PRACTICAL IMPLEMENTATION OF THE ACTIVE SET METHOD FOR SUPPORT
VECTOR MACHINE TRAINING WITH SEMI-DEFINITE KERNELS

by

CHRISTOPHER GARY SENTELLE
B.S. of Electrical Engineering University of Nebraska-Lincoln, 1993
M.S. of Electrical Engineering University of Nebraska-Lincoln, 1995

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2014

Major Professor: Michael Georgiopoulos

ABSTRACT

The Support Vector Machine (SVM) is a popular binary classification model due to its superior generalization performance, relative ease-of-use, and applicability of kernel methods. SVM training entails solving an associated quadratic programming (QP) that presents significant challenges in terms of speed and memory constraints for very large datasets; therefore, research on numerical optimization techniques tailored to SVM training is vast. Slow training times are especially of concern when one considers that re-training is often necessary at several values of the models regularization parameter, C, as well as associated kernel parameters.

The active set method is suitable for solving SVM problem and is in general ideal when the Hessian is dense and the solution is sparse–the case for the ℓ_1-loss SVM formulation. There has recently been renewed interest in the active set method as a technique for exploring the entire SVM regularization path, which has been shown to solve the SVM solution at all points along the regularization path (all values of C) in not much more time than it takes, on average, to perform training at a single value of C with traditional methods. Unfortunately, the majority of active set implementations used for SVM training require positive definite kernels, and those implementations that do allow semi-definite kernels tend to be complex and can exhibit instability and, worse, lack of convergence. This severely limits applicability since it precludes the use of the linear kernel, can be an issue when duplicate data points exist, and doesn't allow use of low-rank kernel approximations to improve tractability for large datasets.

The difficulty, in the case of a semi-definite kernel, arises when a particular active set results in a singular KKT matrix (or the equality-constrained problem formed using the active set is semi-definite). Typically this is handled by explicitly detecting the rank of the KKT matrix. Unfortunately, this adds significant complexity to the implementation; and, if care is not taken, numerical

instability, or worse, failure to converge can result. This research shows that the singular KKT system can be avoided altogether with simple modifications to the active set method. The result is a practical, easy to implement active set method that does not need to explicitly detect the rank of the KKT matrix nor modify factorization or solution methods based upon the rank.

Methods are given for both conventional SVM training as well as for computing the regularization path that are simple and numerically stable. First, an efficient revised simplex method is efficiently implemented for SVM training (SVM-RSQP) with semi-definite kernels and shown to out-perform competing active set implementations for SVM training in terms of training time as well as shown to perform on-par with state-of-the-art SVM training algorithms such as SMO and SVMLight. Next, a new regularization path-following algorithm for semi-definite kernels (Simple SVMPath) is shown to be orders of magnitude faster, more accurate, and significantly less complex than competing methods and does not require the use of external solvers. Theoretical analysis reveals new insights into the nature of the path-following algorithms. Finally, a method is given for computing the approximate regularization path and approximate kernel path using the warm-start capability of the proposed revised simplex method (SVM-RSQP) and shown to provide significant, orders of magnitude, speed-ups relative to the traditional grid search where re-training is performed at each parameter value. Surprisingly, it also shown that even when the solution for the entire path is not desired, computing the approximate path can be seen as a speed-up mechanism for obtaining the solution at a single value. New insights are given concerning the limiting behaviors of the regularization and kernel path as well as the use of low-rank kernel approximations.

To my loving wife who patiently endured the sacrifices and provided encouragement when I lost my way, my beautiful son who was born during this endeavour, and my parents who gave me encouragement when I needed it.

ACKNOWLEDGMENTS

I would like to thank my committee and my advisors Dr. Michael Georgiopoulos and Dr. Georgios Anagnostopoulos. I am grateful for their guidance, wisdom, insight, and friendship, which I will carry with me far beyond this endeavor.

I would like to thank my loving wife for her unending patience and support during this effort and for keeping me grounded all those times I wanted to give up. I couldn't have done this on my own. I would also like to thank my son who in later years may remember those times I was busy working and couldn't play.

I would also like to thank all of the Machine Learning Lab members who have been instrumental in my success in various ways and, in some cases, having to endure reading or critiquing some of my work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Vapnik and his colleagues at AT&T Bell Laboratories [93] introduced the Support Vector Machine (SVM), and it has become an increasingly popular machine learning technique-nearly becoming ubiquitous within various fields of the academic community as well as within industry. Some example applications of SVM include face recognition [68], medical applications such as x-ray image processing [23] and heart rate classification [97], radar target recognition applications [51], financial forecasting applications [98], [90], [88], and bioinformatics [9]. An extensive survey of SVM applications can be found in [8], and excellent tutorials on the subject can be found in [7] and [15].

The SVM is popular in large part due to its excellent generalization performance or its ability to accurately classify previously unseen examples or patterns. This is due to its strong theoretical foundation based upon statistical learning theory [93]. Additionally, the SVM has an interesting and easy to understand geometrical interpretation. While the SVM classifier was originally posed as a linear classifier, kernel methods can be applied-dramatically enriching its application to a wide range of non-linear problems. In addition, to supervised, binary classification, the SVM framework has been applied to semi-supervised training [14], outlier detection as a one-class SVM [78], and regression [93], [96], to name only a few.

SVM training is performed by solving an associated QP (Quadratic Programming) problem. This QP problem can be solved with conventional numerical optimization techniques. However, the dual QP problem possesses a dense Hessian, which grows in size quadratically with dataset size. Therefore, the SVM problem can be quickly become intractable for traditional off-the-shelf optimization routines due to both memory constraints and convergence rates. Even recently, with state-of-the-art SVM-tailored algorithms, slow training times associated with the SVM often forces

1

the practitioner to choose an alternative machine learning technology [46], [11]. Furthermore, the practitioner must adjust kernel parameters as well as a regularization parameter in order to achieve optimal performance, which often requires re-training. For example, in a comparison with Multi-Objective Genetic Adaptive Resonant Theory (MO-GART) [45], the authors found the SVM was orders of magnitude slower when performing a grid search to find the optimal parameters. As a result, research on numerical optimization techniques tailored to the SVM problem is vast and on-going.

The active set method is naturally suited to solving the numerical optimization problem associated with SVM training, for which, a significant amount of research has been dedicated, see, for example, [77], [84], [38], [67], [95], [10], [1], [96], [65], and [61]. The active set method is shown to be a strong candidate when the Hessian is dense [77] and the solution is sparse [95], which is often the case especially for the dual SVM formulation when applying kernel methods. The popular decomposition-based Sequential Minimal Optimization (SMO) [69] and SVMLight [41] algorithms can, in fact, be seen to belong to the class of active set methods. However, the decomposition approaches work by optimizing a fixed subset of the variables at each iteration (SMO only optimizes two variables at a time), which represents a form of coordinate descent. Therefore, while alleviating memory consumptions issues associated with large optimization problems, the methods can suffer from slow convergence. Relative to the decomposition approaches, the traditional active set method has the following distinct advantages (1) the ability to perform incremental/decremental training and support for efficient leave-one-out (LOO) estimation [10], (2) improved accuracy [96], [77], and (3) improved stability and convergence properties over a wider range of the regularization parameter, C, [95], [1], [96]. Efficiently implemented active set methods have been shown to be competitive, overall, with decomposition approaches [77], [79] and are expected to outperform decomposition methods when the fraction of support vectors is small [95], [96]. More recently, the active set method has become prominent for the important class of regularization path-following

methods [38].

Unfortunately, the conventional active set method, without modification, requires a positive definite Hessian (or kernel function) when working with the dual SVM formulation. Otherwise, an active set may be encountered where the corresponding equality-constrained problem is singular. This limits applicability of the method to a wide range of problems including cases where where a linear kernel is desired. Further, duplicate data points can create a semi-definite Hessian even when the kernel function is positive definite, and near-singularity may occur due to numerical precision. Most importantly, this prevents the use of low-rank kernel approximations [85], [27] for speeding up the active set method. The majority of active set implementations reported in the SVM literature are limited to the application of positive definite kernels.

The active set method can be successfully modified to address semi-definite Hessians. In this case, the active set method detects when a singular system occurs and computes a descent direction that will adjust the active set so that the next system of equations is non-singular. However, care must be taken to ensure numerical stability and to prevent cycling in these cases. This is successfully done, for example, by Scheinberg *et al.* within the SVM-QP algorithm [77]. For the regularization path following algorithm, Ong *et al.* [67] solve a modified linear programming problem that accounts for the null space when a singular matrix is detected. In general, these methods either require a modified factorization such as such as QR, SVD, and modified Cholesky and/or require a selective solution method based upon the number of singular eigenvalues. Therefore, these techniques add significant complexity, provide additional sources of numerical instability, and may ultimately prevent convergence. Further, more recent and important methods for computing the regularization path referred to as approximate or sub-optimal regularization path-following algorithms [30], [44] do not yet address the semi-definite kernel at all.

The purpose of this research is to address the issues related to use of semi-definite kernels with the

active set method. This research largely shows that a change can be made to the pivoting strategy of the active set method that automatically avoids the singular system of equations. This results in a method that maintains the simplicity of the active set implementation for positive definite kernels while providing superior performance and numerical stability compared to existing methods in the face of semi-definite kernels. Having resolved issues associated with the semi-definite kernel, this research introduces a novel initialization method for the regularization path following algorithm, introduces new approximate regularization path and kernel path algorithms for semi-definite kernels, and investigates, for the first time, the use of low-rank kernel approximations within the active set method. Specifically, the contributions of this research are as follows:

- Introduce an efficiently implemented revised simplex method for quadratic programming for SVM training with semi-definite kernels that outperforms existing implementations in terms of training time and is competitive with decomposition methods such as SMO and SVMLight.

- Introduce a regularization path following algorithm for semi-definite kernels that maintains the simplicity of the original method for positive definite kernels while significantly outperforming the competing method in both training time and numerical stability. An extensive theoretical analysis is provided along with a proof of convergence. New insight is also provided into the potential failure of existing methods to accurately compute the regularization path.

- Introduce a novel initialization method for the regularization path following algorithm for the case of unequal class sizes that works within the framework of the existing algorithm avoiding the need for an external solver. This initialization method is only suitable for algorithms that can handle semi-definite kernels. New insights into the initialization problem are discussed.

- Introduce a method for performing an approximate path search as well as a fast grid search from within the SVM-RSQP algorithm that is competitive with the regularization path following algorithms in terms of speed while also handling semi-definite kernels. This also represents the first known approximate path method that demonstrates success for the ℓ_1-loss SVM with semi-definite kernels. New insight is provided into the practical benefits of the approximate path search versus a traditional logarithmic grid search.

- Introduce a novel method for performing approximate kernel path parameter searches within the SVM-RSQP algorithm, that is, again, able to handle semi-definite cases. This represents the first known *approximate* kernel path method.

- The first known results are reported on using low-rank kernel approximations with the active set method. This represents an important step towards improving tractability of the active set method for very large datasets in terms of both memory consumption as well as training time. Additional insight is given on the effectiveness of the incomplete Cholesky factorization for low-rank kernel approximation from the perspective of the kernel path.

The remainder of this document is organized as follows. Chapter 2 provides an overview of the Support Vector Machine while Chapter 3 provides an overview of the active set method. Chapter 4 discusses the literature review associated with the broad spectrum of training methods applied to solving the SVM problem as well as literature review of the active set method. Chapter 5 discusses the revised simplex method for quadratic programming as a SVM training method for semi-definite kernels. The details of implementation are provided along with comparisons to other state-of-the-art implementations. Chapter 6 discusses a new technique for solving the regularization path for semi-definite kernels. In this chapter, a detailed review of the regularization path algorithm is given along with detailed analysis and discussion. New insight is given into the nature of the initialization problem as well as a new initialization method is introduced for unequal class sizes. Chapter 7

discusses efficient solution of the approximate regularization and kernel paths within the SVM-RSQP framework. This chapter also discusses performance of low-rank kernel approximation within the SVM-RSQP algorithm. Conclusions and summary are provided in Chapter 8.

Notation

Within this research, vectors are denoted with bold-face, lower case, scalars with lower case and matrices with upper case. Subscripts denote a component of a vector such as $\mathbf{x_i}$ and a set of subscripts with lower case denotes a single element of a matrix. For example, q_{ij} is the element corresponding to row i and column j of Q. A single index in lower case denotes a single column from a matrix. For example, $\mathbf{q_i}$ is the ith column of the matrix Q. The notation $|\mathcal{S}|$ refers to the cardinality of set $\mathcal{S}$. The matrix, Q, is referenced a number of times in this paper and is defined as $Q_{ij} = y_i y_j K(\mathbf{x_i}, \mathbf{x_j})$, where $y \in \{-1, +1\}$ and $K(\cdot, \cdot)$ is a positive kernel associated with a, potentially infinite-dimensional, reproducing kernel Hilbert space. The notation $\|\mathbf{w}\|_{\mathcal{H}}^2 \triangleq \langle \mathbf{w}, \mathbf{w} \rangle_{\mathcal{H}}$ corresponds to a dot product formed in the Hilbert space $\mathcal{H}$.

CHAPTER 2: SUPPORT VECTOR MACHINE

Overview

In its basic form, the goal of the binary SVM classifier is to find a separating hyperplane, given a set of data patterns $\mathbf{x_i} \in \mathbb{R}^d$ and corresponding binary labels $y_i \in \{+1, -1\}$, that correctly partitions the labeled data points while maximizing the margin between the hyperplane and the closest data point $\mathbf{x_i}$ to the hyperplane. The hyperplane, as depicted in Figure 2.1, can be described by the equation $\langle \mathbf{w}, \mathbf{x_i} \rangle + b = 0$. The vector $\mathbf{w}$ is normal to the hyperplane and the distance from the origin to the hyperplane is $\frac{b}{\|\mathbf{w}\|_2}$. The predicted label then becomes $\text{sgn}\left(\langle \mathbf{w}, \mathbf{x_i} \rangle + b\right)$ where $\text{sgn}\left(\cdot\right)$ is the signum function and is +1 or -1 for positive, negative quantities, respectively.

A margin is added requiring a minimum distance between each data point and the hyperplane as follows

$$\langle \mathbf{w}, \mathbf{x_i} \rangle - b \geq 1, y_i = +1 \tag{2.1}$$

$$\langle \mathbf{w}, \mathbf{x_i} \rangle - b \leq -1, y_i = -1 \tag{2.2}$$

or, more concisely,

$$y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle - b\right) \geq 1 \tag{2.3}$$

The *functional* distance between any pattern and the hyperplane is measured as $\|\langle \mathbf{w}, \mathbf{x_i} \rangle - b\|$ and is equal to 1 for any pattern lying on the margin.
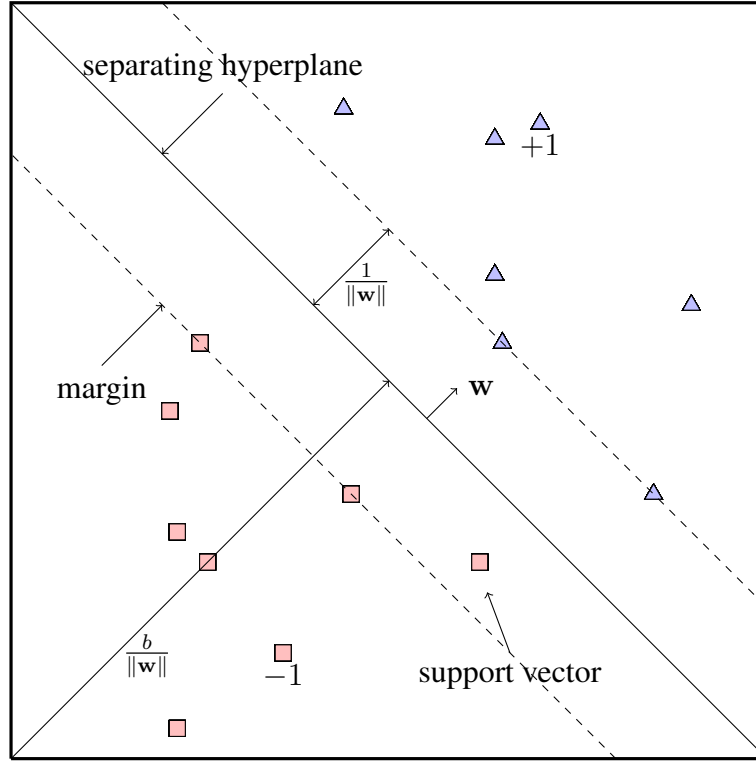
Figure 2.1: The support vector machine finds a separating hyperplane between two classes of data maximizing the margin between the hyperplane and closest data point. In the soft-margin formulation, a few data points are allowed to cross the margin.

The *geometric* distance between the hyperplane and any pattern $\mathbf{x}$ can be found by forming the projection of the pattern onto the unit normal vector to the hyperplane as follows $\langle \frac{\mathbf{w}}{\|\mathbf{w}\|}, \mathbf{x} \rangle - \frac{b}{\|\mathbf{w}\|}$. For points on the positive-side margin, $\langle \mathbf{w}, \mathbf{x} \rangle - b = 1$, the *geometric* distance becomes

$$\frac{1+b}{\|\mathbf{w}\|} - \frac{b}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}, \tag{2.4}$$

This is the *geometric* width and is commonly referred to as the margin width. The total *geometric* margin width, from the $+1$ to the -1 functional margin, is then $\frac{2}{\|\mathbf{w}\|}$.

The task of finding a separating hyperplane with maximal margin width can be posed as an optimization problem as follows.

$$\min_{\mathbf{w}} \quad \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.5}$$

$$\text{s.t.} \quad y_i\left(\langle\mathbf{w},\mathbf{x_i}\rangle + b\right) \geq 1 \quad \forall i \tag{2.6}$$

Squaring $\|\mathbf{w}\|$ as well as multiplying by the constant coefficient $\frac{1}{2}$ is done for convenience, i.e. consider that $\frac{\partial}{\partial\mathbf{w}}\frac{1}{2}\|\mathbf{w}\|^2 = \mathbf{w}$.

This is referred to as the hard-margin formulation of the SVM. This formulation is impractical due to the underlying assumption that data patterns are linearly separable or a hyperplane perfectly separating positively- and negatively-labeled data points can be found. In fact, a feasible solution (solution satisfying all constraints) does not exist in the instance where the data is not linearly separable. This is remedied with the soft-margin formulation, which allows some data points to violate the constraints.

$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \tag{2.7}$$

$$\text{s.t.} \quad y_i\left(\langle\mathbf{w},\mathbf{x_i}\rangle + b\right) - 1 + \xi_i \geq 0 \quad \forall i \tag{2.8}$$

$$\xi_i \geq 0 \quad \forall i \tag{2.9}$$

A set of non-negative slack variables, ξ_i, specify the degree of violation of the constraints for each data point. The additional term in the objective is referred to as a regularization term and penalizes the use of slack variables. Without the penalty term, the addition of slack variables would result in a solution consisting of an infinite margin width with all points inside the margin. This formulation adds a user-specified parameter, C, that adjusts the trade between margin width and margin violation.

The previous formulation is also referred to as the ℓ_1-loss SVM. An alternative is the ℓ_2-loss SVM formulation [50] as follows

$$\min_{\mathbf{w},\xi} \quad \|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i^2 \tag{2.10}$$

$$\text{s.t.} \quad y_i \left(\langle \mathbf{w}, \mathbf{x_i} \rangle + b \right) - 1 + \xi_i \geq 0 \tag{2.11}$$

Note that the non-negativity constraints associated with ξ_i are no longer necessary and that the regularization term within the objective function essentially consists of the square of the ℓ_2- norm of $\boldsymbol{\xi}$, i.e. $\|\boldsymbol{\xi}\|_2^2$. This formulation has the advantage that the dual objective function, discussed later, becomes positive definite, which simplifies optimization. However, the solution in terms of the slack variables, ξ_i, will be less sparse (fewer zero values). Sparseness is normally desired during the testing phase [46]. Further, Koshiba *et. al* [50] reported that training times tended to be faster for the ℓ_1 problem while accuracy was not substantially different between the two formulations.

Instead of working with the primal formulation, the majority of SVM implementations work with dual formulation, which can be derived as follows. Consider the following standard form optimization problem

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{2.12}$$

$$\text{s. t. } g_i(\mathbf{x}) \leq 0 \tag{2.13}$$

$$h_j(\mathbf{x}) = 0 \tag{2.14}$$

where $i \in \{1, ..., m\}$ for a set of m inequality constraints and $j \in \{1, ..., n\}$ for a set of n equality constraints. The Lagrangian of this problem is formed as follows.

$$L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\nu}) = f(\mathbf{x}) + \sum_{i=1}^{m} \mu_i g_i(\mathbf{x}) + \sum_{i=1}^{n} \nu_i h_i(\mathbf{x}) \tag{2.15}$$

10

The variables μ_i and ν_i are referred to as Lagrange multipliers or dual variables (sometimes referred to as KKT multipliers as well). The dual function is formally defined as (see also [6])

$$g\left(\boldsymbol{\mu}, \boldsymbol{\nu}\right) = \inf_{\mathbf{x}} L\left(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\nu}\right) \tag{2.16}$$

where $\inf$ is the infimum or the greatest lower bound and is simply the minimum if the greatest lower bound is in the domain of $g\left(\right)$. The dual optimization problem is

$$\max g\left(\boldsymbol{\mu}, \boldsymbol{\nu}\right) \tag{2.17}$$

$$\text{s. t. } \mu_i \geq 0 \ \forall \ i \tag{2.18}$$

The Lagrangian for the ℓ_1-loss SVM soft-margin formulation is

$$L\left(\mathbf{w}, b, \alpha\right) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i \left(\langle \mathbf{x_i}, \mathbf{w}\rangle + b\right) - 1 + \xi_i\right) + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} r_i \xi_i \tag{2.19}$$

The stationary condition is found by finding the derivative with respect to each of the primal variables as follows,

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x_i} = 0 \tag{2.20}$$

$$\frac{\partial L}{\partial \mathbf{b}} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.21}$$

$$\frac{\partial L}{\partial \xi_\mathbf{i}} = C - \alpha_i - r_i = 0 \tag{2.22}$$

The variables α_i (or Lagrange multipliers) are the dual variables. Upon substituting the first two

expressions back into the Lagrangian, the dual function becomes

$$-\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_{i=1}^{n}\alpha_i \tag{2.23}$$

which results in the following dual problem

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i=1}^{n}\alpha_i \tag{2.24}$$

$$\text{s. t.} \qquad \sum_{i=1}^{n} y_i\alpha_i = 0 \tag{2.25}$$

$$0 \le \alpha_i \le C \ \ \forall i \tag{2.26}$$

The last inequality results from $r_i \ge 0$, $\alpha_i \ge 0$ and $C = r_i + \alpha_i$. Therefore, the regularization parameter, C, now becomes an upper bound on the dual variables α_i. Of interest also is the fact the quantities $\mathbf{x}_i$ only appear within the dot product, which allows application of the "kernel trick" for classifying non-linear problems, discussed later.

The dual formulation has an interesting physical interpretation as shown in Figure 2.2. Note that there is a dual variable α_i for each of the inequality constraints in the primal problem and effectively for each of the data points. At the solution, $\alpha_i = 0$ for all data points that are outside the margin, $0 \le \alpha_i \le C$ for all points on the margin and $\alpha_i = C$ for all points inside the margin. The data points with $\alpha_i > 0$ are referred to as support vectors. The set of support vectors are often further partitioned into the set of non-bound support vectors, $\alpha_i < C$, and bound support vectors, $\alpha_i = C$. Since the dual variables are the Lagrange multipliers for the inequality constraints within the primal problem, $\alpha_i = 0$ implies that the solution is not sensitive to that particular constraint.
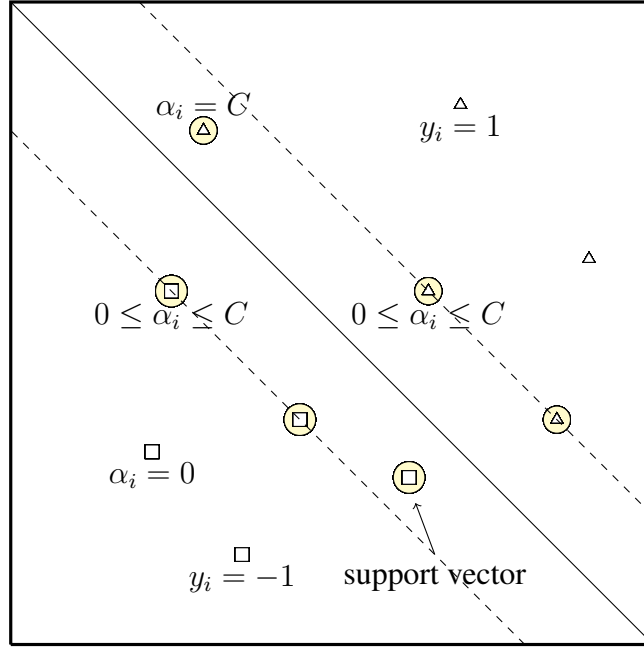
Figure 2.2: Physical interpretation of the SVM dual soft-margin formulation. Yellow circles denote the support vectors, which are the data points within or on the margin.

As a result, an interesting characteristic of the support vector machine is that these constraints, and corresponding data points, can be removed from the problem without affecting the solution.

Kernel Methods

A non-linear mapping, possibly to a higher dimensional space, can enable algorithms restricted to linearly separable problems to be applied to the non-linear problem [15]. The SVM dual formulation is advantageous, as will be shown, since the data vectors appear strictly within a dot-product term. Consider a non-linear mapping $\mathbf{z}_i = \phi(\mathbf{x}_i)$ where $\phi : \mathcal{X} \to \mathcal{H}$ represents a non-linear mapping to a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$.
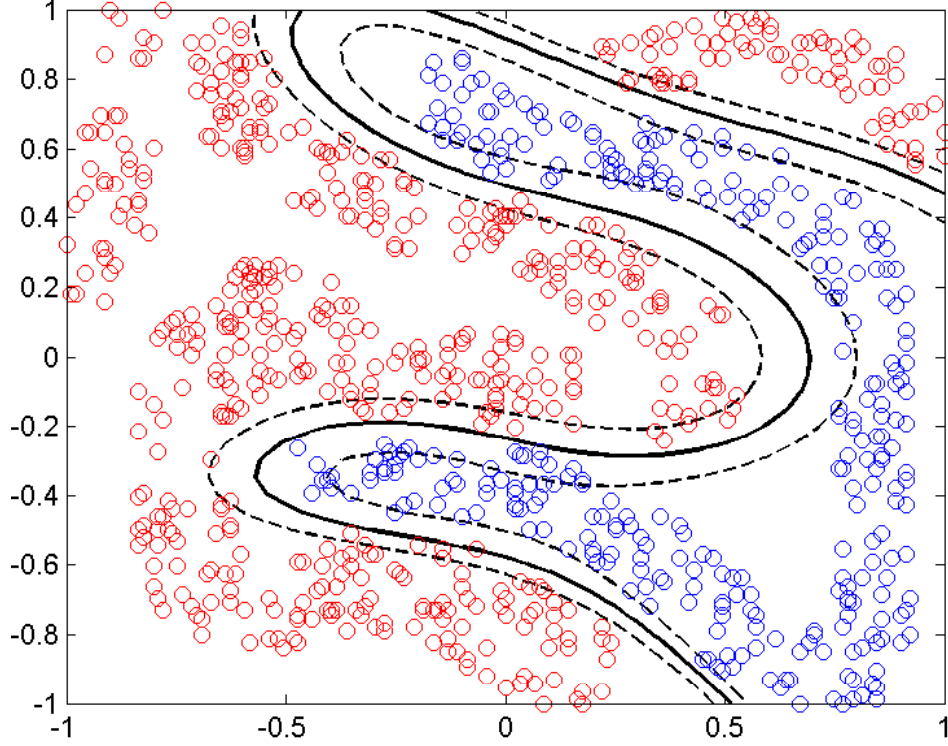
Figure 2.3: Depiction of non-linear margin created with kernel methods. By applying kernel methods the SVM can generate non-linear separating surfaces.

The dual formulation can be be rewritten as

$$\min_{\alpha} \quad \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \phi\left(\mathbf{x}_i\right), \phi\left(\mathbf{x}_j\right) \rangle_{\mathcal{H}} - \sum_{i=1}^{n} \alpha_i \tag{2.27}$$

$$\text{s. t.} \qquad \sum_{i=1}^{n} y_i \alpha_i = 0 \tag{2.28}$$

$$0 \leq \alpha_i \leq C \ \ \forall i \tag{2.29}$$

where the notation $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is used to denote the dot product performed in the RKHS, $\mathcal{H}$. In general, computing the non-linear mapping $\phi()$ might be intractable, for example, when the mapping function is to an infinite-dimensional space. Instead of explicitly computing the mapping, the

dot-product term can be replaced with a kernel function $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$ in what is often referred to as a "kernel trick". The kernel function performs a dot-product in the non-linearly mapped space–combining the mapping and dot-product steps. Using the "kernel trick", the dual formulation now becomes

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) - \sum_{i=1}^{n}\alpha_i \tag{2.30}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} y_i\alpha_i = 0 \tag{2.31}$$

$$0 \leq \alpha_i \leq C \ \ \forall i \tag{2.32}$$

Some of the most commonly used kernels are defined in Table 2.1. In addition to these, any symmetric, positive semi-definite kernel function can be used [15] [Proposition 3.5]. Mercer's theorem provides the necessary and sufficient conditions for the kernel to behave as an inner product within the non-linearly mapped space. Kernels that satisfy these conditions are referred to as Mercer kernels. Kernels can be derived to allow similarity comparisons between patterns derived from a wide variety of both numerical, categorical features as well as even unconventional features. For example, Leslie *et al.* [55] develop a string-kernel for protein classification.

Table 2.1: Commonly Used Kernel Functions for SVM Training

Linear	$\mathbf{x}^T\mathbf{y}$
Polynomial	$\left(\mathbf{x}^T\mathbf{y} + 1\right)^d$
Gaussian	$\exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|_2^2}{2\sigma^2}\right)$
Sigmoid	$\tanh\left(\kappa\mathbf{x}^T\mathbf{y} - \delta\right)$

15

Applying Kernel Methods with the Primal Formulation

Kernel methods can also be applied to the primal formulation of the SVM such as in [13], [43], and [83] using the *representer theorem* (see [13], [15]). Here, the SVM problem is formulated as

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_{\mathcal{H}}^2 + C\sum_{i=1}^{n}\ell(\mathbf{w},b;\mathbf{x}_i,y_i) \tag{2.33}$$

where, for the soft-margin formulation, the hinge-loss function is employed

$$\ell(\mathbf{w},b;\mathbf{x},y) \triangleq \left[1 - y_i\left(\langle\mathbf{w},\phi(\mathbf{x}_i)\rangle_{\mathcal{H}} + b\right)\right]_+ \tag{2.34}$$

In general, $\ell(\mathbf{w},b;\mathbf{x}_i,y_i)$ can be any loss function including the quadratic loss of the ℓ_2-loss SVM formulation [50]. In this formulation, $\mathbf{w}$ and $\phi(\mathbf{x}_i)$ both belong to the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$. The kernel function $k(\cdot,\mathbf{x})$ is defined such that $k(\cdot,\mathbf{x}) = \phi(\mathbf{x})$ and, therefore,

$$\langle k(\cdot,\mathbf{x}_i), k(\cdot,\mathbf{x}_j)\rangle_{\mathcal{H}} = k(\mathbf{x}_i,\mathbf{x}_j). \tag{2.35}$$

The representer theorem suggests that the optimal solution $\mathbf{w}^*$ can be written in terms of kernel functions evaluated at the data points in the problem as follows

$$\mathbf{w}^* = \sum_{i=1}^{n}\beta_i k(\cdot,\mathbf{x_i}) \tag{2.36}$$

Therefore, the representer theorem allows the problem to be rewritten as

$$\min_{\boldsymbol{\beta},b} \frac{1}{2}\boldsymbol{\beta}^T K\boldsymbol{\beta} + C\sum_{i=1}^{n}\left[1 - y_i\left(\sum_{j=1}^{n}\beta_j k(\mathbf{x}_i,\mathbf{x}_j) + b\right)\right]_+ \tag{2.37}$$

The following equivalent formulation is derived by introducing a set of slack variables ξ_i,

$$\min_{\boldsymbol{\beta}, b, \boldsymbol{\xi}} \quad C\mathbf{1}^T\boldsymbol{\xi} + \tfrac{1}{2}\boldsymbol{\beta}^T K \boldsymbol{\beta} \tag{2.38}$$

$$\text{s.t.} \quad \xi_i \geq 0, \ \ \xi_i \geq 1 - y_i\left(\mathbf{k}_i^T\boldsymbol{\beta} + b\right) \ \ \forall i \tag{2.39}$$

Note that the variables β_i are not the Lagrange multipliers, α_i, associated with the dual SVM soft-margin formulation presented earlier. The dual formulation is again derived by forming the Lagrangian

$$\mathcal{L}(\boldsymbol{\beta}, b, \boldsymbol{\xi}) = [C\mathbf{1} - \mathbf{r} - \boldsymbol{\alpha}]^T\boldsymbol{\xi} + \frac{1}{2}\boldsymbol{\beta}^T K \boldsymbol{\beta} + \boldsymbol{\alpha}^T\left[\mathbf{1} - Y\left(K\boldsymbol{\beta} + b\mathbf{1}\right)\right] \tag{2.40}$$

where Y is a diagonal matrix containing the labels, y_i along the diagonal and $\mathbf{1}$ is a vector of ones of size n. Taking the partial derivatives with respect to the primal variables yields

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}} = C\mathbf{1} - \mathbf{r} - \boldsymbol{\alpha} = 0 \tag{2.41}$$

which combined with the fact that $\boldsymbol{\alpha}_i \geq 0$ and $\mathbf{r}_i \geq 0$ for all i implies $0 \leq \alpha_i \leq C$. The partial derivative with respect to b yields

$$\frac{\partial\mathcal{L}}{\partial b} = \boldsymbol{\alpha}^T Y \mathbf{1} = \boldsymbol{\alpha}^T \mathbf{y} = 0 \tag{2.42}$$

and, finally, taking the partial derivative with respect to $\boldsymbol{\beta}$ gives

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\beta}} = K\boldsymbol{\beta} - KY\boldsymbol{\alpha} = 0 \tag{2.43}$$

where a solution is $\boldsymbol{\beta} = Y\boldsymbol{\alpha}$ (unique if K is positive definite). Substituting this expression along with (2.41) into (2.40) yields the dual objective

$$\mathcal{L}(\boldsymbol{\alpha}) = \mathbf{1}^T \boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \tag{2.44}$$

along with the constraints $\boldsymbol{\alpha}^T \mathbf{y} = 0$ and $0 \leq \alpha_i \leq C$. Note that $Q \triangleq YKY$.

