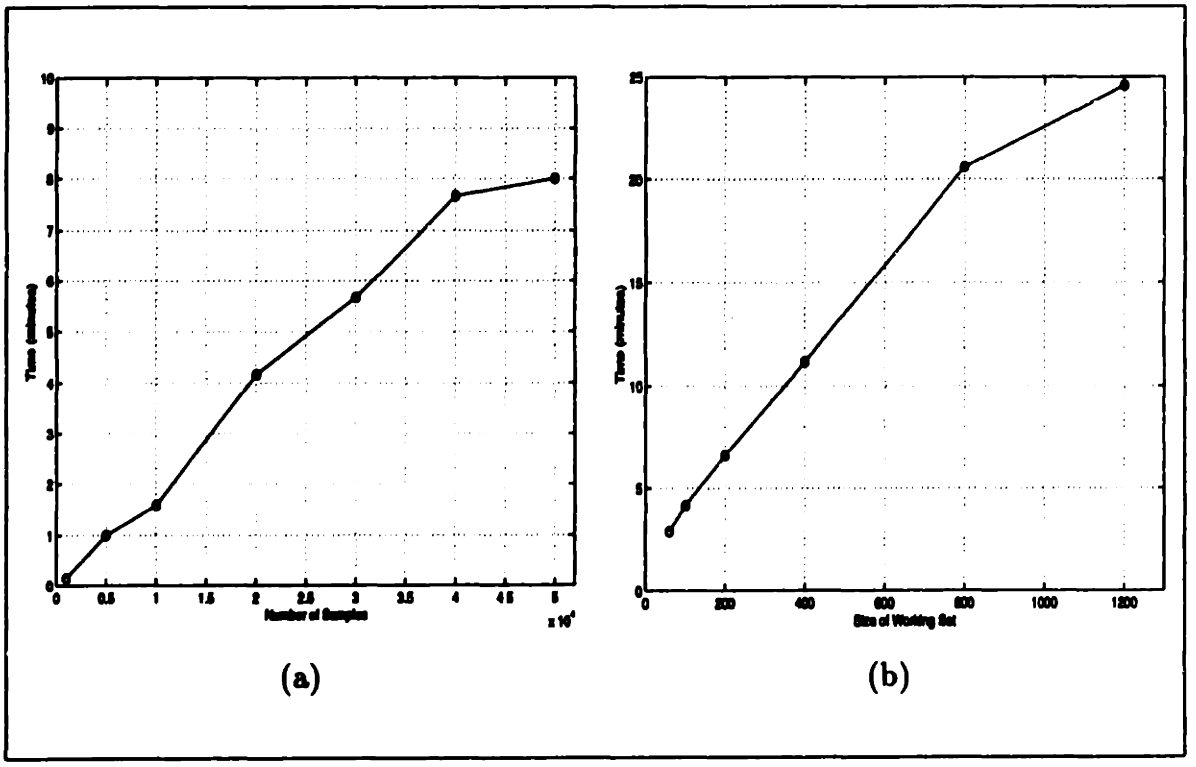


Figure 5-2: (a) Training time Vs. Number of data points. A working set of size 100 was used for this experiment. (b) Training time Vs. Size of the working set.

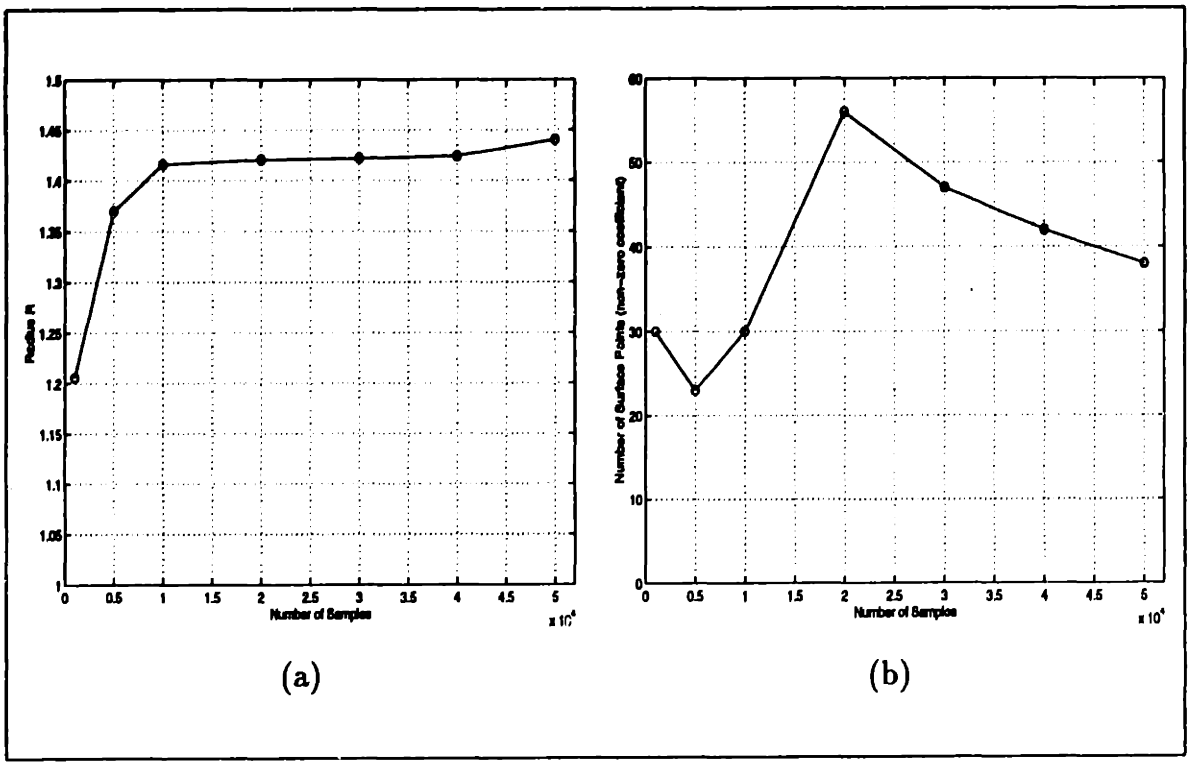


Figure 5-3: (a) Radius R Vs. Number of data points. (b) Number of non-zero coefficients (surface points) Vs. Number of data points.

5.3 Using the VC-Bound to choose the *best* SVM

The previous section explains how to formulate and computationally solve the problem of finding R. This means that the upper bound on the VC-dimension h given by (5.2) can be computed, and using it, we can compute the VC-bound (5.1) for a predetermined set of parameters. This is already an achievement. However, as we introduced at the beginning of this chapter, the question is: can one select a *better* set of kernel parameters (degree of polynomial, sigma of Gaussian, etc.) or a better data representation (scaling, location of origin, etc.) so that the bound on h is smaller and better generalization is expected? Moreover, the objective is to formulate a mathematical program to model the minimization of the VC-bound over the set of kernel parameters. We will explain this approach in section 5.3.2.

5.3.1 Previous Work

There are very few references that consider the use of the approximate VC-bound to estimate the optimal kernel parameters.

Schölkopf et. al. describe in [91] how they use the VC-bound to select the degree of the polynomial kernel in their OCR application. The approach used was to train the SVM and find R for polynomials ranging from degrees 2 to 7, to then select the lowest VC-bound. This experiments are also reported in [107], and in most cases show that a low VC-bound implies a better generalization.

Joachins [52] does not compute R, but bounds it by pre-normalizing his data. He then uses the VC-bound to estimate the best parameter setting (degree of polynomial and sigma of the Gaussian[2]). His results also reinforce the idea that in most cases a low VC-bound suggests a better generalization.

A recent tutorial by Burges [14] reports on the observation that although the estimated VC-bound is *loose*, its behavior seems to be highly *predictive* of the actual risk.

In a draft of his most recent book, Vapnik [108] also suggests that the ultimate

[2]He uses a discrete step of 0.2 to cope with the continuous nature of sigma.

goal is that of minimizing R^2W^2, but no mathematical formulation is given.

5.3.2 An Alternative formulation for Structural Risk Minimization

We start this section by defining a *parameterized kernel* as a function $K(\mathbf{x}, \mathbf{y}, \beta)$ such that β is a subset of the parameters of the kernel $K(\mathbf{x}, \mathbf{y})$. For example, we can have:

- Degree of polynomial:

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

- Data scaling:

$$K(\mathbf{x}, \mathbf{y}, \beta) = (\sum_{i=1}^{dim(input)} \mathbf{x}_i\mathbf{y}_i\beta_i + 1)^d$$

- Location of the origin:

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

- Weight of the linear terms:

$$K(\mathbf{x}, \mathbf{y}, \beta) = (\mathbf{x}^T\mathbf{y} + \beta)^d$$

- Sigma of the Gaussian RBF:

$$K(\mathbf{x}, \mathbf{y}, \beta) = e^{\beta\|\mathbf{x}-\mathbf{y}\|}$$

Our objective will be to find an optimal setting of the kernel parameters with respect to the VC-bound. In order to do this, we aim at minimizing a cost function that involves R^2W^2 and some penalty (in our case linear) on how correctly the data is separated. The first term controls the structure, while the second one the empirical risk.

Before we write the formulation, we formulate the dual QP (5.11) used for computing R as:

$$\begin{aligned}
&\text{Minimize} \quad r \\
&\quad \Pi, r \\
&\text{subject to} \\
&\qquad K(\mathbf{x}_i,\mathbf{x}_i) - 2\sum_{j=1}^{m}\pi_j K(\mathbf{x}_j,\mathbf{x}_i) + \sum_{k,j=1}^{m}\pi_k\pi_j K(\mathbf{x}_k,\mathbf{x}_j) \leq r \quad i = 1,\ldots,m \\
&\qquad \Pi^T\mathbf{1} = 1 \\
&\qquad \Pi, r \geq 0
\end{aligned} \tag{5.16}$$

where the set of constraints are equivalent to requiring: distance(sphere center,$\mathbf{x}_i$) $\leq r$ measured in feature space.

As we did in chapter 4, we write the primal reformulation of the SVM training problem as:

$$\begin{aligned}
&\text{Minimize} \quad F(\Lambda, b, \Xi) = \tfrac{1}{2}\Lambda^T Q \Lambda + C\sum_{i=1}^{m}\xi_i \\
&\text{subject to} \\
&\qquad y_i(\sum_{j=1}^{m}\lambda_j y_j K(\mathbf{x}_i,\mathbf{x}_j) + b) \geq 1-\xi_i \qquad i = 1\ldots m \\
&\qquad \xi_i, \lambda_i \geq 0 \qquad i = 1\ldots m \\
&\qquad b \quad free
\end{aligned} \tag{5.17}$$

where $Q_{ij} = y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$.

By combining the formulations (5.16) and (5.17), we obtain:

$$\text{Minimize} \quad r[\sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j, \beta)] + C\sum_{i=1}^{m} \xi_i$$

$$\mathbf{\Pi}, r, \mathbf{\Lambda}, b, \mathbf{\Xi}, \beta$$

subject to

$$\begin{aligned}
K(\mathbf{x}_i, \mathbf{x}_i, \beta) - 2\sum_{j=1}^{m} \pi_j K(\mathbf{x}_j, \mathbf{x}_i, \beta) + \sum_{k,j=1}^{m} \pi_k \pi_j K(\mathbf{x}_k, \mathbf{x}_j, \beta) &\leq r \quad i = 1, \ldots, m \\
y_i[\sum_{j=1}^{m} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j, \beta) + b] + \xi_i &\geq 1 \quad i = 1, \ldots, m \\
\mathbf{\Pi}^T \mathbf{1} &= 1 \\
\mathbf{\Pi}, r, \mathbf{\Lambda}, \mathbf{\Xi} &\geq 0 \\
b &\quad free
\end{aligned} \tag{5.18}$$

where β might need to be constrained so that $K(\mathbf{x}, \mathbf{y}, \beta)$ is a valid Mercer kernel (see chapter 2, page 55).

This formulation can be easily adapted to only compute the radius R enclosing the support vectors by adding appropriate slack variables, penalties and constraints. This may be more appropriate if we consider recent bounds proposed by Vapnik in [108].

We must remark that this problem can be hard to solve due to the nature of the constraints. Further study of this formulation and its solution should be considered in the near future.

Computational Results on a toy problem:

We solved formulation (5.18) for a toy problem with 20 data points and 2-dimensional input using MINOS 5.4. The purpose was to verify if the formulation would yield the desired solution and to compare its value with the one obtained using a good *initial guess* for the kernel parameters. Input data in $\Re^2$ was also very convenient because it allowed us to plot the *VC-surface* (which is nothing else but the cost function of problem (5.18)) as a function of the kernel parameters.

We used a parameterized kernel of the form:

$$K(\mathbf{x}, \mathbf{y}, \beta) = (\sum_{i=1}^{2} \mathbf{x}_i \mathbf{y}_i \beta_i + 1)^2$$

to explore the scaling parameter β. This kernel is particularly interesting, since it can be seen as a *feature selector*, i.e., under reasonable initial scaling, β_i weights the importance of the i-th input and can be driven down to zero if that helps in reducing the complexity of the classifier without hurting its discriminative capability.

The data used appears in Figure 5-4(a). For this example, the inital guess was to leave the scaling unchanged, that is, feed the data in its original format. For this initial guess, the function value was:

$$R^2 W^2 + C \sum_{i=1}^{m} \xi_i \;\; = \;\; 23.0360$$

By using the new approach (5.18), we obtained:

$$R^2 W^2 + C \sum_{i=1}^{m} \xi_i \;\; = \;\; 14.4483$$

To double-check the performance of the code, we generated a plot of the VC-surface as a function of the kernel parameters, and obtained a value of 14.4476, which is remerkably close to the one obtained using out formulation. Figure 5-4(b) presents a contour plot of the surface, while Figure 5-5 contains 3D plots of it.

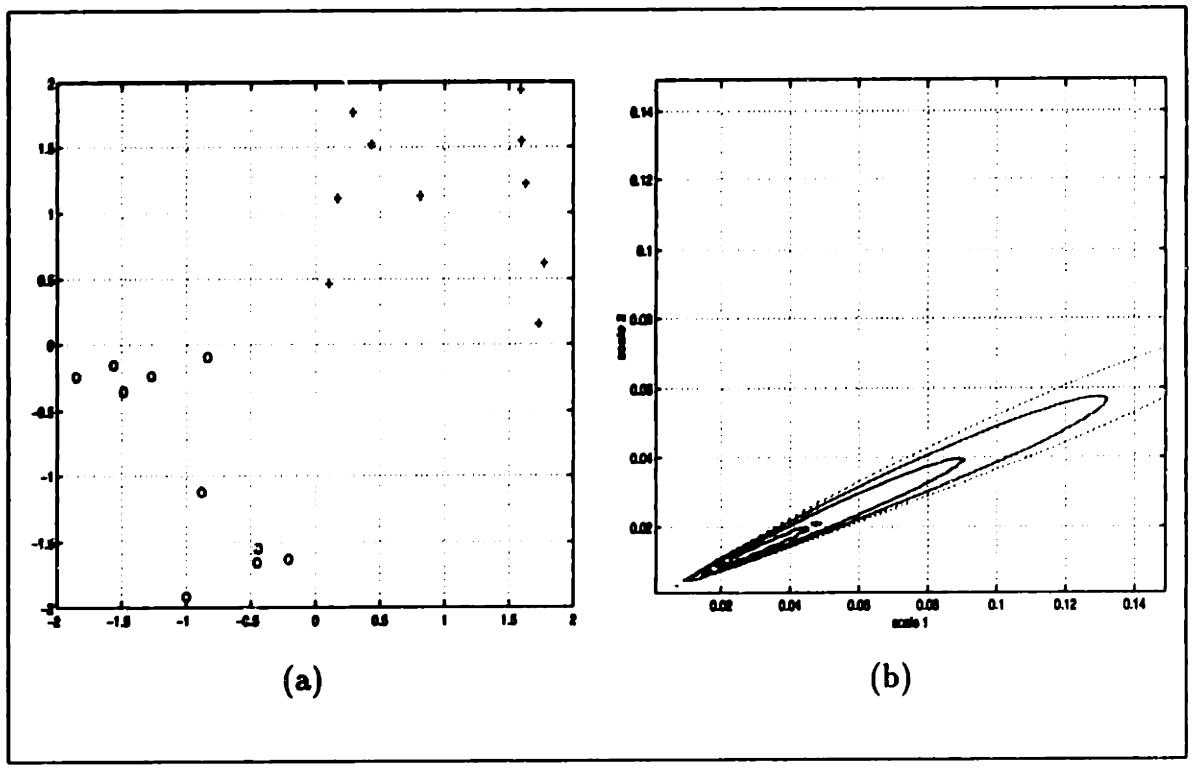


Figure 5-4: (a)The data points used and the labels.(b)The contour plot of the VC-surface for values:14.5, 14.6, 14.7, 14.8, 14.9 and 15

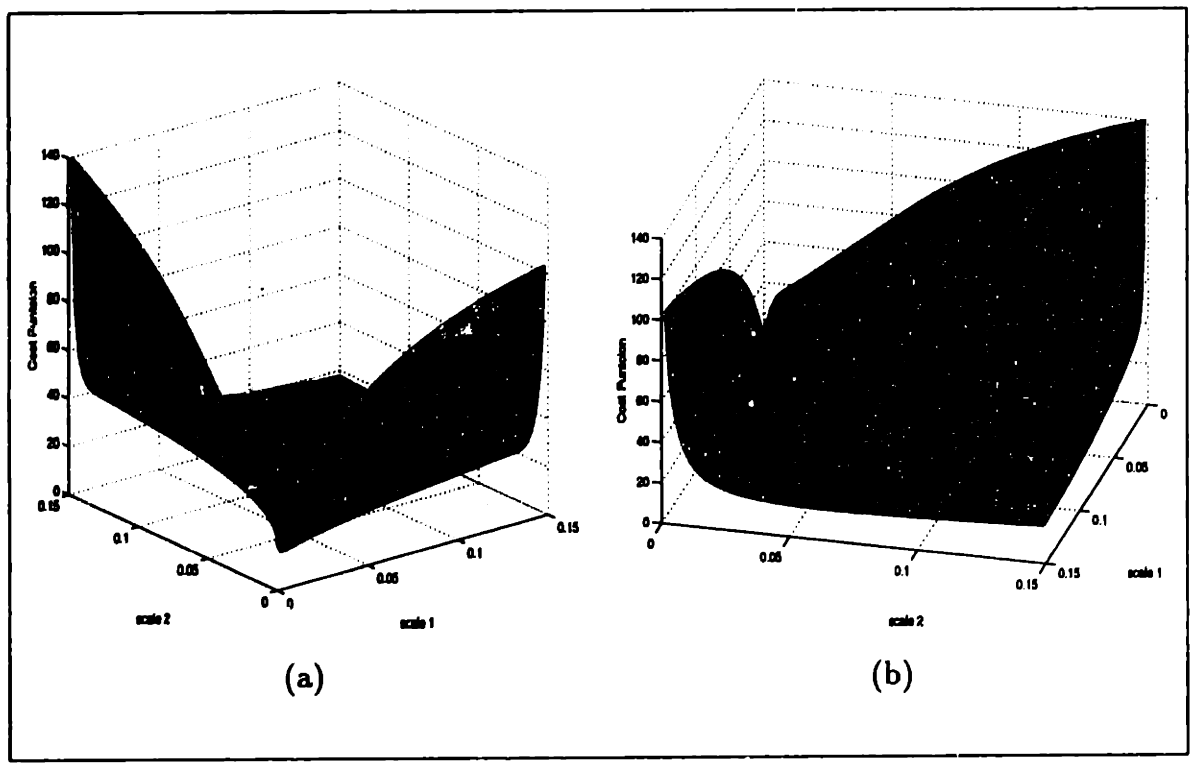


Figure 5-5: The VC-surface from different view points.

