Using Interior Point Methods for Large-scale Support Vector Machine training

Kristian Woodsend

Doctor of Philosophy
University of Edinburgh
2009

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Kristian Woodsend*)

Acknowledgements

As I approach the point where I am ready to submit, it is time to look back and acknowledge the many people that have helped me get here.

I owe most thanks to my supervisor Jacek Gondzio, for his patience and guidance as a mentor while I attempted to grow into a competent researcher. I have relied on his expert knowledge of interior point methods to guide me to the techniques really worth doing. While the work here is mine, the best ideas are really his.

Andreas Grothey and Marco Colombo have helped throughout my years of study. Their interior point solver OOPS has been a fundamental tool. Without their thorough understanding of how it works—and perhaps more importantly their patience in explaining it to me—this work would have been so much harder. The structure-conveying modelling language in Chapter 11 is joint work with Jacek, Andreas, Marco and Jonathan Hogg.

I would also like to thank Roger Fletcher, Bo Kågström, Ken McKinnon, Katya Scheinberg, Sören Sonnenberg and Gaetano Zanghirati. They all have heard me present aspects of this work at various stages, and through their ideas, encouragement or the perceptiveness of their questions, they have all in some way pushed my research onwards.

I am also grateful to the anonymous referees of the journals JMLR and COAP, and the referees for the INYS "High Performance Scientific Computing" workshop in Lithuania, for all their comments and careful reading of my submissions. The feedback that I received by submitting to those places has improved this thesis as well.

I am indebted to the PASCAL organisers for permission to use the data sets, Professor Zanni for providing the PGPDT software, and Dr Durdanovic for assistance with the Milde software. This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF, http://www.ecdf.ed.ac.uk). The ECDF is partially supported by the eDIKT initiative (http://www.edikt.org).

Finally I would like to express my gratitude to Liz, who as well as proof-reading this thesis and creating the diagrams, has supported me in countless ways during the last few years.

To my father—

I would never have had the courage to shift to research without his unconventional career as an example.

Abstract

Support Vector Machines (SVMs) are powerful machine learning techniques for classification and regression, but the training stage involves a convex quadratic optimization program that is most often computationally expensive. Traditionally, active-set methods have been used rather than interior point methods, due to the Hessian in the standard dual formulation being completely dense. But as active-set methods are essentially sequential, they may not be adequate for machine learning challenges of the future. Additionally, training time may be limited, or data may grow so large that cluster-computing approaches need to be considered.

Interior point methods have the potential to answer these concerns directly. They scale efficiently, they can provide good early approximations, and they are suitable for parallel and multi-core environments. To apply them to SVM training, it is necessary to address directly the most computationally expensive aspect of the algorithm. We therefore present an exact reformulation of the standard linear SVM training optimization problem that exploits separability of terms in the objective. By so doing, per-iteration computational complexity is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n)$. We show how this reformulation can be applied to many machine learning problems in the SVM family.

Implementation issues relating to specializing the algorithm are explored through extensive numerical experiments. They show that the performance of our algorithm for large dense or noisy data sets is consistent and highly competitive, and in some cases can outperform all other approaches by a large margin. Unlike active set methods, performance is largely unaffected by noisy data. We also show how, by exploiting the block structure of the augmented system matrix, a hybrid MPI/OpenMP implementation of the algorithm enables data and linear algebra computations to be efficiently partitioned amongst parallel processing nodes in a clustered computing environment.

The applicability of our technique is extended to nonlinear SVMs by low-rank approximation of the kernel matrix. We develop a heuristic designed to represent clusters using a small number of features. Additionally, an early approximation scheme reduces the number

of samples that need to be considered. Both elements improve the computational efficiency of the training phase.

Taken as a whole, this thesis shows that with suitable problem formulation and efficient implementation techniques, interior point methods are a viable optimization technology to apply to large-scale SVM training, and are able to provide state-of-the-art performance.

Contents

Chapter 1

Introduction

Conventional wisdom is that, while interior point methods may be a reasonable choice of numerical optimization technique for small or medium-sized subproblems, they are not suitable in themselves for tackling large-scale support vector machine training (Kaufman, 1999; Shevade et al., 2000; Cristianini and Shawe-Taylor, 2000; Schölkopf and Smola, 2002). The objective of this thesis is to reconsider that view.

Support Vector Machines (SVMs) are powerful machine learning techniques for classification and regression. They were developed by Vapnik (1998), and are based on statistical learning theory. Since then, they have been applied to a wide range of applications, with excellent results, and so they have received significant interest.

Taking binary classification as the fundamental problem, SVMs follow the geometrically intuitive approach of finding a hyperplane that divides the two classes. The approach to training, which aims to both maximize the width of the margin that surrounds the separating hyperplane and minimize the classification errors incurred, involves a convex quadratic optimization program (QP) that is unfortunately computationally expensive.

A quadratic program is an optimization problem with an objective function comprising linear and quadratic terms, and linear constraints. If the Hessian matrix that defines the quadratic objective function is positive definite or semi-definite, the problem is said to be convex, and any local minimizer will also be the global minimizer. It is always possible to solve this type of problem in a finite number of operations, but the effort required to do so depends on the characteristics of the problem: in some cases the number of operations will scale cubically or even factorially with the number of variables, while in others the QP is only fractionally more difficult to solve than a similarly-sized linear program.

Solving a QP containing inequality constraints is harder than a similar one containing

only equality constraints. It is necessary to find which constraints are active at the solution, and this requires an iterative approach. Nocedal and Wright (1999) list several standard methods:

- *Active set methods* have been used since 1970s, and they are most effective for small to medium-sized problems, or hyper-sparse problems (Hall and McKinnon, 2005). The simplex method (Dantzig, 1963) can be viewed as an active set method specialized for linear programs. Using an initially feasible iterate, the algorithm makes a guess at the optimal active set of constraints, known as the working set, and solves a QP subproblem where these constraints are imposed as equalities. If the guess is incorrect, the method uses Lagrange multiplier information to drop one index from the working set and add a new one.

- The *gradient-projection method* is similar, but allows more rapid changes to the working set to accelerate the procedure. It tends to work well when the constraints are very simple, for instance when they take the form of bounds.

- *Interior point methods* (IPMs) work by perturbing the optimality conditions relating to complementarity through the use of logarithmic barriers, and so delay the split between active and inactive constraints for as long as possible. This will be described in more detail in Chapter 2.

The potential-reduction method of Karmarkar (1984), which provided a theoretical result of polynomial complexity, marks the beginning of efforts to develop interior point methods, although the use of logarithmic barriers in nonlinear optimization had been known for some time—they were introduced by Frisch (1955) and described in detail by Fiacco and McCormick (1968). The most efficient infeasible interior point algorithms have bounds $\mathcal{O}(n \log \frac{1}{\epsilon})$, where n is the number of variables and ϵ is the tolerance required to terminate (Mizuno, 1994; Roos, 2006).

There is a considerable gap between theoretical results and practice though, and an iteration count of $\mathcal{O}(\log n)$ or $\mathcal{O}(n^{\frac{1}{4}})$ is to be expected from IPM software (Andersen et al., 1996). It is this property, that the required number of iterations grows very slowly with the size of the problem, which makes interior point methods so attractive for large-scale optimization problems.

Seen from this background, it is something of an anomaly that IPM are not the optimization method of choice for large-scale SVM training. Yet, almost all the standard methods

are variants of the active set method (see Section 4). There are good reasons why active set methods have dominated. The SVM training problem has these characteristics:

- Both the primal and dual formulations scale with the number of data points n rather than the feature space dimension.
- The primal formulation of the training problem is large, as the number of constraints also scales with n.
- The dual formulation has n variables but only one constraint. It has the advantage of allowing nonlinear discrimination through kernel-based learning, but the result is a dense $n \times n$ Hessian matrix.

Using standard interior point methods, both primal and dual formulations would have a per-iteration complexity of $\mathcal{O}(n^3)$. This complexity result makes applying SVMs to large-scale data-sets challenging, and in practice the optimization problem is intractable by general-purpose IPM solvers. Active set methods using the dual form have the advantage that they reduce the problem to a series of smaller sub-problems.

It is possible that existing approaches may not be adequate for machine learning challenges of the future. Large-scale considerations such as scalability and efficiency are now becoming important issues, with the exponential increase in storage capacity and computing power, and fields such as text categorisation, image recognition, and bioinformatics generating huge real-world data sets. The data may grow so large that they do not fit into the memory of a single computer, and cluster-computing approaches have to be considered. Training can also be limited in time: it may be preferable to have a good approximation to the classification function early rather than a lengthy training time to achieve a solution to full accuracy.

In addition, the active set techniques used by standard software are essentially sequential—they choose a small subset of variables to form the active set at each iteration, and this selection is based upon the results of the previous iteration. Due to the dependencies between each iteration and the next, parallel implementations have distributed the work of each subproblem even though a large number of iterations may be required. There have been only a few approaches developed for training SVMs in parallel, yet multiple-core computers are becoming the norm. It is notable that of the 44 submissions to compete in the PASCAL Challenge on Large-scale Learning (Sonnenburg et al., 2008), only 3 entries were parallel methods.

In this work, our aim is to develop methods which address the above concerns directly: methods that scale efficiently, that can provide good early approximations, and that are suitable for parallel and multi-core environments. Rather than look for simplifications, we stick strictly to the original optimization problems of Vapnik (1998). Additionally, we concentrate on problems where the number of *samples* n is much greater than the number of *attributes* m.

1.1 Thesis outline

The structure of this thesis is as follows:

- Chapter 2 is a description of interior point methods, concentrating on the practical aspects of the solver that will benefit from specialization.
- Chapter 3 describes SVMs, and Chapter 4 outlines the optimization approaches to SVM training proposed so far.
- My contribution starts at Chapter 5, where we present an exact reformulation of the standard linear SVM training optimization problem that, through exploiting separability in the objective, directly addresses the most computationally expensive aspect of the IPM algorithm. By so doing, per-iteration computational complexity is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n)$. We also show how this reformulation can be applied to many machine learning problems in the SVM family.
- Chapter 6 discusses issues related to the implementation of the formulation. The implementation is based on the serial IPM solver `HOPDM` (Gondzio, 1995). We investigate techniques for specializing the solver, and make a comparison with other linear SVM software.
- We then adapt the formulation to a parallel clustered computing environment in Chapter 7, using the IPM solver `OOPS` (Gondzio and Grothey, 2007), and describe how to exploit the problem structure to handle both data access and linear algebra aspects efficiently.
- In problems where $n \gg m$, there is a large number of variables but only a relatively small number (around $m+1$) of them are not at their bounds. In Chapter 8 we use approximation and column generation methods to identify these variables; doing so enables the size of the problem given to the IPM solver to be reduced.

- Nonlinear kernels are a powerful extension to SVMs, allowing them to handle data sets that are not linearly separable. Chapter 9 investigates approximating the kernel matrix with a low-rank matrix, allowing the techniques of previous chapters to be applied. The per-iteration complexity of the optimization algorithm depends quadratically on the rank of the approximating matrix, so Chapter 10 proposes a heuristic designed to capture clustering information with as low rank a matrix as possible.

- Then in Chapter 11 we take a different approach. Algebraic modelling languages are important tools in optimization, as they allow the problems to be specified to solvers in a language close to mathematical notation, and so remove the need for the error-prone work of describing the problem using a programming language. We investigate if, through extending an algebraic modelling language to convey structural information, it would have been possible for the specializations in previous chapters to be handled automatically.

- Finally, Chapter 12 provides our conclusions and then discusses further research avenues.

1.2 Related publications

Some of this work has appeared already, or is in the process of being published. The separable formulation of Chapters 5 and 6 is the basis of Woodsend and Gondzio (2009b). The parallel approach of Chapter 7 is in Woodsend and Gondzio (2009c). Our implementation was evaluated as part of the PASCAL Challenge on Large-scale Learning (Sonnenburg et al., 2008), and a summary was presented at an ICML 2008 workshop. The description of a parallel partial Cholesky decomposition algorithm appeared in Woodsend and Gondzio (2009a), while the heuristic for choosing columns in Chapter 10 was shown as a poster at a workshop on large-scale learning at NIPS 2007. The modelling language in Chapter 11 has been described in Grothey et al. (2009); Colombo et al. (2009), although the application to SVMs presented here is new.

1.3 Notation

Finally, let us now briefly describe the notation used in this thesis. x_i is the attribute vector for the i^{th} data point, and it consists of the observation values directly. There are n samples

in the training set, and m attributes in each vector x_i. X is the $m \times n$ matrix whose columns are the attribute vectors x_i associated with each point. The classification label for each data point is denoted by $y_i \in \{-1, 1\}$. The variables $w \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$ are used for the primal variables ("weights") and dual variables (α in SVM literature) respectively, and $w_0 \in \mathbb{R}$ for the bias of the hyperplane. Scalars and column vectors are denoted using lower case letters, while upper case letters denote matrices. D, S, U, V, Y and Z are the diagonal matrices of the corresponding vectors.

We have used a rather loose notation to describe the order of complexity of calculations. $\mathcal{O}(nm^2+m^3)$ indicates that there are two components of the algorithm which could dominate, one of order nm^2 and another of m^3.

The literature on IPMs and SVMs have both developed their own conventions on notation. We have kept the SVM convention of using x and y to represent input and output data, but in general we follow IPM conventions. To aid the reader, the notation is summarized in Appendix C.

Chapter 2

Interior point methods

Interior point methods represent state-of-the-art techniques for solving linear, quadratic and non-linear optimization problems. This chapter gives an outline of the derivation of an interior point method for bound and equality-constrained quadratic programs, but in the main it concentrates on the key implementation issues surrounding the development of an efficient interior point solver for this type of problem. For a full discussion on theoretical aspects, see Roos et al. (2005); Wright (1997); Ye (1997).

2.1 Concepts and definitions

A linear program (LP) is an optimization problem where the aim is to find the vector of real variables $z \in \mathbb{R}^n$ that minimizes (or possibly maximizes) a linear objective function and subject to ("s.t.") a set of linear constraint functions. The standard LP formulation takes the form

$$\begin{aligned} \min_{z} \quad & c^T z \\ \text{s.t.} \quad & Az = b \\ & z \geq 0, \end{aligned} \tag{2.1}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. Inequalities can be handled by introducing slack or surplus variables, transforming the inequalities into equalities. Upper bounds on z can be handled in a similar manner.

A quadratic program (QP) is an optimization problem involving a quadratic objective function, which can be written in the matrix form $\frac{1}{2} z^T Q z$, in addition to any linear objective function and linear constraints. SVM training optimization problems are of this type. Descriptions of interior point methods normally start with the LP formulation, but one of the

advantages of IPMs is the little modification they require to adapt to QPs and incorporate upper bounds, and so let us describe an interior point method for this type of problem directly.

We are interested in solving the general convex quadratic problem

$$\begin{aligned} \min_{z} \quad & \tfrac{1}{2} z^T Q z + c^T z \\ \text{s.t.} \quad & Az = b \\ & 0 \le z \le u, \end{aligned} \tag{2.2}$$

where $u \in \mathbb{R}^n$ is a vector of upper bounds, $Q \in \mathbb{R}^{n \times n}$ is a symmetric positive semi-definite matrix, and the constraint matrix $A \in \mathbb{R}^{m \times n}$ is assumed to have full rank. We need to introduce dual variables $s \in \mathbb{R}^n$ for the lower bound $z \ge 0$, $v \in \mathbb{R}^n$ for the upper bound $u - z \ge 0$, and $\lambda \in \mathbb{R}^m$ as Lagrange multipliers for the equality constraints. From Lagrangian duality, the corresponding dual problem is

$$\begin{aligned} \max_{z,s,v,\lambda} \quad & b^T \lambda - u^T v - \tfrac{1}{2} z^T Q z \\ \text{s.t.} \quad & A^T \lambda - Qz + s - v = c \\ & u - z \ge 0 \\ & z, s, v \ge 0. \end{aligned} \tag{2.3}$$

The optimal point $(z^*, s^*, v^*, \lambda^*)$ satisfies the Karush-Kuhn-Tucker (KKT) first and second order optimiality conditions (for a description of these conditions, see Fletcher, 1987, chapter 9). The first order optimality conditions for (2.2) are:

$$Az = b \tag{2.4a}$$

$$A^T \lambda + s - v - Qz = c \tag{2.4b}$$

$$ZSe = 0 \tag{2.4c}$$

$$(U - Z)Ve = 0 \tag{2.4d}$$

$$u - z \ge 0 \tag{2.4e}$$

$$z, s, v \ge 0, \tag{2.4f}$$

where e denotes the vector of all 1s. The first order conditions require primal feasibility (2.4a), dual feasibility (2.4b), and also *complementarity* between each bounded primal variable and the corresponding Lagrange multiplier for that bound: one of each pair (z_i, s_i) must

equal zero, either $z_i = 0$, or the dual variable for that lower bound, $s_i = 0$, or both; this is (2.4c). The same applies for upper bounds: either the variable z_i is at its upper bound, so $u_i - z_i = 0$, or the corresponding dual variable $v_i = 0$, or both (2.4d). Second order optimality conditions state that if the problem (2.2) has a convex quadratic objective function and linear constraints, the solution to (2.4) will be the global minimizer (Fletcher, 1987, Theorem 9.4.1). This is guaranteed if Q is positive semidefinite.

Due to the complementarity condition at the solution, we know that:

from (2.4c),	$z_i^* = 0$	and/or	$s_i^* = 0$	$\forall i \in \{1,\ldots,n\}$,
and from (2.4d),	$z_i^* = u_i$	and/or	$v_i^* = 0$	$\forall i \in \{1,\ldots,n\}$.

We can define a *basic* set $\mathscr{B}$, a *nonbasic* set $\mathscr{N}$, and an *upper-bounded* set $\mathscr{U}$ to all be subsets of the index set of variables, as follows:

$$\mathscr{B} = \{i \in \{1,\ldots,n\} : 0 < z_i^* < u_i\},$$
$$\mathscr{N} = \{i \in \{1,\ldots,n\} : s_i^* > 0\},$$
$$\mathscr{U} = \{i \in \{1,\ldots,n\} : v_i^* > 0\}.$$

It is clear that the sets $\mathscr{B}, \mathscr{N}$ and $\mathscr{U}$ must be disjoint, otherwise the complementarity conditions of (2.4) would be violated. Partitioning of the index set, where $\mathscr{B} \cup \mathscr{N} \cup \mathscr{U} = \{1,\ldots,n\}$, requires *strict complementarity*, where exactly one of each pair (z_i, s_i) equals zero, and similarly for the pairs $(u_i - z_i, v_i)$. Unlike the Simplex method where both variables in a pair can be zero, interior point methods will always generate a strictly complementary solution.

2.2 Outline of interior point method

The fundamental innovation of interior point methods is to keep the complementarity pairs (z, s), and similarly $(u - z, v)$, roughly in balance. Thus the partitioning of variables into the sets $\mathscr{B}$, $\mathscr{N}$ and $\mathscr{U}$ is delayed for as long as possible. The current iterate is forced into the "interior" of the bounded region by modifying the complementarity product equations (2.4c)

and (2.4d) to:

$$\begin{aligned} Az &= b \\ A^T\lambda + s - v - Qz &= c \\ ZSe &= \mu e \\ (U-Z)Ve &= \mu e \\ u - z &\geq 0 \\ z, s, v &\geq 0. \end{aligned} \tag{2.5}$$

where $\mu > 0$. This is equivalent to rewriting the primal problem with the objective function augmented by logarithmic barriers, a technique for handling constraints in nonlinear optimization (Fiacco and McCormick, 1968):

$$\begin{aligned} \min_z \quad & \tfrac{1}{2} z^T Q z + c^T z - \mu \sum_{j=1}^{n} \ln z_j - \mu \sum_{j=1}^{n} \ln(u_j - z_j) \\ \text{s.t.} \quad & Az = b \\ & 0 < z < u. \end{aligned}$$

The parameter μ defines a family of approximate solutions, which form a *central path* that converges to the optimal solution of the original QP (2.2) as $\mu \to 0$ (see Wright, 1997).

We can capture the perturbed first order optimality conditions (2.5) in the system $F(z, \lambda, s, v) = 0$, by defining a nonlinear mapping $F : \mathbb{R}^{3n+m} \to \mathbb{R}^{3n+m}$ as

$$F(z, \lambda, s, v) \equiv \begin{bmatrix} Az - b \\ A^T\lambda - Qz + s - v - c \\ ZSe - \mu e \\ (U-Z)Ve - \mu e \end{bmatrix}.$$

Newton's method is used to solve this system of nonlinear equations, by finding a step $(\Delta z, \Delta\lambda, \Delta s, \Delta v)$ that satisfies

$$F(z, \lambda, s, v) + \nabla F(z, \lambda, s, v)(\Delta z, \Delta\lambda, \Delta s, \Delta v)^T = 0,$$

which, simply by rearranging the above equation, is found by solving

$$\begin{bmatrix} A & 0 & 0 & 0 \\ -Q & A^T & I & -I \\ S & 0 & Z & 0 \\ -V & 0 & 0 & U-Z \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\lambda \\ \Delta s \\ \Delta v \end{bmatrix} = \begin{bmatrix} b - Az \\ c - A^T\lambda + Qz - s + v \\ \mu e - ZSe \\ \mu e - (U-Z)Ve \end{bmatrix}. \tag{2.6}$$

It should be noted that while the first two terms in F are linear, the remaining two are nonlinear, and thus the Newton method can only approximately solve system (2.5).

It is straightforward to see that if an initial point $(z^{(0)}, \lambda^{(0)}, s^{(0)}, v^{(0)})$ is feasible, then for any step size $\alpha \in (0,1]$, the point $(z^{(0)}, \lambda^{(0)}, s^{(0)}, v^{(0)}) + \alpha(\Delta z, \Delta\lambda, \Delta s, \Delta v)$ is also feasible. Indeed, for primal feasibility,

$$A(z^{(0)} + \alpha\Delta z) = Az^{(0)} + \alpha A\Delta z = b + \alpha(0) = b,$$

and for dual feasibility,

$$\begin{aligned} &A^T(\lambda^{(0)} + \alpha\Delta\lambda) + (s^{(0)} + \alpha\Delta s) - (v^{(0)} + \alpha\Delta v) - Q(z^{(0)} + \alpha\Delta z) \\ =&A^T\lambda^{(0)} + s^{(0)} - v^{(0)} - Qz^{(0)} + \alpha(A^T\Delta\lambda + \Delta s - \Delta v - Q\Delta z) \\ =&c + \alpha(0) \\ =&c. \end{aligned}$$

A *feasible IPM algorithm*, once started with a feasible initial point (i.e. one that satisfies the equations (2.4a) and (2.4b)), is therefore able to maintain feasibility throughout the solution process. However, it is not easy to find an initial point $(z^{(0)}, \lambda^{(0)}, s^{(0)}, v^{(0)})$ that satisfies all the conditions in (2.5). *Infeasible interior point method algorithms* introduce residuals into the system of linear equations

$$\begin{bmatrix} A & 0 & 0 & 0 \\ -Q & A^T & I & -I \\ S & 0 & Z & 0 \\ -V & 0 & 0 & U-Z \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\lambda \\ \Delta s \\ \Delta v \end{bmatrix} = \begin{bmatrix} r_b \\ r_c \\ \mu e - ZSe \\ \mu e - (U-Z)Ve \end{bmatrix}. \tag{2.7}$$

where

$$r_b = b - Ax,$$
$$\text{and} \qquad r_c = c - A^T\lambda + Qz - s + v,$$

which allow the errors associated with primal and dual infeasibility to be properly taken into account. These residuals displace the primal and dual constraint boundaries so that any initial point $(z^{(0)}, \lambda^{(0)}, s^{(0)}, v^{(0)})$ is again interior to the convex hull of constraints. The method requires only that $z^{(0)}, u - z^{(0)}, s^{(0)}$ and $v^{(0)}$ are positive.

We are now in a position to write out an outline infeasible interior point method algorithm, and this is shown in Algorithm 1. The residuals are reduced as the algorithm progresses, and generally they reduce faster than the reduction in μ. This is because the first two equations in the system (2.5) are linear, and the residuals in these two equations are reduced by a factor of $(1-\alpha)$ when the step α in the Newton direction is taken. A full step will reduce the residuals to zero.

Algorithm 1 Prototype infeasible IPM algorithm

Require: Initial point $(z^0, \lambda^0, s^0, v^0)$: $z^0 > 0,\ u - z^0 > 0,\ s^0 > 0,\ v^0 > 0$

1: $(z, \lambda, s, v) := (z^0, \lambda^0, s^0, v^0)$
2: $\mu := \frac{1}{2n}((z^0)^T s^0 + (u - z^0)^T v^0)$
3: **while** stopping criteria are not fulfilled **do**
4: Set new target by reducing μ
5: Solve system (2.7) to determine direction $(\Delta z, \Delta\lambda, \Delta s, \Delta v)$
6: Determine step size $\alpha > 0 : (z, u - z, s, v) + \alpha(\Delta z, -\Delta z, \Delta s, \Delta v) > 0$
7: Make step $(z, \lambda, s, v) := (z, \lambda, s, v) + \alpha(\Delta z, \Delta\lambda, \Delta s, \Delta v)$
8: **end while**
9: **return** (z, s, v, λ)

2.3 Practicalities

In this section we discuss the issues that make the algorithm efficient: the linear algebra approaches, choice of starting point, termination criteria, the handling of free variables, and the use of the predictor-corrector technique. Finally we draw these together into Algorithm 2 so as to use it as a reference in subsequent chapters.

There are other important practical aspects involved in producing an efficient infeasible IPM code, for instance the design of a neighbourhood to ensure convergence, the rate of reduction in μ, or the use of different step lengths in primal and dual space. We found it was not necessary to adapt these parts when specializing for the SVM training problem, and

our approach does not differ from standard practice (Andersen et al., 1996; Gondzio, 1995; Wright, 1997).

2.3.1 Linear algebra considerations

The major work of a single iteration consists of solving the set of linear equations (2.7). The practical efficiency of a solver is highly dependent on the linear algebra used in this stage. Through elimination of the variables Δs and Δv, using

$$\Delta s = Z^{-1}(\mu e - ZSe - S\Delta z) \tag{2.8a}$$

$$\Delta v = (U - Z)^{-1}(\mu e - (U - Z)Ve + V\Delta z), \tag{2.8b}$$

the system can be transformed into the smaller *augmented system equations*:

$$\begin{bmatrix} -(Q + \Theta^{-1}) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} \hat{r_c} \\ r_b \end{bmatrix}, \tag{2.9}$$

where

$$\Theta^{-1} \equiv Z^{-1}S + (U - Z)^{-1}V \tag{2.10a}$$

$$\hat{r_c} \equiv r_c + \left((U - Z)^{-1} - Z^{-1}\right)\mu e + s - v. \tag{2.10b}$$

The system can be further reduced by eliminating

$$\Delta z = \left(Q + \Theta^{-1}\right)^{-1}\left(A^T\Delta\lambda - \hat{r_c}\right) \tag{2.11}$$

from the augmented system, to give the *normal equations*

$$M\Delta\lambda = \hat{r_b}, \tag{2.12}$$

where

$$M \equiv A\left(Q + \Theta^{-1}\right)^{-1}A^T \in \mathbb{R}^{m\times m} \tag{2.13a}$$

$$\hat{r_b} \equiv r_b + A\left(Q + \Theta^{-1}\right)^{-1}\hat{r_c}. \tag{2.13b}$$

An advantage of this form is that the matrix M is positive definite, and a direct approach such as Cholesky decomposition can be applied to solve (2.12). This requires calculating and

then factorizing M, and these steps are the most computationally expensive operations of the algorithm.[1] The exact cost depends on the sparsity pattern of A and the ease with which $(Q+\Theta^{-1})$ can be inverted. Interior point methods are efficient for solving quadratic programs when the matrix Q is easily invertible. For instance, a diagonal Q takes $\mathcal{O}(n)$ operations to invert, while if the matrix is dense the time taken to invert Q is a prohibitive $\mathcal{O}(n^3)$—in such a case, it is advantageous to solve system (2.9). Once $(Q+\Theta^{-1})^{-1}$ is known, $\mathcal{O}(nm^2)$ operations are required to form M.

2.3.2 Starting point

Infeasible interior point methods will converge from any initial point, but practical experience has shown (and the theoretical results from Roos, 2006, show) that the initial point should be reasonably central (not close to a boundary) and the initial residuals should be small, otherwise a larger number of iterations will be required.

Mehrotra (1992) proposed a two-step approach to finding an initial point. The first step involves solving a pair of least-squares problems to produce a point, which aims to satisfy primal and dual feasibility:

$$\text{(P)} \quad \min_{z} \quad \tfrac{1}{2}\left(z^T z + (u-z)^T(u-z)\right) \quad \text{s.t.} \quad Az = b \tag{2.14a}$$

$$\text{(D)} \quad \min_{s,v,\lambda} \quad \tfrac{1}{2}\left(s^T s + v^T v\right) \quad \text{s.t.} \quad A^T\lambda + s - v = c \tag{2.14b}$$

This may result in some components of z, $u-z$, s or v being outside of the positive quadrant; the second step is to project these components inside.

The solution to (2.14) satisfies the following system of linear equations:

$$\text{(P)} \quad \begin{bmatrix} -2I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda' \end{bmatrix} = \begin{bmatrix} -u \\ b \end{bmatrix} \tag{2.15a}$$

$$\text{(D)} \quad \begin{bmatrix} -2I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} s \\ \lambda \end{bmatrix} = \begin{bmatrix} c \\ 0 \end{bmatrix}, \quad v = -s. \tag{2.15b}$$

$\lambda' \in \mathbb{R}^m$ is a vector of the same dimension as λ, but has no role in the subsequent optimization. The advantage of the above method is that the systems (2.15) are close in structure to the augmented system (2.9). By modifying the $(1,1)$ block of the matrices in (2.15) from $-2I$ to

[1] If A and Q are sparse, the sparsity pattern of M is constant through all iterations. IPM implementations generally also have an *analysis* phase, where heuristics are used to determine a permutation of M that will give a sparse Cholesky decomposition. This phase can also involve considerable computational effort.

$-(Q+2I)$, and setting $\Theta^{-1} = 2I$, it is possible to reuse the data storage structures that will be needed for (2.9). Solving these two systems requires one factorization and two backsolves, a computational cost similar to one IPM iteration.

2.3.3 Termination criteria

Interior point methods do not give an exact vertex solution, but approach the solution asymptotically. It is necessary to decide when to terminate, which is usually done when the first order optimality conditions (2.4) and the *duality gap* (the difference between the primal objective (2.2) and the dual objective (2.3)) are all met to within a relative tolerance ϵ:

$$\begin{aligned} \frac{\|Az-b\|}{1+\|b\|} &\leq \epsilon \\ \frac{\|A^T\lambda+s-v-Qz-c\|}{1+\|c\|} &\leq \epsilon \\ \frac{\|c^Tz-b^T\lambda+u^Tv+z^TQz\|}{1+\|b^T\lambda-u^Tv-\frac{1}{2}z^TQz\|} &\leq \epsilon \end{aligned} \tag{2.16}$$

The algorithm continues until both infeasibilities and the duality gap (which is proportional to μ) fall below required tolerances. An alternative termination criterion, based on the change in angle of the SVM hyperplane, is discussed in Section 6.1.4.

2.3.4 Predictor-corrector

Most primal-dual interior point method implementations incorporate the predictor-corrector algorithm proposed by Mehrotra (1992), which calculates a *corrector direction* to compensate for the error made when the nonlinear equations (2.4c) and (2.4d) are linearized (see Wright, 1997, Chapter 10, for a full description). The result is an iterate that is closer to the central path, at the cost of an additional backsolve involving matrix M.

Mehrotra's corrector can be applied multiple times, but practical experience has shown that there is little benefit after the initial corrector. A better approach in practice is that proposed by Gondzio (1996) and recently refined by Colombo and Gondzio (2008), which modifies Mehrotra's corrector direction for selected elements where the complementarity product is either very large or very small, in order to increase the size of the step. The worth of applying the corrector multiple times depends on the relative computational effort invovled in solving the system compared to calculating and factorizing M. This trade-off is discussed in Section 6.1.5.

2.3.5 Free variables

Variables that have no natural upper and lower bound are described as *free variables*. There are several methods for handling them:

1. Splitting the variables into positive and negative components to fit the standard form;
2. Setting lower and upper bounds that are guaranteed to be inactive;
3. Not applying any log barrier functions for these variables.

We discuss each of these approaches in turn. Let us assume that we have a QP with bounded variables z and free variables w.

$$\begin{aligned} \min_{w,z} \quad & \tfrac{1}{2} w^T Q_w w + \tfrac{1}{2} z^T Q_z z + c_w^T w + c_z^T z \\ \text{s.t.} \quad & A_w w + A_z z = b \\ & w \text{ free}, \\ & 0 \le z \le u. \end{aligned} \tag{2.17}$$

We can reformulate the problem (2.17) to be in the standard form by splitting the free variables w into positive and negative components w^+ and w^- where

$$w = w^+ - w^-, \qquad w^+, w^- \ge 0,$$

and using this substitution in (2.17). This is the technique employed in `HOPDM` (Gondzio, 1995), and has the advantage that no modification to the method is required. The normal equation matrix can be formulated in the usual way, where

$$A(Q+\Theta^{-1})^{-1}A^T \equiv \begin{bmatrix} A_w & -A_w & A_z \end{bmatrix} \begin{bmatrix} Q_w + \Theta_{w^+}^{-1} & -Q_w & \\ -Q_w & Q_w + \Theta_{w^-}^{-1} & \\ & & Q_z + \Theta_z^{-1} \end{bmatrix}^{-1} \begin{bmatrix} A_w^T \\ -A_w^T \\ A_z^T \end{bmatrix}.$$

This approach unfortunately can lead to numerical problems. Logically, one of each pair (w^+, w^-) should be zero, but the logarithmic barriers force both elements in the pair away from zero. With no opposing barrier, both components are free to grow large provided the difference between them remains correct. The resulting small pivot values of Θ^{-1} can dominate the corresponding elements of $(Q+\Theta^{-1})^{-1}$. A practical remedy, implemented in `HOPDM`, is to shift each element of the pair, by subtracting a suitable quantity, and reduce

them to a moderate size.

An approach that removes this difficulty, yet stays within the standard formulation, is to calculate maximum feasible bounds from the constraints, if the problem allows this to be done easily. This is the approach we adopted in Section 6.1.3. It is interesting to note that this is the opposite of the approach recommended by Mészáros (1998b), where bounds that can be proved to never be active due to other constraints are removed.

Alternatively, we can handle free variables more naturally, by forming KKT conditions to (2.17) directly. Variables w have no logarithmic barrier (Mészáros, 1998b), resulting in the modified KKT conditions:

$$\begin{aligned} A_w w + A_z z &= b \\ A_w^T \lambda - Q_w w &= c_w \\ A_z^T \lambda - Q_z z + s - v &= c_z \\ ZSe &= 0 \\ (U - Z)Ve &= 0 \\ u - z &\geq 0 \\ z, s, v &\geq 0 \end{aligned}$$

Note that block eliminations lead to the augmented system where there is no Θ_w term.

$$\begin{bmatrix} -Q_w & 0 & A_w^T \\ 0 & -(Q_z + \Theta_z^{-1}) & A_z^T \\ A_w & A_z & 0 \end{bmatrix} \begin{bmatrix} \Delta w \\ \Delta z \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} r_{c_w} \\ r_{c_z} \\ r_b \end{bmatrix}.$$

This is the technique we implemented using `OOPS` (Gondzio and Grothey, 2007), in Chapter 7. The linear algebra approach depends on Q_w. If it is positive definite, normal equations can be formed. If Q_w is only positive semi-definite, either regularization can be used (Altman and Gondzio, 1999) or the augmented matrix solved directly.

2.3.6 Presolve

Sparse matrix problems often benefit from a *presolving* stage: a process before the optimization algorithm which attempts to reduce the problem size, and thus simplify the problem. It eliminates constraints and columns through inspection, using techniques such as removing empty or singleton rows and columns, or replacing fixed variables with their value. See Andersen et al. (1996, section 5) for a full description. In the formulation for SVM training

presented in Chapter 5, the constraint matrix is almost always dense. We found the presolve operation provided no benefit, and so it was best to turn it off completely.

2.3.7 Reference IPM algorithm

Algorithm 2 is an infeasible IPM algorithm, using the techniques described above, that will be used as a reference algorithm in future chapters.

Algorithm 2 Reference infeasible interior point algorithm

1: (z, λ, s, v) := Starting point from Section 2.3.2
2: $\mu := \frac{1}{2n}(z^T s + (u-z)^T v)$
3: **while** termination criteria are not fulfilled **do**
4: Calculate matrix $M \equiv A(Q + \Theta^{-1})^{-1} A^T$
5: Factorize matrix M using Cholesky decomposition
6: Calculate search direction $(\Delta z, \Delta\lambda, \Delta s, \Delta v)$ by solving $M\Delta\lambda = -\hat{r}_b$ and backsolving for other variables
7: Determine step size $\alpha > 0 : (z, u-z, s, v) + \alpha(\Delta z, -\Delta z, \Delta s, \Delta v) > 0$
8: Make step $(z, \lambda, s, v) := (z, \lambda, s, v) + \alpha(\Delta z, \Delta\lambda, \Delta s, \Delta v)$
9: Use multiple correctors to modify (z, λ, s, v) to a more central point
10: Set new target by reducing μ
11: **end while**
12: **return** (z, λ, s, v)

Chapter 3

Support vector machines

There are some tasks related to classification, recognition and prediction for which we do not know how to write a computer program that can perform them. It is possible, though, to provide the computer with a large number of samples from which it could develop a discrimination method by itself. An example is the recognition of hand-written digits. This is the aim of machine learning: to find a general rule that explains data, given only a sample of limited size (Herbrich, 2002).

In *supervised learning*, the machine[1] is provided as an input a set of n training samples, where each sample $i = 1 \dots n$ is in the form of an attribute vector $x_i \in \mathbb{R}^m$, and with each sample is associated a target y_i which can either be a label or a value. For classification problems, the label assigns the sample to one class in a set of discrete classes. In regression, $y_i \in \mathbb{R}$ is a real-valued target.

The goal of classification is to take an input vector x, and from this information assign it to exactly one of the classes. The input space is divided into *decision regions*, and the boundaries that separate regions are *decision boundaries* or *decision surfaces*, determined by a real-valued function $f : X \subset \mathbb{R}^m \to \mathbb{R}$. For linear models of classification, which are the best understood of learning functions, f is a linear function of x and the decision surface is a hyperplane.

In general we will concentrate on binary classification, where there are exactly two classes. This is the simplest classification task. It is possible to extend the approach to other machine learning problems, some of which are described in Section 3.3. Discussion of how to extend the approaches to nonlinear functions is in Chapter 9.

[1] By *machine*, we mean a computer program designed to find this general rule from the limited sample of data, and subsequently make predictions based on it.

3.1 Linear binary classification

For some probabilistic models, it is most convenient to use a binary representation for the target value $y_i \in \{0,1\}$ (Bishop, 2006, chapter 4); but models such as support vector machines use a more symmetric approach based on sign, where $y_i \in \{-1,+1\}$. For the linear discriminant model, the predicted target value $f(x)$ is a linear function of the input vector x

$$f(x) = w^T x + w_0, \tag{3.1}$$

where $w \in \mathbb{R}^m$ is known as the *weight vector* and is the normal to the decision hyperplane, and $w_0 \in \mathbb{R}^1$ is called the *bias* (often denoted b in machine learning literature), or sometimes the *threshold* (θ, with the sign inverted, using conventions from statistics).

3.1.1 Hyperplane decision boundary

The geometric interpretation of linear binary classification is that the input space is split into the two class regions by the hyperplane decision boundary $f(x) = 0$, as shown in Figure 3.1. The vector w defines the direction perpendicular to the hyperplane—consider two points x_a and x_b that both lie in the hyperplane, then $w^T(x_a - x_b) = 0$. A change in w_0 causes a parallel shift in the hyperplane, and in total $m+1$ parameters are needed to define a hyperplane in $\mathbb{R}^m$.

There are two important measures of the *margin* γ_i, the perpendicular distance between a point x_i and the hyperplane:

- The *functional margin* uses the measure $y_i(w^T x_i + w_0)$, and it is always positive for a correctly classified point.
- The *geometric margin* measures the Euclidean distance from the point to the hyperplane, by compensating for $\|w\|_2$:

$$\gamma_i = \frac{1}{\|w\|_2} y_i(w^T x_i + w_0) \tag{3.2}$$

The *margin of the hyperplane* γ is the minimum geometric margin of all correctly classified points. A hyperplane that maximizes γ is known as the *maximal margin hyperplane*.