<center>Generalization Performance and Structural Risk Minimization</center>

Many classifiers work to minimize empirical risk defined as

$$R(f) = \frac{1}{2n} \sum_{i=1}^{n} L(y_i, f(\mathbf{x_i})) \tag{2.45}$$

where $f(\mathbf{x})$ is the discriminant or scoring function and incorporates classifier knowledge, $\mathbf{x} \in \mathbb{R}^d$ is a training pattern with dimensionality d, $y_i \in \{+1, -1\}$ is the corresponding label, and there are n data points. The function $L(\cdot, \cdot)$ is a loss function that measures the similarity between the desired labels y_i and output of the classifier. An example loss function often used for classification is the $0-1$ loss function, which returns 1 if $y \neq f(\mathbf{x})$. The empirical risk is minimized by reducing the number of labeling errors the classifier makes. The empirical risk is only an estimate of the true risk

$$R(f) = \frac{1}{2} \int L(y, f(\mathbf{x})) dP(\mathbf{x}, y) \tag{2.46}$$

True risk is derived based upon complete knowledge of the probability distribution $P(\mathbf{x}, y)$, which can only be *estimated* from the given training data. In general, it is expected that minimization of the empirical risk will result in over-training, unless the training samples are of sufficient size to accurately reflect the true underlying statistics.

The SVM, instead of empirical risk minimization, performs structural risk minimization. Structural risk minimization attempts to balance empirical risk and capacity. Capacity is loosely related to the ability of the classifier to fit arbitrarily complex surfaces through the data. A classifier with infinite capacity, in theory, is capable of driving the empirical risk to zero.

The notion of capacity is formalized with the concept of VC (Vapnik Chervonenkis) dimension. The VC dimension is a measure of the maximum number of data points that can be accurately classified given all possible combinations of assigned labels. This can be used to provide an upper bound on the true risk with probability less than $1 - \eta$ as follows

$$R(f) \leq R_{emp}(f) + \sqrt{\frac{h\log\left(\frac{2n}{h}\right) - \log\left(\frac{\eta}{4}\right)}{n}} \tag{2.47}$$

where h is the (VC) dimension, and n is the number of training data points. As expected, this indicates that the upper bound on the true risk is increased with increased capacity.

The goal of structural risk minimization is to minimize this upper bound as opposed to minimizing the empirical risk. Vapnik [93] shows us that maximizing the margin within the SVM framework is equivalent to structural risk minimization. This, then, is the genesis of the generalization performance observed for the support vector machine. For an excellent tutorial on this subject see Burges [7].

<div align="center">Alternative SVM Derivation using Rademacher Complexity</div>

The SVM primal formulation also naturally arises from a Rademacher complexity generalization bound perspective. Specifically, the SVM primal problem can be shown to be exactly equivalent to minimizing the upper bound on the true risk. This is based upon the following theorem from Mohri *et al.* [63]

Theorem 1 (Theorem 4.5, [63]). *For fixed $\rho > 0$, $\forall\ h \in H$, and $\forall\ \delta > 0$*

$$R(h) \leq \hat{R}_p(h) + \frac{2}{\rho}\hat{R}_S(h) + 3\sqrt{\frac{ln\frac{2}{\delta}}{2n}} \tag{2.48}$$

where

$$
\begin{aligned}
R(h) &\triangleq E\left\{[yh(\mathbf{x}) \leq 0]\right\} & (2.49)\\
\hat{R}_p(h) &\triangleq \frac{1}{n}\sum_{i=1}^{n}\Phi_p(y_i h(\mathbf{x}_i)) & (2.50)\\
\hat{R}_S(h) &\triangleq E_{\boldsymbol{\sigma}}\left\{\sup_{h\in H}\sum_{i=1}^{n}\sigma_i h(\mathbf{x}_i)\right\} & (2.51)
\end{aligned}
$$

This theorem provides an upper bound on the risk $R(h)$ for a classifier with a discriminant function $h \in H$. The hypothesis space $H \subseteq \mathbb{R}^{\mathcal{X}}$ contains the space of functions implemented by a classifier. The quantity $\hat{R}_p(h)$ is referred to as the empirical margin loss, and the quantity $\hat{R}_S(h)$ is referred to as the empirical Rademacher complexity of H. The random variable σ_i is independently and identically distributed (*i.i.d.*) and drawn from a Rademacher distribution where there is an equal probability of drawing either a $+1$ or -1. The function $\Phi_p(u)$ within the expression for empirical margin loss is defined as follows

$$\Phi_p(u) = \begin{cases} 1, & u \leq 1 \\ 1-p, & 0 \leq u \leq p \\ 0, & u > p \end{cases} \tag{2.52}$$

For the SVM problem, the hypothesis space consists of the following

$$H = \{h : \mathbf{x} \mapsto \langle \mathbf{w}, \phi(\mathbf{x}) \rangle_{\mathcal{H}}, \mathbf{x} \in \mathcal{X}, \mathbf{w}, \phi(\mathbf{x}) \in \mathcal{H} \mid \|\mathbf{w}\| \leq R\} \tag{2.53}$$

where $\mathcal{H}$ is a RKHS with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and reproducing kernel, k, and the feature mapping $\phi : \mathcal{X} \to \mathcal{H}$ is used. The quantity $R > 0$ can be assigned any positive value and controls the "richness" of H. Define the following space of valid vectors $\mathbf{w}$

$$\Omega_{\mathbf{w}} \triangleq \{\mathbf{w} \in \mathcal{H} : \|\mathbf{w}\|_{\mathcal{H}} \leq R\} \tag{2.54}$$

For the SVM hypothesis space, the empirical Rademacher complexity becomes

$$\hat{R}_S(h) = \frac{1}{n} E_\sigma \left\{ \sup_{\mathbf{w} \in \Omega_{\mathbf{w}}} \sum_{i=1}^{n} \sigma_i \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle_{\mathcal{H}} \right\} \tag{2.55}$$

Application of the Cauchy-Schwartz inequality followed by Jensen's inequality yields

$$\hat{R}_S(h) = \frac{1}{n} E_\sigma \left\{ \sup_{\mathbf{w} \in \Omega_{\mathbf{w}}} \left\langle \mathbf{w}, \sum_{i=1}^{n} \sigma_i \phi(\mathbf{x}_i) \right\rangle_{\mathcal{H}} \right\} \tag{2.56}$$

$$\leq \frac{1}{n} E_\sigma \left\{ \sup_{\mathbf{w} \in \Omega_{\mathbf{w}}} \|\mathbf{w}\|_{\mathcal{H}} \left\| \sum_{i=1}^{n} \sigma_i \phi(\mathbf{x}_i) \right\|_{\mathcal{H}} \right\} \tag{2.57}$$

$$= \frac{R}{n} E_\sigma \left\{ \sqrt{\left\| \sum_{i=1}^{n} \sigma_i \phi(\mathbf{x}_i) \right\|_{\mathcal{H}}^2} \right\} \tag{2.58}$$

$$\leq \frac{R}{n} \sqrt{E_\sigma \left\{ \left\| \sum_{i} \sigma_i \phi(\mathbf{x}_i) \right\|_{\mathcal{H}}^2 \right\}} \tag{2.59}$$

where the supremum over $\mathbf{w} \in \Omega_{\mathbf{w}}$ results in $\|\mathbf{w}\|_{\mathcal{H}} = R$. It is straight-forward to show this becomes

$$\hat{R}_S(h) \leq \frac{R}{n} \sqrt{\text{trace}(K)} \tag{2.60}$$

21

where $K_{i,j} \triangleq \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$ and noting that $E[\sigma_i \sigma_j] = 0$ for $i \neq j$ and $E[\sigma_i^2] = 1$. For the empirical margin loss assume that $k(\mathbf{x}, \mathbf{x}') \leq r \ \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$, which implies $\text{trace}(K) \leq rn$. The function $\Phi_p(u)$ is not convex; however, it is possible to bound the expression as follows

$$\Phi_p(u) \leq [p - u]_+ \tag{2.61}$$

Given this, using $p = 1$ results in

$$\hat{R}_1(h) \leq \frac{1}{n} \sum_{i=1}^{n} [1 - y_i h(\mathbf{x}_i)]_+ \tag{2.62}$$

However, the expression on the r.h.s. is simply the hinge-loss or $\hat{R}_1(h) \leq \hat{R}_{hinge}(h)$. Therefore, the upper bound on empirical risk (or 0-1 loss) for the SVM hypothesis space with probability $1 - \delta$ becomes

$$R(h) \leq \hat{R}_{hinge}(h) + 2\sqrt{\frac{r}{n}} R + 3\sqrt{\frac{ln\frac{2}{\delta}}{2n}} \tag{2.63}$$

It turns out that minimizing the SVM primal objective is exactly equivalent to the task of minimizing the r.h.s. of the above expression, which can be demonstrated as follows. First, the problem of minimizing $R(h)$ for the SVM is expressed as

$$\min_{\mathbf{w} \in \mathcal{H}, R:R \geq \|\mathbf{w}\|_{\mathcal{H}}} \frac{1}{n} \sum_{i=1}^{n} [1 - y_i \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle_{\mathcal{H}}]_+ + 2\sqrt{\frac{r}{n}} R \tag{2.64}$$

Clearly, for a fixed $\mathbf{w}$, then the objective is minimized when $R = \|\mathbf{w}\|_{\mathcal{H}}$, therefore, the following equivalent problem is solved

$$\min_{\mathbf{w} \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} [1 - y_i \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle_{\mathcal{H}}]_+ + 2\sqrt{\frac{r}{n}} \|\mathbf{w}\|_{\mathcal{H}}^2 \tag{2.65}$$

where squaring the last term ($\|\mathbf{w}\|_{\mathcal{H}}^2$) yields an equivalent problem. After scaling (multiplying the

22

objective by $1/4\sqrt{n/r}$ and defining $C \triangleq 1/4\sqrt{rn}$, the SVM primal formulation (without bias) results

$$\min_{\mathbf{w} \in \mathcal{H}} C \sum_{i=1}^{n} [1 - y_i \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle_{\mathcal{H}}]_+ + \frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 \qquad (2.66)$$

Therefore, the primal formulation directly minimizes the RC-based upper bound on empirical risk, $R(h)$, for the SVM.

Solving the SVM QP Problem

Both the primal and dual formulations of the SVM optimization problem are convex since the objective function and set of constraints are both convex. As a result, the SVM possesses a global minimum, which is guaranteed to be unique for the dual formulation if the kernel matrix is positive definite. The primal formulation is ideal for large datasets where the number of variables are on the order of the number of features within the dataset. However, the number of inequality constraints are on the order of the dataset size for the primal formulation and immediate support for kernel functions is not provided (except as noted previously and also discussed in Chapelle *et al.* [13]).

The dual formulation readily admits the use of kernel functions; however, the number of variables are on the order of the size of the dataset. The Hessian of the dual formulation is generally dense while the solution tends to be sparse (for the ℓ_1-loss formulation). The memory storage requirements associated with the Hessian represents the primary challenge for the dual formulation since the memory requirement grows quadratically with dataset size. In addition, the lack of sparsity in the Hessian precludes the use of sparse methods to mitigate memory usage and computational complexity. The dual formulation is the most widely used due to its kernel function support while the primal formulation may be more commonly used with very large datasets when a linear kernel is acceptable.

If the training set size is sufficiently small so that the Hessian can fit within memory, an "off-the-shelf" or conventional quadratic programming or convex optimization algorithm such as MINOS [64], LOQO [91], QPOPT [31], and SNOPT [32] may be employed. However, the problem can quickly become intractable for these methods even for moderately-sized problems. Therefore, the most commonly used optimization techniques are specifically tailored to the SVM formulation. A detailed review of optimization methods for the SVM formulation is given in Chapter 4.

To gain a basic understanding of the majority of techniques reported in the literature for SVM training, it is important to review the conditions of optimality for the convex quadratic programming problem. The KKT (Karush-Kuhn-Tucker) conditions provide the *necessary* conditions of optimality for an optimization problem with both equality and inequality constraints [66]. Consider the following general problem,

$$\min_{x} f\left(\mathbf{x}\right) \tag{2.67}$$

$$\text{s.t. } g_i\left(\mathbf{x}\right) \geq 0 \tag{2.68}$$

$$h_j\left(\mathbf{x}\right) = 0 \tag{2.69}$$

where $i \in \{1, ..., m\}$ for a set of m inequality constraints and $j \in \{1, ..., n\}$ for a set of n equality constraints. The Lagrangian is

$$\mathcal{L}\left(\mathbf{x}, \mu, \nu\right) = f\left(\mathbf{x}\right) - \sum_{i=1}^{m} \mu_i g_i\left(\mathbf{x}\right) - \sum_{j=1}^{n} \lambda_i h_j\left(\mathbf{x}\right) \tag{2.70}$$

The variables μ_i and ν_i are referred to as Lagrange multipliers (or KKT multipliers). The KKT conditions consist of the stationary condition

$$\nabla_{\mathbf{x}} \mathcal{L}\left(\mathbf{x}^*, \mu^*, \nu^*\right) = 0 \tag{2.71}$$

24

the primal feasibility conditions

$$g_i(\mathbf{x}^*) \geq 0 \ \forall \ i \in \{1, ..., m\} \tag{2.72}$$

$$h_j(\mathbf{x}^*) = 0 \ \forall \ j \in \{1, ..., n\} \tag{2.73}$$

the dual feasibility conditions

$$\mu_i^* \geq 0 \tag{2.74}$$

and, finally, the complementary conditions

$$g_i(\mathbf{x}^*) \mu_i^* = 0 \tag{2.75}$$

The complementary condition requires one or both quantities $g_i(\mathbf{x}^*)$ or μ_i^* be zero. The solution satisfies the *strict* complementary condition if only one quantity, not both, is zero. The Lagrange multipliers provide a measure of the problem sensitivity to each of the constraints. If the multiplier for the inequality constraint is zero, $\mu_i^* = 0$, this implies that the inequality constraint is automatically satisfied at the optimal solution, and, in fact, could be removed from the set of constraints without affecting the optimal solution. On the other hand, a large value for the Lagrange multiplier indicates the problem is highly sensitive to the associated constraint.

In general, the KKT conditions are not sufficient for optimality; however, it can be shown that the KKT conditions are necessary *and* sufficient for a convex problem, and, are, therefore, necessary and sufficient conditions for optimality for the SVM problem. The Lagrangian for the ℓ_1-loss SVM formulation is as follows

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r}) = \frac{1}{2}\|\mathbf{w}\|_{\mathcal{H}}^2 + C\sum_i \xi_i - \sum_i \alpha_i \left(y_i \left(\langle \mathbf{x_i}, \mathbf{w}\rangle_{\mathcal{H}} + b\right) - 1 + \xi_i\right) - \sum_i r_i \xi_i \tag{2.76}$$

and the optimality conditions for the SVM primal formulation are

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x_i} = 0 \tag{2.77}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.78}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}} = C - \alpha_i - r_i = 0 \tag{2.79}$$

$$y_i \left(\langle \mathbf{x_i}, \mathbf{w} \rangle + b \right) - 1 + \xi_i \geq 0 \tag{2.80}$$

$$\xi_i \geq 0 \tag{2.81}$$

$$\alpha_i \geq 0 \tag{2.82}$$

$$r_i \geq 0 \tag{2.83}$$

$$\alpha_i \left(y_i \left(\langle \mathbf{x_i}, \mathbf{w} \rangle + b \right) - 1 + \xi_i \right) = 0 \tag{2.84}$$

$$r_i \xi_i = 0 \tag{2.85}$$

The variables α_i and r_i are both referred to as the dual variables of the problem and the constraints $\alpha_i \geq 0, r_i \geq 0$ are dual constraints. In cases where the dual formulation of the SVM is worked with directly, the optimality conditions are derived as follows. The Lagrangian for the dual formulation becomes

$$\mathcal{L} \left(\boldsymbol{\alpha}, \beta, \mathbf{s}, \mathbf{t} \right) = \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j k \left(\mathbf{x}_i, \mathbf{x}_j \right) - \sum_i \alpha_i - \beta \sum_i y_i \alpha_i - \sum_i s_i \alpha_i - \sum_i t_i \left(C - \alpha_i \right) \tag{2.86}$$

and the resulting optimality conditions are

$$\sum_j y_i y_j \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) - 1 - \beta y_i - s_i + t_i = 0 \ \ \forall i \tag{2.87}$$

$$\alpha_i \geq 0 \tag{2.88}$$

$$\left(C - \alpha_i\right) \geq 0 \tag{2.89}$$

$$s_i \geq 0 \tag{2.90}$$

$$t_i \geq 0 \tag{2.91}$$

$$\sum_i y_i \alpha_i = 0 \tag{2.92}$$

$$s_i \alpha_i = 0 \tag{2.93}$$

$$t_i \left(C - \alpha_i\right) = 0 \tag{2.94}$$

The dual formulation optimality conditions are referenced extensively in this research.

CHAPTER 3: ACTIVE SET METHOD

The active set method can be used for solving the class of problems with a quadratic objective along with a set of linear equality and inequality constraints. Therefore, it is a suitable method for solving the QP problem associated with the SVM. The premise of the active set method is as follows. If it is known *a priori* which of the inequality constraints are satisfied as equality constraints and which can be ignored at the solution, then an equivalent equality- constrained problem can be posed and simply solved using the method of Lagrange. Unfortunately, the equivalent equality-constrained problem is not known in advance. The active set method sequences through the set of inequality constraints to form a series of equality constrained problems guided by gradient information and feasibility. The active set method terminates upon finding an equality-constrained problem satisfying the remaining inequality constraints as well as the conditions of optimality.

The active set method is related to the simplex method [20], but extends to quadratic programming. The key difference is that the optimal solution for quadratic programming is no longer guaranteed to be on a vertex of the feasible region. In fact, the simplex method can be extended to quadratic programming in a straight-forward manner.

In general, the active set method can have worst-case exponential convergence where it degrades into a combinatorial search. On the other hand, methods such as the interior point method have proven polynomial convergence. Fortunately, in practice, the active set method is rarely seen to exhibit exponential performance and is still considered to be competitive against methods such as the interior point method. Further, convergence is proven in the absence of degeneracy, which can be resolved, if it occurs, by either modifying the set of constraints or with Bland's pivoting rule [4].

Consider the following optimization problem.

$$\min_{\mathbf{x}} \ \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{f}^T\mathbf{x} \tag{3.1}$$

$$\text{s. t. } \mathbf{A}\mathbf{x} = \mathbf{b} \tag{3.2}$$

$$\mathbf{C}\mathbf{x} \geq \mathbf{d} \tag{3.3}$$

The following partitions the inequality constraints $\mathbf{C}\mathbf{x} \geq \mathbf{d}$ into the set of active constraints $\mathcal{A}$ and inactive constraints $\mathcal{I}$ where

$$\mathbf{c_i}^T\mathbf{x} = d_i \ \forall \ i \in \mathcal{A} \tag{3.4}$$

$$\mathbf{c_i}^T\mathbf{x} \geq d_i \ \forall \ i \in \mathcal{I} \tag{3.5}$$

and $\mathbf{c}_i^T$ is the ith row of C. The active constraints, are then, those constraints that are satisfied strictly as an equality constraint while the remaining inactive constraints may be satisfied as inequality constraints. Note that it is possible for an inactive constraint to be satisfied as an equality constraint.

Active Set Algorithm

The active set method solves the following equality-constrained problem at each iteration, k given a partition into the set of active constraints $\mathcal{A}_k$ and inactive constraints $\mathcal{I}_k$,

$$\min_{\mathbf{x}} \ \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{f}^T\mathbf{x} \tag{3.6}$$

$$\text{s. t. } \mathbf{A}\mathbf{x} = \mathbf{b} \tag{3.7}$$

$$\mathbf{c_i}^T\mathbf{x} = d_i \ \forall \ i \in \mathcal{A}_k \tag{3.8}$$

The method begins with some initial feasible solution $\mathbf{x}_0$ and corresponding partition $\mathcal{A}_0$ or $\mathcal{I}_0$. The initial feasible solution is one that satisfies the feasibility conditions (satisfies the set of constraints) without necessarily being optimal. In many cases, the initial solution can be found trivially. Otherwise, the two-phase method or M-method, for example, may be used to find an initial feasible solution [66].

The KKT optimality conditions for the posed QP problem are

$$\mathbf{Q}\mathbf{x} + \mathbf{f} - \mathbf{r}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) - \sum_{i \in \mathcal{I} \cup \mathcal{A}} \lambda_i \mathbf{c_i} = 0 \tag{3.9}$$

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{3.10}$$

$$\mathbf{C}\mathbf{x} \geq \mathbf{d} \tag{3.11}$$

$$\lambda_i \geq 0 \ \forall i \tag{3.12}$$

$$\lambda_i \left(\mathbf{c}_i^T \mathbf{x} - d_i \right) = 0 \tag{3.13}$$

The components λ_i represent the set of Lagrange multipliers associated with each constraint $\mathbf{c}_i \mathbf{x} \geq d$. Per the complementary condition (last condition above), given a solution $\mathbf{x}^*$ then $\lambda_i = 0$ for $\mathbf{c}_i^T \mathbf{x}_i^* > d_i$ and $\lambda_i \geq 0$ otherwise. The Lagrange multiplier can be viewed as the sensitivity of the solution to the constraint. For $\lambda_i > 0$, the solution $\mathbf{x}^*$ is sensitive to the constraint with the magnitude of λ_i representing the degree of sensitivity while for $\lambda_i = 0$, the constraint could be removed without impacting the solution. The active set method ensures feasibility (constraints are satisfied) at all times with optimality being achieved when $\lambda_i \geq 0$ for all i.

Starting with the initial feasible solution, the active set method finds and removes an active constraint $i \in \mathcal{A}$ such that $\lambda_i < 0$. The active set is, then, adjusted and instead of directly computing

$\mathbf{x}_k$ using (3.6), the descent direction $\mathbf{p} = \mathbf{x}_k - \mathbf{x}_{k-1}$ is solved as follows

$$\min_{\mathbf{p}} \quad \tfrac{1}{2}\mathbf{p}^T Q \mathbf{p} + \mathbf{p}^T Q \mathbf{x}_{k-1} + \mathbf{f}^T \mathbf{p} \tag{3.14}$$

$$\text{s. t.} \qquad \mathbf{A}\mathbf{p} = 0 \tag{3.15}$$

$$\mathbf{c_i}^T \mathbf{p} = 0 \; \forall \, i \in \mathcal{A}_k \tag{3.16}$$

It is straight-forward to show the following system of equations solves the equality-constrained problem above

$$\begin{pmatrix} Q & -B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} -\mathbf{f} - Q\mathbf{x}_{k-1} \\ \mathbf{0} \end{pmatrix} \tag{3.17}$$

where B is defined as

$$B = \begin{pmatrix} A \\ C_{\mathcal{A}_k} \end{pmatrix} \tag{3.18}$$

and $C_{\mathcal{A}_k}$ contains the rows of C indexed by $\mathcal{A}_k$. This system of equations is often referred to as the KKT (Karush-Kuhn-Tucker) system and the matrix as the KKT matrix.

This descent direction is followed until either an inactive constraint becomes infeasible or the solution $\mathbf{x}_k$ is reached. That is, given $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha\mathbf{p}$ the maximum $\alpha \in [0, 1]$ is found such that none of the inactive constraints are violated, or

$$\mathbf{c}_i^T \left(\mathbf{x}_{k-1} + \alpha\mathbf{p} \right) \geq d_i \; \forall \, i \in \mathcal{I}_k. \tag{3.19}$$

For $\mathbf{c}_i^T \mathbf{p} > 0$ this will be true for any value of α. However, for $\mathbf{c}_i^T \mathbf{p} < 0$, feasibility is satisfied for α such that

$$\alpha \leq \frac{d_i - \mathbf{c}_i^T \mathbf{x}_k}{\mathbf{c}_i^T \mathbf{p}} \tag{3.20}$$

The maximum distance that can be taken along the descent path, α, is computed as

$$\alpha = \min \left\{ 1, \frac{d_i - \mathbf{c}_i^T \mathbf{x}_k}{\mathbf{c}_i^T \mathbf{p}} \; \forall \, i \in \mathcal{I} \cap \mathbf{c}_i^T \mathbf{p} < 0 \right\} \quad (3.21)$$

If $\alpha \neq 1$, then the limiting inactive constraint is added to $\mathcal{A}_k$. Following this, a new descent direction is computed and the process repeated until a full descent can taken without requiring an active set transition or $\alpha_i = 1$. This, represents a new feasible solution. The algorithm terminates if optimality is reached ($\lambda_i \geq 0$ for all i), otherwise a new constraint such that $\lambda_i < 0$ is selected to leave active set and the process repeats.

The method just described is referred to as a primal active set method since feasibility is maintained on the set of primal constraints throughout all iterations. The dual active set method (see [35]), instead, maintains feasibility on the set of dual constraints found by forming the Wolfe dual of the original optimization problem. Instead of explicitly forming the dual, the method works by maintaining all of the optimization conditions with the exception of feasibility.

Efficient Solution of the KKT System

The null space method (see [66]) can be used for solving the KKT system (3.17). Consider, again, the KKT system

$$\begin{pmatrix} Q & -B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} -\mathbf{c} \\ \mathbf{b} \end{pmatrix} \quad (3.22)$$

where $\mathbf{c} = \mathbf{b} + Q\mathbf{x}_{k-1}$ and $\mathbf{b} = 0$, $Q \in \mathbb{R}^{n \times n}$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{R}^n$, $B \in \mathbb{R}^{m \times n}$, and $\mathbf{b} \in \mathbb{R}^m$, with n variables and m constraints. Use $Z \in \mathbb{R}^{n \times (n-m)}$ to represent the basis for the null space of B such that $BZ = 0$ and further define $Y \in \mathbb{R}^{n \times m}$ to be any matrix such that the combined matrix $[Z|Y]$

is full rank. The vector $\mathbf{x}$ can be decomposed as follows

$$\mathbf{x} = Z\mathbf{x}_z + Y\mathbf{x}_y \tag{3.23}$$

Upon substituting this expression for $\mathbf{x}$ within the equality constraint $B\mathbf{x} = \mathbf{b}$ the following is obtained.

$$B\left(Z\mathbf{x}_z + Y\mathbf{x}_y\right) = \mathbf{b} \tag{3.24}$$

$$BY\mathbf{x}_y = b \tag{3.25}$$

where the last expression results from the fact that $BZ = 0$. From this perspective, the component $Y\mathbf{x}_y$ can be seen as a particular solution while the component $Z\mathbf{x}_z$ explores the infinite space of solutions where $B\mathbf{x} = \mathbf{b}$ continues to be satisfied. From Equation 3.22, the following sets of equations are obtained

$$QY\mathbf{x}_y + QZ\mathbf{x}_z - B^T\boldsymbol{\lambda} = -\mathbf{c} \tag{3.26}$$

$$BY\mathbf{x}_y = \mathbf{b} \tag{3.27}$$

Multiplying the first equation by Z^T results in

$$Z^T QZ\mathbf{x}_z = -Z^T\mathbf{c} + Z^T B^T\boldsymbol{\lambda} - Z^T QY\mathbf{x}_y \tag{3.28}$$

$$Z^T QZ\mathbf{x}_z = -Z^T\mathbf{c} - Z^T QY\mathbf{x}_y \tag{3.29}$$

The reduced Hessian $Z^T QZ$ will be positive definite if the matrix in Equation (3.22) is non-singular [Nocedal, Lemma 16.1] [66]; therefore, a Cholesky factorization can be used. In addition, the size of $Z^T QZ$ will increase or decrease by one row/column at each iteration due to the incremental nature of the active set method. As a result, efficient rank-one updates can be used to update

33

$Z^T Q Z$ that represent a $O\left(n^2\right)$ computational complexity as opposed to the $O\left(n^3\right)$ complexity that would be incurred if the Cholesky factorization had to be recomputed from scratch at each iteration.

Finally, the component, $\boldsymbol{\lambda}$, is computed by multiplying the first equation of (3.22) by Y^T to obtain

$$Y^T Q \mathbf{x} + Y^T B^T \boldsymbol{\lambda} = -Y^T \mathbf{c} \tag{3.30}$$

$$(BY)^T \boldsymbol{\lambda} = -Y^T (\mathbf{c} + Q\mathbf{x}) \tag{3.31}$$

The null space method is a suitable method as long as Z can be computed efficiently. However, there is not necessarily one unique solution for Z and it is possible to have poor scaling that results in ill-conditioning of the reduced Hessian [66]. Often, if an analytical form for Q is not known, a separate QR factorization is used $Z = QR$ and maintained with rank-one updates. Once Z is updated, the Cholesky factorization of the reduced Hessian is subsequently updated.

If the reduced Hessian is large, for example when there are a large number of variables with a limited number of constraints, memory consumption may become an issue. In these instances, iterative methods, such as the Krylov subspace methods can be used. The conjugate residual may be used on the KKT system since it can handle indefinite matrices, otherwise, the conjugate gradient can be used along with the null space method to solve the equations involving the reduced Hessian matrix.

Convergence and Degeneracy

It is relatively straight-forward to show that the active set method convergences for a strictly convex quadratic programming problem assuming a strict decrease of the objective is achieved at each iteration (see [66]). Degeneracy occurs when no progress is made upon adding or removing an

active constraint (or $\alpha = 0$ results when computing (3.21)) and can arise when a linearly dependent constraint or a *weakly* active constraint ($\lambda_i = 0$ for $i \in \mathcal{A}_k$) is encountered. If this occurs, it is possible to revisit a previous active set for some iteration $p \geq q$ where $\mathcal{A}_p = \mathcal{A}_q$, which can result in cycling and lack of convergence. In most cases, degeneracy is a rare event, especially if linearly dependent constraints are pruned before solving the problem, and, therefore, methods for handling this may not be necessary. However, as both Nocedal [66] and Fletcher [28] suggest, methods borrowed from linear programming, such as Bland's pivoting rule [4], can be applied.

Semi-Definite Hessian

Difficulties arise within the active set method when the Hessian, Q, is semi-definite. This is of importance within the SVM community since the widely used linear kernel as well as the existence of duplicate data points will result in a semi-definite QP problem. Perhaps most importantly, low-rank kernel approximations, which provide significant speed-ups for SVM training, cannot be used unless the active set method is modified to allow semi-definite Hessians. The problem occurs when an active set, $\mathcal{A}_k$, is chosen that results in the KKT system defined in Equation 3.17 becoming singular. The conventional methodology for handling this scenario is as follows. First, the singular system is detected by either explicitly solving for the rank of the KKT matrix or by detecting the singularity during updates to a factorization of the KKT matrix. For example, a zero or near-zero diagonal entry may be detected while updating the Cholesky factorization. If the KKT matrix is singular, a descent direction is carefully chosen that will modify the active set so that the next KKT system is non-singular. Care must be taken in choosing the descent direction to ensure cycling is avoided. This is the basis of the SVM-QP algorithm, an active set implementation for SVM training with semi-definite kernels [77].

Alternatively, the active set method can be modified to avoid working sets yielding a singular

KKT system; thus eliminating the need for specialized solve routines and their inherent numerical difficulties. The inertia-controlling method for general quadratic programming, [33], an example implementation for indefinite Hessians, actively performs pivots to control the inertia (number of positive, negative, and singular eigenvalues) of the KKT system. Instead of actively tracking the inertia of the KKT system, it is possible, in the case of a semi-definite Hessian, to design an algorithm with a pivoting strategy that automatically avoids the singular KKT system. The revised simplex method for semi-definite quadratic programming, introduced by Rusin [76], is one such example. The primary focus of this research is to demonstrate the utility of Rusin's method for SVM training and to extend these concepts to the important class of regularization path-following algorithms.