Chapter 6

Applications

6.1 Face Detection in Images

This section introduces a Support Vector Machine application for detecting vertically oriented and unoccluded frontal views of human faces in grey level images. It handles faces over a wide range of scales and works under different lighting conditions, even with moderately strong shadows.

The face detection problem can be defined as follows: Given as input an arbitrary image, which could be a digitized video signal or a scanned photograph, determine whether or not there are any human faces in the image, and if there are, return an encoding of their location. The encoding in this system is to fit each face in a bounding box defined by the image coordinates of the corners.

Face detection as a computer vision task has many applications. It has direct relevance to the face recognition problem, because the first important step of a fully automatic human face recognizer is usually identifying and locating faces in an unknown image. Face detection also has potential application in human-computer interfaces, surveillance systems, census systems, etc.

From the standpoint of this paper, face detection is interesting because it is an example of a natural and challenging problem for demonstrating and testing the potentials of Support Vector Machines. There are many other object classes and phenomena in the real world that share similar characteristics, for example, tumor

anomalies in MRI scans, structural defects in manufactured parts, etc. A successful and general methodology for finding faces using SVM's should generalize well for other spatially well-defined pattern and feature detection problems.

It is important to remark that face detection, like most object detection problems, is a difficult task due to the significant pattern variations that are hard to parameterize analytically. Some common sources of pattern variations are facial appearance, expression, presence or absence of common structural features, like glasses or a moustache, light source distribution, shadows, etc.

This system works by testing candidate image locations for local patterns that appear like faces using a classification procedure that determines whether or not a given local image pattern is a face or not. Therefore, the face detection problem is approached as a classification problem given by examples of 2 classes: faces and non-faces.

6.1.1 Previous Systems

The problem of face detection has been approached with different techniques in the last few years. This techniques include Neural Networks [12] [88], detection of face features and use of geometrical constraints [114], density estimation of the training data [63], labeled graphs [54] and clustering and distribution-based modeling [100][99].

Out of all these previous works, the results of Sung and Poggio [100][99], and Rowley *et al.* [88] reflect systems with very high detection rates and low false positive detection rates.

Sung and Poggio use clustering and distance metrics to model the distribution of the face and non-face manifold, and a Neural Network to classify a new pattern given the measurements. The key of the quality of their result is the clustering and use of combined Mahalanobis and Euclidean metrics to measure the distance from a new pattern and the clusters. Other important features of their approach is the use of non-face clusters, and the use of a bootstrapping technique to collect important non-face patterns. One drawback of this technique is that it does not provide a principled way to choose some important free parameters like the number of clusters it uses.

Similarly, Rowley *et al.* have used problem information in the design of a retinally connected Neural Network that is trained to classify faces and non-faces patterns. Their approach relies on training several NN emphasizing subsets of the training data, in order to obtain different sets of weights. Then, different schemes of arbitration between them are used in order to reach a final answer.

The approach to the face detection system with a SVM uses no prior information in order to obtain the decision surface, this being an interesting property that can be exploited in using the same approach for detecting other objects in digital images.

6.1.2 The SVM Face Detection System

This system, as it was described before, detects faces by exhaustively scanning an image for face-like patterns at many possible scales, by dividing the original image into overlapping sub-images and classifying them using a SVM to determine the appropriate class, that is, face or non-face. Multiple scales are handled by examining windows taken from scaled versions of the original image.

Clearly, the major use of SVM's is in the classification step, and it constitutes the most critical and important part of this work. Figure 6-2 gives a geometrical interpretation of the way SVM's work in the context of face detection.

More specifically, this system works as follows:

1. A database of face and non-face 19×19 pixel patterns, assigned to classes +1 and -1 respectively, is trained on, using the support vector algorithm. A 2nd-degree polynomial kernel function and an upper bound $C = 200$ are used in this process obtaining a perfect training error.

2. In order to compensate for certain sources of image variation, some preprocessing of the data is performed:

   - **Masking:** A binary pixel mask is used to remove some pixels close to the boundary of the window pattern allowing a reduction in the dimensionality of the input space from 19×19=361 to 283. This step is important in the

reduction of background patterns that introduce unnecessary noise in the training process.

- **Illumination gradient correction:** A best-fit brightness plane is subtracted from the unmasked window pixel values, allowing reduction of light and heavy shadows.
- **Histogram equalization:** A-histogram equalization is performed over the patterns in order to compensate for differences in illumination brightness, different cameras response curves, etc.

3. Once a decision surface has been obtained through training, the run-time system is used over images that do not contain faces, and misclassifications are stored so they can be used as negative examples in subsequent training phases. Images of landscapes, trees, buildings, rocks, etc., are good sources of false positives due to the many different *textured* patterns they contain. This bootstrapping step, which was successfully used by Sung and Poggio [100] is very important in the context of a face detector that learns from examples because:

   - Although negative examples are abundant, negative examples that are useful from a learning point of view are very difficult to characterize and define.
   - By approaching the problem of object detection, and in this case of face detection, by using the paradigm of binary pattern classification, the two classes, object and non-object are not equally complex since the non-object class is broader and richer, and therefore needs more examples in order to get an accurate definition that separates it from the object class. Figure 6-1 shows an image used for bootstrapping with some misclassifications, that were later used as negative examples.

4. After training the SVM, we incorporate it as the classifier in a run-time system very similar to the one used by Sung and Poggio [100][99] that performs the following operations:

- Re-scale the input image several times.
- Cut 19×19 window patterns out of the scaled image.
- Preprocess the window using masking, light correction and histogran equalization.
- Classify the pattern using the SVM.
- If the class corresponds to a face, draw a regtangle aroung the face in the output image.

Figure 6-3 reflects the system's architecture at run-time.

6.1.3 Experimental Results on Static Images

To test the run-time system, we used two sets of images. The set A, contained 313 high-quality images with same number of faces. The set B, contained 23 images of mixed quality, with a total of 155 faces. Both sets were tested using our system and the one by Sung and Poggio [100][99]. In order to give true meaning to the number of false positives obtained, it is important to state that set A involved 4,669,960 pattern windows, while set B 5,383,682. Table 6.1 shows a comparison between the 2 systems.

	Test Set A		Test Set B	
	Detection Rate	False Detections	Detection Rate	False Detections
Ideal System	100 %	0	100%	0
SVM	97.12 %	4	74.19%	20
Sung & Poggio	94.57 %	2	74.19%	11

Table 6.1: Performance of the SVM face detection system

Figures 6-4, 6-5, 6-6, 6-7, 6-8, 6-9, 6-10 and 6-11 present some output images of our system. These images were not used during the training phase of the system.

6.1.4 Extension to a Real-Time System

The system presented in the previous two sections spends approximately 6 seconds (SparcStation 20) on a 320×240 pixels grey level image. Although this is faster by

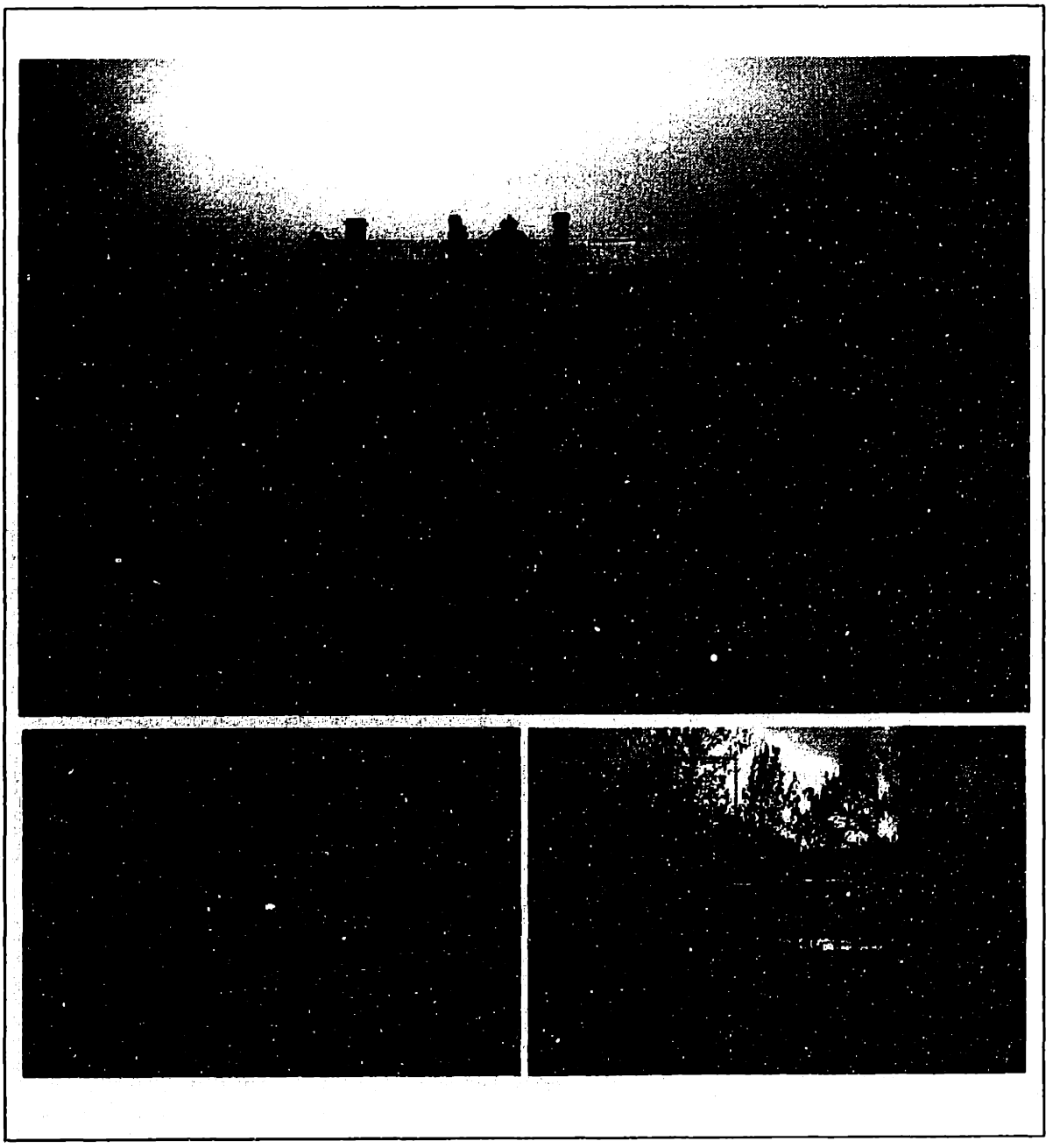


Figure 6-1: Some false detections obtained with the first version of the system. This false positives were later used as negative examples (class -1) in the training process

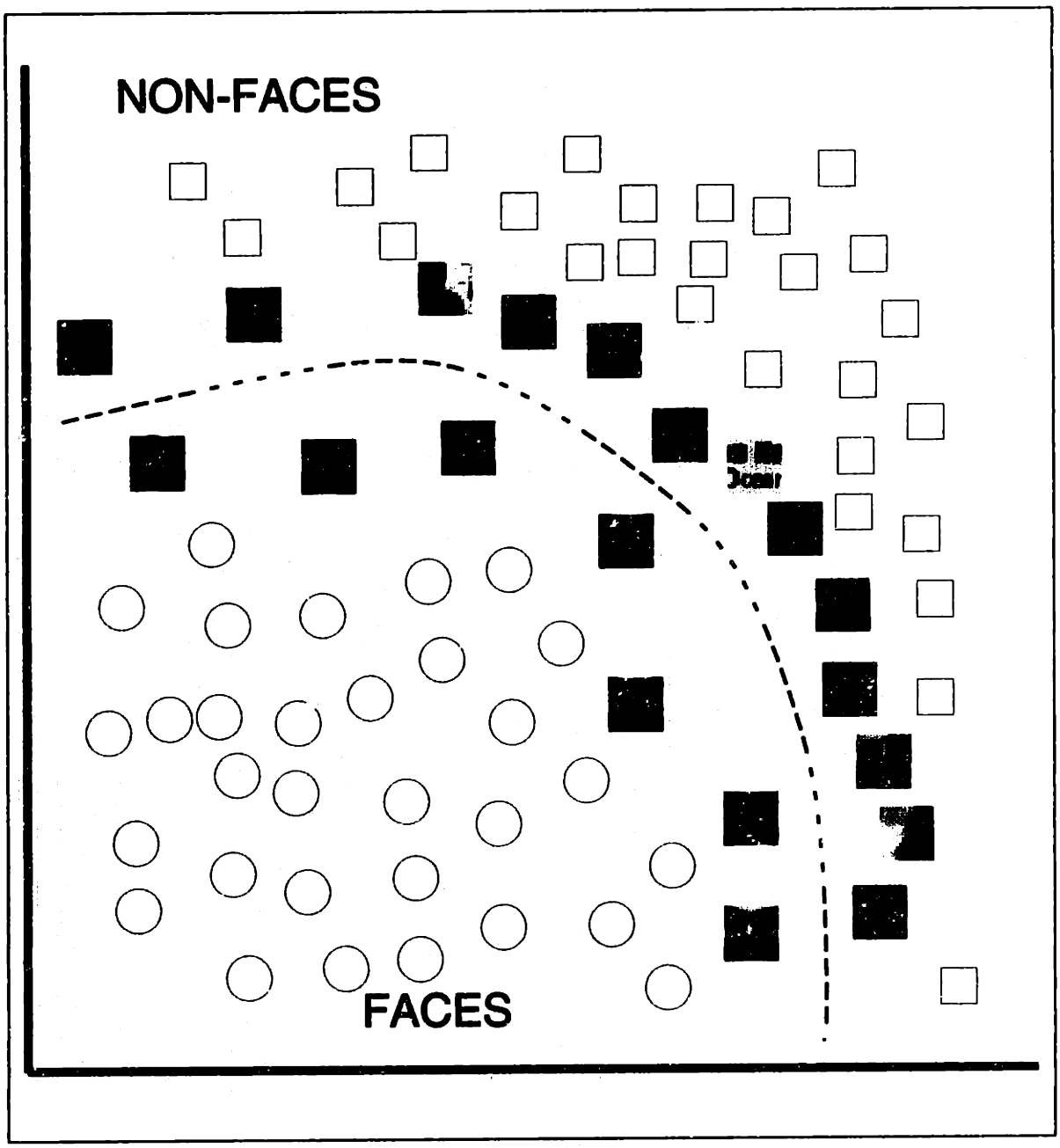


Figure 6-2: Geometrical Interpretation of how the SVM separates the face and non-face classes. The patterns are real support vectors obtained after training the system. Notice the small number of total support vectors and the fact that a higher proportion of them correspond to non-faces.