A useful insight into the geometric interpretation is to compare it with classification approaches based on hyperrectangles, such as the classification algorithm C4.5 (Quinlan, 1993), as shown in Figure 3.2. This algorithm builds a discriminant tree in a "divide and

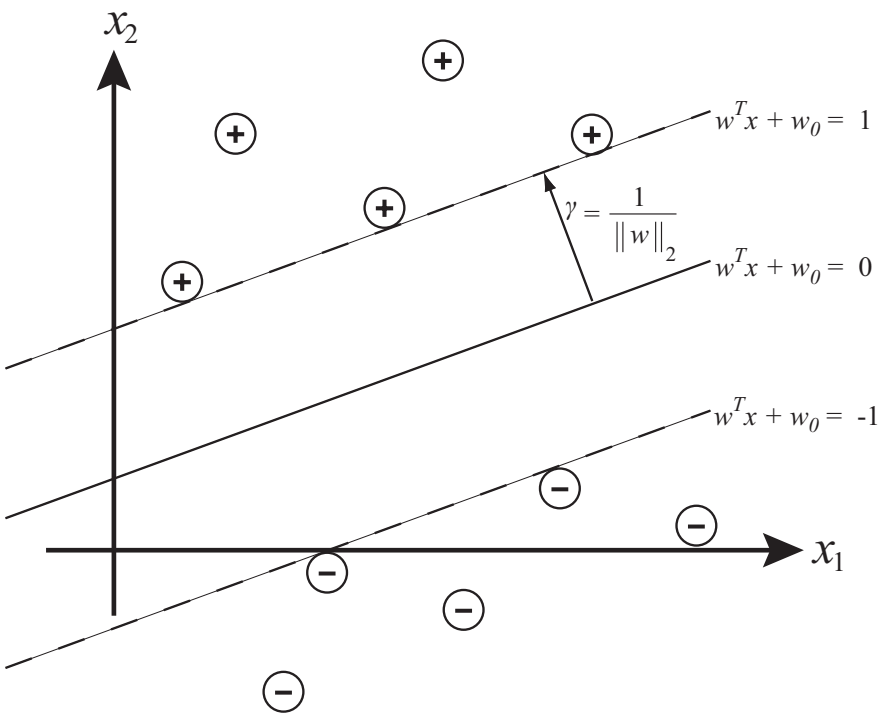


Figure 3.1: Separable data in two dimensions, showing the hyperplane decision boundary and the separating margin γ.

conquer" manner, by testing one attribute at a time within a hyperrectangle region and finding the boundary that gives the best discrimination. If the class regions are not naturally hyperrectanges, an increase in the number of training samples can lead to unlimited growth in the number of regions.

3.1.2 Rosenblatt's Perceptron

The Perceptron (Rosenblatt, 1958), occupies an important place in the history of pattern recognition algorithms, as it was the first iterative algorithm for learning linear classifications, and where the learning was mistake-driven. Other linear discriminants have since been developed, for instance based on Fisher's criterion, least squares error or logistic regression (see Webb, 2002). The Perceptron is interesting, however, as it anticipates many of the ideas used in support vector machines. It uses the linear discriminant function (3.1) and the decision boundary $f(x) = 0$ described above. It also uses ± 1 target values, which allows the inequality $y_i(w^T x_i + w_0) > 0$ to identify correctly classified points. Larger margins were shown to be in some sense better, and the extension to nonlinear feature mappings was also considered.

The Perceptron algorithm is shown as Algorithm 3. It starts with an initial weight vector

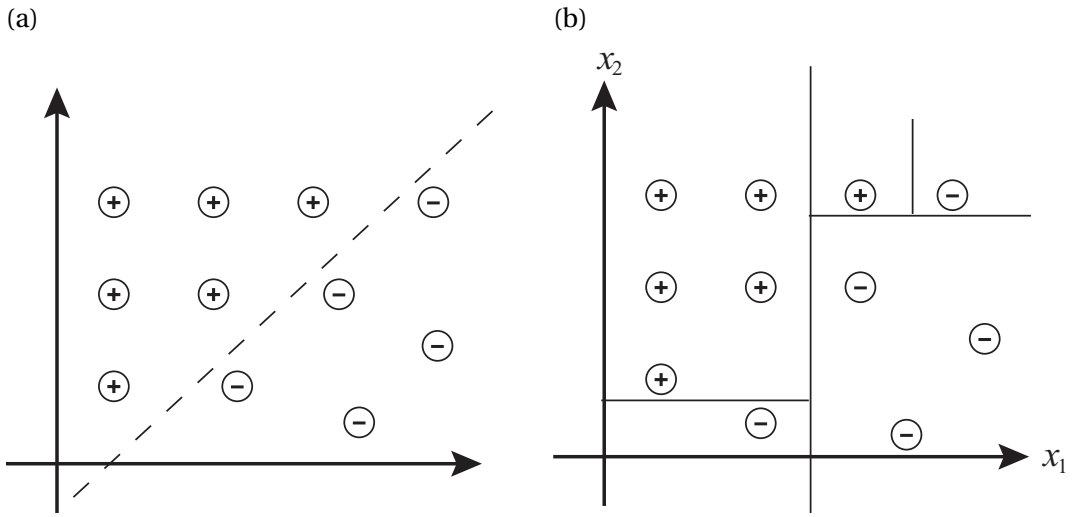


Figure 3.2: Comparison of the discriminant functions generated by (a) SVMs (hyperplane) and (b) the C4.5 algorithm (hyperrectangles) (Quinlan, 1993).

Algorithm 3 Rosenblatt's Perceptron algorithm

Require: Initial weights $w^{(0)} = 0,\ w_0^{(0)} = 0$
Require: Learning rate $\eta \in \mathbb{R}$
1: $(w, w_0) := (w^{(0)}, w_0^{(0)})$
2: **repeat**
3: **for** each sample $i = 1 \ldots n$ in the training set **do**
4: $$(w, w_0) := \begin{cases} (w, w_0) & \text{if } y_i(w^T x_i + w_0) \geq 0, \\ (w, w_0) + \eta y_i(x_i, 1) & \text{if } y_i(w^T x_i + w_0) < 0. \end{cases}$$
5: **end for**
6: **until** there are no misclassified samples
7: **return** (w, w_0)

$w^{(0)}$ and bias $w_0^{(0)}$, and updates them every time it encounters a point that is wrongly classified by the current discriminant function. The size of the updates is controlled by a learning rate parameter η. The algorithm terminates when it has minimized the misclassification error to zero. Rather than using the total number of misclassified samples as a measure of error, which involves discontinuities each time the decision hyperplane crosses a data point, Rosenblatt introduced a continuous error function known as the *perceptron criterion*:

$$E_P(w) \equiv \sum_{i=1}^{n} \max(-y_i(w_T x_i + w_0), 0). \tag{3.3}$$

If a sample is incorrectly classified, the weight vector is updated to be made closer to classifying the sample correctly, shown at line 4.

Data sets that can be separated into classes exactly by the decision function are *separable*[2]. One of the advantages of the Perceptron algorithm is that it will converge to a solution in a polynomial number of steps, provided the training set is linearly separable. Quite how many steps depends on the margin of the hyperplane γ. If w is a unit vector, and we define a radius r from the origin that encloses all samples, $r \equiv \max_{1\le i\le n} \|x_i\|_2$, then the number of mistakes made by the algorithm is at most $\left(\frac{2r}{\gamma}\right)^2$; for proof, see Novikoff (1963), Herbrich (2002, Appendix B.4) or Cristianini and Shawe-Taylor (2000, Theorem 2.3). Theoretical justification is given to the idea that a large margin is an indicator of a better classifier, if only in terms of the number of mistakes made during learning. Unfortunately there is no way of telling a nonseparable problem from slow convergence until the algorithm terminates.

3.1.3 Risk minimization

If it is impossible to separate the classes linearly, nonlinear discriminant functions may be able to do so. The Perceptron included the concept of linearizing non-linear function using the mapping $\phi(x)$ and the discriminant function $f(x) = w^T\phi(x) + w_0$, thereby increasing the complexity of the model by introducing more parameters. If the aim is solely to reduce the misclassification error, there is a danger of *overfitting*, which involves the machine memorizing the training set rather than learning from it (and so reducing the misclassification error to zero). This negatively affects the ability of the machine to *generalize* from the training set to unseen samples.

Support vector machines are a principled method to avoid overfitting, motivated by *sta-*

[2] Note that this is a different meaning of the word *separable* than that of Chapter 5, where separable refers to a function that is the sum of single-variable sub-functions.

tistical learning theory (Vapnik, 1998, 1999). This theory introduces the *Vapnik-Chervonenkis dimension*, which measures the complexity of functions. The key result is that the maximization of the confidence in an approximating function (which means minimizing its complexity) corresponds to maximization of the hyperplane separation margin.

This has to be balanced against minimizing the misclassification error (the empirical risk) of the approximation based on knowledge of the current data. Vapnik describes this balance as the *Structural Risk Minimization principle*.

3.2 Formulating the SVM optimization problem

In common with the description of binary linear classifiers above, a support vector machine is a classification learning machine that learns a mapping between the features and the target label of a set of data points known as the *training set*. It uses a hyperplane $w^T x + w_0 = 0$ to separate the data set and predict the class of further data points. The binary labels are represented by ± 1.

SVMs introduce a fixed functional half-margin of 1, so from (3.2) the geometric half-margin is $\frac{1}{\|w\|_2}$. Maximizing the margin is equivalent to minimizing $\|w\|_2$. A vector $\xi \in \mathbb{R}^n$ of misclassification errors is introduced to cope with the possibility of nonseparable data, where ξ_i measures the distance from the point x_i to the hyperplane margin (see Figure 3.3).

Using the Structural Risk Minimization principle, the objective of SVM training is a balance between maximising the separation margin between the two hyperplanes and minimising the misclassification error. Possible error functions are illustrated in Figure 3.4. It is standard practice to use the 2-norm for the margin and a 1-norm for misclassification errors, similar to the error function (3.3) used by the Perceptron. This generates the convex quadratic optimization problem:

$$\begin{aligned} \min_{w,w_0,\xi} \quad & \frac{1}{2} w^T w + \tau e^T \xi \\ \text{s.t.} \quad & Y(X^T w + w_0 e) \geq e - \xi \\ & \xi \geq 0 \end{aligned} \tag{3.4}$$

where X and Y are as described in Section 1.3, e is the vector of all ones, and τ is a positive constant that parameterises the problem.

Due to the convex nature of the problem, a Lagrangian function associated with (3.4) can

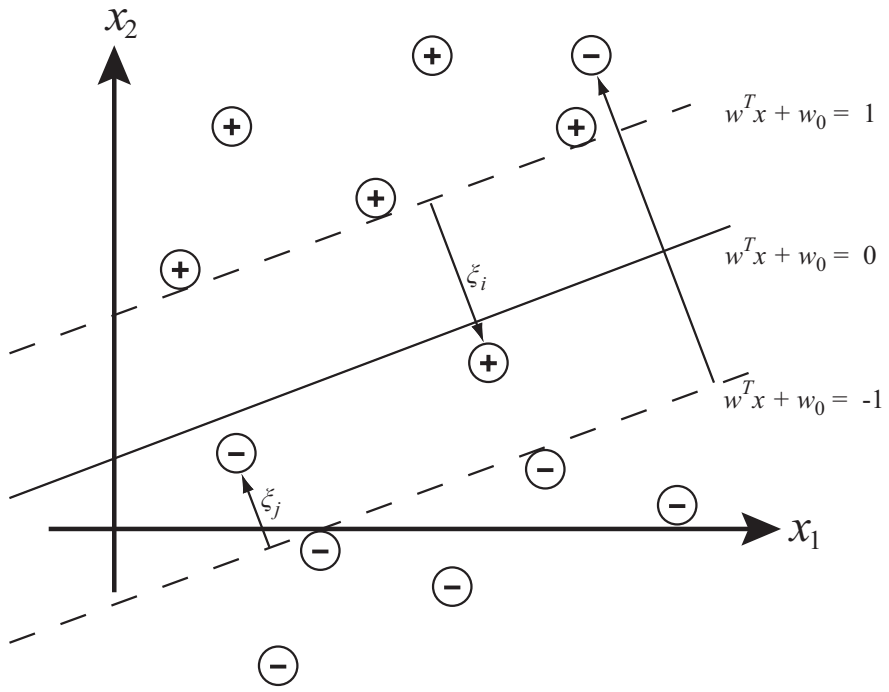


Figure 3.3: Non-separable data in two dimensions. Note that misclassification errors are included for both misclassified samples ξ_i and samples within the margin ξ_j.

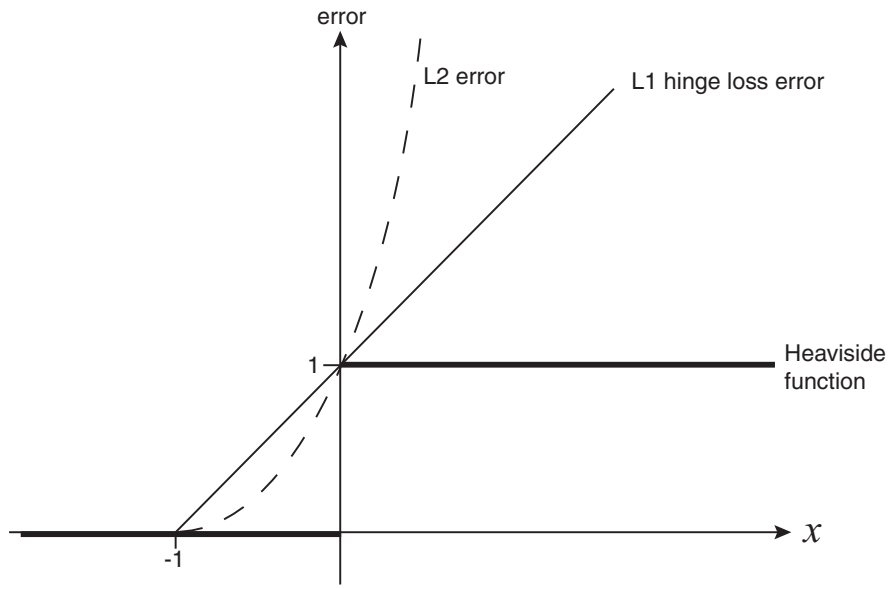


Figure 3.4: The L1 and L2 convex misclassification error functions approximate the Heaviside function, which is discontinuous, non-differentiable and non-convex. The L2 error function is continuous and differentiable, but penalizes outliers much more than the Heaviside function. The L1 hinge loss is the Perceptron error function (3.3) shifted along the x-axis.

be formulated,

$$\mathscr{L}(w, w_0, \xi, z, \zeta) = \frac{1}{2} w^T w + \tau e^T \xi - \sum_{i=1}^{n} z_i \left(y_i (w^T x_i + w_0) - 1 + \xi_i \right) - \zeta^T \xi$$

where $z \in \mathbb{R}^n$ is the vector of Lagrange multipliers associated with the inequality constraints that describe the hyperplane separating the points, and $\zeta \in \mathbb{R}^n$ is the vector of Lagrange multipliers associated with the non-negativity constraint on ξ. The solution to (3.4) will be at the saddle point of the Lagrangian. Partially differentiating the Lagrangian function then gives relationships between the primal variables w, w_0 and ξ, and the dual variables z at optimality:

$$w = XYz \tag{3.5a}$$

$$y^T z = 0 \tag{3.5b}$$

$$0 \le z \le \tau e. \tag{3.5c}$$

Substituting these relationships back into the Lagrangian function gives the dual problem formulation

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Y X^T X Y z - e^T z \\ \text{s.t.} \quad & y^T z = 0 \\ & 0 \le z \le \tau e. \end{aligned} \tag{3.6}$$

3.3 Extensions

The SVM framework has been extended to a wide range of machine learning problems. In this section we briefly introduce the ones that we will take further in this thesis.

3.3.1 ν-SVM

The ν-SVM, proposed by Schölkopf et al. (2000), is a reformulation of the standard binary classification where the penalty parameter τ for misclassifications is replaced by a parameter $\nu \in (0, 1]$ and a variable functional margin $\rho \in \mathbb{R}$. The primal problem is modified to:

$$\begin{aligned} \min_{w,\xi,\rho} \quad & \frac{1}{2} w^T w - \nu\rho + \frac{1}{n} e^T \xi \\ \text{s.t.} \quad & Y(X^T w + w_0 e) \ge \rho e - \xi \\ & \xi \ge 0, \quad \rho \ge 0 \end{aligned} \tag{3.7}$$

ν has an easier interpretation than τ: it can be seen as an upper bound on the fraction of training set samples that are in margin error ($\xi_i > 0$), and a lower bound on the fraction of samples that are support vectors ($z_i > 0$). Forming the Lagrangian in the usual way, the dual formulation becomes:

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Y X^T X Y z \\ \text{s.t.} \quad & y^T z = 0 \\ & e^T z \geq \nu \\ & 0 \leq z \leq \frac{1}{n} e. \end{aligned} \tag{3.8}$$

There is a direct equivalence between $\nu-$SVM and the standard SVM described in Section 3.2: If (3.8) has a solution where $\rho > 0$, then setting $\tau = \frac{1}{n\rho}$ in (3.6) will give an equivalent (although rescaled) hyperplane.

3.3.2 L2-SVM

The use of the 2-norm of w in the QP's objective can be justified in that it captures the Euclidean distance of the geometric margin. The use of the 1-norm for the misclassification penalty vector ξ, however, is more arbitrary. It has also become standard to consider using the 2-norm for ξ (shown in Figure 3.4), giving the primal objective:

$$\begin{aligned} \min_{w,\xi} \quad & \frac{1}{2} w^T w + \frac{\tau}{2} \xi^T \xi \\ \text{s.t.} \quad & Y(X^T w + w_0 e) \geq e - \xi \\ & \xi \geq 0. \end{aligned}$$

Thus the dual formulation becomes:

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T (Y X^T X Y + \frac{1}{\tau} I) z - e^T z \\ \text{s.t.} \quad & y^T z = 0 \\ & 0 \leq z \leq \tau e. \end{aligned} \tag{3.9}$$

The relationship (3.5a) holds for the L2 classification problem.

3.3.3 Universum

An approach to binary classification was proposed (Weston et al., 2006) where the problem is augmented with an additional data set belonging to the same domain (but not the same

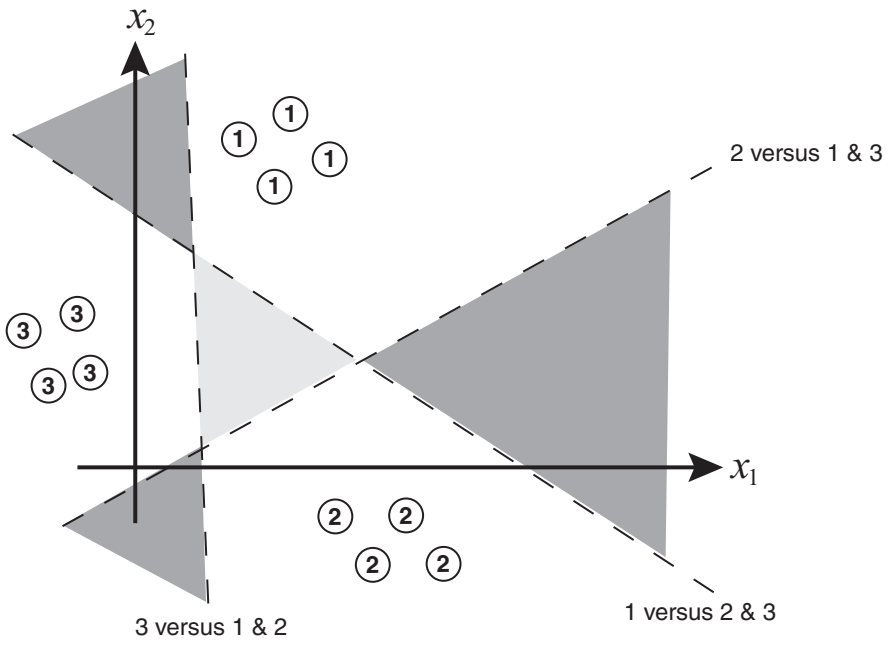


Figure 3.5: Multiple binary SVMs classifying three classes, using the one-versus-the-rest approach. The light grey area is unclassified, while the dark grey areas are claimed by two of the three classifiers.

classes) as the classification data. This additional data is called the Universum (Vapnik, 2006), as intuitively it captures a general backdrop. The SVM is trained to label points from the distributions of the binary classification sets, but make no strong statement for the Universum distribution.

3.3.4 Multi-class classification

It is possible to extend SVM binary classification to the more general case of k classes, by repeatedly applying binary classifications:

1. The *one-versus-the-rest* approach, proposed by Vapnik (1998), uses $k-1$ classifiers each of which solves a binary problem of separating the class against everything else.

2. The alternative *one-versus-one* classifier trains binary classifiers for every possible pair of classes. Majority voting by the classifiers is used to determine the class of a test sample.

Neither approach is perfect. For instance, both can produce ambiguous regions in the input space, as shown in Figure 3.5. The first approach suffers from imbalanced data, while the second generates many training problems.

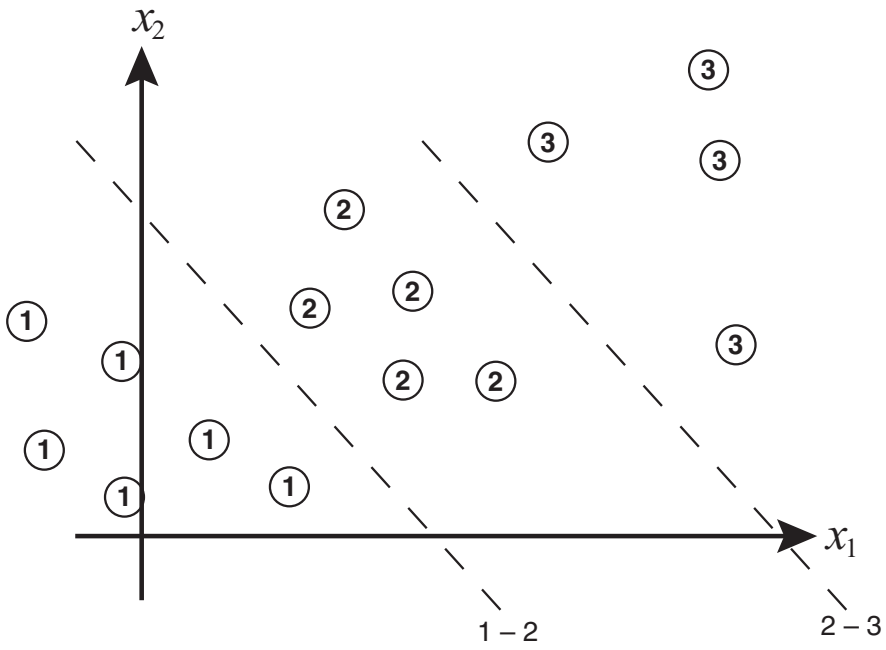


Figure 3.6: Ordinal regression uses parallel and ordered hyperplanes.

3.3.5 Ordinal regression

Ordinal regression refers to a learning technique that bridges multi-class classification and metric regression. Training samples are labelled with an ordinal number; in other words the classification categories are ranked. The task of the supervised learning problem is to predict the position of new samples on the ordinal scale. Unlike metric regression problems, the label numbers are discrete and the metric distances between labels do not have any real significance. Unlike multiple classification problems, order information is present. The approaches proposed by Chu and Keerthi (2005) involve $k-1$ parallel hyperplanes to separate each class from the next, reducing the training problem to a single QP (Figure 3.6).

3.3.6 Regression

Support Vector Regression uses similar techniques to SVMs in order to learn a mapping between the input vector x and a real-valued target value y. Rather than lying the correct side of the hyperplane margin, correctly determined samples should lie *within* the margin, now of half-width ϵ, shown in Figure 3.7.

3.3.7 Nonlinear discriminant functions and kernels

Kernel-based estimation techniques, such as support vector machines, have been shown to be powerful nonlinear classification methods. These techniques build a linear model

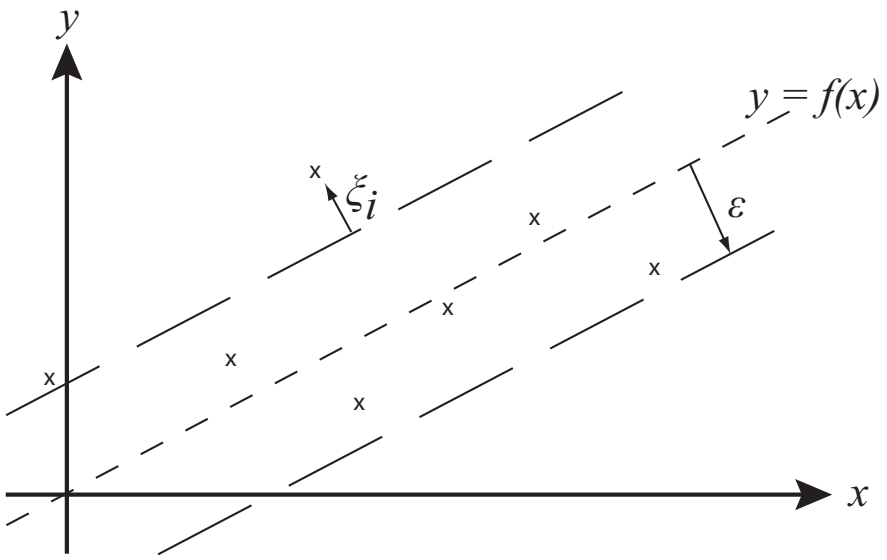


Figure 3.7: Support vector regression penalizes points ξ_i that lie outside a hyperplane margin of half-width ϵ.

in *feature space*, where the input attribute vectors x have been transformed by means of a (possibly infinite dimensional) nonlinear mapping $x \rightarrow \Phi(x)$. The discriminant function that defines the decision boundary is modified from (3.1) to a nonlinear form:

$$f(x) = w^T \Phi(x) + w_0,$$

which enables the data points to be separated by a nonlinear curve or by clustering (Figure 3.8).

It may be possible to generate nonlinear discriminants by defining the feature mappings explicitly, but normal practice is to construct a kernel function $K(x_i, x_j) \equiv \Phi(x_i)^T \Phi(x_j)$ by means of Mercer's theorem that identifies the feature space implicitly through inner products (Cristianini and Shawe-Taylor, 2000; Schölkopf and Smola, 2002). Some "general-purpose" kernels provided by Vapnik (1998) are:

$$\text{Polynomial} \qquad K(x_i, x_j) = \left(x_i^T x_j + 1\right)^d \tag{3.10a}$$

$$\text{RBF} \qquad K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2} \tag{3.10b}$$

$$\text{Sigmoid} \qquad K(x_i, x_j) = \left(1 + e^{\nu x_i^T x_j - c}\right)^{-1} \tag{3.10c}$$

but it is also possible to construct more specialized kernels (Shawe-Taylor and Cristianini, 2004).

For some kernels it is possible to determine Φ. For instance, the polynomial kernel (3.10a)

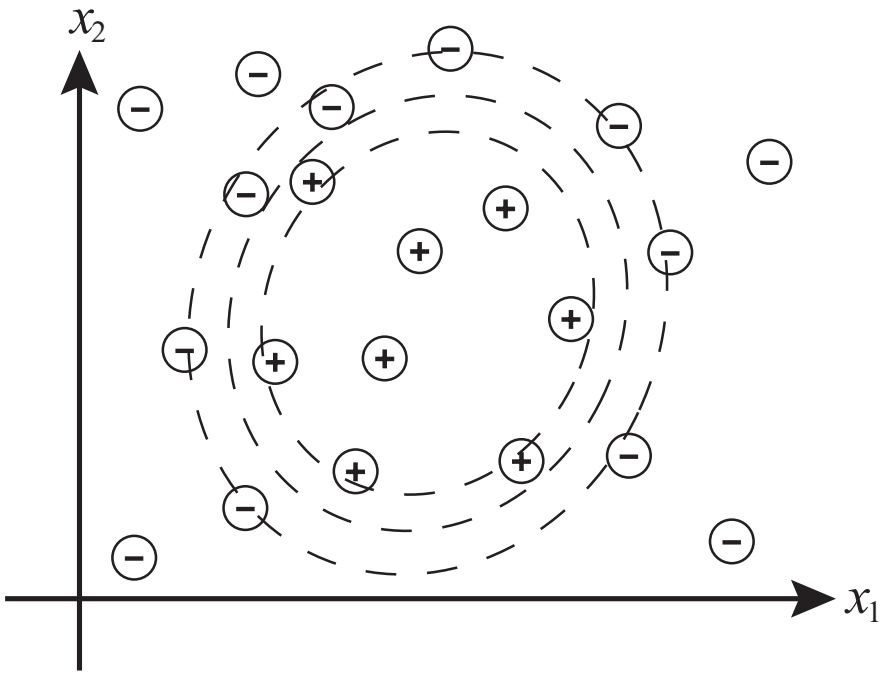


Figure 3.8: Two classes separable by a polynomial function of degree 2.

with $d = 2$ can be described by the mapping $\Phi : \mathbb{R}^m \to \mathbb{R}^{m_p}$, where m is the number of input features, $v \in \mathbb{R}^m$ is an input vector and $m_p = \binom{m+2}{2}$:

$$\Phi(v) = [\Phi_1(v), \Phi_2(v), \Phi_3(v), 1]^T \tag{3.11a}$$

$$\Phi_1(v) = [v_1, \ldots, v_m] \tag{3.11b}$$

$$\Phi_2(v) = [v_1 v_2, \ldots, v_1 v_m, v_2 v_3, \ldots, v_2 v_m, \ldots, v_{m-1} v_m] \tag{3.11c}$$

$$\Phi_3(v) = [v_1^2, \ldots, v_m^2] \tag{3.11d}$$

If the number of input features m is small, it may still be practical to use an explicit mapping and apply a linear classifier (Gertz and Griffin, 2009; Jung et al., 2008). Generally though the kernel approach is preferred. Along with providing an implicit data representation in some linear space, the kernel function also implies the function class that will be used in the learning process. Additionally, in some sense the kernel function is a measure of pair-wise similarity between samples.

The full set of pairwise similarity measures can be represented by a kernel matrix $K \in \mathbb{R}^{n \times n}$,

where each element is given by $K_{ij} = K(x_i, x_j)$. The dual formulation (3.6) becomes

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Y K Y z - e^T z \\ \text{s.t.} \quad & y^T z = 0 \\ & 0 \le z \le \tau e \,. \end{aligned} \tag{3.12}$$

Because it incorporates the kernel function, the dual formulation (3.12) is the preferred optimization problem for nonlinear SVMs.

Chapter 4

Previous approaches to SVM training

It is not practical to use general-purpose solvers for training SVMs, since, using the notation of (2.2), the solver expects to be provided with (A, Q, b, c, u) explicitly. Both primal and dual formulations lead to very demanding computational and memory requirements. Much research has gone into finding more practical approaches, including reducing the size of the data set to a representative sample (e.g., Lee and Mangasarian, 2001; Evgeniou and Pontil, 2002), or identifying more meaningful linear combinations of features (e.g., Keerthi et al., 2006). In this chapter we concentrate on numerical optimization methods. The performance of many of the implementations described here is discussed in Section 6.2 and Section 6.3.

4.1 Active set methods

The standard approach to training SVMs is to build a solution through the solving of a sequence of small scale problems. *Chunking* (see Vapnik, 1998, Section 12.1.1) removes rows and columns corresponding to zero Lagrange multipliers, but this still results in a large QP. Decomposition methods (Osuna et al., 1997; Lucidi et al., 2007) solve a series of smaller, fixed-size QPs, until none of the KKT conditions is violated. The technique can be applied to an arbitrarily large data set. State-of-the-art software $\texttt{SVM}^{\texttt{light}}$ (Joachims, 1999) uses this technique, with an IPM solver for subproblems.

Sequential Minimal Optimization (Platt, 1999) takes the concept of decomposition to an extreme and reduces the size of the working set to just two variables. The QP subproblem can then be solved analytically. Heuristics are used to select which variables should enter the working set. Due to the simplicity of the optimization algorithm, and good scaling properties in practice, this technique has been implemented in many SVM software packages, for

example SVMTorch (Collobert and Bengio, 2001), LIBSVM (Chang and Lin, 2001), Milde (Durdanovic et al., 2007), LaRank (Bordes et al., 2005).

These are all fundamentally active-set techniques, which work well when the separation into active and non-active variables is clear, or in other words when the data is separable by a hyperplane. With noisy data, the set of support vectors is not so clear, and the performance of these algorithms deteriorates. The Incremental Active Set (INCAS) method (Fine and Scheinberg, 2002) is an approach where variables change set one at a time, which makes it better able to handle noisy data.

4.2 Gradient projection algorithms

The simple structure of the constraints in dual formulation (3.6) makes gradient projection algorithms a promising approach. These methods involve successive projections on the feasible region, which are not expensive. Serafini et al. (2005) implemented two projection gradient algorithms with variable step lengths for medium-scale SVM problems. Using it as a solver of subproblems within a decomposition technique, they reported better performance than $\text{SVM}^{\text{light}}$ (Joachims, 1999) using pr-LOQO. Dai and Fletcher (2006) proposed an algorithm based on secant approximation, gradient projection and adaptive non-monotone line search, for singly linearly constrained quadratic programs such as (3.6). This algorithm was incorported by Zanni et al. (2006) into a parallel solver framework.

4.3 Cutting-plane algorithms

Other optimization techniques have also been tried. In the software SVM^{perf} (Joachims, 2006), an equivalent reformulation was developed with fewer variables and thus suited to cutting-plane algorithms, where time to converge is linear in the size of the training set. The idea is that the misclassification risk term $e^T\xi$ in (3.4) can be sufficiently well represented by only a small number of terms (much less than n) in a piece-wise linear approximation. OCAS (Franc and Sonnenburg, 2008) improves the method by selecting the next cutting plane nearer the optimum of the reduced master problem. Smola et al. (2008) generalize the use of subgradients to a wider range of machine learning problems. These approaches are limited to linear SVMs.

4.4 Interior point methods

A straightforward implementation of the standard SVM dual formulation using IPM requires the inversion of the dense $n \times n$ matrix Q. Doing this step directly involves $\mathcal{O}(n^3)$ operations, making this approach unusable for anything but the smallest problems. However, several approaches using IPM technology and based on different formulations have been researched (Ferris and Munson, 2003; Gertz and Griffin, 2009; Goldfarb and Scheinberg, 2005; Chang et al., 2008). They all have in common an aim to exploit the low-rank structure of the kernel matrix and reduce the problem to one where the only matrix to be inverted has dimension of order $m \times m$, where m is the number of features. This gives a per-iteration computational complexity of $\mathcal{O}(nm^2)$, a significant improvement if $n \gg m$.

A common approach is to use low-rank corrections in the representation of the Newton system, and exploit it through implicit inverse representation by applying the Sherman-Morrison-Woodbury (SMW) formula (Golub and van Loan, 1983, Section 2.1.3). An algorithm based on the dual formulation (3.6) and applying the SMW formula has a computational complexity of $\mathcal{O}(nm^2)$ for the multiplications and $\mathcal{O}(m^3)$ for the inversion at each IPM iteration (Ferris and Munson, 2003); a similar approach working in primal space has the same complexity (Gertz and Griffin, 2009).

The SMW formula has been widely used in interior point methods, where it often runs into numerical difficulties. There are two main causes of the difficulties: if the matrix Θ^{-1} to be inverted is ill-conditioned; and if there is near-degeneracy in the data matrix (XY). Ill-conditioning of the scaling matrix Θ^{-1} is a feature of IPMs, especially in the later iterations. Near-degeneracy in (XY) will occur if there are multiple data points which lie along or close to the separating hyperplanes, and this is accentuated if the data are not well scaled. Neither of these problems can really be avoided by a SMW-based algorithm. In (Goldfarb and Scheinberg, 2005), data sets of this type were constructed where an SMW-based algorithm required many more iterations to terminate, and in some cases stalled before achieving an accurate solution. The authors also showed that this situation arises in real-world data sets.

Goldfarb and Scheinberg (2005) proposed an alternative technique based on Product Form Cholesky Factorization, implemented for SVMs by Fine and Scheinberg (2001). In this technique, a Cholesky factorization is computed for a very sparse matrix and then updated to take into account each of the $m+1$ dense columns of A. The approach has the same complexity $\mathcal{O}(nm^2)$ as the previous approaches (although a small multiple of flops are required), but better numerical properties: LDL^T Cholesky factorization of the IPM normal

equation matrix with variables following the central path has the property that L remains numerically stable despite D becoming increasingly ill-conditioned, which happens in the later iterations of IPM algorithms (Goldfarb and Scheinberg, 2004). Although the authors exploit symmetries in their technique to reduce the computations required, their approach suffers from some memory caching inefficiencies because each feature is handled separately (this is investigated in Section 6.2). It is also intrinsically sequential, and so does not facilitate a parallel computing implementation.

4.5 Modifying the optimization problem

There have been several approaches proposed to modify the optimization problems into ones that are faster or easier to solve, but that provide learning machines close to SVMs.

The Robust Linear Program (Bennett and Mangasarian, 1992) pre-dates SVMs and anticipates many of the key ideas. For the discriminant function, it finds the hyperplane that maximizes 1-norm distances to the data, which results in a LP formulation. Soft margins are introduced to handle nonseparable data. Bradley and Mangasarian (2000) apply the method to large data sets, and Sra (2006) developed an active-set approach to the same formulation.

The Successive Overrelaxation approach of Mangasarian and Musicant (1999) uses quadratic programming, but modifies the objective of (3.4) to include a w_0 term that minimizes the location of the hyperplane relative to the origin:

$$\begin{aligned} \min_{w,w_0,\xi} \quad & \frac{1}{2}(w^T w + w_0^2) + \tau e^T \xi \\ \text{s.t.} \quad & Y(X^T w + w_0 e) \geq e - \xi \\ & \xi \geq 0. \end{aligned} \tag{4.1}$$

The motivation for doing so is that in the dual formulation, compared to (3.6), the linear constraint no longer exists:

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Y (X^T X + e e^T) Y z - e^T z \\ \text{s.t.} \quad & 0 \leq z \leq \tau e. \end{aligned} \tag{4.2}$$

Without the complicating constraint, the authors propose to solve the problem using very small subproblems and an implicit inverse of the Hessian matrix.

In (Mangasarian and Musicant, 2000), the authors combine this modification with the

2-norm for the misclassification error vector ξ, as in (3.3.2), a continuously differentiable function, as the nonnegativity constraint on ξ is no longer needed:

$$\begin{aligned} \min_{w,w_0,\xi} \quad & \frac{1}{2}(w^T w + w_0^2) + \frac{\tau}{2}\xi^T \xi \\ \text{s.t.} \quad & Y(X^T w + w_0 e) \geq e - \xi \end{aligned}$$

Compared to (3.6) for standard SVMs, the corresponding dual problem (4.3) has a Hessian matrix in the objective function that is positive definite, no equality constraint and no upper bound on z. The only constraint present is a simple nonnegativity one.

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Y (X^T X + e^T e) Y z - e^T z \\ \text{s.t.} \quad & z \geq 0. \end{aligned} \tag{4.3}$$

This simplified optimization problem, suitable for unconstrained optimization techniques, has been used as the basis of an active set approach (Mangasarian and Musicant, 2000), a finite Newton method (Keerthi and DeCoste, 2005), and a coordinate descent method (Hsieh et al., 2008) which is implemented in the LIBLINEAR software package (Fan et al., 2008). For its L1-SVM, LIBLINEAR implements a slightly modified version of (4.2).

Chapter 5

Separable formulation for linear SVMs

In this chapter, we present a family of efficient and numerically stable IPM-based formulations which unify from an optimization perspective 1-norm classification, 2-norm classification, universum classification, ordinal regression and ϵ-insensitive regression. We show that all these problems can be equivalently reformulated as very large, yet structured, separable QPs. By *separable* we mean that the objective can be expressed as a sum of single-variable functions,

$$f(x) = \sum_{i=1}^{n} f_i(x_i),$$

even if this is not possible for the constraints. Exploiting separability has been investigated for general sparse convex QPs (Vanderbei, 1997; Mészáros, 1998a), but not for the SVM problem. We take the implementation of this formulation further in subsequent chapters.