CHAPTER 4: LITERATURE REVIEW OF SVM TRAINING METHODS

Solving the SVM Problem

The SVM formulation represents a convex optimization problem, and, while there are a number of methods available from the numerical optimization community for solving this type of problem, the Support Vector Machine (SVM) formulation is particularly challenging since the number of constraints is on the order of the dataset size and the Hessian matrix, when solving the dual formulation, scales quadratically with the data set size. This is an issue especially for optimization algorithms that must store the entire Hessian in memory or, even worse, must compute the inverse of the Hessian. As an example, a data set containing only 10,000 data points would require 800 MBytes of RAM to store the entire Hessian with double floating point precision. Many off-the-shelf numerical optimization cannot be applied due to the memory constraints or may exhibit slow training times or poor convergence properties for even moderately sized data sets. As a result, the pursuit of faster training methods remains a very active area of research in the SVM community. A sample of the relevant techniques include, interior point methods [100], [27], active set methods [77], [80], [95], [10], [61], [84] gradient projection [18], [66], [106], [101], [52], Newton-like methods [46], projected conjugate gradient [99], geometry-based methods such as convex hull-based methods [48], [82], approximate minimum enclosing ball (CoreSVM) [89], stochastic gradient descent [83], cutting planes [42], sequential minimal optimization (SMO) [71], [49], [25], chunking [92], [68], [5], and decomposition [41], [52], [103]. In addition to these approaches, other key methods include convex-hull based techniques [48], [105], [102] as well as CoreSVM [89], DirectSVM [75], Smooth SVM (SSVM) [54], Reduced SVM [53], and Projected Conjugate Gradient (PCG) [99]. Figure 4.1 provides a limited, top-level summary of some of the important methodologies but does not truly convey the extensive literature on this topic. By far the most

popular optimization methods appear to be SVMLight and the SMO algorithm as almost all new methodologies are compared against these last two. The SMO algorithm tends to be fast, has linear memory requirements, and is easy to implement. However, SVM training is still slow compared to the training times for other machine learning technologies, and there is a push to improve training performance for very large datasets. The following provides a high-level survey of some of the more important reported techniques.
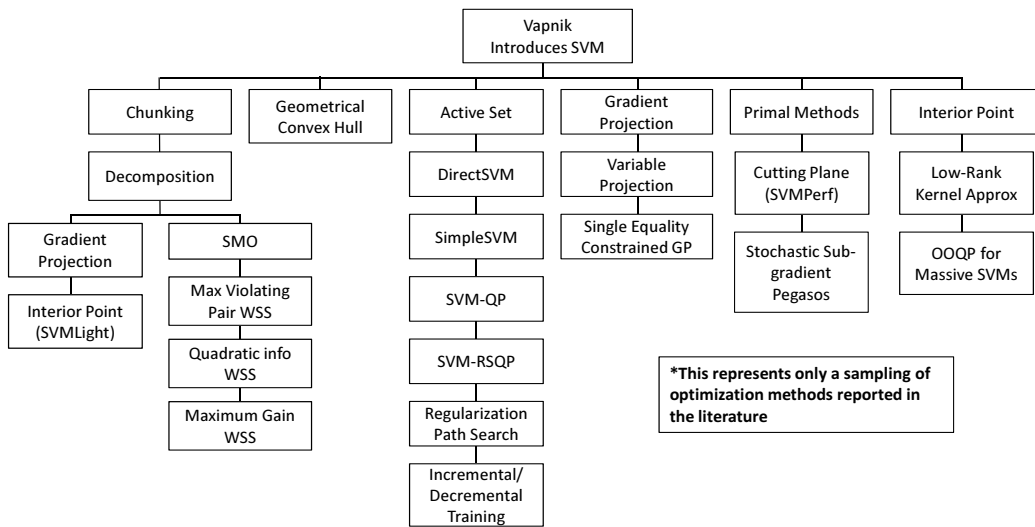
Figure 4.1: A depiction of the key optimization algorithms applied to solve the SVM quadratic programming problem. This represents only a sample of the techniques that have been published.

Decomposition Methods

Vapnik originally introduced an optimization algorithm for the SVM problem that included a methodology later referred to as "chunking" to solve the memory issues potentially associated with large training problems [92]. Boser *et al.* [5] employ and describe the chunking method in their work. The chunking method starts with an initial subset of the data as a training set. Optimization is performed on that subset of data to find the maximal margin hyperplane and, then,

those data points that are not support vectors $\alpha_i = 0$ are discarded from the "chunk" and a fixed number of data points are chosen from the remaining training set that are incorrectly classified or are on the wrong side of the margin to form a new "chunk". This process is repeated until all data points are on the correct side of the margin and the "chunk" contains the set of support vectors $\alpha_i > 0$. Each chunk is optimized with commercially available optimization software.

The disadvantage of the chunking method is that the chunk size will eventually grow to be on the order of the number of support vectors identified in the final solution. Typically, the number of support vectors will grow with the training set size, and eventually, the algorithm may encounter difficulties with memory. Osuna *et al.* [68] introduced a decomposition method, which instead of allowing a variable sub-problem size, employs a fixed sub-problem size. The authors establish a working set $\mathcal{B}$ whose size $|\mathcal{B}| < n$ where n is the training set size. The algorithm requires that $|\mathcal{B}|$ is sufficiently large to contain all of the support vectors identified in the final solution. The authors employ the MINOS algorithm for solving the sub-problem and apply the algorithm to the face detection problem. The primary advantage of this algorithm, over the chunking algorithm, is that memory consumption can be reliably predicted. The most important contribution, however, is that the authors prove that replacing non-support vectors within the current working set $\mathcal{B}$ with variables from the non-working set $\mathcal{N}$ that violate the KKT conditions will result in a strict decrease of the objective function. This later becomes the basis for all decomposition methods.

The decomposition method gained popularity with the introduction of SVMLight by Joachims [41] and Sequential Minimal Optimization (SMO) by Platt [71]. Unlike the method introduced by Osuna, these decomposition methods employ a fixed working set size $|\mathcal{B}|$ that does not necessarily contain all of the support vectors. In the method reported by Joachims, a small even-sized working set size, q, is chosen, typically between $q = 2$ and $q = 20$. Joachims introduced a new working set selection strategy as an optimization problem based upon the feasible direction method by Zoutendijk. The problem is solved efficiently by performing a partial sort. A series of QP sub-

problems, of size q, are then solved. The SVMLight method employs a primal-dual interior point method introduced by Vanderbei [91] and implemented by Smola [86] (pr_LOQO) to solve the quadratic programming sub-problem. In general, any suitable quadratic programming method can be applied. The author also introduces the concept of *shrinking*. The idea behind *shrinking* is to identify data points that are not likely to become support vectors and eliminate those variables from the quadratic programming problem. This can realize a significant speed-up in some problems as the working set selection problem size can be greatly reduced.

In addition to *shrinking*, *kernel caching* can be employed to further reduce kernel evaluations, usually the most expensive computation of the algorithm. Joachims introduces the notion of employing a LRU (least-recently used) kernel caching scheme to take advantage of the fact that the algorithm tends to revisit certain columns of the kernel matrix repeatedly. The idea of *shrinking* and *kernel caching* have become a mainstay of the most efficiently implemented SVM training algorithms.

As an alternative to the the primal-dual interior point method for solving the sub-problem associated with decomposition, more recently, Serafini *et al.* [81] introduced a gradient projection method referred to as a generalized variable projection method (GPVM) for solving the sub-problem. The authors indicate that the GPVM method is a viable alternative to MINOS and LOQO. LOQO is the sub-problem solver employed by *SVMLight* and MINOS is employed in the algorithm reported by Osuna *et al.*. Zanni *et al.* [104] further extend the GPVM algorithm to a parallel implementation for large datasets.

The Sequential Minimal Optimization (SMO) algorithm introduced by Platt [71] takes decomposition to an extreme using a working set size of two. At this extreme, the sub-problem has an analytical solution and, therefore, doesn't require a numerical solver. For working set selection, the algorithm employs a set of heuristics, which alternate between exploring all data points and the

group of non-bound support vectors looking for KKT violators. The SMO algorithm maintains an error cache enabling a significant speedup. The SMO algorithm is easy to implement using pseudocode provided in [71]. Compared to other algorithms, SMO employs a large number of cheap iterations; it is not unusual for iterations to exceed one million iterations.

Following Platt's initial introduction of SMO, Keerthi *et al.* pointed out inefficiencies in the heuristics employed for selecting each working set [49]. The authors noted that the original SMO algorithm attempts to estimate the offset, b, at each iteration and suggested that there is a range of possible values for b that can be derived from the KKT optimality conditions, i.e. $[b_{lo}, b_{up}]$. The test for optimality within the SMO algorithm requires an estimate for b, which is based upon the current working set at each iteration within the original SMO implementation. However, since there is a range of possible values for b, there can be a loss of efficiency as the algorithm might be delayed in detecting optimality. As a result, Keerthi *et al.* suggest using a dual threshold system (b_{lo}, b_{up}) when testing for optimality. Their implementation showed considerable improvement in terms of training speed when compared to the original SMO algorithm. In addition, the working set selection method discussed by the authors is equivalent to the method proposed by Joachims for SVMLight [41] when $q = 2$. Fan *et al.* [25] provide, yet, a faster implementation of SMO. They introduce a new working set selection method that includes 2nd order information. The current implementation of LIBSVM [12], a popular and highly referenced SMO implementation, is based upon this publication.

Glasmachers *et al.* [34], introduce a maximum gain working set selection method. The maximum gain working set method finds a pair of points that will maximize the change in the objective function. In order to accurately analyze the affect of any pair of points on the objective function, the affect of clipping must be considered. One method for determining the "gain" of a particular pair of points is to test the step made by the two data points and observe the change in the objective function. Note that similar methods are applied in the simplex method for linear programming

such as the steepest edge method [62]. In general, a greedy search for the pair of points that will maximize the gain would require $l\,(l-1)$ or $O(l^2)$ tests where l in the worst case can be the dataset size. Instead of greedily testing all points, the authors suggest making use of the kernel cache associated with the two data points last updated to find a new pair of points for update. A proof of convergence for the hybrid MVP (most violating pair), related to Keerthi's [49] method, and MG (maximum gain) working set selection is provided.

The convergence properties of decomposition methods have been studied [47], and asymptotic convergence was shown for a generalized SMO algorithm, which includes the working set selection introduced in [49] as a special case. The proof of asymptotic convergence was extended to the generalized working set selection method introduced by Joachims by Lin *et al.* [58]. In [57] Lin further shows that the SMO algorithm exhibits worst-case linear convergence rates.

Primal Methods

Chapelle *et al.* [13] introduced the notion of solving the primal problem instead of the Wolfe dual problem. They point out that more progress is made against the primal objective versus the dual objective when a fixed number of conjugate gradient iterations are performed. They also point out that kernel methods can still be applied in the primal domain by using the *Representer* theorem.

Subsequently, Joachims *et al.* [42] introduced SVMPerf, a cutting planes method for solving the primal problem with a linear kernel on large datasets. Shalev *et al.* [82] introduce the Pegasos algorithm, which solves the primal problem by employing a stochastic subgradient methodology along with a trust region method. The author reports a several order of magnitude improvement in training speed when compared to SVMLight and a significant improvement when compared to SVMPerf [42]. Both implementations are very fast (SVMPerf and Pegasos), and more importantly,

have a convergence rate that does not depend upon the dataset size but rather the desired accuracy, ϵ, making them ideal for very large datasets. The Pegasos algorithm [83] claims faster convergence than SVMPerf since it does not have a square dependence upon ϵ.

However, these methods are limited to the linear kernel. In the case of Pegasos, the authors propose employing the *Representer* Theorem, as suggested by Chapelle, to enable kernel methods; however, details on implementation are not given. The SVMPerf algorithm can be modified to use non-linear kernels, [43]; however, this works again with dual variables and employs kernel approximations to make the problem tractable.

The SMO algorithm can also be implemented more efficiently in the primal domain when using a linear kernel. In this case, the number of variables is determined by data dimensionality instead of data set size. LIBLINEAR [24] is an efficient implementation based upon the SMO algorithm LIBSVM [25]. Finally, Keerthi *et al.* [46] introduce a Newton-like method for solving the primal problem for a linear kernel on very large datasets. They show that they are able to outperform SMO and SVMLight.

Interior Point Method

The interior point method appears advantageous as it possesses a guarantee of polynomial convergence unlike other methods such as the active set method which can exhibit worst-case exponential convergence. Further, the interior point method is considered ideal for problems having dense Hessians. While the active set method explores the convex polytope formed from the set of linear inequality constraints, or, in general, the border of the feasible region, the interior point method explores the interior of the feasible region and only settles on the border at optimality. Interior point techniques have been successfully applied to problems with millions of training samples [100].

The SVMLight algorithm, [41], a decomposition method, also uses an interior point method as an inner solver.

In Fine *et al.*, [27], the authors introduce a primal-dual interior point method for the SVM. In general, the system of equations solved by the interior point method at each iteration is expected to be large, dense and ill-conditioned. As a result, they use a method based upon an incomplete Product Form Cholesky factorization to factorize the system of equations. They show that the alternative, using a Sherman-Morrison-Woodbury (SMW) formula for performing low-rank updates to the inverse of the matrix, can become numerically unstable. They also employ an incomplete Cholesky factoriztion as a low-rank approximation to the kernel matrix in effort to reduce the size of the problem. The approximation method has a $O(nk)$ storage and $O(nk^2)$ computational complexity, where k is the rank of the approximation and n is the number of data points. They report favorable performance against the SMO and SVMLight algorithms for the linear kernel.

Ferris *et al.* [26] introduce an interior point method, referred to as OOQP, for solving "massive" support vector machines. They use a Mehrotra predictor-method and employ the Sherman-Morrison-Woodbury (SMW) technique for perform performing low-rank updates to an inverse to increase the efficiency. In their implementation they found that the number of iterations remained constant as the dataset size increased while the time per iteration scaled linearly with the dataset size and quadratically with the data dimensionality (when using a linear kernel). They show that the interior point method can be employed to solve a non-separable problem with 60 million examples in 15 to 22.5 hours on a standard work station. While they reported excellent results for the linear kernel, it is not clear how the method would scale in the case of a non-linear kernel such as the RBF kernel. Here, the implicit dimension can be very large (based upon the effective rank of the matrix) and the problem may quickly become intractable for large datasets.

Woodsend *et al.* [100] introduce yet another interior point implementation based upon separability

of the primal, dual variables. The author's point out that active set methods can become inefficient with noisy data sets (those that are not clearly linearly separable) and that the advantage of the IPM method is that a decision is not made concerning the active set until the algorithm is near the solution. Note that the authors cite numerous decomposition methods, which are not active set methods in the traditional sense such as those introduced in [77], [61], [10], [95]. The authors further claim to create an algorithm that has much improved numerical stability when compared to SMW methodologies while avoiding inefficiencies exposed in the product Cholesky factorization method introduced by Fine *et al.* and providing a level of parallelism that allows for easier adaptation to a parallel computing environment. The reported method was shown to outperform all methods with the exception of the LIBLINEAR algorithm [24], a version of LIBSVM adapted for use of linear kernels on large-scale datasets.

Overall, while the interior point method appears competitive for linear kernels, competitive performance does not seem to have been reported for non-linear kernels. One reason for this is likely due to the fact that significant reductions in memory and or computations are not as readily realized for non-linear kernels, even with low-rank approximations. Further, a key issue for the interior point method is that computational complexity has been shown to grow cubically with the size of the data set [27].

<center>Geometric Approaches</center>

In another class of methods, the SVM problem is posed as a geometry problem where the optimal separating hyperplane is derived from the convex hull of the data points for each class. Given the smallest convex hull containing each class is found, the perpendicular bisector of the line joining the closest points between the two convex hulls represents the maximum margin separating hyperplane. The "kernel" trick can be applied to these formulations. A detailed discussion of this

<center>45</center>

methodology can be found in [3].

Keerthi *et al.* [48] introduce, initially, a method that finds the nearest points between two convex hulls formed from each class of data. In this paper, they derive the nearest point problem formulation for the SVM problem and show that there is a direct correspondence between this nearest point algorithm and the original SVM problem meaning that the solution to the nearest point problem directly translates into the solution of the original problem. They derive a fast, iterative method for solving the nearest point problem and depict results competitive with Platt's original implementation of SMO. Research is continuing in this area such as in [105] where a random sampling approach is used to apply the convex hull method to large data sets and [102] where a fast, incremental learning method based upon convex hulls is introduced.

Gradient Projection Methods

The gradient projection method is a form of steepest descent where the gradient is projected onto the feasible region defined by the set of constraints. The gradient projection does not need to take the inverse of the Hessian and is efficient and simple to implement if the constraints are simple, such as in the case for simple box constraints. More importantly, the gradient projection does not typically require the Hessian be explicitly stored and appears to be ideal for large datasets. Further, unlike the active set method, multiple constraints can be activated and/or deactivated at each iteration. Unfortunately, for the SVM optimization problem, the gradient projection step is complicated by the existence of the single equality constraint, and, therefore, many of the SVM implementations specifically address this.

Dai *et al.* [19] introduce a gradient projection method for singly linearly constrained quadratic problems with box or bound constraints. In their method they alternate between solving a larger,

simpler bound-constrained problem without the equality constraint and a non-linear optimization problem for updating the Lagrange multiplier associated with the equality constraint. The non-linear optimization is solved with an outer secant method while a gradient projection is employed on the larger problem. The premise is that it is now easier to apply the gradient project method to the larger bound-constrained optimization problem after having removed the equality constraint.

Lee *et al.* [52] report on a primal-dual hybrid formulation of the SVM problem that becomes a minimax optimization problem but eliminates the equality constraint. The problem is solved with alternating gradient projection steps between the primal and dual variables. Zhu *et al.* [106] introduce a method for dealing with the single equality constraint within the context of solving the total variation problem for image restoration. In their method, they solve saddle point problem represented by the primal/dual hybrid problem. Wright recommends the use of this method for the SVM [101] and a version of this is introduced for solving the semi-parametric SVM in [52] from within, again, the context of a decomposition-based approach. A semi-parametric SVM admits the insertion of additional equality constraints that allow specification of *a priori* knowledge. Nonetheless, this results in multiple equality constraints in addition to the single equality constraint normally specified.

Serafini *et al.* [81] and Zanni *et al.* [104] both report on gradient projection methods for SVM training as sub-problem solvers for the decomposition method. In addition, Zanni *et al.* report on a parallel processing implementation for very large datasets.

Active Set Methods

The active set method is generally ideal for small to medium-sized problems [66]; however, it is also pointed out in Scheinberg [77] that the active set method is ideal when the Hessian is dense and

Vishwanathan *et al.* [95] point out the active set is superior when the solution is sparse, both valid for the SVM problem. The active set method, as compared to traditional decomposition methods, provides an *exact* solution rather than *approximate* solution and inherently supports incremental training. Typically, the active set method can be expected to converge with the number of iterations being at least on the order of the number of active constraints identified in the final solution [66]. For the SVM problem, then, the number of iterations is expected to be on the order of the number of support vectors, and this implies the number of iterations will scale linearly with the dataset size.

Several active set methods have been introduced in the literature for solving the SVM problem [77], [84] [38], [95], [10], [96] and [61]. An early version was introduced by Cauwenberghs *et al.* [10] where they introduced an incremental/decremental training algorithm. In their approach, they derive the updates (adiabatic increments) necessary for adding a single data point to a fully trained system. The idea is that the entire system can be trained by starting with a single data point and incrementally adding data points until a single pass has been made through all of the data points. Similarly, a method can be derived for updating the solution when a single data point is removed leading to an efficient way for performing a leave-one-out (LOO) estimation of generalization performance.

The Active SVM (ASVM) introduced by Mangasarian *et al.* [61] employs an active set method to efficiently solve the 2-norm soft margin problem (L2-SVM) (or ℓ_2- loss SVM) when using a linear kernel. They show the ℓ_2-loss formulation results in a simplification of the box constraints to a set of non-negativity constraints and eliminates the equality constraint. They employ the Sherman Woodbury Morrison (SMW) formula to take advantage of the fact that the kernel matrix rank depends upon the data dimensionality for a linear kernel and introduce a method for performing efficient incremental updates to the inverse of the Hessian. The authors report that the algorithm is suitable for massive data sets. A similar active set method is derived in [56], which also relies on

a the ℓ_2-loss formulation of the SVM problem.

The SimpleSVM algorithm introduced by Vishwanathan *et al.* [95]. defines a working set $\mathcal{W}$ consisting of only the non-bound support vectors $0 < \alpha < C$, also referred to as free variables. At each iteration, a data point belonging to the set of bound support vectors $\alpha = C$ or non-support vectors $\alpha = 0$ that violates the KKT optimality conditions is chosen for addition to the working set. Operation of the algorithm follows the basic active set implementation where the "working set" is the set of inactive constraints. After adding a point to the working set, a descent direction is computed and followed while variables in the working set that hit $\alpha_i = 0$ or $\alpha_i = C$ are transitioned out of the working set. The algorithm is initialized by selecting a pair of opposite-labeled data points that are closest neighbors using an efficient log-linear heuristic or randomized selection heuristic. The authors also recommend the use of a low-rank kernel approximations [85], [27] to improve speed and memory consumption; however, details are not given and results appear to be reported without this feature. Nonetheless, they report significant performance improvements over the Nearest Point Algorithm (NPA) [48] and SMO [71] when the fraction of support vectors is small.

In Shilton *et al.* [84], the authors show that the equality constraint within the SVM dual formulation represents a nuisance constraint and can be eliminated by using a primal-dual hybrid formulation of the SVM problem. This formulation becomes a min-max problem where minimization is performed against a subset of dual variables while maximization is perform against a subset of primal variables. However, it is still possible for the algorithm to encounter singular systems of equations when the Hessian is semi-definite. Therefore, the algorithm must explicitly detect rank degeneracies and adapt the solution method. Interestingly, the authors also point out the importance of choosing the descent direction when rank deficiencies are detected to ensure cycling and convergence failure does not result.

Kim *et al.* introduce a fast active set method for L$_1$ regularized regression that employs a form of block pivoting. The conventional active set method performs single pivoting where a single variable enters and/or leaves the set of active constraints at one time while block pivoting allows multiple transitions. The authors incorporate a backup routine, performing a series of single pivots, in those cases where cycling is detected and, therefore, providing a guarantee of convergence. They show favorable comparisons to the LARS (Least Angle Regression) algorithm [22], a popular method for solving this type of problem. However, in this work, it appears to be the case that the full system of equations involving the non-active constraints is re-solved at each iteration. Therefore, for a typical active set method, the gains in performance achieved by reducing the number of iterations may not be as significant where efficient rank-one updates to the system of equations can be performed at each iteration.

Scheinberg [77] introduced an efficiently implemented active set method for SVM training based upon a primal-dual active set method given by Nocedal [66]. Since the working set (set of non-bound support vectors) increases or decreases by one at each iteration, a Cholesky factorization can be maintained with rank-one updates. A new speed-up method referred to as *sprinting* is introduced where pricing is performed on a subset of the most negatively priced variables until they are optimized and new subsets chosen. With the combination of *kernel caching*, *sprinting*, and *shrinking*, the author reports competitive performance with SVMLight. The SVM-QP algorithm method handles semi-definite kernels by explicitly detecting rank deficiencies during the course of updating the Cholesky factorization and employing modified Cholesky factorizations in rank deficient cases.

Overall, while good success has been reported with the active set method, the method still suffers from problems with memory consumption as well as a large number of iterations when contending with problems having a significant fraction of support vectors. All implementations of the active set method, thus far, have a $O\left(m^2\right)$ memory consumption requirement where m is the number of

non-bound support vectors. In addition, since a single variable is entered into the working set at a time, the minimum number of iterations is dictated by the number of support vectors and is usually several factors of the number of support vectors.

Regularization Path Following Algorithms

Hastie *et al.* [38] introduced an active set method that finds the SVM solution for the entire regularization path. The regularization path following algorithm represents a form of active set method. However, in this case, the algorithm starts with an optimal solution for some small value of the regularization parameter C and increases C while adjusting the active set as necessary to maintain optimality. The solution path is piece-wise linear in C with breakpoints coinciding with those values of C where the active set is adjusted. The entire regularization path can be computed in not much more time than it takes a conventional algorithm, such as SMO, to find a solution at a single value of the regularization parameter. Hastie *et al.*, for example, report their implementation solves the entire regularization path in a time which is 50 percent more than the time it takes LIBSVM to solve one model. In [59], the regularization path is extended to the ν-SVM and ν-SVR algorithms and an efficient, incremental LOO (leave-one-out) error validation method is included for finding the optimal regularization parameter. In [72], the method is extended to the one-class SVM.

The method by Hastie *et al.* requires a positive definite kernel matrix and relies on an external method for initialization when class sizes are unequal. Ong *et al.* [67] introduced an improved regularization path-following algorithm that extends the method to semi-definite kernels. Their method explicitly detects when a singular system occurs and solves for the null space. A linear programming problem is solved using an external solver to compute the next breakpoint in the path, and the SMO algorithm is used for initialization. More recently Dai *et al.*, [16] independently discovered singular matrices can be avoided by performing single active set transitions. Their

51

method relies on randomized diagonal loading of the kernel matrix to prevent the occurrence of multiple transition events at a breakpoint in the regularization path and enforce the necessary single transitions. In separate works, Dai *et al.* [17] introduce a path-following algorithm for finding the initial solution in the case of unequal class sizes.

Typically, the SVM solution is evaluated using cross-validation or a test data set at each of the breakpoints. However, the number of breakpoints is on the order of the data set size and usually 4 to 5 times the number of data points [38]. It is reasonable to expect that classifier performance may not vary significantly between every breakpoint. Karasuyama *et al.* [44] and Giesen *et al.* [30] introduced methods for finding the approximate regularization path. Specifically, Karasuyama *et al.* [44] introduce a sub-optimal path-following algorithm, which results in significant speedups by relaxing the KKT conditions when identifying successive breakpoints. They employ a tie-breaking mechanism to prevent cycling in the face of degeneracies that are introduced by the approximate path algorithm. Giesen *et al.* provide an approximate path algorithm for the ℓ_2-loss SVM. They point out their method, compared to traditional grid search methods used with conventional SVM training algorithms, is beneficial when using cross-validation to find the optimal regularization parameter due to the guarantees provided on relative accuracy between breakpoints. The *approximate* methods also typically require a positive definite Hessian (or positive definite kernel matrix) with the exception of the method by Giesen *et al.*, which solves the ℓ_2-loss SVM. Adequate methods do not yet seem to exist for solving the *approximate* path for the ℓ_1-loss SVM with a semi-definite kernel.

As an alternative to path following algorithms, a warm start or incremental strategy can be implemented with the traditional active set method to speed-up the grid search. For example, DeCoste *et al.* [21] show significant improvements in grid search training times with a modified version of the LIBSVM and SVMLight algorithms that allow warm starts. Similar warm start implementations are suggested in [84], [87], and [29] for active set methods. Rather than finding the sequence of

regularization values, these methods must be given a set of values to execute. A criticism against this approach is that the solution behavior between the arbitrarily selected regularization parameter values is unknown [30], [44]. Therefore, no guarantees of optimality can be given when a grid search is performed.

Other Approaches

In Tsang *et al.* [89], an approach based upon a minimum enclosing ball (MEB) is introduced for SVM training. They show that many of the SVM formulations including the ℓ_2-loss SVM and ℓ_1- SVM classifiers, and ℓ_2-loss SVM one-class classifiers can be posed as a MEB problem. In general, finding a minimum enclosing ball is computationally difficult, however, this can be ameliorated with an iterative method that incrementally identifies the furthest data point from the current MEB at each iteration, adds that data point to the problem, and solves for the new MEB until no new points are found. To further facilitate speed-up, they employ random sampling to reduce the computational complexity involved in finding the furthest data point from the current MEB using a result from [85] indicating that a relatively small random sample size can be chosen such that the furthest data point in the random sample is among 5% of the furthest data points in the entire sample space with 95% probability. Interestingly, the algorithm employs the SMO algorithm to solve the QP problem associated with each MEB problem. Similar to the SVMPerf and Pegasos methods, this method is shown to have an asymptotic convergence of $O\left(\frac{1}{\epsilon}\right)$ in terms of the number of iterations.

The DirectSVM [74] algorithm, instead of trying to solve the SVM QP problem directly, takes a more geometric approach. Their algorithm starts by identifying the nearest neighbor points between opposite classes and forms a hyperplane as the perpendicular bisector of the segment between the two points. The worst violator, in terms of being on the wrong side of the hyperplane,

is chosen, and the hyperplane is rotated according to a heuristic derived by the authors. The computational complexity per step of this algorithm is driven by the need to subtract all previous rotations as part of the updated heuristic. The methodology supports the application of kernel methods, and the authors claim classification performance on par with that achieved when solving the SVM QP problem directly while providing much faster training times.

The Smooth SVM (SSVM), introduced by Lee *et al.* [54], converts the ℓ_2-loss SVM problem into an equivalent unconstrained optimization problem. A Newton-like method is used to solve the optimization problem after applying a sigmoid smoothing function to create a twice differentiable objective function. Lee *et al.*, later, extend the SSVM idea to very large datasets (Reduced SVM (RSVM) [53]) by implementing a random sampling methodology.

Finally, Wen *et al.* [99] introduce a fast projected conjugate gradient. They build a series of sub-problems using a projection operator based upon the current working set and apply a conjugate gradient method for a fixed number of iterations. They include a heuristic for selecting variables to optimize that is based upon starting with an initial random selection and adding points that exhibit the highest amount of error relative to the current separating hyperplane. They claim good performance when compared to the SVMFu [73] algorithm even though their implementation is still in MATLAB. The SVMFu algorithm is a publicly available implementation that incorporates features from both SVMLight and SMO implementations. Note that the fast projected conjugate gradient method maintains many similarities to the class of active set methods.

CHAPTER 5: REVISED SIMPLEX METHOD FOR SEMI-DEFINITE KERNELS

The SVM quadratic programming problem is not strictly convex if a semi-definite kernel is employed, such as the linear kernel, or if a positive definite kernel is used and there are duplicate data points. The latter case can be resolved by finding and removing duplicate data points; however, this may not always be feasible or practical. The active set method, then, can encounter singular systems of equations and must be prepared to handle this. Rusin [76] introduced a revised simplex method for quadratic programming where the Hessian is semi-definite and singular matrices are automatically avoided. The revised simplex method for quadratic programming is a variant of the simplex method for linear programming [20]. The key difference from the linear analogue is that the optimal solution is no longer guaranteed to be located on a vertex of the convex polytope formed from the set of inequality constraints. The following describes an efficient implementation of the revised simplex method for quadratic programming with semi-definite kernels introduced by Rusin.