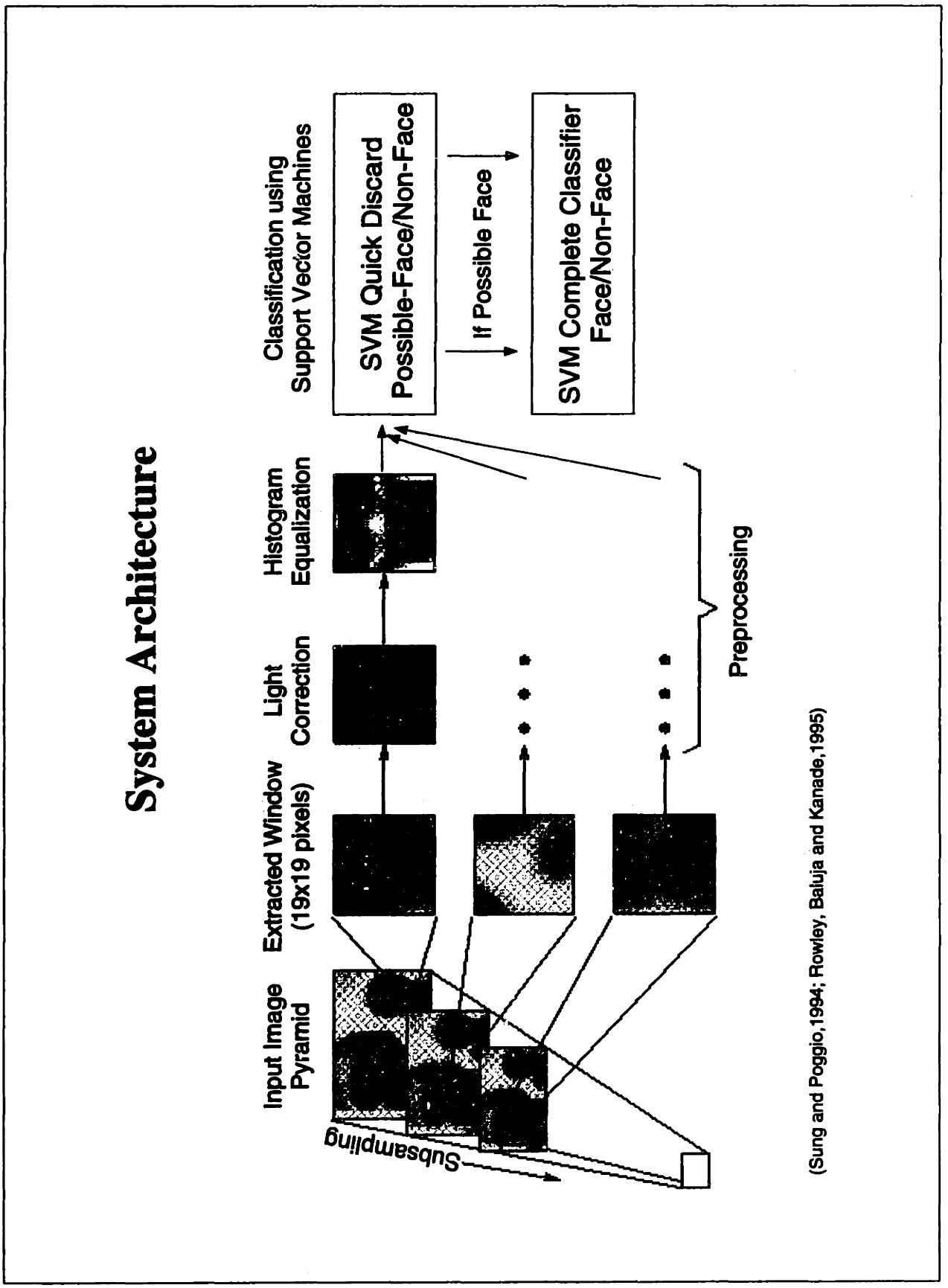


Figure 6-3: System Architecture at Run-Time

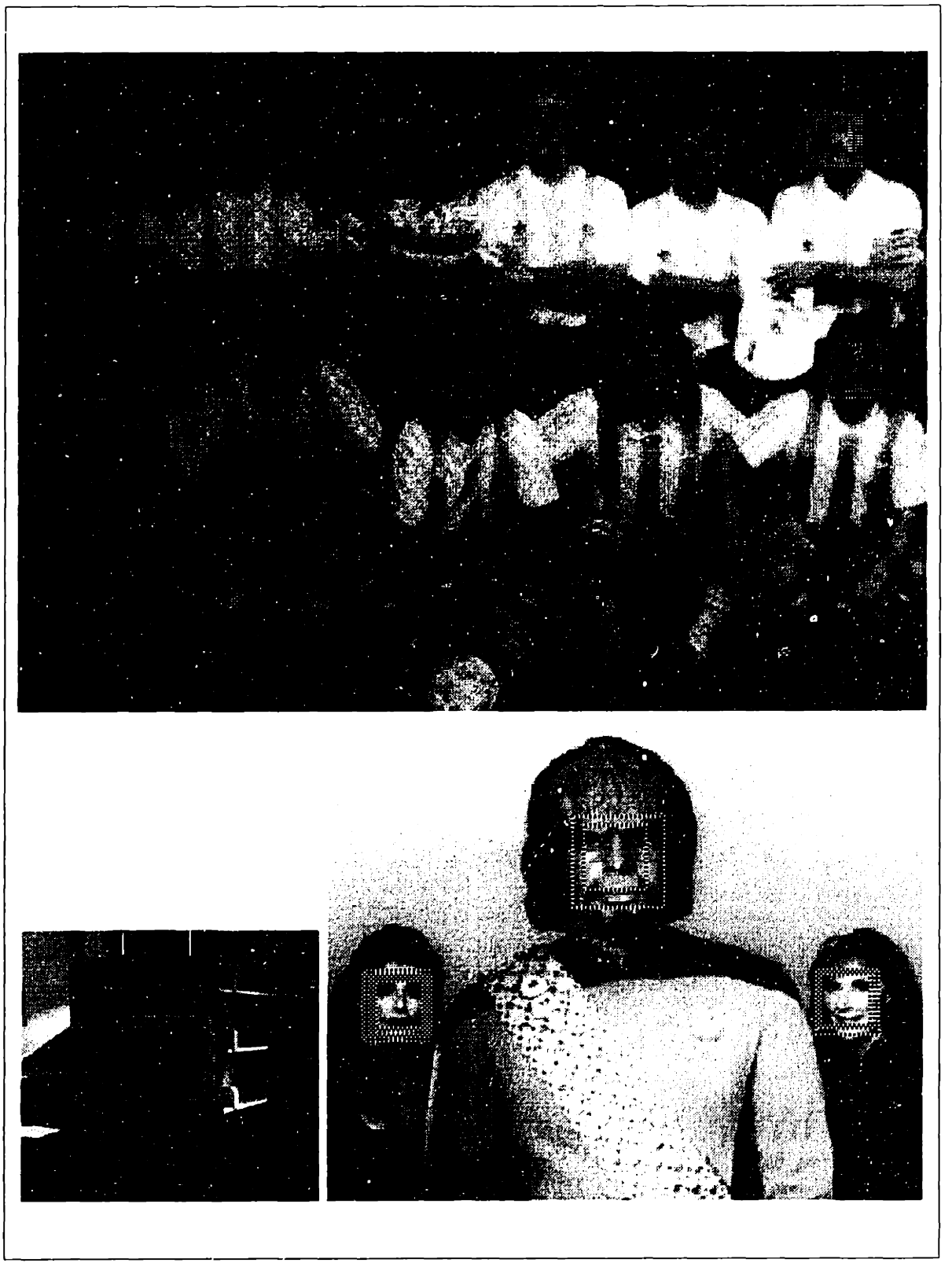


Figure 6-4: Faces

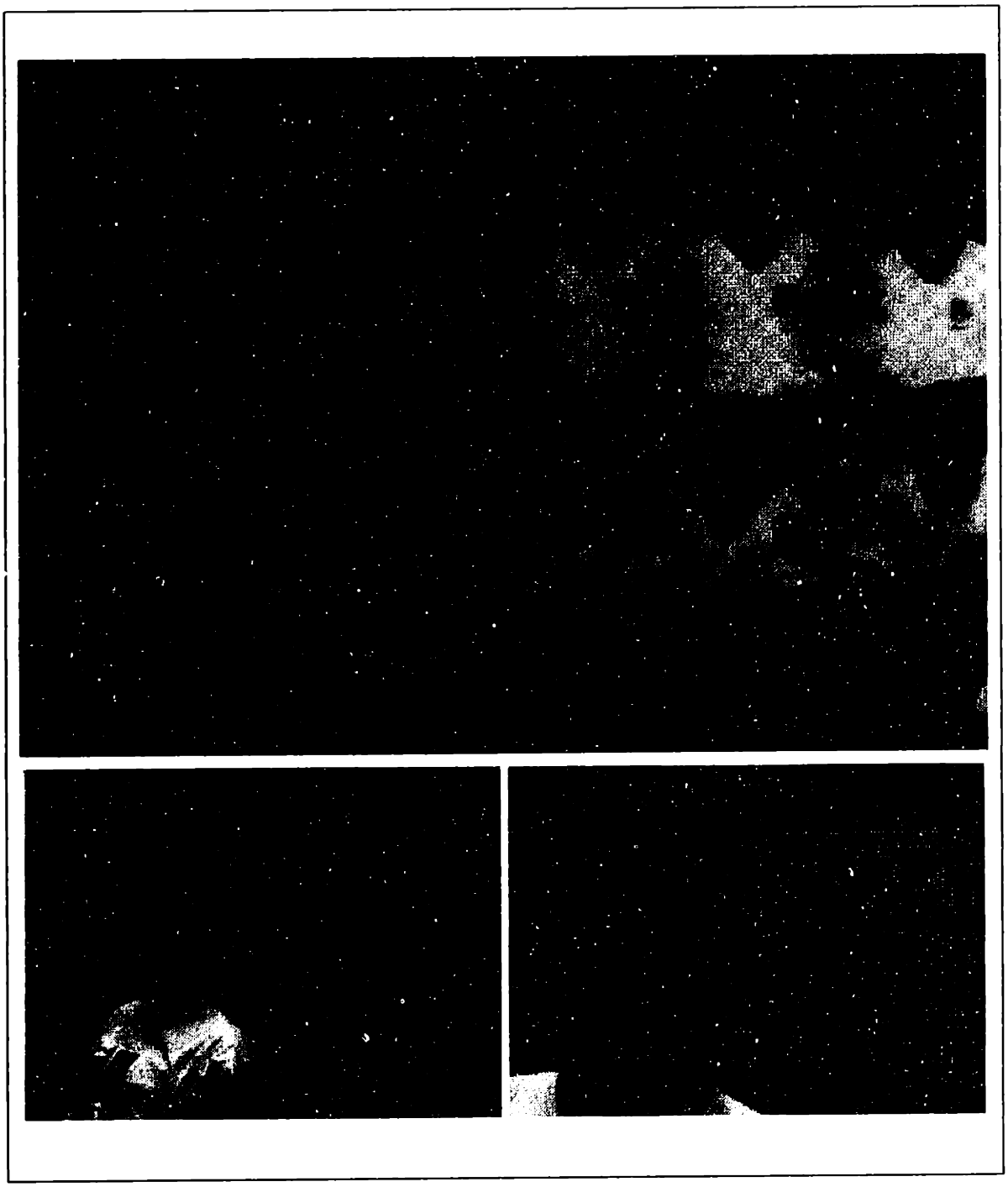


Figure 6-5: Faces

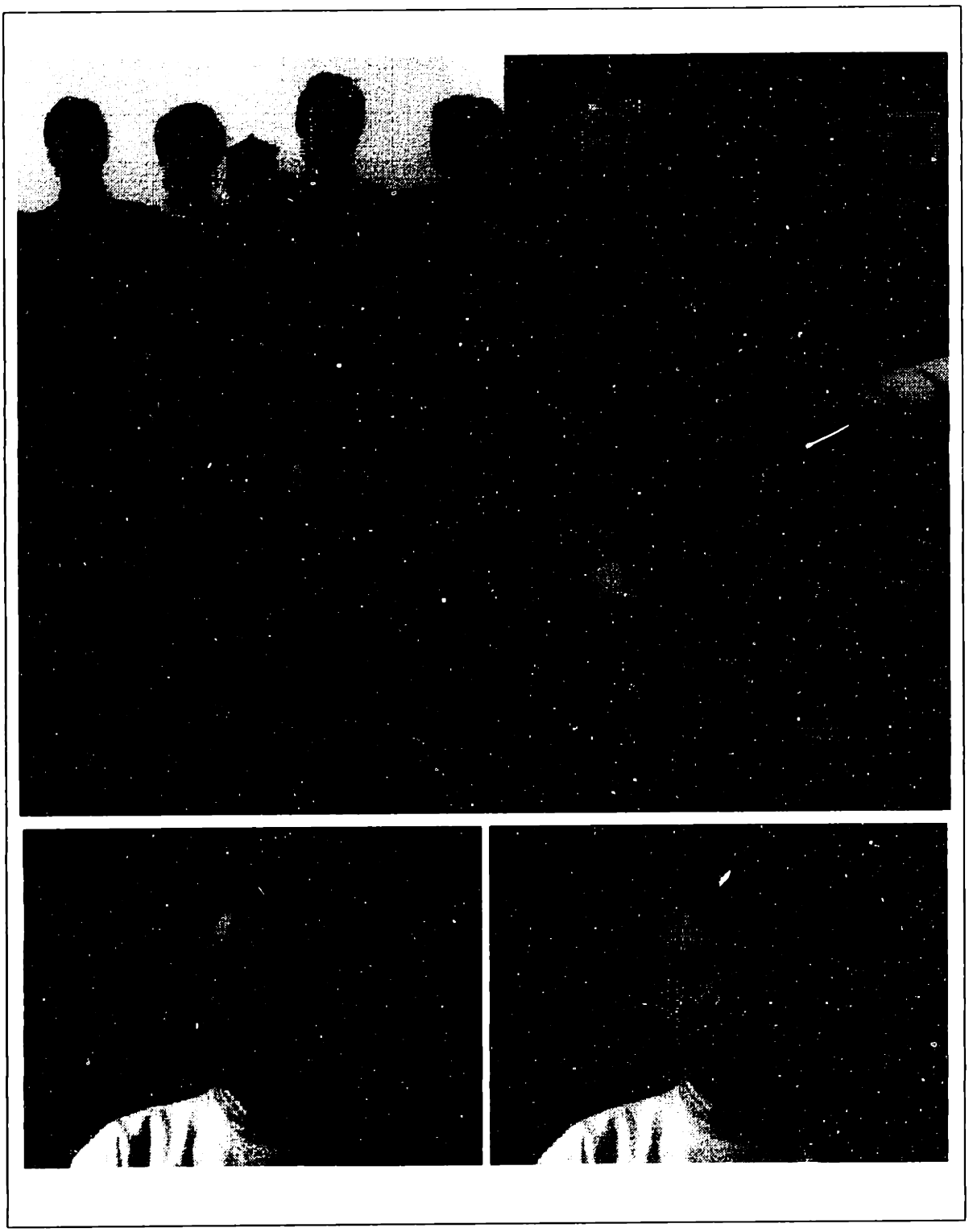


Figure 6-6: Faces

Figure 6-7: Faces

Figure 6-8: Faces

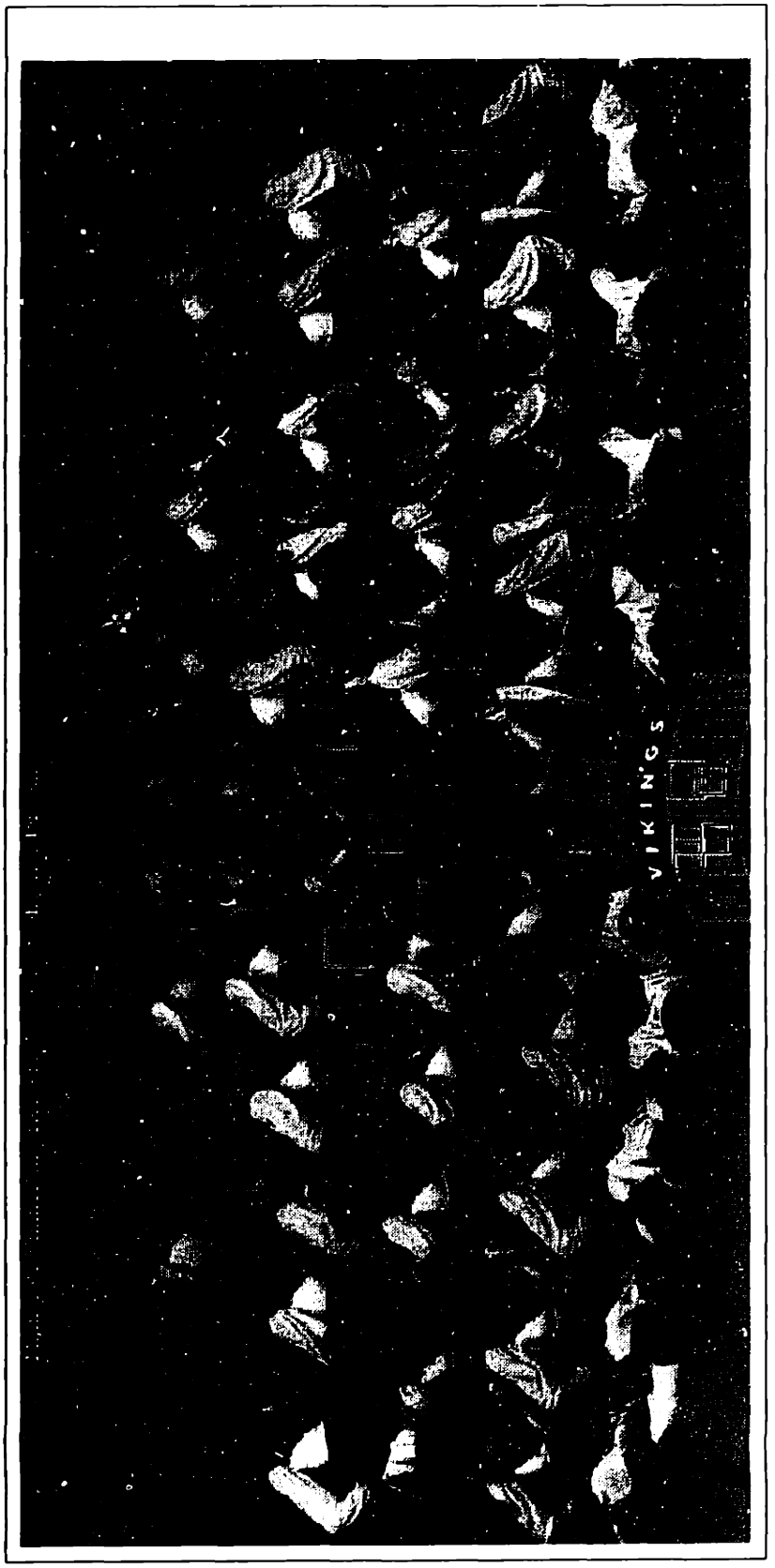


Figure 6-9: Faces

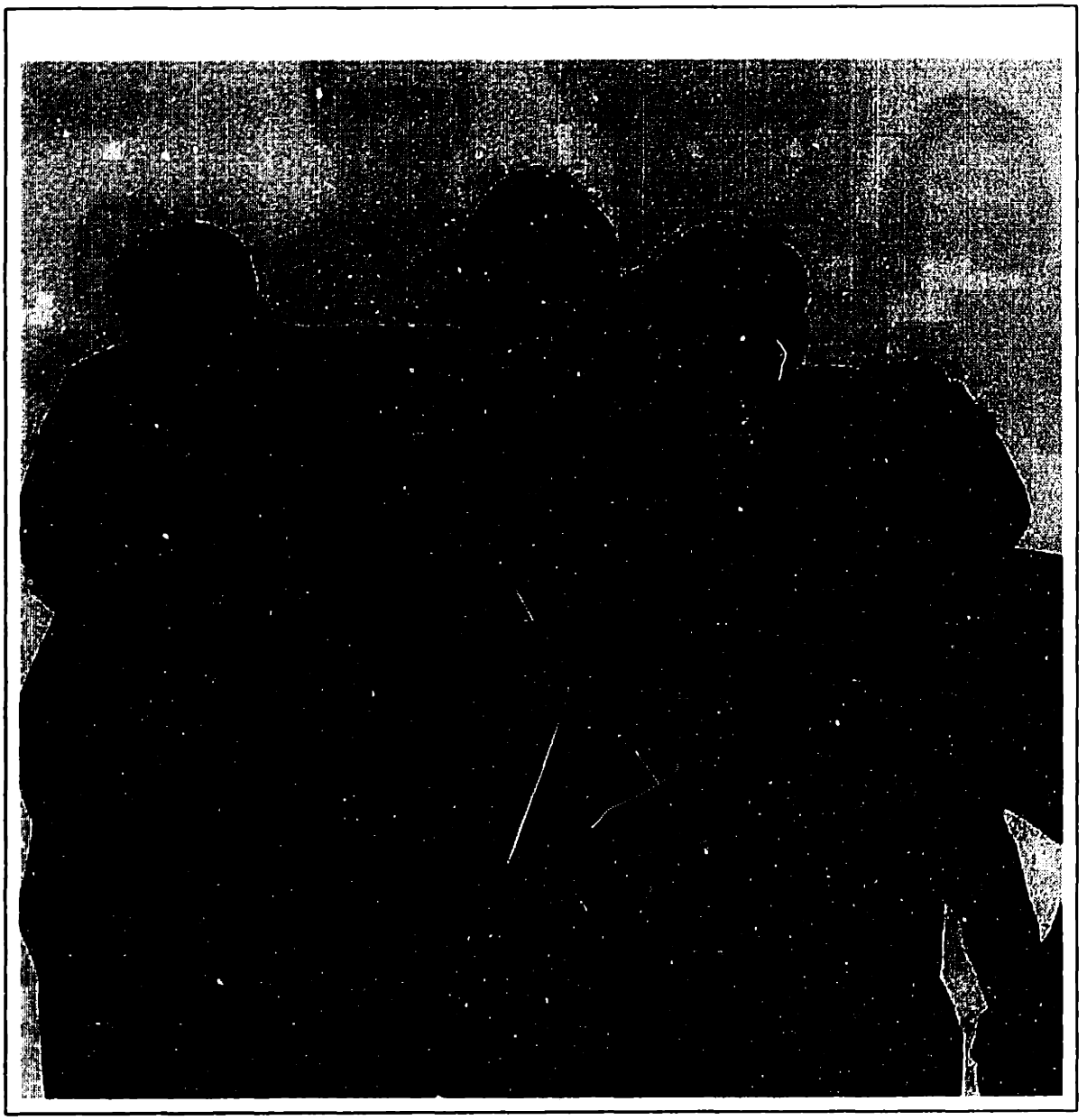


Figure 6-10: Faces

Figure 6-11: Faces

a factor of 2 than Rowley's system, and by a factor of 35 than Sung and Poggio's system, it is not fast enough to be used as a run-time system. In order to build a run-time version of the system, we took the following steps:

1. We ported the C code developed on the SUN environment to a Windows NT Pentium 200 Mhz computer, added a Matrox RGB frame grabber and a Hitachi 3-chip color camera. No special hardware was used to speed-up the computational burden of the system.

2. We collected several color images with faces and extracted from them areas with skin and non-skin pixels. A data set of 6.000 examples was collected. Figure 6-12 presents a plot of 1600 (out of the 6.000 total) examples of skin/non-skin data.

3. We trained a SVM classifier using the skin/non-skin data. The input variables were normalized green and red values, that is, g/(r+g+b) and r/(r+g+b) respectively. Figure 6-13 presents an image captured by the system and its corresponding skin-detection output.

4. A very primitive motion detector based on thresholded frame differencing was coded in order to identify areas of movement and use them as focus of attention. Motion was not a requirement in order to be detected by the system since every so many frames (20 in the current implementation) this step was skipped and the whole image was scanned.

5. A hierarchical system was put together using as a first step the motion detection module. The skin detection system was used as second layer to identify candidate locations of faces. The face/non-face SVM classifier described in the previous two sections was used over the grey level version of the candidate locations.

The whole system achieves rates of 4 to 5 frames per second. Figure 6-14 presents a couple of images captured our PC-based Color Reai Time face detection system.

6.1.5 Future Directions

Future research can be divided into the following categories or topics:

1. **Simplification of the SVM:** As we pointed out in chapter 4 of this thesis, one drawback for using SVMs in some real-life applications is the large number of arithmetic operations that are necessary to classify a new input vector. Usually, this number is proportional to the dimension of the input vector and the number of support vectors obtained. In the case of face detection, for example, this is $\approx$ 283,000 multiplications per pattern!

   Since a closed form solution exists for the case of kernel functions that are 2nd. degree polynomials, we are using a simplified SVM [13] in our current exper-

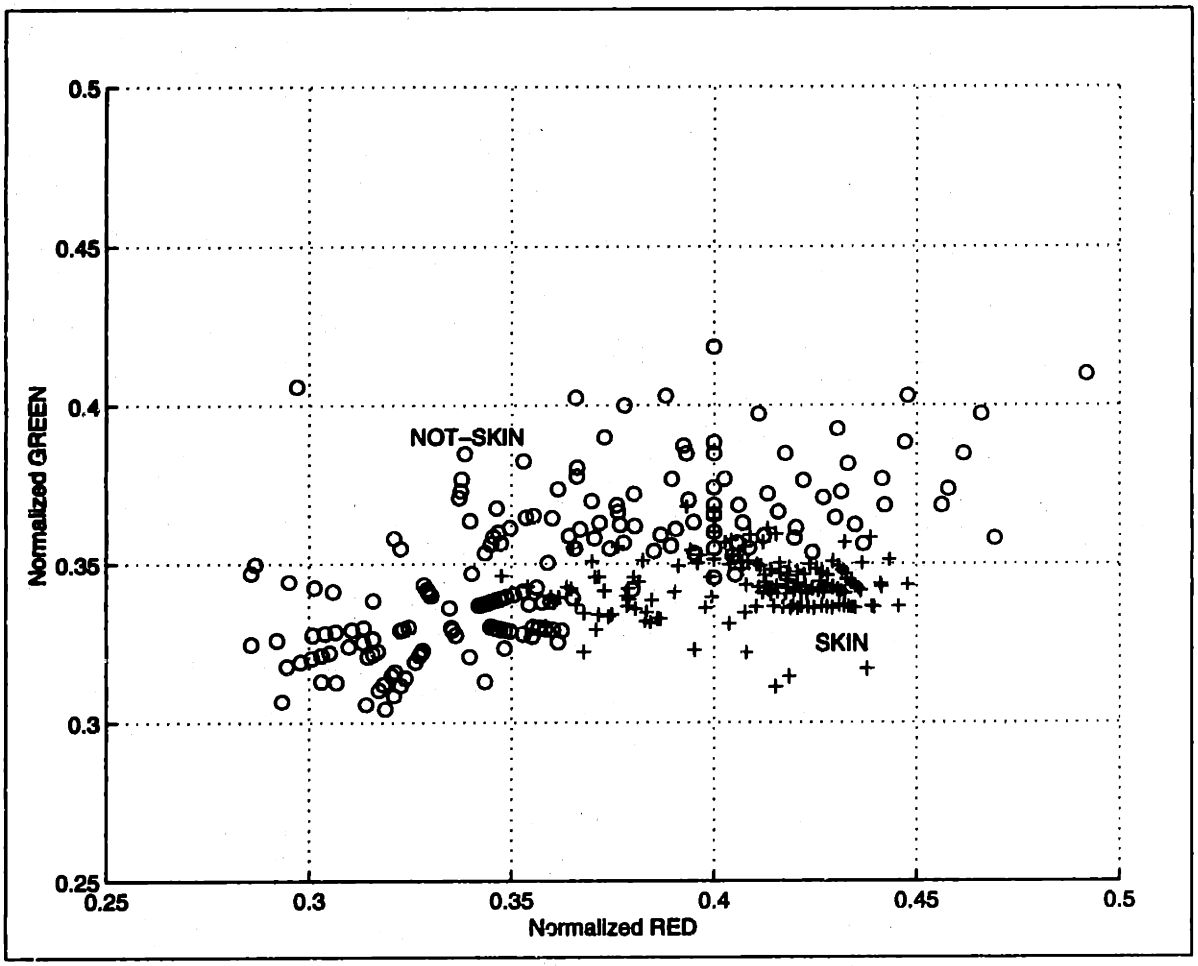


Figure 6-12: 1600 examples of Skin and Non-Skin color samples. The plot is done using normalized green vs. normalized red values.

imental face detection system that gains an acceleration factor of 20, without degrading the quality of the classifications.

In order to use other more sophisticated kernels, techniques like the ones described in chapter 4 could be used. This is considered as another area where further improvement can be achieved.

2. **Use of multiple classifiers:** The use of multiple classifiers offers possibilities that can be faster and/or more accurate. Rowley *et al.* [88] have successfully combined the output from different neural networks by means of different schemes of arbitration in the face detection problem. Sung and Poggio [100][99] use a first classifier that is very fast as a way to quickly discard patterns that are *clearly* non-faces. These two references are just examples of the combination

Figure 6-13: An example of the skin detection module implemented using SVMs.

of different classifiers to produce better systems. The classifiers to be combined do not have to be of the same kind. An interesting type of classifier that could be considered is Discriminant Adaptative Nearest Neighbor due to Hastie *et al.* [46][45].

3. **Focus of attention and quick discarding:**Our current experimental face detection system performs an initial quick-discarding step using a SVM trained to separate *clearly* non-faces from *probable* faces using just 14 averages taken from different areas of the window pattern. This classification can be done about 300 times faster and is currently discarding more than 99% of input patterns. More work can be done in in this area.

4. **Better color representation:** In our real-time system we rely on the information given by the human skin classifier. By using better and more reliable color representations that are, for example, less sensitive to light variations, the system as a whole can also be improved.

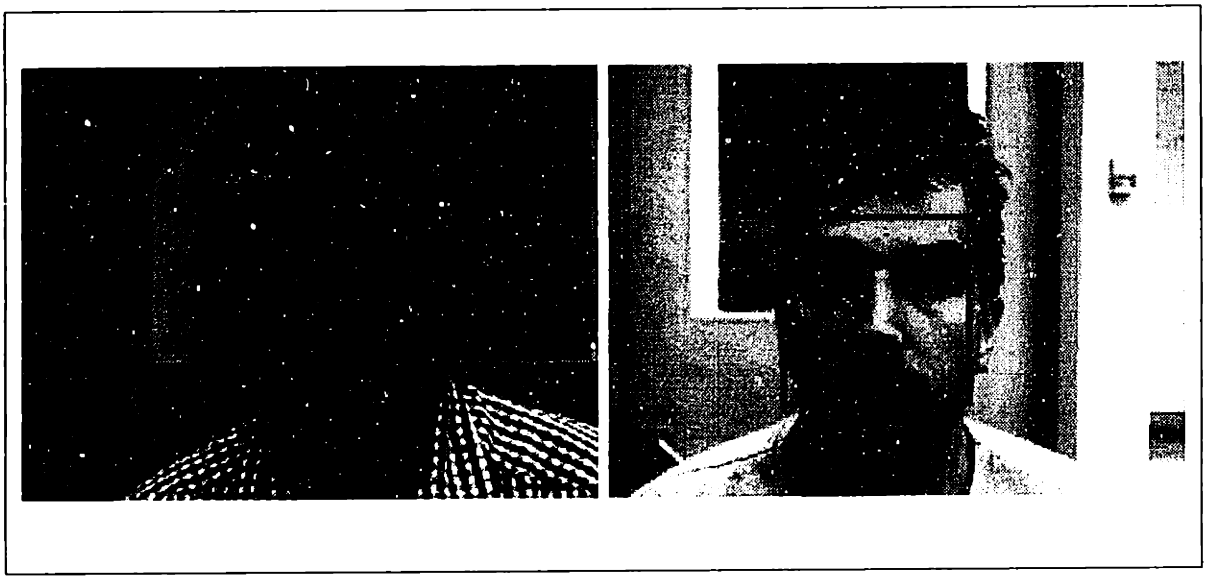


Figure 6-14: Face detection on the PC-based Color Real-Time system.

6.2 People Detection

This section presents a trainable object detection architecture that is applied to detecting people in static images of cluttered scenes. This problem poses several challenges. People are highly non-rigid objects with a high degree of variability in size, shape, color, and texture. Unlike previous approaches, this system learns from examples and does not rely on any a priori (hand-crafted) models or on motion.

The detection technique is based on the novel idea of the *wavelet template* that defines the shape of an object in terms of a subset of the wavelet coefficients of the image. It is invariant to changes in color and texture and can be used to robustly define a rich and complex class of objects such as people. We show how the combination of a powerful classifier like the SVM and the invariant properties and computational efficiency of the wavelet template define an effective tool for object detection.

6.2.1 Introduction and Previous Work

The problem of object detection has seen a high degree of interest over the years. The fundamental problem is how to characterize an object class. In contrast to the case of pattern classification, where we need to decide between a relatively small number of classes, the detection problem requires us to differentiate between the