By showing how optimality conditions between the primal weight variables $w \in \mathbb{R}^m$ and the dual variables $z \in \mathbb{R}^n$ can be used to derive equivalent formulations, we present a new, unified approach for SVM training that combines the separability of the primal formulation with the small number of constraints of the dual formulation. As will be seen, our separable formulations introduce m additional variables and constraints to the standard dual problems, but such an approach enables an IPM algorithm with a complexity that is linear in the data set size.

5.1 Classification

The optimization problems for linear binary classification were described in Section 3.2. Using the form $Q = (XY)^T(XY)$ enabled by the linear kernel, we can rewrite the quadratic objective in terms of w, and ensure the relationship (3.5a) between w and z holds at opti-

mality by introducing it into the constraints. Consequently, we can state the classification problem (3.6) as the following separable QP:

$$\min_{w,z} \quad \frac{1}{2} w^T w - e^T z \tag{5.1a}$$

$$\text{s.t.} \quad w - XYz = 0 \tag{5.1b}$$

$$-y^T z = 0 \tag{5.1c}$$

$$0 \le z \le \tau e. \tag{5.1d}$$

Variables of this problem form a tuple $(w, z) \in \mathbb{R}^{m+n}$, the matrix of the quadratic form in the objective is no longer dense, but simplified to the diagonal matrix

$$Q = \begin{bmatrix} I_m & 0 \\ 0 & 0_n \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}$$

while the constraint matrix is in the form:

$$A = \begin{bmatrix} I_m & -XY \\ 0 & -y^T \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+n)}.$$

Determining the Newton step requires calculating the matrix product:

$$\begin{aligned} M &\equiv A(Q + \Theta^{-1})^{-1} A^T \\ &= \begin{bmatrix} (I_m + \Theta_w^{-1})^{-1} + XY\Theta_z YX^T & XY\Theta_z y \\ y^T \Theta_z YX^T & y^T \Theta_z y \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}. \end{aligned} \tag{5.2}$$

We need to solve $A(Q + \Theta^{-1})^{-1} A^T \Delta\lambda = \hat{r}_b$ for $\Delta\lambda$ (2.12). Building the matrix (5.2) is the most expensive operation (step 4 of Algorithm 2), of order $\mathcal{O}(n(m+1)^2)$, while inverting the resulting matrix (step 5) is of order $\mathcal{O}((m+1)^3)$.

Gertz and Wright (2003) developed a near-equivalent formulation for M, through successive block eliminations of the general form of the QP, as part of their OOQP software. Their software appears to have received little attention from the machine learning community.

To determine the hyperplane, we also require the value of the bias w_0, a variable in the primal problem (3.4). Note that the element of λ corresponding to the constraint $-y^T z = 0$ is in fact the variable w_0. Using our approach and a primal-dual interior point method, we can obtain w_0 directly from the optimal solution found by the solver. This is in contrast to active-set methods, where w_0 has to be estimated from a subset of z values.

5.2 2-norm classification

We use the same technique to develop a formulation for the 2-norm SVM (3.9), leading to a separable QP with the following diagonal Hessian matrix Q:

$$Q = \begin{bmatrix} I_m & 0 \\ 0 & \frac{1}{\tau} I_n \end{bmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}. \tag{5.3}$$

5.3 ν-SVM

Similar techniques can be applied to the ν-SVM, by introducing into (3.8) a slack variable $s_\nu \in \mathbb{R}$ for the inequality constraint and rewriting it as the following separable QP:

$$\begin{aligned} \min_{w,s_\nu,z} \quad & \frac{1}{2} w^T w - e^T z \\ \text{s.t.} \quad & w - XYz = 0 \\ & e^T z - s_\nu = \nu \\ & -y^T z = 0 \\ & 0 \le z \le \tau e, \quad s_\nu \ge 0. \end{aligned} \tag{5.4}$$

Compared to (5.1) for normal L1 classification, an extra variable and constraint have been added but the same separable structure is present. With the variables ordered (w, s_ν, z), the quadratic objective is determined by the following diagonal matrix:

$$Q = \begin{bmatrix} I_m & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0_n \end{bmatrix} \in \mathbb{R}^{(m+1+n)\times(m+1+n)}$$

while the constraint matrix is in the form:

$$A = \begin{bmatrix} I_m & 0 & -XY \\ 0 & -1 & e^T \\ 0 & 0 & -y^T \end{bmatrix} \in \mathbb{R}^{(m+2)\times(m+1+n)}.$$

5.4 Universum SVM

Introduced in Section 3.3.3, Universum SVM classification Weston et al. (2006) is an approach to binary classification where an additional data set, called the Universum (Vapnik, 2006),

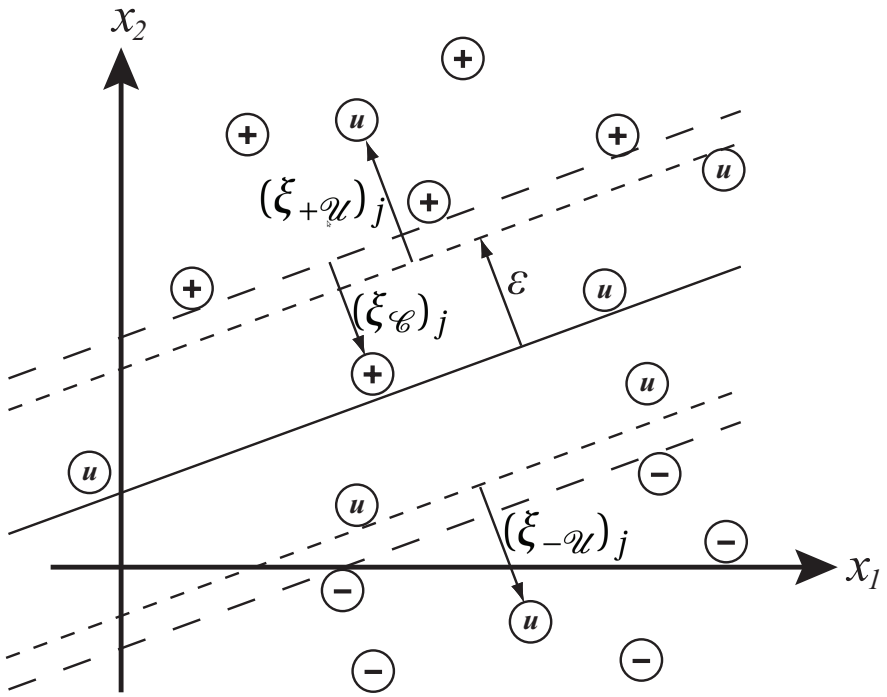


Figure 5.1: Binary classification including Universum data.

captures a general backdrop. The SVM is trained to label points from the distributions of the binary classification sets, but make no strong statement for the Universum distribution.

Let $\mathscr{C}$ be the set of classification points, with data $X_{\mathscr{C}}$ and labels $y_{\mathscr{C}}$. Similarly, let $\mathscr{U}$ be the Universum set with data $X_{\mathscr{U}}$ and no labels. As with normal binary classification, data points are penalized for being on the wrong side of the hyperplane margin, measured by error $\xi_{\mathscr{C}} \in \mathbb{R}^{|\mathscr{C}|}$. Samples in the Universum set should lie close to the hyperplane; $\xi_{+\mathscr{U}} \in \mathbb{R}^{|\mathscr{U}|}$ and $\xi_{-\mathscr{U}} \in \mathbb{R}^{|\mathscr{U}|}$ are the errors if they are more than ϵ above or below the hyperplane (Figure 5.1). It is possible to use different parameters for misclassification errors in the two sets, here shown as $\tau_{\mathscr{C}}$ and $\tau_{\mathscr{U}}$. The primal formulation is then:

$$\begin{aligned} \min_{w,w_0,\xi_{\mathscr{C}},\xi_{+\mathscr{U}},\xi_{-\mathscr{U}}} \quad & \frac{1}{2} w^T w + \tau_{\mathscr{C}} e^T \xi_{\mathscr{C}} + \tau_{\mathscr{U}} \left(e^T \xi_{+\mathscr{U}} + e^T \xi_{-\mathscr{U}} \right) \\ \text{s.t.} \quad & Y_{\mathscr{C}} \left(X_{\mathscr{C}}^T w + w_0 e \right) \geq e - \xi_{\mathscr{C}} \\ & X_{\mathscr{U}}^T w + w_0 e \geq \epsilon e - \xi_{+\mathscr{U}} \\ & X_{\mathscr{U}}^T w + w_0 e \leq -\epsilon e + \xi_{-\mathscr{U}} \\ & \xi_{\mathscr{C}}, \xi_{+\mathscr{U}}, \xi_{-\mathscr{U}} \geq 0. \end{aligned}$$

A dual formulation can be developed by following the procedure described in Section 3.2 of forming the Lagrangian and partially differentiating with respect to the primal variables.

Let us define the block matrices and vectors $X = \begin{bmatrix} X_{\mathscr{C}}, & X_{\mathscr{U}}, & X_{\mathscr{U}} \end{bmatrix}$, $y = \begin{bmatrix} y_{\mathscr{C}} \\ e_{\mathscr{U}} \\ -e_{\mathscr{U}} \end{bmatrix}$, $Y = \text{diag}(y)$, $c = \begin{bmatrix} -e_{\mathscr{C}} \\ \epsilon e_{+\mathscr{U}} \\ \epsilon e_{-\mathscr{U}} \end{bmatrix}$, and dual variables for the hyperplane constraints $z = \begin{bmatrix} z_{\mathscr{C}} \\ z_{+\mathscr{U}} \\ z_{-\mathscr{U}} \end{bmatrix}$.
Using this notation, the dual formulation becomes:

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Y X^T X Y z + c^T z \\ \text{s.t.} \quad & -y^T z = 0 \\ & 0 \le z_{\mathscr{C}} \le \tau_{\mathscr{C}} e_{\mathscr{C}} \\ & 0 \le z_{+\mathscr{U}}, z_{-\mathscr{U}} \le \tau_{\mathscr{U}} e_{\mathscr{U}}. \end{aligned}$$

Using the relationship between w and z at optimality

$$w = X_{\mathscr{C}} Y_{\mathscr{C}} z_{\mathscr{C}} + X_{\mathscr{U}} z_{+\mathscr{U}} - X_{\mathscr{U}} z_{-\mathscr{U}}$$

the Universum classification problem can be transformed into an equivalent separable formulation

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2} w^T w + c^T z \\ \text{s.t.} \quad & w - XYz = 0 \\ & -y^T z = 0 \\ & 0 \le z_{\mathscr{C}} \le \tau_{\mathscr{C}} e_{\mathscr{C}} \\ & 0 \le z_{+\mathscr{U}}, z_{-\mathscr{U}} \le \tau_{\mathscr{U}} e_{\mathscr{U}}. \end{aligned} \tag{5.5}$$

5.5 Ordinal regression

Ordinal regression bridges multi-class classification and metric regression, as it involves classification categories that are ranked. Unlike metric regression problems, however, the label numbers are discrete and the metric distances between labels do not have any real significance. Also, order information is present, unlike multiple classification problems.

Several approaches have been proposed to move beyond using multiple classification techniques or from naively transforming the ordinal scales into numerical values and solving as a standard regression problem. In the formulation of Herbrich et al. (2000), the goal is to learn a function $f(x) = w^T x + w_0$ which correctly orders the samples, so that $f(x_i) > f(x_j) \Leftrightarrow$

$y_i > y_j$ for any pair of examples (x_i, y_i) and (x_j, y_j). Using the set of pairings $\mathscr{P} = \{(i,j) : y_i > y_j\}$, the authors formulate the following ordinal regression SVM:

$$\begin{aligned} \min_{w,\xi} \quad & \frac{1}{2} w^T w + \tau \sum_{(i,j)\in\mathscr{P}} \xi_{ij} \\ \text{s.t.} \quad & w^T(x_i - x_j) \geq 1 - \xi_{ij} \qquad \forall (i,j) \in \mathscr{P} \\ & \xi_{ij} \geq 0 \qquad\qquad\qquad\quad \forall (i,j) \in \mathscr{P}. \end{aligned}$$

The objective promotes a large-margin linear function $f(x)$ that also minimizes the number of pairs of training examples that are incorrectly ordered. The formulation has the same structure as the classification SVM, and therefore it is open to the reformulation in Section 5.1.

There are two main disadvantages with the above formulation. The first is that the hyperplane bias w_0 does not play a role in the optimization problem, but has to be estimated afterwards. The second disadvantage is that the number of constraints and the number of variables ξ grow quadratically with the training data set size. Partly in order to address that, two new approaches were proposed by Chu and Keerthi (2005) for support vector ordinal regression where the size of the training problem is linear in the number of samples.

The first formulation ("explicit thresholds") takes only the ranks immediately adjacent to each separating hyperplane to determine each threshold w_0^j. They introduce the constraints $w_0^{(j-1)} \geq w_0^j \quad \forall j$ explicitly on the thresholds to enforce the correct ordering. The reverse ordering of w_0^j is due to us using $w^T x + w_0^j = 0$ to define the hyperplane. Assume that there are r classes, indexed with $j \in \mathscr{J} = \{1, 2, \ldots, r\}$, each with n^j data samples. $r-1$ parallel hyperplanes separate the classes; the hyperplane with bias w_0^j separates class j from class $j+1$. $X^j \in \mathbb{R}^{m \times n^j}$ is the data matrix for class j. We define the misclassification error vector $\xi_+^j \in \mathbb{R}^{n^j}$ and dual variables $z_+^j \in \mathbb{R}^{n^j}$ for points in class j which should lie above the hyperplane $j-1$, and similarly errors $\xi_-^j \in \mathbb{R}^{n^j}$ and dual variables $z_-^j \in \mathbb{R}^{n^j}$ for points in class j below hyperplane j. These are shown in Figure 5.2. Variables ξ_-^j and z_-^j are defined for all classes $j = 1, \ldots, r-1$, while ξ_+^j and z_+^j are defined for all classes $j = 2, \ldots, r$, but we can write a simplified but equivalent formulation if we add auxiliary variables $\xi_+^1, z_+^1, \xi_-^r, z_-^r = 0$. We also introduce dual variables $\beta^j \in \mathbb{R}$ for each of the ordering constraints. Again it simplifies the formulation if we set $w_0^0 = +\infty$ and $w_0^r = -\infty$. Then, with $j = 1, \ldots, r$, the primal formulation

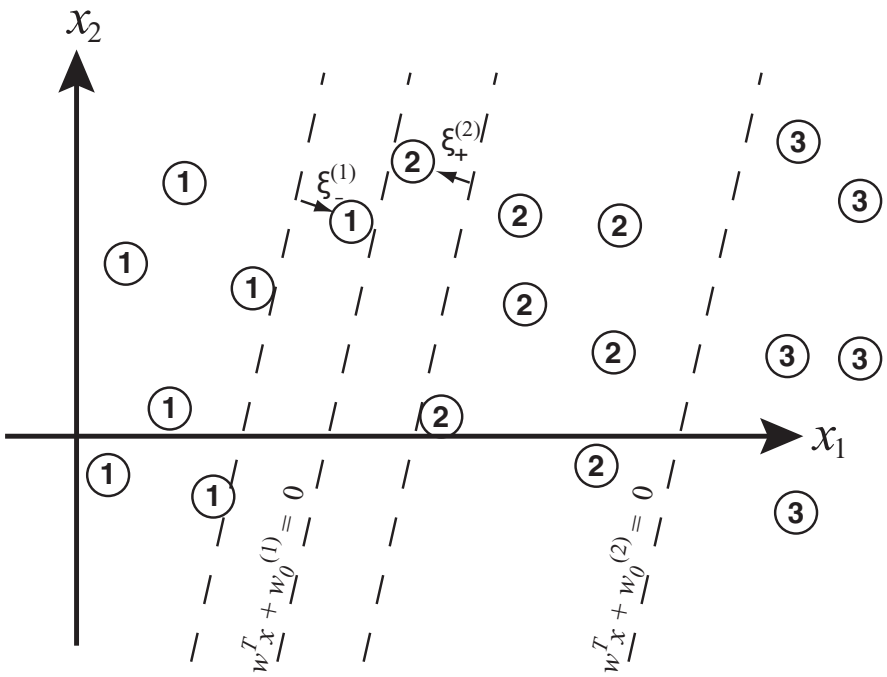


Figure 5.2: Explicit threshold ordinal regression uses parallel separating hyperplanes, but determines each of their positions using the data from the pair of adjacent classes.

is:

$$\begin{aligned}
\min_{w,w_0^j,\xi_-^j,\xi_+^j} \quad & \frac{1}{2}w^T w + \tau \sum_{j=1}^{r} \left(e^T \xi_-^j + e^T \xi_+^j \right) \\
\text{s.t.} \quad & (X^j)^T w + w_0^j e \leq -e + \xi_-^j \qquad \forall\, j \\
& (X^j)^T w + w_0^{(j-1)} e \geq e - \xi_+^j \qquad \forall\, j \\
& w_0^{j-1} \geq w_0^j \qquad \forall\, j \\
& \xi_-^j, \xi_+^j \geq 0 \qquad \forall\, j.
\end{aligned}$$

By following the same Lagrange duality technique as used in Section 3.2, and applying the relationship between w and $(z_+^j - z_-^j)$ at optimality, an equivalent separable formulation is:

$$\begin{aligned}
\min_{w,z_-,z_+,\beta} \quad & \frac{1}{2}w^T w - \sum_j e^T (z_+^j + z_-^j) \\
\text{s.t.} \quad & w = \sum_j X^j (z_+^j - z_-^j) \\
& e^T z_-^j + \beta^j = e^T z_+^{j+1} + \beta^{j+1} \qquad \forall\, j = 1, \ldots, r-1 \\
& 0 \leq z_-^j, z_+^j \leq \tau e \qquad \forall\, j \\
& \beta^j \geq 0 \qquad \forall\, j.
\end{aligned} \tag{5.6}$$

In the second formulation ("implicit thresholds") of Chu and Keerthi (2005), there are no constraints to correctly order the hyperplane biases. Instead, samples from all of the classes

are used to define each threshold, and this approach ensures the correct ordering. X^j is defined as before. $\xi_+^{jk} \in \mathbb{R}^{n^k}$ is the misclassification error vector for class k that should lie above the hyperplane j, and similarly $\xi_-^{jk} \in \mathbb{R}^{n^k}$ for classes lying below hyperplane j. $z_+^{jk} \in \mathbb{R}^{n^k}$ and $z_-^{jk} \in \mathbb{R}^{n^k}$ are the dual variables for the hyperplane constraints. With $j = 1, \ldots, r-1$, the primal formulation is then:

$$\begin{aligned}
\min_{w, w_0, \xi_-, \xi_+} \quad & \frac{1}{2} w^T w + \tau \sum_{j=1}^{r-1} \left(\sum_{k=1}^{j} e^T \xi_-^{jk} \quad + \sum_{k=j+1}^{r} e^T \xi_+^{jk} \right) \\
\text{s.t.} \quad & (X^k)^T w + w_0^j e \leq -e + \xi_-^{jk} \quad \forall\, j \text{ and } k = 1, \ldots, j \\
& (X^k)^T w + w_0^j e \geq e - \xi_+^{jk} \quad \forall\, j \text{ and } k = j+1, \ldots, r \\
& \xi_-^{jk} \geq 0 \quad \forall\, j \text{ and } k = 1, \ldots, j \\
& \xi_+^{jk} \geq 0 \quad \forall\, j \text{ and } k = j+1, \ldots, r.
\end{aligned}$$

Using the relationship $w = -\sum_{j=1}^{r-1} \left(\sum_{k=1}^{j} X^k z_-^{jk} - \sum_{k=j+1}^{r} X^k z_+^{jk} \right)$, our equivalent separable formulation is:

$$\begin{aligned}
\min_{z_-, z_+} \quad & \frac{1}{2} w^T w - \sum_{j=1}^{r-1} \left(\sum_{k=1}^{j} e^T z_-^{jk} + \sum_{k=j+1}^{r} e^T z_+^{jk} \right) \\
\text{s.t.} \quad & w + \sum_{j=1}^{r-1} \left(\sum_{k=1}^{j} X^k z_-^{jk} - \sum_{k=j+1}^{r} X^k z_+^{jk} \right) = 0 \\
& \sum_{k=1}^{j} e^T z_-^{jk} = \sum_{k=j+1}^{r} e^T z_+^{jk} \quad \forall\, j \\
& 0 \leq z_-^{jk} \leq \tau e \quad \forall\, j \text{ and } k = 1, \ldots, j \\
& 0 \leq z_+^{jk} \leq \tau e \quad \forall\, j \text{ and } k = j+1, \ldots, r.
\end{aligned}$$

5.6 Regression

Support Vector Regression (SVR) uses similar techniques to learn a mapping between the input vector x and a real-valued target value y. In ϵ-insensitive SVR, the loss function is defined as $L^\epsilon \equiv max(0, |y - f(x)| - \epsilon)$. This loss is represented by the errors ξ_i and $\hat{\xi}_i$ if the predicted value $f(x_i)$ of the point x_i is above or below the band around y of half-width ϵ. Full descriptions are again given in Cristianini and Shawe-Taylor (2000); Vapnik (1998, 1999).

The dual variables $z, \hat{z}$ are the Lagrange multipliers associated with the two sets of constraints. The objective function minimizes risk using the SRM principle (balancing the complexity of the function against misclassifications in the training data), resulting in the

primal optimization problem

$$\begin{aligned} \min_{w,\xi,\hat{\xi}} \quad & \frac{1}{2} w^T w + \tau e^T (\xi + \hat{\xi}) \\ \text{s.t.} \quad & y - (X^T w + w_0 e) \leq \epsilon e + \xi \\ & (X^T w + w_0 e) - y \leq \epsilon e + \hat{\xi} \\ & \xi, \hat{\xi} \geq 0 \end{aligned}$$

and its dual

$$\begin{aligned} \min_{z,\hat{z}} \quad & \frac{1}{2} (z - \hat{z})^T X^T X (z - \hat{z}) - y^T (z - \hat{z}) + \epsilon e^T (z + \hat{z}) \\ \text{s.t.} \quad & e^T (z - \hat{z}) = 0 \\ & 0 \leq z, \hat{z} \leq \tau e. \end{aligned} \tag{5.7}$$

The relationship between w and $(z, \hat{z})$ is now $w = X(z - \hat{z})$.

We exploit separability in a similar way for Support Vector Regression, with the introduction into standard dual formulation (5.7) of the auxiliary variable $\bar{z} \equiv z - \hat{z}$ and the relationship $w = X(z - \hat{z})$:

$$\begin{aligned} \min_{w,z,\hat{z},\bar{z}} \quad & \frac{1}{2} w^T w + \epsilon e^T (z + \hat{z}) - y^T \bar{z} \\ \text{s.t.} \quad & -z + \hat{z} + \bar{z} = 0 \\ & w - X\bar{z} = 0 \\ & e^T \bar{z} = 0 \\ & 0 \leq z, \hat{z} \leq \tau e. \end{aligned} \tag{5.8}$$

We define decision variables $(w, z, \hat{z}, \bar{z})$ and the corresponding constraint matrix

$$A = \begin{bmatrix} 0 & -I & I & I \\ I & 0 & 0 & -X \\ 0 & 0 & 0 & e^T \end{bmatrix},$$

while both Q and Θ are diagonal matrices. We need to set bounds on $\bar{z}$ (i.e., $-\tau e \leq \bar{z} \leq \tau e$) and on w (see Section 2.3.5) so that $\Theta_{\bar{z}}$ and Θ_w are defined. The matrix $M \equiv A(Q + \Theta^{-1})^{-1} A^T$ requiring factorization is therefore

$$M = \begin{bmatrix} \Theta_z + \Theta_{\hat{z}} + \Theta_{\bar{z}} & -\Theta_{\bar{z}} X^T & \Theta_{\bar{z}} e \\ -X\Theta_{\bar{z}} & (I_m + \Theta_w^{-1})^{-1} + X\Theta_{\bar{z}} X^T & -X\Theta_{\bar{z}} e \\ e^T \Theta_{\bar{z}} & -e^T \Theta_{\bar{z}} X & e^T \Theta_{\bar{z}} e \end{bmatrix}.$$

The block Cholesky factorization LDL^T of matrix M can be computed efficiently as follows:

$$LDL^T = M = \begin{bmatrix} I_n & \\ F & L_m \end{bmatrix} \begin{bmatrix} D_n & \\ & D_m \end{bmatrix} \begin{bmatrix} I_n & F^T \\ & L_m^T \end{bmatrix},$$

where

$$D_n = \Theta_z + \Theta_{\hat{z}} + \Theta_{\bar{z}}$$

$$F = \begin{bmatrix} -X \\ e^T \end{bmatrix} \Theta_{\bar{z}} D_n^{-1},$$

while L_m and D_m are found from the Cholesky factorization

$$L_m D_m L_m^T = \begin{bmatrix} (I_m + \Theta_w^{-1})^{-1} & \\ & 0 \end{bmatrix} + \begin{bmatrix} -X \\ e^T \end{bmatrix} (\Theta_{\bar{z}} - \Theta_{\bar{z}} D_n^{-1} \Theta_{\bar{z}}) \begin{bmatrix} -X^T & e \end{bmatrix}.$$

The formation of this smaller matrix is an $\mathcal{O}(n(m+1)^2)$ operation, while the factorization is of order $\mathcal{O}((m+1)^3)$. Calculation of the other variables require $\mathcal{O}(n)$ operations.

5.7 Conclusion

Support Vector Machines are a powerful machine learning technique. In previous chapters we have made the case that it is not straightforward to extend it to very large-scale problems. Due to the Hessian in the standard dual formulation being completely dense, interior point methods have not traditionally been used. Instead, standard SVM tools have mainly been based around active-set methods. These work well for small and separable problems, but when the split between basic and non-basic variables becomes less clear (as is the case with noisy data sets), the performance of these algorithms starts to scale exponentially with the number of samples. Previous IPM-based approaches have exploited the structure of the linear kernel, to give algorithms with an overall complexity of $\mathcal{O}(nm^2)$. However, these algorithms have suffered from either numerical instability through use of the Sherman-Morrison-Woodbury formula, or memory caching inefficiencies.

In this chapter we have presented a new, unified set of formulations for 1-norm and 2-norm classification, universum and ordinal classification, and ϵ-insensitive regression. These all exploit the separability of the Hessian in the objective of the standard SVM primal formulation, while keeping a small number of constraints as in the dual. As with the other IPM-based approaches, it has a per-iteration complexity of $\mathcal{O}(nm^2 + m^3)$. It relies upon

Cholesky decomposition for its numerical stability. All m features are considered simultaneously during the normal equations matrix calculation procedure, allowing for a more efficient implementation in terms of memory caching.

Chapter 6

Numerical experiments and results

In Chapter 5, we showed how interior point methods can be specialized to exploit separability efficiently in a range of linear SVM training problems. In terms of computational complexity, our approach matches the other IPM-based techniques mentioned in Chapter 4.

In this chapter, performance is investigated through extensive numerical experiments, using the standard L1-SVM linear binary classification problem. We show that our approach gives consistent and highly competitive training times, and for some problems it outperforms other implementations of the same classification problem (including the active-set and cutting plane algorithms mentioned above) by a large margin.

The numerical results in Section 6.1 were based on artificial data sets. The training data sets used were created by uniformly sampling points in the space $[-1,+1]^m$. A separating hyperplane was defined by choosing random integer values for w in the range [-1000,1000]. Points were labelled based on the hyperplane. For the non-separable data sets, the required proportion of data points were randomly selected and misclassified. In contrast to the training data, the test data sets contain no misclassified points. The evaluations in Section 6.3 use real-world data sets.

6.1 Specialization techniques

We implemented the L1-SVM formulation (5.1) in `HOPDM` (Gondzio, 1995; Altman and Gondzio, 1999) which was modified to perform matrix multiplications in dense mode using the BLAS library (Lawson et al., 1979). The experiments were performed using an Intel Pentium 4 PC running at 3GHz, with 1GB RAM and 1024KB cache. BLAS functions were provided by Intel's

Technique	Non-optimized BLAS library	Intel optimized BLAS library
Multiplication of individual elements	296.36	296.36
Outer products (DSYRK)	25.45	20.50
Matrix-vector multiplication (DGEMV)	27.18	22.00
Block-based matrix multiplication (DGEMM)	15.27	8.58

Table 6.1: Time taken (in seconds) to train an SVM, comparing different techniques to calculate $A(Q+\Theta^{-1})^{-1}A^T$. The data set given contained 5000 points of 320 attributes. 14 IPM iterations were required.

Maths Kernel Library [1].

6.1.1 Dense linear algebra operations

A comparison of several techniques for calculating $A(Q+\Theta^{-1})^{-1}A^T$ is shown in Table 6.1. Any technique that takes advantage of the structure of the problem gave better performance than multiplying the elements individually. Computing $A(Q+\Theta^{-1})^{-1}A^T$ by outer products most directly exploits the structure (as $(Q+\Theta^{-1})$ is diagonal), but the computation using DGEMM on blocks of 32 data points at a time gave the best performance, probably due to better use of the CPU's cache.

6.1.2 Confirmation of scalability

The complexity analysis in the previous chapter gave the computations required for each iteration as $\mathcal{O}(nm^2)$ if $n \gg m$. To verify this, the software was trained first using data sets with 255 features. Figure 6.1(a) shows that the length of time taken by an iteration varies linearly with the number of samples n. However, the total time taken by the algorithm increases super-linearly with the number of samples, as the number of iterations required also increases slowly.

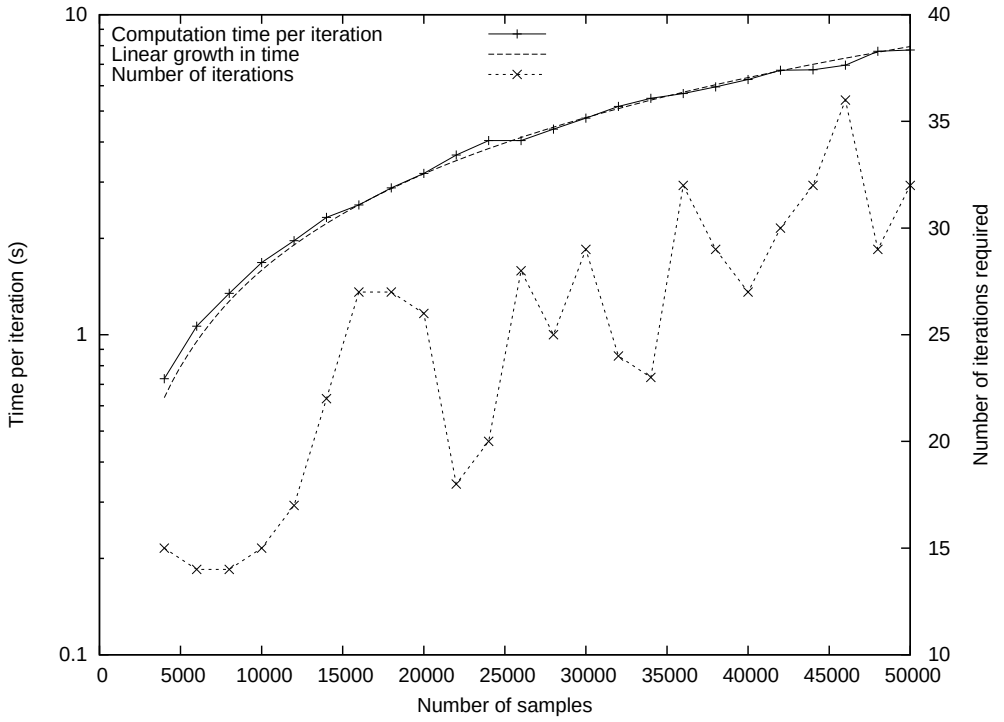
The experiment was repeated for the number of features, using a data set of 20,000 samples, and the number of features varied. Figure 6.1(b) shows that, once there is a reasonably large number of features, approximately $m > 250$, the algorithm scales quadratically with m, while the number of iterations required remains roughly the same.

6.1.3 Bounds on w

In the formulation (5.1) w is free, while the standard IPM formulation (2.2) requires all variables to be in the positive quadrant. Several approaches to this problem were described

[1] http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/index.htm

(a)



(b)

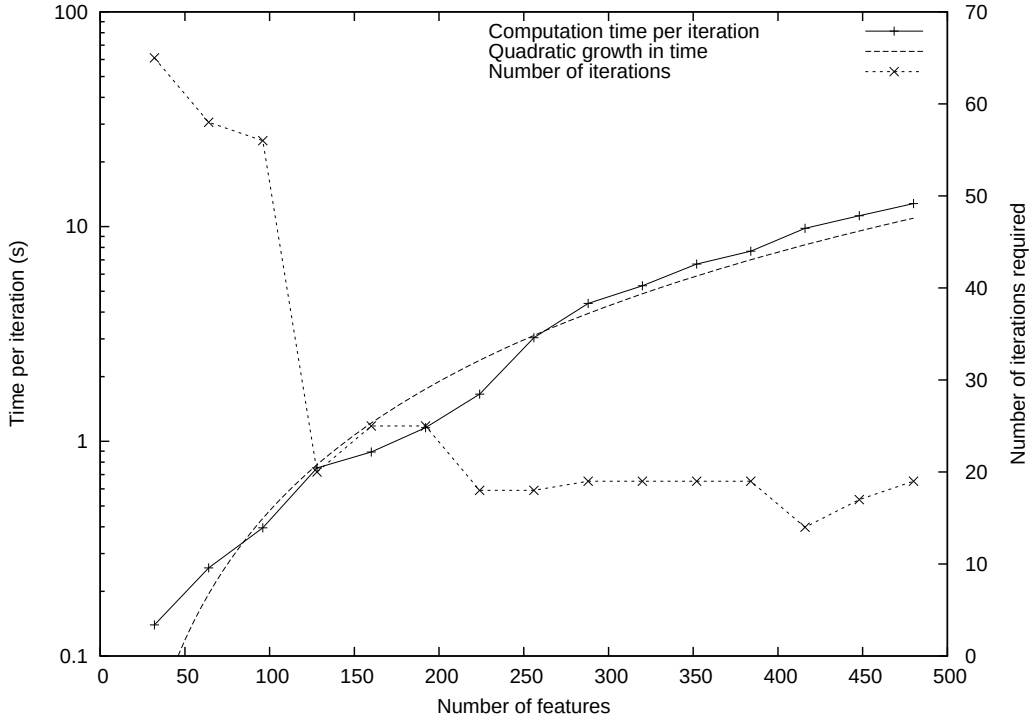


Figure 6.1: The computational scalability of iteration time is predictable, and results confirm $\mathcal{O}(nm^2)$. The number of iterations required is less predictable, but grows slowly with n. (a) Computational complexity and iteration count with respect to the number of samples n, using fully separable data sets with 255 features. (b) Computational complexity and iteration count with respect to the number of features m. Data sets were fully separable with 20,000 samples.

in Section 2.3.5. The approach we adopted is to define bounds $l_w \le w \le u_w$, and the problem can then be adjusted appropriately to shift the bounds to $0 \le w' \le u_w - l_w$. From (5.1), $w = XYz$, so bounds can be safely set as $\tau \sum_i \min(y_i x_{ij}, 0) \le w_j \le \tau \sum_i \max(y_i x_{ij}, 0)$. For problems where the solution set of support vectors is sparse, the optimal values for w differ substantially from either bound. Since large bounds affect the numerical accuracy of the algorithm, it is useful to tighten the bounds to within say a couple of orders of magnitude of the true values of w once these are known (e.g. when searching for the best parameters through repeated training).

6.1.4 Accuracy due to termination criteria

Several termination criteria are possible. Normally the measure of most interest for an optimization problem is the value of the objective function, and the algorithm stops when this value is reached to within a set relative error, e.g. 10^{-8}, but for SVMs the objective value is not of interest so may not be a good basis for termination. Similarly, the errors associated with primal and dual feasibility can be monitored, and the algorithm terminated when these are within a small tolerance.

The approach normally used for SVM is to monitor the set of support vectors, and terminate the algorithm when this is stable. The KKT complementarity conditions are used to determine the support vectors.

With the formulation presented in this paper, we have access to the weights variables w directly. It is therefore possible to monitor these values, and measure the change in the angle ϕ of the normal to the hyperplane between iteration $i-1$ and i:

$$\cos\phi = \frac{(w^{(i-1)})^T w^{(i)}}{\|w^{(i-1)}\| \|w^{(i)}\|}$$

We conducted experiments to see how these measures relate to classification accuracy, using a training set of 20,000 samples and 255 features, with 5% misclassifications, and a separable test set of the same size.

Figure 6.2 shows how the duality gap and $\sin\phi$ decrease as the IPM algorithm progresses. Primal feasibility was reached quickly, while it took the algorithm longer to attain dual feasibility. All measures were sensitive to the scale of the bounds on w, which made it hard to define a set tolerance for any of the measures. In particular, the values of w decrease with each iteration, so it is not useful to monitor these to see if they are converging to their final values.

A noticeable feature of the figures is that a high classification accuracy is achieved at an early stage in the algorithm, long before the number of support vectors has stabilized, indicating that the hyperplane has been accurately identified at this point. Although at this stage the values of the weights change in scale, proportionally they are stable, as can be seen by measuring $\sin\phi$ (Figure 6.2). Once suitable bounds on w have been established, a tolerance of 10^{-4} on $\sin\phi$ could be used to give earlier termination.

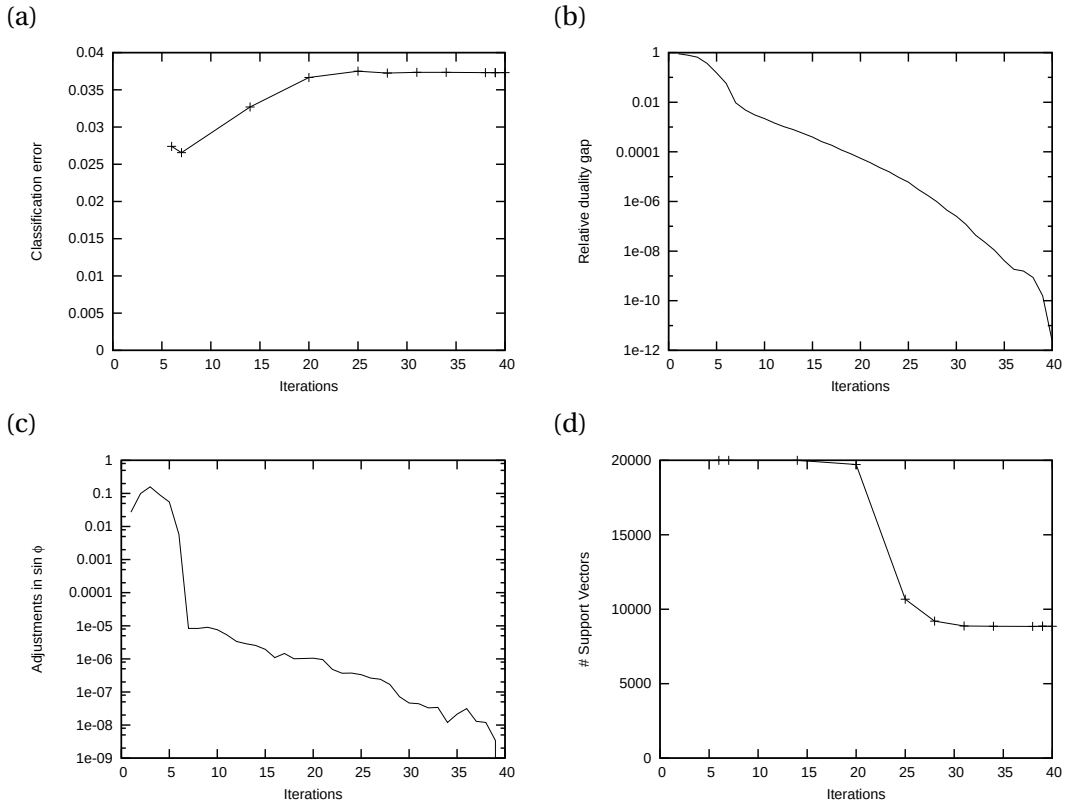


Figure 6.2: Performance of the algorithm relative to the number of IPM iterations, using a training data set with 5% of points misclassified, 20,000 samples and 255 attributes. (a) Classification error using an unseen test set. (b) Error in the value of the objective. (c) Change in angle of the normal to the separating hyperplane. (d) The number of support vectors.