Revised Simplex

The revised simplex method for quadratic programming, the details of which can be found in [76] and reviewed here, solves the following general form

$$\min_{\mathbf{x} \geq 0} \mathbf{p}^T \mathbf{x} - \frac{1}{2} \mathbf{x}^T H \mathbf{x} \tag{5.1}$$

$$\text{subject to } A\mathbf{x} = \mathbf{b}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{p} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $\mathbf{b} \in \mathbb{R}^m$. We can find the conditions for optimality starting with deriving the Lagrangian as

$$L\left(\mathbf{x}, \pi\right) = \mathbf{p}^T \mathbf{x} - \frac{1}{2}\mathbf{x}^T H \mathbf{x} - \boldsymbol{\pi}^T \left(A\mathbf{x} - \mathbf{b}\right) \tag{5.2}$$

where, $\boldsymbol{\pi}$ is $\mathbb{R}^m$ and represents the set of Lagrange multipliers applied to each constraint. Taking the derivative, $\frac{\partial L}{\partial \mathbf{x}}$, the reduced cost becomes

$$\boldsymbol{\delta} \triangleq \mathbf{p} - H\mathbf{x} - A\boldsymbol{\pi} \tag{5.3}$$

and the First Order Necessary Conditions (FONC) (or KKT conditions) are

$$\mathbf{Ax} = \mathbf{b} \tag{5.4}$$

$$\mathbf{x} \geq \mathbf{0} \tag{5.5}$$

$$\boldsymbol{\delta} \geq \mathbf{0} \tag{5.6}$$

$$\boldsymbol{\delta} \cdot \mathbf{x} = 0 \tag{5.7}$$

where the first two conditions are referred to as the feasibility conditions, the last condition is the complementary slackness condition, and the third condition is the stationarity condition. Note that these conditions are also *sufficient* for optimality if the problem is convex, or, in this case, H is not indefinite. Similar to the simplex method for linear programming, the variable $\mathbf{x}$ is partitioned into the set of basic variables $\mathbf{x}_i > 0$ (inactive constraints) and non-basic variables $\mathbf{x}_i = 0$ (active constraints). Using the index set $\mathcal{B}$ to denote the set of basic variables, the optimality conditions in

terms of those variables becomes,

$$\begin{pmatrix} H_{\mathcal{B}} & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}_{\mathcal{B}} \\ \boldsymbol{\pi} \end{pmatrix} = \begin{pmatrix} \mathbf{p}_{\mathcal{B}} \\ \mathbf{b} \end{pmatrix}, \mathbf{x}_{\mathcal{B}} > 0 \tag{5.8}$$

where $\mathbf{B}$ is comprised of the columns of $\mathbf{A}$ corresponding to basic variables $\mathbf{x}_{\mathcal{B}}$, and $\mathbf{p}_{\mathcal{B}}$ and $H_{\mathcal{B}}$ are the corresponding portions of $\mathbf{p}$ and H corresponding to the basic variables. The essence of the revised simplex method is as follows. First, an initial basic feasible solution is found. This solution satisfies the feasibility as well as complementary conditions while not being optimal since at least one component of $\boldsymbol{\delta}$ is less than zero. From this point, a variable not in the basic set is chosen such that $\delta_i < 0$, and the corresponding variable x_i is increased. The change in $\mathbf{x}_{\mathcal{B}}$ and $\boldsymbol{\pi}$, as x_i is increased, can be found as follows

$$\begin{pmatrix} H_{\mathcal{B}} & \mathbf{B}^T \\ \mathbf{B} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{q_i} \\ \mathbf{a_i} \end{pmatrix} \tag{5.9}$$

where i is the index for the newly added variable, x_i, $\mathbf{q}_i$ is the column from H indexed by i and rows indexed by $\mathcal{B}$, $\mathbf{a_i}$ is the column of A indexed by i, and the quantities $\mathbf{h}$, g represent the descent direction for $\mathbf{x}_{\mathcal{B}}$ and $\boldsymbol{\pi}$, respectively. The matrix in Equation 5.9 is referred to as the basis matrix (also referred to as the KKT matrix). The descent direction is followed according to $\mathbf{x}_{\mathcal{B}} \leftarrow \mathbf{x}_{\mathcal{B}} - \theta \mathbf{h}$, $\boldsymbol{\pi} \leftarrow \boldsymbol{\pi} - \theta g$ where $\theta > 0$. The variable x_i is increased until either a basic variable is no longer feasible or $\delta_i = 0$ and the complementary condition is satisfied for the newly added variable. The following expression results from using Equation (5.3) to find δ_i as a function of $\mathbf{h}$ and g

$$\gamma = q_{ii} - \mathbf{a_i}^T \mathbf{g} - \mathbf{q_i}^T \mathbf{h} \tag{5.10}$$

where δ_i is now updated according to $\delta_i \leftarrow \delta_i - \theta\gamma$. The complementary condition, then, is reached when $\theta = \frac{\delta_i}{\gamma}$. Therefore, the minimum value for θ is found within the interval $\left[0, \frac{\delta_i}{\gamma}\right]$ where either some variable in $\mathcal{B}$ becomes infeasible or the basic feasible solution is reached,

$$\theta = \min_{j \in \mathcal{B}} \left[\frac{x_j}{h_j} \middle| h_j > 0, \frac{\delta_i}{\gamma} \middle| \gamma < 0 \right] \tag{5.11}$$

If all $h_j \leq 0$ and simultaneously $\gamma \geq 0$, then the solution is unbounded and progress halts. Otherwise, if a basic variable becomes infeasible, then that variable is removed from $\mathcal{B}$ and the index, i, is added to $\mathcal{B}$ and progress is continued along a new search path where the complementary condition is maintained for all variables within $\mathcal{B}$ except i. The new descent direction is computed as

$$\begin{pmatrix} H_{\mathcal{B}} & \mathbf{B}^T \\ \mathbf{B} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{e_i} \\ 0 \end{pmatrix} \tag{5.12}$$

$$\gamma = -1, \tag{5.13}$$

where the coordinate vector $\mathbf{e}_i$ is a unit vector with a 1 in the location corresponding to the newly added basic variable x_i. Note that $\gamma = -1$ after adding the new variable to $\mathcal{B}$. The search path continues to be followed, removing basic variables that become infeasible, until the complementary condition is satisfied for the newly added variable or $\theta = \frac{\delta_i}{\gamma}$ or $\theta = -\delta_i$. Once achieving a new basic feasible solution, a new point is identified with $\delta_i < 0$ to be added to the basic set. If $\delta_i > 0$ for all i, the algorithm terminates with an optimal solution.

Guarantee of Non-singularity

The revised simplex method introduced by Rusin is unique from the typical active set method for quadratic programming in that the set of equations solved at each iteration, or the basis matrix, is guaranteed to be non-singular. The following theorems, introduced in [76] and reiterated here for convenience, depict how non-singularity is maintained. The following shows that if the basis matrix is non-singular, then adding a new variable will result in a non-singular basis matrix

Theorem 2 ([76]). *Given a non-singular basis matrix*

$$M = \begin{pmatrix} H_{\mathcal{B}} & B^T \\ B & 0 \end{pmatrix} \tag{5.14}$$

and $\mathbf{h}$, $\mathbf{g}$ *computed according to*

$$\begin{pmatrix} H_{\mathcal{B}} & \mathbf{B}^T \\ \mathbf{B} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{q_i} \\ \mathbf{a_i} \end{pmatrix} \tag{5.15}$$

and $\gamma = q_{ii} - \mathbf{a_i}^T \mathbf{g} - \mathbf{q_i}^T \mathbf{h}$, *if* $\gamma \neq 0$ *then the matrix*

$$\bar{M} = \begin{pmatrix} H_{\mathcal{B}} & \mathbf{q_i} & B^T \\ \mathbf{q_i}^T & q_{ii} & \mathbf{a_i} \\ B & \mathbf{a_i}^T & 0 \end{pmatrix} \tag{5.16}$$

is non-singular.

The following shows that the basis matrix will remain nonsingular in the scenario where at least one variable has already been removed from the basic set and an additional variable is removed.

59

Theorem 3 ([76]). *Given the nonsingular matrix*

$$
M = \begin{pmatrix} H_\mathcal{B} & \mathbf{q_r} & B^T \\ \mathbf{q_r}^T & q_{rr} & \mathbf{a_r} \\ B & \mathbf{a_r}^T & 0 \end{pmatrix},
$$

(5.17)

if $\mathbf{g}$, $\mathbf{h}$, *and* h_r *are computed as*

$$
\begin{pmatrix} H_\mathcal{B} & \mathbf{q_r} & B^T \\ \mathbf{q_r}^T & q_{rr} & \mathbf{a_r} \\ B & \mathbf{a_r}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ h_r \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{q_i} \\ \mathbf{a_i} \\ q_{ii} \end{pmatrix}
$$

(5.18)

and $h_r \neq 0$, H *is negative semi-definite, then*

$$
\bar{M} = \begin{pmatrix} H_\mathcal{B} & \mathbf{q_i} & B^T \\ \mathbf{q_i}^T & q_{ii} & \mathbf{a_i} \\ B & \mathbf{a_i}^T & 0 \end{pmatrix}
$$

(5.19)

is nonsingular.

The final case to consider is when an infeasible variable is removed prior to adding the new variable to the basic set essentially representing a variable swap.

Theorem 4 ([76]). *Given the nonsingular matrix*

$$
M = \begin{pmatrix} H_\mathcal{B} & \mathbf{q_i} & B^T \\ \mathbf{q_i}^T & q_{ii} & \mathbf{a_i} \\ B & \mathbf{a_i}^T & 0 \end{pmatrix},
$$

(5.20)

if **g**, **h** *are computed as*

$$
\begin{pmatrix} H_{\mathcal{B}} & \mathbf{q_i} & B^T \\ \mathbf{q_i}^T & q_{ii} & \mathbf{a_i} \\ B & \mathbf{a_i}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ h_r \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{e_i} \\ 0 \\ 0 \end{pmatrix}
\tag{5.21}
$$

and $h_r \neq 0$, H is negative semi-definite, then the basis matrix

$$
\bar{M} = \begin{pmatrix} H_{\mathcal{B}} & B^T \\ B & 0 \end{pmatrix}
\tag{5.22}
$$

A few key observations can be made. First, the variable identified to enter the basic set is not added to the basis matrix unless either a variable in the basic set is first removed or adding the new variable immediately results in a basic feasible solution that satisfies the complementary condition (or $\theta = \frac{\delta_i}{\gamma}$). Secondly, the complementary conditions for the set of variables already in the basic set are maintained at all times. In particular, upon removing a basic variable due to in-feasibility, Equation 5.12 ensures the complementary condition continues to be satisfied for data points in the basic set while driving the pricing variable δ_i for the newly added variable to zero.

In contrast, other active set implementations often immediately adjust the basis matrix (or KKT matrix) for the newly identified basic variable before removing other variables-unnecessarily creating a singular matrix. In addition, these methods do not necessarily ensure the complementary conditions are satisfied until all variables have been removed from the basic set and a new feasible solution is found. In some sense, the method reported here can be viewed as a simple modification to existing active set methods that avoids unnecessarily creating a singular basis (or KKT) matrix.

Solving SVM with the Revised Simplex Method

Recall the ℓ_1-loss SVM dual formulation

$$\min \quad \tfrac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha} \tag{5.23}$$

$$\text{s.t.} \quad \mathbf{y}^T \boldsymbol{\alpha} = 0$$

$$0 \le \alpha_i \le C \;\forall i$$

where $\boldsymbol{\alpha}, \mathbf{y} \in \mathbb{R}^{n \times 1}$ and $Q \in \mathbb{R}^{n \times n}$ when there are n data points. This can be transformed to the form specified for the revised simplex formulation with the following substitutions.

$$\mathbf{x} \triangleq \begin{pmatrix} \boldsymbol{\alpha} \\ \mathbf{r} \end{pmatrix} \tag{5.24}$$

$$\mathbf{p} \triangleq \begin{pmatrix} -\mathbf{1} \\ \mathbf{0} \end{pmatrix} \tag{5.25}$$

$$H \triangleq \begin{pmatrix} -Q & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{5.26}$$

$$\mathbf{b} \triangleq \begin{pmatrix} \mathbf{0} \\ C\mathbf{1} \end{pmatrix} \tag{5.27}$$

$$A \triangleq \begin{pmatrix} \mathbf{y}^T & \mathbf{0} \\ I & I \end{pmatrix} \tag{5.28}$$

where the slack variables $\mathbf{r}$ and corresponding equality constraint $\mathbf{r} + \boldsymbol{\alpha} = C\mathbf{1}$ is introduced to convert the SVM box constraints into the form required by the revised simplex method, $\mathbf{x}, \mathbf{p}, \mathbf{b}, \in \mathbb{R}^{2n \times 1}$, and $H, A \in \mathbb{R}^{2n \times 2n}$. A partition is formed from the set of non-support vectors, $\alpha_i = 0$,

$r_i = C$, non-bound support vectors $0 < \alpha_i, r_i < C$, and bound support vectors $\alpha_i = C$, $r_i = 0$. From this point forward, define the index sets $\mathcal{I}_o$, $\mathcal{I}_s$, and $\mathcal{I}_c$ to contain the indices for the set of non-support vectors, non-bound support vectors, and bound support vectors, respectively, and the subscript notation o, s, and c to refer to the corresponding components of a vector or matrix.

With the appropriate substitutions, the constraint matrix B becomes

$$B = \begin{pmatrix} \mathbf{y_s}^T & \mathbf{y_c}^T & \mathbf{0} & \mathbf{0} \\ I_s & I_c & I_o & I_s \end{pmatrix} \tag{5.29}$$

and the set of basic variables becomes

$$\mathbf{x}_{\mathcal{B}} \triangleq \begin{pmatrix} \boldsymbol{\alpha}_s \\ \boldsymbol{\alpha}_c \\ \mathbf{r_o} \\ \mathbf{r_s} \end{pmatrix} \tag{5.30}$$

The matrices I_o, I_s, I_c contain the columns of the identity matrix corresponding to the non-support vectors, non-bound support vectors, and bound support vectors, respectively. The vectors $\mathbf{h}_s$, $\mathbf{h}_c$, $\mathbf{u}_o$, $\mathbf{u}_r$ to refer to the descent directions associated with $\boldsymbol{\alpha}_s$, $\boldsymbol{\alpha}_c$, $\mathbf{r_o}$, and $\mathbf{r_s}$, respectively. Upon

adding a variable α_i to the basic set, the descent direction is computed as follows,

$$
\begin{pmatrix}
-Q_{ss} & -Q_{sc} & \cdots & \cdots & \cdots & \mathbf{y_s} & \mathbf{I_s}^T \\
-Q_{cs} & -Q_{cc} & \cdots & \cdots & \cdots & \mathbf{y_c} & \mathbf{I_c}^T \\
\vdots & \vdots & & & & \vdots & \mathbf{I_\hat{o}}^T \\
\vdots & \vdots & & \ddots & & \vdots & \mathbf{I_s}^T \\
\vdots & \vdots & & & & \vdots & \mathbf{e_i}^T \\
\mathbf{y_s}^T & \mathbf{y_c}^T & \cdots & \cdots & \cdots & 0 & \vdots \\
\mathbf{I_s} & \mathbf{I_c} & \mathbf{I_\hat{o}} & \mathbf{I_s} & \mathbf{e_i} & \cdots & 0
\end{pmatrix}
\begin{pmatrix}
\mathbf{h_s} \\
\mathbf{h_c} \\
\mathbf{u_\hat{o}} \\
\mathbf{u_s} \\
u_i \\
g_\beta \\
\mathbf{g}
\end{pmatrix}
=
\begin{pmatrix}
-\mathbf{q_{si}} \\
-\mathbf{q_{ci}} \\
\mathbf{0} \\
0 \\
0 \\
\mathbf{y_i} \\
\mathbf{e_i}
\end{pmatrix}
\tag{5.31}
$$

where g_β is the Lagrange multiplier associated with the constraint $\mathbf{y}^T\boldsymbol{\alpha} = 0$ and $\mathbf{g} \in \mathbb{R}^n$ corresponds to the remaining equality constraints $\alpha_i + r_i = C$. Note that u_i is the descent for r_i, which is already in the basic set and is equal to C prior to adding α_i to the basic set. The index set $\mathcal{I}_\hat{o}$ contains the indices for the non-support vectors minus the index i or $\mathcal{I}_\hat{o} = \mathcal{I}_o \setminus i$. Immediately, $\mathbf{u_s} = -\mathbf{h_s}$, $\mathbf{g_s} = \mathbf{g_o} = \mathbf{0}$, $u_i = 1$, $\mathbf{h_c} = \mathbf{0}$, and $\mathbf{u_o} = \mathbf{0}$. The following is then obtained for $\mathbf{h_s}$, g_β

$$
\begin{pmatrix}
-\mathbf{Q_{ss}} & \mathbf{y_s} \\
\mathbf{y_s}^T & 0
\end{pmatrix}
\begin{pmatrix}
\mathbf{h_s} \\
g_\beta
\end{pmatrix}
=
\begin{pmatrix}
-\mathbf{q_{si}} \\
y_i
\end{pmatrix}
\tag{5.32}
$$

The expression for $\mathbf{g_c}$ in terms of $\mathbf{h_s}$ and g_β becomes

$$
\mathbf{g}_c = q_{ci} + Q_{cs}\mathbf{h_s} - \mathbf{y_c}g_\beta
\tag{5.33}
$$

Similarly, the following is employed to compute the descent direction upon choosing a variable

$r_i = 0$ (corresponding to $\alpha_i = C$) to enter the basic set.

$$
\begin{pmatrix}
-Q_{ss} & -Q_{s\hat{c}} & -\mathbf{q_{si}} & \cdots & \cdots & \mathbf{y_s} & I_s{}^T \\
-Q_{\hat{c}s} & -Q_{\hat{c}\hat{c}} & -\mathbf{q_{\hat{c}i}} & & & \mathbf{y_{\hat{c}}} & I_{\hat{c}}{}^T \\
-\mathbf{q_{si}}{}^T & -\mathbf{q}_{\hat{c}i}^T & -q_{ii} & \cdots & \cdots & y_i & \mathbf{e_i}{}^T \\
\vdots & & \vdots & \ddots & & \vdots & I_o{}^T \\
\vdots & & \vdots & & & \vdots & I_s{}^T \\
\mathbf{y_s}{}^T & \mathbf{y}_{\hat{c}}^T & y_i & \cdots & \cdots & 0 & \vdots \\
I_s & I_{\hat{c}} & \mathbf{e_i} & I_o & I_s & \cdots & 0
\end{pmatrix}
\begin{pmatrix}
\mathbf{h_s} \\
\mathbf{h_{\hat{c}}} \\
h_i \\
\mathbf{u_o} \\
\mathbf{u_s} \\
g_\beta \\
\mathbf{g}
\end{pmatrix}
=
\begin{pmatrix}
\mathbf{0} \\
\mathbf{0} \\
0 \\
\mathbf{0} \\
0 \\
\mathbf{0} \\
\mathbf{e_i}
\end{pmatrix}
\tag{5.34}
$$

Note that, in this case, h_i corresponds to the descent direction for α_i and $\mathcal{I}_{\hat{c}} = \mathcal{I}_c \setminus i$. Again, this simplifies to

$$
\begin{pmatrix}
-\mathbf{Q_{ss}} & \mathbf{y_s} \\
\mathbf{y_s}{}^T & 0
\end{pmatrix}
\begin{pmatrix}
\mathbf{h_s} \\
g_\beta
\end{pmatrix}
=
\begin{pmatrix}
\mathbf{q_{si}} \\
-y_i
\end{pmatrix}
\tag{5.35}
$$

Combining these results into a simple expression for each case results in

$$
\begin{pmatrix}
-\mathbf{Q_{ss}} & \mathbf{y_s} \\
\mathbf{y_s}{}^T & 0
\end{pmatrix}
\begin{pmatrix}
\mathbf{h_s} \\
g_\beta
\end{pmatrix}
= h_i
\begin{pmatrix}
\mathbf{q_{si}} \\
-y_i
\end{pmatrix}
\tag{5.36}
$$

where

$$
h_i =
\begin{cases}
-1, & \alpha_i = 0 \\
1, & \alpha_i = C
\end{cases}
\tag{5.37}
$$

Given this, the expression for γ is easily found as

$$
\gamma = -q_{ii} + h_i \left(y_i g - \mathbf{q}_{si}^T \mathbf{h} \right)
\tag{5.38}
$$

65

Following the point where an infeasible variable is removed from the basic set, the new descent direction is computed as follows.

$$
\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ g \end{pmatrix} = h_i \begin{pmatrix} \mathbf{e_i} \\ 0 \end{pmatrix}
\tag{5.39}
$$

where the index set $\mathcal{I}_s$ now includes i and $\gamma = -1$.

Initial Basic Feasible Solution

For the SVM problem, the basic initial feasible solution is bound by setting $\alpha_i = 0$, (equivalently, $r_i = C$) for all i and arbitrarily choose the 1$^{\text{st}}$ component of $\boldsymbol{\alpha}$ or α_1 to be in the basic set. Therefore the initial solution using Equation 5.8 becomes

$$
\begin{pmatrix} q_{11} & \cdots & y_1 & \mathbf{e}_1^T \\ \vdots & \ddots & \vdots & I \\ y_1 & \cdots & 0 & \cdots \\ \mathbf{e}_1^T & I & \vdots & \ddots \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \mathbf{r} \\ \beta \\ \boldsymbol{\pi} \end{pmatrix} = \begin{pmatrix} -1 \\ \mathbf{0} \\ 0 \\ \mathbf{1}C \end{pmatrix}
\tag{5.40}
$$

which results in

$$
\alpha_i = 0 \quad \forall i
\tag{5.41}
$$

$$
r_i = C \quad \forall i
\tag{5.42}
$$

$$
\beta = -y_1
\tag{5.43}
$$

Since the slack variables are dealt with implicitly, the following is obtained

$$\alpha_i = 0 \ \forall i \tag{5.44}$$

$$\beta = -y_1 \tag{5.45}$$

$$\mathcal{I}_o = \{2, \cdots, N\}, \mathcal{I}_s = \{1\}, \mathcal{I}_c = \emptyset \tag{5.46}$$

Pricing

At optimality, $\delta_i \geq 0$ for all i with $\delta_i = 0$ when $x_i > 0$ to satisfy the complementary condition. In the case $\alpha_i = 0$ (and $r_i = C$), it is easy to show that the reduced pricing δ_i becomes,

$$\delta_i = -1 - \beta y_i - \pi_i - q_{is}\alpha_s - q_{ic}\alpha_c \tag{5.47}$$

and for the case $\alpha_i = C$ ($r_i = 0$)

$$\delta_i = -\pi_i \tag{5.48}$$

After using Equation (5.8) to solve for π, the following is obtained

$$
\begin{pmatrix}
-Q_{ss} & -Q_{sc} & \cdots & \cdots & \mathbf{y_s} & \mathbf{I_s}^T \\
-Q_{cs} & -Q_{cc} & & & \mathbf{y_c} & \mathbf{I_c}^T \\
\vdots & & \ddots & & & \mathbf{I_o}^T \\
\vdots & & & & & \mathbf{I_s}^T \\
\mathbf{y_s}^T & \mathbf{y_c}^T & & & \ddots & \vdots \\
\mathbf{I_s} & \mathbf{I_c} & \mathbf{I_o} & \mathbf{I_s} & \cdots & 0
\end{pmatrix}
\begin{pmatrix}
\alpha_s \\
\alpha_c \\
\mathbf{r_o} \\
\mathbf{r_s} \\
\beta \\
\pi
\end{pmatrix}
=
\begin{pmatrix}
-1 \\
-1 \\
0 \\
0 \\
0 \\
C\mathbf{1}
\end{pmatrix}
\tag{5.49}
$$

67

from which immediately $\boldsymbol{\pi}_o = \mathbf{0}, \boldsymbol{\pi}_s = \mathbf{0}$ and, as a result,

$$\boldsymbol{\pi}_c = -\mathbf{1} + Q_{cs}\boldsymbol{\alpha}_s + Q_{cc}\boldsymbol{\alpha}_c - \mathbf{y}_c\beta, \tag{5.50}$$

and for the case $\alpha_i = 0$ or $i \in \mathcal{I}_o$ since $\boldsymbol{\pi}_o = \mathbf{0}$,

$$\boldsymbol{\delta}_o = -\mathbf{1} + Q_{os}\boldsymbol{\alpha}_s + Q_{oc}\boldsymbol{\alpha}_c - \beta\mathbf{y}_o \tag{5.51}$$

and the following is obtained for $\alpha_i = C$

$$\boldsymbol{\delta}_c = \mathbf{1} - Q_{cs}\boldsymbol{\alpha}_s - Q_{cc}\boldsymbol{\alpha}_c + \beta\mathbf{y}_c. \tag{5.52}$$

The pricing variables can be incrementally updated noting that $\Delta\boldsymbol{\delta}_c = -Q_{cs}\Delta\boldsymbol{\alpha}_s + \Delta\beta\mathbf{y}_c$ and $\Delta\boldsymbol{\delta}_o = Q_{os}\Delta\boldsymbol{\alpha}_s + \Delta\beta\mathbf{y}_o$. This avoids recomputing $Q_{oc}\boldsymbol{\alpha}_c$ or $Q_{cc}\boldsymbol{\alpha}_c$ at each iteration. However, as an alternative, these latter quantities can be directly stored and incrementally updated as points enter and leave the set of bound support vectors, $\mathcal{I}_c$, as is done in this implementation.

The pricing step can be a significant fraction of the computation time per iteration with a computational complexity of $O((N_o + N_c) \times N_s)$ where N_o is the number of non-support vectors, N_c is the number of bound support vectors and N_s is the number of non-bound support vectors. A straight-forward methodology for speeding up the pricing step is to reduce $N_o + N_c$ or the number of variables for which pricing is computed. The strategy employed in SVMLight referred to as *shrinking* [41] is equally applicable here. The *shrinking* strategy attempts to find variables (or data points) that are not likely to change as progress towards the solution is made. For example, identifying bound support vectors or non-support vectors that are likely to remain as such at the solution. Another method *shrinking* [77] is analogous to a partial pricing scheme used in the simplex method for linear programming. This method performs pricing on a small subset of the most

negatively priced variables until all of those become positive. At that point, a new set of most negatively priced variables is chosen, and the process continues. The authors, Scheinberg *et al.* show this method outperformed the *shrinking* strategy within the context of the SVM-QP algorithm (an active set implementation). It is also possible to combine *shrinking* and *sprinting*.

Efficient Solution of the Inner Sub-problem

The Cholesky factorization has the advantage of requiring fewer operations and possessing improved numerical stability compared to other factorizations. Further, the Cholesky factorization can be easily updated with rank-one updates. The goal, then, is to apply the Cholesky factorization when solving for the descent direction at each iteration. However, the Cholesky factorization cannot be directly applied to the basis matrix (KKT matrix) since it is indefinite. Fortunately, since the basis matrix is guaranteed to be non-singular, this is easily remedied by using the null-space method [66]. Note that as an alternative, the Cholesky factorization can be applied directly to the quantity Q_{ss} in the basis matrix in a method analogous to that reported in [77]; however, Q_{ss} can be singular even if the basis matrix is non-singular, thus, requiring a modified form of the Cholesky factorization.

Null Space Method for SVM

Following the details of the null space method described in Chapter 3, the constraint matrix becomes the vector $\mathbf{y}^T$ from which the null space Z is easily found to be given by the following orthonormal, sparse, matrix,

$$Z = \begin{pmatrix} [-y_1 y_2, ..., -y_1 y_n] \\ I \end{pmatrix}. \tag{5.53}$$

The matrix Y, such that $[Z|Y]$ becomes full rank, simply becomes $Y = \mathbf{e_1}$ where $\mathbf{e_1}$ is a unit vector with a 1 as the first entry.

Using the null space method, the quantities $\mathbf{h} \triangleq \mathbf{h}_s$ and $g \triangleq g_\beta$ in Equations (5.36), (5.39) are found as follows.

$$-Q_{ss}Z\mathbf{h_z} - Q_{ss}Y\mathbf{h_y} + y_s g = \mathbf{u}$$
$$\mathbf{y_s}^T Y\mathbf{h_y} = v, \tag{5.54}$$

where $\mathbf{h} = Z\mathbf{h_z} + Y\mathbf{h_y}$. Multiplying the first equation by Z^T results in the following

$$-Z^T Q Z\mathbf{h_z} = Z^T \left(\mathbf{u} + QY\mathbf{h_y}\right) \tag{5.55}$$

where $\mathbf{u}$ and v are defined as the right side of Equations (5.36), (5.39). The sequence of steps for solving $\mathbf{h}$, g are, first, to solve $\mathbf{h}_y$ using the second equation in Equation (5.54), followed by solving for $\mathbf{h}_z$ in Equation (5.55), followed by solving for g using the first equation in Equation (5.54). Note that further simplifications are possible using the analytical forms of Z and Y. A Cholesky factorization $R^T R = Z^T G Z$ can be maintained with rank-one updates in an efficient manner as reported in Sentelle *et al.* [80].

Updating the Cholesky Factorization

Upon adding a basic variable, a row and column will be added to Z as well as Q_{ss}. The Cholesky factorization is, then, updated by solving the following for $\mathbf{r}$ and ρ as follows

$$\begin{pmatrix} R^T & 0 \\ \mathbf{r}^T & \rho \end{pmatrix} \begin{pmatrix} R & \mathbf{r} \\ \mathbf{0} & \rho \end{pmatrix} = \tilde{Z}^T \begin{pmatrix} Q & \mathbf{q} \\ \mathbf{q}^T & \sigma \end{pmatrix} \tilde{Z}, \tag{5.56}$$

where

$$\tilde{Z} = \begin{pmatrix} Z & -y_1 y_i \mathbf{e_1} \\ \mathbf{0}^T & 1 \end{pmatrix}, \tag{5.57}$$

$\mathbf{e}_i$ is the unit coordinate vector with a 1 in the ith position, and $\mathbf{q}, \sigma$ represent the row/column from the kernel matrix corresponding to the newly added variable. The null space Z is also augmented where y_i is the newly added label and y_1 is the label corresponding to the first index in the basic set. Expanding (5.56), results in the following

$$\begin{pmatrix} Z^T Q Z & -y_1 y_i Z^T Q \mathbf{e_1} + Z^T \mathbf{q} \\ -y_1 y_i \mathbf{e_1}^T Q Z + \mathbf{q}^T Z & \mathbf{e_1}^T Q \mathbf{e_1} - 2 y_1 y_i \mathbf{e_1}^T \mathbf{q} + \sigma \end{pmatrix} \tag{5.58}$$

Therefore, $\mathbf{r}$ and ρ can be solved as follows.

$$R^T \mathbf{r} = -y_1 y_i Z^T Q \mathbf{e_1} + Z^T \mathbf{q} \tag{5.59}$$

$$\mathbf{r}^T \mathbf{r} + \rho^2 = \mathbf{e_1}^T Q \mathbf{e_1} - 2 y_1 y_i \mathbf{e_1}^T \mathbf{q} + \sigma \tag{5.60}$$

Actual implementation can be made efficient relying on the fact that Z is sparse, $\mathbf{e_1}$ essentially selects a single row/column or element, and R is upper triangular allowing $\mathbf{r}$ to be computed using a simple forward substitution.

Upon removing a variable from the basic set, the Cholesky factorization must be down-dated to contain one less row and column. In the case where the last row/column is removed, the last row/column of R is simply removed. In general, there are (2) cases to consider. The first case occurs when the 1st row/column is removed and the second case when any other variable is removed. In the latter case, a series of permutations are performed to move the variable to be removed to the last row/column of Q and last column of Z as follows

$$P_{i-1}^T Z^T P_i^T P_i Q P_i^T P_i Z P_{i-1} = P_{i-1}^T Z^T Q Z P_{i-1} \tag{5.61}$$

where P_i is used to permute the ith row/column of Q to the last row/column and to permute the ith row of Z to the last row, and P_{i-1} is employed to permute the $i-1$ column of Z to the last column. Since P_i is orthogonal ($P_i^T P_i = I$), the projection P_i can be eliminated, and this allows the permutation, P_{i-1}, to be directly applied to the Cholesky factorization as follows.

$$P_{i-1}^T Z^T Q Z P_{i-1} = P_{i-1}^T R^T R P_{i-1} = H^T H \tag{5.62}$$

The permutation operation transforms the upper triangular matrix R to an upper Hessenberg matrix H. Consider the following example with the upper right triangular matrix $R \in \mathbb{R}^{4 \times 4}$

$$\begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{pmatrix} \tag{5.63}$$

The following permutation operator will move the 2nd column of R to the last

$$P_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.64}$$

which, upon application, results in the upper Hessenberg matrix (upper triangular matrix with additional non-zero entries on the sub-diagonal)

$$RP_2 = H = \begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & 0 \\ 0 & 0 & x & 0 \end{pmatrix} \tag{5.65}$$

A series of Givens rotations [36] are employed to convert the upper Hessenberg back into an upper triangular matrix by zeroing out the sub-diagonal entries. To apply the Givens rotation, the parameters c and s are found such that

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix} \tag{5.66}$$

where $c = cos(\theta)$, $s = sin(\theta)$. These quantities are typically solved with the following expressions $r = \sqrt{a^2 + b^2}$, $c = \frac{a}{r}$ and $s = \frac{-b}{r}$ noting that θ is not directly solved. Note that the Givens rotation is a 2 by 2 matrix; however, the following Givens matrix allows the rotation to be applied to any

two arbitrary elements of a vector $\mathbf{x}$

$$
G\left(i, j, \theta\right) = \begin{pmatrix}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & & \vdots & & \vdots \\
0 & \cdots & c & \cdots & -s & \cdots & 0 \\
\vdots & & \vdots & \ddots & \vdots & & \vdots \\
0 & \cdots & s & \cdots & c & \cdots & 0 \\
\vdots & & \vdots & & \vdots & \ddots & 0 \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 0
\end{pmatrix}
\tag{5.67}
$$

$G\left(i, j, \theta\right) \mathbf{x}$ performs a clockwise rotation of $\mathbf{x}$ by an angle of θ in the i, j plane (i^{th}, j^{th} components are affected). The transform $RG\left(i, j, \theta\right)$ will affect the ith, j^{th} rows of R. In the example given earlier, a series of Givens rotations $G(3, 4, \theta_2)G(2, 3, \theta_1)H$ are performed to zero out the (2) non-zero entries on the sub-diagonal. This will result in the following upper triangular matrix

$$
G(3, 4, \theta_2)G(2, 3, \theta_1)H = \begin{pmatrix}
x & x & x & x \\
0 & x & x & x \\
0 & 0 & x & x \\
0 & 0 & 0 & x
\end{pmatrix}
\tag{5.68}
$$

Note that $G\left(i, j, \theta\right)$ is orthogonal, and, therefore, $H^T G(i, j, \theta)^T G(i, j, \theta)H = H^T H$. After returning H to an upper triangular form, the last row/column is simply removed and the update is complete.

Removing the first variable row/column from the factorization is more involved. Note that the first row of Z is of the form $y_1 \left[y_2, ...y_n\right]$ and a simple permutation cannot be employed to transform

this. However, the permutation P_1 can be applied to move the 1st row/column of Q and 1st row of Z as follows.

$$Z^T P_1^T P_1 Q P_1^T P_1 Z \tag{5.69}$$

Applying P_1 to Z performs the following transformation

$$P_1 Z = \begin{pmatrix} I^{(n-1)\times(n-1)} \\ [-y_1 y_2, \ldots, -y_1 y_n] \end{pmatrix} \tag{5.70}$$

Compare this with the desired result

$$\begin{pmatrix} [-y_2 y_3, \ldots, -y_2 y_n] \\ I^{(n-2)\times(n-2)} \end{pmatrix} \tag{5.71}$$

Therefore, the following transform converts $P_1 Z$ to the desired form as follows

$$A = \begin{pmatrix} [-y_2 y_3, \ldots, -y_2 y_n] \\ I^{(n-2)\times(n-2)} \end{pmatrix} \tag{5.72}$$

The complete transformation is then

$$A^T Z^T P_1^T P_1 Q P_1^T P_1 Z A = A^T Z^T Q Z A = A^T R^T R A = H^T H \tag{5.73}$$

Again, this transform generates an upper Hessenberg matrix, and the matrix is transformed back into an upper triangular matrix with a series of Givens rotations. The last column/row is then removed to complete the update.