object class and the rest of the world. As a result, the class description for object detection must have large discriminative power to handle the cluttered scenes it will be presented with. Furthermore, in modeling complicated classes of objects (e.g. faces, pedestrians) the intra-class variability itself is significant and difficult to model. Since it is not known how many instances of the class are presented in the scene, if any, the detection problem cannot easily be solved using methods such as maximum-a-posteriori probability (MAP) or maximum likelihood models. Consequently, the classification of each pattern in the image must be done independently; this makes the decision problem susceptible to miss instances of the class and to false positives.

Figure 6-15: Examples of images of people in the training database. The examples vary in color, texture, view point (either frontal or rear) and background.

There has been a body of work on people detection (Tsukiyama & Shirai, 1985[102], Leung & Yang, 1987[57][56], Rohr, 1993[87], Chen & Shirai, 1994[23]); these approaches are heavily based on motion and hand crafted models. An important aspect of our system is that the model is automatically learned from examples and avoids the use of motion and explicit segmentation.

As we mentioned in section 6.1.1 one of the successful systems in the area of trainable object detection in cluttered scenes is the face detection system of Sung and Poggio [100]. They model face and non-face patterns in a high dimensional space and derive a statistical model for the class of frontal human faces. Similar face detection systems have been developed by others (Vaillant, et al.[103], Rowley, et al.[88], Moghaddam and A. Pentland [63]). We describe our face detection system in section 6.1.

Frontal human faces, despite their variability, share very similar patterns (shape and the spatial layout of facial features) and their color space is very constrained. This is not the case with pedestrians. Figure 6-15 shows several typical images of

people in our database. These images illustrate the difficulties of pedestrian detection; there is significant variability in the patterns and colors within the boundaries of the body. The detection problem is also complicated by the absence of constraints on the image background. Given these problems, direct analysis of pixel characteristics (e.g. intensity, color and texture) is not adequate. This section presents a new approach motivated by an earlier piece of work done by Sinha [94] [95], who derived a new invariant called the *ratio template* and applied it to face detection.

A ratio template encodes the ordinal structure of the brightness distribution on a face. It consists of a set of inequality relationships between the average intensities of a few different face-regions. This design was motivated by the observation that while the absolute intensity values of different regions change dramatically under varying illumination conditions, their mutual ordinal relationships (binarized ratios) remain largely unaffected. Thus, for instance, the forehead is typically brighter than the eye-socket regions for all but the most contrived lighting setups. A small set of such relationships, collectively called a ratio template, provides a powerful constraint for face detection. The emphasis on the use of qualitative relationships also renders the ratio template construct perceptually plausible (the human visual system is poor at judging absolute brightnesses but remarkably adept at making ordinal brightness comparisons). In [95] a scheme for learning such relationships from examples was presented and tested on synthetic images. However, this work left some important issues open. These include a formalization of the template structure in terms of simple primitives, a rigorous learning scheme capable of working with real images, and also the question of applicability to other, possibly more complex, object classes such as pedestrians.

We present an extension of the ratio template, called the *wavelet template*, and address some of these issues in the context of pedestrian detection. The wavelet template consists of a set of regular regions of different scales that correspond to the support of a subset of significant wavelet functions. The relationships between different regions are expressed as constraints on the values of the wavelet coefficients. The wavelet template can compactly express the structural commonality of a class

of objects and is computationally efficient. We show that it is learnable from a set of examples and provides an effective tool for the challenging problem of detecting pedestrians in cluttered scenes. We believe that the learnable wavelet template represents a framework that is extensible to the detection of complex object classes other than pedestrians.

6.2.2 Image Representation and the wavelet template

In this section, we review the Haar wavelet, describe a denser (redundant) transform, and define the wavelet template.

The Haar dictionary

In this section, we survey the properties of wavelets. A more detailed treatment can be found in [61] and other standards references on wavelets.

As motivated by the work on the template ratio, we were looking for an image representation which captures the relationship between average intensities of neighboring regions. This suggests the use of a family of basis functions, such as the Haar wavelets, which encode such relationships along different orientations. The Haar wavelet representation has also been used for image database retrieval, Jacobs *et al.*[51], where the largest wavelet coefficients were used as a measure of similarity between two images. In our work, the wavelet representation is used to capture the structural similarities between various instances of the class. In Figure 6-16, we depict the 3 types of 2-dimensional Haar wavelets. These types include basis functions which capture change in intensity along the horizontal direction, the vertical direction and the diagonals (or corners). Since the wavelets that the standard transform generates have irregular support, we use the non-standard 2-dimensional DWT where, at a given scale, the transform is applied to each dimension sequentially before proceeding to the next scale [98]. The results are Haar wavelets with square support at all scales.