6.1.5 Multiple correctors

The use of multiple correctors (Gondzio, 1996; Colombo and Gondzio, 2008) can reduce the number of IPM iterations required, by improving the centrality of the current iterate. Several correctors can be calculated by repeatedly solving $M\Delta\lambda = -\hat{r}_b$ for $\Delta\lambda$. The same factorization of M is used for all the correctors in an iteration, so it is advantageous to perform multiple corrections when the effort involved in the back-solves (here $\mathcal{O}(nm+m^2)$) is significantly

less than that of factorizing M (here $\mathcal{O}(nm^2 + m^3)$).

We conducted experiments to show the comparative performance of the algorithm using multiple correctors, against the algorithm using a single Mehrotra's corrector. Both the number of iterations and the overall time were improved by using multiple correctors. For example, using a data set of 20,000 samples and 255 features, an average of 2.8 correctors were used in each iteration, and the optimization required 2 fewer iterations.

6.1.6 Stability in case of near-linear dependency in X

We used the data set of Goldfarb and Scheinberg (2005) that caused an algorithm using the Sherman-Morrison-Woodbury update to fail. This data set (shown in Figure 6.3) causes degeneracy in the matrix (XY), as there are multiple data points which lie along the separating hyperplanes. Scaling one of the dimensions accentuates the numerical instability. With our algorithm, there was no penalty in performance, with the number of IPM iterations required always around 20 no matter the scaling imposed on the data set (this is similar performance to that reported by Goldfarb and Scheinberg for their Product Form Cholesky Factorization algorithm). Stability of our approach is a consequence of the use of primal-dual regularization, which replaces the linear systems in the interior point method with better conditioned ones. This is achieved by adding dynamically chosen small quadratic proximal terms to primal and dual objectives in the IPM algorithm. Such an addition improves the conditioning of the linear systems without slowing down the convergence of IPM. For a detailed description of the method and extensive numerical results which demonstrate its advantages the reader is referred to Altman and Gondzio (1999).

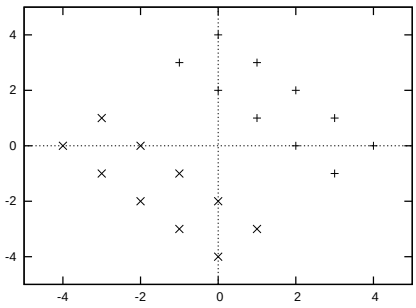


Figure 6.3: The data set of Goldfarb and Scheinberg (2005) causing degeneracy in the constraint matrix.

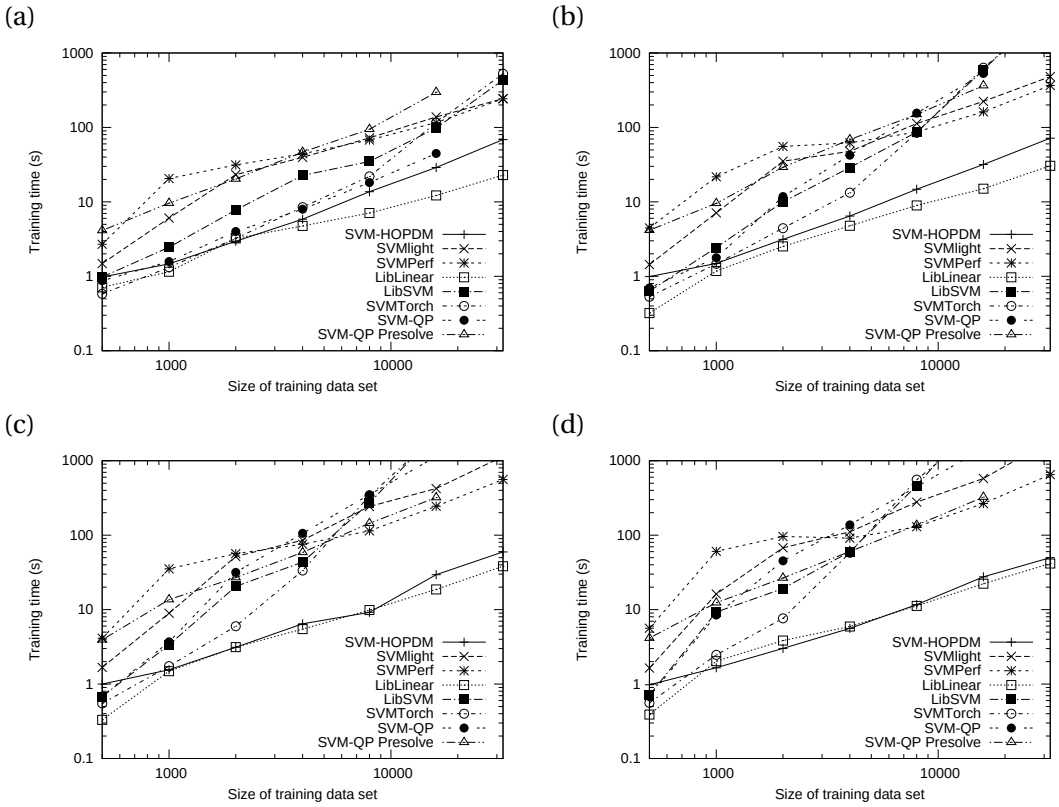


Figure 6.4: Comparison of efficiency of `SVM-HOPDM` against other algorithms, with respect to data set size, as noise is increased. Artificial data sets with 255 attributes were used. $\tau = 1$. (a) Fully separable. (b) 1% misclassified. (c) 5% misclassified. (d) 10% misclassified.

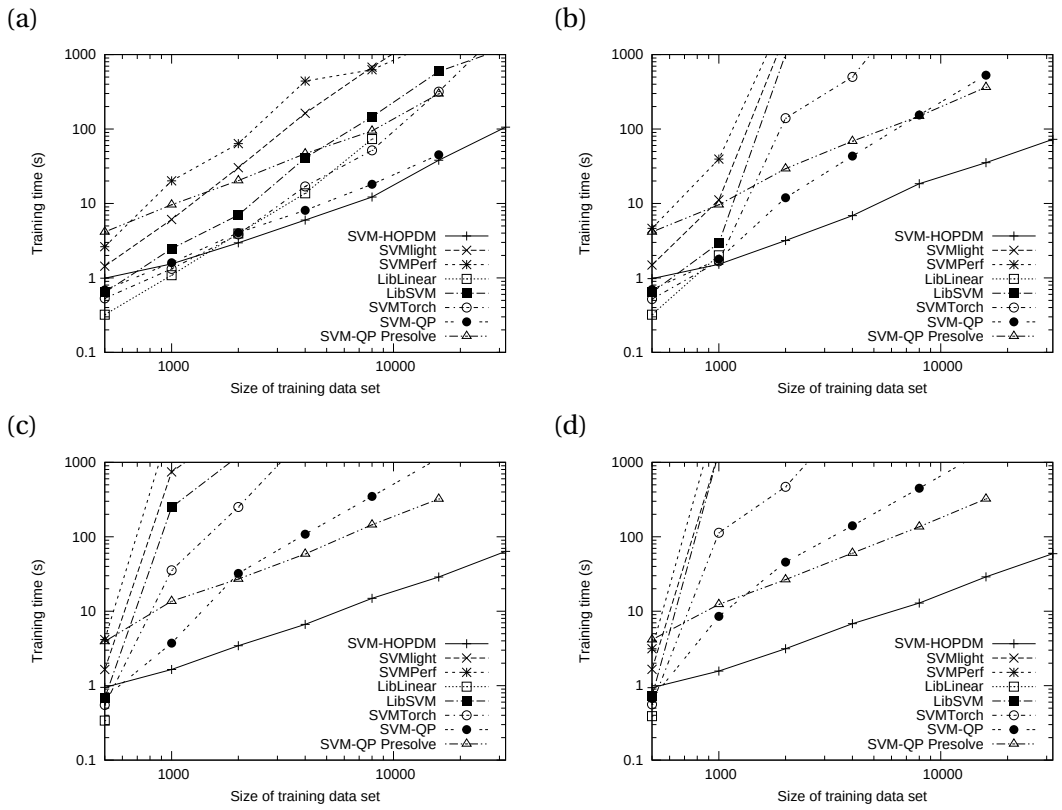


Figure 6.5: Comparison of algorithm efficiency, similar to Figure 6.4 but with higher penalty for misclassifications ($\tau = 100$). (a) Fully separable. (b) 1% misclassified. (c) 5% misclassified. (d) 10% misclassified.

6.2 Comparison against standard tools

To assess the performance of our algorithm SVM-HOPDM, we tested it against a range of state-of-the-art SVM tools: $\text{SVM}^{\text{light}}$ (Joachims, 1999), SVM^{perf} (Joachims, 2006), LIBLINEAR (Fan et al., 2008), LIBSVM (Chang and Lin, 2001) and SVMTorch (Collobert and Bengio, 2001). We also included the SVM-QP active set algorithm (Fine and Scheinberg, 2002) and the IPM-based algorithm (SVM-QP Presolve) (Fine and Scheinberg, 2001) that uses the Product Form Cholesky Factorization described earlier (Goldfarb and Scheinberg, 2005). They were all used with their software performance options (such as cache size) set to their default values. We conducted SVM training experiments using synthetically-constructed data sets as described earlier, with 255 features. SVMTorch has been used as the comparison tool for other IPM-based techniques, e.g. Ferris and Munson (2003); Gertz and Griffin (2009); Goldfarb and Scheinberg (2005).

Figure 6.4 shows the comparative efficiency of the algorithms as the size of the data set is increased, with a relatively low penalty for misclassifications ($\tau = 1$). For separable data sets (a) all the algorithms show linear or sublinear scaling. But the relative performance changes dramatically when the data set contains noise, which is typically the case with real-world data sets. We used synthetic data sets again, but this time introduced noise by choosing (b) 1%, (c) 5% and (d) 10% of the points randomly and swapping their target label. The computation time required by the active set methods is greatly increased by introducing noise, while other algorithms are less affected.

The experiments were repeated for a higher misclassification penalty of $\tau = 100$ (Figure 6.5). It can be clearly seen that all except the IPM algorithms are greatly affected by the level of noise, with training times increased by several orders of magnitude, or the algorithms fail to converge.

The training times of SVM-HOPDM and SVM-QP Presolver, both based on interior point methods, are similar in all eight cases, yet there was almost an order of magnitude difference between the two algorithms. This difference cannot be accounted for by a complexity analysis. We investigated this further using Valgrind's Cachegrind cache simulator[2] set as a Pentium 4 processor cache, and the results for four data sets are shown in Table 6.2. The algorithms required different numbers of iterations which complicated the comparison, so we considered only the functions associated with forming and solving the normal system matrix, and this accounted for some 70% to 80% of the instructions executed (the "coverage" in Table

[2] http://valgrind.org/

6.2). The final two columns of the table show instruction count and runtime ratios for the two programs. It is clear that the number of executed instructions does not explain the whole increase in runtime. The number of data read cache misses (which is determined by how the algorithm accesses the data structure) is also an important factor in runtime performance, yet it is rarely discussed in comparisons of computational complexity of algorithms.

Data set		SVM-HOPDM			SVM-QP presolver			SVM-QP/ SVM-HOPDM	
n	m	Time	Cov.	D2mr	Time	Cov.	D2mr	Instructions	Time
10000	63	3.69	70%	2.68%	16.44	72%	6.99%	2.36	4.46
10000	127	10.66	83%	1.73%	55.46	80%	7.56%	1.94	5.21
10000	255	15.67	79%	1.34%	127.53	79%	7.72%	2.73	8.14
20000	63	9.13	76%	2.97%	40.74	76%	6.95%	2.15	4.46

Table 6.2: Comparison of SVM-HOPDM and SVM-QP Presolver in terms of instructions and cache misses. Synthetic data sets of dimension n samples and m features were used. Time is the total runtime of the program, running with hardware cache, in seconds. Coverage is the proportion of the program included in the instruction and cache miss count. D2mr is the proportion of Level 2 data read misses to total Level 2 data reads. The final two columns show the ratio between the two programs, for instructions and runtime. The increase in runtime of SVM-QP Presolver cannot be accounted for by instructions alone, and cache performance has a significant effect.

6.3 Real-world data sets

To investigate what performance results can be expected in real-world applications, we used the standard data sets Adult, Covtype, MNIST, SensIT and USPS.[3] Each problem was solved using a linear kernel with $\tau = 1, 10$ and 100. Table 6.3 shows the wall-clock times to perform the training (including time taken to read the data).

The same results are shown as a performance profile (Dolan and Moré, 2002) in Figure 6.6. Here, the runtime $t_{s,p}$ for each solver $s \in \mathscr{S}$ on problem $p \in \mathscr{P}$ is transformed into a ratio to the fastest solver for problem p:

$$r_{s,p} = \frac{t_{s,p}}{\min_{s \in \mathscr{S}} t_{s,p}}.$$

The performance profile is the cumulative distribution function of these ratios for each solver

$$\rho_s(T) = \frac{\text{size}\{p \in \mathscr{P} : |r_{s,p} \leq T\}}{\text{size}\{\mathscr{P}\}},$$

[3] All data sets are available from the LIBSVM collection at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. Due to memory restrictions, some data sets were reduced to the sizes given in Table 6.3. SVM-QP had tighter memory restrictions, so the data sets were further reduced and the times linearly scaled up; this is probably fair for the SVM-QP presolver but is rather favourable for the active set solver.

Data set ($n \times m$)	τ	SVM-HOPDM	SVM^{light}	SVM^{perf}	LIB-LINEAR	LIBSVM	SVMTorch	SVM-QP	SVM-QP presolve
Adult	1	16.5	87.7	280.7	1.6	192.4	621.8	164.5	188.8
32561 × 123	10	26.5	1043.3	3628.0	9.3	857.7	5046.0	284.1	206.8
	100	27.9	10447.4	29147.2	64.2	5572.1	44962.5	544.8	216.9
Covtype	1	47.7	992.4	795.6	8.5	2085.8	2187.9	731.8	405.6
150000 × 54	10	52.7	6021.2	12274.5	34.3	2516.7	10880.6	971.6	441.3
	100	55.4	66263.8	58699.8	235.2	6588.0	74418.1	1581.8	457.4
MNIST	1	79.6	262.9	754.1	9.3	197.1	660.1	233.0	1019.1
10000 × 780	10	83.4	3425.5	8286.8	65.4	1275.2	5748.1	349.4	1104.4
	100	86.2	NC	196789.0	NC	11456.4	54360.6	602.5	1267.1
SensIT	1	55.2	913.5	8418.3	53.6	2542.0	2814.4	535.2	456.7
78823 × 100	10	60.1	7797.4	> 125000	369.1	7867.8	21127.8	875.4	470.7
	100	63.6	NC	> 125000	NC	49293.7	204642.6	1650.1	489.3
USPS	1	13.2	15.0	40.9	4.4	10.4	7.7	51.2	117.4
7291 × 256	10	14.2	147.4	346.6	27.7	20.9	23.9	64.7	127.4
	100	14.3	1345.2	2079.5	NC	93.8	142.4	86.9	143.8

Table 6.3: Comparison of training times using real-world data sets. Each data set was trained using $\tau = 1, 10$ and 100. NC indicates that the method did not converge to a solution.

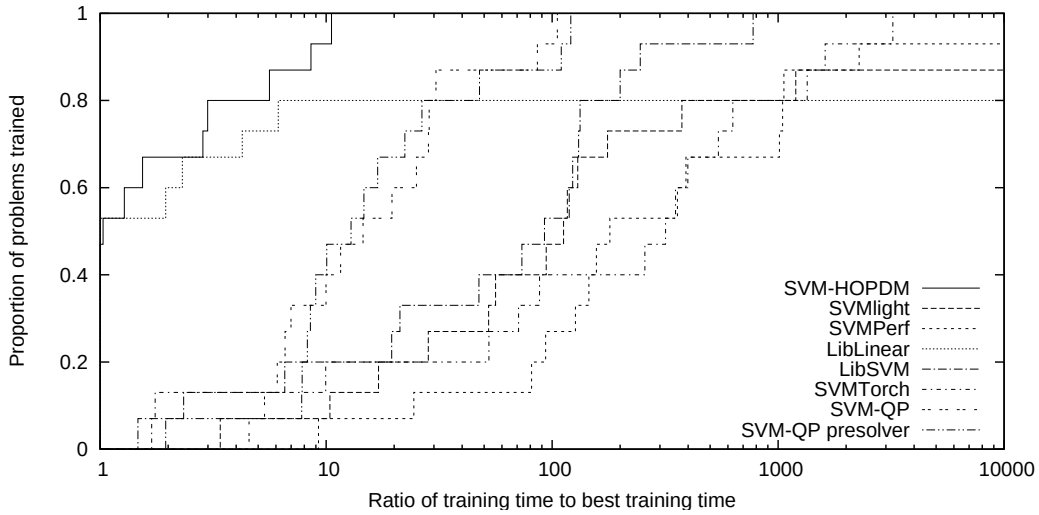


Figure 6.6: Performance profile of the SVM tools on the problems in Table 6.3.

that is the proportion of problems which can be solved with a runtime ratio $r_{s,p}$ less than T. The value of $\rho_s(1)$ is the proportion of problems that solver s wins over other solvers. The value of $\lim_{T\to\infty} \rho_s(T)$ is the proportion of problems solved at all. A high proportion of problems solved with small ratios $r_{s,p}$ is shown by the profile being close to the upper left axes.

The results confirm that real-world data sets do tend to be noisy, as most methods take considerably longer with high τ misclassification penalty values. The performance profile highlights that LIBLINEAR is the fastest for many of the problems, generally involving low values of τ. Note that LIBLINEAR uses the modified optimization problem (4.3) that removes the equality constraints (discussed in Section 4.5). For higher values of τ, however,

Dataset	τ	HOPDM	LIBLINEAR	SVMTorch
Adult	1	85.01%	84.98%	84.95%
32561 × 123	10	84.98%	84.96%	85.01%
	100	84.95%	84.95%	84.97%
Covtype	1	61.59%	61.59%	60.85%
150000 × 54	10	61.92%	61.92%	59.84%
	100	61.92%	61.92%	61.46%
MNIST	1	86.31%	86.31%	86.40%
10000 × 780	10	86.43%	86.40%	86.41%
	100	86.27%	—	86.40%
SensIT	1	85.78%	85.42%	85.78%
78823 × 100	10	85.80%	85.43%	85.82%
	100	85.79%	—	85.85%
USPS	1	96.41%	97.11%	97.11%
7291 × 256	10	97.21%	97.11%	97.21%
	100	97.01%	—	96.66%

Table 6.4: Comparison of prediction accuracy on unseen test sets. For all except Covtype, we used the standard test sets. Covtype does not have a standard test set, so we used the first 150000 samples of the data set for training and the final 100000 samples as the test set (there was no overlap). The results show in terms of accuracy, our method is broadly equivalent to other methods.

SVM-HOPDM is faster than the other solvers. This is due to the training time of our algorithm being roughly constant, relative to the value of τ. It was at most one order of magnitude slower than the fastest solver for any problem, which was the best performance of any of the solvers in this regard: other solvers were two or three orders of magnitude slower, or failed to converge at all. We consider this dependability, in terms of both predictable training times and ability to train with a wide range of τ values, to be a valuable feature of our algorithm.

Table 6.4 confirms that there is no penalty to be paid in terms of prediction accuracy. Experiments using unseen test samples show that the prediction accuracy of our formulation is comparable with other methods.

6.4 Conclusions

Numerical experiments showed that the performance of our algorithm for large dense or noisy data sets is consistent and highly competitive, and in some cases can outperform all other approaches by a large margin. Unlike active set methods, performance is largely unaffected by noisy data. Using multiple correctors, tightening the bounds on w, and monitoring the angle of the normal to the hyperplane all positively contributed to efficiency.

It is possible to extend these formulations to non-linear kernels, by approximating the

kernel with a low-rank outer product representation such as partial Cholesky factorization (Fine and Scheinberg, 2001). We investigate this in Chapter 9.

It is also possible to develop this algorithm to handle very large-scale problems in parallel. The key computation part is the calculation of the matrix M; as described earlier, this was handled on a block basis using the BLAS library. By dividing the sample points equally amongst the processors, the block-based matrix multiplications can be performed in parallel with no communication required between the processors. Then at the end of the multiplication, a single gather operation is required on the $(m+1) \times (m+1)$ matrix at each processor to form the matrix M and then factorize it. This is the subject of the next chapter.

Chapter 7

Parallel implementation

Few approaches have been developed for training SVMs in parallel. The active set techniques used by standard software are essentially sequential—they choose a small subset of variables to form the active set at each iteration, and this selection is based upon the results of the previous iteration. It is not clear how to efficiently implement such an algorithm in parallel, due to the large number of iterations required and the dependencies between each iteration and the next. Yet multiple-core computers are becoming the norm, and data sets are becoming ever larger. It is notable that of the 44 submissions to compete in the PASCAL Challenge on Large-scale Learning (Sonnenburg et al., 2008), only 3 entries were parallel methods.

Parallelization schemes so far proposed have involved splitting the training data to give smaller, separable optimization sub-problems which can be distributed amongst the processors. Dong et al. (2003) used a block-diagonal approximation of the kernel matrix to derive independent optimization problems. The resulting SVMs were used to filter out samples that were likely not to be support vectors. A SVM was then trained on the remaining samples, using the standard serial algorithm. Collobert et al. (2002) proposed a mixture of multiple SVMs where single SVMs are trained on subsets of the training set and a neural network is used to assign samples to different subsets.

Another approach is to use a variation of the standard SVM algorithm that is better suited to a parallel architecture. Tveit and Engum (2003) developed an exact parallel implementation of the Proximal SVM (Fung and Mangasarian, 2001), which classifies points by assigning them to the closer of two parallel planes. Compared to the standard SVM formulation, the single constraint is removed and the result is an unconstrained QP; this substantially changes the learning task, and it is not clear how applicable the Proximal SVM formulation is to real-world data sets.

There have only been a few parallel methods in the literature which train a standard SVM on the whole of the data set. We briefly survey the methods of Zanghirati and Zanni (2003), Graf et al. (2005), Durdanovic et al. (2007) and Chang et al. (2008).

The algorithm of Zanghirati and Zanni (2003) decomposes the SVM training problem into a sequence of smaller, though still dense, QP sub-problems. Zanghirati and Zanni implement the inner solver using variable gradient projection method, which is able to work efficiently on relatively large dense inner problems, and is suitable for implementing in parallel. The performance of the inner QP solver was improved in Zanni et al. (2006).

In the cascade algorithm introduced by Graf et al. (2005), the SVMs are layered. The support vectors given by the SVMs of one layer are combined to form the training sets of the next layer. The support vectors of the final layer are re-inserted into the training sets of the first layer at the next iteration, until the global KKT conditions are met. The authors show that this feedback loop corresponds to standard SVM training.

The algorithm of Durdanovic et al. (2007), implemented in the Milde software, is a parallel implementation of the sequential minimal optimization. The objective function of the dual form (3.6) is expressed in terms of partial gradients. Variables are selected to enter the working set, based on the steepest descent direction, and whether the variables are free to move within their box constraints. A second working set method considers pairwise contributions. Very large data sets can be split across processors. When a variable z_i enters the working set, the owner processor broadcasts the corresponding data vector x_i. All nodes calculate kernel functions and update their portion of the gradient vector. Although many of the operations within an iteration are parallelizable, a very large number of sequential outer iterations are still required. The authors use a hybrid approach to parallelization similar to ours described below, involving a multi-core BLAS library, but its use is limited to Level 1 and 2 operations.

Finally, the approach of Chang et al. (2008) sequences a parallel implementation of partial Cholesky factorization (Fine and Scheinberg, 2001) with an adaption to a parallel environment of the IPM method of Ferris and Munson (2003) that uses the SMW formula.

Most of the previous approaches (Durdanovic et al. 2007 is the exception) have considered the parallel computer system as a cluster of independent processors, communicating through a message passing scheme such as MPI (MPI-Forum, 1995). Advances in technology have resulted in systems where several processing cores have access to a single memory space, and such symmetric multiprocessing (SMP) architectures are becoming prevalent. OpenMP (OpenMP Architecture Review Board, 2008) has proven to work effectively on shared memory

systems, while MPI can be used for message passing between nodes. MPI can also be used to communicate between processors within an SMP node, but it is not immediately clear that this is the most efficient technique.

Most high performance computing systems are now clusters of SMP nodes. On such hybrid systems, a combination of message passing between SMP nodes and shared memory techniques inside each node could potentially offer the best parallelization performance from the architecture, although previous investigations have revealed mixed results (Smith and Bull, 2001; Rabenseifner and Wellein, 2003).

A standard approach to combining the two schemes involves OpenMP parallelization inside each MPI process, while communication between the MPI processes is made only outside of the OpenMP regions. Rabenseifner and Wellein (2003) refer to this style as "master-only". In this chapter, we propose a parallel linear SVM algorithm that adopts this hybrid approach to parallelization. It trains the SVM using the full data set. The separable form (5.1) is used with an interior point method to give efficient optimization, and Cholesky decomposition to give good numerical stability. MPI is used to communicate between clusters, while within clusters we take advantage of the availability of highly efficient OpenMP-based BLAS implementations. Data is distributed evenly amongst the processors. Our approach directly tackles the most computationally expensive part of the optimization, namely the inversion of the dense Hessian matrix, through providing an efficient implicit inverse representation. By exploiting the structure of the problem, we show how this can be parallelized with excellent parallel efficiency. The resulting implementation is significantly faster at SVM training than active set methods, and it allows SVMs to be trained on data sets that would be impossible to fit within the memory of a single processor.

7.1 Implementing the QP for parallel computation

To apply (5.1) to truly large-scale data sets, it is necessary to employ linear algebra operations that exploit the block structure of the formulation (Gondzio and Sarkissian, 2003; Gondzio and Grothey, 2007). Between clusters, the emphasis is on partitioning the linear algebra operations to minimize interdependencies between processors. Within clusters, the emphasis is on accessing memory in the most efficient manner.

7.1.1 Linear algebra operations between nodes

We use the augmented system matrix $H = \begin{bmatrix} -Q-\Theta^{-1} & A^T \\ A & 0 \end{bmatrix}$ corresponding to problem formulation (5.1), where Q and A are as defined in Section 5.1, and Θ is defined in (2.10a). As Q and Θ are diagonal, this results in H having a symmetric bordered block diagonal structure. We can break H into blocks:

$$H = \begin{bmatrix} H_1 & & & & A_1^T \\ & H_2 & & & A_2^T \\ & & \ddots & & \vdots \\ & & & H_p & A_p^T \\ A_1 & A_2 & \dots & A_p & 0 \end{bmatrix}, \tag{7.1}$$

where $H_i = -(Q_i + \Theta_i^{-1})$ are actually diagonal and A_i result from partitioning the data set evenly across the p processors. Due to the "arrow-head" structure of H, a block-based Cholesky decomposition of the matrix $H = LDL^T$ will be guaranteed to have the structure:

$$H = \begin{bmatrix} L_1 & & & \\ & \ddots & & \\ & & L_p & \\ L_{A_1} & \dots & L_{A_p} & L_C \end{bmatrix} \begin{bmatrix} D_1 & & & \\ & \ddots & & \\ & & D_p & \\ & & & D_C \end{bmatrix} \begin{bmatrix} L_1^T & & & L_{A_1}^T \\ & \ddots & & \vdots \\ & & L_p^T & L_{A_p}^T \\ & & & L_C^T \end{bmatrix}$$

Exploiting this structure allows us to compute the blocks L_{A_i} in parallel. Terms that form the Schur complement can be calculated in parallel but must then be gathered, and the corresponding blocks L_C and D_C computed serially. This requires the exchange of matrices of size $(m+1) \times (m+1)$ between processors.

$$H_i = L_i D_i L_i^T \quad \Rightarrow D_i = -(Q_i + \Theta_i^{-1}), \quad L_i = I \tag{7.2a}$$

$$L_{A_i} = A_i L_i^{-T} D_i^{-1} = A_i H_i^{-1} \tag{7.2b}$$

$$C \equiv -\sum_{i=1}^{p} A_i H_i^{-1} A_i^T \tag{7.2c}$$

$$= L_C D_C L_C^T \tag{7.2d}$$

Matrix C is a dense matrix of relatively small size $(m+1) \times (m+1)$, and the Cholesky decomposition $C = L_C D_C L_C^T$ is performed in the normal way on a single processor. It is possible that a coarse-grained parallel implementation of Cholesky decomposition could give

better performance (Luecke et al., 1992), but we did not include this in our implementation as the time taken to perform the decomposition is negligible compared to computing C.

Once the representation $H = LDL^T$ above is known, we can use it to compute the solution of the system $H \begin{bmatrix} \Delta z \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r_c \\ r_b \end{bmatrix}$ through back-substitution. $\Delta z'$, $\Delta\lambda'$ and $\Delta\lambda''$ are vectors used for intermediate calculations, with the same dimensions as Δz and $\Delta\lambda$.

$$\Delta\lambda'' = L_C^{-1}(r_b - \sum_{i=1}^{p} L_{A_i} r_{c_i}) \tag{7.3a}$$

$$\Delta\lambda' = D_C^{-1}\Delta\lambda'' \tag{7.3b}$$

$$\Delta\lambda = L_C^{-T}\Delta\lambda' \tag{7.3c}$$

$$\Delta z_i' = D_i^{-1} r_{c_i} \tag{7.3d}$$

$$\Delta z_i = \Delta z_i' - L_{A_i}^T \Delta\lambda \tag{7.3e}$$

For the formation of LDL^T, equations (7.2a) and (7.2b) can be calculated on each processor individually. Outer products (7.2c) are then calculated, and the results gathered onto a single master processor to form C; this requires each processor to transfer $\frac{1}{2}(m+1)(m+2)$ elements. The master processor performs the Cholesky decomposition of C (7.2d). Each processor needs to calculate $L_{A_i} r_{c_i}$, which again can be performed without any inter-processor communication, and the results are gathered onto the master processor. The master processor then performs the calculations in equations (7.3a), (7.3b) and (7.3c) of the back-substitution. Vector $\Delta\lambda$ is broadcast to all processors for them to calculate equations (7.3d) and (7.3e) locally.

7.1.2 Linear algebra operations within nodes

Within each node, the bulk of operations are due to the contribution of each processor $A_i H_i^{-1} A_i^T$ to the calculation of the Schur complement in (7.2c), and to a lesser extent the calculation of L_{A_i} in (7.2b).

The standard technology for dense linear algebra operations is the BLAS library. Much of the effort to produce highly efficient implementations of BLAS Level 3 (matrix-matrix operations) have concentrated on the routine GEMM, for good reason: Kågström et al. (1998) showed that it is possible to develop an entire BLAS Level 3 implementation based on a highly optimized GEMM routine and a small amount of BLAS Level 1 and Level 2 routines. Their approach focused on efficiently organizing the accessing of memory, both through structur-

ing the data for locality and through ordering operations within the algorithm. Matrices are partitioned into *panels* (block rows or block columns) and further partitioned into blocks of a size that fits in the processor's cache, where access times to the data is much shorter. Herrero (2006) has pursued these concepts further, showing that it is possible to develop an implementation offering competitive performance without the need for hand-optimized routines.

Goto and van de Geijn (2008) have shown that another limiting factor is the process of looking up mappings in the page table between virtual and physical addresses of memory. A more efficient approach ensures that the mappings for all the required data reside in the Translation Look-aside Buffer, effectively a cache for the page table. In practice, the best way of achieving this is to recast the matrix-matrix multiplications as a sum of panel-panel multiplications, repacking each panel into a contiguous buffer. This is the approach implemented in GotoBLAS, the library used in our implementation.

To perform GEMM $C := AB + C$, the algorithm described in Goto and van de Geijn (2008) divides the matrices into panels and uses three optimized components.

1. Divide matrix B into block row panels. Each panel $B_{p\bullet}$ contains all the columns we need, but fewer rows than the original matrix B. As required, pack $B_{p\bullet}$ into a contiguous buffer.

2. Divide matrix A into block column panels $A_{\bullet p}$, so that the inner dimensions of $A_{\bullet p}$ and $B_{p\bullet}$ match. Further divide A into blocks A_{ip}. As required, pack block A_{ip} into a contiguous buffer, so that by the end it is transposed and in the L2 cache.

3. Considering each block A_{ip} in turn, perform the multiplication $C_{i\bullet} := A_{ip}B_{p\bullet} + C_{i\bullet}$, with $B_{p\bullet}$ brought into the cache in column strips.

Additionally, it is possible in a multi-core system to coordinate the packing of $B_{p\bullet}$ between the processors, avoiding redundancy and improving performance.

Similar techniques using panels and blocks can be applied to Cholesky factorization (Buttari et al., 2009), but again these are not included in our implementation as the factorization of C is a relatively small part of the algorithm.

Returning to the SVM training problem, by casting the main computation of our algorithm in terms of matrix-matrix multiplications, we can take advantage of the above improvements for a multi-threaded architecture:

1. Consider a subblock of the constraint matrix A, consisting of all rows and the number of columns around the same size as $m+1$. Call this A_i.

2. Calculate L_{A_i} for this subblock, using (7.2b). This involves Level 1 operations, but these can be vectorized by the compiler.

3. Calculate $C := C + L_{A_i} A_i^T$ using the GEMM algorithm described above.

The performance gain of this approach is investigated in the next section.

7.2 Performance

The approach described below was implemented using the `OOPS` interior point solver (Gondzio and Grothey, 2007). In this section we compare a hybrid OpenMP/MPI version of our software with a version that only uses MPI, and also our implementation against three other parallel SVM solvers. Data sets are taken from the PASCAL Challenge on Large-scale Learning, and the sizes we used are shown in Table 7.1. Due to memory restrictions, we reduced the number of samples in the FD and DNA datasets. Additionally, the DNA data set was modified from categories to binary features, increasing m by a factor of 4. The data sets were converted into a simple feature representation in `SVM`$^{\texttt{light}}$ format[1]. The software was run on a cluster of quad-core 3GHz Intel Xeon processors, each with 2GB RAM. The GotoBLAS library was used for BLAS functions, with the number of OpenMP threads set to 4, to match the number of cores. We also used the LAM implementation of the MPI library.

Data set	n	m
Alpha	500000	500
Beta	500000	500
Gamma	500000	500
Delta	500000	500
Espilon	500000	2000
Zeta	500000	2000
FD	2560000	900
OCR	3500000	1156
DNA	6000000	800

Table 7.1: PASCAL Challenge on Large-scale Learning data sets used in this chapter.

To compare the hybrid approach (using the techniques described in Sections 7.1.1 and 7.1.2) with pure MPI (using Section 7.1.1 only), we used the data sets alpha to zeta. The results are shown in Figure 7.1. They consistently show that, although the pure MPI approach has

[1] The data file format is described on the webpage http://svmlight.joachims.org/.

better properties in terms of parallel efficiency, the hybrid approach is always computationally more efficient. We believe this is a result of the multi-core processor architecture. The cores are associated with relatively small local cache memories, and such an architecture demands a fine-grained parallelism where, to reduce bus traffic, an operation is split into tasks that operate on small portions of data (Buttari et al., 2009). OpenMP is better suited to this fine-grained parallelism.

Dataset	# cores	τ	OOPS	PGPDT	PSVM	Milde
Alpha	16	1	39	3673	1684	(80611)
		0.01	50	4269	4824	(85120)
Beta	16	1	120	5003	2390	(83407)
		0.01	48	4738	4816	(84194)
Gamma	16	1	44	—	1685	(83715)
		0.01	49	7915	4801	(84445)
Delta	16	1	40	—	1116	(57631)
		0.01	46	9492	4865	(84421)
Epsilon	32	1	730	—	17436	(58488)
		0.01	293	—	36319	(56984)
Zeta	32	1	544	—	14368	(22814)
		0.01	297	—	37283	(68059)
FD	32	1	3199	—	—	(39227)
		0.01	2152	—	—	(52408)
OCR	32	1	1361	—	—	(58307)
		0.01	1330	—	—	(36523)
DNA	48	1	2668	—	—	—
		0.01	6557	—	—	14821

Table 7.2: Comparison of parallel SVM training software on PASCAL data sets. Times are in seconds. In all cases except the DNA dataset, the Milde software ran but did not terminate within 24 hours of runtime, so the numbers in brackets show when it was within 1% of its final objective value; — indicates that the software failed to load the problem.

We made a comparison with other parallel software PGPDT (Zanni et al., 2006), PSVM (Chang et al., 2008), and Milde (Durdanovic et al., 2007), all using a linear kernel, and the results are shown in Table 7.2. With the exception of Milde (which has its own message passing implementation), the LAM implementation of the MPI library was used.

We required an objective value accuracy of $\epsilon = 0.01$, and chose two values for τ within the range set in the Challenge, so we believe the training tasks are representative. In keeping with the evaluation method of the Challenge, the timings shown are for training and do not include time spent reading the data. The PSVM algorithm includes an additional partial Cholesky factorization procedure, which we also do not include in the training times. To make the training equivalent, the rank of the factorization was set to be the number of features m. The Milde software includes a number of termination criteria but not one based on the

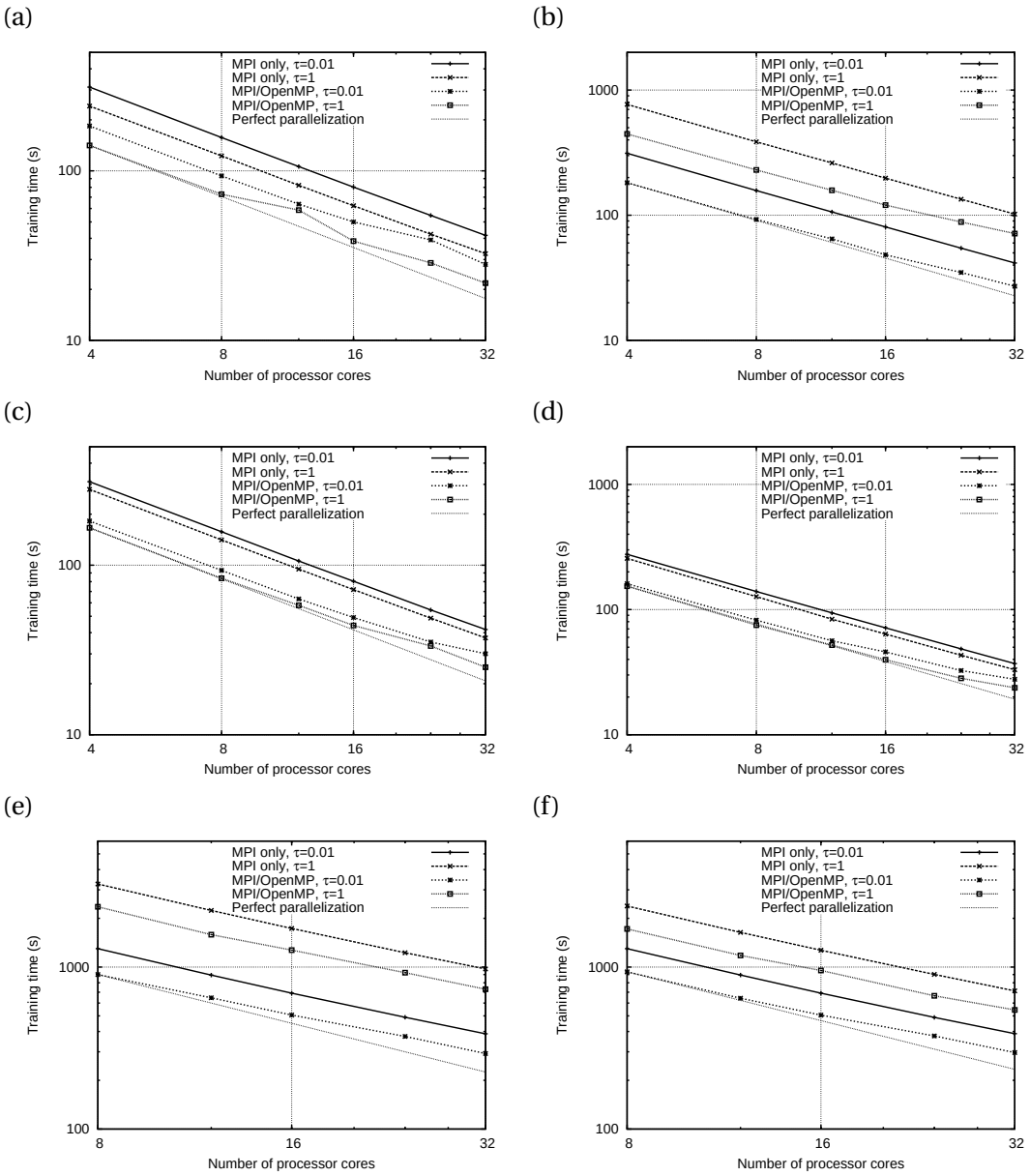


Figure 7.1: SVM training time with respect to the number of processors, for the PASCAL data sets (a) alpha, (b) beta, (c) gamma, (d) delta, (e) epsilon and (f) zeta. For each data set we trained using two values of τ. The results show that, although the pure MPI approach shows better parallel efficiency properties, the hybrid approach is always computationally more efficient.