Algorithm 1 Revised Simplex method for SVM Training (SVM-RSQP)

1: $\alpha_i \leftarrow 0 \quad \forall i,$
2: $\beta \leftarrow -y_1,$
3: $\mathcal{I}_o \leftarrow \{2...N\}, \mathcal{I}_c \leftarrow \emptyset, \mathcal{I}_s \leftarrow \{1\}$
4: **loop**
5:     Calculate the reduced cost, δ_j using 5.51, 5.52
6:     $i = \arg\min_j \delta_j$
7:     **if** $\delta_i \geq -tol$ **then**
8:         **stop** {optimality achieved}
9:     **end if**
10:     $\mathbf{h_o} \leftarrow \mathbf{0}, \mathbf{h_c} \leftarrow \mathbf{0}$
11:     $d \leftarrow \begin{cases} -1 & i \in \mathcal{I}_o \\ 1 & i \in \mathcal{I}_c \end{cases}$ {Choose descent direction of entering variable according to status}
12:     solve $\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_s} \\ g_\beta \end{pmatrix} = d \begin{pmatrix} \mathbf{q_{si}} \\ -y_i \end{pmatrix}$ {Solve descent}

        $h_i \leftarrow d$ {Descent for newly added variable}

        $\gamma \leftarrow -q_{ii} + d\left(y_i g_\beta - \mathbf{q_{si}^T h_s}\right)$ {Descent for pricing of newly added variable}
13:     $\mathcal{I}_s \leftarrow \mathcal{I}_s \cup \{i\}$ {Add new variable to non-bound set}

        {Optimize until complementary reached for new variable}
14:     **while** $\delta_i < -tol$ **do**
15:         $i_r \leftarrow \arg\min_{j \in \mathcal{I}_s} \left\{ \frac{\alpha_j - C[h_j < 0]}{h_j} \right\}$ {Find max descent}
16:         $\theta \leftarrow \frac{\alpha_{i_r} - C[h_{i_r} < 0]}{h_{i_r}}$
17:         **if** $\gamma \geq 0$ or $\theta < \frac{\delta_i}{\gamma}$ **then**
18:             **if** $\mathbf{h_r} < 0$ **then**
19:                 $\mathcal{I}_c \leftarrow \mathcal{I}_c \cup \{i_r\}$ {Variable became infeasible, add to bound support vector set}
20:             **else**
21:                 $\mathcal{I}_o \leftarrow \mathcal{I}_o \cup \{i_r\}$ {Variable became infeasible, add to non-support vector set}
22:             **end if**
23:         $\mathcal{I}_s \leftarrow \mathcal{I}_s \setminus \{i_r\}, \delta_i \leftarrow \delta_i - \theta\gamma$
24:         $\beta \leftarrow \beta - \theta g_\beta, \alpha \leftarrow \alpha - \theta\mathbf{h}$
25:         **else**
26:             $\theta \leftarrow \frac{\delta_i}{\gamma}, \delta_i \leftarrow 0$ {Complementarity reached for new variable}
27:             $\beta \leftarrow \beta - \theta g_\beta, \alpha \leftarrow \alpha - \theta\mathbf{h}$
28:             **break**
29:         **end if**
30:         solve $\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_s} \\ g_\beta \end{pmatrix} = d \begin{pmatrix} -\mathbf{e_i} \\ 0 \end{pmatrix}$ {Recompute descent}
31:         $\gamma \leftarrow -1$
32:     **end while**
33: **end loop**

Results and Discussion

To assess the revised simplex method for SVM training (denoted as SVM-RSQP), it is compared against the popular training algorithms LIBSVM [12] (based upon the improved SMO algorithm [25]), SVMLight [41], and SVM-QP [77], an efficiently implemented active set method. Testing is done on a sample of small to medium-sized datasets along with a few, selected large datasets.

The SVM-RSQP was implemented in C++ and uses the Blitz++ matrix template library [94]. For convenience and to allow objective comparison without compiler bias, the SVM-QP method, available at the COIN-OR website [60], was converted from a FORTRAN implementation to C++ using the Blitz++ library. The Blitz++ matrix template library provides a complete and fast implementation of matrix objects and operators, and has been shown to perform on par with FORTRAN implementations in benchmarks [94]. The Blitz++ library supports configuration of matrices to use column-major ordering as well as 1-based indexing making the translation of FORTRAN code to equivalent C++ straightforward. All algorithms were compiled as MEX functions and invoked from within the MATLAB environment.

The datasets are listed in Table 5.1. The *adult-1a* and *adult-a* datasets were obtained from [70], *ocr-0* and *ocr-9* are based upon the dataset obtained from the United States Postal Service (USPS) OCR dataset [40], and the remaining datasets were obtain from the UCI repository [2]. The *letter-g*, *ocr-0*, and *ocr-9* are composed as described in [77] where *letter-g* classifies the letter "G" from the remaining letters, *ocr-0* classifies the number "0" from the remaining numbers and *ocr-9* classifies the number "9" from the remaining numbers. For the *Forest Covertype* (*covtype*) dataset, a total of 50,000 data points was randomly drawn from the first 2 types (classes) to create a binary classification problem. All attributes are normalized to scale between 0 and 1.

Table 5.1: Datasets used for SVM-RSQP assessment

Dataset	N	d
adult-1a	1,605	123
adult-a	32,561	123
abalone	4,177	7
letter-g	20,000	16
ocr-0	7,291	256
ocr-9	7,291	256
spam	4,601	57
splice	1,000	60
covtype	100,000	54

Experimental results are listed in Table 5.2, for the linear kernel, and in Table 5.3 for the RBF kernel. Table 5.4 also contains results for large datasets. These results were previously reported in by Sentelle *et al.* in [80]. The total training time, number of iterations, as well as the total number of support vectors (NSV) and number of bound support vectors (NBSV) is shown for each of the algorithms. The RBF kernel is defined as $exp(-\gamma\|x_i - x_j\|_2^2)$ and the values for C as well as γ are chosen for each dataset based upon the best results achieved for 10-fold cross validation. The stopping criterion of 10^{-3} was used in all cases. The SVM-QP algorithm was configured to use *sprinting* as described in [77] with a subset of 50 non-support vectors and bound support vectors chosen for pricing at a time. The SVM-RSQP was also configured to use a form of *sprinting* where 100 data points, not necessarily split evenly between non-support and bound support vectors, are optimized at a time.

In all cases, the experiments reveal that SVM-RSQP is competitive, in terms of training time, with SVM-QP, SVMLight, as well as LIBSVM. SVM-RSQP is, on average, two times faster than SVM-QP for small datasets and nearly two times faster for the larger *adult-a* dataset and approximately

11 percent faster for *covtype*. SVM-RSQP was also competitive with the SVMLight, usually on par with or faster than SVMLight in terms of training time. SVMLight was faster in only two cases, *ocr-0* and *ocr-9* datasets with the linear kernel, where it was faster than all reported algorithms by nearly a factor of 2. On the other hand, SVMLight was found to be typically slower, in some cases by an order of magnitude and did not converge within a reasonable amount of time for the *covtype* dataset.

The overall results show that SVM-RSQP, as anticipated, while not being a orders of magnitude faster than SVM-QP, is at least faster by a constant factor. This is largely due to the computational efficiency gained by SVM-RSQP in solving Equation 5.36. As reported in [80], given N_s non-bound support vectors, SVM-RSQP requires $2N_s^2 + N_s + O(1)$ computations to solve Equation 5.36. SVM-QP must anticipate the possibility of a singular matrix and, therefore, the number of computations is dependent upon the rank of the basis matrix. In the case of full rank, SVM-QP requires $3N_s^2 + N_s + O(1)$ floating point computations. However, when the basis matrix is singular, a modified Cholesky factorization is employed and a particular solution is chosen using the basis for the null space. In this instance, there are $2N_s^2 + O(N_s)$ floating point computations. In addition, pivoting may need to occur to ensure a stable Cholesky factorization in this instance, which incurs an additional $O(N_s^2)$ computations. Therefore, on average, it is anticipated SVM-RSQP requires $1/3^{rd}$ fewer operations to solve Equation 5.36 than SVM-QP. This appears to be consistent with the reported results since the SVM-RSQP algorithm is more than 50 percent faster than SVM-QP in many cases.

A question arises as to whether the SVM-RSQP algorithm is capable of improving on the number of iterations relative to the SVM-QP algorithm given the guarantees on non-singularity. The SVM-QP algorithm must choose a specific descent direction (or solution) out of potentially infinite solutions when the basis matrix is singular. In the worst case, it is possible that a chosen descent direction could result in little or no progress on the solution; however, it seems equally

possible that the chosen descent directions could result in faster convergence. Table 5.5 depicts comparisons between SVM-QP and SVM-RSQP with *sprinting* and *shrinking* disabled for simulated datasets. Two-class datasets are generated from a unit normal distribution with 500 data points in each class. One class is generated as $\mathcal{N}(0,1)$ or as a unit normal distribution centered on the origin while the other class is generated according to $\mathcal{N}(0.1,1)$. The number of features as well as the regularization parameter C is varied. The total training time, KKT solve time (time to solve the equivalent of Equation 5.36), the number of iterations, as well as the number of infinite iterations in the case of SVM-QP (number of cases where the basis matrix is singular) is reported on. In all but one case, the SVM-RSQP algorithm converges with slightly fewer iterations than the SVM-QP algorithm. In the worst case, the SVM-QP algorithm requires an additional 334 iterations for $C = 1000$ and 200 features. This suggests that the SVM-QP algorithm experiences a small loss of efficiency associated with having to compute the null space and select an arbitrary descent direction. Remarkably, the number of infinite iterations does not appear to significantly impact the total number of iterations. These results also show, as predicted, that the SVM-RSQP algorithm is 2 to 3 times faster in all cases when the number of features is greater than or equal to 50. In fact, the for $C = 1000$ with 200 features, the SVM-RSQP algorithm is nearly 2.5 times faster at solving the KKT system, which contributes to an overall 50 percent improvement in total training time.

LIBSVM is a state-of-the-art SMO implementation and still considered to be one of the fastest algorithms. This method is suitable for very large datasets since memory requirements are $O(n)$ where n is the dataset size. The largest comparative study of performance of the conventional active set methods against LIBSVM was done in the work by Sentelle *et al.* [80], the results of which are shown again and discussed here. In the majority of cases, the SVM-RSQP algorithm remains competitive with LIBSVM typically being no more than twice as fast or slower than LIBSVM. The one exception is for the *adult-a* dataset and RBF kernel where LIBSVM was nearly 3 times faster.

On the other hand, SVM-RSQP was approximately 2 times faster on the *covtype* dataset, which is a larger dataset where overall training times are much more significant.

Several factors can affect relative performance between SVM-RSQP and LIBSVM. First, the LIB-SVM algorithm represents a form of block coordinate descent, which can be slow to converge if the problem is poorly scaled. Secondly, the active set method must solve a basis matrix that is of size $O(N_{sv}^2)$ where N_{sv} is the number of non-bound support vectors. For a linear kernel, it can be easily shown that there will no more than $O(d)$ non-bound support vectors, where d is the number of features. Unfortunately, for the non-linear kernel, N_{sv} can vary as a function of the kernel parameters as well as the regularization parameter C. Finally, subtleties associated with implementation of kernel caching, shrinking, sprinting, and other partial pricing mechanisms can affect performance.

Overall, for the conventional active set method the number of iterations will be roughly 2 to 3 times the number of active constraints at the solution [66], or, in this case, the number of support vectors. In experiments, both SVM-QP and SVM-RSQP are observed to perform roughly 2 to 5 iterations per support vector. On the other hand, the SMO algorithm can take many more iterations although each iteration is cheap in terms of computational complexity. For example, LIBSVM takes nearly 20 times as many iterations to converge for the *abalone* dataset and RBF kernel even though it is only 2 times slower.

In conclusion, the proposed revised simplex method for SVM training can be seen as an improvement to existing active set methods that enables use of semi-definite kernels without a need for explicitly detecting singular matrices. This significantly simplifies implementation and allows more options in terms of matrix factorization and storage. Relative to the SVM-QP algorithm, the SVM-RSQP algorithm exhibits slightly improved efficiency in terms of the number of iterations and requires fewer operations to solve the system of equations at each iteration. Overall, the SVM-

RSQP algorithm was found to be nearly two times faster than SVM-QP in some cases. Overall, the SVM-RSQP algorithm represents a successful practical implementation of the active set method for SVM training with semi-definite kernels and appears to be a preferred alternative to existing active set implementations within the SVM community.

Table 5.2: Performance Comparison for the Linear Kernel

Dataset	Algorithm	Training Accuracy	Time (sec)	Iter	NSV	NBSV
adult-1a linear $C = 0.5$	SVM-RSQP	85.73	0.41	2948	602	542
	SVM-QP	85.73	0.80	3019	602	542
	SVMLight	85.79	0.48	1799	602	541
	LIBSVM	85.73	**0.23**	4986	602	540
abalone linear $C = 128$	SVM-RSQP	79.41	**0.97**	11361	2053	2044
	SVM-QP	79.41	2.50	14101	2054	2044
	SVMLight	79.41	50.32	114775	2056	2044
	LIBSVM	79.41	1.36	44433	2054	2044
letter-g linear $C = .0313$	SVM-RSQP	96.14	**2.63**	4800	1553	1542
	SVM-QP	96.14	4.50	4997	1550	1538
	SVMLight	96.14	6.39	685	1623	1510
	LIBSVM	96.14	3.64	804	1569	1527
spam linear $C = 512$	SVM-RSQP	93.39	**1.97**	7837	919	866
	SVM-QP	93.39	5.45	10849	920	866
	SVMLight	93.39	83.36	98128	923	854
	LIBSVM	93.39	3.58	127240	945	856
splice linear $C = 0.5$	SVM-RSQP	83.80	**0.14**	1976	437	382
	SVM-QP	83.80	0.31	2294	437	382
	SVMLight	83.90	0.18	1489	438	381
	LIBSVM	83.90	0.28	5057	437	381
ocr-0 linear $C = .0313$	SVM-RSQP	99.45	2.42	1404	357	281
	SVM-QP	99.45	4.70	1662	360	279
	SVMLight	99.45	**1.24**	480	361	278
	LIBSVM	99.45	3.63	1283	360	278
ocr-9 linear $C = .125$	SVM-RSQP	98.70	3.25	1913	466	375
	SVM-QP	98.70	7.47	2219	466	372
	SVMLight	98.70	**2.26**	1674	469	371
	LIBSVM	98.70	4.44	2514	467	372

Table 5.3: Performance Comparison for the RBF Kernel

Dataset	Algorithm	Training Accuracy	Time	Iter	NSV	NBSV
adult-1a rbf $C = 2, \gamma = .03125$	SVM-RSQP	85.92	0.41	2573	671	573
	SVM-QP	85.92	0.72	2463	671	573
	SVMLight	85.92	0.58	351	672	572
	LIBSVM	85.92	**0.27**	889	671	573
abalone rbf $C = 2048, \gamma = .5$	SVM-RSQP	81.09	**1.81**	11584	1900	1813
	SVM-QP	81.09	4.81	13689	1900	1814
	SVMLight	-[1]	-	-	-	-
	LIBSVM	81.09	4.08	228435	1900	1812
letter-g rbf $C = 8, \gamma = 8$	SVM-RSQP	99.98	5.19	1281	662	54
	SVM-QP	99.98	7.94	1417	662	54
	SVMLight	99.98	9.51	3325	671	54
	LIBSVM	99.98	**2.55**	4633	669	54
spam rbf $C = 2048, \gamma = .125$	SVM-RSQP	95.20	**1.91**	4689	814	663
	SVM-QP	95.20	3.89	5176	814	663
	SVMLight	95.22	24.33	85032	831	659
	LIBSVM	95.22	2.77	77645	819	658
splice rbf $C = 32, \gamma = .03125$	SVM-RSQP	100.00	0.77	721	511	11
	SVM-QP	100.00	1.23	712	510	11
	SVMLight	100.00	0.42	1629	520	10
	LIBSVM	100.00	**0.41**	5144	518	10
ocr-0 rbf $C = 8, \gamma = .03125$	SVM-RSQP	100.00	**4.00**	602	464	0
	SVM-QP	100.00	4.55	649	464	0
	SVMLight	100.00	4.72	459	465	0
	LIBSVM	100.00	4.58	1288	464	0
ocr-9 rbf $C = 32, \gamma = .03125$	SVM-RSQP	100.00	4.64	677	492	0
	SVM-QP	100.00	5.05	703	492	0
	SVMLight	100.00	5.75	1232	493	0
	LIBSVM	100.00	**4.45**	2204	492	0

Table 5.4: Performance Comparison for Large Datasets

Dataset	Algorithm	Training Accuracy	Time	Iter	NSV	NBSV
adult-a linear $C = 2$	SVM-RSQP	85.01	372.39	66457	11480	11387
	SVM-QP	85.00	548.20	87072	11481	11386
	SVMLight	85.01	1048.36	99976	11571	11327
	LIBSVM	85.00	**186.97**	112817	11541	11344
adult-a rbf $C = 32, \gamma = .03125$	SVM-RSQP	87.56	707.81	38544	11499	9062
	SVM-QP	87.56	1101.13	59786	11494	9053
	SVMLight	87.57	6028.00	100224	12008	8770
	LIBSVM	87.55	**255.50**	100274	11760	8864
covtype rbf $C = 8192, \gamma = .125$	SVM-RSQP	85.22	**8471.86**	208978	37926	36846
	SVM-QP	85.23	9580.77	315533	37953	36871
	SVMLight	-	-	-	-	-
	LIBSVM	85.21	16344.67	6214081	38103	36694

Table 5.5: Algorithm Efficiency Comparison for Simulated Gaussian Datasets

Features	C	SVM-QP				SVM-RSQP		
		Time (sec)	KKT (sec)	Iter	Inf Iter	Time (sec)	KKT (sec)	Iter
5	10	.25	.003	3251	985	.23	.004	3301
5	1000	.30	.004	3621	1784	.27	.005	3587
50	10	1.56	.135	4100	88	1.42	.073	3888
50	1000	2.5	.211	5326	2324	1.92	.106	5240
100	10	3.06	.312	3362	32	2.28	.183	3394
100	1000	6.7	.721	5818	2218	5.45	.443	5768
200	10	4.72	1.13	2864	23	3.77	.494	2598
200	1000	12.0	2.52	4934	1102	8.24	1.12	4600

CHAPTER 6: SIMPLE SVM REGULARIZATION PATH FOLLOWING ALGORITHM

The Support Vector Machine (SVM), while possessing excellent generalization performance, still requires tuning of the regularization parameter C, as well as kernel parameters for optimal generalization performance. This usually entails retraining across several different values of these parameters. A popular method for doing this is the "grid search" where retraining occurs across a grid of C values and kernel parameters. Performance is measured at each parameter setting using a separate validation data set or through cross-validation to find the one yielding the best test performance. Unfortunately, this can be a time-consuming process especially when there is limited or no *a priori* knowledge of the valid range of parameters. Hastie *et al.* [38] introduced *SVMPath*, a homotopy method for solving the entire regularization path (solution path as function of C). This method relies on the fact that the SVM training solution is piece-wise linear in terms of C. That is, given a solution to the SVM dual formulation at a particular value of C, the dual variables such that $0 < \alpha_i < C$ vary linearly according to C until either a data point on the margin ($0 < \alpha_i < C$) begins to violate a feasibility constraint ($\alpha_i > C$ or $\alpha_i < 0$) or a new point enters the margin (data point with $\alpha_i = 0$ or $\alpha_i = C$). When either of these conditions occur, the active set must be modified in order to continue, therefore, representing a breakpoint in the piecewise linear path. The regularization path essentially starts at some small value of C and increases the value while identifying breakpoints and adjusting the active set as it continues. Hastie *et al.* reported that the path following algorithm computed the entire path in a time that was faster than the average time required for SMO to compute a solution at a single value of C when performing a "grid search".

While this is an important contribution, the proposed method can only be used with positive definite kernels. This rules out use of the linear kernel as well as those cases where duplicate data points

exist. The existence of duplicate data points is an issue even for the positive definite kernel, and while it is possible to preemptively remove duplicate data points, this may not be practical for large datasets. In other important work, Ong *et al.* [67] introduced an improvement that resolves issues with using the semi-definite kernel as well as duplicate data points. While successful, the method has the following limitations: (i) an SVD or QR factorization method is employed for explicitly detecting rank deficiencies and finding the basis of the null space, (ii) an external solver is employed to solve a posed linear programming problem at each iteration, (iii) the method can become unstable or exhibit cycling, which must be resolved using a backup routine, and (iv) the method incorporates the SMO training algorithm for both initialization and path recovery.

In the course of computing the regularization path, it is possible for more than one active set transition to be identified at each break-point, especially when using a semi-definite kernel or if duplicate data points exist. In the original method reported by Hastie *et al.*, all of the identified transitions are taken. However, further active set transitions may be identified at the same breakpoint after performing the initial transitions. The SVMPath algorithm proceeds with strictly increasing C values, ignoring these additional transitions. This can lead to a breakdown and subsequent computation of an invalid solution path [67]. Furthermore, it is possible for all constraints to become active (or empty margin set), in which case, the algorithm is forced to perform a re-initialization and a valid portion of the path may be skipped.

The Simple SVM Path algorithm, introduced here, works by performing single active set transitions at each iteration and allowing repeat events (C value does not vary between transitions). This method automatically avoids singular systems of equations, the empty margin set case, and the need for an external solver and specialized factorizations. The proposed algorithm also incorporates a new initialization method for unequal class sizes that works within the framework of the existing algorithm and, therefore, does not require an external solver. Detailed theoretical analysis is given to (i) show how singular system of equations and empty margin sets are avoided, (ii)

87

provide detail analysis on the occurrence of degeneracy and cycling, and (iii) prove convergence showing how cycling is avoided with Bland's pivoting rule [4].

Review of the Regularization Path Algorithm

The following reviews the regularization path algorithm introduced by Hastie *et al.* [38] while providing some additional detail and insight. Recall, again, the SVM soft-margin primal formulation given the dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ and $\mathbf{x}_i \in \mathbb{R}^d$ where d is the data dimensionality and $y_i \in \{+1, -1\}$ represents the label for each point,

$$\min_{\mathbf{w}, b, \xi} \quad \tfrac{1}{2}\|\mathbf{w}\|_{\mathcal{H}}^2 + C\sum_{i=1}^n \xi_i \tag{6.1}$$

$$\text{subject to} \quad y_i\left(\langle \mathbf{w}, \mathbf{z}_i\rangle_{\mathcal{H}} + b\right) \geq 1 - \xi_i, \xi_i \geq 0 \forall i \tag{6.2}$$

where $\mathbf{z}_i = \phi\left(\mathbf{x}_i\right)$ and $\phi : \mathcal{X} \to \mathcal{H}$ represents a non-linear mapping to a Reproducing Kernel Hilbert Space (RKHS), $\mathbf{w} \in \mathcal{H}$. The goal is to find the solution to this problem as a function of C. The above can be recast in terms of λ where $\lambda \triangleq \frac{1}{C}$ for convenience.

$$\min_{\mathbf{w}, b, \xi} \quad \tfrac{\lambda}{2}\|\mathbf{w}\|_{\mathcal{H}} + \sum_{i=1}^n \xi_i \tag{6.3}$$

$$\text{subject to} \quad 1 - y_i f\left(x_i\right) \leq \xi_i; \xi_i \geq 0; f\left(\mathbf{x_i}\right) = \mathbf{w}^T \phi\left(\mathbf{x_i}\right) + b$$

This can, then, be converted to a dual formulation as follows. First, the Lagrangian of the primal becomes

$$L_P : \sum_{i=1}^n \xi_i + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^n \alpha_i\left(1 - y_i f\left(\mathbf{x_i}\right) - \xi_i\right) - \sum_{i=1}^n \gamma_i \xi_i \tag{6.4}$$

yielding

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \frac{1}{\lambda} \sum_{i=1}^{n} \alpha_i y_i \mathbf{x_i} \tag{6.5}$$

$$\frac{\partial L_P}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} y_i \alpha_i = 0 \tag{6.6}$$

$$\frac{\partial L_P}{\partial \xi_i} = 0 \quad \Rightarrow \quad 1 - \alpha_i - \gamma_i = 0 \tag{6.7}$$

The KKT complementary conditions along with the remaining inequality constraints are

$$\alpha_i \left(1 - y_i f\left(x_i \right) - \xi_i \right) = 0 \tag{6.8}$$

$$\gamma_i \xi_i = 0 \tag{6.9}$$

$$\alpha_i \geq 0 \tag{6.10}$$

$$\xi_i \geq 0 \tag{6.11}$$

$$\gamma_i \geq 0 \tag{6.12}$$

Using this, the following equivalent conditions are shown to be satisfied at optimality.

$$0 < \alpha_i < 1 \quad \Rightarrow \quad y_i f\left(x_i \right) = 1 \tag{6.13}$$

$$\alpha_i = 1 \quad \Rightarrow \quad y_i f\left(x_i \right) \leq 1 \tag{6.14}$$

$$\alpha_i = 0 \quad \Rightarrow \quad y_i f\left(x_i \right) \geq 1 \tag{6.15}$$

As reported in Hastie *et al.*, each case is easy to show using the KKT conditions as follows. $\alpha_i = 0$ implies that $\gamma_i = 1$ from (6.7) and $\xi_i = 0$ from (6.9). From (6.3) $1 - y_i f\left(x_i \right) \leq 0$ or $y_i f\left(x_i \right) \geq 1$. Similarly, if $\alpha_i = 1$ then $\gamma_i = 0$ from (6.7), and $1 - y_i f\left(x_i \right) - \xi_i = 0$ from (6.8). Using the fact that $\xi_i \geq 0$, then $y_i f\left(x_i \right) = 1 - \xi_i \Rightarrow y_i f\left(x_i \right) \leq 1$. Finally, if $0 < \alpha_i < C$ then $\gamma_i = 1 - \alpha_i$ and

$0 < \gamma_i < 1$ implying that $\xi_i = 0$ from (6.9) and $1 - y_i f(x_i) - \xi_i = 0$ implies that $y_i f(x_i) = 1$ since $\xi_i = 0$. The following sets are defined based upon these conditions

$$\mathcal{E} = \{i : y_i f(x_i) = 1, 0 \leq \alpha_i \leq 1\} \tag{6.16}$$

$$\mathcal{L} = \{i : y_i f(x_i) < 1, \alpha_i = 1\} \tag{6.17}$$

$$\mathcal{R} = \{i : y_i f(x_i) > 1, \alpha_i = 0\} \tag{6.18}$$

where $\mathcal{E}$ or elbow contains the points on the margin, $\mathcal{L}$ or left contain points that are inside the margin, not necessarily misclassified if on the correct side of the hyperplane, and $\mathcal{R}$ refers to those points outside the margin and correctly classified.

The Lagrangian dual function by definition is

$$\mathcal{L}_D(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = \inf_{\mathbf{w}, b, \boldsymbol{\xi}} \mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\gamma}) \tag{6.19}$$

where $\mathcal{L}_P()$ is defined in (6.4). Upon substituting in the expressions from (6.5), (6.6) and (6.7), the following is obtained

$$\mathcal{L}_D : \sum_i \alpha_i - \frac{1}{2\lambda} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_j}) \rangle_{\mathcal{H}} - b \sum_i \alpha_i y_i \tag{6.20}$$

and replacing the dot product with a Mercer kernel, $k(,)$, yields

$$\mathcal{L}_D : \sum_i \alpha_i - \frac{1}{2\lambda} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x_i}, \mathbf{x_j}) - b \sum_i \alpha_i y_i \tag{6.21}$$

Recalling the dual feasibility constraints

$$\sum_i \alpha_i y_i = 0 \tag{6.22}$$

$$0 \leq \alpha_i \leq 1 \ \ \forall i \tag{6.23}$$

the dual formulation becomes

$$\min_{\boldsymbol{\alpha}} \ \ \mathbf{1}^T\boldsymbol{\alpha} - \tfrac{1}{2\lambda}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \tag{6.24}$$

$$\text{s. t.} \ \ \ 0 \leq \alpha_i \leq 1 \ \ \forall i \tag{6.25}$$

$$\mathbf{y}^T\boldsymbol{\alpha} = 0 \tag{6.26}$$

where the objective is negated to obtain an equivalent minimization problem. Note that the dual solution for the typical SVM formulation, in terms of C, can be simply obtained from the solution to this dual formulation using $\alpha_i' = C\alpha_i = \alpha_i/\lambda$.

The goal of the regularization path algorithm is to trace out the solution of this optimization problem as the regularization parameter λ varies. At a particular ℓ^{th} breakpoint, an equality-constrained problem is formed using the active set defined by $\mathcal{E}_\ell$, $\mathcal{R}_\ell$, and $\mathcal{L}_\ell$. The sets $\mathcal{L}_\ell$ and $\mathcal{R}_\ell$ represent the set of equality constraints, i.e. $\alpha_i = 1$ and $\alpha_i = 0$, respectively. The remaining inequality constraints represented by $i \in \mathcal{E}_\ell$ ($0 \leq \alpha_i \leq 1$) will be automatically satisfied by the solution to the equality-constrained problem. The relationship between α_i^ℓ for $i \in \mathcal{E}_\ell$ and λ can be derived from this problem. From here on, the following notation will be used. For matrix A, $A_\mathcal{R}$ refers to both rows and columns of A indexed by $\mathcal{R}$, $A_{\mathcal{R},\mathcal{L}}$ refers to rows indexed by $\mathcal{R}$ and columns indexed by $\mathcal{L}$ and, for any vector $\mathbf{x}$, $\mathbf{x}_\mathcal{R}$ refers to entries of $\mathbf{x}$ indexed by $\mathcal{R}$.

The dual formulation can be rewritten as follows using $Q_{i,j} \triangleq y_i y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right)$ and noting that $\boldsymbol{\alpha}_\mathcal{R} =$

0.

$$-\frac{1}{2\lambda}\begin{pmatrix}\boldsymbol{\alpha}_{\mathcal{E}}\\\boldsymbol{\alpha}_{\mathcal{L}}\end{pmatrix}^T\begin{pmatrix}Q_{\mathcal{E}} & Q_{\mathcal{E},\mathcal{L}}\\Q_{\mathcal{L},\mathcal{E}} & Q_{\mathcal{L}}\end{pmatrix}\begin{pmatrix}\boldsymbol{\alpha}_{\mathcal{E}}\\\boldsymbol{\alpha}_{\mathcal{L}}\end{pmatrix}+\mathbf{1}^T\begin{pmatrix}\boldsymbol{\alpha}_{\mathcal{L}}\\\boldsymbol{\alpha}_{\mathcal{E}}\end{pmatrix}-\beta_0\begin{pmatrix}\mathbf{y}_{\mathcal{E}}^T & \mathbf{y}_{\mathcal{L}}^T\end{pmatrix}\begin{pmatrix}\boldsymbol{\alpha}_{\mathcal{E}}\\\boldsymbol{\alpha}_{\mathcal{L}}\end{pmatrix} \tag{6.27}$$

Taking the derivative with respect to $\boldsymbol{\alpha}_{\mathcal{E}}$ and setting this to 0 yields

$$-\frac{1}{\lambda}Q_{\mathcal{E}}\boldsymbol{\alpha}_{\mathcal{E}}-\frac{1}{\lambda}Q_{\mathcal{E},\mathcal{L}}\boldsymbol{\alpha}_{\mathcal{L}}+\mathbf{1}-\beta_0\mathbf{y}_{\mathcal{E}}=0 \tag{6.28}$$

Using $\alpha_0 = \lambda b$ to simplify notation,

$$-\frac{1}{\lambda}Q_{\mathcal{E}}\boldsymbol{\alpha}_{\mathcal{E}}-\frac{1}{\lambda}Q_{\mathcal{E},\mathcal{L}}\boldsymbol{\alpha}_{\mathcal{L}}+\mathbf{1}-\frac{1}{\lambda}\alpha_0\mathbf{y}_{\mathcal{E}}=0 \tag{6.29}$$

Immediately at the transition point ℓ where $\lambda = \lambda^\ell$ the following is obtained

$$-\frac{1}{\lambda^\ell}Q_{\mathcal{E}}\boldsymbol{\alpha}_{\mathcal{E}}^\ell-\frac{1}{\lambda^\ell}Q_{\mathcal{E},\mathcal{L}}\boldsymbol{\alpha}_{\mathcal{L}}+\mathbf{1}-\frac{1}{\lambda^\ell}\alpha_0^\ell\mathbf{y}_{\mathcal{E}}=0 \tag{6.30}$$

Multiplying each of the previous two expressions by λ and λ^ℓ, respectively, and subtracting, results in

$$Q_{\mathcal{E}}\left(\boldsymbol{\alpha}_{\mathcal{E}}-\boldsymbol{\alpha}_{\mathcal{E}}^\ell\right)+\mathbf{y}_{\mathcal{E}}\left(\alpha_0-\alpha_0^\ell\right)=\left(\lambda-\lambda^\ell\right)\mathbf{1} \tag{6.31}$$

The constraint $\mathbf{y}^T\boldsymbol{\alpha}=0$ yields $\mathbf{y}_{\mathcal{E}}^T\boldsymbol{\alpha}_{\mathcal{E}}=-\mathbf{y}_{\mathcal{L}}^T\mathbf{1}$ recalling that $\boldsymbol{\alpha}_{\mathcal{L}}=1$. Additionally, using the fact that $\mathbf{y}_{\mathcal{E}}\boldsymbol{\alpha}_{\mathcal{E}}^\ell=-\mathbf{y}_{\mathcal{L}}\mathbf{1}$ and subtracting the two expressions, the expression $\mathbf{y}_{\mathcal{E}}^T\left(\boldsymbol{\alpha}_{\mathcal{E}}-\boldsymbol{\alpha}_{\mathcal{E}}^\ell\right)=0$ is

obtained. Combining these expressions results in

$$\begin{pmatrix} Q_\ell & \mathbf{y}_\ell \\ \mathbf{y}_\ell^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}_\mathcal{E} - \boldsymbol{\alpha}_\mathcal{E}^\ell \\ \alpha_0 - \alpha_0^\ell \end{pmatrix} = \begin{pmatrix} \left(\lambda - \lambda^\ell\right) \mathbf{1} \\ 0 \end{pmatrix} \tag{6.32}$$

where $\mathbf{y}_\ell$ and Q_ℓ represents shorthand notation for $\mathbf{y}_{\mathcal{E}_\ell}$ and $Q_{\mathcal{E}_\ell}$.