The standard Haar basis is not dense enough for our application. For the 1-dimensional transform, the distance between two neighboring wavelets at level n (with support of size 2^n) is 2^n. For better spatial resolution, we need a set of redundant basis

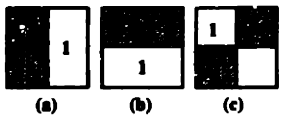


Figure 6-16: The 3 types of 2-dimensional non-standard Haar wavelets; (a) vertical, (b) horizontal, (c) corner.

functions, or an overcomplete *dictionary*, where the distance between the wavelets at scale n is $\frac{1}{4}2^n$. We call this a *quadruple density* dictionary. As one can easily observe, the straightforward approach of shifting the signal and recomputing the DWT will *not* generate the desired dense sampling. However, one can observe that in the standard wavelet transform, after the scaling and wavelet coefficients are convolved with the corresponding filters there is a step of downsampling. If we do not downsample the wavelet coefficients we generate wavelets with *double density*, where wavelets of level n are centered every $\frac{1}{2}2^n$.

To generate the quadruple density dictionary, we compute the scaling coefficients with double density by not downsampling them. The next step is to calculate double density wavelet coefficients on the two sets of scaling coefficients — even and odd — separately. By interleaving the results of the two transforms we get quadruple density wavelet coefficients. For the next scale we keep only the even scaling coefficients of the previous level and repeat the quadruple transform on this set only; the odd scaling coefficients are dropped off. Since only the even coefficients are carried along at all the scales, we avoid an *explosion* in the number of coefficients, yet provide a dense and uniform sampling of the wavelet coefficients at all the scales. As with the regular DWT, the time complexity is $O(n)$ in the number of pixels n. The extension for the 2-dimensional transform is straightforward.

The wavelet template

The ratio template defines a set of constraints on the appearance of an object by defining a set of regions and a set of relationships on their average intensities. The relationships can require, for example, that the ratio of intensities between two specific regions falls within a certain range. We address the issues of learning these relation-

ships, using the template for detection, and its efficient computation by establishing the ratio template in the natural framework of Haar wavelets. Each wavelet coefficient describes the relationship between the average intensities of two neighboring regions. If we compute the transform on the image intensities, the Haar coefficients specify the intensity differences between the regions; computing the transform on the log of the image intensities produces coefficients that represent the log of the ratio of the intensities. Furthermore, the wavelet template can describe regions with different shapes by using combinations of neighboring wavelets with overlapping support and wavelets of different scales. The wavelet template is also computationally efficient since we can compute the transform once for the whole image and look at different sets of coefficients for different spatial locations.

Learning the pedestrian template

As shown in Figure 6-15, it is easy to observe that there are no consistent patterns in the color and texture of pedestrians or their backgrounds in arbitrary cluttered scenes in unconstrained environments. This lack of clearly discernible interior features is circumvented by relying on (1) differences in the intensity between pedestrian bodies and their backgrounds and (2) consistencies within regions inside the body boundaries. We interpret the wavelet coefficients as either indicating an almost uniform area, i.e. *no-change*, if their absolute value is relatively small, or as indicating *strong change* if their absolute value is relatively large. The wavelet template we seek to identify will consist solely of wavelet coefficients (either vertical, horizontal or corner) whose types (*change/no-change*) are both clearly identified and *consistent* along the ensemble of pedestrian images; these comprise the *important* coefficients.

The basic analysis to identify the template consists of two steps: first, we normalize the wavelet coefficients relative to the rest of the coefficients in the patterns; second, we analyze the averages of the normalized coefficients along the ensemble. We have collected a set of 564 color images of people (Figure 6-15) for use in the template learning. All the images are scaled and clipped to the dimensions 128×64 such that the people are centered and approximately the same size (the distance from the shoul-

ders to feet is about 80 pixels). In our analysis, we restrict ourselves to the wavelets at scales of 32×32 pixels (one array of 15×5 coefficients for each wavelet class) and 16×16 pixels (29×13 for each class). For each color channel (RGB) of every image, we compute the quadruple dense Haar transform and take the coefficient value to be the largest absolute value among the three channels. The normalization step computes the average of each coefficient's class ($\{vertical, horizontal, corner\} \times \{16, 32\}$) over all the pedestrian patterns and divides every coefficient by its corresponding class average. We calculate the averages separately for each class since the power distribution between the different classes may vary.

To begin specifying the template, we calculate the average of each normalized coefficient over the set of pedestrians. A base set of 597 color images of natural scenes of size 128×64 that do not contain people were gathered to compare with the pedestrian patterns and are processed as above. Tables 6.2(a) and 6.2(b) show the average coefficient values for the set of vertical Haar coefficients of scale 32×32 for both the non-pedestrian and pedestrian classes. Table 6.2(a) shows that the process of averaging the coefficients within the pattern and then in the ensemble does not create spurious patterns; the average values of these non-pedestrian coefficients are near 1 since these are random images that do not share any common pattern. The pedestrian averages, on the other hand, show a clear pattern, with strong response (values over 1.5) in the coefficients corresponding to the sides of the body and weak response (values less than 0.5) in the coefficients along the center of the body.

We use a gray level coding scheme to visualize the patterns in the different classes of coefficients the values of the coefficients and display them in the proper spatial layout. Coefficients close to 1 are gray, stronger coefficients are darker, and weaker coefficients are lighter. Figures 6-17(a)-(d) show the color coding for the arrays of coarse scale coefficients (32×32) and Figures 6-17(e)-(g) show the arrays of coefficients of the finer scale, (16×16).

Figure 6-17(a) shows the vertical coefficients of random images; as expected, this figure is uniformly gray. The corresponding images for the horizontal and corner coefficients, not shown here, are similar. In contrast, the coefficients of the people,

(a) (b) (c) (d) (e) (f) (g)

Figure 6-17: Ensemble average values of the wavelet coefficients coded using gray level. Coefficients whose values are above the template average are darker, those below the average are lighter. (a) vertical coefficients of random scenes. (b)-(d) vertical, horizontal and corner coefficients of scale 32×32 of images of people. (e)-(g) vertical, horizontal and corner coefficients of scale 16×16 of images of people.

Figures 6-17(b)-(d), show clear patterns, with the different classes of wavelet coefficients being tuned to different types of structural information. The vertical wavelets, Figure 6-17(b), capture the sides of the pedestrians. The horizontal wavelets, Figure 6-17(c), respond to the line from shoulder to shoulder and to a weaker belt line. The corner wavelets, Figure 6-17(d), are better tuned to corners, for example, the shoulders, hands and feet. The wavelets of finer scale in Figures 6-17(e)-(g) provide better spatial resolution of the body's overall shape and smaller scale details such as the head and extremities appear clearer. Two similar statistical analyses using a) the wavelets of the log of the intensities and b) the sigmoid function as a *soft threshold* on the normalized coefficients yields results that are similar to the intensity differencing wavelets. It is intriguing that a basic measure like the ensemble average provides clear identification of the template as shown in Figure 6-17.

The template derived from learning uses a set of 29 coefficients that are consistent along the ensemble either as indicators of *change* or *no-change*. There are 6 vertical and 1 horizontal coefficients at the scale of 32×32 and 14 vertical and 8 horizontal at the scale of 16×16. These coefficients serve as the feature vector for the classification problem.

6.2.3 The detection system

Once we have identified the important basis functions, we can use various classification techniques to learn the relationships between the wavelet coefficients that define the pedestrian class. In this section, we present the overall architecture of the detection system and the training process. We conclude with experimental results of the detection system.

System architecture

The system detects people in arbitrary positions in the image and in different scales. To accomplish this task, the system is trained to detect a pedestrian centered in a 128×64 pixel window. Once the training stage is completed, the system is able to detect pedestrians at arbitrary positions by shifting the 128×64 window, thereby scanning all possible locations in the image. This is combined with iteratively resizing the image to achieve multi-scale detection; in our experiments, we scale the image from 0.2 to 1.5 times its original size, at increments of 0.1. At any given scale, instead of recomputing the wavelet coefficients for every window in the image, we compute the transform for the whole image and do the shifting in the coefficient space. A shift of one coefficient in the finer scale corresponds to a shift of 4 pixels in the window and a shift in the coarse scale corresponds to a shift of 8 pixels. Since most of the coefficients in the wavelet template are at the finer scale (the coarse scale coefficients hardly change with a shift of 4 pixels), we achieve an effective spatial resolution of 4 pixels by working in the wavelet coefficient space.

System training

To train our system, we use a database of frontal and rear images of people from outdoor and indoor scenes. The initial non-people in the training database are patterns from natural scenes not containing people. The combined set of positive and negative examples form the initial training database for the classifier. A key issue with the training of detection systems is that, while the examples of the target class, in this

case pedestrians, are well defined, there are no typical examples of non-pedestrians. The main idea in overcoming this problem of defining this extremely large negative class is the use of *bootstrapping* training [100]. After the initial training, we run the system over arbitrary images that do not contain any people. Any detections are clearly identified as false positives and are added to the database of negative examples and the classifier is then retrained with this larger set of data. These iterations of the bootstrapping procedure allows the classifier to construct an incremental refinement of the non-pedestrian class until satisfactory performance is achieved. This bootstrapping technique is illustrated in Figure 6-18.

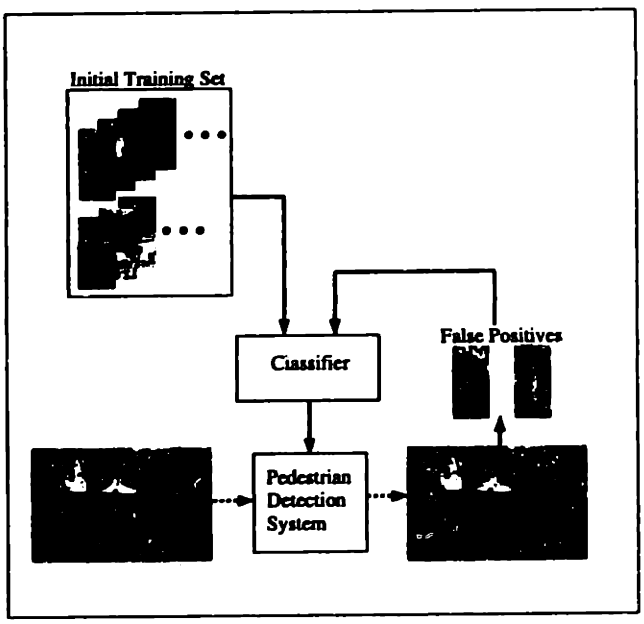


Figure 6-18: Incremental bootstrapping to improve the system performance.

Classification schemes

In Section 6.2.2 we described the identification of the significant coefficients that characterize the pedestrian class. These coefficients are used as the feature vector for various classification methods.

Basic template matching

The simplest classification scheme is to use a basic template matching measure. As in Section 6.2.2, the normalized template coefficients are divided into two categories: coefficients above 1 (indicating strong change) and below 1 (weak change). For every

novel window, the wavelet coefficients are compared to the pedestrian template. The matching value is the ratio of the coefficients in agreement. A similar approach was used in [94] for face detection with good results. While this basic template matching scheme is very simple — better classification techniques can be applied — it is interesting to see how well it will perform on this more complex task.

Support vector machines

Instead of the simple template matching paradigm we can use a more sophisticated classifier which will learn the relationship between the coefficients from given sets of positive and negative examples. The classifier can learn more refined relationships than the simple template matching scheme and therefore can provide more accurate detection. For our classification problem, we find that using a polynomial of degree two as the kernel provides good results.

It should be observed, that from the view point of the classification task, we could use the whole set of coefficients as a feature vector. However, using all the wavelet functions that describe a window of 128×64 pixels, over a few thousands, would yield vectors of very high dimensionality, as we mentioned earlier. The training of a classifier with such a high dimensionality would in turn require too large an example set. The template learning stage of Section 6.2.2 serves to select the basis functions relevant for this task and to reduce their number considerably (to a very reasonable 29).

6.2.4 Experimental results

To evaluate the system performance, we start with a database of 564 positive examples and 597 negative examples. The system then undergoes the bootstrapping cycle detailed in Section 6.2.3. For this system, the support vector system goes through three bootstrapping steps, ending up with a total of 4597 negative examples. For the template matching version a threshold of 0.7 (70% matching) was empirically found to yield good results.

Out-of-sample performance is evaluated over a test set consisting of 72 images for both the template matching scheme and the support vector classifier. The test

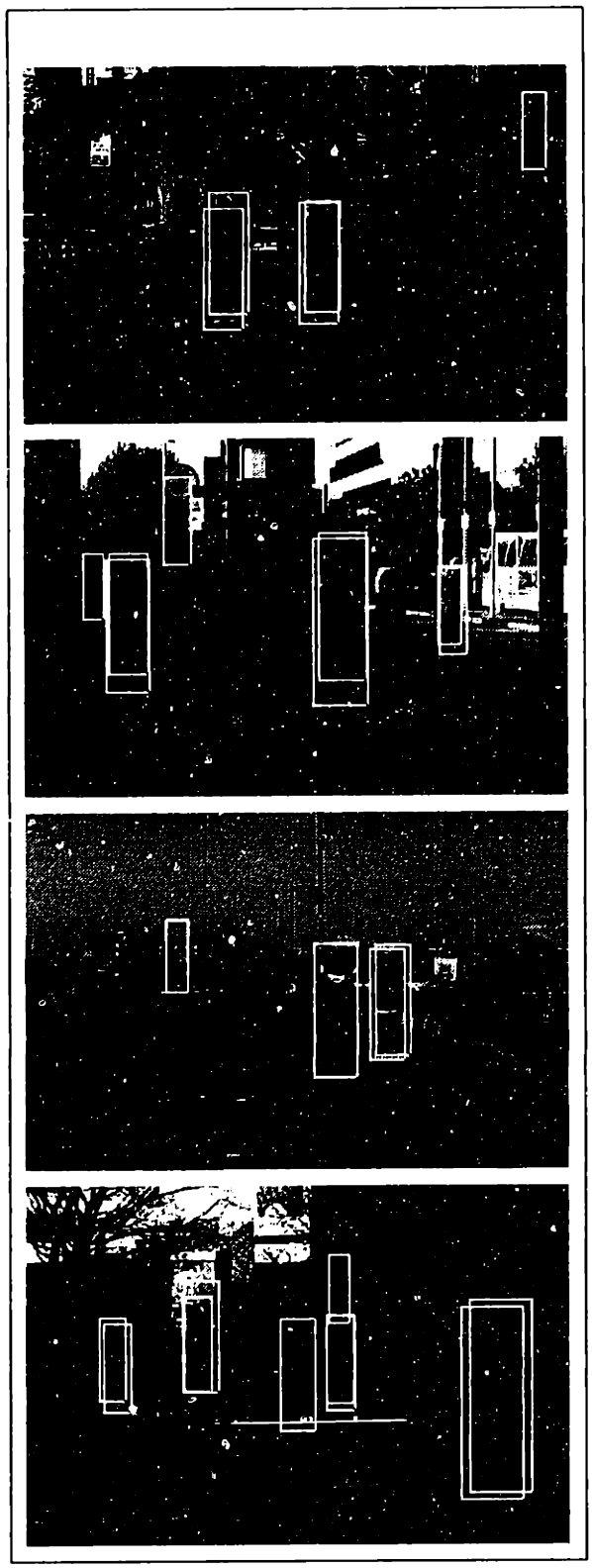


Figure 6-19: Results from the pedestrian detection system. These are typical images of relatively complex scenes that are used to test the system.

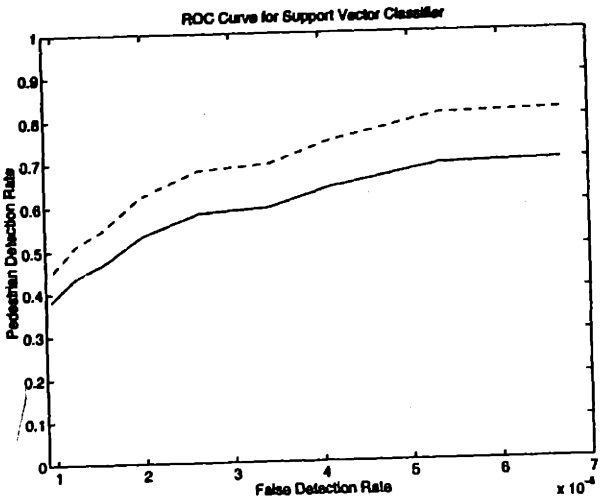


Figure 6-20: ROC curves for the support vector detection system; the bottom curve is over the entire test set, the top curve is over the *high quality* set.

images contain a total of 165 pedestrians in frontal or near-frontal poses; 24 of these pedestrians are only partially observable (e.g. with body regions that are indistinguishable from the background). Since the system was not trained with partially observable pedestrians, we would not even expect a perfectly trained system (with the current template) to detect these instances. To give a fair account of the system, we present statistics for both the total set and the set of 141 *high quality* pedestrian images. Results of the tests are presented in Table 6.3 for representative systems using template matching and support vectors.

The template matching system has a pedestrian detection rate of 52.7%, with a false positive rate of 1 for every 5,000 windows examined. The success of such a straightforward template matching measure suggests that the template learning scheme extracts non-trivial structural regularity within the pedestrian class.

For the more sophisticated system with the support vector classifier, we perform a more thorough analysis. In general, the performance of any detection system exhibits a tradeoff between the rate of detection and the rate of false positives. Performance drops as we impose more stringent restrictions on the rate of false positives. To capture this tradeoff, we vary the sensitivity of the system by thresholding the output and evaluate the ROC curve, given in Figure 6-20. From the curve we can see, for example, that if we have a tolerance of one false positive for every 15,000 windows

examined, we can achieve a detection rate of 69.6%, and as high as 81.6% on the *high quality* set. As we expect, the support vector classifier with the bootstrapping training performs better than the *naive* template matching scheme.