Dataset	τ	OOPS			LibLinear		LaRank	
		n	# cores	Time	n	Time	n	Time
Alpha	1	500,000	16	39	500,000	147	500,000	3354
	0.01			50		112		2474
Beta	1	500,000	16	120	500,000	135	500,000	6372
	0.01			48		112		1880
Gamma	1	500,000	16	44	500,000	(8845)	500,000	—
	0.01			49		348		20318
Delta	1	500,000	16	40	500,000	(13266)	500,000	—
	0.01			46		429		—
Epsilon	1	500,000	32	730	250,000	316	500,000	5599
	0.01			293		265		2410
Zeta	1	500,000	32	544	250,000	278	500,000	—
	0.01			297		248		—
FD	1	2,560,000	32	3199	500,000	231	500,000	1537
	0.01			2152		193		332
OCR	1	3,500,000	32	1361	250,000	181	500,000	5695
	0.01			1330		121		4266
DNA	1	6,000,000	48	2668	600,000	144	600,000	300
	0.01			6557		30		407

Table 7.3: Comparison of our parallel SVM training software with linear SVM software LIBLINEAR and LARANK, again on PASCAL data sets. Times for training are in seconds. For the parallel software OOPS, each core had access to 2GB memory. The serial codes had access to 8GB memory, and the larger datasets were reduced in size to fit. For LIBLINEAR, brackets indicate that the iteration limit was reached. For LARANK, — indicates that the software did not terminate within 24 hours.

objective. Using the default criteria of maximum gradient below ϵ resulted in the software never terminating in all but one case within a 24 hour runtime limit. To be closer to the spirit of the Challenge, we show the time taken to be within 1% of the objective value at the end of 24 hours when the program was terminated prematurely, although in many cases the output indicated that the method was not yet converging.

The results show that our approach described in this chapter and implemented in OOPS is typically one to two orders of magnitude faster than the other parallel SVM solvers, terminates reliably, and training times are reasonably consistent for different values of τ.

Unfortunately there were no other linear SVM implementations in the Parallel track of the PASCAL Challenge. Instead, we show in Table 7.3 training time results for two linear SVM codes that did participate in the PASCAL Challenge: LIBLINEAR (Fan et al., 2008) which won the linear SVM track, and LARANK (Bordes et al., 2007) specialised for linear SVMs. Both codes ran serially, using the memory of 4 processor cores (8GB RAM in total). To make training possible, it was necessary to reduce the size of the larger datasets from the sizes given in Table 7.1; the number of samples used each time are shown as n in Table 7.3.

Dataset	OOPS	LibLinear	LaRank
Alpha	**0.1345**	0.1601	0.1606
Beta	**0.4988**	**0.4988**	0.5001
Gamma	**0.1174**	0.1185	0.1187
Delta	**0.1344**	0.1346	0.1355
Epsilon	**0.0341**	0.4935	0.4913
Zeta	**0.0115**	0.4931	0.4875
FD	**0.2274**	0.2654	0.3081
OCR	**0.1595**	0.1660	0.1681

Table 7.4: Accuracy measured using area under precision recall curve. These values are taken from the test results tables of the Evaluation pages of the PASCAL Challenge website.

The presentation of the results is slightly unusual in that training times are not directly connected to accuracy results against a test set. This is because labelled validation and test data sets have not been made publicly available. Performance statistics related to the precision recall curve were evaluated on the Challenge website, and so instead we reproduce in Table 7.4 the results from the website for area under the precision recall curve results for the test dataset, relating to LIBLINEAR, LARANK and our implementation. Additionally, Appendix A contains charts comparing our approach with a wider set of those linear L1-SVM approaches that participated. In general the precision of our method is consistent with the best of the other linear SVM methods that participated.

Taking the results in Tables 7.3 and 7.4 together, it is clear that LIBLINEAR in particular is a very efficient implementation, even in the cases Alpha to Delta where it is working with the full dataset. Table 7.4 clearly indicates that our approach consistently finds a high-quality solution to the separating hyperplane, measured in terms of classification accuracy, whereas for LIBLINEAR the quality of the solution is lower. In many cases, the reduction in prediction accuracy is only small. The results for the Epsilon and Zeta datasets in particular show, however, that for some problems the quality of LIBLINEAR's solution can be substantially worse.

7.3 Conclusions

In this chapter, we have shown how to develop a hybrid parallel implementation of linear Support Vector Machine training. The approach allows the entire data set to be used, and consists of following the steps:

1. Reformulating the problem to remove the dense Hessian matrix.

2. Using interior point method to solve the optimization problem in a predictable time,

and Cholesky decomposition to give good numerical stability of implicit inverses.

3. Exploiting the block structure of the augmented system matrix, to partition the data and linear algebra computations amongst parallel processing nodes efficiently.

4. Within SMP nodes, casting the main computations as matrix-matrix multiplication where possible, partitioning the matrices to obtain better data locality, and utilizing highly efficient BLAS implementation for a multi-threaded architecture.

The above steps were implemented in `OOPS`. Our results show that, for all cases, the hybrid implementation was faster than one using purely MPI, even though the MPI version had better parallel efficiency. We used the hybrid implementation to solve very large problems from the PASCAL Challenge on Large-scale Learning, of up to a few million data samples. On these problems the approach described in this paper was highly competitive, and showed that even on data sets of this size, training times in the order of minutes are possible. Our implementation, restricted to one SMP node, was evaluated as part of the PASCAL Challenge on Large-scale Learning, where it came top of the Parallel Track and won the award for "Best Student" overall.

The performance within the SMP nodes could be improved further, through developing a specialized matrix multiplication routine for calculating the matrix C. The symmetry of C could be exploited, and only the lower triangular section calculated. Furthermore, steps 2 and 3 of the procedure at the end of Section 7.1.2 could be combined. As $C = A_i H_i^{-1} A_i^T$, each panel of A_i would not need to be brought into the cache twice.

This discussion has been limited to linear kernels. In Chapter 9, the technique is extended to handle non-linear kernels, by first pre-processing using partial Cholesky factorization with pivoting.

The method described above requires all sample data to be loaded into memory. It is possible to improve data handling and increase the storage capacity somewhat, for instance storing the data compactly and expanding sections into floating point numbers when needed by the BLAS routines (Durdanovic et al., 2007), but the scaling is still $\mathcal{O}(nm^2)$. A more promising direction is to develop methods that are able to safely ignore or remove data points from consideration as the algorithm progresses, at the same time as exploiting structure. This is explored in the next chapter.

Chapter 8

Reducing the number of samples

The algorithms we have described so far (Chapters 6 and 7) consider all samples at each iteration. They therefore have a computational complexity of $\mathcal{O}(n)$. Improving on this result to give sublinear scaling can only be achieved by reducing the number of samples considered, either initially or as the algorithm progresses.

It has been known for some time (Pontil and Verri, 1997) that only around $m+1$ support vectors are needed to define the hyperplane margins. For large-scale problems, the vast majority of samples will either be on the correct side of the margin or in margin error. The set of samples $\mathcal{S}$ can be partitioned into a *basic* set $\mathcal{B}$, a *non-basic* set $\mathcal{N}$, and an *upper-bounded* set $\mathcal{U}$, so $\mathcal{B} \cup \mathcal{N} \cup \mathcal{U} = \mathcal{S}$. The sets $\mathcal{B}, \mathcal{N}$, and $\mathcal{U}$ were introduced in Chapter 2.

The relationship between the margin error ξ_i and the dual variable z_i is well understood (for example, see Schölkopf and Smola, 2002, Table 7.2). The connection to set membership is summarized in Table 8.1. Identifying the optimal set partitioning is arguably the most important aspect of solving the problem. Once the partitioning is known, the problem reduces to a QP problem involving just the small number of variables in $\mathcal{B}$. The active set methods described in Chapter 4 spend the vast majority of time identifying the correct partitioning.

Position of sample	ξ_i	z_i	Set	Description
Correct side of hyperplane margin	$\xi_i = 0$	$z_i = 0$	$\mathcal{N}$	Not a SV
On hyperplane margin	$\xi_i = 0$	$0 < z_i < \tau$	$\mathcal{B}$	In-bound SV
Wrong side of hyperplane margin	$\xi_i > 0$	$z_i = \tau$	$\mathcal{U}$	Bound SV

Table 8.1: Using ξ_i and z_i values to partition samples.

This chapter draws on several sources. The first is SVM decomposition (Osuna et al., 1997), an active set method described in Chapter 4. A guess is made of the optimal set partitioning,

and samples are chosen for the working set. From the solution of the subproblem, a new working set is chosen. The process continues until the correct set partitioning is found. If the size of the working set is limited, this approach is able to work on any size of data set.

Column generation is another decomposition approach, closely related to cutting planes (Cheney and Goldstein, 1959; Kelley, 1960). It is attractive for problems where the number of columns in the constraint matrix greatly exceeds the number of rows, or where the work involved in pricing columns is either very high or involves specialist algorithms. The idea of column generation is to work only with a sufficiently meaningful subset of variables, forming the *restricted master problem* (RMP). Classically, these variables are not the ones of the original problem, but a set of extreme points and rays in solution space (Dantzig and Wolfe, 1960). Other possible solutions can be unambiguously described as a convex combination of these points. An iteration of a column generation algorithm consists of first solving the RMP to determine the value of the dual variables λ, and then secondly of identifying new columns (possible solutions) that look attractive and adding them as primal variables.

There are known serious computational difficulties with this approach. The method finds an approximate solution quickly by cutting out large sections of the solution space, but then it takes a long time to find the optimal set partitioning. This is known as the *long tail effect* (Gilmore and Gomory, 1963). While the primal objective value monotonically decreases to its optimum value, practical experience has shown that the dual solution oscillates wildly. Several techniques have been proposed to stabilize the algorithm, including reduced step size (Wentges, 1997) or regularized bundle methods (Kiwiel, 1990). Relevant surveys can be found in Goffin and Vial (1999); Lemaréchal (2001); Lübbecke and Desrosiers (2005). Column generation has historically been closely connected with the Simplex method. At every iteration, therefore, the subproblem is solved to optimality to obtain a vertex solution, and a single new column is added to the basis. Gondzio and Sarkissian (1996) demonstrate advantages of using primal-dual interior point methods in the context of column generation. Using the notion of $\mu-$centres, it is possible to solve restricted master problems in the earlier iterations to a lower tolerance to quickly build a rough approximation. The adaptation to IPM also enables many columns to be added at one time.

Returning to SVMs, Jung et al. (2008) investigated reducing the number of samples considered by the solver. They use the primal formulation and IPM method of Gertz and Griffin (2009), so this is equivalent to a reduction in constraints. Complementarity pair ratios are used to indicate if the constraint is active. This information is used at each IPM iteration to remove inactive constraints. The same technique can be applied to our formulation (5.1)

resulting in a reduction in columns, but a disadvantage is that removing columns will disrupt the contiguous data storage, and it will no longer be possible to use BLAS Level 3 operations. Our results (Table 6.1) show that a large proportion of columns will need to be identified at each iteration to compensate for the loss of efficient cache handling, although the loss could be offset somewhat by repacking the data contiguously at each iteration.

Before describing our approach, it is useful to consider the characteristics of SVM training problems. The standard dual formulation (3.6) has n variables and only one constraint, while our separable formulation (5.1) has $n+m$ variables and $m+1$ constraints. Both therefore have relative dimensions attractive for a column generation approach. As well as a large number of variables, it is to be expected that a large proportion of them are at their upper bounds. For example, Keerthi and DeCoste (2005) used five real-world data sets. In all, the ratio of the number of support vectors to the number of training examples was a substantial fraction, and for three of them it was between 0.6 and 0.75. This is typical of results in the literature. Steinwart (2004) provides the theoretical result that the number of support vectors is a fraction of the number of samples.

In this chapter we apply active set, column generation and sample reduction ideas to the IPM-based method used in Chapter 6, with the aim of improving the scalability of the algorithm. The next section outlines the algorithm. It explores techniques for determining an initial set partition, for subsequent repartitioning, and criteria for terminating. In Section 8.2 we describe the analogy with rigid body dynamics, to provide insight into the behaviour of the algorithm. Section 8.3 contains some numerical results, before the conclusion in Section 8.4.

8.1 A reduced column interior point approach

Taking the separable formulation (5.1), we partition the variables z into a working set $\mathcal{W}$, and sets of fixed variables $\mathcal{N}$ and $\mathcal{U}$ as described above, to give the *reduced master problem*:

$$\begin{aligned} \min_{w,z_{\mathcal{W}}} \quad & \frac{1}{2}w^T w - (e_{\mathcal{W}}^T z_{\mathcal{W}} + e_{\mathcal{N}}^T z_{\mathcal{N}} + e_{\mathcal{U}}^T z_{\mathcal{U}}) \\ \text{s.t.} \quad & w - ((XY)_{\mathcal{W}} z_{\mathcal{W}} + (XY)_{\mathcal{N}} z_{\mathcal{N}} + (XY)_{\mathcal{U}} z_{\mathcal{U}}) = 0 \\ & y_{\mathcal{W}}^T z_{\mathcal{W}} + y_{\mathcal{N}}^T z_{\mathcal{N}} + y_{\mathcal{U}}^T z_{\mathcal{U}} = 0 \\ & w \text{ free}, \quad 0 \le z_{\mathcal{W}} \le \tau e_{\mathcal{W}}, \quad z_{\mathcal{N}} = 0, \quad z_{\mathcal{U}} = \tau e_{\mathcal{U}}. \end{aligned} \tag{8.1}$$

The vectors $e_{\mathcal{W}}$, $e_{\mathcal{N}}$, $e_{\mathcal{U}}$ have dimensions equal to the size of the sets indicated. The RMP is equivalent to the original problem if $\mathcal{W} \supseteq \mathcal{B}^*$, $\mathcal{N} \subseteq \mathcal{N}^*$ and $\mathcal{U} \subseteq \mathcal{U}^*$, where $\mathcal{B}^*$, $\mathcal{N}^*$ and $\mathcal{U}^*$

are the sets at optimality. Our aim is to find this partitioning. To highlight that the working set can be larger than $|\mathscr{B}^*|$, contain columns from $\mathscr{N}^*$ and $\mathscr{U}^*$ and still be optimal, we use the notation $\mathscr{W}$ rather than $\mathscr{B}$ for the working set. For size, we typically used $|\mathscr{W}| \approx 10m$.

Forming the Lagrangian function from the objective function and equality constraints of (8.1) gives

$$\mathscr{L}(w, z, \lambda, \lambda_0) = \frac{1}{2} w^T w - e^T z - \lambda^T (w - XYz) + \lambda_0 (y^T z),$$

where λ and λ_0 are dual variables corresponding to the constraints. The vector $\nabla_z L$ is known as the *reduced costs*, where

$$\frac{\partial \mathscr{L}}{\partial z_i} = -1 + y_i x_i^T \lambda + y_i \lambda_0.$$

Reduced costs are calculated using the values of (λ, λ_0) from the RMP. If for an element i, the reduced cost $\frac{\partial \mathscr{L}}{\partial z_i} < 0$, then increasing variable z_i will reduce the objective value. This will only be possible for variables that have not reached the upper bound of their box constraint, in other words variables that are in the sets $\mathscr{B}$ or $\mathscr{N}$. Therefore, variables in $\mathscr{N}$ with negative reduced costs need to be moved to either $\mathscr{W}$ or $\mathscr{U}$ before the optimal partitioning can be found. Similarly, if an element i is in either $\mathscr{B}$ or $\mathscr{U}$, and the reduced cost $\frac{\partial \mathscr{L}}{\partial z_i} > 0$, then decreasing the variable z_i will reduce the objective value. Note that through the KKT relationship between (λ, λ_0) and (w, w_0), if $\frac{\partial \mathscr{L}}{\partial z_i} < 0$, then $\xi_i = -\frac{\partial \mathscr{L}}{\partial z_i}$, where ξ_i is the misclassification error. It is necessary to solve the RMP to optimality to ensure that the KKT relationship holds.

A prototype algorithm is given as Algorithm 4. We discuss approaches to each step below.

Algorithm 4 Prototype column generation algorithm

1: Determine an initial partition $\mathscr{W}, \mathscr{N}$ and $\mathscr{U}$
2: **repeat**
3: Solve the RMP to obtain dual variables (λ, λ_0)
4: Generate new partition using (λ, λ_0)
5: **until** termination criteria met

8.1.1 Initial partition

A classical approach involves making a random selection of columns to enter the working set $\mathscr{W}$, and placing all other variables in $\mathscr{N}$. This initial selection can lead to a misleading solution for the dual variables. A good approximation of the dual variables will provide a more accurate choice of columns. We propose to use a least squares approximation of the QP (5.1). The aim is to find a point that satisfies the equality constraints, while at the same

time promoting points near the centre of the feasible box.

$$\begin{array}{ll} \min\limits_{w,z} & \frac{1}{2}\left(w^T w + z^T z + (\tau e - z)^T (\tau e - z)\right) \\ \text{s.t.} & w - XYz = 0 \\ & y^T z = 0 \\ & w,\, z \text{ free.} \end{array} \tag{8.2}$$

The optimal solution of (8.2) satisfies the following system of linear equations:

$$\begin{bmatrix} I_m & 0 & -I_m & 0 \\ 0 & 2I_n & (XY)^T & -y \\ -I_m & XY & 0 & 0 \\ 0 & -y^T & 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ z \\ \lambda \\ \lambda_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \tau e \\ 0 \\ 0 \end{bmatrix} \tag{8.3}$$

This problem has a structure similar to (2.14), the starting point due to Mehrotra described in Section 2.3.2. Like (2.14), the advantage of using (8.3) is that the data structure is the same as that of augmented system (2.9) used in the main IPM algorithm. Note that z of (5.1) should satisfy $z \in [0, \tau e]$, and relaxation (8.2) can produce points outside of the box constraint. As discussed in Section 2.3.2, this needs to be corrected if the solution is to be used as a starting point for the IPM algorithm. But here as an approximation scheme, we use the dual variable values directly without correcting for this infeasibility.

To investigate the two schemes for finding an initial hyperplane, we used the standard data sets Adult, MNIST, SensIT and USPS.[1] For the scheme based on a random selection of columns, we chose the first $10m$ columns of the data set to enter the working set and solve (8.1). The approximation scheme solves (8.2) on the entire data set. In all cases, $\tau = 1$ and a terminating objective value precision of 0.01 were used. Results are shown in Table 8.2. For each data set, we compare the hyperplane of the approximation to the hyperplane obtained by the full SVM training problem, and show the angle between the norms, the ratio in $\|w\|_2$ (the reciprocal of which is the ratio of margin widths), and the ratio in the hyperplane bias w_0. Also shown is the prediction accuracy of each hyperplane on an unseen test set. The results show that both approximation schemes are able to produce good separating hyperplanes, as the prediction accuracy of each is close to that of the optimal SVM hyperplane. Using the

[1] All data sets and test sets are available from the LIBSVM collection at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. For MNIST and USPS, the classification task was to discriminate digits 0–4 from 5–9. For SensIT, the task was to discriminate category 3 from the other two.

first $10m$ samples gave a better approximation of the margin width and the final value of w_0 in each cases, but solving (8.2) gave a separating hyperplane more closely aligned with the optimal one.

		Optimal	Approx using (8.2)	Approx using $10m$ samples
Adult	Angle	0°	31.9°	61.9°
	$\|w\|_2$ ratio	1.000	0.299	1.024
	w_0 ratio	1.000	0.218	0.892
	Prediction	0.850	0.846	0.840
MNIST	Angle	0°	33.7°	50.9°
	$\|w\|_2$ ratio	1.000	0.266	0.816
	w_0 ratio	1.000	0.283	0.903
	Prediction	0.878	0.864	0.866
Sensit	Angle	0°	18.3°	63.1°
	$\|w\|_2$ ratio	1.000	0.411	1.008
	w_0 ratio	1.000	0.701	0.933
	Prediction	0.856	0.858	0.854
USPS	Angle	0°	34.4°	46.3°
	$\|w\|_2$ ratio	1.000	0.408	0.964
	w_0 ratio	1.000	0.299	0.663
	Prediction	0.867	0.865	0.860

Table 8.2: Two approximation schemes are compared to the optimal hyperplane found by solving (5.1) on the whole data set. For each data set, the rows give the angle between weight vectors w, the ratio in $\|w\|_2$ (the reciprocal of margin width), the ratio in the hyperplane bias w_0, and the prediction accuracy of each hyperplane on an unseen test set.

8.1.2 Repartitioning the variables

Reduced costs are used to choose new columns. One technique that we could use is to repartition the variables using the scheme:

i in set $\mathcal{U}$ and $\frac{\partial \mathcal{L}}{\partial z_i} > +\epsilon_0 \Rightarrow$ Move i to set $\mathcal{W}$

i in set $\mathcal{N}$ and $\frac{\partial \mathcal{L}}{\partial z_i} < -\epsilon_0 \Rightarrow$ Move i to set $\mathcal{W}$

i in set $\mathcal{W}$ and $\frac{\partial \mathcal{L}}{\partial z_i} < 0 \Rightarrow$ Move i to set $\mathcal{U}$

i in set $\mathcal{W}$ and $\frac{\partial \mathcal{L}}{\partial z_i} > 0 \Rightarrow$ Move i to set $\mathcal{N}$

Here, all variables that are wrongly partitioned in $\mathcal{N}$ or $\mathcal{U}$ are moved first into the working set $\mathcal{W}$, and then to $\mathcal{U}$ or $\mathcal{N}$, if required, at the next iteration. $\epsilon_0 = 10^{-4}$ is used to reduce changes in set membership due to numerical errors. The disadvantage of this approach is that, as we are expecting $|\mathcal{U}^*|$ to be some sizeable fraction of n, the number of iterations required of the algorithm will be $\mathcal{O}(n)$.

For sublinear scaling, the algorithm needs to allow variables to move directly between $\mathcal{N}$ and $\mathcal{U}$. The approach we use is to have thresholds $\epsilon_{\mathcal{N}}$ and $\epsilon_{\mathcal{U}}$ on the reduced costs. Values

of $\epsilon_{\mathcal{N}} = 0.2$ and $\epsilon_{\mathcal{U}} = 1.0$ worked well in practice. This approach seems reasonable as the reduced cost measures the distance from the hyperplane margin, and a large distance will give us confidence in allocating the variable to the set according to the scheme:

$$\begin{aligned}
&\text{If} \quad i \text{ in set } \mathcal{U} \text{ and } \tfrac{\partial \mathcal{L}}{\partial z_i} > \epsilon_{\mathcal{N}} \quad \Rightarrow \text{Move } i \text{ to set } \mathcal{N},\\
&\text{else if } i \text{ in set } \mathcal{U} \text{ and } \tfrac{\partial \mathcal{L}}{\partial z_i} > \epsilon_0 \quad \Rightarrow \text{Move } i \text{ to set } \mathcal{W}.\\
&\text{If} \quad i \text{ in set } \mathcal{N} \text{ and } \tfrac{\partial \mathcal{L}}{\partial z_i} < -\epsilon_{\mathcal{U}} \Rightarrow \text{Move } i \text{ to set } \mathcal{U},\\
&\text{else if } i \text{ in set } \mathcal{N} \text{ and } \tfrac{\partial \mathcal{L}}{\partial z_i} < \epsilon_0 \quad \Rightarrow \text{Move } i \text{ to set } \mathcal{W}.
\end{aligned}$$

Jung et al. (2008) show that an effective way to remove samples (constraints in their formulation, variables in ours) is to consider the ratio of complementarity pairs. Considering the KKT conditions (2.4c) and (2.4d), the ratio $\frac{z_i}{s_i} \to 0$ is an indication that $i \in \mathcal{N}^*$, while $\frac{\tau - z_i}{v_i} \to 0$ is an indication that $i \in \mathcal{U}^*$. An analysis of different indicator functions in the context of interior point methods is contained in El-Bakry et al. (1994), including Tapia indicators that use information on changes in variables from one interior point iteration to the next. We used the measure $\frac{z_i}{s_i} \cdot \frac{v_i}{(\tau - z_i)}$ to compare these two ratios and decide the new set for i according to the scheme:

$$\begin{aligned}
&\text{If} \quad \tfrac{z_i}{s_i} \cdot \tfrac{v_i}{(\tau - z_i)} > \rho \Rightarrow \text{Move } i \text{ to set } \mathcal{U},\\
&\text{else if} \quad \tfrac{z_i}{s_i} \cdot \tfrac{v_i}{(\tau - z_i)} < \tfrac{1}{\rho} \Rightarrow \text{Move } i \text{ to set } \mathcal{N}.
\end{aligned}$$

Otherwise, the column i is kept in the working set. In practice, $\rho = 10^2$ was found to be a good indication level.

An approach that allows the variables to be repartitioned is nearer to an active set method than to a column generation method, as previous solutions do not remain in the working set. We considered that the computational cost of keeping all previous columns in the working set is unacceptably high, compared to solving the QP (5.1) directly.

8.1.3 Termination criteria

The optimal solution is reached when an optimal set partition is found, in other words there are no elements in $\mathcal{N}$ with negative reduced costs, and none in $\mathcal{U}$ where the reduced costs are positive. Unfortunately, continuing the algorithm until this stage is reached invites the long-tail effect mentioned in the introduction. In the SVM context, this level of precision is not necessary, as the real goal is to find a good hyperplane, defined by the dual variables.

An alternative is to consider the gap between upper and lower bounds on the objective function. Primal objective value is obtained from the RMP, and gives an upper bound on the full problem (further columns will only be used if they reduce the objective value). A lower bound can be found by calculating the objective of (3.4) using the dual variables to

define the hyperplane variables (w, w_0), and the misclassification error $\xi_i = \max(-\frac{\partial \mathscr{L}}{\partial z_i}, 0)$. The algorithm terminates when the difference between the two objective values is small.

Another possibility is to terminate when the hyperplane is stable, but as noted above with column generation, stability in the dual solution is not something we should expect.

8.2 Rigid body dynamics

There is an exact analogy between the support vector machine classification problem and the field of physics known as rigid body dynamics, which concerns forces acting on an object that cannot change its shape. The link is well known, and is indeed where the term "support vector" comes from. The analogy provides useful insights into both the reductions we use and the instabilities they cause, so before we discuss how the method performs in practice, let us describe this analogy, building on the interpretation provided in Schölkopf and Smola (2002, section 1.4).

A rigid body occupies a volume of space and has a particular shape that cannot change. It is useful to imagine the body being made up of many particles, where each particle i has mass m_i and position r_i. Forces act on the body, and we imagine that there is a particle i at exactly the point where the force vector f_i acts. Due to its rigidity, the body can undergo only translation (a change in position) and rotation. The total external force f acting on the body is the sum of the individual forces f_i. If $f \neq 0$ the resulting force is *unbalanced*, and this will cause a change in the position of the body.

Forces can also cause the body to rotate; this is described as torque. The torque vector depends on the point where it is measured, and it can be thought of as the axis of rotation around that point. The torque at the point r_0 due to the force f_i is the moment of f_i around r_0.

An important concept is centre of mass r_{CM}, which allows many particles to be represented by a single composite mass. The position of the centre of mass is defined as

$$r_{CM} \equiv \frac{\sum m_i r_i}{\sum m_i}.$$

The resultant force f can be thought of as acting at this point. If torque is measured at r_{CM}, it removes any consideration of forces causing translation, and forces f_i acting through r_{CM} do not exert any torque. If the resultant force f and the torque at r_{CM} are both zero, the rigid body is in *equilibrium*, meaning that it is mechanically stable.

Applying the analogy to SVMs, we can think of each support vector x_i exerting a perpendicular force of size z_i on a solid sheet lying along the hyperplane, pulling the hyperplane towards the point x_i in order that the point moves to the correct side of the hyperplane margin. Note that support vectors have $z_i > 0$. There are some unusual properties with this analogy. Each force f_i acts parallel to the normal of the hyperplane. Only points that lie on or are on the wrong side of the hyperplane margin exert a force, and so according to our analogy only these points have mass m_i. We can consider that m_i is proportional to z_i. On the hyperplane, the force may be any value within the bounds. But away from the hyperplane, the force is of fixed size, independent of the distance to the hyperplane.

The conditions for equilibrium of the rigid body are encapsulated in the constraints of (5.1). Let the data set be divided into a positive class $\{i : y_i = +1\}$ and a negative class $\{i : y_i = -1\}$, with $X^{(+)}$, $X^{(-)}$ the data and $z^{(+)}$, $z^{(-)}$ the dual variables for each class. The constraint (5.1c) can be rewritten as

$$e^T z^{(+)} = e^T z^{(-)},$$

and so this constraint describes that the forces from the positive and negative classes must be balanced, resulting in no translation of the hyperplane. Considering the constraint (5.1b), the position $r_{CM}^{(+)} \equiv X^{(+)} z^{(+)}$ can be thought of as a centre of mass for positive class support vectors, and similarly $r_{CM}^{(-)} \equiv X^{(-)} z^{(-)}$ for negative class support vectors. The centre of mass of the whole system will lie on a line between these two points. The constraint (5.1b) can be rewritten as

$$w = X^{(+)} z^{(+)} - X^{(-)} z^{(-)},$$

so the points r_{CM}, $r_{CM}^{(+)}$ and $r_{CM}^{(-)}$ are aligned along the hyperplane normal w. As the force at $r_{CM}^{(+)}$ is in the direction w, and the force at $r_{CM}^{(-)}$ is in the direction $-w$, both will act through r_{CM}, resulting in no torque, so the second condition for equilibrium is met.

The analogy provides several insights. It shows clearly that the position of the hyperplane will be affected if the class sizes are unbalanced. Outliers have a large effect in terms of torque if they are close to the margin but far from the centre of mass, rather than far from the hyperplane. Most importantly, if new masses are introduced into the system that are not in the line connecting $r_{CM}^{(+)}$ and $r_{CM}^{(-)}$, they will exert torque on the hyperplane regardless of how small the masses are in comparison to the centres $r_{CM}^{(+)}$ and $r_{CM}^{(-)}$. This is the analogy of new columns being added to the working set. The torque can only be balanced, and the new

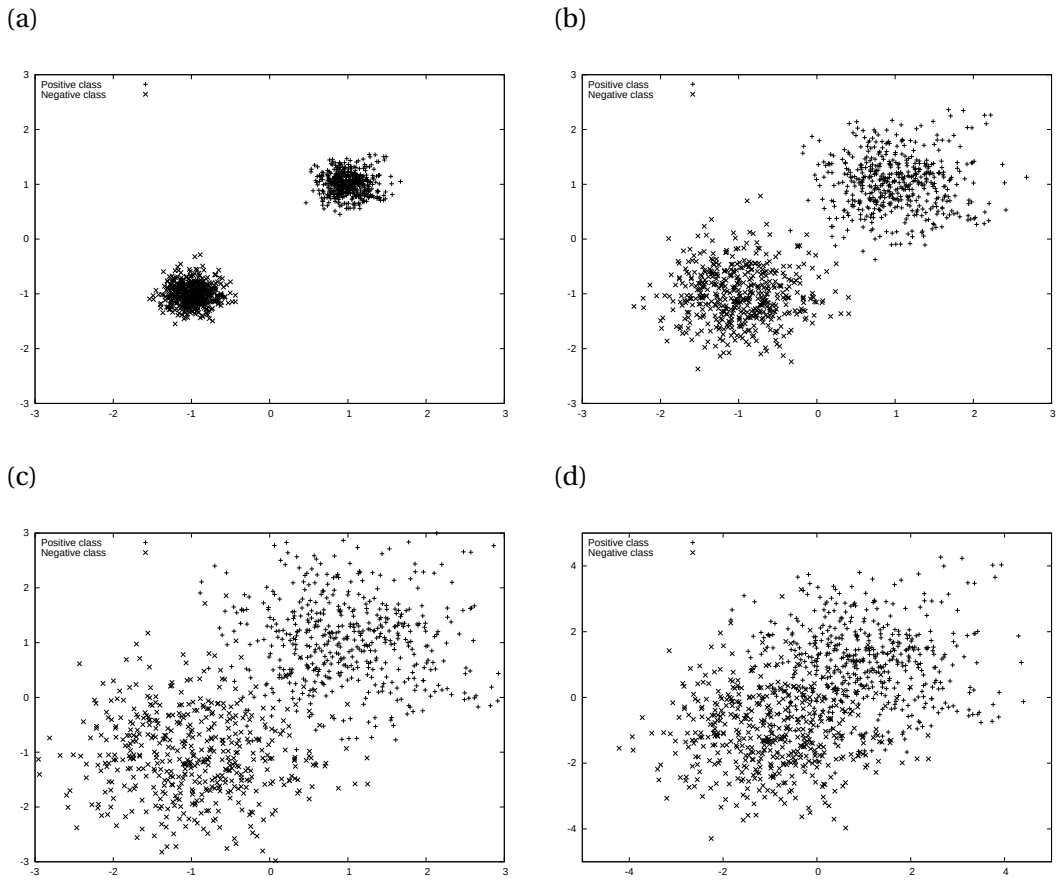


Figure 8.1: Data sets for $m = 2$, containing two normally-distributed clusters. (a) $\sigma = 0.2$ gives two clusters that are clearly linearly separable. (b) With $\sigma = 0.5$, there are still two obvious clusters but they are no longer linearly separable. (c) $\sigma = 0.8$ and (d) $\sigma = 1.2$ cause the two clusters to overlap considerably.

hyperplane determined, by other point masses in the working set.

8.3 Numerical performance

To investigate the behaviour and performance of the algorithm, we created data sets where each class was clustered. The two clusters were each distributed normally in $\mathbb{R}^m$, with zero correlation between the m features. By adjusting the standard deviation, we could control the overlap between the two clusters. We also varied the level of mislabelling of samples to introduce noise into the data set. Figures 8.1 and 8.2 show some small data sets of this form where $m = 2$.

We investigated the orientation of the hyperplane using such data sets with $m = 2$ and a maximum working set size of 20. Figure 8.3 shows how the hyperplane normal changes at each iteration compared to the optimal one. Note that the margin width and the position

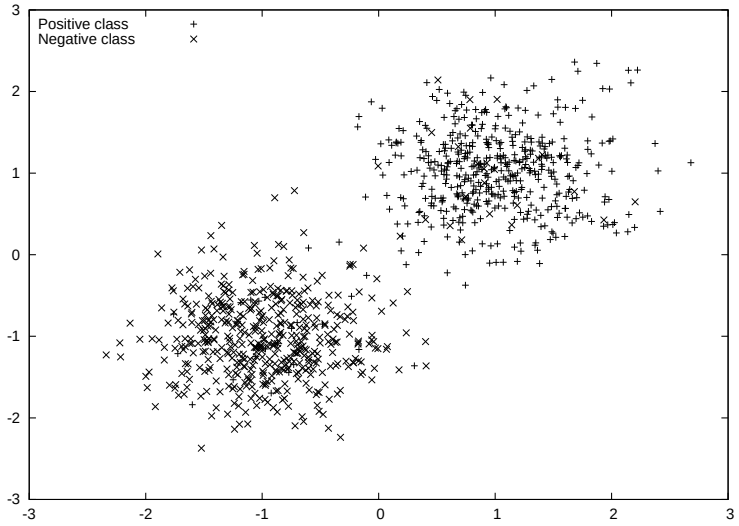


Figure 8.2: Data set where $m = 2$, $\sigma = 0.5$ and 5% of samples are mislabelled.

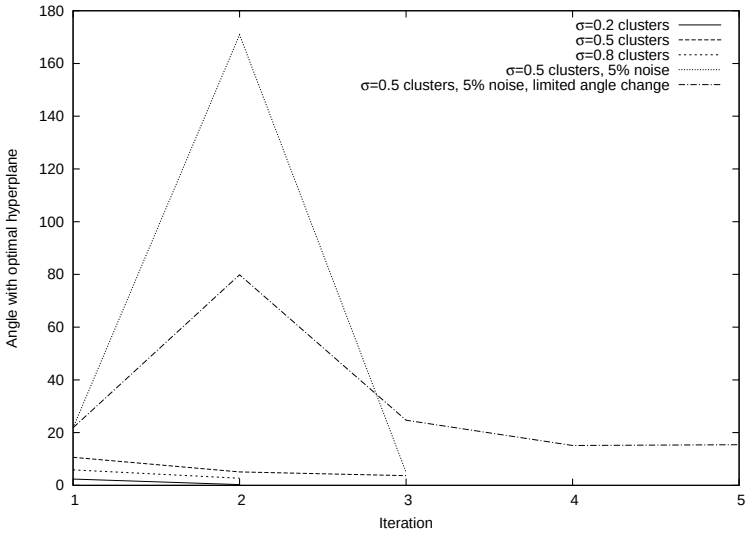


Figure 8.3: Angle of the hyperplane normal at each iteration to the optimal one, for data sets shown in Figures 8.1 and 8.2.

w_0 are also changing at each iteration, and both will affect the working set. For the data sets with no noise, the method found a good approximation to the hyperplane quickly, and made progress towards the optimal one. This was true even for the data sets with overlapping clusters. With the noisy data set (Figure 8.2), the hyperplane flipped almost completely at the second iteration. The reason for this behaviour can be understood by considering the rigid body dynamics analogy. After the first iteration, the system has been reduced to two larger masses at $r_{CM}^{(+)}$ and $r_{CM}^{(-)}$, plus a few smaller point masses that represent the variables in the working set. The masses at $r_{CM}^{(+)}$ and $r_{CM}^{(-)}$ represent relatively large misclassification errors compared to the working set variables. Inverting the hyperplane will reduce the misclassification error for $r_{CM}^{(+)}$ and $r_{CM}^{(-)}$ to zero. As the correctly classified samples in $\mathcal{N}$ are not part of the system, the cost of the change is not seen until the next iteration. We reduced this behaviour by including columns that represent "centres of mass" for the variables in $\mathcal{N}$, one for each class. This had the effect of limiting the change in the hyperplane angle to 90°, although it did not help the hyperplane settle to its optimal value.

As described in Section 8.2, the hyperplane can also be unstable if the working set changes, as new columns introduce torque into a finely balanced system. We found that introducing some samples from $\mathcal{N}$ that lie close to the margin helped.

To more closely represent real-world large-scale data sets, we created similarly-clustered data sets with $m = 100, n = 100,000$. Results for these data sets using $\tau = 1$ are shown in Table 8.3. For the data sets with overlapping classes, the algorithm found it hard to determine the optimal hyperplane with the limited information available at each iteration. A larger working set and some tuning of $\epsilon_{\mathcal{N}}$ were required to make the algorithm terminate.

Better results were obtained when we used the approximation (8.3) on the whole data set, and chose samples with the highest reduced costs to form the initial working set. For each of the data sets, only one iteration (Algorithm 4, step 2) was required to find a solution within tolerance. Compared to training using the whole data set, speed-ups of $3-5$ times were achieved.

The above observations were incorporated into Algorithm 5. Decision regions are illustrated in Figure 8.4. The algorithm was applied to the real world data sets used above. These data sets have similar linear separability to each other, at 85%–88% prediction accuracy. They are also unlikely to have the simple and symmetrical clusters of our constructed data sets. Results are shown in Table 8.4, comparing Algorithm 5 to training using full data. With Algorithm 5, it was necessary to use a working set size of around $\frac{n}{2}$ to ensure only one iteration of the algorithm was required. Choosing the maximum working set sizes given in Table 8.4

Data set		Full data	First cols as WS		Solve (8.3) for WS		WS size
σ	Noise	Time	Time	Its	Time	Its	
0.2	0	4.06	0.14	1	0.95	1	1000
0.5	0	5.25	1.11	3	0.97	1	1000
0.5	0.05	5.20	2.26	5	0.94	1	1000
0.8	0	4.03	1.77	4	1.10	1	1000
1.2	0	4.05	12.38	20	1.18	1	2000
1.5	0	4.64	25.95	27	1.32	1	3000

Table 8.3: SVM training of data sets comprising two clusters, each cluster normally distributed with standard deviation σ. The *noise* gives the proportion of samples mislabelled. The *Full data* column gives times for solving (5.1) on the whole data. The other two methods are implementations of Algorithm 4. The first fills the initial working set with the first samples of the data set. The second solves (8.3) for the whole data set, and chooses the samples with the highest reduced costs to form the initial working set. Times are in seconds. *Its* is the number of outer iterations of Algorithm 4 required. Working set sizes for both methods are given in the final column.

ensured that all columns with reduced costs worse than -0.2 were included. In practice, we are able to remove around half the columns and obtain an answer of comparable accuracy, with a typical time saving of around 25%.