Defining $\boldsymbol{\delta} \triangleq \frac{1}{\lambda - \lambda^\ell} \left(\boldsymbol{\alpha}_\mathcal{E} - \boldsymbol{\alpha}_\mathcal{E}^\ell\right)$, $\delta_0 \triangleq \frac{1}{\lambda - \lambda^\ell} \left(\alpha_0 - \alpha_0^\ell\right)$, yields

$$\begin{pmatrix} Q_\ell & \mathbf{y}_\ell \\ \mathbf{y}_\ell^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta} \\ \delta_0 \end{pmatrix} = \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix} \tag{6.33}$$

For future use, the l.h.s. of the above equation is defined as,

$$A_\ell \triangleq \begin{pmatrix} Q_\ell & \mathbf{y}_\ell \\ \mathbf{y}_\ell^T & 0 \end{pmatrix} \tag{6.34}$$

Given the solution $\boldsymbol{\delta}$, δ_0, then,

$$\alpha_j = \alpha_j^\ell + \left(\lambda - \lambda^\ell\right) \delta_j, j \in \{0\} \cup \mathcal{E}_\ell \tag{6.35}$$

The solution, in terms of α_j for $j \in \mathcal{E}$, then, varies linearly as a function of λ. At some point as λ is decreased from λ^ℓ, a variable α_j will become infeasible ($\alpha_j < 0$ or $\alpha_j > 1$) and the active set must change. As $\alpha_j \in \mathcal{E}$ and α_0 vary, the quantity $f\left(\mathbf{x}_i\right)$ for those points in $\mathcal{L}$ and $\mathcal{R}$ also varies. An active set change is also required when $y_i f\left(\mathbf{x}_i\right)$ approaches 1 for those points $\mathcal{L}$ or $\mathcal{R}$.

Given the linear relationship on $\alpha_\mathcal{E}$, the relationship for $f\left(\mathbf{x_i}\right)$ as a function of λ can be derived as follows.

$$f\left(\mathbf{x}_i\right) = \mathbf{w}^T \phi(\mathbf{x_i}) + b \tag{6.36}$$

93

or, expressed in terms of the Lagrange dual variables and using the Mercer kernel, becomes

$$f(\mathbf{x}_i) = \frac{1}{\lambda} \left[\sum_j \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j) + \alpha_0 \right] \tag{6.37}$$

where $\alpha_0 = \lambda b$. Finally, $f(\mathbf{x}_i)$ is partitioned as follows

$$f(\mathbf{x}_i) = \frac{1}{\lambda} \left[\sum_{j \in \mathcal{E}} \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{j \notin \mathcal{E}} \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j) + \alpha_0 \right] \tag{6.38}$$

Expressing $f(\mathbf{x}_i)$ in terms of $\lambda - \lambda^\ell$ results in the following

$$\lambda f(\mathbf{x}_i) - \lambda^\ell f^\ell(\mathbf{x}_i) = \sum_{j \in \mathcal{E}} \left(\alpha_j - \alpha_j^\ell \right) y_j k(\mathbf{x}_i, \mathbf{x}_j) + \left(\alpha_0 - \alpha_0^\ell \right) \tag{6.39}$$

where $f^\ell(\mathbf{x_i})$ is computed $f(\mathbf{x_i})$ based upon the values of $\alpha_j \forall j \in \mathcal{E}_\ell \cup 0$ at λ^ℓ. and, therefore,

$$\frac{1}{\lambda - \lambda^\ell} \left[\lambda f(\mathbf{x}_i) - \lambda^\ell f(\mathbf{x}_i) \right] = \sum_{j \in \mathcal{E}} \delta_j y_j k(\mathbf{x_i}, \mathbf{x_j}) + \delta_0 \tag{6.40}$$

Defining

$$h^\ell(\mathbf{x}_i) \triangleq \sum_{j \in \mathcal{E}} \delta_j y_j k(\mathbf{x}_i, \mathbf{x}_j) + \delta_0 \tag{6.41}$$

yields

$$\lambda f(\mathbf{x_i}) = \left(\lambda - \lambda^\ell \right) h^\ell(\mathbf{x_i}) + \lambda^\ell f^\ell(\mathbf{x_i}) \tag{6.42}$$

or

$$f(\mathbf{x}_i) = \frac{\lambda^\ell}{\lambda} \left[f^\ell(\mathbf{x}_i) - h^\ell(\mathbf{x}_i) \right] + h^\ell(\mathbf{x}_i) \tag{6.43}$$

As mentioned earlier, when $y_i f(\mathbf{x}_i)$ reaches 1 for some point in $\mathcal{L}, \mathcal{R}$, that point must be transitioned to $\mathcal{E}$ before progress can continue. The point at which this occurs is computed as follows

$$\frac{\lambda^{\ell}}{\lambda} \left[f^{\ell}(\mathbf{x}_i) - h^{\ell}(\mathbf{x}_i) \right] + h^{\ell}(\mathbf{x}_i) = y_i \tag{6.44}$$

or

$$\lambda = \lambda^{\ell} \left(\frac{f^{\ell}(\mathbf{x}_i) - h^{\ell}(\mathbf{x}_i)}{y_i - h^{\ell}(\mathbf{x}_i)} \right) \tag{6.45}$$

Finally, using Equations (6.45) and (6.35), the maximum λ is found such that $\lambda < \lambda^{\ell}$ at which one or more of the following occurs

1. The quantity $y_i f(\mathbf{x_i})$ approaches 1 for one or more points from $\mathcal{L}$ or $\mathcal{R}$.

2. One or more points from $\mathcal{E}$ have α_i approach 1 or 0.

Once found, the sets are adjusted and $\lambda^{\ell+1}$ is assigned to λ. The process continues, finding the breakpoints at which one of the above conditions occurs and adjusting $\mathcal{R}$, $\mathcal{L}$, and $\mathcal{E}$ until either $\mathcal{L}$ becomes empty, for a linearly separable dataset, or $\lambda^{\ell+1} \to 0$.

Initialization

The regularization path algorithm is initialized by finding the largest λ, or λ^0, such that for all $\lambda > \lambda^0$ the active set remains constant and $\boldsymbol{\alpha} = \boldsymbol{\alpha}^*$ where $\boldsymbol{\alpha}^*$ is the solution at λ^0. Define the index sets $\mathcal{I}_+$ and $\mathcal{I}_-$ containing the set of positively-labeled and negatively-labeled data points, respectively. Correspondingly, the quantities n_+ and n_- represent the number of positively-labeled and negatively-labeled data points. There are two cases to consider corresponding to when $n_- = n_+$ and when $n_- \neq n_+$. The following Lemmas, introduced in Hastie *et al.* [38], describe the form of the initial solution at λ_0 in each case as well as show that the solution does not change for $\lambda > \lambda^0$. The following is for the first case when $n_- = n_+$.

Lemma 1 ([38]). *For λ sufficiently large, all the $\alpha_i = 1$. The initial $b \in [-1, 1]$ – any value gives the same loss $\sum_{i=1}^{n} \xi_i = n_+ + n_-$*

The following describes the initial solution for the second case when $n_+ > n_-$, which is applicable w.l.o.g to the case when $n_- > n_+$.

Lemma 2 ([38]). *With $\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i x_i$, let*

$$\{\alpha_i^*\} \quad = \quad \arg\min ||\mathbf{w}||^2 \tag{6.46}$$

$$s.\ t.\ \alpha_i \in [0, 1] \ \ \forall i \in \mathcal{I}_+, \alpha_i = 1 \ \ \forall i \in \mathcal{I}_-, \sum_{i \in \mathcal{I}_+} \alpha_i = n_- \tag{6.47}$$

Then for some λ^0 we have that for all $\lambda > \lambda^0$, $\alpha_i = \alpha_i^*$.

These are easily shown [38], by first noting that $\mathbf{w} \to 0$ as $\lambda \to \infty$ according to Equation (6.5), and, therefore, the primal problem simply becomes

$$\min_{b} \quad \sum_{i=1}^{n} \xi_i \tag{6.48}$$

$$\text{s.t.} \quad 1 - y_i b \le \xi_i; \xi_i \ge 0 \tag{6.49}$$

Since $\xi_i > 0$, then

$$\xi_i = [1 - b]_+, \quad \forall i \in \mathcal{I}_+ \tag{6.50}$$

$$\xi_i = [1 + b]_+, \quad \forall i \in \mathcal{I}_- \tag{6.51}$$

defining

$$[x]_+ = \begin{cases} x & x > 0 \\ \\ 0 & x \le 0 \end{cases} \tag{6.52}$$

96
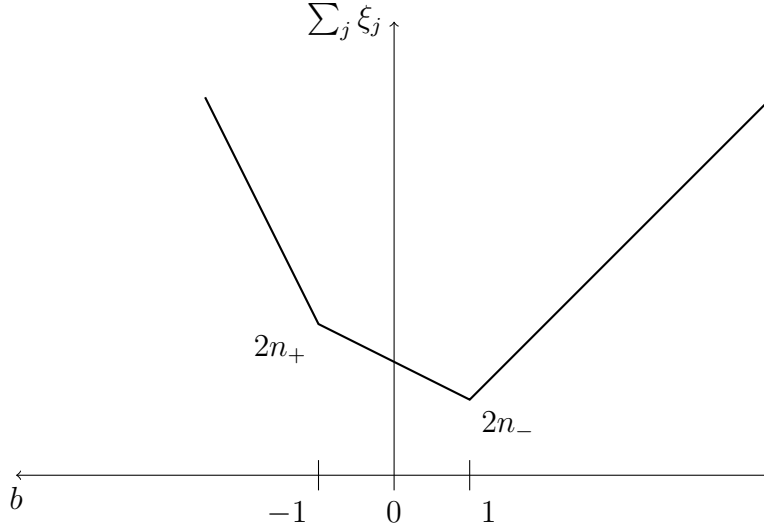
Figure 6.1: Depiction of the piecewise nature of the objective function as a function of b

Therefore $\sum_{i=1}^{n} \xi_i = n_+ \left[1 - b\right]_+ + n_- \left[1 + b\right]_+$ with the following results for several cases of b

$$\sum_{i=1}^{n} \xi_i = n_+ \left(1 - b\right), \qquad b < -1 \tag{6.53}$$

$$\sum_{i=1}^{n} \xi_i = n_- \left(1 + b\right), \qquad b > 1 \tag{6.54}$$

$$\sum_{i=1}^{n} \xi_i = n_+ + n_- + (n_- - n_+)b, \quad -1 \le b \le 1 \tag{6.55}$$

Figure 6.1 depicts the piece-wise nature of $\sum_j \xi_j$ as a function of b for the scenario where $n_+ > n_-$. In general, the minimum is achieved at a value of $b = \mathrm{sgn}\,(n_+ - n_-)$. For example, if $n_+ > n_-$, then a minimum of $2n_-$ is achieved for $b = 1$. On the other hand, if $n_+ = n_-$, the minimum is achieved for b in the interval $[-1, 1]$ and the cost function becomes $n_+ + n_-$. Immediate consequences are (i) $a_i = 1 \forall i \in \mathcal{I}_-$ (ii) $a_i \le 1 \forall i \in \mathcal{I}_+$. The first case follows since $\xi_i > 0$ for all $i \in \mathcal{I}_-$ implying $\gamma_i = 0$ from (6.8) and subsequently $\alpha_i = 1$ from (6.7). The second case occurs since it must also hold that $\sum_{i \in \mathcal{I}_+} a_i = n_-$ and, therefore, there will be at least one

index $i \in \mathcal{I}_+$, such that $a_i < 1$ unless $n_+ = n_-$, then $a_i = 1 \forall i \in \mathcal{I}_+$.

Departing slightly from the derivation in Hastie, it is interesting to derive the initial solution when λ is large but finite, again assuming, without loss of generality, $n_+ \geq n_-$ Choosing some finite λ sufficiently large such that $\boldsymbol{\alpha}$ remains constant as λ is varied results in the following primal problem

$$\min_{b, \boldsymbol{\xi}} \qquad \sum_i \xi_i \tag{6.56}$$

$$\text{s. t.} \quad y_i \left(\frac{1}{\lambda} \sum_j \alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b \right) \geq 1 - \xi_i \ \forall i \tag{6.57}$$

where $\mathbf{w}$, λ, and $\boldsymbol{\alpha}$ are constant and assumed known *a priori*. For the case $i \in \mathcal{I}_+$,

$$\xi_i = \left[1 - \frac{1}{\lambda} \sum_j \alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - b \right]_+ \tag{6.58}$$

and for $i \in \mathcal{I}_-$

$$\xi_i = \left[1 + \frac{1}{\lambda} \sum_j \alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b \right]_+ \tag{6.59}$$

A simplification is made by using $\mathbf{w}^* = \sum_j \alpha_j y_j \mathbf{x}_j$, and $\mathbf{w} = \frac{1}{\lambda} \mathbf{w}^*$, to obtain

$$\xi_i(b) = \left[1 - \frac{1}{\lambda} \mathbf{w}^{*T} \mathbf{x_i} - b \right]_+ \tag{6.60}$$

for $i \in \mathcal{I}_+$ and

$$\xi_i(b) = \left[1 + \frac{1}{\lambda} \mathbf{w}^{*T} \mathbf{x_i} + b \right]_+ \tag{6.61}$$

for $i \in \mathcal{I}_-$. Therefore, $\xi_i > 0$ for $b < 1 - \frac{1}{\lambda} \mathbf{w}^{*T} \mathbf{x_i}$, $i \in \mathcal{I}_+$, and $\xi_i > 0$ for $b > -1 - \frac{1}{\lambda} \mathbf{w}^{*T} \mathbf{x_i}$, $i \in \mathcal{I}_-$. Figure 6.2 depicts an example cost function, in terms of b, with $n_+ > n_-$. The minimum occurs, in this example, in an interval where $\xi_1 = 0$ while the remaining components are greater than 0. This implies, as expected, that there is at least one component of $\boldsymbol{\alpha}$ such that $\alpha_i < 1$ satisfied by

α_1 in this case. Also, in Figure 6.2, the curves for $\xi_i, i \in \mathcal{I}_+$ will translate to the left (towards decreasing b) as λ is decreased, for the cases $\mathbf{w}^{*T}\mathbf{x}_i > 0$, while, similarly, the curves for $\xi_i, i \in \mathcal{I}_-$ will translate to the right with decreasing λ, for the cases $\mathbf{w}^{*T}\mathbf{x}_i < 0$. It is important to note, in this example, that the interval of optimality for b is determined by ξ_2 and ξ_3, both corresponding to points in $\mathcal{I}_+$. As λ is initially decreased, the interval for b will be determined by points in $\mathcal{I}_+$ until,

$$\max_{i \in \mathcal{I}_-} \left(-1 - \frac{1}{\lambda} \mathbf{w}^{*T}\mathbf{x}_i \right) > \max_{i \in \mathcal{I}_+, \alpha_i < 1} \left(1 - \frac{1}{\lambda} \mathbf{w}^{*T}\mathbf{x}_i \right) \tag{6.62}$$

where the interval for b is

$$\left[\max_{i \in \mathcal{I}_+, \alpha_i < 1} \left(1 - \frac{1}{\lambda} \mathbf{w}^{*T}\mathbf{x}_i \right), \min_{i \in \mathcal{I}_+, \alpha_i = 1} \left(1 - \frac{1}{\lambda} \mathbf{w}^{*T}\mathbf{x}_i \right) \right] \tag{6.63}$$

$$\left[1 - \frac{1}{\lambda} \min_{i \in \mathcal{I}_+, \alpha_i < 1} \mathbf{w}^{*T}\mathbf{x}_i, 1 - \frac{1}{\lambda} \max_{i \in \mathcal{I}_+, \alpha_i = 1} \mathbf{w}^{*T}\mathbf{x}_i \right] \tag{6.64}$$

Eventually, as λ further decreases a point is reached where

$$-1 - \frac{1}{\lambda} \min_{i \in \mathcal{I}_-} \mathbf{w}^{*T}\mathbf{x}_i = 1 - \frac{1}{\lambda} \max_{i \in \mathcal{I}_+, \alpha_i < 1} \mathbf{w}^{*T}\mathbf{x}_i \tag{6.65}$$

or at

$$\lambda^c = \frac{\max_{i \in \mathcal{I}_+, \alpha_i < 1} \mathbf{w}^{*T}\mathbf{x}_i - \min_{i \in \mathcal{I}_-} \mathbf{w}^{*T}\mathbf{x}_i}{2} \tag{6.66}$$

Here, the interval for b is no longer strictly determined by all points in $\mathcal{I}_+$, but is determined by both points in $\mathcal{I}_+$ and $\mathcal{I}_-$ and the interval for b becomes

$$\left[1 - \frac{1}{\lambda} \min_{i \in \mathcal{I}_-} \mathbf{w}^{*T}\mathbf{x}_i, 1 - \frac{1}{\lambda} \max_{i \in \mathcal{I}_+, \alpha_i = 1} \mathbf{w}^{*T}\mathbf{x}_i \right] \tag{6.67}$$
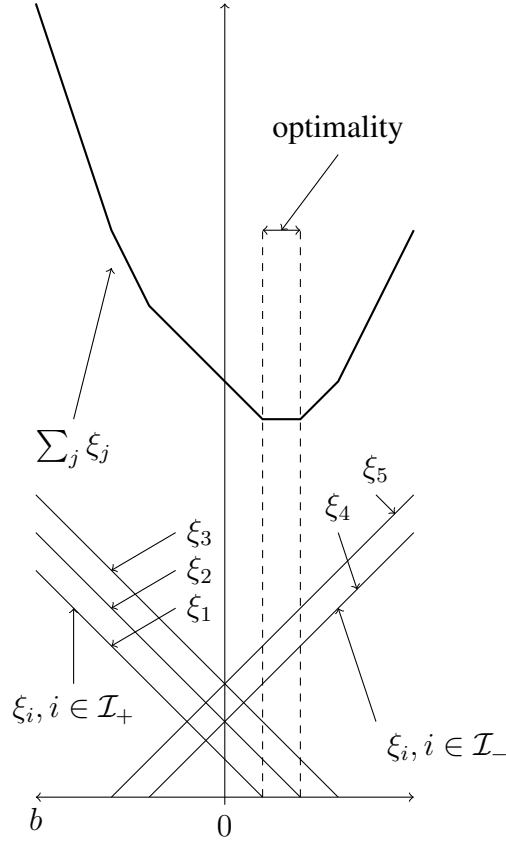
Figure 6.2: Depiction of the cost function and components ξ_i for finite but large λ

As λ continues to decrease the interval shrinks until, at some point, the upper and lower bounds are equal

$$-1 - \frac{1}{\lambda^0}\mathbf{w}^{*T}\mathbf{x}_{i-} = 1 - \frac{1}{\lambda^0}\mathbf{w}^{*T}\mathbf{x}_{i+} \tag{6.68}$$

where $i_+ = \arg\max_{i \in \mathcal{I}_+, \alpha_i = 1} \mathbf{w}^{*T}\mathbf{x}_i$ and $i_- = \arg\min_{i \in \mathcal{I}_-} \mathbf{w}^{*T}\mathbf{x_i}$.
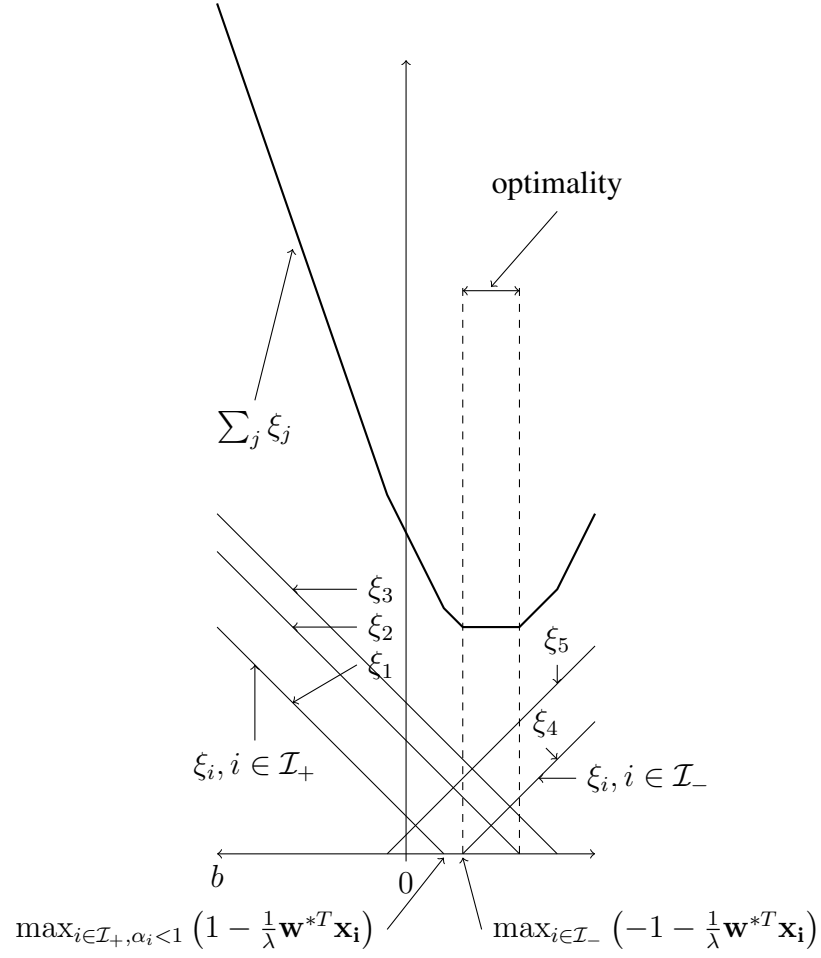
Figure 6.3: Depiction of the interval for b when $\lambda^c > \lambda > \lambda^0$ where the interval is bounded by points in both $\mathcal{I}_-$ and $\mathcal{I}_+$.

Therefore, λ^0 becomes

$$\lambda^0 = \frac{\mathbf{w}^{*T}\mathbf{x}_{i+} - \mathbf{w}^{*T}\mathbf{x}_{i-}}{2}, \tag{6.69}$$

and the quantity b is computed as

$$b = -1 - \frac{1}{\lambda^0}\mathbf{w}^{*T}\mathbf{x}_{i-} = 1 - \frac{1}{\lambda^0}\mathbf{w}^{*T}\mathbf{x}_{i+}. \tag{6.70}$$

Prior to reaching λ^0, b can take on any value in the interval. For the case when $n_+ = n_-$, for example, choosing b to be at one of the extremes of the interval results in the corresponding data point ($\mathbf{x}_{i+}$ or $\mathbf{x}_{i-}$) being on the margin. The opposing margin will then translate with decreasing λ until both points are on the margin. Typically, the active set method chooses at least one data point to be in $\mathcal{E}$ and will select one of the extremes. On the other hand, if $n_+ > n_-$, then the positive margin must be placed according to the data points such that $0 < \alpha_i < 1$. In this case, the opposing margin will translate until reaching the first data point of the opposite class.

Generalizing initialization for all kernel functions $g()$, then for $n_+ \geq n_-$, $i_+ = \arg\max_{i \in \mathcal{I}_+, \alpha_i = 1} g^* (\mathbf{x}_i)$, $i_- = \arg\min_{i \in \mathcal{I}_-} g^* (\mathbf{x}_i)$, and $g^* (\mathbf{x}_i) = \sum_j \alpha_j^* y_j k (\mathbf{x}_i, \mathbf{x}_j)$, $\alpha_i^* = 1 \ \forall i$ the starting solution becomes

$$\lambda^0 = \frac{g^* (\mathbf{x}_{i+}) - g^* (\mathbf{x}_{i-})}{2} \tag{6.71}$$

$$b = - \left(\frac{g^* (\mathbf{x}_{i+}) + g^* (\mathbf{x}_{i-})}{g^* (\mathbf{x}_{i+}) - g^* (\mathbf{x}_{i-})} \right) \tag{6.72}$$

In conclusion, a starting point for the regularization path algorithm can be trivially found when $n_+ = n_-$ where $\alpha_i = 1 \forall i$ and λ^0, b are derived from Equations (6.71) and (6.72). For the case $n_+ \neq n_-$, the initial solution $\boldsymbol{\alpha}^*$ is not known *a priori*. As a result, the starting point cannot be found from Equation (6.71). Further, Lemma 2 [38] only specifies the form of the starting solution. In this case, an external solver is required to find the initial starting solution. This can be done by either directly solving the optimization problem posed in Lemma 2 (such as in [17]) or by employing a training algorithm such as SMO to find the starting solution for some large λ. In the latter case, the true starting solution λ^0 is not found, but instead, a solution is found at some arbitrarily large λ corresponding to the user's interval of interest.

Algorithm 2 Hastie *et al.* Entire Regularization Path

1: Compute initial solution $\boldsymbol{\alpha}^*$ according to Lemma 1 or Lemma 2
2: Find initial solution

$$\lambda^0 = \frac{g\left(\mathbf{x}_{i+}\right) - g\left(\mathbf{x}_{i-}\right)}{2} \tag{6.73}$$

$$b = -\frac{g\left(\mathbf{x}_{i+}\right) + g\left(\mathbf{x}_{i-}\right)}{g\left(\mathbf{x}_{i+}\right) - g\left(\mathbf{x}_{i-}\right)} \tag{6.74}$$

$$i_+ = \arg\max_{i \in \mathcal{I}_+^1} g\left(\mathbf{x}_i\right), i_- = \arg\min_{i \in \mathcal{I}_-^1} g\left(\mathbf{x}_i\right) \tag{6.75}$$

where $\mathcal{I}_+^1 \subset \mathcal{I}_+$ and $\mathcal{I}_-^1 \subset \mathcal{I}_-$ such that $\alpha_i = 1$.
3: Initialize $f^0\left(\mathbf{x}_i\right) = b + \frac{1}{\lambda^0} g\left(\mathbf{x}_i\right)$
4: Set $\mathcal{E} = \{i_+, i_-\}$
5: Solve

$$\begin{pmatrix} Q_\ell & \mathbf{y}_\ell \\ \mathbf{y}_\ell^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta} \\ \delta_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6.76}$$

where ℓ refers to the state at the last ℓ^{th} breakpoint.
6: Compute

$$\lambda_j = \begin{cases} \frac{\lambda^\ell \delta_j - \alpha_j^l}{\delta_j}, \delta_j < 0 \\ \frac{1 + \lambda^\ell \delta_j - \alpha_j^l}{\delta_j}, \delta_j > 0 \end{cases} \tag{6.77}$$

7: Find max λ_j such that $\lambda_j < \lambda_\ell$
8: Compute

$$\lambda_j = \lambda^\ell \frac{f^\ell\left(\mathbf{x}_j\right) - h^\ell\left(\mathbf{x}_j\right)}{y_j - h^\ell\left(\mathbf{x}_j\right)} \tag{6.78}$$

9: Find max λ_j for $j \notin \mathcal{E}_\ell$ such that $\lambda_j < \lambda^\ell$
10: Set $\lambda^{\ell+1}$ to the largest of λ_j computed from step 7 and step 9
11: Adjust values

$$\alpha_j = \alpha_j^\ell - \left(\lambda^\ell - \lambda^{\ell+1}\right)\delta_j, j \in \{0\} \cup \mathcal{E}_\ell \tag{6.79}$$

$$f^{\ell+1}\left(\mathbf{x}_j\right) = \frac{\lambda^\ell}{\lambda^{\ell+1}} \left[f^\ell\left(\mathbf{x}_j\right) - h^\ell\left(\mathbf{x}_j\right)\right] + h^\ell\left(\mathbf{x}_j\right) \tag{6.80}$$

12: Adjust the sets $\mathcal{E}, \mathcal{R}, \mathcal{L}$
13: If $\mathcal{L}$ is empty or a minimum λ is reached, terminate, else go to step 5

Table 6.1: Example Toy Problem

	x_1	x_2	y
1	0.7	0.3	1
2	0.5	0.5	1
3	2.0	2.0	-1
4	1.0	3.0	-1
5	0.75	0.75	1
6	1.75	1.75	-1

Analysis of SVMPath

Analysis of a Toy Problem

As pointed out, the *SVMPath* algorithm cannot be used with semi-definite kernels or when dupli-cate data points exist. It is instructive to analyze the behavior of *SVMPath* in more detail using the Toy Problem introduced by Ong *et al.* [67]. The Toy problem is listed in Table 6.1. The following Q matrix is derived for the toy problem.

$$Q = \begin{pmatrix} 0.58 & 0.5 & -2.0 & -1.6 & 0.75 & -1.75 \\ 0.5 & 0.5 & -2.0 & -2.0 & 0.75 & -1.75 \\ -2.0 & -2.0 & 8.0 & 8.0 & -3.0 & 7.0 \\ -1.6 & -2.0 & 8.0 & 10.0 & -3.0 & 7.0 \\ 0.75 & 0.75 & -3.0 & -3.0 & 1.125 & -2.625 \\ -1.75 & -1.75 & 7.0 & 7.0 & -2.625 & 6.125 \end{pmatrix} \tag{6.81}$$

Since $n_- = n_+$, the initial solution is found by simply assigning $\alpha_i = 1$ for all i, which results in $\mathbf{w}^* = (-2.8, -5.2)$ and $\mathbf{w}^* \mathbf{x}_{i+} = -3.52$ and $\mathbf{w}^* \mathbf{x}_{i-} = -18.4$. Therefore,

$$\lambda^0 = \frac{-3.52 - (-18.4)}{2} = 7.44 \tag{6.82}$$

and

$$b = -\left(\frac{-3.52 + -18.4}{-3.52 - (-18.4)}\right) = 1.47311, \tag{6.83}$$

and, therefore,

$$\alpha_0 = \lambda^0 b = 10.9599348 \tag{6.84}$$

$$\alpha_i = 1 \quad \forall i \in \{1, ..., 6\} \tag{6.85}$$

Since $i_+ = 1$ and $i_- = 4$, then $\mathcal{E}_0 = \{1, 4\}$.

$$A_\ell = \begin{pmatrix} 1 & .58 & -1.6 \\ -1 & -1.6 & 10.0 \\ 0 & 1 & -1 \end{pmatrix}, \boldsymbol{\delta} = \begin{pmatrix} .271002 \\ .271002 \end{pmatrix}, \delta_0 = 1.2764 \tag{6.86}$$

and, for $j \in \mathcal{E}_0$,

$$\boldsymbol{\alpha} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - (7.44 - \lambda) \begin{pmatrix} .27102 \\ .27102 \end{pmatrix} \tag{6.87}$$

105

Table 6.2: Candidate λ at $\lambda^0 = 7.44$

I	λ	Move
1	3.75	$\mathcal{E} \to \mathcal{R}$
2	3.7501	$\mathcal{L} \to \mathcal{E}$
3	3.75	$\mathcal{L} \to \mathcal{E}$
4	3.75	$\mathcal{E} \to \mathcal{R}$
5	0.0003	$\mathcal{L} \to \mathcal{E}$
6	2.2857	$\mathcal{L} \to \mathcal{E}$

Solving for $\alpha_j = 0$ the next event becomes $\lambda = 3.75$ for both α_1, α_4. Solving the pricing variables results in

$$f^0 = \begin{pmatrix} 0.9999 \\ 0.935475 \\ -.67742763 \\ -1.000008 \\ .6666583 \\ -.40861043 \end{pmatrix}, h^0 = \begin{pmatrix} 1.0 \\ 0.8699186 \\ -.34959 \\ -1.0 \\ 0.6666 \\ -.1463414 \end{pmatrix} \tag{6.88}$$

and, using Equation (6.45),

$$\begin{pmatrix} \lambda_2 \\ \lambda_3 \\ \lambda_5 \\ \lambda_6 \end{pmatrix} = \begin{pmatrix} 3.7501 \\ 3.7500 \\ 0.0003 \\ 2.2857 \end{pmatrix} \tag{6.89}$$

The candidate events are summarized in Table 6.2.