In Figure 6-19 we show typical images that are used to test the system. These are very cluttered scenes crowded with complex patterns. Considering the complexity of these scenes and the difficulties of pedestrian detection in natural outdoor scenes, we consider the above detection rate to be high. It is interesting to observe that most of the false positives are patterns with overall proportions similar to the human body. We believe that additional training and refinement of the current system will reduce the false detection rate further.

The system is currently trained only on frontal and rear views of pedestrians. Training the classifier to handle side views can be accomplished in an identical manner and can be considered as a natural extension to the current system.

6.2.5 Remarks

In this section, we have introduced the idea of a wavelet template and demonstrated how it can be learned and used for pedestrian detection in a cluttered scene. The wavelet template defines an object as a set of regions and relationships among them. A key idea is to use a wavelet basis to represent the template, yielding not only a computationally efficient algorithm but also an effective learning scheme.

The success of the wavelet template for pedestrian detection comes from its ability to capture high-level knowledge about the object class (structural information expressed as a set of constraints on the wavelet coefficients) and incorporate it into the low-level process of interpreting image intensities. Attempts to directly apply low-level techniques such as edge detection and region segmentation are likely to fail in the type of images we analyze since these methods are not robust, are sensitive to spurious details, and give ambiguous results. Using the wavelet template, only significant information that characterizes the object class — as obtained in the learning phase — is evaluated and used.

In summary, in our approach a pedestrian template is learned from examples

and then used for classification. It is important to realize that this is not the only interpretation of our approach, though it is the one originally suggested by [94] and is the one emphasized throughout this section. An alternative, and more general, point of view considers the step of learning the template as a dimensionality reduction stage. Using all the wavelet functions that describe a window of 128×64 pixels would yield vectors of very high dimensionality, as we mentioned earlier. The training of a classifier with such a high dimensionality would in turn require too large a data set. The template learning stage of Section 6.2.2 serves to select the basis functions relevant for this task and to reduce their number considerably (to a very reasonable 29). A classifier – such as the support vector machine – can then be trained on the data set. From this point of view, learning the pedestrian detection task consists of two learning steps: 1) dimensionality reduction, that is, task-dependent basis selection and 2) training the classifier. In this interpretation, a template in the strict sense of the word is neither learned nor used.

In any case, it seems that the approach we have described, combined with the related strategy used previously to learn face detection, may well generalize to several other object detection tasks.

1.18	1.14	1.16	1.09	1.11
1.13	1.06	1.11	1.06	1.07
1.07	1.01	1.05	1.03	1.05
1.07	0.97	1.00	1.00	1.05
1.06	0.99	0.98	0.98	1.04
1.03	0.98	0.95	0.94	1.01
0.98	0.97	0.96	0.91	0.98
0.98	0.96	0.98	0.94	0.99
1.01	0.94	0.98	0.96	1.01
1.01	0.95	0.95	0.96	1.00
0.99	0.95	0.92	0.93	0.98
1.00	0.94	0.91	0.92	0.96
1.00	0.92	0.93	0.92	0.96

(a)

0.62	0.74	0.60	0.75	0.66
0.76	0.92	0.54	0.88	0.81
1.07	1.11	0.52	1.04	1.15
1.38	1.17	0.48	1.08	1.47
1.65	1.27	0.48	1.15	1.71
1.62	1.24	0.48	1.11	1.63
1.44	1.27	0.46	1.20	1.44
1.27	1.38	0.46	1.34	1.27
1.18	1.51	0.46	1.48	1.18
1.09	1.54	0.45	1.52	1.08
0.94	1.38	0.42	1.39	0.93
0.74	1.08	0.36	1.11	0.72
0.52	0.74	0.29	0.77	0.50

(b)

Table 6.2: Normalized vertical coefficients of scale 32×32 of images with (a) random natural scenes (without people), (b) pedestrians.

	Detection Rate	*False Positive Rate (per window)*
Template Matching	52.7% (61.7%)	1:5,000
SVM	69.7% (81.6%)	1:15,000

Table 6.3: Performance of the pedestrian detection system; values in parentheses are for the set of "high quality" pedestrian images.

6.3 Nonlinear Prediction of Chaotic Time Series

We tested SVRM on the same data base of chaotic time series that was used in by Casdagli in [21] to compare the performances of different approximation techniques, including polynomial and rational approximation, local polynomial techniques, Radial Basis Functions, and Neural Networks. The SVM performs better than the approaches presented in [21]. We also study, for a particular time series, the variability in performance with respect to the few free parameters of SVM.

The section is organized as follows: In section 6.3.1 we formulate the problem of time series prediction and see how it is equivalent to a regression problem. In section 6.3.2 we introduce the chaotic time series used to benchmark previous regression methods in [21]. Section 6.3.3 contains the experimental results and the comparison with the techniques presented in [21]. Section 6.3.4 focuses on a particular series, the Mackey-Glass, and examines the relation between parameters of the SVM and generalization error.

6.3.1 Time Series Prediction and Dynamical Systems

For the purpose of this section a *dynamical system* is a smooth map $F : R \times \mathcal{S} \to \mathcal{S}$ where $\mathcal{S}$ is an open set of an Euclidean space. Writing $F(t, \mathbf{x}) = F_t(\mathbf{x})$, the map F has to satisfy the following conditions:

1. $F_0(\mathbf{x}) = \mathbf{x}$;
2. $F_t(F_s(\mathbf{x})) = F_{s+t}(\mathbf{x}) \quad \forall s, t \in R$

For any given initial condition $\mathbf{x}_0 = F_0(\mathbf{x})$ a dynamical system defines a trajectory $\mathbf{x}(t) = F_t(\mathbf{x}_0)$ in the set $\mathcal{S}$. The *direct* problem in dynamical systems consists in analyzing the behavior and the properties of the trajectories $\mathbf{x}(t)$ for different initial conditions $\mathbf{x}_0$. We are interested in a problem similar to the inverse of the problem stated above. We are given a finite portion of a time series $x(t)$, where x is a component of a vector $\mathbf{x}$ that represents a variable evolving according to some unknown dynamical system. We assume that the trajectory $\mathbf{x}(t)$ lies on a manifold with fractal

dimension D (a "strange attractor"). Our goal is to be able to predict the future behavior of the time series $x(t)$. Remarkably, this can be done, at least in principle, without knowledge of the other components of the vector $\mathbf{x}(t)$. In fact, Takens embedding theorem [101] ensures that, under certain conditions, for almost all τ and for some $m \leq 2D + 1$ there is a smooth map $f : \mathcal{R}^m \rightarrow \mathcal{R}$ such that:

$$x(n\tau) = f\left(x((n-1)\tau),\ x((n-2)\tau), \ldots, x((n-m)\tau)\right) \tag{6.1}$$

The value of m used is called the *embedding dimension* and the smallest value for which (6.1) is true is called the minimum embedding dimension, m^*. Therefore, if the map f were known, the value of x at time $n\tau$ is uniquely determined by its m values in the past. For simplicity of notation we define the m-dimensional vector

$$\tilde{\mathbf{x}}_{n-1} \equiv \left(x((n-1)\tau),\ x((n-2)\tau), \ldots, x((n-m)\tau)\right)$$

in such a way that eq. (6.1) can be written simply as $x(n\tau) = f(\tilde{\mathbf{x}}_{n-1})$. If N observations $\{x(n\tau)\}_{n=1}^{N}$ of the time series $x(t)$ are known, then one also knows $N-m$ values of the function f, and the problem of learning the dynamical system becomes equivalent to the problem of estimating the unknown function f from a set of $N-m$ sparse data points in $\mathcal{R}^m$. Many regression techniques can be used to solve problems of this type. In this section we concentrate on using the Support Vector Regression Machine (SVRM).

6.3.2 Benchmark Time Series

We tested the SVM regression technique on the same set of chaotic time series that Casdagli used in [21] to test and compare several approximation techniques.

The Mackey-Glass time series

We considered two time series generated by the Mackey-Glass delay-differential equation [60]:

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t-\Delta)}{1+x(t-\Delta)^{10}}, \tag{6.2}$$

with parameters $\Delta = 17, 30$ and embedding dimensions $m = 4, 6$ respectively. We denote these two time-series by MG_{17} and MG_{30}. In order to be consistent with [21] the initial condition for the above equation was $x(t) = 0.9$ for $0 \leq t \leq \Delta$, and the sampling rate $\tau = 6$. The series were generated by numerical integration using a fourth order Runge-Kutta method.

The Ikeda map

The Ikeda map [50] is a two dimensional time series which is generated iterating the following map:

$$f(x_1, x_2) = (1 + \mu(x_1 \cos\omega - x_2 \sin\omega), \mu(x_1 \sin\omega + x_2 \cos\omega)), \tag{6.3}$$

where $\omega = 0.4 - 6.0/(1 + x_1^2 + x_2^2)$. In [21] Casdagli considered both this time series, that we will denote by $Ikeda_1$, and the one generated by the fourth iterate of this map, which has a more complicated dynamic, and that will be denoted by $Ikeda_4$.

The Lorenz time series

We also considered the time series associated to the variable x of the Lorenz differential equation [58]:

$$\dot{x} = \sigma(y - x), \quad \dot{y} = rx - y - xz, \quad \dot{z} = xy - bz \tag{6.4}$$

where $\sigma = 10$, $b = \frac{8}{3}$, and $r = 28$. We considered two different sampling rates, $\tau = 0.05$ and $\tau = 0.20$, generating the two time series $Lorenz_{0.05}$ and $Lorenz_{0.20}$. The series were generated by numerical integration using a fourth order Runge-Kutta method.

6.3.3 Comparison with Other Techniques

In this section we report the results of the SVM on the time series presented above, and compare them with the results reported in [21] using different approximation techniques (polynomial, rational, local polynomial, Radial Basis Functions with multi-quadrics as basis function, and Neural Networks). In all cases, a time series $\{x(n\tau)\}_{n=1}^{N+M}$ was generated as follows: the first N points were used for training and the remaining M points were used for testing. In all cases N was set to 500, except for the $Ikeda_4$, for which $N = 100$, while M was always set to 1000. The data sets we used were the same that were used in [21]. Following [21], denoting by $\tilde{f}_N$ the predictor built using N data points, the following quantity was used as a measure of the generalization error of $\tilde{f}_N$:

$$\sigma^2(\tilde{f}_N) = \frac{1}{M} \sum_{n=N+1}^{M} \frac{(x(n\tau) - \tilde{f}_N(\tilde{\mathbf{x}}_{n-1}))^2}{\text{Var}} \tag{6.5}$$

where Var is the variance of the time series.

For each series we choose the kernel, K, and parameters of the kernel that gave us the smallest generalization error. This is consistent with the strategy adopted in [21]. The results are reported in table (6.4). The last column of the table contains the results of our experiments, while the rest of the table is from [21] with parameters and kernels set as in the remaining part of this section:

- **Mackey-Glass time-series:** the kernel K was chosen to have the following form:

$$K(\mathbf{x}, \mathbf{x}_i) = \sum_{d=1}^{m} \frac{\sin((\nu + 1/2)(x^{(d)} - x_i^{(d)}))}{\sin(0.5(x^{(d)} - x_i^{(d)}))} \tag{6.6}$$

  where $x^{(d)}$ is the d-th component of the m-dimensional vector $\mathbf{x}$, and a ν is an integer. This kernel generates an approximating function that is an additive trigonometric polynomial of degree ν, and corresponds to features ϕ_i that are

trigonometric monomials up to degree ν. We tried various values for ϵ and C. The embedding dimension for the series MG_{17} and MG_{30} were $m = 4$ and $m = 6$ in accordance with the work by Casdagli, and we used $\nu = 200$ and $\epsilon = 10^{-3}$.

- **Lorenz time-series:** For the $\text{Lorenz}_{0.05}$ and $\text{Lorenz}_{0.20}$ series the polynomial kernels of order 6 and 10 were used. The embedding dimensions used were 6 and 10 respectively. The value of ϵ used was 10^{-3}.

- **Ikeda map:** a B-spline of order 3 was used as the kernel, and the value of ϵ was 10^{-3}.

	Poly	Rational	$\text{Loc}^{d=1}$	$\text{Loc}^{d=2}$	RBF	N.Net	SVM
MG_{17}	$-1.95^{(7)}$	$-1.14^{(2)}$	-1.48	-1.89	-1.97	-2.00	-2.36 (258)
MG_{30}	$-1.40^{(4)}$	$-1.33^{(2)}$	-1.24	-1.42	-1.60	-1.5	-1.87 (341)
$Ikeda_1$	$-5.57^{(12)}$	$-8.01^{(8)}$	-1.71	-2.34	-2.95	-	-6.21 (374)
$Ikeda_4$	$-1.05^{(14)}$	$-1.39^{(14)}$	-1.26	-1.60	-2.10	-	-2.31 (427)
$Lor_{0.05}$	$-4.62^{(6)}$	$-4.30^{(3)}$	-2.00	-3.48	-3.54	-	-4.76 (389)
$Lor_{0.20}$	$-1.05^{(5)}$	$-1.39^{(6)}$	-1.26	-1.60	-2.10	-	-2.21 (448)

Table 6.4: Estimated values of $\log_{10}\sigma(\tilde{f}_n)$ for the SVM algorithm and for various regression algorithms, as reported in [21]. The degrees used for the best rational and polynomial regressors are in superscripts beside the estimates. $\text{Loc}^{d=1}$ and $\text{Loc}^{d=2}$ refer to local approximation with polynomials of degree 1 and 2 respectively. The numbers in parenthesis near the SVM estimates are the number of support vectors obtained by the algorithm. The Neural Networks results which are missing were also missing in [21].

6.3.4 Sensitivity to Parameters and Embedding Dimension

In this section we report our observations on how the generalization error and the number of support vectors vary with respect to the free parameters of the SVM and to the choice of the embedding dimension. The parameters we analyze are therefore C, ϵ, the dimensionality of the feature space D, and the embedding dimension m. All of these results are for the MG_{17} series.

Figure 6-21a demonstrates that C has little effect on the generalization error (the plot spans over 7 orders of magnitude). The parameter C has also little effect on the number of support vector, as shown in figure 6-21b, which remains almost constant in the range $10^{-2} - 10^{2}$. The results were similar for kernels with low ($D = 2$), high ($D = 802$) and infinite dimensionality of the feature spaces.

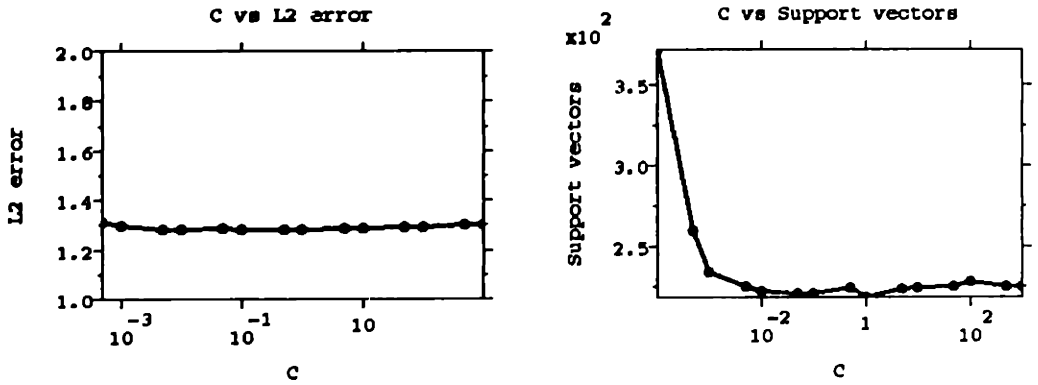


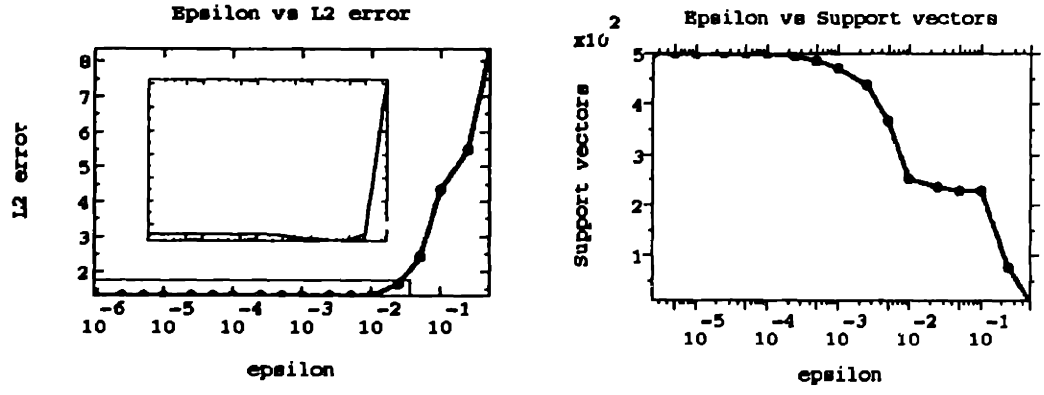
Figure 6-21: (a) The L^2 generalization error versus C for the MG_{17} series. (b) The number of support vectors versus C for the same series. The kernel was an additive trigonometric polynomial with $\nu = 200$.