Algorithm 5 Reduced column algorithm

Require: Data set $\mathcal{S}$

1: $\mathcal{N} = \mathcal{S}, \mathcal{W} = \mathcal{U} = \{\varnothing\}$
2: Solve (8.3) using the whole data set to obtain (w, w_0)
3: $(\lambda, \lambda_0) := (w, w_0)$
4: Move columns $i \in \mathcal{N}$ to $\mathcal{U}$ $\forall i : \frac{\partial \mathcal{L}}{\partial z_i} < -\epsilon_{\mathcal{U}}$
5: Choose columns $i \in \mathcal{N} : \frac{\partial \mathcal{L}}{\partial z_i} \geq -\epsilon_{\mathcal{U}}$ with the largest reduced cost $-\frac{\partial \mathcal{L}}{\partial z_i}$, and move them from $\mathcal{N}$ to $\mathcal{W}$ until the working set capacity is reached.
6: **repeat**
7: Solve the RMP (8.1) to obtain dual variables (λ, λ_0)
8: Determine new partition:
Move columns $i \in \mathcal{W}$ to $\mathcal{U}$ $\forall i : \frac{z_i}{s_i} \cdot \frac{v_i}{(\tau - z_i)} > \rho$
Move columns $i \in \mathcal{W}$ to $\mathcal{N}$ $\forall i : \frac{z_i}{s_i} \cdot \frac{v_i}{(\tau - z_i)} < \frac{1}{\rho}$
Move columns $i \in \mathcal{N}$ to $\mathcal{U}$ $\forall i : \frac{\partial \mathcal{L}}{\partial z_i} < -\epsilon_{\mathcal{U}}$
Move columns $i \in \mathcal{U}$ to $\mathcal{N}$ $\forall i : \frac{\partial \mathcal{L}}{\partial z_i} > \epsilon_{\mathcal{N}}$
Choose columns $i \in \left\{\mathcal{N} : \frac{\partial \mathcal{L}}{\partial z_i} < \epsilon_0\right\} \cup \left\{\mathcal{U} : \frac{\partial \mathcal{L}}{\partial z_i} > \epsilon_0\right\}$ with the largest reduced cost $|\frac{\partial \mathcal{L}}{\partial z_i}|$, and move to $\mathcal{W}$ until the working set capacity is reached.
9: **until** $\frac{\text{duality gap}}{\text{primal objective}} <$ tolerance

8.4 Discussion

In this chapter, we have investigated an approach for reducing the sample size considered by the IPM algorithm. The SVM training QP has a very small number of active variables, making

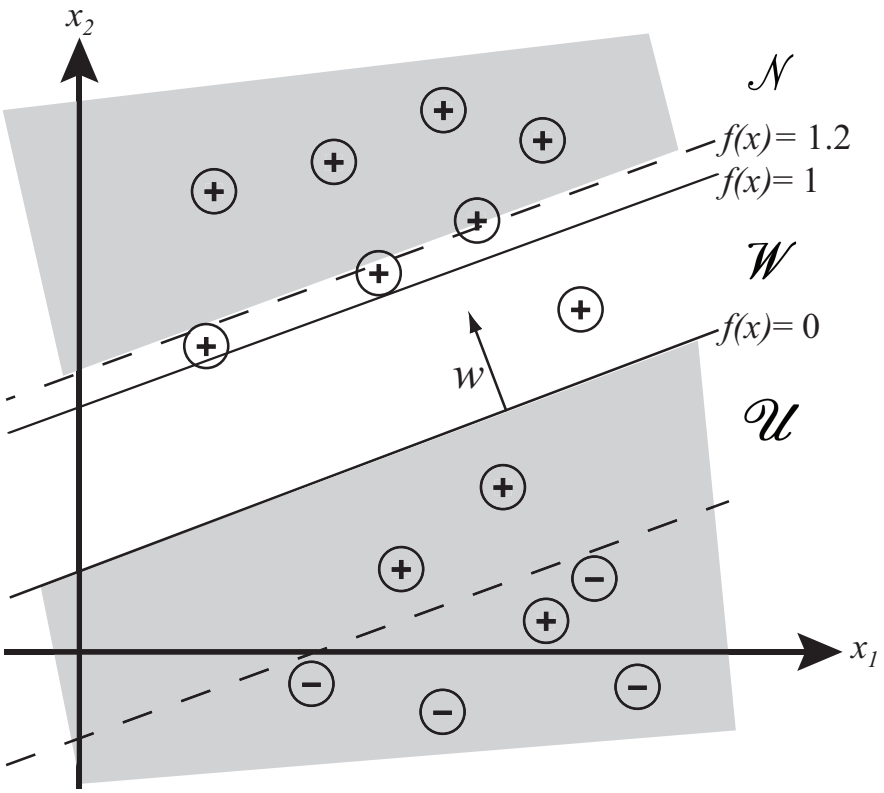


Figure 8.4: Decisions regions used by Algorithm 5 to assign positive class samples $i \notin \mathcal{W}$ to $\mathcal{N}, \mathcal{W}$ or $\mathcal{U}$. Decision regions for the negative class samples are of course the reflection of these around $f(x) = 0$.

Data set		Full data		Column reduction			
Name	$(n \times m)$	Time	Accuracy	Time	Accuracy	Its	WS size
Adult	(32561×123)	2.61	0.850	1.94	0.848	1	15000
MNIST	(60000×780)	71.87	0.878	52.65	0.874	1	35000
Sensit	(78823×100)	4.37	0.856	4.68	0.863	1	35000
USPS	(7291×256)	1.65	0.867	1.03	0.867	1	3700

Table 8.4: Applying Algorithm 5 to real world data sets.

column generation decomposition methods look attractive.

Using generated data sets where we could control the noise level and degree of overlap between clusters, we found that where there is a clear linear separation between the classes, the method works well. But if the data is not linearly separable, it is harder to find the optimal set partition and width of the hyperplane margin. Like column generation methods, the approach suffers from instability in the dual problem unless there are enough columns in the working set to adequately define the solution space. A rigid body dynamics analogy provides insight.

The algorithm could be improved by introducing column generation stabilization techniques, for example through regularized bundle methods, but arguably this is moving too close to the active set methods described in Chapter 4. Our experiments in Chapter 6 showed that active set methods require many iterations to train on nonseparable data. The strength of our IPM approach in such cases is that, through controlling the complementarity products, the set partitioning is delayed as long as possible. A good dual solution (hyperplane) is found before the support vectors are identified, and the dual solution is more important for prediction.

A better approach is to combine the idea behind column generation—to work with a meaningful subset of variables—with the strength of IPM at handling the more difficult set partitioning. In summary, Algorithm 5 consists of determining an initial approximation to the hyperplane, removing those variables to $\mathscr{N}$ or $\mathscr{U}$ where we are confident of their set membership, and then solving the RMP with the remaining variables using the techniques of previous chapters.

Arguably, finding the hyperplane is more important than finding the optimal set partition. Our approximation (8.2) to the SVM QP gave highly accurate discriminant hyperplanes for each data set we tested, and this suggests some alternative approaches to the training problem. We could develop a two-stage approach. At the first stage, the hyperplane normal is determined by solving (8.2). Then the second stage consists of solving a reduced QP where the direction of w is fixed, and only width of the margin and the bias w_0 are variable. This will reduce the number of rows required from $(m+1)$ to 2. Another possibility is to solve (8.2) alone, as it is already a good approximation to the SVM QP training problem. As both of these possibilities are outside of the original SVM QP, and even remove the need for interior point methods, they are rather out of scope for this thesis and are not investigated here further.

Chapter 9

Nonlinear SVMs

Non-linear kernels are a powerful extension to the Support Vector Machine technique, allowing them to handle data sets that are not linearly separable. One of the main advantages of the dual formulation (3.12) is that the feature mapping can be performed implicitly through kernel functions, since the original attribute vectors appear only in terms of inner products (see Section 3.3.7).

The matrix resulting from a non-linear kernel is normally dense, but researchers have noted that it is possible to make a good low-rank approximation of the kernel matrix (Smola and Schölkopf, 2000). Approaches considered to find such a low-rank approximation include Nyström's approximation method (Williams and Seeger, 2001; Drineas and Mahoney, 2005), Bregman matrix divergence measures (Kulis et al., 2006), partial Cholesky decomposition with pivoting (Fine and Scheinberg, 2001), and the inclusion of class label information (Bach and Jordan, 2005). In this chapter we investigate the approach of Fine and Scheinberg (2001), develop a parallel algorithm for the approximation using Cholesky decomposition, and look to improve the quality of the approximation through pivot selection heuristics.

9.1 Partial decomposition of non-linear kernels

For a positive semidefinite matrix K, Cholesky decomposition can be used to compute $LL^T \equiv K$. *Cholesky decomposition with partial pivoting* produces the first r columns of the matrix L (corresponding to the r largest pivots) and leaves the other columns as zero, giving an approximation of the matrix K of rank r. This is an attractive algorithm for partial decomposition, since its complexity is linear with the number of samples, it is faster than eigenvalue decomposition, and it exploits the symmetry of K. Fine and Scheinberg (2001)

used this algorithm, based on general ones presented in Golub and van Loan (1983, algorithms 4.4-2 and 5.1-2), with an overall complexity of $\mathcal{O}(2nr^2 + 2nmr)$, where r is the rank of the approximation and m the number of input attributes. Their approach assumes that K is known explicitly, and requires the diagonal values to be determined repeatedly; however, in the case of SVMs the kernel matrix is known only implicitly and calculating its values is an expensive process (indeed, it forms the bulk of the computation time for active set algorithms). We therefore extended Fine and Scheinberg's algorithm to calculate each kernel element only once, giving a complexity of $\mathcal{O}(nr^2 + nmr)$, at the expense of $\mathcal{O}(n)$ additional memory storage; we present this algorithm as Algorithm 6.

Algorithm 6 Cholesky decomposition with partial pivoting $LL^T + \mathrm{diag}(d) \approx K$

for $i = 1 : n$ **do**
  $d_i := K_{ii}$ // Initialise the diagonal
end for
// Calculate a maximum of r columns
for $i = 1 : r$ **do**
  if $\sum_{j=i}^{n} d_j > \epsilon_{tol}$ **then**
    find $j^* : d_{j^*} = \max_{j=i:n} d_j$
    // Interchange rows i and j^*
    $L_{i,1:i} \leftrightarrow L_{j^*,1:i}$
    $d_i \leftrightarrow d_{j^*}$
    // Calculate column i
    $L_{i,i} := \sqrt{d_i}$
    $L_{i+1:n,i} := (K_{i+1:n,i} - L_{i+1:n,1:i}(L_{i,1:i})^T)/L_{i,i}$
    // Update the diagonal
    $d_i := 0$
    for $j = i + 1 : n$ **do**
      $d_j := d_j - (L_{j,i})^2$
    end for
  else
    $r := i - 1$
    return
  end if
end for

Fine and Scheinberg (2001) use the approximation $K \approx LL^T$, but with Algorithm 6 the approximation $K \approx LL^T + \mathrm{diag}(d)$ can be determined at no extra expense. The optimization

formulation (5.1) becomes:

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2}(w^T w + z^T D z) - e^T z \\ \text{s.t.} \quad & w - (YL)^T z = 0 \\ & -y^T z = 0 \\ & 0 \le z \le \tau e. \end{aligned} \tag{9.1}$$

Note that we can still exploit the separability of the objective as the Hessian is still diagonal, so the computational complexity remains unchanged.

To test the performance, we used a synthetic data set based on the 8×8 chessboard pattern and the serial software `SVM-HOPDM` used in Chapter 6. Training sets could be made arbitrarily large, and noise added by mis-labelling samples. We used the RBF kernel function (3.10b) to define the kernel matrix. The condition number of K is largely determined by γ. When $\gamma \to \infty$, K tends to the identity matrix, a full rank matrix. When $\gamma \to 0$, K tends to a matrix with all elements of value one, a rank one matrix. In existing active-set methods, learning time is predominantly influenced by the penalty parameter τ; in contrast, it is the parameter γ that dominates learning time with low-rank approximation methods.

Typical performance in terms of training time and test set classification accuracy is shown in Figure 9.1. Accuracy approaches a limit for $r > 180$ for this problem, even though the trace and Frobenius norm measurements of the error in the approximation continue to decrease (see Figure 9.2). Table 9.1 shows performance compared to `SVMTorch` for a range of noise levels and τ values. At $r = 200$, classification accuracy is comparable. As in the case of linear kernels, higher noise and τ levels greatly increase the training time of `SVMTorch` but have little effect on `SVM-HOPDM`.

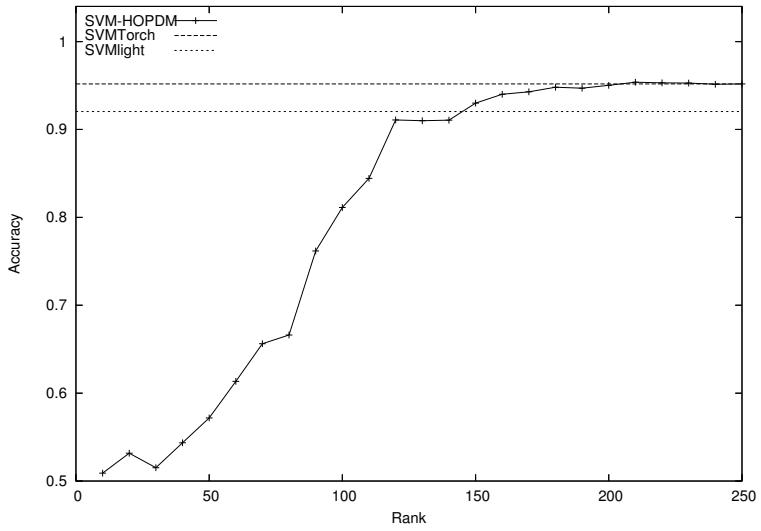
Several real-world data sets were also used to assess the effectiveness of the approach:

USPS [1] United States Postal Service data set of hand-written digits. This data set comprises 7291 training and 2007 test patterns, represented as dense vectors of dimension 257 with entries between 0 and 255. The digits have been classified by hand and labelling is highly accurate. The classification task set was to discriminate the digit 4 from the others, as set by Williams and Seeger (2001).

Adult [2] This data set was compiled by Platt and taken from the SMO home page. The goal was to predict whether a household has an income greater than $50000. The data set

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
[2] http://research.microsoft.com/users/jplatt/smo.html

(a)



(b)

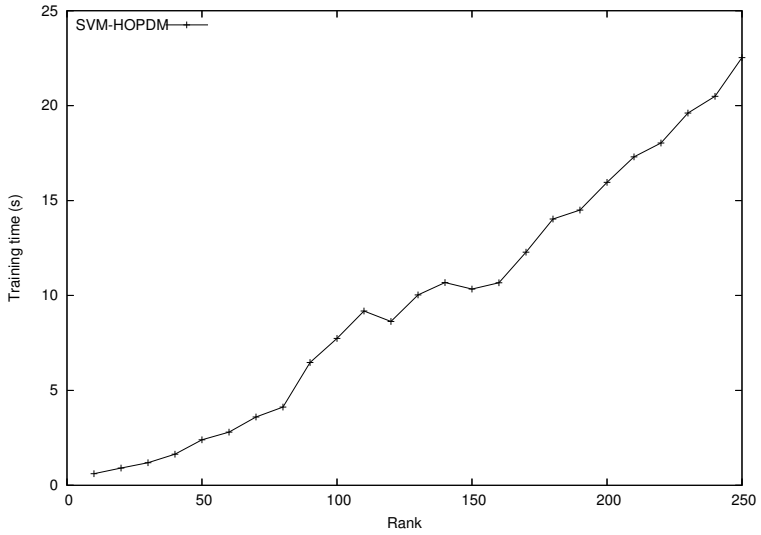


Figure 9.1: Performance of the `SVM-HOPDM` algorithm on the chessboard pattern, relative to the rank of the approximation. The training data set used had 10,000 samples with 5% misclassified. Parameters were $\gamma = 0.5$ and $\tau = 10^4$. (a) Classification accuracy using an unseen test set. (b) Training time in seconds. `SVMTorch` took 4991.6s, while `SVM`$^{\texttt{light}}$ took 14772s to solve the same problem.

Noise	τ	SVM-HOPDM		SVMTorch	
		Accuracy (%)	Time (s)	Accuracy (%)	Time (s)
0%	100	94.48	12.26	94.46	59.59
0%	10^4	97.82	18.46	97.80	813.63
5%	100	92.54	12.64	92.06	103.19
5%	10^4	95.02	15.96	95.18	4991.61
10%	100	90.56	12.38	90.82	125.79
10%	10^4	93.32	13.32	93.84	8121.46

Table 9.1: Comparison of training times and accuracy using RBF-based SVMs on chessboard data sets with various levels of mis-labelling. For the low-rank approximation method of SVM-HOPDM, 200 columns were used. The penalty parameter τ affects SVMTorch to a much greater extent than SVM-HOPDM. Results for SVM^{light} are not shown as they are always dominated by SVMTorch.

is relatively sparse due to nominal attributes converted to binary. After discretization of the continuous attributes, there are 123 binary features, with approximately 14 non-zeros per example. We used the largest training data set of 32562 samples, and the test set of 16282 samples.

SensIT [3] This data set was compiled from an extensive real world experiment (Duarte and Hu, 2004), where data was collected on types of moving vehicles by using a wireless distributed sensor networks. We used the data set with 100 dense attributes, combining acoustic and seismic data. There were 78,823 samples in the training set and 19,705 in the test set. The classification task set was to discriminate class 3 from the other two. This is a relatively noisy data set—benchmark accuracy is around 85%.

Results are shown in Table 9.2. The USPS data set is dense but the labelling is highly accurate. SVMTorch solves it very quickly and with high accuracy; SVM-HOPDM cannot match either measure. The training times are much longer for the Adult and SensIT data sets than can be attributed to the number of samples; SVMTorch is affected by the noise level. Training times using SVM-HOPDM are much lower (as much as one or two orders of magnitude smaller) and more consistent. The classification accuracy of the technique based on low-rank approximation approaches that of the active-set method, but rarely matches or exceeds it (for example, see Figure 9.3). It is possible that increased rank would improve classification accuracy with the SensIT data, as in our experiments accuracy continued to improve until the memory limit of the computer was reached.

Measuring the trace of the error between the kernel matrix and its approximation is efficient to perform, and as can be seen in Figure 9.2 is a consistent indicator of the Frobenius

[3] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Data set	γ	τ	SVMTorch		SVM^{light}		SVM-HOPDM LL^T		SVM-HOPDM $LL^T + D$	
			Time	Acc	Time	Acc	Time	Acc	Time	Acc
USPS	0.0078	1	4.82	98.56	13.84	98.56	18.31 $r = 240$	98.01	16.75 $r = 240$	98.21
USPS	0.0078	100	4.20	99.15	14.01	99.15	19.17 $r = 250$	98.8	10.59 $r = 170$	98.26
USPS	0.0078	10000	4.23	99.15	13.99	99.15	15.92 $r = 240$	98.6	10.11 $r = 170$	98.26
Adult	0.0163	1	144	85.17	234	84.82	30.66 $r = 150$	85.04	19.02 $r = 90$	84.93
Adult	0.0163	10	356	85.11	782	85.15	21.94 $r = 80$	85.04	9.51 $r = 50$	85.16
Adult	0.0163	100	2757	84.81	12197	84.79	29.24 $r = 100$	85.08	33.2 $r = 150$	85.16
SensIT	0.0400	1	946	86.80	4143	86.79	327.18 $r = 240$	86.16	213.15 $r = 240$	86.40
SensIT	0.0400	10	3873	87.51	21288	87.50	362.25 $r = 240$	86.33	230.23 $r = 240$	86.36
SensIT	0.0400	100	31809	88.18	261829	88.20	269.3 $r = 240$	86.56	206.83 $r = 240$	86.75

Table 9.2: Results from training RBF-based SVMs from real-life data sets USPS, Adult and SensIT. Training times are in seconds, and accuracy shown as a percentage. For the low-rank approximation methods, the training problems were solved for various r, using multiples of 10. We show the results and the value of r that gave the best accuracy.

norm of the error. The real-world data sets used here have a quickly decaying eigenvalue spectrum, as can be seen in Figure 9.2 where the benefit of adding further vectors to the approximation of the Adult data set quickly tails off. With such data sets, there is an improvement in the error, as measured by Frobenius norm, by using the residual diagonal in the approximation, i.e. $K \approx LL^T + D$. Figure 9.3 shows that this approximation in fact makes little difference to classification accuracy, but it does make the problem easier to solve, since fewer and a more consistent number of iterations are required. With this approximation, the Hessian is much closer to a positive definite matrix (only r elements of the diagonal of the Hessian are zero, those corresponding to columns in L), giving a near-strictly convex optimization problem. For RBF kernels, the kernel matrix is guaranteed positive definite unless there are collocated samples.

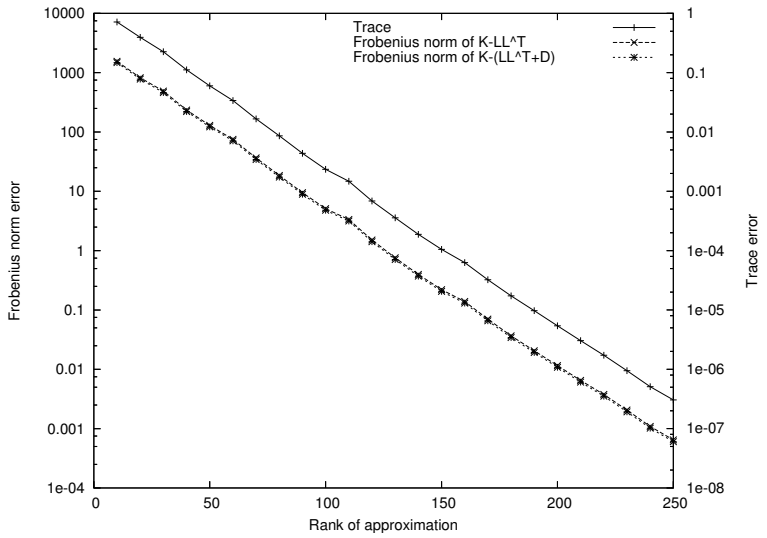
Turning now to the classification of an unseen test set after training is completed, it is worth noting that the time taken to classify is considerably reduced using the method based on low-rank Cholesky decomposition: this method requires $\mathcal{O}(r^2 + rm)$ operations to label each test sample, whilst the active set methods require $\mathcal{O}(nm)$ operations due to the number of support vectors typically being a proportion of the number of samples. The difference can be seen with the data sets used in this study; for example, `SVM-HOPDM` took 5.1s to classify the SensIT test set, while `SVMTorch` took 273s.

9.2 Parallel partial Cholesky decomposition

Algorithm 7 describes how to perform partial Cholesky decomposition in a parallel environment. As data sets may be large, the data is segmented between the processors. To determine pivot candidates, all diagonal elements are calculated (steps 1–4). Then, for each of the r columns, the largest diagonal element is located (steps 7–8). The processor p^* that owns the pivot row j^* calculates the corresponding row of L, and this row forms part of the "basis" B (steps 10–13). The basis and the original features x_{j^*} need to be known by all processors, so this information is broadcast (step 14). With this information, all processors can update the section of column i of L for which they are responsible, and also update corresponding diagonal elements (steps 16–19).

Although the algorithm requires the processors to be synchronised at each iteration, little of the data needs to be shared amongst the processors: the bulk of the communication between processors (step 14) is limited to a vector of length m and a vector of at most length r. Note that matrix K is known only implicitly, through the kernel function, and calculating

(a)



(b)

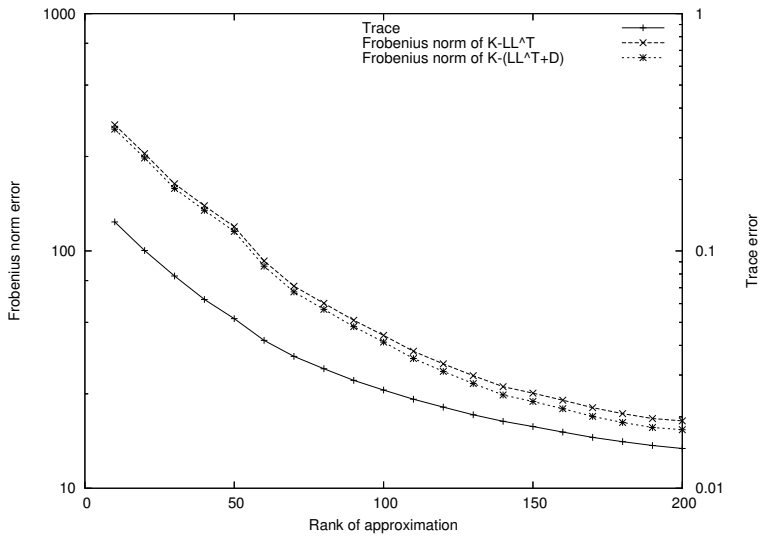
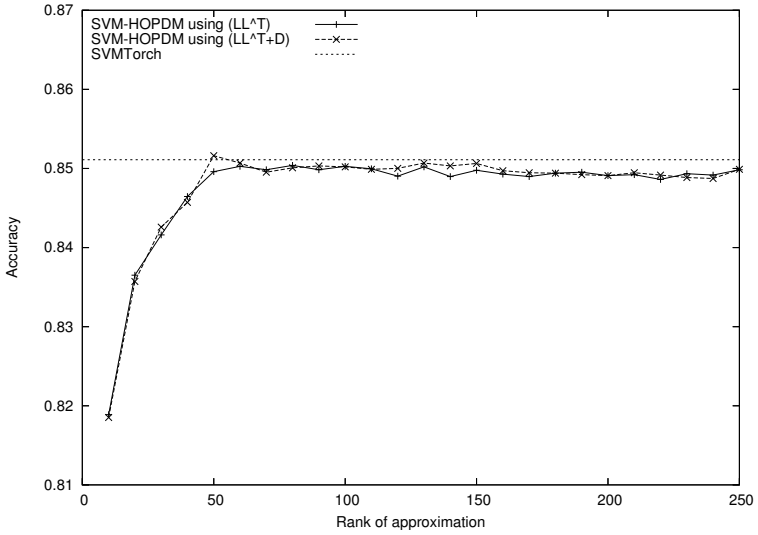


Figure 9.2: Errors in the low-rank approximation of the kernel matrix, for the chessboard and Adult data sets. (a) Chessboard data set with $\gamma = 0.5$ (b) Adult data set with $\gamma = 0.01$

(a)



(b)

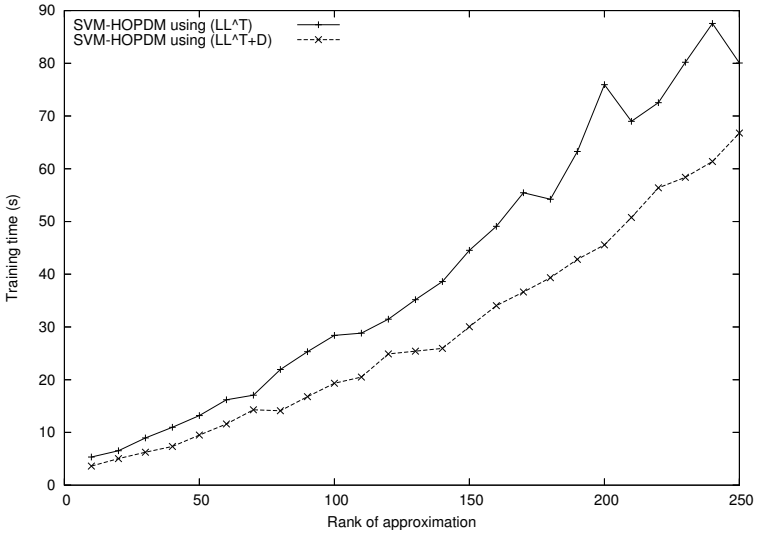


Figure 9.3: Accuracy and training times for the Adult data set ($\gamma = 0.0163, \tau = 10$), using the approximations $K \approx LL^T$ and $K \approx LL^T + D$, with respect to the rank of the approximation. (a) Accuracy classifying the test set (b) Training time in seconds

Algorithm 7 Parallel Cholesky decomposition with partial pivoting
$LL^T + \text{diag}(d) \approx K$

Input:

n_p Number of samples on each processor p

r Required rank of approximation matrix L

X_p Processor-local features data set

Output:

B Global basis matrix

L Partial Cholesky decomposition of processor-local data

d Diagonal of residual matrix $K - LL^T$

$\mathcal{J} := \{1 \dots n_p\}$
for $j \in \mathcal{J}$ **do**
  $d_j := K_{jj}$ // Initialise the diagonal
end for
// Calculate a maximum of r columns
for $i = 1 : r$ **do**
  if $\sum_p \sum_{j=i}^{n_p} d_j > \epsilon_{tol}$ **then**
    On each machine, find local $j_p^* : d_{j_p^*} = \max_{j \in \mathcal{J}} d_j$
    Locate global $(j^*, p^*) : (j^*, p^*) = \arg\max_p d_{j_p^*}$ // e.g. using MPI_MAXLOC
    if machine is p^* **then**
      $B_{i,:} := L_{j^*}$ // Move row j^* to basis
      $\mathcal{J} := \mathcal{J} \setminus j^*$
      $B_{ii} := \sqrt{d_{j^*}}$
      $d_{j^*} := 0$
      Broadcast features x_{j^*}, and basis row $B_{i,:}$.
    end if
    // Calculate column i on all processors, where $\mathcal{J}$ is set of rows not moved to basis
    $L_{\mathcal{J},i} := (K_{\mathcal{J},i} - L_{\mathcal{J},1:i}(L_{i,1:i})^T)/B_{i,i}$
    // Update the diagonal
    for $j \in \mathcal{J}$ **do**
      $d_j := d_j - (L_{j,i})^2$
    end for
  else
    $r := i - 1$
    return
  end if
end for

its values is an expensive process. The algorithm therefore calculates each kernel element required to form L only once, giving a complexity of $\mathcal{O}(nr^2 + nmr)$.

In conclusion, Cholesky decomposition with partial pivoting is an efficient way to produce a low-rank approximation of the kernel matrix, and doing so allows the separable formulation and implementations of previous chapters to be applied to the training of non-linear SVMs. Including the residual diagonal elements in the approximation improves the operation of the IPM algorithm. The decomposition remains an approximation though, and in our experiments with real-world data sets there remained a small gap in prediction accuracy and a point where increasing the rank gave little added benefit. Pivots need to be chosen wisely, and this is the topic of the next chapter.

Chapter 10

Pivot selection in nonlinear kernels

Since the IPM algorithm has a per-iteration complexity of $\mathcal{O}(nr^2)$, it is of greater benefit to our nonlinear SVM training approach if we can reduce the rank r of the low-rank approximation, compared to a similar reduction in the number of samples n (Chapter 8). In the approximation schemes described in the previous chapter, pivots were chosen by applying the greedy heuristic algorithm of selecting the largest diagonal element each time, which corresponds to the largest eigenvalue. The main advantage of the heuristic is that the number of computationally-expensive kernel evaluations is restricted to nr, considerably less than if the kernel matrix was evaluated fully.

In this chapter we show that this heuristic is not the best for approximating clustered data using a low-rank matrix, if the residual diagonal forms part of the approximation. We therefore develop a heuristic for pivot selection, that aims to capture as much information of the original kernel matrix as possible in the low rank approximation, whilst still restricting the number of kernel computations involved.

10.1 Approximating clusters and outliers

Consider a data-set that consists of two clusters A and B, both with a large number (n_A and n_B respectively, and $n_A \approx n_B$) of very closely situated points, and n_C outlying points C which are far from either cluster (see Figure 10.1). If the Radial Basis Kernel is used for this data set, it will generate a kernel matrix with a large block of elements equal to or very close to 1 corresponding to cluster A, a similar block corresponding to cluster B, and a diagonal block very close to the identity matrix corresponding to points C. Using the notation $e_{n_A} = (1,1,\ldots,1)^T \in \mathbb{R}^{n_A}$, $e_{n_B} = (1,1,\ldots,1)^T \in \mathbb{R}^{n_B}$ and where $\alpha \in (0,1)$, the kernel matrix can

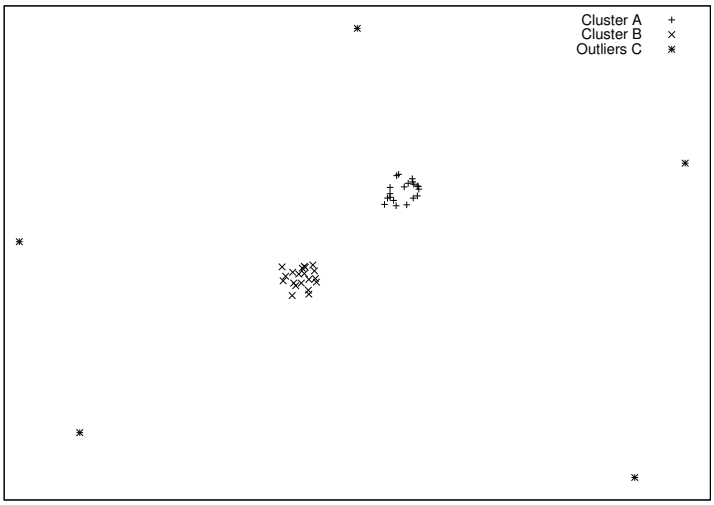


Figure 10.1: Data set consisting of two clusters plus outliers.

be closely approximated by

$$K^{(0)} = \begin{bmatrix} e_{n_A} e_{n_A}^T & \alpha e_{n_A} e_{n_B}^T & & (A) \\ \alpha e_{n_B} e_{n_A}^T & e_{n_B} e_{n_B}^T & & (B) \\ & & I_{n_C} & (C) \end{bmatrix}$$

if A and B are reasonably close, while the points of C are far from both the A and B clusters. In other words, the large blocks corresponding to clusters A and B can be very well approximated by rank-one matrices. Note that all the pivots have the same value at this stage.

Next we show by example that such a kernel matrix, resulting from natural clusters in the data set, cannot be well approximated by a partial Cholesky decomposition LL^T with pivoting based on the largest diagonal element unless the rank of L is at least $n_c + 2$. Suppose a small rank, say $r = 2$ is used. Proposition 10.1.1 gives the error in the approximation of $K^{(0)}$ using such an approximation, when rank $r = 2$. Alternatively, Proposition 10.1.2 shows that an approximation using $LL^T + D$ with L built of merely two columns, and where pivots are chosen on some measure based on clustering, has minimal error. Let the notation $\mathcal{O}(\epsilon)$ indicate a number very close to zero, and blocks in the form $\mathcal{O}(\epsilon)_{n_A \times n_A}$ indicate sub-matrices where diagonal elements are zero and non-diagonal elements are very close to zero.

Proposition 10.1.1. *If a low-rank approximation LL^T of matrix $K^{(0)}$ is constructed, with the rank $r = 2$ and at each iteration the largest pivot is chosen, the lowest Frobenius norm of the residual matrix is $\|K^{(0)} - LL^T\|_F^2 \approx (1-\alpha^2)n_B^2 + n_C - 1$.*

Proof. If we assume that the first pivot chosen at random is taken from cluster A (if $n_A \approx n_B$

then clusters A and B are interchangable), the first approximation of the kernel matrix is

$$K^{(0)} = \begin{bmatrix} e_{n_A} \\ \alpha e_{n_B} \\ 0_{n_C} \end{bmatrix} \begin{bmatrix} e_{n_A}^T & \alpha e_{n_B}^T & 0_{n_C}^T \end{bmatrix} + K^{(1)},$$

$$\text{where} \qquad K^{(1)} = \begin{bmatrix} \mathcal{O}(\epsilon)_{n_A \times n_A} & \mathcal{O}(\epsilon)_{n_A \times n_B} & \\ \mathcal{O}(\epsilon)_{n_B \times n_A} & (1-\alpha^2) e_{n_B} e_{n_B}^T & \\ & & I_{n_C} \end{bmatrix}.$$

At the next iteration, the largest pivot will be found among diagonal elements corresponding to points in C, giving

$$K^{(0)} = \begin{bmatrix} e_{n_A} \\ \alpha e_{n_B} \\ 0_{n_C} \end{bmatrix} \begin{bmatrix} e_{n_A}^T & \alpha e_{n_B}^T & 0_{n_C}^T \end{bmatrix} + \begin{bmatrix} 0_{n_A+n_B} \\ 1 \\ 0_{n_C-1} \end{bmatrix} \begin{bmatrix} 0_{n_A+n_B}^T & 1 & 0_{n_C-1}^T \end{bmatrix} + K^{(2)}$$

$$\text{where} \quad K^{(2)} = \begin{bmatrix} \mathcal{O}(\epsilon)_{n_A \times n_A} & \mathcal{O}(\epsilon)_{n_A \times n_B} & & \\ \mathcal{O}(\epsilon)_{n_B \times n_A} & (1-\alpha^2) e_{n_B} e_{n_B}^T & & \\ & & 0 & \\ & & & I_{n_C-1} \end{bmatrix}.$$

The Frobenius norm of this residual matrix is $\|K^{(0)} - LL^T\|_F^2 \approx (1-\alpha^2) n_B^2 + n_C - 1$. □

By using a similar argument we can prove that unless all pivots corresponding to points in C are eliminated, no pivot from the cluster B can be chosen. Hence we need a rank of at least $n_C + 2$ to reduce the error of the approximation to an $\mathcal{O}(\epsilon)$ level.

Proposition 10.1.2. *If an approximation $LL^T + D$ of matrix $K^{(0)}$ is constructed, where a point in cluster A and a point in cluster B are chosen to form the two columns of L, and D is the diagonal of the residual matrix, then the Frobenius norm of the residual matrix is* $\|K^{(0)} - (LL^T + D)\|_F = \|\mathcal{O}(\epsilon)_{(n_A+n_B)\times(n_A+n_B)}\|_F$.

Proof. If, instead of choosing the pivots based on the largest elements in the diagonal, we

choose a pivot from cluster A and a pivot from cluster B, then we get

$$K^{(0)} = \begin{bmatrix} e_{n_A} \\ \alpha e_{n_B} \\ 0_{n_C} \end{bmatrix} \begin{bmatrix} e_{n_A}^T & \alpha e_{n_B}^T & 0_{n_C}^T \end{bmatrix}$$
$$+ \begin{bmatrix} 0_{n_A} \\ \sqrt{1-\alpha^2} e_{n_B} \\ 0_{n_C} \end{bmatrix} \begin{bmatrix} 0_{n_A}^T & \sqrt{1-\alpha^2} e_{n_B}^T & 0_{n_C}^T \end{bmatrix} + K^{(2)}$$
$$\text{where} \quad K^{(2)} = \begin{bmatrix} \mathcal{O}(\epsilon)_{(n_A+n_B)\times(n_A+n_B)} & \\ & I_{n_C} \end{bmatrix},$$

with a Frobenius norm $\|K^{(0)} - LL^T\|_F^2 \approx n_C$. If cluster B is large and C small, then this is obviously preferable. Additionally, if we include a residual diagonal into our approximation $(LL^T + D)$, the norm of the residual error matrix can be reduced further to $\|K^{(0)} - (LL^T + D)\|_F = \|\mathcal{O}(\epsilon)_{(n_A+n_B)\times(n_A+n_B)}\|_F$. □

We can see from this example that the algorithm that chooses the largest pivot will have to choose all the outliers C before it will select any point from cluster B. The approximation will need to be of higher rank than the number of outliers. We can remedy this by using the form $K \approx LL^T + D$ and packing all elements corresponding to outliers C into matrix D, without increasing the rank of L.

10.2 Bounding the error in the approximation

If we approximate the original kernel matrix using the matrix $LL^T + D$, and use it to form the Hessian of the SVM training QP (9.1), how close is the perturbed problem to the original one? One method of judging is to measure the error in the objective value. To do this, we consider the original non-linear SVM dual formulation (3.12), with the kernel matrix forming part of the Hessian, and the approximate QP in which the new (approximate) Hessian $\tilde{K}$ is used. Let $f(z) = \frac{1}{2} z^T Y K Y z^T - e^T z$ and $\tilde{f}(z) = \frac{1}{2} z^T Y \tilde{K} Y z^T - e^T z$ denote the objective functions in the original problem (3.12) and in its approximation respectively. Bounds for the error in the approximated objective function are given in Theorem 10.2.1.