If some small working precision is assumed, there are a total of 4 points that can be moved at

106

$\lambda = 3.75$. The *SVMPath* algorithm accepts all of the transitions resulting in $\mathcal{E}_1 = \{2,3\}$, $\mathcal{L}_1 = \{5,6\}$ and $\mathcal{R}_1 = \{1,4\}$. After performing the transition, an incremental update to $f(x)$ and α is performed yielding

$$\alpha_0^1 = 10.9599348 - (7.44 - 3.75)1.2764 = 6.25 \tag{6.90}$$

$$\boldsymbol{\alpha}^1 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}, f^1 = \begin{pmatrix} 1.0000 \\ 1.0000 \\ -1.0000 \\ -1.0000 \\ 0.6667 \\ -0.6667 \end{pmatrix} \tag{6.91}$$

This results in

$$A_\ell = \begin{pmatrix} 1 & 0.5 & -2.0 \\ -1 & -2.0 & 8.0 \\ 0 & 1 & -1 \end{pmatrix}, \boldsymbol{\delta} = \begin{pmatrix} 0.4444 \\ 0.4444 \end{pmatrix}, \delta_0 = 1.6667 \tag{6.92}$$

and, for $j \in \mathcal{E}_1$,

$$\boldsymbol{\alpha} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - (3.75 - \lambda) \begin{pmatrix} 0.4444 \\ 0.4444 \end{pmatrix} \tag{6.93}$$

The next event occurs at $\lambda = 1.5$ for both α_2, α_3 and for points in $\mathcal{L}_1, \mathcal{R}_1$,

$$
h^1 = \begin{pmatrix} 1.0000 \\ 1.0000 \\ -1.0000 \\ -1.0000 \\ 0.6667 \\ -0.6667 \end{pmatrix}
\tag{6.94}
$$

and, therefore,

$$
\begin{pmatrix} \lambda_1 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \end{pmatrix} = \begin{pmatrix} NaN \\ 3.7500 \\ -0.0000 \\ 0.0000 \end{pmatrix}
\tag{6.95}
$$

Note the fact that $f_1^1 = 1$ and $h_1^1 = 1$ implies there is no change in f_1 as λ decreases. However, a divide by zero error can occur if an attempt is made to compute the next event for point 1 according to (6.45). This highlights the need to detect the case $f_1^\ell - h_1^\ell = 0$. A summary of the next possible events is listed in Table 6.3. The *SVMPath* algorithm requires $\lambda^{\ell+1}$ strictly less than λ^ℓ, therefore, the event $\lambda = 1.5$ is chosen, ignoring the $\lambda = 3.75$ case, and $\mathcal{L}_2 = \{5, 6\}$, $\mathcal{E}_2 = \{\}$ and $\mathcal{R}_2 = \{1, 2, 3, 4\}$. Performing updates

$$
\alpha_0^2 = 6.25 - (3.75 - 1.5)1.6667 = 2.5
\tag{6.96}
$$

Table 6.3: Candidate λ at $\lambda^1 = 3.75$

I	λ	Move
2	1.5	$\mathcal{E} \to \mathcal{R}$
3	1.5	$\mathcal{E} \to \mathcal{R}$
4	3.75	$\mathcal{R} \to \mathcal{E}$
5	0.00	$\mathcal{L} \to \mathcal{E}$
6	0.00	$\mathcal{L} \to \mathcal{E}$

$$\boldsymbol{\alpha}^2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, f^2 = \begin{pmatrix} 1.0000 \\ 1.0000 \\ -1.0000 \\ -1.0000 \\ 0.6667 \\ -0.6667 \end{pmatrix} \tag{6.97}$$

At this point $\mathcal{E}$ is empty and reinitialization is required. Computing $\mathbf{w}^* = \sum_{i \in \mathcal{L}} y_i \mathbf{x_i} = (-1, -1)$, $\mathbf{w}^* \mathbf{x_{i+}} = -1.5$, $\mathbf{w}^* \mathbf{x_{i-}} = -3.5$ the next event becomes

$$\lambda^3 = \frac{-1.5 - (-3.5)}{2} = 1.0 \tag{6.98}$$

and

$$b^3 = -\left(\frac{-1.5 + -3.5}{-1.5 - (-3.5)} \right) = 2.5 \tag{6.99}$$

and now $\mathcal{L}_3 = \{\}, \mathcal{E}_3 = \{5, 6\}, \mathcal{R}_3 = \{1, 2, 3, 4\}$. The α values have not changed, i.e., $\boldsymbol{\alpha}^3 = \boldsymbol{\alpha}^2$.

109

$\alpha_0^3 = 2.5$ and

$$f^3 = \begin{pmatrix} 1.0000 \\ 1.0000 \\ -1.0000 \\ -1.0000 \\ 0.6667 \\ -0.6667 \end{pmatrix} \tag{6.100}$$

Since $\mathcal{L}$ is now empty, the algorithm terminates.

Multiple Regularization Paths

Looking again at the case $\lambda^1 = 3.75$, a question arises as to what other combination of events could have been taken. In order to satisfy the KKT conditions at $\lambda = 3.75$ any combination of the points $\{1, 2, 3, 4\}$ can be chosen to belong to $\mathcal{E}_2$. There is some confusion, however, since Hastie defines the set $\mathcal{E}$ as the one where $y_i f(\mathbf{x}_i) = 1$ and $0 \leq \alpha_i \leq 1$. Based upon this, all points should belong to $\mathcal{E}$. However, it is really only necessary that a chosen partition into $\mathcal{E}, \mathcal{L}$, and $\mathcal{R}$ continues to satisfy the KKT conditions as λ decreases. One can determine if the solution at $\ell = 2$, for example, satisfies the KKT conditions by looking more closely at the behavior as λ decreases. Consider, for example, that instead of taking all transitions at $\lambda = 3.75$, a partition is chosen such that $\mathcal{E}_2 = \{1, 2, 3\}$. The following results are obtained

$$\boldsymbol{\delta} = \begin{pmatrix} 0.4444 \\ 0.5333 \\ -0.0889 \end{pmatrix} \tag{6.101}$$

which can be rewritten in the following form

$$\boldsymbol{\alpha} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} - (3.75 - \lambda) \begin{pmatrix} 0.4444 \\ 0.5333 \\ -0.0889 \end{pmatrix} \tag{6.102}$$

Note that the solution, while initially satisfying the KKT conditions, will immediately violate the feasibility conditions upon decreasing λ since α_1 decreases with λ. In this case, the problem is detected by noting that $\alpha_1 = 0$ while $\delta_1 > 0$. Therefore, another partition must be chosen before λ can be allowed to decrease. In general, it must be the case that $\delta_i \geq 0$ when $\alpha_i = 1$ and $\delta_i \leq 0$ when $\alpha_i = 0$. Similarly, it must be the case that $y_i \left(f^\ell (\mathbf{x}_i) - h^\ell (\mathbf{x}_i) \right) \geq 0$ for $i \in \mathcal{R}$ and $y_i \left(f^\ell (\mathbf{x}_i) - h^\ell (\mathbf{x}_i) \right) \leq 0$ for $i \in \mathcal{L}$ when $y_i f^\ell (\mathbf{x}_i) = 1$. All possible regularization paths can be enumerated by considering all possible combinations of transitions at each breakpoint. For each combination of transitions, the next breakpoint $\lambda^{\ell+1}$ is computed, and the case $\lambda^\ell = \lambda^{\ell+1}$ is allowed. The result of doing this for the Toy Problem is depicted in Figure 6.4.

There are several that can be made (i) there is a path where the KKT matrix, A_ℓ, becomes singular but many more where this is avoided (ii) there is one path where $\mathcal{E}$ becomes empty and an interesting portion of the regularization path is skipped (iii) there are many paths where an empty margin set, $\mathcal{E}$ is avoided, and (iv) there are several cases where repeat λ values occur or $\lambda^{\ell+1} = \lambda^\ell$ before continuing with a strictly smaller value. It is also important to note that the sequence of distinct λ values as well as α_0 values is the same regardless of the path taken with the exception of the interval between $\lambda = 1.5$ and $\lambda = 1.0$ where two different paths exist.
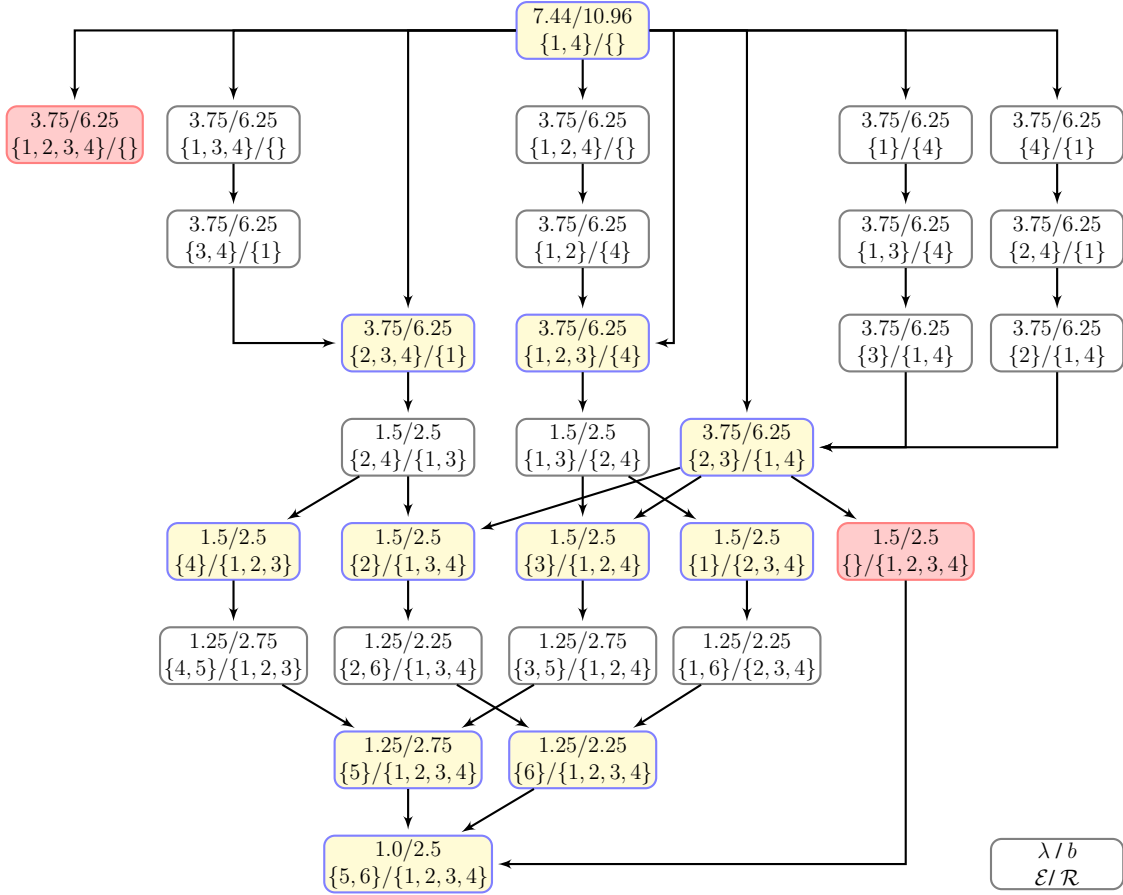
Figure 6.4: All possible solution paths for the toy problem. Cells highlighted in red represent scenarios where either a singular system of equations results or the margin set becomes empty. Cells highlighted in yellow represent states from which λ can be decreased without requiring further active set transitions

Empty Margin Set

It is important to consider the case when the margin set is allowed to become empty in more detail. Note that there are two possible values for α_0 at $\lambda = 1.25$, that is, $\alpha_0 = 2.25$ or $\alpha_0 = 2.75$ depending upon whether point 5 or 6 is in $\mathcal{E}$ at $\lambda = 1.25$. This implies at least two different regularization paths exist between $\lambda = 1.5$ and $\lambda = 1.0$.

(a) Path (a) or (b), $\lambda = 1.5$ (b) Path (a), $\lambda = 1.25$ (c) Path (b), $\lambda = 1.25$ (d) Path (a) or (b), $\lambda = 1.0$
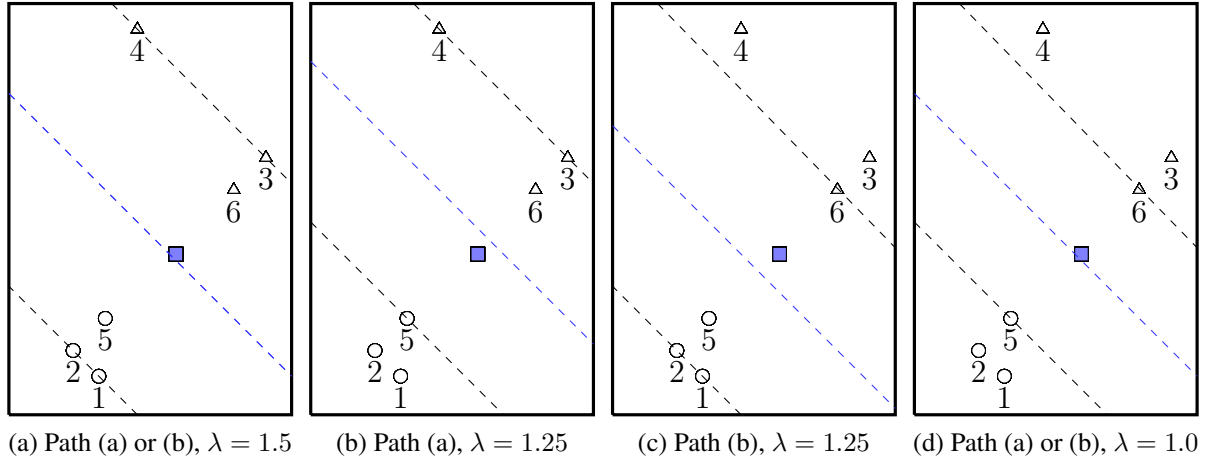
Figure 6.5: Depiction of multiple paths between $\lambda = 1.5$ and $\lambda = 1.0$ for the Toy Problem. SVMPath skips this portion of the path proceeding directly to $\lambda = 1.0$ from $\lambda = 1.5$. Two different paths exist depending upon whether points 1 or 2 or points 3 or 4 remain on the margin while λ is decreased from 1.5. A test data point (blue square) is classified differently for path (a) and (b) at $\lambda = 1.25$

The existence of these multiple paths can be verified by finding a solution at $\lambda = 1.25$ using a conventional SVM training algorithm (e.g. SMO) and randomly permuting the order in which data is presented. It is interesting to note that since a different value of α_0 is achieved along each path, errors against previously unseen test data may well be different for each path.

Multiple regularization paths result when at some λ^ℓ there is a situation where all data points are candidates for transitioning out of $\mathcal{E}$ and data points exist on both margins. When a single data point is left in $\mathcal{E}$, the corresponding margin becomes pinned to that data point while the opposing margin translates. Therefore, there are at least (2) choices corresponding to whether at least one point remains on the $y = +1$ or $y = -1$ margin. Figure 6.5 depicts the two possible scenarios for the Toy Problem. In one instance, points 3 and/or 4 remain in $\mathcal{E}$ while the opposing margin translates until reaching point 5 at $\lambda = 1.25$. Point 5 is then placed in $\mathcal{E}$ while the remaining points are removed. The margin at point 5 then becomes pinned while the opposing margin translates

113

until hitting point 6 at $\lambda = 1.0$. In a different path, points 1 and/or 2 remain on the margin while the opposing margin translates until hitting point 6 at $\lambda = 1.25$. As shown in Figure 6.5, classification for a strategically-placed data point may depend upon the regularization path taken. In this example the impact on overall performance is minimal; however, this could be important in different scenarios.

Simple SVMPath

The Simple SVMPath algorithm is inspired by noting that the empty margin set as well as singular KKT system tends to occur in cases where more than one active set transition takes place. On the other hand, the previous analysis suggests that the regularization path can be successfully computed by performing only a single transition at a time and allowing repeat events, $\lambda^\ell = \lambda^{\ell+1}$. It is important to note that the sets $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{L}$ are mutually exclusive and cannot be used for tracking an active set. For example, points with $y_i f(\mathbf{x}_i) = 1$ and $\alpha_i = 0$ or $\alpha_i = 1$ are forced to belong to $\mathcal{E}$. This partition is employed to identify transitions within the SVMPath algorithm, but represents a primary source of difficulty especially when accommodating semi-definite kernels and will not allow single transitions. Therefore, the following sets are defined where $\mathcal{F}$ contains the subset of variables in $\mathcal{E}$ that are free to vary (inactive constraints), $\mathcal{R}^*$ contains points not in $\mathcal{F}$ where $\alpha_i = 0$, and $\mathcal{L}^*$ contains points not in $\mathcal{F}$ and $\alpha_i = 1$. As a result, points in $\mathcal{F}$ will correspond to the set of inactive constraints while those in $\mathcal{R}^*$ and $\mathcal{L}^*$ correspond to the set of active constraints and the sets now overlap in terms of the KKT optimality conditions. For example, points with $y_i f(\mathbf{x}_i) = 1$ and $\alpha_i = 1$ can belong to either $\mathcal{F}$ or $\mathcal{L}^*$ depending upon whether the points is chosen to belong to the inactive set, $\mathcal{F}$ or the active set $\mathcal{L}^*$.

The descent direction $\boldsymbol{\delta}$ and δ_0 as well as $h^\ell(\mathbf{x}_i)$ is computed as before using Equations (6.41) and (6.35); however, instead, using the index sets $\mathcal{F}$, $\mathcal{R}^*$, and $\mathcal{L}^*$. As a result, the components Q_ℓ and

$\mathbf{y}_\ell$ in Equation (6.33) are now indexed by $\mathcal{F}_\ell$ instead of $\mathcal{E}_\ell$. Specifically,

$$\alpha_j = \alpha_j^\ell + \left(\lambda - \lambda^\ell\right)\delta_j, j \in \{0\} \cup \mathcal{F}_\ell \tag{6.103}$$

and, for points in $\mathcal{L}^*$ and $\mathcal{R}^*$

$$f\left(\mathbf{x}_i\right) = \frac{\lambda^\ell}{\lambda}\left[f^\ell\left(\mathbf{x}_i\right) - h^\ell\left(\mathbf{x}_i\right)\right] + h^\ell\left(\mathbf{x}_i\right) \tag{6.104}$$

At each step, the next event is found by selecting the minimum λ_j such that $\lambda_j \geq \lambda^\ell$ where a point in $\mathcal{F}_\ell$ enters $\mathcal{R}^*$ or $\mathcal{L}^*$ or vice versa a point in $\mathcal{R}_\ell^*$ or $\mathcal{L}_\ell^*$ enters $\mathcal{F}$.

The event at which a point in $\mathcal{F}_\ell$ transitions to $\mathcal{L}^*$ or $\mathcal{R}^*$ is computed as follows

$$\lambda_j = \frac{1 + \lambda^\ell \delta_j - \alpha_j^\ell}{\delta_j} \tag{6.105}$$

for $\delta_j < 0$ and for $\delta_j > 0$

$$\lambda_j = \frac{\lambda^\ell \delta_j - \alpha_j^\ell}{\delta_j} \tag{6.106}$$

For a point in $\mathcal{L}_\ell^*$ or $\mathcal{R}_\ell^*$, once again, the event at which a point enters $\mathcal{F}$ is computed as follows

$$\lambda_j = \lambda^\ell \frac{\left(f_j^\ell - h_j^\ell\right)}{\left(y_j - h_j^\ell\right)} \tag{6.107}$$

Note in the above equation that $y_i - h_i^\ell = 0$ implies $y_i h_i^\ell = 1$, and it is simple to show that for $y_i f^\ell\left(\mathbf{x}_i\right) > 1$ the quantity $y_i f\left(\mathbf{x}_i\right)$ will increase as λ decreases and, similarly, $y_i f\left(\mathbf{x}_i\right)$ will decrease if $y_i f^\ell\left(\mathbf{x}_i\right) < 1$. Therefore, for data points having $h_i^\ell = 1$, the divide by zero operation can be avoided and $\lambda_j = 0$ since these points will not transition for any value of λ. Similarly, λ_j need not be computed for $y_i\left(f_i^\ell - h_i^\ell\right) > 0$ for $i \in \mathcal{R}_\ell$ and $y_i\left(f_i^\ell - h_i^\ell\right) < 0$ for $i \in \mathcal{L}_\ell$. In

summary, the next event is computed as follows

$$
\lambda_j = \begin{cases}
\frac{(1+\lambda^\ell \delta_j - \alpha_j^\ell)}{\delta_j}, & j \in \mathcal{F}_\ell, \delta_j < 0 \\[2mm]
\frac{(\lambda^\ell \delta_j - \alpha_j^\ell)}{\delta_j}, & j \in \mathcal{F}_\ell, \delta_j > 0 \\[2mm]
\lambda^\ell \frac{\left(f_j^\ell - h_j^\ell\right)}{\left(y_j - h_j^\ell\right)}, & j \in \mathcal{L}_\ell^*, \left|y_j - h_j^\ell\right| > \tau, y_j \left(f_j^\ell - h_j^\ell\right) > -\tau \\[2mm]
\lambda^\ell \frac{\left(f_j^\ell - h_j^\ell\right)}{\left(y_j - h_j^\ell\right)}, & j \in \mathcal{R}_\ell^*, \left|y_j - h_j^\ell\right| > \tau, y_j \left(f_j^\ell - h_j^\ell\right) < \tau \\[2mm]
0 & \text{otherwise}
\end{cases}
\tag{6.108}
$$

where a set of tolerances have been incorporated, τ. Once computed, the next event is chosen

where $\lambda^{\ell+1} = \lambda_j$ such that $\left(\lambda_j - \lambda^\ell\right) / \lambda^\ell \leq \tau$. The complete algorithm is described as follows.

Algorithm 3 Simple SVM Path Algorithm

1: Given $\hat{\lambda}$, $K \in \mathbb{R}^{n \times n}$, $\mathbf{y} \in \mathcal{Y}^n$, $\rho > 0$, $\tau > 0$
2: **if** $N_+ \neq N_-$ **then**
3:     Augment kernel matrix using artificial variable method (see section on initialization).
4: **end if**
5: compute λ^0, b from (6.71), (6.72)
6: $\alpha_0 \leftarrow \lambda^0 b$
7: $\alpha_i \leftarrow 1 \forall i$
8: Compute initial f_i according to (6.38) (using $\mathcal{F}_0$ instead of $\mathcal{E}$)
9: **while** $\lambda > \hat{\lambda}$ and $\mathcal{L}^*$ is not empty **do**
10:     Solve the KKT system (6.33)
11:     Solve λ_j using (6.108)
12:     Find the smallest λ_j such that $\left(\lambda_j - \lambda^\ell\right) / \lambda^\ell \leq \tau$
13:     **if** there is a tie **then**
14:         Choose the minimum index j
15:     **end if**
16:     $\alpha_i \leftarrow \alpha_i^\ell - \left(\lambda^\ell - \lambda_j\right) \beta$ for $i \in \mathcal{F}_\ell, i = 0$
17:     $f_i \leftarrow \frac{\lambda}{\lambda_j}(f_i^\ell - h_i^\ell) + h_i^\ell$
18:     Move point j to new set $\mathcal{F}, \mathcal{L}^*$, or $\mathcal{R}^*$
19:     $\lambda^{\ell+1} \leftarrow \lambda_j$
20: **end while**

The regularization path-following algorithm can exhibit numerical instability due to incremental updating of $\boldsymbol{\alpha}$, α_0, and $f(\mathbf{x_i})$ according to Equations (6.103) and (6.104). These quantities can be recomputed during each iteration to improve stability at the expense of significantly increased training times. The SVMPath algorithm, for example, recomputes these quantities and the authors state that methods for improving numerical stability with incremental updates is an open area of research [38]. In this research, careful application of the tolerances prescribed in Equation (6.108) is found to be necessary to guarantee numerical stability. In addition, the proposed tolerances are also critical for preventing near-singular KKT systems. Since repeat events are allowed, it is also important to include a tolerance when finding the largest λ_j such that $\lambda_j \geq \lambda^\ell$. In fact, the largest λ_j is chosen such that $\left(\lambda_j - \lambda^\ell\right)/\lambda^\ell \leq \tau$. Normalization is necessary since λ can vary by orders of magnitude between its largest and smallest value.

Analysis

In the analysis of the *SVMPath* algorithm for a toy problem, the regularization path following algorithm was shown to skip a portion of the regularization path when the margin set, $\mathcal{E}$ is allowed to become empty. This scenario is automatically avoided when only single transitions are allowed to occur. Specifically, the following shows that $\mathcal{F}$ never becomes empty since, if there is only one point in $\mathcal{F}$, that point will never be identified for transition.

Proposition 1. *If $\mathcal{F}$ contains only a single point, i, then the component α_i will remain constant as λ decreases with $y_i f(x_i) = 1$.*

This easily follows since when $|\mathcal{F}| = 1$, the solution to (6.33) is $\delta_0 = y_i$, $\delta_i = 0$. When only a single point remains in $\mathcal{F}$, the associated margin will become "pinned" while the opposing margin

translates as λ decreases. This proceeds until another point enters the margin and becomes a candidate for entering $\mathcal{F}$. While the empty margin set is avoided, the potential existence of multiple regularization paths is not addressed. For example, in the toy problem, only one of the two paths between $\lambda = 1.5$ and $\lambda = 1.0$ will be taken. As an alternative, it is possible to enumerate all of the alternative solutions paths. This is not done within this research.

The following set of theorems indicate that the system of equations (6.33) will never become singular when performing single transitions. More specifically, a point that is on the margin but not in $\mathcal{F}_\ell$ and that is linearly related to the points in $\mathcal{F}_\ell$ such that adding the point to $\mathcal{F}_\ell$ would result in a singular system will never be identified to transition into $\mathcal{F}$.

Theorem 5. *A data point on the margin, $\mathbf{x}_i$, and not in $\mathcal{F}_\ell$, such that adding the point to $\mathcal{F}_\ell$ yields a singular matrix, A_ℓ, will remain on the margin, or $y_i f(\mathbf{x}_i) = 1$, as λ is decreased.*

Proof. Upon adding the point i to $\mathcal{F}$ at iteration ℓ, (6.33) becomes

$$
\begin{pmatrix} Q_{\mathcal{F}} & \mathbf{q}_i & \mathbf{y}_{\mathcal{F}} \\ \mathbf{q}_i^T & q_{ii} & y_i \\ \mathbf{y}_{\mathcal{F}}^T & y_i & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta} \\ \delta_i \\ \delta_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \tag{6.109}
$$

where $\mathbf{q}_i$ contains the rows of Q indexed by $\mathcal{F}$ and the column corresponding to the data point i. Since the system of equations is singular, $\mathbf{u}$, u_y can be found such that $\mathbf{q}_i = Q\mathbf{u} + \mathbf{y}_{\mathcal{F}} u_y$, $y_i = \mathbf{y}_{\mathcal{F}}^T \mathbf{u}$. In order for a solution to exist, it must also be true that $\mathbf{u}^T \mathbf{1} = 1$. When i is not in $\mathcal{F}$, then $Q_{\mathcal{F}} \left(\boldsymbol{\alpha}_{\mathcal{F}} - \boldsymbol{\alpha}_{\mathcal{F}}^\ell \right) + \mathbf{y}_{\mathcal{F}} \left(\alpha_0 - \alpha_0^\ell \right) = \mathbf{1} \left(\lambda - \lambda^\ell \right)$ from (6.33) and

$$
\lambda y_i f(\mathbf{x}_i) - \lambda^\ell y_i f^\ell (\mathbf{x}_i) \tag{6.110}
$$

$$
= \mathbf{q}_i \left(\boldsymbol{\alpha}_{\mathcal{F}} - \boldsymbol{\alpha}_{\mathcal{F}}^\ell \right) + y_i \left(\alpha_0 - \alpha_0^\ell \right) \tag{6.111}
$$

From this

$$\lambda y_i f\left(\mathbf{x}_i\right) - \lambda^\ell y_i f^\ell\left(\mathbf{x}_i\right) \tag{6.112}$$

$$= \left(\mathbf{u}^T Q_{\mathcal{F}} + u_y \mathbf{y}_{\mathcal{F}}^T\right)\left(\boldsymbol{\alpha}_{\mathcal{F}} - \boldsymbol{\alpha}_{\mathcal{F}}^\ell\right) + \mathbf{u}^T \mathbf{y}_{\mathcal{F}}\left(\alpha_0 - \alpha_0^\ell\right) \tag{6.113}$$

$$= \mathbf{u}^T\left(\mathbf{1}\left(\lambda - \lambda^\ell\right)\right) \tag{6.114}$$

$$= \lambda - \lambda^\ell \tag{6.115}$$

and, therefore, $y_i f\left(\mathbf{x}_i\right) = y_i f^\ell\left(\mathbf{x}_i\right) = 1$. □

Implicitly, this means that $y_i\left(f_i^\ell - h_i^\ell\right) = 0$ or $y_i h_i^\ell = 1$. As a result, via (6.108), $\lambda_i = 0$ will be computed for the linearly dependent point and it will never be identified to enter $\mathcal{F}_\ell$ unless another point first leaves $\mathcal{F}_\ell$. This result also implies that the point will remain on the margin as λ is decreased. The following corollary easily follows for duplicate data points

Corollary 1. *Given any data point $\mathbf{x}_i$ not in $\mathcal{F}_\ell$, that is on the margin, $y_i f^\ell\left(\mathbf{x}_i\right) = 1$, and that is a duplicate of a data point in $\mathcal{F}_\ell$, i.e., $\mathbf{x}_i = \mathbf{x}_j$ and $y_i = y_j$ for some j in $\mathcal{F}_\ell$, that point will remain on the margin and $y_i f\left(\mathbf{x}_i\right) = 1$ as λ decreases.*

In the case of duplicate data points, only one of the duplicate data points is placed in $\mathcal{F}_\ell$ at any time. Only when the duplicate data point in $\mathcal{F}$ transitions out of $\mathcal{F}$ is another available for transition into $\mathcal{F}$. As an example, when transitioning duplicate data points from $\mathcal{L}^*$ to $\mathcal{R}^*$, a first duplicate data point is added to $\mathcal{F}$ and, as λ is decreased, its corresponding variable α_i is driven to zero. Once this occurs, the first data point is transitioned into $\mathcal{R}^*$ and another of the duplicate data points remaining in $\mathcal{L}^*$ is free to transition into $\mathcal{F}$. This process is repeated, not necessarily sequentially (other points not related to the duplicate data points may transition sets), until all of the duplicate data points have $\alpha_i = 0$ and are in $\mathcal{R}^*$. This incurs additional iterations on the order of the number of duplicate data points; however the impact is expected to be minimal when the duplicate data

points are a small fraction of the total data points.

In summary, by performing single active set transitions at a time, (i) the proposed algorithm automatically precludes the scenario where the margin set or $\mathcal{F}$ is allowed to become empty and, therefore, avoids the risk of skipping a portion of the regularization path, (ii) avoids creating singular systems of equations and the need for specialized factorization or rank-detecting mechanisms, and (iii) automatically handles duplicate data points.

Degeneracy and Cycling

The active set method is proven to converge in the absence of degeneracies. A degeneracy occurs when an active set change does not result in a corresponding decrease in the objective function. When this occurs there is a risk of cycling where a series of active set transitions are repeated with no further progress being made. In the majority of active set implementations, this is disregarded as a minor problem and is often ignored.

In the case of the regularization path algorithm, the task of finding the next event $\lambda^{\ell+1}$ can be posed as a linear programming problem as follows

$$
\begin{aligned}
\min \quad & \lambda && \text{(6.116)} \\
\text{subject to} \quad & \lambda \leq \lambda^{\ell} \\
& 0 \leq \alpha_i \leq 1 \ \ \forall i \in \mathcal{F}_{\ell} \\
& y_i f(\mathbf{x}_i) \geq 1 \ \ \forall i \in \mathcal{R}_{\ell}^{*} \\
& y_i f(\mathbf{x}_i) \leq 1 \ \ \forall i \in \mathcal{L}_{\ell}^{*}
\end{aligned}
$$

Therefore, the case where $\lambda = \lambda^{\ell}$ represents a form of degeneracy since the objective value does

120

not decrease. This is also true when considering the original dual objective function since no changes are made to $\boldsymbol{\alpha}$, α_0, or λ. In the case of the SSVMP algorithm, cycling will occur if a series of active transitions with $\lambda^\ell = \lambda^{\ell+1}$ results in a revisit of a previous partition into $\mathcal{F}_\ell$, $\mathcal{R}_\ell^*$, or $\mathcal{L}_\ell^*$. For an example of degeneracy, consider, again, the analysis of toy problem depicted in Figure 6.4. There are several cases where $\lambda^\ell = \lambda^{\ell+1} = 3.75$ and a single active set transition occurs. For example, consider the path where $\lambda^1 = 3.75$, $\mathcal{E}_1 = \{1\}$, $\mathcal{R}_1 = \{4\}$ followed by $\lambda^2 = 3.75$, $\mathcal{E}_2 = \{1,3\}$, and $\mathcal{R}_2 = \{4\}$, which finally ends in $\lambda^4 = 3.75$, $\mathcal{E}_4 = \{2,3\}$, and $\mathcal{R}_4 = \{1,4\}$. In the analysis, cycling was prevented since a previous partition into $\mathcal{E}$, $\mathcal{L}$, and $\mathcal{R}$ was actively avoided. However, in the course of algorithm execution, cycling would occur if, for example, step 4 revisited the partitioning at step 1 or $\mathcal{E}_4 = \{1\}$ and $\mathcal{R}_4 = \{4\}$.