The parameter ϵ has a strong effect on the number of support vectors and on the generalization error, and its relevance is related to D. In order to see why this happens, remember (see equation 2.1 in section 2.1) that if $R(\lambda)$ and $R_{\text{emp}}(\lambda)$ are respectively the expected risk and empirical risk, with probability $1 - \eta$:

$$I[\mathbf{c}] \leq I_{\text{emp}}[\mathbf{c}] + \tau\sqrt{\frac{h(\log\frac{2N}{h} + 1) - \log\frac{\eta}{4}}{N}}, \tag{6.7}$$

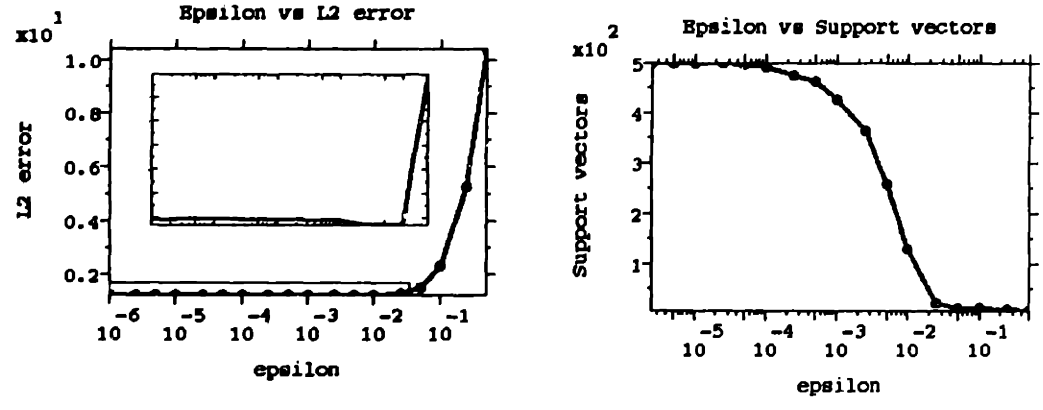
where h is the VC-dimension of the approximation scheme. It is known that the VC-dimension satisfies $h \leq \min(\frac{R^2(A+1)^2}{\epsilon^2}, D) + 1$[108], where R is the radius of the smallest sphere that contains all the data points in the feature space, A is a bound on the norm of the vector of coefficients. When D is small, the VC-dimension h is not dependent on ϵ and the second term on the bound of the generalization error is constant and therefore a very small ϵ does not cause overfitting. For the same reason when D is large the term is very sensitive to ϵ and overfitting occurs for small ϵ. Numerical results confirm this. For example, figures 6-22 and 6-23

which correspond to feature spaces of 802 and infinite dimensions, respectively, show overfitting. (The kernels used were the additive trigonometric polynomial with $\nu = 200$ and a B-spline of order 3, respectively). Figure 6-24 corresponds to a feature space of 10 dimensions and there is no overfitting. (The kernel used was the additive trigonometric polynomial with $\nu = 2$.)



(a) (b)

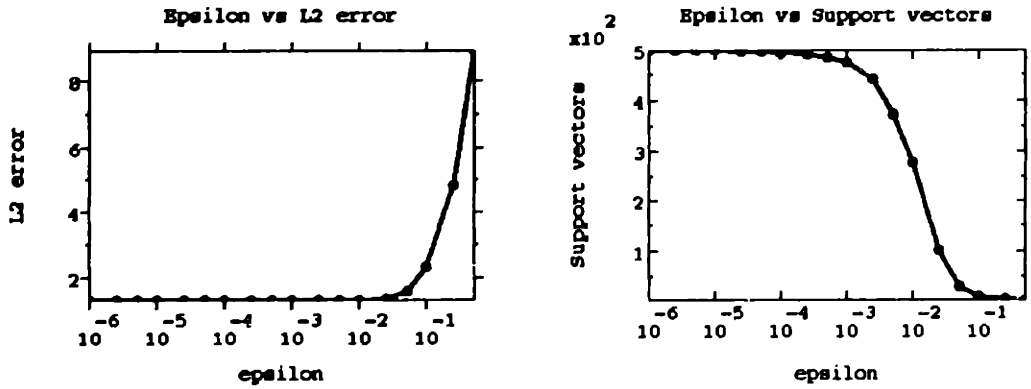
Figure 6-22: (a) The L^2 generalization error versus ϵ with a 802 dimensional feature space. The inset magnifies the boxed region in the lower left section of the plot. Note that overfitting occurs. (b) The number of support vectors versus ϵ for the same feature space.



(a) (b)

Figure 6-23: (a) The L^2 generalization error versus ϵ with an infinite dimensional feature space. The inset magnifies the boxed region in the lower left section of the plot. Note that ovefitting occurs (b) The number of support vectors versus ϵ for the same feature space.

The effect of the embedding dimension m on generalization error was also examined. According to Takens theorem the generalization error should decrease as m approaches the minimum embedding dimension, m^*. Above m^* there should be no



(a) (b)

Figure 6-24: (a) The L^2 generalization error versus ϵ with a 10 dimensional feature space. Note that there is no overfitting (b) The number of support vectors versus ϵ for the same feature space.

decrease in the generalization error. However, if the regression algorithm is sensitive to overfitting the generalization error can increase for $m > m^*$. The minimal embedding dimension of the MG_{17} series is 4. Our numerical results demonstrate that the SVM does not overfit for the case of a low dimensional kernel and overfits slightly for high dimensional kernels, see figure 6-25. The additive trigonometric polynomial with $\nu = 200$ and 2 were used for this figure.

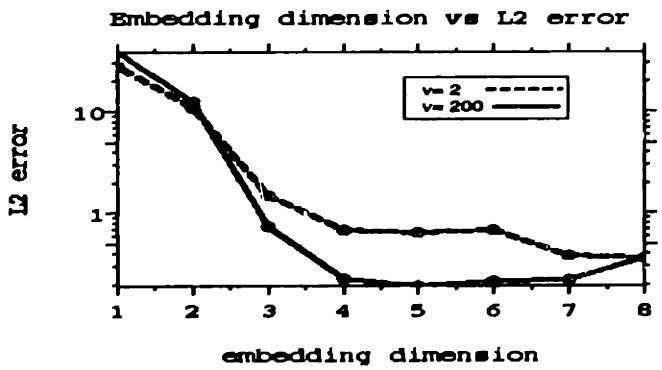


Figure 6-25: The L^2 generalization error versus the embedding dimension. The solid line is for a 802 dimensional feature space and the dashed line is for a 10 dimensional feature space.

6.3.5 Remarks

The SVM algorithm showed excellent performances on the data base of chaotic time series, outperforming the other techniques in the benchmark in all but one case. The generalization error is not sensitive to the choice of C, and very stable with respect to ϵ in a wide range. The variability of the performances with ϵ and D seems consistent with the theory of VC bounds.

Chapter 7

Contributions, Future Research and Conclusions

7.1 Contributions

The contributions of this thesis can be summarized as follows:

1. An active set algorithm for training Support Vector Machines that exploits its geometrical properties, guarantees global optimality, and makes possible the solution of problems with hundreds of thousands of data points that were previously unsolvable.

2. An implementation of this algorithm for both pattern classification and regression problems using MINOS 5.4 [71] [72] as the solver of the subproblems generated by the algorithm. This implementation has been tested under several computer architectures (Sun, Silicon Graphics and IBM PCs) and operating systems (Sun OS, Solaris, Irix and Microsoft Windows NT).

3. A Support Vector Regression Machine solution to the problem of reducing the run-time complexity when performing classification and estimation for the class of problems where many of the non-zero variables λ_i have an active upper bound (i.e., $\lambda_i = C$).

4. A primal reformulation of the SVM training problem which yields the same decision surface obtained by solving the *traditional* QP training problem. This formulation is used for reducing the run-time complexity when many of the non-zero variables λ_i have an active upper bound (i.e., $\lambda_i = C$), and is also used in an alternative formulation to Structural Risk Minimization.

5. An active set algorithm for finding the radius of the smallest sphere that contains the data points in feature space (i.e., the high-dimensional space where data points are mapped in nonlinear SVMs). This algorithm also exploits the geometrical properties of the problem, guarantees global optimality, and solves previously unsolvable problems.

6. An implementation of (5) that also uses MINOS 5.4 as the solver for the subproblems generated by the algorithm.

7. An alternative formulation to Structural Risk Minimization that finds not only the support vectors, but also optimizes over some key parameters that drive the behavior of the kernel-based mapping.

8. Applications of SVMs to face detection, people detection and time series analysis. These applications are important since they have been the first to follow the OCR application developed by Vapnik and his co-workers at AT&T Research.

7.2 Future Research

As stated in section 3.6, there are two main topics regarding SVM training that should be explored further: the solver and the pivoting strategy. A detailed comparison between MINOS 5.4 [72] and QP solvers like LSSOL [38], LOQO [104][105], CPLEX (version 4.0 or above), SQOPT 5.0 [39] [41] and QPOPT 1.0 [39] [40] could help us in reducing the training time even further. Regarding the pivoting strategy, we must say that our implementation is not as sophisticated as we would like, since several promising heuristics (see section 3.6) have not been implemented. This should also be considered in the near future.

Another area where future work should concentrate is the study and understanding of the geometry behind the mapping induced by the kernel operator. The work we have presented in this thesis using parameterized kernels and radius/margin ratios has been just a small step in what should be an active area of research in the near future.

Not least important, we must say that being able to apply the SVM technique to real-life problems is essential to its *survival.* How to actively embed prior knowledge of the problem at hand within the SVM, particularly within the kernel, can be considered an immediate goal. The relationship between the process of *feature selection* and whether (or when) it makes sense to perform the latter is also an open question that should be investigated.

In this thesis we have offered three applications of the technique to real-life problems where we have matched or exceeded the state-of-the-art benchmarks. We believe there will be more to follow.

7.3 Conclusions

This thesis marks the conclusion of three years of hard work and research surrounding a new, excellent, and very promising tool. We believe that even in applications where tailored and hand-crafted techniques can have the leading edge, Support Vector Machines will give many researchers an outstanding (and sometimes tough to outperform!) benchmark or starting point. Its elegance, simplicity and sound mathematical framework make SVMs a very unique tool within the reach of all machine learning researchers.

An important conclusion that can be drawn from this thesis is the immense contribution that optimization theory and tools can offer to the artificial intelligence community, in particular to the research in machine learning. Undoubtly, there will be more to follow.

Appendix A

Methods used in preliminary work

A.1 Zoutendijk's Method (case of linear constraints)

[117][2]:

In order to solve a nonlinear problem of the form:

$$\begin{aligned} &\text{Maximize} && f(\mathbf{x}) \\ &\text{subject to} && \\ & && A\mathbf{x} \leq \mathbf{b} \\ & && E\mathbf{x} = \mathbf{e} \end{aligned} \tag{A.1}$$

this method follows the following skeletal approach:

1. Find $\mathbf{x}_1$ with $A_1\mathbf{x}_1 = \mathbf{b}_1$, $A_2\mathbf{x}_1 < \mathbf{b}_2$ and $E\mathbf{x}_1 = \mathbf{e}$, partitioning $A^T = [A_1^T, A_2^T]$ and $\mathbf{b} = [\mathbf{b}_1, \mathbf{b}_2]$. Let $k = 1$.

2. Given $\mathbf{x}_k$, $A_1\mathbf{x}_k = \mathbf{b}_1$ and $A_2\mathbf{x}_k < \mathbf{b}_2$, find $\mathbf{d}_k$, the optimal solution of:

$$\begin{aligned} &\text{Maximize} \quad \nabla f(\mathbf{x}_k) \cdot \mathbf{d} \\ &\text{subject to} \\ &\qquad A_1\mathbf{d} \qquad \leq \mathbf{0} \\ &\qquad E\mathbf{d} \qquad = \mathbf{0} \\ &\qquad -\mathbf{1} \leq \mathbf{d} \qquad \leq \mathbf{1} \end{aligned} \tag{A.2}$$

3. If $\nabla f(\mathbf{x}_k) \cdot \mathbf{d}_k = 0$, Stop. Else go to 4.

4. Find a step-size δ_k, solution to:

$$\begin{aligned} &\text{Maximize} \quad f(\mathbf{x}_k + \delta\mathbf{d}) \\ &\text{subject to} \\ &\qquad 0 \leq \delta \qquad \leq \delta_{max} \end{aligned} \tag{A.3}$$

where

$$\delta_{max} = \begin{cases} \min_{\hat{d}_i > 0}(\frac{\hat{b}_i}{\hat{d}_i}) & \text{if } \hat{\mathbf{d}} \not\leq \mathbf{0} \\ \infty & \text{if } \hat{\mathbf{d}} \leq \mathbf{0} \end{cases}$$

with $\hat{\mathbf{b}} = \mathbf{b}_2 - A_2\mathbf{x}_k$ and $\hat{\mathbf{d}} = A_2\mathbf{d}_k$.

5. Let $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{d}_k$. Let $k = k + 1$. Go to step 2.

Step 2 involves solving a linear program, which is usually very easy to .In the case of training a SVM, step 2 becomes:

$$
\begin{aligned}
&\text{Maximize} \quad f(\mathbf{d}) = (\mathbf{1} - D\mathbf{\Lambda}_k) \cdot \mathbf{d} \\
&\text{subject to} \\
&\qquad \mathbf{\Lambda} \cdot \mathbf{d} = 0 \\
&\qquad -1 \leq d_i \leq 0 \qquad \text{for } \lambda_i = C \\
&\qquad 0 \leq d_i \leq 1 \qquad \text{for } \lambda_i = 0 \\
&\qquad -1 \leq d_i \leq 1 \qquad \text{otherwise}
\end{aligned}
\tag{A.4}
$$

and step 4 selects $\delta_k = \min(\delta_{opt}, \delta_{max})$, where:

$$
\delta_{opt} = \frac{\mathbf{d} \cdot \mathbf{1} - \mathbf{d} \cdot D\mathbf{\Lambda}}{-2\mathbf{d} \cdot D\mathbf{d}} \qquad \text{and} \qquad \lambda_{max} = \min_{d_i \neq 0} \{ \min_{\lambda_i < C, d_i > 0} (\frac{C - \lambda_i}{d_i}), \min_{\lambda_i > 0, d_i < 0} (\frac{-\lambda_i}{d_i}) \}
$$

A.2 MINOS 5.4:

Wolfe's Reduced Gradient method depends upon reducing the dimensionality of the problem by representing all variables in terms of an independent set of them. Under non-degeneracy assumptions (to facilitate this brief description), a program of the form:

$$
\begin{aligned}
&\text{Minimize} \quad f(\mathbf{x}) \\
&\text{subject to} \\
&\qquad A\mathbf{x} = \mathbf{b} \\
&\qquad \mathbf{x} \geq \mathbf{0}
\end{aligned}
$$

can be decomposed into $A = [B, N]$, $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ with B non-singular, $\mathbf{x}_B > \mathbf{0}$ and $\mathbf{x}_N \geq \mathbf{0}$.

By denoting the gradient $\nabla f(\mathbf{x}) = (\nabla_B f(\mathbf{x}), \nabla_N f(\mathbf{x}))$ and a direction vector $\mathbf{d} = (\mathbf{d}_B, \mathbf{d}_N)$, the system $A\mathbf{d} = \mathbf{0}$ holds for any choice of $\mathbf{d}_N$ by letting $\mathbf{d}_B = -B^{-1}\mathbf{d}_N$.

Defining $\mathbf{r} = (\mathbf{r}_B, \mathbf{r}_N) \equiv \nabla f(\mathbf{x}) - \nabla_B f(\mathbf{x}) B^{-1} A = (\mathbf{0}, \nabla_N f(\mathbf{x}) - \nabla_B f(\mathbf{x}) B^{-1} N)$ as the *reduced gradient*, it follows that $\nabla f(\mathbf{x}) \cdot \mathbf{d} = \mathbf{r}_N \cdot \mathbf{d}_N$. Therefore, in order to have a feasible direction $\mathbf{d}$ to be an improving feasible direction (feasibility and $\nabla f(\mathbf{x}) \cdot \mathbf{d} < 0$), a vector $\mathbf{d}_N$ must be chosen such that $\mathbf{r}_N \cdot \mathbf{d}_N < 0$ and $d_j \geq 0$ for $x_j = 0$. This can be accomplished by choosing $\mathbf{d}_B = -B^{-1}\mathbf{d}_N$ and:

$$d_{N_j} = \begin{cases} -r_j & \text{if } r_j \leq 0 \\ -x_j r_j & \text{if } r_j > 0 \end{cases}$$

for $j \in N$.After determining the improving feasible direction $\mathbf{d}$, a line-search procedure is used to determine the step-size, and an improved solution is obtained.

Reduced gradient methods allow all components of $\mathbf{d}_N$ to be non-zero. On the opposite side, for example, the simplex method for linear programming examines a similar direction-finding problem, but allow only one component of $\mathbf{d}_N$ to be non-zero at a time. It is interesting to see that although the second strategy looks too restrictive, the first one also can result in a slow progress, as sometimes only small step-sizes are possible due to the fact that many components are changing simultaneously.

In order to reach a compromise between the two strategies mentioned above, the set of non basic variables $\mathbf{x}_N$ can be further partitioned into $(\mathbf{x}_S, \mathbf{x}_{N'})$, with the corresponding decomposition of $N = [S, N']$ and $\mathbf{d}_N = (\mathbf{d}_S, \mathbf{d}_{N'})$. The variables $\mathbf{x}_s$ are called *superbasic variables*, and are intended to be the *driving force* of the iterates while $\mathbf{x}_{N'}$ is fixed and $\mathbf{x}_B$ is adjusted to maintain feasibility [71].