Theorem 10.2.1. *Let z^* be the optimal solution of (3.12) and $\tilde{z}^*$ be the optimal solution of the approximate QP. Let $K = \tilde{K} + \Delta K$. Then the error in the objective value $|f(z^*) - \tilde{f}(\tilde{z}^*)|$ is*

bounded by

$$\frac{1}{2}z^{*T}Y\Delta KYz^* \leq f(z^*) - \tilde{f}(\tilde{z}^*) \leq \frac{1}{2}\tilde{z}^{*T}Y\Delta KY\tilde{z}^*.$$

Proof. First we observe that the two QP problems differ only in the objective functions; the constraints are identical. Hence any feasible solution of (3.12) is also a feasible solution of the approximate problem. From the definitions of $f(z)$ and $\tilde{f}(z)$, for any point z,

$$\begin{aligned} f(z) - \tilde{f}(z) &= \frac{1}{2}z^T YKYz - \frac{1}{2}z^T Y\tilde{K}Yz \\ &= \frac{1}{2}z^T Y\Delta KYz. \end{aligned} \tag{10.1}$$

For the upper bound, consider the point z^*. It is the minimizer of $f(\cdot)$, therefore $f(z^*) \leq f(\tilde{z}^*)$. Applying (10.1) to the point $\tilde{z}^*$ and substituting in this inequality gives

$$f(z^*) - \tilde{f}(\tilde{z}^*) \leq \frac{1}{2}\tilde{z}^{*T}Y\Delta KY\tilde{z}^*.$$

Similarly, the point $\tilde{z}^*$ minimizes $\tilde{f}(\cdot)$, so $\tilde{f}(\tilde{z}^*) \leq \tilde{f}(z^*)$. Substituting this relationship for $f(z^*)$ into (10.1) at z^* gives

$$f(z^*) - \tilde{f}(\tilde{z}^*) \geq \frac{1}{2}z^{*T}Y\Delta KYz^*. \qquad \square$$

A good approximation will keep these bounds as tight as possible. Below we attempt to quantify this. Let us define the matrix norm $\|\Delta K\|_e := \sum_i \sum_j |\Delta K_{ij}|$, the sum of all absolute values in ΔK.

Corollary 10.2.2. *If the matrix K in QP (3.12) is approximated by the matrix $\tilde{K}$, where $K = \tilde{K} + \Delta K$, the error in the objective due to the approximation is bounded by*

$$|f(z^*) - \tilde{f}(\tilde{z}^*)| \leq \frac{1}{2}\tau^2\|\Delta K\|_e.$$

Proof. In (3.12) z is bounded by $0 \leq z \leq \tau e$, and only the support vectors themselves will have non-zero values, but obviously we can only know the values of z^* after the approximation is made. Following Theorem 10.2.1, a worst case for the bounds of the error, $|f(z^*) - \tilde{f}(\tilde{z}^*)|$,

can be constructed by taking all elements of z to their bounds:

$$\begin{aligned}|f(z^*) - \tilde{f}(\tilde{z}^*)| &\leq \frac{1}{2}\sum_i\sum_j \max(\tau^2 y_i y_j \Delta K_{ij}, 0) \\ &\leq \frac{1}{2}\tau^2 \sum_i\sum_j |y_i y_j \Delta K_{ij}| \\ &= \frac{1}{2}\tau^2 \sum_i\sum_j |\Delta K_{ij}|\end{aligned}$$

as $y_i, y_j \in \{-1, +1\}$.

Using the definition of $\|\Delta K\|_e$, the maximum error in the objective is then

$$|f(z^*) - \tilde{f}(\tilde{z}^*)| \leq \frac{1}{2}\tau^2 \|\Delta K\|_e,$$

and clearly minimizing $\|\Delta K\|_e$ will reduce the bound on the error. □

Corollary 10.2.3 highlights that only elements of ΔK in the row and column of two support vectors actually contribute to the approximation error. This leads to the conclusion that it is only necessary to calculate the elements of the approximation matrix $\tilde{K}$ that occur at the intersection of rows and columns corresponding to support vectors (both bound and in-bound support vectors, see Table 8.1), and doing so for the optimal set of support vectors will reduce the error in the objective to zero. From Corollary 10.2.3, an iterative approach could be developed, where the approximation is continuously improved by choosing columns corresponding to support vectors from previous iterations.

Corollary 10.2.3. *Let $\tilde{z}_{SV}$ be the subvector of $\tilde{z}$ formed by removing entries equal to zero, to leave only the support vectors. Let ΔK_{SV} and Y_{SV} be the submatrices of ΔK and Y with the corresponding rows and columns removed. The error in the objective due to the approximation described in Theorem 10.2.1 will be upper bounded by:*

$$\|f(z^*) - \tilde{f}(\tilde{z}^*)\| \leq \frac{1}{2}\tilde{z}_{SV}^{*T} Y_{SV} \Delta K_{SV} Y_{SV} \tilde{z}_{SV}^*.$$

Proof. Elements of $\tilde{z}^*$ not included in $\tilde{z}_{SV}^*$ are equal to zero, so

$$\tilde{z}^{*T} Y \Delta K Y \tilde{z}^* = \tilde{z}_{SV}^{*T} Y_{SV} \Delta K_{SV} Y_{SV} \tilde{z}_{SV}^*.$$

□

10.3 A cost measure for selecting pivots

In section 10.1 we observed that the diagonal elements of K corresponding to outlying data points will remain large during partial Cholesky decomposition, yet other elements in the columns will be small. An algorithm that selects columns by choosing the largest diagonal element will construct columns of L based on such points, yet they can be adequately represented by a diagonal matrix D. In that example, the decision whether a pivot should contribute to a diagonal term of D or a column of L was straightforward. In real-life examples this decision is less obvious. Moreover, we wish to keep the rank of L as small as possible, hence we would like to encourage an early choice of essential columns which indeed should contribute to L. We start our discussion from an analysis of a simplified example in which two attractive pivot candidates are available.

Let the kernel submatrix at an iteration be

$$K = \begin{bmatrix} u_1 & 0 & \bar{u}^T \\ 0 & v_1 & \bar{v}^T \\ \bar{u} & \bar{v} & \bar{K} \end{bmatrix}. \tag{10.2}$$

The columns containing u and v are the candidates for pivoting. For simplicity, we consider them after being permuted to the top of the matrix. The diagonal elements u_1 and v_1 are pivot candidates, while $\bar{u}$ and $\bar{v}$ are vectors containing the rest of the column. $\bar{K}$ contains the rest of the matrix. To describe the effect on the residual matrix after a pivot candidate is chosen, we make the following assumption.

Assumption 10.3.1. *The kernel matrix is such that $K \geq 0$, and for all pivot column candidates $K - \frac{1}{u_1} u u^T \geq 0$.*

Assumption 10.3.1 implies that all elements of the Schur complement will be reduced in absolute value as the result of the pivoting. First let us observe that, by construction, kernel matrices are positive semi-definite. For some types of kernel, it is possible to say more: for instance $K \geq 0$ for a polynomial kernel with an even power, and $K > 0$ for the commonly used RBF kernel. Unfortunately for later iterations this is only an approximation: $K \geq 0$ cannot be guaranteed in later iterations even for an initial kernel matrix using the radial basis kernel. However, we have noted empirically that the vast majority of elements of the Schur complement remain positive during at least the early iterations of Cholesky decomposition. We exploit these observations for the purpose of developing a heuristic, by

taking that Assumption 10.3.1 will hold at least for the early iterations where this heuristic will be used.

Remark 10.3.2. *If Assumption 10.3.1 holds, the residual matrix after choosing column u will have the norm*

$$\|K - ll^T\|_e = v_1 + 2\|\bar{v}\|_1 + \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2.$$

Proof. If u is chosen for the pivot, the corresponding column l of the Cholesky decomposition will be $\begin{bmatrix} l_1 \\ 0 \\ \bar{l} \end{bmatrix}$, where $l_1 = \sqrt{u_1}$ and $\bar{l} = \frac{1}{l_1}\bar{u}$. The residual matrix $K - ll^T$ resulting when column u is chosen will be

$$K - ll^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & v_1 & \bar{v}^T \\ 0 & \bar{v} & \bar{K} - \bar{l}\bar{l}^T \end{bmatrix}.$$

The entry-wise norm defined before Corollary 10.2.2 will be $\|K - ll^T\|_e = v_1 + 2\|\bar{v}\|_1 + \|\bar{K} - \bar{l}\bar{l}^T\|_e$.

Under the (restrictive) Assumption 10.3.1,

$$\|\bar{K} - \bar{l}\bar{l}^T\|_e = \|\bar{K}\|_e - \|\bar{l}\bar{l}^T\|_e.$$

Furthermore, using the relationships $\bar{l} = \frac{1}{l_1}\bar{u}$ and $l_1 = \sqrt{u_1}$, the norm of the sub-matrix $\bar{K} - \bar{l}\bar{l}^T$,

$$\|\bar{K} - \bar{l}\bar{l}^T\|_e = \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\bar{u}^T\|_e.$$

Using the following: $\|\bar{u}\bar{u}^T\|_e = \sum_i \sum_j (|\bar{u}_i||\bar{u}_j|) = \sum_i |\bar{u}_i| \sum_j |\bar{u}_j| = \|\bar{u}\|_1^2$, we conclude

$$\|\bar{K} - \bar{l}\bar{l}^T\|_e = \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2, \quad \text{and}$$
$$\|K - ll^T\|_e = v_1 + 2\|\bar{v}\|_1 + \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2. \qquad \square$$

There will be an analogous result if column v was used. It is now possible to say which column should be chosen as the pivot: we will prefer column u to v if it gives a smaller error in the residual matrix when compared to the error resulting from choosing column v; in other words, if $\|K - (ll^T)^{(u)}\|_e < \|K - (ll^T)^{(v)}\|_e$.

Remark 10.3.3. *If Assumption 10.3.1 holds, an approximation $LL^T \approx K$ with minimum*

residual is constructed by choosing column u as the pivot in preference to column v if

$$\frac{1}{v_1}\|v\|_1^2 < \frac{1}{u_1}\|u\|_1^2.$$

Proof. Using Remark 10.3.2 for $\|K - ll^T\|_e$, we get

$$\begin{aligned}
\frac{1}{v_1}\|v\|_1^2 &< \frac{1}{u_1}\|u\|_1^2 \\
v_1 + 2\|\bar{v}\|_1 + \frac{1}{v_1}\|\bar{v}\|_1^2 &< u_1 + 2\|\bar{u}\|_1 + \frac{1}{u_1}\|\bar{u}\|_1^2 \\
v_1 + 2\|\bar{v}\|_1 + \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2 &< u_1 + 2\|\bar{u}\|_1 + \|\bar{K}\|_e - \frac{1}{v_1}\|\bar{v}\|_1^2 \\
\|K - (ll^T)^{(u)}\|_e &< \| K - (ll^T)^{(v)}\|.
\end{aligned}$$

□

We can see from this inequality that whether column u is the best choice of pivot depends in a non-linear way on both the pivot value u_1 and the norm of the column $\|u\|_1$. It is not always best to choose the largest pivot.

The measure changes if we use the residual diagonal to improve our approximation, as shown in Proposition 10.3.4. Let the matrix K be approximated using $LL^T + \bar{D} \approx K$, where $\bar{D}$ is the diagonal of $\bar{K}$. The residual error matrix is therefore redefined as $K - (LL^T + \bar{D})$. We consider again matrix (10.2) and the pivot selection that minimizes the error in $\Delta K = K - (LL^T + \bar{D})$.

Proposition 10.3.4. *The pivot corresponding to column u is preferable to that of column v if*

$$2\|\bar{v}\|_1 + \frac{1}{v_1}(\|\bar{v}\|_1^2 - \|\bar{v}\|_2^2) < 2\|\bar{u}\|_1 + \frac{1}{u_1}(\|\bar{u}\|_1^2 - \|\bar{u}\|_2^2).$$

Proof. Using the redefined residual error matrix, then when column u is chosen as the pivot

$$\|\Delta K^{(u)}\|_e = v_1 + 2\|\bar{v}\|_1 + \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2 - v_1 - \|\bar{D}^{(u)}\|_e,$$

where $\bar{D}^{(u)}$ is the diagonal of matrix $\bar{K} - (\bar{l}\bar{l}^T)^{(u)}$. As $\|\bar{l}\|_2^2$ is the sum of the diagonal elements of $\bar{l}\bar{l}^T$, and $\|\bar{l}^{(u)}\|_2^2 = \frac{1}{u_1}\|\bar{u}\|_2^2$,

$$\|\bar{D}^{(u)}\|_e = \|\bar{D}\|_e - \frac{1}{u_1}\|\bar{u}\|_2^2.$$

Column u is the better pivot if

$$2\|\bar{v}\|_1 + \frac{1}{v_1}(\|\bar{v}\|_1^2 - \|\bar{v}\|_2^2) < 2\|\bar{u}\|_1 + \frac{1}{u_1}(\|\bar{u}\|_1^2 - \|\bar{u}\|_2^2).$$

□

We can use the measure

$$m_u = 2\|\bar{u}\|_1 + \frac{1}{u_1}(\|\bar{u}\|_1^2 - \|\bar{u}\|_2^2) \tag{10.3}$$

as the basis for selecting a pivot column. m_u can be thought of as a cost measure for approximating the column with a diagonal rather than providing an exact representation through a column in L.

10.4 An algorithm for Cholesky decomposition using column norms

Algorithm 8 A simplified serial algorithm to construct a Cholesky decomposition with partial pivoting, $\tilde{K} = LL^T + \text{diag}(d)$, using the cost measure m_u to minimize the residual error matrix.

$\mathcal{J} := \{1 \dots n\}$,
$\mathcal{L} := \emptyset, \quad \mathcal{D} := \emptyset, \quad \mathcal{P} := \emptyset, \quad \mathcal{U} := \mathcal{J}$
$d_j := K_{jj} \quad \forall j \in \mathcal{J}$ // Initialise the diagonal
// Calculate a maximum of r columns
for $i = 1 : r$ **do**
  Choose $j^* : d_{j^*} = \max_{j \in \mathcal{U}} d_j$
  Compute column j^* of the Schur complement:
  $S_{kj^*} := K_{kj^*} - \sum_{l=1}^{i} (L_{kl} \cdot L_{j^*l}) \quad \forall k = i+1, \dots, n$
  Compute the "cost" of approximating this column using just the diagonal, and not as a full column of L:
  $\bar{s}_{j^*} := \sum_{l=i+1}^{n} |S_{lj^*}|$
  $\bar{t}_{j^*} := \sum_{l=i+1}^{n} (S_{lj^*})^2$
  $m_{j^*} := 2\bar{s}_{j^*} + \frac{1}{d_{j^*}} \left(\bar{s}_{j^*}^2 - \bar{t}_{j^*} \right)$
  if $d_{j^*} < \epsilon_{\text{small}}$ or $m_{j^*} < \epsilon_{\mathcal{D}}$ **then**
    $\mathcal{D} := \mathcal{D} \cup \{j^*\}, \quad \mathcal{U} := \mathcal{U} \backslash \{j^*\}$
  else if $m_{j^*} > \epsilon_{\mathcal{L}}$ **then**
    $\mathcal{L} := \mathcal{L} \cup \{j^*\}, \quad \mathcal{U} := \mathcal{U} \backslash \{j^*\}$, form new column i of L using $L_{\cdot i} := S_{\cdot j^*}$, update the diagonal: $d_j := d_j - (L_{ji})^2 \quad \forall j \in \mathcal{U} \cup \mathcal{P}$
  else
    For a column where the allocation is not clear, temporarily remove it from consideration:
    $\mathcal{P} := \mathcal{P} \cup \{j^*\}, \quad \mathcal{U} := \mathcal{U} \backslash \{j^*\}$
  end if
  Return columns j to $\mathcal{U}$ that have been in $\mathcal{P}$ for the required number of iterations:
  $\mathcal{U} := \mathcal{U} \cup \{j\}, \quad \mathcal{P} := \mathcal{P} \backslash \{j\}$
  if $\mathcal{U} = \emptyset$ **then**
    $r := i$
    return
  end if
end for

We use (10.3) as the basis of greedy heuristic for selecting columns to form a Cholesky decomposition of the kernel matrix. At each iteration, we choose the column with the highest

measure, while rejecting columns with very small pivots that would increase numerical errors during the decomposition.

A simplified version is given as Algorithm 8. In this algorithm, n is the number of samples, r the maximum allowed number of columns of the Cholesky factor L, and d is the diagonal of the residual matrix $K - LL^T$. The full set of columns $\mathscr{J}$ of the matrix K are partitioned into four disjoint sets: $\mathscr{L}$ (columns allocated to L), $\mathscr{D}$ (allocated to D), $\mathscr{P}$ (columns postponed from consideration in the current iteration) and $\mathscr{U}$ (columns available for consideration, and not yet allocated to either L or D), where $\mathscr{J} = \mathscr{L} \cup \mathscr{D} \cup \mathscr{P} \cup \mathscr{U}$.

In addition, we use a number of algorithm parameters as the basis of decisions: $\epsilon_{\mathscr{L}}$ is a threshold value for including a column in $\mathscr{L}$, while $\epsilon_{\mathscr{D}}$ is a similar threshold value for inclusion in $\mathscr{D}$. We set $\epsilon_{\mathscr{D}} \ll \epsilon_{\mathscr{L}}$. If $\epsilon_{\mathscr{D}} \leq m_u \leq \epsilon_{\mathscr{L}}$ then we postpone the decision regarding column u. To aid numerical stability, columns are not chosen for pivoting if their diagonal element is below a minimum acceptable value ϵ_{small}.

To aid readability, we present the algorithm as a serial procedure. Initially all columns are in $\mathscr{U}$. At each iteration, the column with the largest diagonal element is selected, and the corresponding column of the Schur complement is calculated. From this we calculate the cost measure (10.3). Using this information, a number of options are available: we allocate the column to $\mathscr{D}$ if either the cost measure m_u is small or pivoting using this column would be numerically unstable; if the cost measure is large we allocate the column to $\mathscr{L}$; or we can postpone making a decision on this column by allocating it to $\mathscr{P}$. Columns remain in this set for a number of iterations, before being returned to $\mathscr{U}$. The algorithm terminates when r columns of L have been so constructed, or there are no more columns in $\mathscr{U}$.

In this presentation we assume that the columns are already permuted into the correct order; in the actual implementation, column permutations are handled implicitly. Kernel functions are expensive to evaluate, so individual elements of the matrix K are calculated as required in steps 3 and 6. Computing column j^* of the Schur complement can be performed in parallel, with each processor calculating elements corresponding to samples held locally; this requires the broadcast of the attribute vector x_{j^*} and the first i rows of L.

In practice it is hard to define a suitable value for $\epsilon_{\mathscr{L}}$, as it depends on the matrix K. In our experiments we used a cache of best columns, rather than the single column j^* shown in Algorithm 8, and chose the column with the highest cost measure to enter $\mathscr{L}$. When a column was added to $\mathscr{L}$, the Schur complement for candidate columns held in the cache were updated with the newly added column $L_{\cdot i}$, rather than fully recalculated as shown in Step 6 of Algorithm 8.

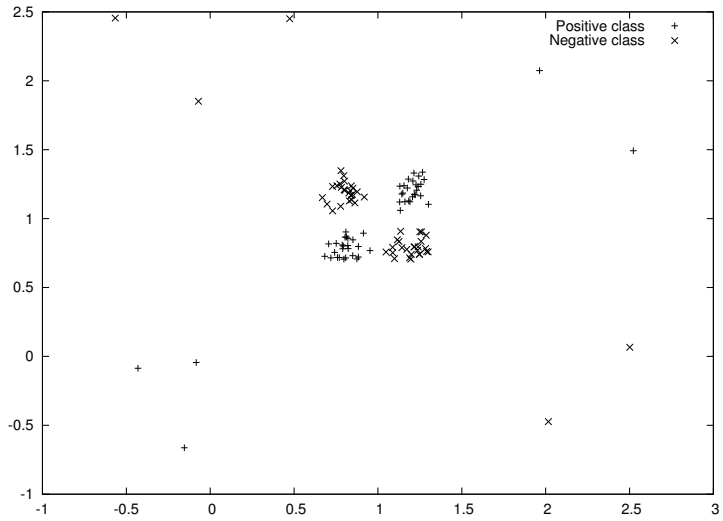


Figure 10.2: Data set containing 4 clusters plus outliers. No points are misclassified within the clusters.

10.5 Numerical example

To investigate how this greedy algorithm performed, we created a data set containing 4 clusters of 25 points each, arranged in an XOR pattern. 10 outliers were added. This is shown in Figure 10.2. The radial basis function was used to create the kernel matrix, with parameter $\gamma = (2\sum_{i=1}^{k} \sigma_i^2)^{-1}$ where σ_i is the standard deviation of attribute i of the original data X.

We compared the greedy heuristic for constructing the partial Cholesky decomposition, by choosing pivots based on the largest diagonal element, and on the largest cost measure m_u (10.3). The performance of the pivot selection methods are shown in Figure 10.3, using the $\|\cdot\|_e$ norm of all non-diagonal elements. The experiments show that when choosing pivots based on diagonal elements, the algorithm does not make fast progress in reducing the error in the approximation until most of the outliers have been chosen. In contrast, using the measure m_u chooses pivots from the clusters to give the best approximation. For example, for an error of 1% of the original kernel matrix, 4 columns are required choosing based on m_u, compared to 14 columns using the largest diagonal element. As the optimization problem associated with SVM training scales to the square of the number of columns r in the partial Cholesky approximation, this represents an order of magnitude difference in the time required for the optimization. Once a few samples have been chosen from each cluster (a total of around 13 or 14 columns), the cost measure is no longer able to clearly identify suitable pivots and the algorithm terminates. The method of choosing largest diagonal elements is able to continue to reduce the error norm further.

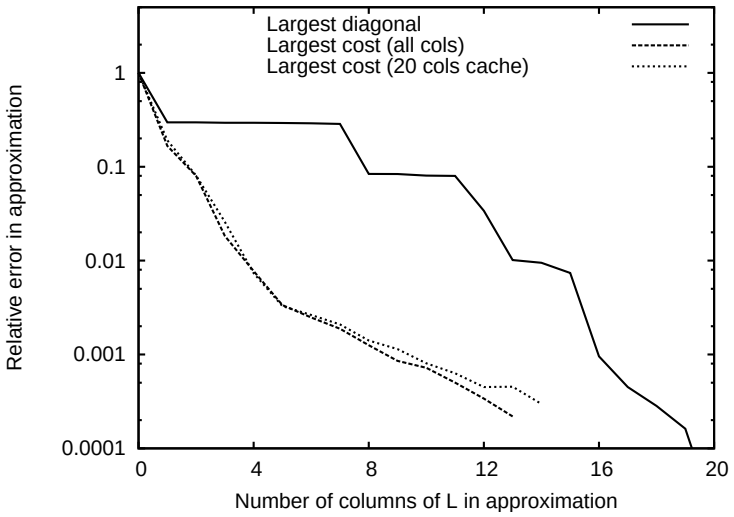


Figure 10.3: The norm of the residual error matrix $\|K-(LL^T+D)\|_e$ against the rank of L in the approximation LL^T+D, using the data set shown in Figure 10.2. The choice of each pivot column based on the cost measure m_u (10.3) is compared against the standard heuristic of choosing the largest diagonal element. Performance of using a cache of 20 columns gives an approximation that is comparable to knowing the whole matrix.

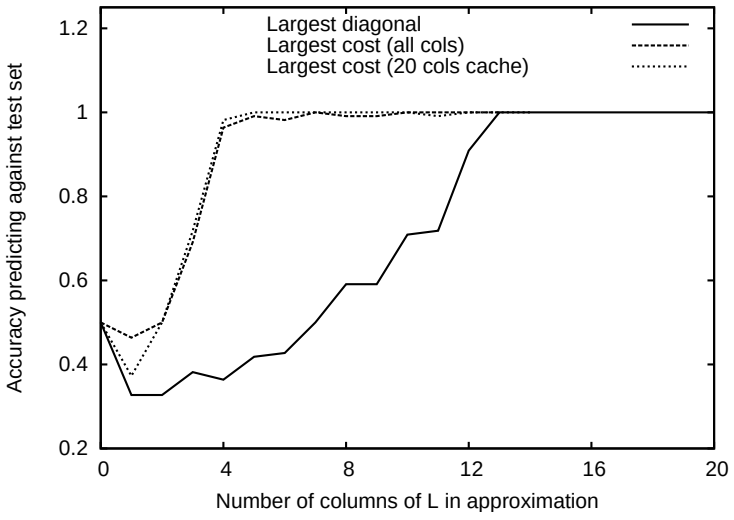


Figure 10.4: Accuracy of SVM training based on the two approximation techniques in classifying an unseen test data set.

As described above, it is computationally too expensive to assess all columns. To avoid this, we maintain a cache of Schur complements of attractive columns. Figure 10.3 shows that, in this particular example, the algorithm using a cache of 20 columns performs almost as well in terms of the approximation matrices it generates as an algorithm that knows the full Schur complement matrix.

We also assessed the accuracy of the resulting SVMs in classifying a previously unseen test set with a similar distribution of data points. The results (Figure 10.4) show that an error of approximately 1% of the original kernel matrix is enough to obtain a very high classification accuracy with this particular data set. This point was reached once a pivot was selected from each of the clusters. By choosing pivots based on our cost measure, the algorithm achieves this level of accuracy with far fewer columns in L than if it chose the largest diagonal elements.

10.6 Conclusion

Non-linear kernels are a powerful extension to the Support Vector Machine technique, allowing them to handle data sets that are not linearly separable. A low-rank approximation of the kernel matrix allows the separable formulation and implementations of previous chapters to be applied to the training of nonlinear SVMs. For efficient operation, it is important to keep the rank of the approximation as low as possible, as the overall algorithm scales with $\mathcal{O}(nr^2)$. Fine and Scheinberg (2001) proposed to use Cholesky decomposition, and to choose pivots by applying the greedy heuristic algorithm of selecting the largest diagonal element each time. We have shown that, for clustered data and the commonly-used RBF kernel, this heuristic will "waste" columns of the low-rank approximation capturing outliers, which carry little information in the kernel matrix, before it starts to capture clusters. From Theorem 10.2.1 the quality of the approximation depends on all elements in the kernel matrix corresponding to support vector pairs. As during the initial approximation the indices of support vectors are not known, we have therefore developed a heuristic for pivot selection that aims to reduce the error on all matrix elements through capturing cluster information, while still restricting the number of kernel computations involved.

While the heuristic we propose is able to generate an approximation of the kernel matrix with fewer columns, it remains an approximation. Theorem 10.2.1 and the Corollary 10.2.3 also suggest that, if high accuracy is really required, an iterative approach to reducing the approximation error could be developed where pivots are chosen only from the set of support

vectors of the last iteration. We have not taken this idea further as yet.

Further work is required for our heuristic approach to become useful in practice. The identification of clusters is an expensive process, and after the main clusters have been represented in the approximation, adding further columns provides little benefit in terms of the final prediction capability. Recognising when this point has been reached is still an open problem. It is worth noting that the kernel matrix is independent of class label information. Therefore the work of identifying clusters may be more worthwhile in multi-class problems, where the low-rank approximation can be reused in each binary classification subproblem.

Another interesting application for very low-rank approximations worth pursuing is in *multiple-kernel learning*. In this application, the matrix K in (3.12) is constructed from a linear combination of kernel matrices K_i, provided that K stays positive semidefinite (Lanckriet et al., 2004). Our techniques can be applied by first calculating a low-rank approximation $K_i \approx L_i L_i^T$ of each contributing kernel matrix K_i, then combining the columns of all the L_i matrices into the matrix L of (9.1). This provides a near-equivalent formulation of multiple-kernel learning without the need for semidefinite programming. It should thus allows multiple-kernel learning to be applied to much larger data sets.

Chapter 11

Applying an algebraic modelling language

Algebraic modelling languages (AML), such as AMPL (Fourer et al., 1993), GAMS (Brooke et al., 1992), AIMMS (Bisschop and Entriken, 1993) or Xpress-Mosel (Colombani and Heipcke, 2002), are important tools in optimization, as they allow the problems to be specified to solvers in a language closer to mathematical notation than would be possible with a programming language. In this chapter we investigate how the structure-conveying modelling language of Grothey et al. (2009); Colombo et al. (2009) could be applied to SVM training problems, and so remove the need for the error-prone work of writing an application interface to a structure-exploiting solver using a programming language. Section 11.2 is a short summary of our additions to the AMPL modelling language. In Section 11.3 they are applied to SVMs.

11.1 Introduction

Modelling languages already pass information to the solver about the sparsity of the matrices that define a problem, but the approach we have used in Chapters 5, 6 and 7 has been related to the block structure of the matrices rather than their sparsity pattern. By block structure we mean the presence of a discernible pattern in the non-zero entries of the Hessian and constraint matrices, where it is possible to subdivide the matrices into blocks, only some of which contain non-zero elements, while the rest are empty. Interior point methods can efficiently exploit the structure to solve large-scale optimization problems, and it therefore seems natural that a goal of a modelling language should be to pass the knowledge about the structure from the modeller to the solver. Much of the specialization of the linear algebra we

have described earlier could then be determined automatically.

Information about the block structure of problems is not immediately apparent from typical AML models. Moreover, AMLs will typically order rows and column of the generated system matrices by constraint and variable set indexing rather than by structural blocks and therefore lose information about the problem structure. One approach is to use heuristics to create an arrowhead structure from the sparsity pattern of the problem matrices (Ferris and Horn, 1998), but these methods are generally computationally expensive, not parallelizable themselves and obtain far from perfect results.

The SET approach of Fragnière et al. (2000) provides general structure-conveying facilities by the modeller declaring an additional structure file to the solver. This file lists the row and column names that make up each block. A disadvantage to this approach is that it involves the solver using repeated string comparisons to unscramble the problem. The authors of AMPL argue in that AMPL's suffix facility can be used to the same effect, but also with the same restrictions (Fourer and Gay, 1999). Other structure-conveying modelling languages have been designed specifically for stochastic programming.

We have addressed these issues by presenting a Structure-conveying Modelling Language (SML) (Grothey et al., 2009). It has been implemented as an extension to AMPL that allows the natural expression of structure present in the problem. [1] The AMPL language is extended by a `block` keyword that groups together model entities and allows them to be repeated over indexing sets. In Colombo et al. (2009) we propose further extensions for stochastic programming, but these are not relevant to the discussion here.

11.2 Language design

The main contribution of SML is the introduction of the `block` keyword, which is used to define a sub-model. Its power comes from the fact that any `block` can be repeated by indexing it over a set. A sub-model definition using the `block` command takes the following form

```
block nameofblock{j in SET}: {
  ...
}
```

Within the scope delimited by `block { ... }` , any number of `set, param, subject to, var, minimize` or indeed further `block` commands can be placed. The understanding is

[1] SML is available for research use at http://www.maths.ed.ac.uk/ERGO/sml/.

that all declarations placed inside the block environment are implicitly repeated over the indexing expression used for the `block` command, resulting in a tree structure of blocks.

Using the principle of encapsulation from object-oriented programming languages, a `block` command defines a scope: within it, entities (sets, variables, constraints) defined inside this block or in any of its parents can be used. Entities defined in the block can be referenced from outside by using the form `nameofblock[j].nameofentity`. Entities defined in nodes that are not direct children or ancestors of the current block cannot be used.

The syntax for objectives is also modified to allow for the inclusion of block-local variables. Objective declaration (through a `minimize` or `maximize` statement) can be placed anywhere in the model, and named. The objective that is passed on to the solver is then the sum of all objectives declared in the submodels with the same name.

11.3 Applying SML to SVM problems

Figure 11.1 shows how the standard SVM binary classification problem can be modelled using SML, based on the separable formulation (5.1). In this model, we assume that the data sets are divided into files. The blocks called `file` provide information to a parallel solver that this is how to partition the problem. The `weights` block contains the weights w as local variables. There are contributions to the objective in all the blocks. The main purpose behind this structure is to indicate to the solver the structure of the Hessian matrix: if nonlinear elements of the objective occur only locally within blocks, the Hessian must have a separable block-diagonal structure.

Constraints that are defined outside of blocks, but that use variables defined within blocks, are "linking constraints". They indicate to the solver that they should form a block of rows at the bottom of the augmented system matrix (7.1), and the preferred linear algebra technique is to reduce the system through the Schur complement method to the smaller dense matrix M in (5.2).

11.4 Discussion

In this chapter, we have used the structure-conveying algebraic modelling language presented in Grothey et al. (2009); Colombo et al. (2009) to formulate models for standard SVM training problems. The aim of the language is to allow models to be expressed elegantly, while at the same time capturing information about their inherent structure that can guide the

```
set FILES;
param M > 0;
param Tau > 0;
set FEATURES = {1..M};

block file {f in FILES} {
  param N_f > 0; // Number of samples in this file
  set SAMPLES = {1..N_f};
  param x{SAMPLES,FEATURES};
  param y{SAMPLES};
  var z{SAMPLES} >= 0, <= Tau;

  minimize obj: - sum{i in SAMPLES} z[i];
}

block weights {
  var w{FEATURES};
  minimize obj: 0.5 * sum{j in FEATURES} w[j]*w[j];
}

// Linking constraints form block row border
subject to {j in FEATURES}:
  weights.w[j] = sum{ f in FILES, i in SAMPLES[f] }
                 file[f].x[i,j]*file[f].y[i]*file[f].z[i];

subject to:
  sum{ f in FILES, i in SAMPLES[f] } file[f].y[i]*file[f].z[i] = 0;
```

Figure 11.1: SML model for binary SVM classification problem.

solver automatically towards efficient linear algebra operations. The SML model for L1-SVM binary classification was shown in Figure 11.1. Appendix B contains models in SML for other SVM problems described in Chapter 5, using the separable QP formulations: L2-SVM classification (see Section 5.2, and Figure B.1 for the model); Universum classification (Section 5.4, Figure B.2); ordinal regression (Section 5.5, Figure B.3); and metric regression (Section 5.6, Figure B.4).

These examples do show how the problem can be provided to a solver in a higher-level and more readable way than directly interfacing with the software through a programming language. Unfortunately it does introduce other complications; in particular, care needs to be taken in setting up the data files. Ordinal regression requires the ordering of the sets to be controlled, and the data are more complicated, but then this is a reflection of the fact that the structure of the problem itself is more complicated.

On the whole, the modelling language does successfully capture the structure of SVM classification training problems, and with some guidance or detection to use dense matrices to store matrix X (currently not implemented), the solver could be led to automatically choose the approach described in Section 7.1. An exception is metric regression, where the block eliminations described in Section 5.6 are not expressed in the SML model.

There are some weaknesses in this, more automated approach. The first is that the hyperplane bias w_0 is missing from the formulation. It is actually the dual variable for the constraint $y^T z = 0$ in (5.1). The interface between modelling language and solver will typically have the features to recover dual variable information, but perhaps w_0 is rather hidden in the model.

A second weakness concerns nonlinear SVMs. In a standard modelling language that supports nonlinear optimization (such as AMPL), it would be natural to write the optimization problem as (3.12), and let the function evaluation capabilities of the AML software compute kernel values as required. In contrast, Section 9.1 proposes the low-rank linearization of the kernel matrix as a data pre-processing stage that runs before the AML software.

In general, a structure-conveying modelling language is not a "silver bullet", in that it cannot replace a carefully-worked reformulation of the problem designed to reduce computational effort: writing SML models for the original SVM training problem in either its primal form (3.4) or dual form (3.6) would not have led automatically to the separable formulation (5.1) that has driven much of this thesis.

Chapter 12

Conclusions

The subject of this thesis has been how to apply interior point methods to large-scale SVM training problems. SVMs are a powerful machine learning technique, but extending them to very large-scale problems is not trivial.

Traditionally, active-set methods have been used rather than interior point methods, due to the Hessian in the standard dual formulation (3.6) being completely dense. Active set methods work well for small and separable problems, but when the split between basic and non-basic variables becomes less clear (as is the case with noisy data sets) the performance of these algorithms starts to scale exponentially with the number of samples. Previous IPM-based approaches have exploited the structure of the linear kernel, to give algorithms with an overall complexity of $\mathcal{O}(nm^2)$. However, these algorithms have suffered from either numerical instability through use of the Sherman-Morrison-Woodbury formula, or memory caching inefficiencies.

In Chapter 5 we presented a new, unified family of formulations for the linear cases of 1-norm and 2-norm classification, ν-SVM, universum and ordinal classification, and ϵ-insensitive regression. All exact reformulations of the original problems, they exploit the separability of the Hessian in the objective of the standard SVM primal formulation while keeping the number of constraints small. Like other IPM-based approaches, this approach has a per-iteration complexity of $\mathcal{O}(nm^2 + m^3)$. It relies upon Cholesky decomposition for its numerical stability. All m features are considered simultaneously during the normal matrix calculation procedure, allowing for a more efficient implementation in terms of memory caching.

Numerical experiments (Chapter 6) showed that the performance of our algorithm for large dense or noisy data sets is consistent and highly competitive, and in some cases can

surpass all other approaches by a large margin. Unlike active set methods, performance is largely unaffected by noisy data. Using multiple correctors, tightening the bounds on w, and monitoring the angle of the normal to the hyperplane all positively contributed to efficiency. Only LIBLINEAR (Fan et al., 2008) rivals our algorithm for performance, and that uses a simplified approximation to the SVM problem where the constraint is removed (see Section 4.5).

It is also possible to develop our algorithm to handle very large-scale problems in parallel, and in Chapter 7 we developed a hybrid MPI/OpenMP implementation of the algorithm to handle such problems using a clustered computing environment. The approach allowed the entire data set to be used. We exploited the block structure of the augmented system matrix, to partition the data and linear algebra computations amongst parallel processing nodes efficiently. Most matrix operations were performed in parallel with no communication required between the processors, using a highly efficient BLAS implementation for a multi-threaded SMP architecture. The results from our implementation showed that, for all cases, the hybrid implementation was faster than one using purely MPI, even though the MPI version had better parallel efficiency. We used the hybrid implementation to solve very large problems from the PASCAL Challenge on Large-scale Learning, of up to a few million data samples. Our approach was highly competitive on these problems, and showed that even on data sets of this size, training times in the order of minutes are possible.

The method described above requires all sample data to be loaded into memory, and has computational complexity of $\mathcal{O}(nm^2)$. In Chapter 8 we investigated techniques to reduce the number of data points considered by the IPM solver. For data sets that were noisy or linearly nonseparable, we found that if the size of the working set was reduced too far, many iterations were required and the performance of the algorithm resembled that of active set methods. A better approach was to find an initial approximation to the hyperplane, use it to remove a smaller number of samples (where we are confident they are far from the hyperplane margin), and then use the strength of our IPM formulation to handle the more difficult final set partitioning.

The techniques described so far are limited to linear kernels. In Chapter 9, the technique is extended to handle non-linear kernels, by approximating the kernel with a low-rank outer product representation such as partial Cholesky factorization (Fine and Scheinberg, 2001). The rank r of the approximation is arguably more important than the number of samples n in determining the computational complexity, as the overall algorithm scales with $\mathcal{O}(nr^2)$. Chapter 10 introduced a metric that aims to ensure that the columns chosen to build a partial

Cholesky factorization capture as much of the matrix information as possible.

In Chapter 11, we used a structure-conveying algebraic modelling language (Grothey et al., 2009; Colombo et al., 2009) to formulate models for standard SVM training problems. On the whole, the modelling language does successfully capture the structure of SVM classification training problems, and the information could guide the solver to choose the approach described in Section 7.1. These examples do show how the problem can be provided to a solver in a higher-level and more readable way than directly interfacing with the software through a programming language. Unfortunately setting up the data files becomes more complicated. The investigation also highlights the limits of the methodology. It conveys the structure of the problem as described using the modelling language, and it cannot devise an efficient reformulation by itself. Writing SML models for the original SVM training problem in either its primal form (3.4) or dual form (3.6) would not have led automatically to the separable formulation (5.1) that has driven much of this thesis. In this sense, the language cannot replace a carefully-worked reformulation of the problem designed to reduce computational effort.

In this work, we have concentrated solely on solving the linear L1-SVM training QP for binary classification, since it can be considered the root problem. It would be interesting to implement the formulations described in Chapter 5 for the other training problems in the SVM family, and ensure that they too provide efficient training; for instance, ν-SVM has been little used because of a lack of efficient training implementations available (Chen et al., 2005).

For our methods to be applied to nonlinear kernels in practice, further work is required on approximating the kernel matrices. The identification of clusters is a computationally expensive process, and it is not yet clear when to stop. Identifying clusters may be more beneficial in multi-class problems, since the kernel matrix is independent of the labels and so the approximation can be reused in each binary classification subproblem.

The scope of this thesis has concentrated on the strict interpretation of the L1-SVM training problem as an equality and box-constrained QP, in line with the original description of Vapnik (1998). Since then, there has been much research to find optimization problems that approximate the standard SVM QP but are faster to solve (see Section 4.5). In Chapter 8 we proposed the QP (8.2) as another such approximation, which could be used either stand-alone or prior to the optimization of hyperplane offset and margin width. It is possible that the IPM techniques and separable formulations in this thesis could provide more such opportunities to find good approximations. Another current direction for SVM research is multiple-kernel learning. Collecting together into L the columns that form low-rank approxi-

mations of each contributing kernel matrix and then solving (9.1), provides an equivalent technique without the need for semidefinite programming. Here is a route to applying multiple-kernel learning to much larger data sets than is currently possible.

In conclusion, interior point methods are a viable optimization technology to apply to large-scale SVM training, provided the issues related to computational complexity are properly addressed through problem formulation and through efficient implementation techniques. They are suitable for parallel and clustered environments. They provide good early approximations, a benefit if training time is limited. We have concentrated our investigations on the linear L1 SVM, but it is clear that interior point methods have the potential to enable many other SVM-related techniques to be applied at the large scale.

Bibliography

A. Altman and J. Gondzio. Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization. *Optimization Methods and Software*, 11:275–302, 1999.

E. Andersen, J. Gondzio, C. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior point methods in mathematical programming*, pages 189–252. Kluwer Academic Publishers, 1996.

F. R. Bach and M. I. Jordan. Predictive low-rank decomposition for kernel methods. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 33–40, New York, NY, USA, 2005. ACM.

K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

J. Bisschop and R. Entriken. *AIMMS The Modeling System*. Paragon Decision Technology, 1993.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In Z. Ghahramani, editor, *Proceedings of the 24th International Machine Learning Conference*, pages 89–96, Corvallis, Oregon, 2007. OmniPress. URL http://leon.bottou.org/papers/bordes-2007.

P. S. Bradley and O. L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13(1):1–10, 2000.

A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, Redwood City, California, 1992.

A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35(1):38–53, January 2009.

C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui. Parallelizing support vector machines on distributed computers. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 257–264, Cambridge, MA, 2008. MIT Press.

P.-H. Chen, C.-J. Lin, and B. Schölkopf. A tutorial on ν-support vector machines. *Applied Stochastic Models in Business and Industry*, 21(2):111–136, 2005.

E. Cheney and A. Goldstein. Newton's method for convex programming and Tchebycheff approximation. *Numer. Math.*, 1:253–268, 1959.

W. Chu and S. S. Keerthi. New approaches to support vector ordinal regression. In *ICML '05: Proceedings of the 22nd international conference on machine learning*, pages 145–152, New York, NY, USA, 2005. ACM.

R. Collobert and S. Bengio. SVMTorch: support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002.

Y. Colombani and S. Heipcke. Mosel: an extensible environment for modeling and programming solutions. In N. Jussien and F. Laburthe, editors, *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 277–290, Le Croisic, France, March, 25–27 2002.

M. Colombo and J. Gondzio. Further development of multiple centrality correctors for interior point method. *Computational Optimization and Applications*, 41(3):277–305, 2008.

M. Colombo, A. Grothey, J. Hogg, K. Woodsend, and J. Gondzio. A structure-conveying modelling language for mathematical and stochastic programming. Technical Report ERGO 09-003, School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, March 2009.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

Y.-H. Dai and R. Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Mathematical Programming*, 106(3):403–421, May 2006.

G. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

G. Dantzig and P. Wolfe. Decomposition principles for linear program. *Operations Research*, 8(1):101–111, Jan-Feb 1960.

E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–13, 2002.

J. Dong, A. Krzyzak, and C. Suen. A fast parallel optimization for training support vector machine. In P. Perner and A. Rosenfeld, editors, *Proceedings of 3rd International Conference on Machine Learning and Data Mining*, pages 96–105, Leipzig, Germany, 2003. Springer Lecture Notes in Artificial Intelligence.

P. Drineas and M. W. Mahoney. On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

M. F. Duarte and Y. H. Hu. Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):826–838, July 2004.

I. Durdanovic, E. Cosatto, and H.-P. Graf. Large scale parallel SVM implementation. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, chapter 5, pages 105–38. MIT Press, 2007.

A. S. El-Bakry, R. A. Tapia, and Y. Zhang. A study of indicators for identifying zero variables in interior-point methods. *SIAM Review*, 36(1):45–72, 1994.

T. Evgeniou and M. Pontil. Support vector machines with clustering for training with very large datasets. In *Methods and Applications of Artificial Intelligence*, page 751. Springer-Verlag, 2002.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

M. C. Ferris and D. J. Horn. Partitioning mathematical programs for parallel solution. *Mathematical Programming*, 80:35–62, 1998.

M. Ferris and T. Munson. Interior point methods for massive support vector machines. *SIAM Journal on Optimization*, 13(3):783–804, 2003.

A. V. Fiacco and G. P. McCormick. *Nonlinear programming: sequential unconstrained minimization techniques*. Wiley, New York, 1968.

S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

S. Fine and K. Scheinberg. INCAS: An incremental active set method for SVM. Technical report, IBM Research Labs, Haifa, 2002.

R. Fletcher. *Practical methods of optimization*. Wiley, Chichester, 2nd edition, 1987.

R. Fourer and D. Gay. Conveying problem structure from an algebraic modelling language to optimization algorithms. Available at `citeseer.ist.psu.edu/236340.html`, accessed 20 February 2008, 1999.

R. Fourer, D. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press, San Francisco, California, 1993.

E. Fragnière, J. Gondzio, R. Sarkissian, and J.-P. Vial. Structure exploiting tool in algebraic modeling languages. *Management Science*, 46:1145–1158, 2000.

V. Franc and S. Sonnenburg. OCAS: optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25nd International Conference on Machine Learning (ICML)*. ACM Press, 2008.

K. R. Frisch. The logarithmic potential method of convex programming. Memorandum, University Institute of Economics, Oslo, May 13 1955.

G. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In F. Provost and R. Srikant, editors, *Proceedings KDD-2001: Knowledge Discovery and Data Mining, August 26-29, 2001, San Francisco, CA*, pages 77–86, New York, 2001. Asscociation for Computing Machinery.

E. M. Gertz and J. D. Griffin. Using an iterative linear solver in an interior-point method for generating support vector machines. *Computational Optimization and Applications*, 2009. doi: 10.1007/s10589-008-9228-z.

E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1):58–81, 2003.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem-part ii. *Operations Research*, 11(6):863–888, Nov-Dec 1963.

J.-L. Goffin and J.-P. Vial. Convex nondifferentiable optimization: a survey focussed on the analytical center cutting plane method. *Optimization Methods and Software*, 1999.

D. Goldfarb and K. Scheinberg. A product-form Cholesky factorization method for handling dense columns in interior point methods for linear programming. *Mathematical Programming*, 99(1):1–34, 2004.

D. Goldfarb and K. Scheinberg. Solving structured convex quadratic programs by interior point methods with application to support vector machines and portfolio optimization. *Submitted for publication*, 2005.

G. H. Golub and C. F. van Loan. *Matrix Computations*. North Oxford Academic, 1st edition, 1983.

J. Gondzio. HOPDM: a fast LP solver based on a primal-dual interior point method. *European Journal of Operational Research*, 85:221–225, 1995.

J. Gondzio. Multiple centrality corrections in a primal-dual method for linear programming. *Computational Optimization and Applications*, 6:137–156, 1996.

J. Gondzio and A. Grothey. Parallel interior point solver for structured quadratic programs: Application to financial planning problems. *Annals of Operations Research*, 152(1):319–339, 2007.

J. Gondzio and R. Sarkissian. Parallel interior point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584, 2003.

J. Gondzio and R. Sarkissian. Column generation with the primal-dual method. Logilab Technical Report 96.4, Department of Management Studies, University of Geneva, Switzerland, June 1996.

K. Goto and R. A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3):1–25, 2008.

H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: the Cascade SVM. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2005. volume 17.

A. Grothey, J. Hogg, K. Woodsend, M. Colombo, and J. Gondzio. A structure-conveying parallelisable modelling language for mathematical programming. In R. Ciegis, D. Henty, B. Kågström, and J. Žilinskas, editors, *Parallel Scientific Computing and Optimization: Advances and Applications*, volume 27 of *Springer Optimization and Its Applications*, pages 147–158. Springer-Verlag, Berlin, 2009.

J. Hall and K. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational Optimization and Applications*, 32(3):259–283, December 2005.

R. Herbrich. *Learning Kernel Classifiers: Theory and Algorithms*. Adaptive computation and machine learning. MIT, Cambridge, Massachusetts, 2002.

R. Herbrich, T. Graepel, and K. Obermayer. *Advances in Large Margin Classifiers*, chapter Large margin rank boundaries for ordinal regression. MIT Press, 2000.

J. R. Herrero. *A Framework for Efficient Execution of Matrix Computations*. PhD thesis, Universitat Politècnica de Catalunya (UPC), July 2006.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML '08: Proceedings of the 25th international conference on machine learning*, 2008.

T. Joachims. Training linear SVMs in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, New York, NY, USA, 2006. ACM.

T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 169–184. MIT Press, 1999.

J. H. Jung, D. P. O'Leary, and A. L. Tits. Adaptive constraint reduction for training support vector machines. *Electronic Transactions on Numerical Analysis*, 31:156–177, 2008.

B. Kågström, P. Ling, and C. van Loan. GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Softw.*, 24 (3):268–302, 1998.

N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4: 373–395, 1984.

L. Kaufman. Solving the quadratic programming problem arising in support vector classification. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 147–167. MIT Press, 1999.

S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6:341–361, 2005.

S. S. Keerthi, O. Chapelle, and D. DeCoste. Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research*, 7:1493–1515, 2006.

J. E. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8: 703–712, 1960.

K. C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46(1):105–122, January 1990. doi: 10.1007/BF01585731.

B. Kulis, M. A. Sustik, and I. S. Dhillon. Learning low-rank kernel matrices. In *Proceedings of the Twenty-third International Conference on machine Learning*, pages 505–512. ICML, June 2006.

G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. Jordan. Learning the Kernel Matrix with Semidefinite Programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Algorithm 539: Basic Linear Algebra Subprograms for Fortran usage [F1]. *ACM Transactions on Mathematical Software*, 5(3): 324–325, September 1979.

Y. J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *Proceedings of the SIAM International Conference on Data Mining*, Philadelphia, 2001. SIAM.

C. Lemaréchal. *Computational Combinatorial Optimization*, chapter Lagrangian Relaxation, pages 112–156. Springer-Verlag, 2001.

M. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

S. Lucidi, L. Palagi, A. Risi, and M. Sciandrone. A convergent decomposition algorithm for support vector machines. *Computational Optimization and Applications*, 38:217–234, 2007.

G. R. Luecke, J. H. Yun, and P. W. Smith. Performance of parallel cholesky factorization algorithms using blas. *The Journal of Supercomputing*, 6(3):315–329, December 1992.

O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10(5):1032–1037, 1999.

O. L. Mangasarian and D. R. Musicant. Active support vector machine classification. In *Advances in Neural Information Processing Systems 13*, pages 577–583, 2000.

S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

C. Mészáros. The separable and non-separable formulations of convex quadratic problems in interior point methods. Technical Report WP 98-3, Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, 1998a.

C. Mészáros. On free variables in interior point methods. *Optimization Methods and Software*, 9(1):121–139, 1998b.

S. Mizuno. Polynomiality of infeasible interior point algorithms for linear programming. *Mathematical Programming*, 67:109–119, 1994.

MPI-Forum. *MPI: A Message-Passing Interface Standard.* University of Tennessee, Knoxville, Tennessee, 1.1 edition, June 1995. URL `http://www.mpi-forum.org`.

J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, August 1999.

A. B. Novikoff. On convergence proofs for perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1963.

OpenMP Architecture Review Board. *OpenMP Application Program Interface*, 3.0 edition, May 2008. URL `http://www.openmp.org`.

E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE, 1997.

J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 185–208. MIT Press, 1999.

M. Pontil and A. Verri. Properties of support vector machines. Technical report, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1997.

J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1993.

R. Rabenseifner and G. Wellein. Comparison of parallel programming models on clusters of SMP nodes. In H. G. Bock, E. Kostina, H. X. Phu, and R. Rannacher, editors, *Modelling, Simulation and Optimization of Complex Processes: Proceedings of the International Conference on High Performance Scientific Computing, March 10-14, 2003, Hanoi, Vietnam*, pages 409–426. Springer, 2003.

C. Roos, T. Terlaky, and J.-P. Vial. *Interior point methods for linear optimization*. Springer, revised edition edition, 2005.

C. Roos. A full-Newton step $O(n)$ infeasible interior-point algorithm for linear optimization. *SIAM Journal on Optimization*, 16(4):1110–1136, 2006.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov 1958.

B. Schölkopf, A. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 2000.

B. Schölkopf and A. J. Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond*. MIT Press, Cambridge, MA, 2002.

T. Serafini, G. Zanghirati, and L. Zanni. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods and Software*, 20(2):353–378, 2005.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy. Improvements to the SMO algorithm for SVM regression. *IEEE Transactions on Neural Networks*, 11(5), September 2000.

L. Smith and M. Bull. Development of mixed mode MPI / OpenMP applications. *Scientific Programming*, 2001.

A. Smola, S. V. N. Vishwanathan, and Q. Le. Bundle methods for machine learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1377–1384. MIT Press, Cambridge, MA, 2008.

A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 911–918, CA, 2000. Stanford University, Morgan Kaufmann.

S. Sonnenburg, V. Franc, E. Yom-Tov, and M. Sebag. PASCAL large scale learning challenge, 2008. URL `http://largescale.first.fraunhofer.de/`.

S. Sra. Efficient large scale linear programming support vector machines. In *Machine Learning: ECML 2006*, pages 767–774. Springer, 2006.

I. Steinwart. Sparseness of support vector machines—some asymptotically sharp bounds. In *Proceedings of the 16th NIPS Conference*, pages 169–184, 2004.

A. Tveit and H. Engum. Parallelization of the incremental proximal support vector machine classifier using a heap-based tree topology. Technical report, IDI, NTNU, Trondheim, 2003.

R. J. Vanderbei. *Linear Programming Foundations and Extensions*. Kluwer, Boston, 1997.

V. Vapnik. Transductive inference and semi-supervised learning. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-supervised learning*, chapter 24, pages 454–472. MIT Press, 2006.

V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2nd edition, 1999.

A. R. Webb. *Statistical pattern recognition*. Wiley, Chichester, UK, 2nd edition, 2002.

P. Wentges. Weighted dantzig-wolfe decomposition of linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.

J. Weston, R. Collobert, F. Sinz, L. Bottou, and V. Vapnik. Inference with the Universum. In *ICML '06: Proceedings of the 23rd international conference on machine learning*, pages 1009–1016. ACM, 2006.

C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. *Advances in Neural Information Processing Systems*, 13:682–688, 2001.

K. Woodsend and J. Gondzio. High-performance parallel support vector machine training. In R. Ciegis, D. Henty, B. Kågström, and J. Žilinskas, editors, *Parallel Scientific Computing and Optimization: Advances and Applications*, volume 27 of *Springer Optimization and Its Applications*, pages 83–92. Springer-Verlag, Berlin, 2009a.

K. Woodsend and J. Gondzio. Exploiting separability in large-scale linear support vector machine training. *Computational Optimization and Applications*, 2009b. doi: 10.1007/s10589-009-9296-8.

K. Woodsend and J. Gondzio. Hybrid MPI/OpenMP parallel linear support vector machine training. *Journal of Machine Learning Research*, 10:1937–1953, Aug 2009c.

S. J. Wright. *Primal-dual interior-point methods*. S.I.A.M., 1997.

Y. Ye. *Interior point algorithms : theory and analysis*. Wiley, 1997.

G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Computing*, 29(4):535–551, 2003.

L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7:1467–1492, 2006.

Appendix A

Test accuracy for Challenge datasets

This appendix contains test accuracy versus training time results from the PASCAL Challenge on Large-scale Learning. Tests were performed independently by the Challenge organizers, using test data sets that were not disclosed to the participants. Charts are taken from the Challenge website (Sonnenburg et al., 2008, accessed 26 March 2009), and compare our implementation (*IPM SVM 2*) to the other L1-SVM codes that participated. The tests were performed on the same single multi-core processor, which restricted our code to using only the techniques described in Section 7.1.2.

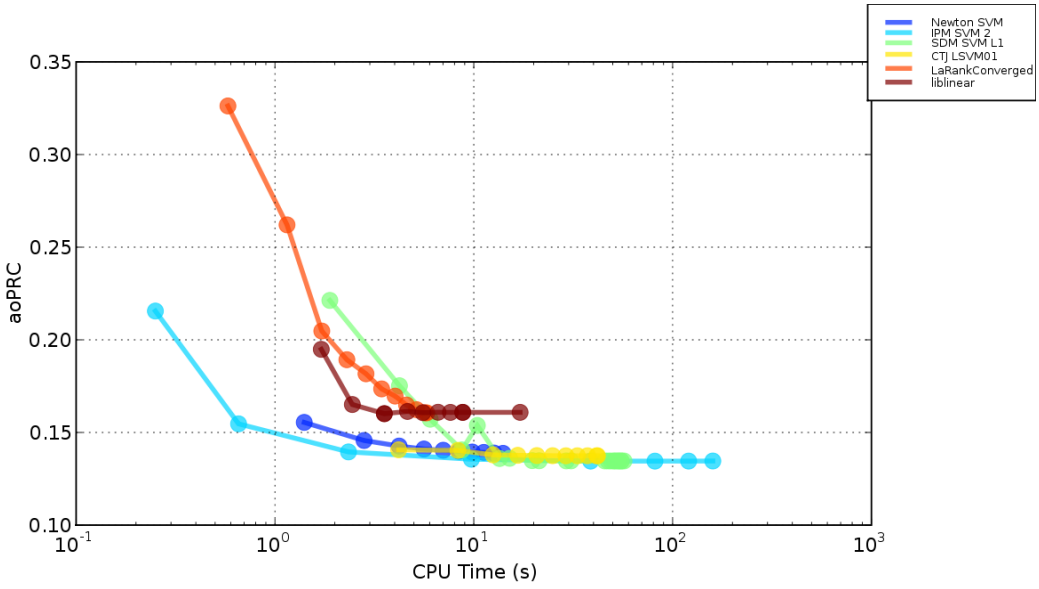


Figure A.1: Alpha data set.

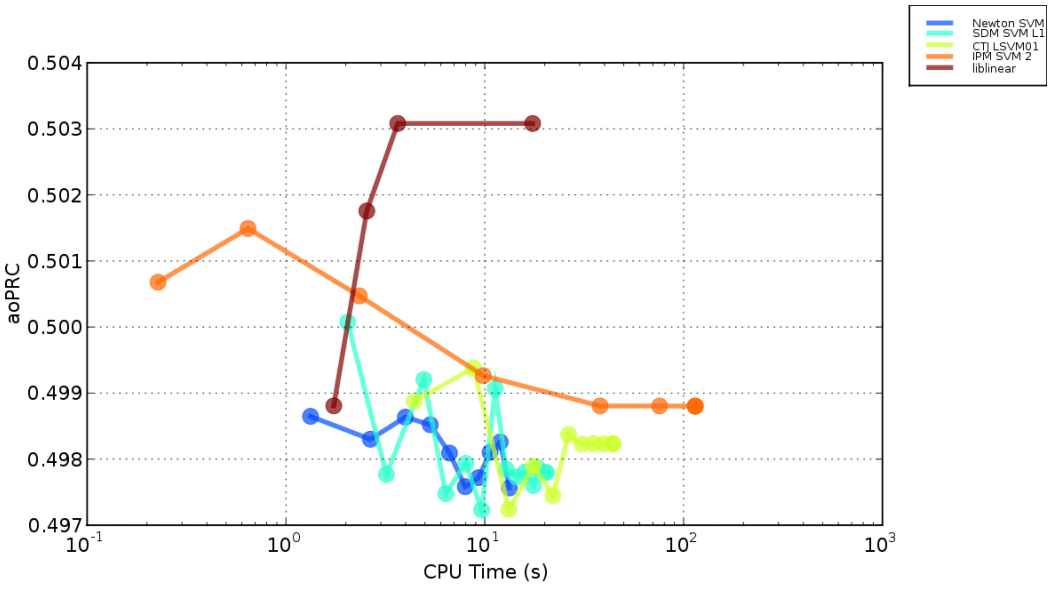


Figure A.2: Beta data set.

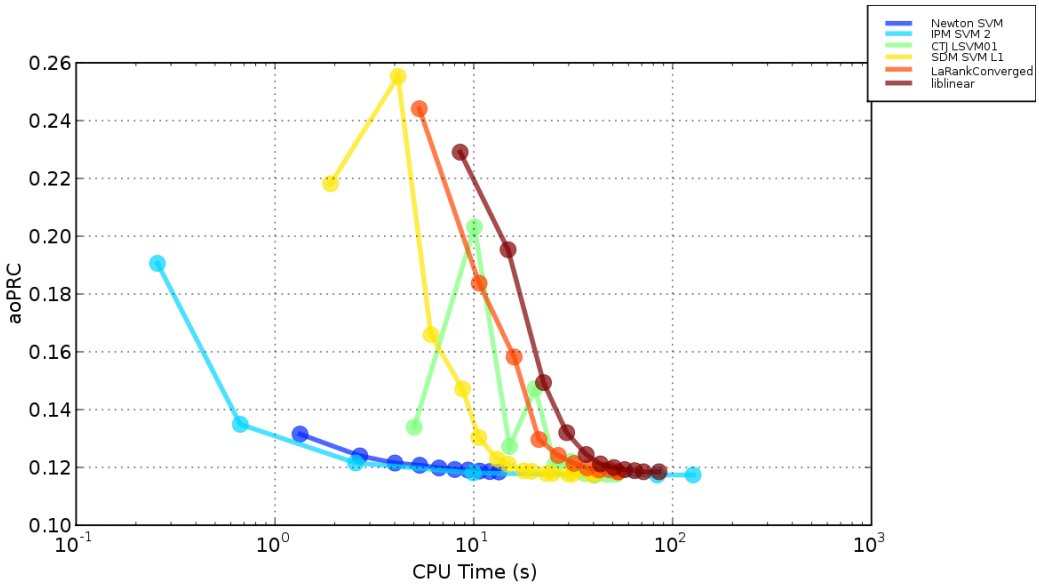


Figure A.3: Gamma data set.

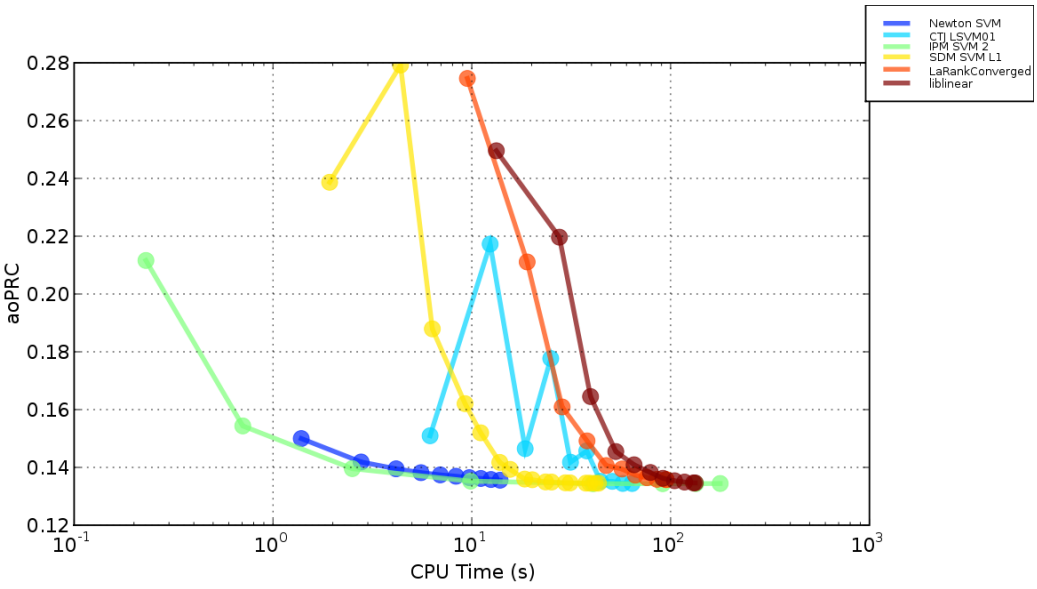


Figure A.4: Delta data set.

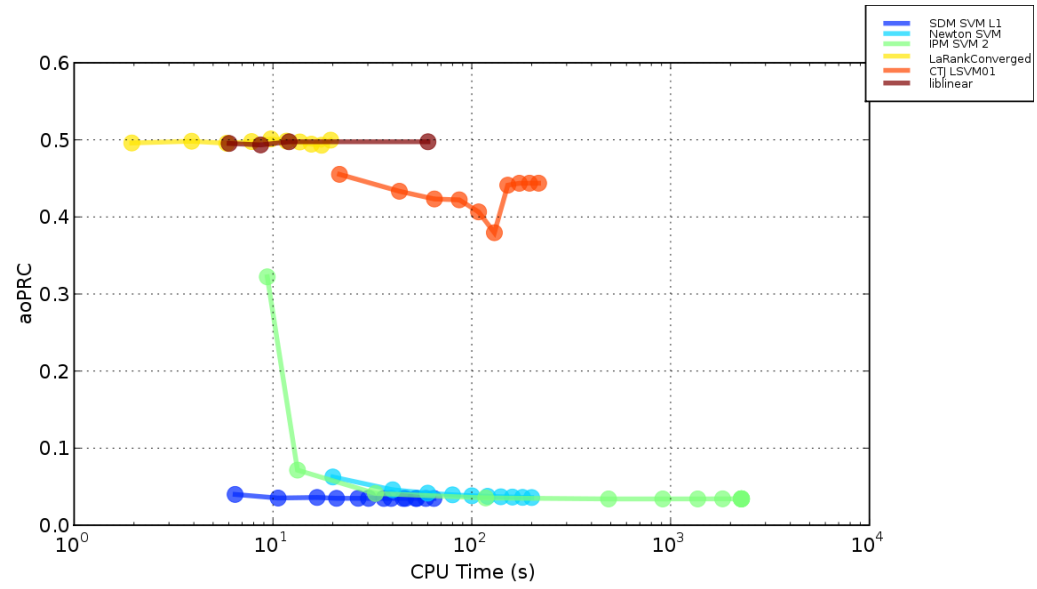


Figure A.5: Epsilon data set.

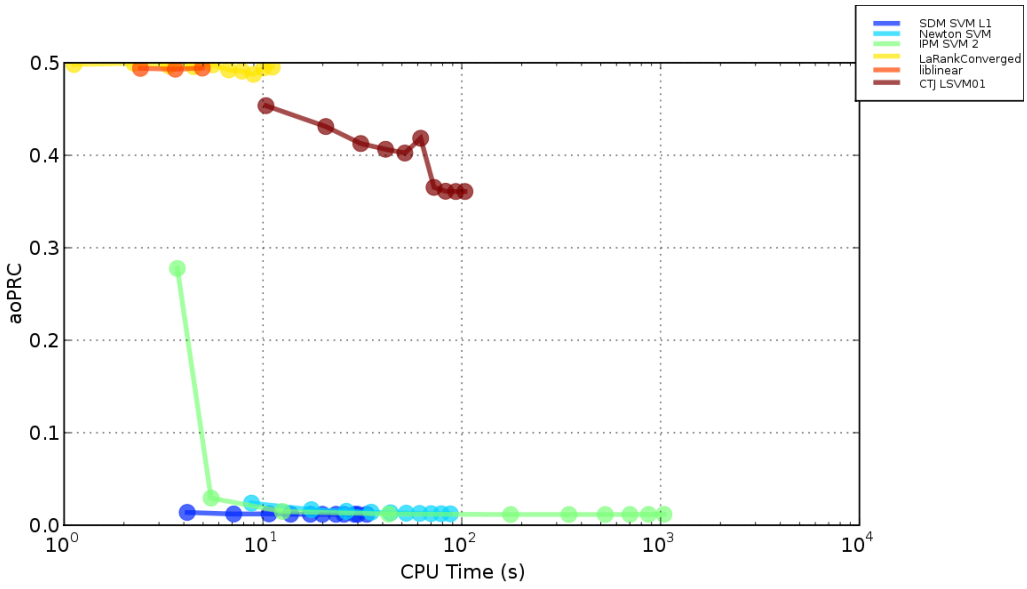


Figure A.6: Zeta data set.

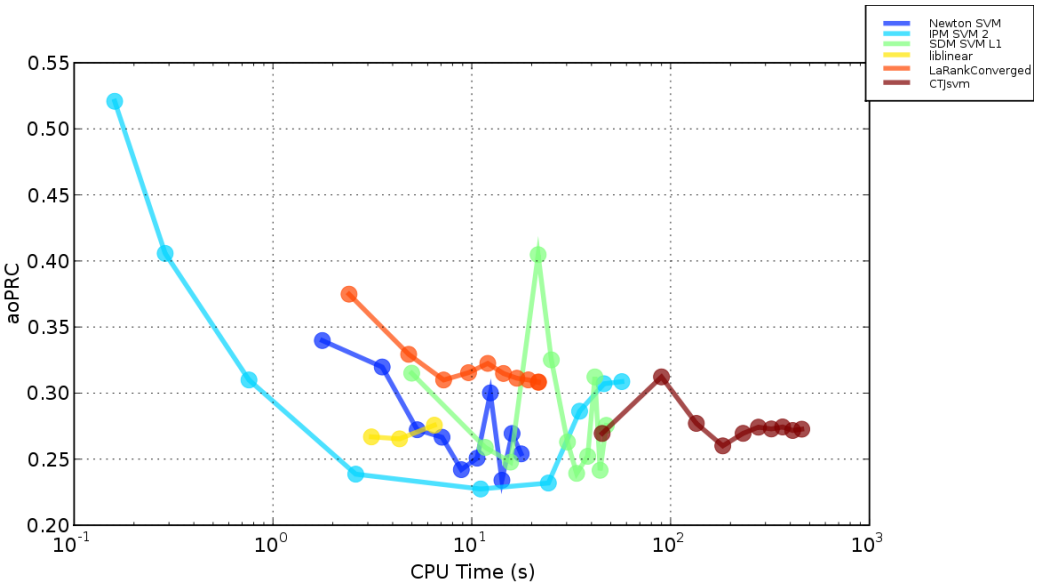


Figure A.7: Face detection data set.

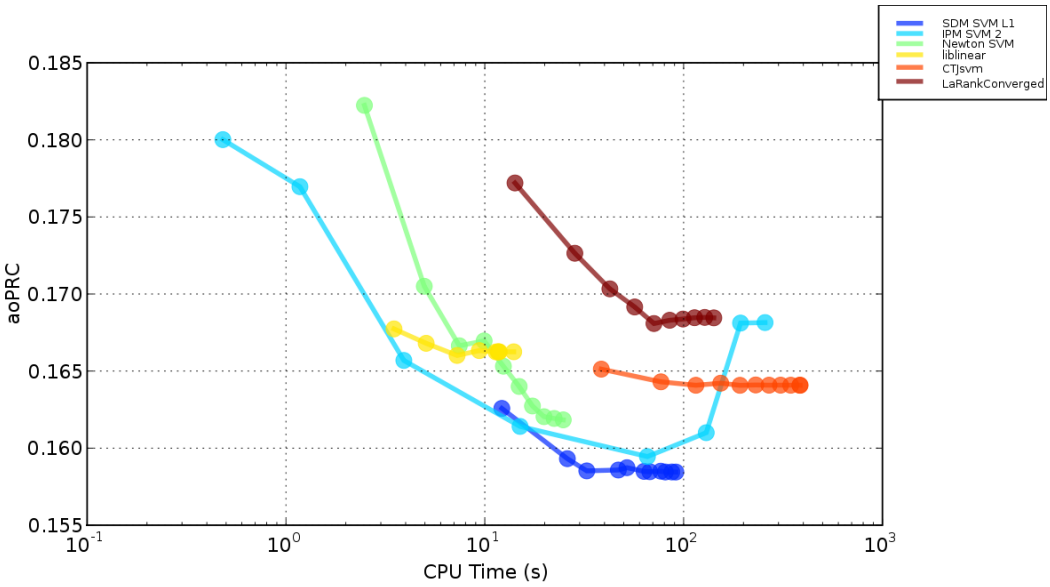


Figure A.8: OCR data set.

Appendix B

SVM training problems written in SML

This appendix contains models for some of the SVM formulations of Chapter 5, written in the structure-conveying algebraic modelling language discussed in Chapter 11.

```
set FILES;
param M > 0;
param Tau > 0;
set FEATURES = {1..M};

block file {f in FILES} {
  param N_f > 0; // Number of samples in this file
  set SAMPLES = {1..N_f};
  param x{SAMPLES,FEATURES};
  param y{SAMPLES};
  var z{SAMPLES} >= 0, <= Tau;

  minimize obj: (1/Tau) * sum{i in SAMPLES} (z[i]*z[i] - z[i]);
}

block weights {
  var w{FEATURES};
  minimize obj: 0.5 * sum{j in FEATURES} w[j]*w[j];
}

// Linking constraints form block row border
subject to {j in FEATURES}:
  weights.w[j] = sum{ f in FILES, i in SAMPLES[f] }
                 file[f].x[i,j]*file[f].y[i]*file[f].z[i];

subject to:
  sum{ f in FILES, i in SAMPLES[f] } file[f].y[i]*file[f].z[i] = 0;
```

Figure B.1: SML model for L2-SVM classification, based on (5.3).

```
set CLASS_FILES, UNIVER_FILES;
param M > 0;
param Tau_C > 0, Tau_U > 0, Epsilon > 0;
set FEATURES = {1..M};

block weights {
  var w{FEATURES};
  minimize obj: 0.5 * sum{j in FEATURES} w[j]*w[j];
}

block class_file {f in CLASS_FILES} {
  param N_f > 0; // Number of samples in this file
  set SAMPLES = {1..N_f};
  param x{SAMPLES,FEATURES};
  param y{SAMPLES};
  var z{SAMPLES} >= 0, <= Tau_C;

  minimize obj: - sum{i in SAMPLES} z[i];
}

block univer_plus_file {f in UNIVER_FILES} {
  param N_f > 0; // Number of samples in this file
  set SAMPLES = {1..N_f};
  param x{SAMPLES,FEATURES};
  var z{SAMPLES} >= 0, <= Tau_U;

  minimize obj: + Epsilon * sum{i in SAMPLES} z[i];
}

block univer_minus_file {f in UNIVER_FILES} {
  param N_f > 0; // Number of samples in this file
  set SAMPLES = {1..N_f};
  param x{SAMPLES,FEATURES};
  var z{SAMPLES} >= 0, <= Tau_U;

  minimize obj: + Epsilon * sum{i in SAMPLES} z[i];
}

// Linking constraints form block row border
subject to {j in FEATURES}:
  weights.w[j] =
    sum{f in CLASS_FILES, i in SAMPLES[f]}
      class_file[f].x[i,j]*class_file[f].y[i]*class_file[f].z[i]
  + sum{f in UNIVER_FILES, i in SAMPLES[f]}
      univer_plus_file[f].x[i,j]*univer_plus_file[f].z[i]
  - sum{f in UNIVER_FILES, i in SAMPLES[f]}
      univer_minus_file[f].x[i,j]*univer_minus_file[f].z[i]

subject to:
  sum{ f in CLASS_FILES, i in SAMPLES[f] } class_file[f].y[i] * class_file[f].z[i]
  + sum{ f in UNIVER_FILES, i in SAMPLES[f] } univer_plus_file[f].z[i]
  - sum{ f in UNIVER_FILES, i in SAMPLES[f] } univer_minus_file[f].z[i]
  = 0;
```

Figure B.2: SML model for Universum classification, based on (5.5).

```
set CLASSES;
param M > 0;
param Tau > 0;
set FEATURES = {1..M};
var beta{CLASSES} >= 0;

block weights {
  var w{FEATURES};
  minimize obj: 0.5 * sum{j in FEATURES} w[j]*w[j];
}

// We assume the samples for each class are in individual files
block class {f in CLASSES} {
  param N_f > 0; // Number of samples in this file
  set SAMPLES = {1..N_f};
  param x{SAMPLES,FEATURES};
  var zp{SAMPLES} >= 0, <= Tau;
  var zn{SAMPLES} >= 0, <= Tau;

  minimize obj: - sum{i in SAMPLES} (zp[i]+zn[i]);
}

// Linking constraints form block row border
subject to {j in FEATURES}:
  weights.w[j] =
  sum{f in CLASSES, i in SAMPLES[f]}
    class[f].x[i,j]*(class_file[f].zp[i]-class_file[f].zn[i]);

subject to {f in CLASSES}:
  sum{ i in SAMPLES[f] } class[f].z[i] + beta[f]
  = sum{ i in SAMPLES[next(f)] } class[next(f)].z[i] + beta[next(f)];
```

Figure B.3: SML model for explicit SVM ordinal regression (5.6).

```
set FILES;
param M > 0;
param Tau > 0, Epsilon > 0;
set FEATURES = {1..M};

block file {f in FILES} {
  param N_f > 0; // Number of samples in this file
  set SAMPLES = {1..N_f};
  param x{SAMPLES,FEATURES};
  param y{SAMPLES};
  var z{SAMPLES} >= 0, <= Tau;
  var zhat{SAMPLES} >= 0, <= Tau;
  var zbar{SAMPLES};

  subject to {i in SAMPLES}:
    -z[i] + zhat[i] + zbar[i] = 0;

  minimize obj: Epsilon * sum{i in SAMPLES} (z[i]+zhat[i]) -
    sum{i in SAMPLES} (y[i]*zbar[i]);
}

block weights {
  var w{FEATURES};
  minimize obj: 0.5 * sum{j in FEATURES} w[j]*w[j];
}

// Linking constraints form block row border
subject to {j in FEATURES}:
  weights.w[j] = sum{ f in FILES, i in SAMPLES[f] }
                 file[f].x[i,j]*file[f].zbar[i];

subject to:
  sum{ f in FILES, i in SAMPLES[f] } file[f].zbar[i] = 0;
```

Figure B.4: SML model for metric regression (5.8).

Appendix C

Notation

In general we follow IPM conventions, with scalars and column vectors denoted using lower case letters, and upper case letters denoting matrices. Unless otherwise defined, they are the diagonal matrices of the corresponding lower case vectors. We have modified the normal SVM notation where it does not follow this convention.

b	constraint constants
c	linear objective terms
e	vector of all 1s
l	lower bounds
m	number of attributes
n	number of samples
r	rank of approximation
r_b	residual associated with primal constraints
r_c	residual associated with dual constraints
s	dual variables associated with lower bound of z
u	upper bounds
v	dual variables associated with upper bound of z
w	weight variables defining normal to hyperplane
w_0	bias of hyperplane, b in SVM literature
x_i	attribute vector for the i^{th} data point, consisting of the observation values directly
y	target values, for binary classification $y_i \in \{-1, 1\}$
z	sample Lagrange multipliers in SVM dual formulation (3.6), α in SVM literature
λ, λ_0	Lagrange multipliers
ξ	misclassification errors
τ	penalty for misclassifications, C in SVM literature
A	Jacobian constraint matrix of QP
D	diagonal matrix forming Cholesky decomposition
I	identity matrix
K	kernel matrix
L	lower triangular matrix forming Cholesky decomposition
Q	Hessian matrix of QP
X	$m \times n$ matrix, each column is the vector x_i
Y	$Y = \text{diag}(y)$
Θ	scaling matrix associated with logarithmic barriers
$\mathscr{B}$	set of basic variables
$\mathscr{L}$	Lagrangian function
$\mathscr{N}$	set of nonbasic variables
$\mathscr{U}$	set of upper-bounded variables
$\mathscr{W}$	set of working set variables