Notice that the direction vector $\mathbf{d}$ can be accordingly decomposed through a linear operator Z of the form:

$$\mathbf{d} = \begin{bmatrix} \mathbf{d}_B \\ \mathbf{d}_S \\ \mathbf{d}_{N'} \end{bmatrix} = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \mathbf{d}_S \equiv Z\mathbf{d}_S \qquad \text{(A.5)}$$

and now the search direction along the surface of the active constraints is char-

acterized as being in the range of a matrix Z which is orthogonal to the matrix of constraint normals, i.e.,

$$AZ = [B, S, N'] \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} = 0. \tag{A.6}$$

By expressing the *directional derivative* $\nabla f(\mathbf{x}) \cdot \mathbf{d}$ as:

$$\nabla f(\mathbf{x}) \cdot \mathbf{d} = \nabla f(\mathbf{x}) \cdot Z\mathbf{d}_S = [\nabla_S f(\mathbf{x}) - \nabla_B f(\mathbf{x}) B^{-1} S]\mathbf{d}_S = \mathbf{r}_S \cdot \mathbf{d}_S \tag{A.7}$$

where $\mathbf{r}_S = \nabla_S f(\mathbf{x}) - \nabla_B f(\mathbf{x}) B^{-1} S$, and the direction finding problem can therefore be reduced to:

$$\begin{aligned} &\text{Minimize} && \mathbf{r}_S \cdot \mathbf{d}_S \\ &\text{subject to} \\ &&& -x_j |r_j| \leq d_j \leq |r_j| \quad \text{for } x_j \text{ superbasic.} \end{aligned} \tag{A.8}$$

Given that the direction finding problem described by equation (A.8) uses a linear approximation to the objective function, slow and zigzagging convergence is likely to happen when the contours of f are *flat* or *thin* in some directions. Therefore, we would expect faster convergence when this approach is upgraded by taking a second-order approximation to f. More formally, the goal is to minimize a second-order approximation to the direction finding problem given by:

$$f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{d} + \frac{1}{2}\mathbf{d} \cdot H(\mathbf{x})\mathbf{d} \tag{A.9}$$

over the linear manifold $A\mathbf{d} = 0$.

Using equations (A.7) and (A.5), (A.9) transforms into:

$$\min\{\mathbf{r}_S \cdot \mathbf{d}_S + \frac{1}{2}\mathbf{d}_S \cdot Z^T H(\mathbf{x}) Z \mathbf{d}_S\} \tag{A.10}$$

where the matrix $Z^T H(\mathbf{x})Z$ is called the *reduced Hessian.*

Setting the gradient of (A.10) equal to zero results in the system of equations:

$$Z^T H(\mathbf{x}) Z \mathbf{d}_S = -\mathbf{r}_S \tag{A.11}$$

Once $\mathbf{d}_S$ is available, a line-search along the direction $\mathbf{d} = Z\mathbf{d}_S$ is performed and a new solution is obtained.

MINOS 5.4 implements (A.11) with certain computational highlights [71]:

1. During the algorithm, if it appears that no more improvement can be made with the current partition $[B, S, N]$, that is, $\|\mathbf{r}_S\| < \varepsilon$, for a suitably chosen tolerance level ε, some of the non-basic variables are added to the superbasics set. Using a *Multiple Pricing* option, MINOS allows the user to specify how many of them to incorporate.

2. If at any iteration a basic or superbasic variable reaches one of its bounds, the variable is made non-basic.

3. The matrices Z, $H(\mathbf{x})$ and $Z^T H(\mathbf{x})Z$ are never actually computed, but are used implicitly

4. The reduced Hessian matrix $Z^T H(\mathbf{x})Z$ is approximated by $R^T R$, where R is a dense upper triangular matrix.

5. A sparse LU factorization of the basis matrix B is used.

Bibliography

[1] M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition. *Automatic and Remote Control*, 25(6):821–837, June 1964.

[2] M. Bazaraa, H. Sherali, and C. Shetty. *NonLinear Programming Theory and Algorithms*. J. Wiley, New York, 2nd edition, 1993.

[3] K. Bennet and J. Blue. A support vector machine approach to decision trees. Technical Report 97-100, Mathematical Sciences Department, Rensselaer Polytechnic Institute,Troy,NY 12180, 1997. This report can be downloaded from http://www.math.rpi.edu/ bennek/svmdt.ps.

[4] K. Bennet and O. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

[5] D. Bertsekas. Projected newton methods for optimization problems with simple constraints. *SIAM J. Control and Optimization*, 20(2):221–246, March 1982.

[6] V. Blanz, B. Schölkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view-based object recognition algorithms using realistic 3d models. In *Proc. International Conference on Artificial Neural Networks*, Berlin, 1996.

[7] I. Bongartz, P. Calamai, and A. Conn. A projection method for l_p norm location-allocation problems. Technical report, Dept. of Systems design Engineering, University of Waterloo, and Thomas J. Watson Research Center, IBM, July 1995.

[8] B. Boser, I. Guyon, and V. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, Pittsburg, PA, 1992. ACM.

[9] P. Bradley and O. Mangasarian. Data mining: Overview and optimization opportunities. Technical Report 98-01, Computer Sciences Department, University of Wisconsin, January 1998.

[10] P. Bradley and O. Mangasarian. Feature selection via concave minimization and support vector machines. Technical Report 98-03, Computer Sciences Department, University of Wisconsin, February 1998.

[11] J. Bunch and L. Kaufman. A computational method for the indefinite quadratic programming problem. *Linear Algebra and its Applications*, 34:341–370, 1980.

[12] G. Burel and D. Carel. Detection and localization of faces on digital images. *Pattern Recognition Letters*, 15:963–967, 1994.

[13] C. Burges. Simplified support vector decision rules. *Procedings of the 13th International Conference on Machine Learning*, pages 71–77, 1996.

[14] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 1998. (in press).

[15] C. Burges and B. Schölkopf. *Improving the Accuracy and Speed of Support Vector Machines*, volume 9 of *Neural Information Processing Systems*. M. Mozer, M. Jordan and T. Petsche (eds), MIT Press, 1997(in press).

[16] C. Burges and V. Vapnik. A new method for constructing artificial neural networks. Technical report, AT&T Bell Laboratories, 101 Crawfords Corner Road Holmdel NJ 07733, May 1995.

[17] P. Calamai and C. Charalambous. Solving multifacility location problems involving euclidean distances. *Naval Research Quarterly*, 27(4):609–620, December 1980.

[18] P. Calamai and A. Conn. A stable algorithm for solving the multifacility location problem involving euclidean distances. *SIAM Journal on Scientific and Statistical Computing*, 1(4):512–526, December 1980.

[19] P. Calamai and A. Conn. A projected newton method for l_p norm location problems. *Mathematical Programming*, 38:75–109, 1987.

[20] T. Carpenter and D. Shanno. An interior point method for quadratic programs based on conjugate projected gradients. *Computational Optimization and Applications*, 2(1):5–28, June 1993.

[21] M. Casdagli. Nonlinear prediction of chaotic time-series. *Physica D*, 35:335–356, 1989.

[22] R. Chakraborty and P. Chaudhuri. Note on geometrical solutions for some minimax location problems. *Transportation Science*, 15(2):164–166, May 1981.

[23] H-J. Chen and Y. Shirai. Detecting multiple image motions by exploiting temporal coherence of apparent motion. *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 899–902, 1994.

[24] R. Chen. Solution of minisum and minimax location-allocation problems with euclidena distances. *Naval Research Logistics Quarterly*, 30:449–459, 1983.

[25] S. Chen, , D. Donoho, and M. Saunders. Atomic decomposition by basis pursuit. Technical Report 479, Department of Statistics, Stanford University, May 1995.

[26] S. Chen. *Basis Pursuit.* PhD thesis, Department of Statistics, Stanford University, November 1995.

[27] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:1–25, 1995.

[28] Y. Le Cun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to Handwritten Zip Code Recognition. *Neural Computation*, 1:541–551, 1989.

[29] Z. Drezner and S. Shelah. On the complexity of the elzinga-hearn algorithm for the 1-center problem. *Mathematics of Operations Research*, 12(2):255–261, May 1987.

[30] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. Vapnik. *Neural Information Processing Systems*, volume 9, chapter Support Vector Regression Machines. MIT Press,(eds)M. Mozer, M. Jordan, and T. Petsche, 1997 (in press).

[31] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1973.

[32] J. Elzinga and D. Hearn. Geometrical solutions for some minimax location problems. *Transportation Science*, 6(4):379–394, November 1972.

[33] J. Elzinga and D. Hearn. The minimum covering sphere problem. *Management Science*, 19(1):96–104, September 1972.

[34] J. Elzinga and D. Hearn. A note on a minimax location problem. *Transportation Science*, 7:100–103, 1973.

[35] J. Elzinga and D. Hearn. The minimum sphere covering a convex polyhedron. *Naval Research Logistics Quarterly*, 21:715–718, 1974.

[36] J. Elzinga, D. Hearn, and W. Randolph. Minimax multifacility location with euclidean distances. *Transportation Science*, 10:321–336, 1976.

[37] R. Francis and A. Cabot. Porperties of a multifacility location problem involving euclidean distances. *Naval Research Logistics Quarterly*, 19:335–353, 1972.

[38] P. Gil, S. Hammarling, W. Murray, M. Saunders, and M. Wright. *User's Guide for LSSOL: A fortran package for constrained linear least-squares and convex quadratic programming.* System Optimization Laboratory,Stanford University, Jan 1986.

[39] P. Gill and W. Murray. Numerically stable methods for quadratic programming. *Mathematical Programming*, 14:349–372, 1978.

[40] P. Gill, W. Murray, and M. Saunders. User's guide for qpopt 1.0: A fortran package for quadratic programming. Technical report, Dept. of Mathematics, UCSD and SOL, Stanford University, August 1995.

[41] P. Gill, W. Murray, and M. Saunders. User's guide for sqopt 5.0: A fortran package for large-scale linear and quadratic programming. Technical report, Dept. of Mathematics, UCSD and SOL, Stanford University, October 1997(draft).

[42] P. Gill, W. Murray, and M. Wright. *Practical Optimization.* Academic Press, 1981.

[43] F. Girosi. An equivalence between sparse approximation and support vector machines. A.I. Memo 1606, MIT Artificial Intelligence Laboratory, 1997. (available at the URL: http://www.ai.mit.edu/people/girosi/svm.html).

[44] I. Guyon, V. Vapnik, B. Boser, L. Bottou, and S. Solla. Structural risk minimization for character recognition. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems IV*, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[45] T. Hastie, A. Buja, and R. Tibshirani. Penalized discriminant analysis. Technical report, Statistics and Data Analysis Research Department, AT&T Bell Laboratories, Murray Hill, New Jersey, May 1994.

[46] T. Hastie and R. Tibshirani. Discriminat adaptative nearest neighbor classification. Technical report, Department of Statistics and Division of Biostatistics, Stanford University, December 1994.

[47] D. Hearn and J. Vijay. Efficient algorithms for the (weighted) minimax location problem. *Operations Research*, 30(4):777–795, july 1982.

[48] P. Huber. Robust estimation of location parameters. *Annals of Mathematical Statistics*, 35(1), 1964.

[49] P.J. Huber. *Robust Statistics.* John Wiley and Sons, New York, 1981.

[50] K. Ikeda. Multiple-valued stationary state and its instability of light by a ring cavity system. *Opt. Commun.*, 30:257, 1979.

[51] C.E. Jacobs, A. Finkelstein, and D.H. Salesin. Fast multiresolution image querying. *SIGGRAPH95*, August 1995. University of Washington, TR-95-01-06.

[52] T. Joachins. Text categorization with support vector machines: Learning with many relevant features. Technical Report LS-8 Report 23, Dortmund University, November 1997.

[53] L. Kaufman. Solving the quadratic programming problem arising in support vector classification. (personal communication).

[54] N. Krüger, M. Pötzsch, and C. v.d. Malsburg. Determination of face position and pose with learned representation based on labled graphs. Technical Report 96-03, Ruhr-Universität, January 1996.

[55] H. Kuhn. A note on fermat's problem. *Mathematical Programming*, 4:98–107, 1973.

[56] M.K. Leung and Y-H. Yang. Human body motion segmentation in a complex scene. *Pattern Recognition*, 20(1):55–64, 1987.

[57] M.K. Leung and Y-H. Yang. A region based approach for human body analysis. *Pattern Recognition*, 20(3):321–39, 1987.

[58] E.N. Lorenz. Deterministic non-periodic flow. *J. Atoms. Sci.*, 26:636, 1969.

[59] R. Love. Locating facilities in three-dimensional space by convex programming. *Naval Research Logistics Quarterly*, 16:503–516, 1969.

[60] M.C. Mackey and J. Glass. Oscillation and chaos in physiological contol systems. *Science*, 197:287, 1977.

[61] S.G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–93, July 1989.

[62] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

[63] B. Moghaddam and A. Pentland. Probabilistic visual learning for object detection. Technical Report 326, MIT Media Laboratory, June 1995.

[64] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[65] E.H. Moore. On properly positive Hermitian matrices. *Bull. Amer. Math. Soc.*, 23(59):66–67, 1916.

[66] J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optimization*, 1(1):93–113, February 1991.

[67] J. Moré and S. Wright. *Optimization Software Guide.* SIAM, 1993.

[68] J. Morris. A further note on a minimax location problem. *Transportation Science*, 10(4):405–408, November 1976.

[69] S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using support vector machines. In *IEEE Workshop on Neural Networks and Signal Processing*, Amelia Island, FL, September 1997.

[70] K. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *Proc. International Conference on Artificial Neural Networks*, 1997. (in press).

[71] B. Murtagh and M. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.

[72] B. Murtagh and M. Saunders. *MINOS 5.4 User's Guide.* System Optimization Laboratory,Stanford University, Feb 1995.

[73] P. Niyogi and F. Girosi. On the Relationship between Generalization Error, Hypothesis Complexity, and Sample Complexity for Radial Basis Functions. Technical Report AIM–1467, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

[74] B.A. Olshausen and D.J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

[75] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *Proc. Computer Vision and Pattern Recognition*, pages 193–199, Puerto Rico, June 16–20 1997.

[76] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *IEEE Workshop on Neural Networks and Signal Processing*, Amelia Island, FL, September 1997.

[77] E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. A.I. Memo 1602, MIT A. I. Lab., 1997.

[78] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *Proc. Computer Vision and Pattern Recognition*, Puerto Rico, June 16–20 1997.

[79] D. Parker. Learning Logic. Technical Report TR-47, Center of Computational Research in Economics and Management Science, MIT, 1985.

[80] T. Poggio and F. Girosi. A Theory of Networks for Approximation and Learning. Technical Report AIM–1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1989.

[81] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[82] T. Poggio and T. Vetter. Recognition and structure from one 2D model view: observations on prototypes, object classes and symmetries. A.I. Memo No. 1347, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1992.

[83] M. Pontil and A. Verri. Direct aspect-based 3-d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997. (submitted).

[84] M. Pontil and A. Verri. Properties of support vector machines. *Neural Computation*, 1998. (in press).

[85] F. Riesz and B. Sz.-Nagy. *Functional Analysis.* Ungar, New York, 1955.

[86] B.D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

[87] K. Rohr. Incremental recognition of pedestrians from image sequences. *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 8–13, 1993.

[88] H. Rowley, S. Baluja, and T. Kanade. Human Face Detection in Visual Scenes. Technical Report 95–158, CMU CS, July 1995. Also in *Advances in Neural Information Processing Systems* (8):875-881.

[89] D. Rumelhart and J. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, Massachusetts, 1986.

[90] M. Schmidt. Identifying speakers with support vectors networks. In *Proceedings of Interface '96*, Sydney, July 1996.

[91] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U.M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, 1995. AAAI Press.

[92] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In *Proc. International Conference on Artificial Neural Networks*, Berlin, 1996.

[93] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical Report 44, Max Planck Institute, December 1996.

[94] P. Sinha. Object Recognition via Image Invariants: A Case Study. *Investigative Ophthalmology and Visual Science*, 35:1735–1740, May 1994.

[95] P. Sinha. Qualitative image-based representations for object recognition. *MIT AI Lab-Memo*, No. 1505, 1994.

[96] A. Smola and B. Schölkopf. On a kernel-based method for pattern recognition, regression, apporximation and operator inversion. Technical Report 1064, GMD First, Berlin and MPI, Tübingen, Germany, 1997.

[97] J. Stewart. Positive definite functions and generalizations, an historical survey. *Rocky Mountain J. Math.*, 6:409–434, 1976.

[98] E.J. Stollnitz, T.D. DeRose, and D.H. Salesin. Wavelets for computer graphics: A primer. *University of Washington, TR-94-09-11*, September 1994.

[99] K. Sung. *Learning and Example Selection for Object and Pattern Detection.* PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory and Center for Bilogical and Computational Learning, December 1995.

[100] K. Sung and T. Poggio. Example-based learning for view-based human face detection. *A.I. MEMO 1521, C.B.C.L Paper 112*, December 1994.

[101] F. Takens. Detecting strange attractors in fluid turbulence. In D. Rand and L.S. Young, editors, *Dynamical Systems and Turbulence.* Springer-Verlag, Berlin, 1981.

[102] T. Tsukiyama and Y. Shirai. Detection of the movements of persons from a sparse sequence of tv images. *Pattern Recognition*, 18(3/4):207–13, 1985.

[103] R. Vaillant, C. Monrocq, and Y. Le Cun. Original approach for the localisation of objects in images. *IEE Proc.-Vis. Image Signal Processing*, 141(4), August 1994.

[104] R. Vanderbei. Interior point methods: Algorithms and formulations. *ORSA J. of Computing*, 6(1):32–34, 1994.

[105] R. Vanderbei. Loqo: An interior point code for quadratic programming. Technical report, Program in Statistics and Operations Research, Princeton University, 1994.

[106] V. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, 1982.

[107] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[108] V. Vapnik. *Statistical learning theory.* J. Wiley(in press), 1998.

[109] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian].* Nauka, 1974.

[110] V. Vapnik and A. Chervonenkis. The necessary and sufficient conditions for consistency in the empirical risk minimization method. *Pattern Recognition and Image Analysis*, 1(3):283–305, 1991.

[111] V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In *Advances in Neural Information Processings Systems 9*, pages 281–287, San Mateo, CA, 1997. Morgan Kaufmann Publishers.

[112] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequences of events to their probabilities. *Th. Prob. and its Applications*, 17(2):264–280, 1971.

[113] G. Wesolowsky. Rectangular distance location under the minimax optimality criterion. *Transportation Science*, 6:103–113, 1972.

[114] G. Yang and T. Huang. Human face detection in a complex background. *Pattern Recognition*, 27:53–63, 1994.

[115] W.Y. Young. A note on a class of symmetric functions and on a theorem required in the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

[116] X. Zhang. Non-linear predictive models for intra-day foreign exchange trading. Technical report, PHZ Partners, August 1993.

[117] G. Zoutendijk. *Methods of Feasible Directions.* D. Van Norstrand, Princeton, N.J., 1960.