Degeneracy can only occur when there are multiple data points with $y_i f(\mathbf{x}_i) = 1$ and $\alpha_i = 0$ or $\alpha_i = 1$. That is, there are several data points that can be simultaneously transitioned between $\mathcal{F}$, $\mathcal{R}^*$, and $\mathcal{L}^*$ for some event λ^ℓ. For example, in the toy problem, it is the case that $\alpha_1 = \alpha_4 = 0.0$ and $\alpha_2 = \alpha_3 = 1.0$ at $\lambda = 3.75$ and, therefore, there are 4 candidate transitions. Using the previous example, degeneracy occurs since upon transitioning point 4 to $\mathcal{R}$ at $\lambda^1 = 3.75$, continued progress was blocked by point 3, which was later transitioned to $\mathcal{E}$ at step $\lambda^2 = 3.75$.

It should be pointed out that degeneracy and cycling is not specific to the *SSVMP* implementation, but is, instead, a characteristic of the regularization path following method. For example, Hastie *et al.* [38] avoid this issue by requiring $\lambda^{\ell+1}$ be strictly less than λ^ℓ, while Ong *et al.*, in the *ISVMP* algorithm [67], incorporate a backup routine that is invoked upon detecting cycling. Finally, in the work by Karasuyama *et al.* [44], an explicit method is included for breaking ties in the face of degeneracy.

It can be easily shown that the following conditions must be met at a particular value of λ before

progress can continue with $\lambda^{\ell+1} < \lambda_\ell$.

$$j \in \mathcal{F}_\ell, \delta_j < 0 \to \alpha_j < 1 \tag{6.117}$$

$$j \in \mathcal{F}_\ell, \delta_j > 0 \to \alpha_j > 0 \tag{6.118}$$

$$j \in \mathcal{R}_\ell^*, y_j f^\ell (\mathbf{x}_j) = 1 \tag{6.119}$$

$$\to y_j \left[f^\ell (\mathbf{x}_j) - h^\ell (\mathbf{x}_j) \right] > 0$$

$$j \in \mathcal{L}_\ell^*, y_j f^\ell (\mathbf{x}_j) = 1 \tag{6.120}$$

$$\to y_j \left[f^\ell (\mathbf{x}_j) - h^\ell (\mathbf{x}_j) \right] < 0$$

When faced with a degeneracy, the *SSVMP* algorithm will automatically choose one point for transition that does not satisfy these conditions. As an example, a point in $\mathcal{F}_\ell$ with $\alpha_i = 1$ will be chosen such that $\delta_i < 0$. In other words, a transition will occur for any data point meeting any of the following criterion.

$$\delta_j < 0 \quad \text{for} \quad j \in \mathcal{F}_\ell, \alpha_j = 1 \tag{6.121}$$

$$\delta_j > 0 \quad \text{for} \quad j \in \mathcal{F}_\ell, \alpha_j = 0 \tag{6.122}$$

$$\left(1 - y_j h_j^\ell\right) < 0 \quad \text{for} \quad j \in \mathcal{R}_\ell^* \tag{6.123}$$

$$\left(1 - y_j h_j^\ell\right) > 0 \quad \text{for} \quad j \in \mathcal{L}_\ell^* \tag{6.124}$$

noting that $y_j f (\mathbf{x}_j) = 1$ for the degenerate data points since they will already be on the margin. At a given λ value, transitions will continue at the same λ value until there are no longer any points satisfying the above conditions and λ can be decreased.

Degeneracy, alone, is not an issue unless it creates a condition for cycling. Therefore, it is important to analyze the prevalence of cycling given the possible forms of degeneracy. In the face of a semi-definite kernel and/or duplicate data points, the potential for cycling becomes an immediate

concern. However, as the following proposition shows, cycling will not occur when the degenerate data points are linearly dependent.

Proposition 2. *Cycling will not occur for the case where a set of degenerate data points are all linearly dependent or where upon adding any one data point to $\mathcal{F}$, the remaining data points have $\mathbf{q}_{i,\mathcal{F}} = \mathbf{u}^T Q_{\mathcal{F}}$ and $y_i = \mathbf{u}^T \mathbf{y}_{\mathcal{F}}$ for some vector $\mathbf{u}$.*

This is easy to show using the result from Theorem 4 since $y_i f(\mathbf{x}_i) = y_i f^{\ell}(\mathbf{x}_i) = 1$ for the data points not in $\mathcal{F}_{\ell}$ that are linearly related to the data points in $\mathcal{F}_{\ell}$. This also implies that $1 - y_i h_i^{\ell} = 0$. Therefore, once one of the linearly dependent data points is transitioned into $\mathcal{F}_{\ell}$, the remaining data points not in $\mathcal{F}_{\ell}$ will have $1 - y_i h_i^{\ell} = 0$ and will not be identified for transition. As shown later, the data point transitioned into $\mathcal{F}_{\ell}$ will also not immediately transition out of $\mathcal{F}$. Therefore, in the case where all of the degenerate data points are linearly dependent, cycling will not occur. This is an important result in that it shows that a semi-definite kernel can be used without risk of cycling as long as single transitions take place.

While this addresses the immediate concern regarding semi-definite kernels and duplicate data points, in general, cycling can still occur for even a positive definite kernel. That is, cycling can occur for a set of degenerate data points that are *not* linearly dependent. However, the occurrence of cycling is still limited by the semi-definiteness (or positive definiteness) of the problem and the fact that the matrix in (6.33) is non-singular. This fact limits the manner in which the quantities $1 - y_i h_i$ and δ_i can change signs as points are transitioned between sets at a fixed λ value as shown in the following.

Theorem 6. *Given any set of data points on the margin that are simultaneously switched between $\mathcal{F}$ and $\mathcal{L}^*$ or $\mathcal{R}^*$, if δ_i is computed when $i \in \mathcal{F}$ and correspondingly $g_i = 1 - y_i h_i$ when $i \notin \mathcal{F}$ it cannot be the case that $sgn(\delta_i) = -sgn(g_i)$ for all i.*

Proof. Given a set of data points in $\mathcal{F}$, $i \in \{1, ..., N\}$, to be transitioned out and a set of data points not in $\mathcal{F}$, $i' \in \{1, ..., N'\}$ to be transitioned into $\mathcal{F}$, prior to transitioning, solve the following

$$
\begin{pmatrix}
Q & \mathbf{q}_1 & \cdots & \mathbf{q}_N & \mathbf{y} \\
\mathbf{q}_1^T & q_{11} & \cdots & q_{1N} & y_1 \\
\vdots & & \ddots & & \vdots \\
\mathbf{q}_N^T & q_{N1} & \cdots & q_{NN} & y_N \\
\mathbf{y}^T & y_1 & \cdots & y_N & 0
\end{pmatrix}
\begin{pmatrix}
\boldsymbol{\delta} \\
\delta_1 \\
\vdots \\
\delta_N \\
\delta_0
\end{pmatrix}
=
\begin{pmatrix}
1 \\
1 \\
\vdots \\
1 \\
0
\end{pmatrix}
\tag{6.125}
$$

Defining $\Lambda_x \triangleq \begin{pmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_N \\ y_1 & y_2 & \cdots & y_N \end{pmatrix}$ and $\boldsymbol{\delta}_x = \begin{pmatrix} \delta_1 & \delta_2 & \cdots & \delta_N \end{pmatrix}$ this simplifies to

$$
\begin{pmatrix}
Q & \mathbf{y} \\
\mathbf{y}^T & 0
\end{pmatrix}
\begin{pmatrix}
\boldsymbol{\delta} \\
\delta_0
\end{pmatrix}
=
\begin{pmatrix}
1 \\
0
\end{pmatrix}
- \Lambda_x \boldsymbol{\delta}_x
\tag{6.126}
$$

and

$$
\Lambda_x^T \begin{pmatrix} \boldsymbol{\delta} \\ \delta_0 \end{pmatrix} + Q_x \boldsymbol{\delta}_x = 1
\tag{6.127}
$$

where Q_x is the portion of the kernel matrix indexed by x for those points initially in $\mathcal{F}$. For the points not in $\mathcal{F}$,

$$
\mathbf{q}_{\acute{1}}^T \boldsymbol{\delta} + q_{\acute{1}1} \delta_1 + \ldots + q_{\acute{1}N} \delta_N + y_{\acute{1}} \delta_0 = y_{\acute{1}} h_{\acute{1}}
\tag{6.128}
$$

$$
\vdots
$$

$$
\mathbf{q}_{\acute{N}}^T \boldsymbol{\delta} + q_{\acute{N}1} \delta_1 + \ldots + q_{\acute{N}N} \delta_N + y_{\acute{N}} \delta_0 = y_{\acute{N}} h_{\acute{N}}
$$

which can be rewritten as

$$\Lambda_{\acute{x}}^T \begin{pmatrix} \boldsymbol{\delta} \\ \delta_0 \end{pmatrix} + Q_{\acute{x}}\boldsymbol{\delta}_x = Y\acute{\mathbf{h}} \tag{6.129}$$

where Y is a diagonal matrix with diagonal entries containing the labels y_i and $Q_{\acute{x}x}$ is a portion of the kernel matrix with rows indexed by the data points not in $\mathcal{F}$ and columns indexed by points in $\mathcal{F}$. Upon swapping points between $\mathcal{F}$ and $\mathcal{L}^*$, $\mathcal{R}^*$, symmetry can be used to immediately yield the following for δ_i and $y_i h_i$

$$\begin{pmatrix} Q & \mathbf{y} \\ \mathbf{y}^T & 0 \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{\delta}} \\ \hat{\delta}_0 \end{pmatrix} = \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix} - \Lambda_{\acute{x}}\boldsymbol{\delta}_{\acute{x}} \tag{6.130}$$

$$\Lambda_{\acute{x}}^T \begin{pmatrix} \hat{\boldsymbol{\delta}} \\ \hat{\delta}_0 \end{pmatrix} + Q_{\acute{x}}\boldsymbol{\delta}_{\acute{x}} = \mathbf{1} \tag{6.131}$$

$$\Lambda_x^T \begin{pmatrix} \hat{\boldsymbol{\delta}} \\ \hat{\delta}_0 \end{pmatrix} + Q_{x\acute{x}}\boldsymbol{\delta}_{\acute{x}} = Y\mathbf{h}_{\acute{\mathbf{x}}} \tag{6.132}$$

Subtracting equations (6.126) from (6.130), (6.129) from (6.131), and (6.127) from (6.132) we have

$$\begin{pmatrix} Q & \mathbf{y} \\ \mathbf{y}^T & 0 \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{\delta}} - \boldsymbol{\delta} \\ \hat{\delta}_0 - \delta_0 \end{pmatrix} = -\Lambda_{\acute{x}}\boldsymbol{\delta}_{\acute{x}} + \Lambda_x\boldsymbol{\delta}_x$$

$$\Lambda_x^T \begin{pmatrix} \hat{\boldsymbol{\delta}} - \boldsymbol{\delta} \\ \hat{\delta}_0 - \delta_0 \end{pmatrix} + Q_{x\acute{x}}\boldsymbol{\delta}_{\acute{x}} - Q_x\boldsymbol{\delta}_x = Y\mathbf{h}_x$$

$$\Lambda_{\acute{x}}^T \begin{pmatrix} \hat{\boldsymbol{\delta}} - \boldsymbol{\delta} \\ \hat{\delta}_0 - \delta_0 \end{pmatrix} + Q_{\acute{x}}\boldsymbol{\delta}_{\acute{x}} - Q_{\acute{x}x}\boldsymbol{\delta}_x = 1 - Y\mathbf{h}_{\acute{\mathbf{x}}}$$

125

Defining $A \triangleq \begin{pmatrix} Q & \mathbf{y} \\ \mathbf{y}^T & 0 \end{pmatrix}$, $\boldsymbol{\gamma}_a \triangleq \begin{pmatrix} \hat{\boldsymbol{\delta}} - \boldsymbol{\delta} \\ \hat{\delta}_0 - \delta_0 \end{pmatrix}$, $\boldsymbol{\delta}_a \triangleq \begin{pmatrix} -\boldsymbol{\delta}_x \\ \boldsymbol{\delta}_{\acute{x}} \end{pmatrix}$ as well as $\Lambda_a \triangleq \begin{pmatrix} \Lambda_x & \Lambda_{\acute{x}} \end{pmatrix}$ we have

$$A\boldsymbol{\gamma}_a = -\Lambda_a \boldsymbol{\delta}_a$$

$$\Lambda_a^T \boldsymbol{\gamma}_a + Q_s \boldsymbol{\delta}_a = \begin{pmatrix} Y\mathbf{h_x} - \mathbf{1} \\ \mathbf{1} - Y\mathbf{h_{\acute{x}}} \end{pmatrix}$$

where $Q_s \triangleq \begin{pmatrix} Q_x & Q_{x\acute{x}} \\ Q_{\acute{x}x} & Q_{\acute{x}} \end{pmatrix}$ and we have

$$\left(Q_s - \Lambda_a^T A^{-1} \Lambda_a \right) \boldsymbol{\delta}_a = \begin{pmatrix} Y\mathbf{h_x} - \mathbf{1} \\ \mathbf{1} - Y\mathbf{h_{\acute{x}}} \end{pmatrix}$$

where the matrix on the l.h.s. is the Schur complement of $M = \begin{pmatrix} A & \Lambda_a \\ \Lambda_a^T & Q_s \end{pmatrix}$ which is a permutation of the KKT matrix when all points are placed in $\mathcal{F}$. In our case, this matrix is non-singular and indefinite with a single negative eigenvalue. If $A \in \Re^{m+1 \times m+1}$ and $Q_s \in \Re^{n \times n}$, then the inertia of A is $In(A) = \{m, 1, 0\}$ and $In(M) = \{m + n, 1, 0\}$. Using the Haynsworth inertia additivity formula [39], the inertia of the Schur complement or $In(M/A) = \{n, 0, 0\}$ or the Schur complement is positive definite. As a result,

$$\boldsymbol{\delta}_a^T \left(Q_s - \Lambda_a^T A^{-1} \Lambda_a \right) \boldsymbol{\delta}_a = \boldsymbol{\delta}_a^T \begin{pmatrix} Y\mathbf{h} - \mathbf{1} \\ \mathbf{1} - Y\mathbf{h_{\acute{x}}} \end{pmatrix} > 0 \tag{6.133}$$

or

$$\boldsymbol{\delta}_x^T \mathbf{g}_x + \boldsymbol{\delta}_{\acute{x}}^T \mathbf{g}_{\acute{x}} > 0 \tag{6.134}$$

126

which, by contradiction, cannot be the case if $sgn(\delta_i) = -sgn(g_i)$. $\square$

For example, this shows that when transitioning a single point, it will never be the case that $\text{sgn}(\delta_i) = -\text{sgn}(1 - y_i h_i^\ell)$ for δ_i computed when the point is in $\mathcal{F}$ and the quantity $1 - y_i h_i$ computed when the point is not in $\mathcal{F}$. This implies that once a single point is transitioned between sets, it will never be transitioned back out. For example, consider a data point with $\alpha_i = 1$ that is on the margin and in $\mathcal{L}^*$. To transition into $\mathcal{F}$, it must be that $(1 - y_i h_i) > 0$. Once the point is transitioned into $\mathcal{F}$, then it must the be case that $\delta_i > 0$.

It is more interesting to analyze the case where there are two or more degenerate data points. For the following analysis, without loss of generality, assume that all data points are on the margin with $\alpha_i = 1$ and are candidates for transitioning between $\mathcal{F}$ and $\mathcal{L}^*$. Binary notation will be used to denote the state of a data point with a 1 denoting a data point being within $\mathcal{F}$ and a 0 denoting a point being within $\mathcal{L}^*$. Additionally, use $\hat{0}$ to denote a point in $\mathcal{L}^*$ where $1 - y_i h_i > 0$ and $\check{0}$ to denote the quantity $1 - y_i h_i < 0$. Similarly, $\hat{1}$ denotes a point in $\mathcal{F}$ with $\delta_i > 0$ and $\check{1}$ for $\delta_i < 0$. The decoration on the bit will be referred to as the sign of the bit. The states $\hat{0}$ or $\hat{1}$ represent ones for which a transition should take place. Given N degenerate data points, a string of N binary digits are concatenated to represent the complete state. This notation is easily generalized to the case where $\alpha_i = 0$ by using $\hat{1}$ for instance to correspond to placing the data point in $\mathcal{F}$ with $\delta_i < 0$ instead of $\delta_i > 0$.

A proposed cycle involving 2 data points on the margin is illustrated in Figure 6.6. This does not represent a valid cycle since the existence of the states $\check{1}\hat{1}$ and $\hat{0}\check{0}$ would imply that $\text{sgn}(\delta_i) = -\text{sgn}(1 - y_i h_i)$ for both data points that are transitioned between these two states. In fact, all scenarios of 2 degenerate data points are easily enumerated, and, it can be shown that cycling will never occur.

127

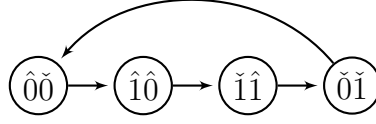$\hat{0}\breve{0} \rightarrow \hat{1}\hat{0} \rightarrow \breve{1}\hat{1} \rightarrow \breve{0}\breve{1}$

Figure 6.6: An example set of transitions for 2 margin data points. This candidate cycle is not valid as it violates Theorem 5. There is, in fact, no case where a cycle exists when only 2 data points are available for transition simultaneously.

$\breve{1}\hat{1}\breve{1} \rightarrow \breve{1}\hat{1}\breve{0} \rightarrow \breve{0}\breve{1}\hat{0} \rightarrow \breve{0}\breve{0}\hat{0} \rightarrow \hat{0}\hat{0}\hat{1} \rightarrow \hat{1}\hat{0}\hat{1}$
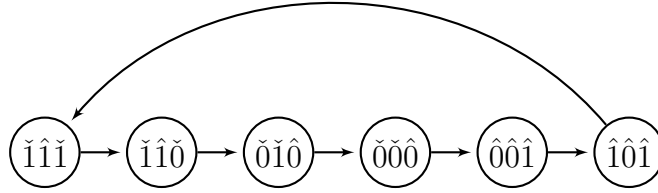
Figure 6.7: An example cycle for 3 margin data points. This is a valid cycle since in no instance is Theorem 5 violated.

The situation is a little different for 3 or more data points. For example, Figure 6.7 illustrates a valid cycle involving 3 data points. This cycle is found by observing that the sign does not change for all bits that change between any two selected states (not necessarily neighboring states). For example, the states $\breve{1}\hat{1}\breve{0}$ and $\hat{0}\hat{0}\hat{1}$ are valid states for the proposed cycle since the sign does not changes for the second bit. Fortunately, this cycle can be broken with the application of Bland's pivoting rule [4]. Bland's pivoting rule was originally applied to the simplex method for linear programming and works by breaking ties in the case of degeneracy. Specifically, the variable with the minimum index is chosen in the case of ties. Similarly, for the SSVMP algorithm, this entails selecting the minimum index in the case where more than one data point is available for transition. In the example of Figure 6.7, if the minimum index corresponds to the least significant digit of the binary representation, then the transition from the 3rd state to the 4th state would not occur. That is, instead of transitioning $\breve{0}\breve{1}\hat{0}$ to $\breve{0}\breve{0}\hat{0}$, the transition would have been $\breve{0}\breve{1}\hat{0}$ to $\breve{0}\hat{1}\hat{1}$, at which point,

128

no further transitions are identified and the cycle is broken.

In general, it can be shown that Bland's pivoting rule, when applied to the *SSVMP* algorithm, prevents cycling. Consider a set of transitions such that $\lambda^p = \lambda^{p+1} = \ldots \lambda^q$ with the sets $\mathcal{F}_p \neq \mathcal{F}_{p+1} \neq \ldots \mathcal{F}_q = \mathcal{F}_p$, and, similarly $\mathcal{L}_p^* \neq \mathcal{L}_{p+1}^* \neq \ldots \mathcal{L}_q^* = \mathcal{L}_p^*$ where, without loss of generality, all transitions are occurring between the sets $\mathcal{F}$ and $\mathcal{L}^*$. The following uses the result from Theorem 5 to show that Bland's pivoting rule prevents cycling. This represents a proof of convergence for the *SSVMP* algorithm when considering the proof of convergence for active set methods (see, for example, Nocedal [66]) along with the fact that cycling is avoided.

Theorem 7. *Cycling is avoided and the number of repeat events where $\lambda^\ell = \lambda^{\ell+1}$ is finite if the minimum index for transition is chosen in the case of ties.*

Proof. Given a set of N points involved in cycling, consider the data point with the highest index in the group, $k = N$. At some iteration, $p \leq u \leq q - 1$, the point leaves $\mathcal{F}$ and, for cycling to occur, re-enters $\mathcal{F}$ at some iteration $u + 1 \leq v \leq q$. At both iterations u and v the following must be satisfied for $i < N$, otherwise the pivoting rule would be violated

$$\alpha_i = 0, i \in \mathcal{F} \to \delta_i < 0 \tag{6.135}$$

$$\alpha_i = 1, i \in \mathcal{F} \to \delta_i > 0 \tag{6.136}$$

$$\alpha_i = 0, i \notin \mathcal{F} \to g_i > 0 \tag{6.137}$$

$$\alpha_i = 1, i \notin \mathcal{F} \to g_i < 0 \tag{6.138}$$

For any point that transitions between iterations u and v, this implies it must be the case $\text{sgn}(\delta_i) = -\text{sgn}(g_i)$. In addition, for the last point, N, either $\alpha_i = 0, \delta_i > 0$ or $\alpha_i = 1, \delta_i < 0$ at iteration u and $\alpha_i = 0, g_i < 0$ or $\alpha_i = 1, g_i > 0$ at iteration v which implies that $\text{sgn}(\delta_i) = -\text{sgn}(g_i)$ for the last point as well. This implies $\text{sgn}(\delta_i) = -\text{sgn}(g_i)$ for all i in $\mathcal{F}$ at iteration u and not in $\mathcal{F}$ at

129

iteration v, or vice versa, which cannot occur. □
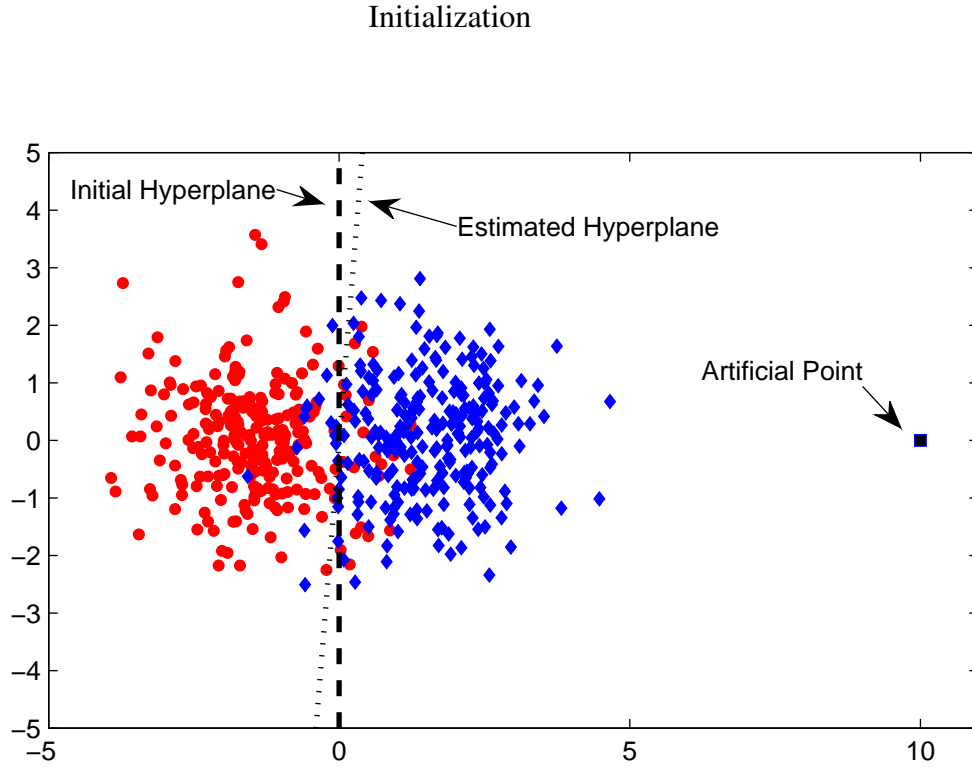
Figure 6.8: Example dataset containing unequal class sizes. The initialization method adds duplicate artificial data points to the smaller class that are beyond the most extreme data point for that class. The error between the approximate hyperplane (dotted) versus the actual initial hyperplane (dashed line) is shown.

As shown previously, a simple closed form solution exists for the initial or starting solution in the case of equal class sizes. However, in the case of unequal class sizes an external solver is often required to find the solution at some arbitrarily large λ value. The problem formulation posed in [Lemma 2, Hastie [38]] may also be directly solved to find the initial solution. The goal, here, is to provide an initialization method that can work from within the framework of the path following algorithm without the need for an external solver.

The two-phased method (*e.g.* see Nocedal [66]) is often used in linear programming (simplex method) to find an initial feasible solution when it cannot be trivially found. The two-phased method works by augmenting the original problem with a set of auxiliary variables from which an initial feasible solution can be trivially found. Optimization is broken into two phases with the task, during phase I, of optimizing out the auxiliary variables. At that point, an initial feasible solution to the original problem is found and optimization can continue on the original problem during phase II.

This is the inspiration for a new initialization routine for the regularization path following algorithm. Instead of auxiliary variables, artificial data points are added to create equal class sizes from which a trivial starting solution can be found. The artificial data points are designed so that regularization path following algorithm will initially optimize out the artificial variables (all are in $\mathcal{R}^*$). Once this occurs, the starting solution for the original problem is found and the algorithm continues computing the desired regularization path in phase II.

Fortunately, the SSVMP algorithm is capable of working with duplicate data points. Therefore, a single artificial data point can be created and duplicated to create equal class sizes. The artificial data point must be chosen so that no other data points within the same class transitions out of $\mathcal{L}^*$ until $\alpha_i = 0$ for all of the artificial data points. One method for doing this is to place the artificial data point beyond the most extreme data point for the corresponding class relative to the initial separating hyperplane, or without loss of generality, $\tilde{\mathbf{w}}^T \phi(\mathbf{x_t}) > \max_{i \in \mathcal{I}+} \tilde{\mathbf{w}}^T \phi(\mathbf{x}_i)$ for $N_+ < N_-$ where $\mathbf{x_t}$ is the artificial data point, $\tilde{\mathbf{w}}$ is the initial separating hyperplane, and $\phi(\mathbf{x}_i) : \mathbf{x}_i \in \mathbb{R}^d \to \mathcal{H}$, Of course, $\tilde{\mathbf{w}}$ is not known *a priori*. However, the quantity $\mathbf{w}^* = \sum_i y_i \phi(\mathbf{x}_i)$, provides a reasonable estimate of the initial hyperplane especially when $|N_- - N_+|$ is much less than $N_- + N_+$. The artificial data point is then placed at a large distance along the vector $y_t \mathbf{w}^* / \|\mathbf{w}^*\|$ where y_t is the label of the artificial data point. Therefore, the artificial data point is placed according to $\phi(\mathbf{x}_t) = y_t \rho \mathbf{w}^*$ where ρ is used to specify the extent of the displacement.

The displacement ρ needs to be chosen so that the artificial data point is well beyond the most extreme data point for the smaller class size allowing for an additional margin for the error in estimate $\mathbf{w}^*$. Care must also be taken to ensure ρ is not too large as this can result in poor scaling of the kernel matrix and numerical instability. As a safety mechanism, the algorithm can detect and report when a data point from the smaller class enters the margin prior to all of the artificial data points leaving the margin. Instead of finding $\mathbf{x}_t$, the kernel matrix can be augmented according to $K_{it} = \rho y_t \sum_j y_j K_{ij}$ and $K_{tt} = \rho^2 \sum_{i,j} y_i y_j K_{ij}$. Since duplicate data points are being used, memory consumption is mitigated by augmenting the kernel matrix with a single row/column.

Figure 6.8 illustrates an example of the initialization method with the artificial data point. The estimated hyperplane $\mathbf{w}^*$ as well as a notional example of the initial hyperplane $\tilde{\mathbf{w}}$ are shown. While there is a small error in the estimated hyperplane, the estimate is still useful for placing the artificial data point well beyond the most extreme data point for the class. Initially, one of the margins will be pinned to the set of the artificial data points. As λ is decreased, the opposing margin will translate as well as both margins will rotate. The artificial data point should be placed sufficiently far so that other data points within the same class do not enter the margin until all of the duplicate data points are no longer on the margin but outside the margin.

Efficient Implementation

The KKT system in Equation (6.33), can be solved using the null-space method [66] and Cholesky factorization exactly as prescribed in Chapter 5.

Table 6.4: Datasets Used for SSVMP Performance Comparisons

Dataset	N	N_+	d
adult-1a	1605	395	123
abalone	4177	2081	10
australian	690	307	14
diabetes	768	500	8
german	1000	300	24
heart	270	150	13
ionosphere	351	225	34
sonar	208	97	60
spam	4601	1813	57
splice	1000	517	60
svmguide1	3089	2000	4
wdbc	569	212	30
web-1a	2477	2405	300

Results and Discussion

Experiments were performed with the *SSVMP* and *ISVMP* algoritihms on a Windows 2003 8-core x64 server workstation with 16 GBytes of RAM. Both algorithms were implemented within the MATLAB R2011a environment with multi-core support enabled. The *ISVMP* algorithm was obtained from the website referenced in [67].

The datasets listed in Table 6.4 were chosen to provide a wide range of complexity and sizes. Datasets such as the *adult-1a* and *web-1a*, for example, possess sparse features and represent a significant challenge for the regularization path-following algorithms. The dataset *web-1a* was obtained from the SVM website at Microsoft Research [70], *abalone*, *australian*, *diabetes*, *german*, *heart*, *ionosphere*, *sonar*, *spam* and Breast Cancer Wisconsin Diagnostic *wdbc* were obtained from

the UCI repository [2], and, finally, *splice*, *adult-1a*, and *svmguide1* were obtained from the LIB-SVM website [12] where many of the UCI repository datasets can also be obtained. The features for each dataset were normalized to have zero mean and unit variance.

The *ISVMP* algorithm was configured to use default tolerances including a stopping criterion of 10^{-6} while *SSVMP* was configured with a stopping tolerance of 10^{-6}. Both algorithms were configured to stop computing the regularization path when $\lambda < 10^{-3}$. The *ISVMP* algorithm was configured to start the path at $\lambda = 10^6$ while *SSVMP* used the proposed initialization method, with $\rho = 0.01$. In both cases, the kernel cache was made large enough to accommodate storing the entire kernel matrix. This was done to avoid comparing kernel caching strategies and to highlight the differences in performance associated with computing the actual regularization path.

The results are summarized in Table 6.5 and Table 6.6. The time associated with computing the regularization path, number of events, initialization time, as well as the number of repeat events is reported for *SSVMP*. The path time, initialization time, number of events as well as the number of times the backup routine is invoked is reported for the *ISVMP* algorithm. The path time in both cases does not include the initialization time.

The *SSVMP* algorithm remained competitive with the *ISVMP* implementation in almost all cases being faster by several factors in some cases and by orders in magnitude in others. For example, with the *adult-1a* dataset and *web-1a* datasets, the *SSVMP* algorithm was more than a factor of 1000 times faster. The *ISVMP* fails to compute the path in one case for the *heart* dataset and RBF kernel while the *SSVMP* algorithm was successful in all cases.

The *ISVMP* algorithm appears to be significantly slower when the backup routine is invoked multiple times, a sign of instability. The backup routine is used as a recovery technique when a violation of the KKT conditions or cycling is detected signifying a breakdown in the path computation. When this occurs, the *ISVMP* algorithm proceeds backwards along the path to the last known good

point and invokes an external solver to restart the path. While appearing to adequately recover in terms of finding the correct solution, there is a significant cost in terms of computational time. Interestingly, this appears to occur most frequently for the linear kernel and the *adult-1a* and *web-1a* datasets, which contain sparse features. With sparse features, the number of opportunities for generating a singular KKT matrix greatly increases. These results demonstrate that the *SSVMP* algorithm can handle these cases without introducing instability, and, although a similar backup routine can be implemented with the *SSVMP* algorithm, it was found to be unnecessary. Surprisingly, the backup routine is invoked a number of times by the *ISVMP* algorithm for the RBF kernel as well. While there should be no occurrence of singular matrices in this case, it is possible there are some issues due to numerical instability. Finally, it is of interest to note that the number of events reported for both algorithms are nearly the same even when path breakdowns occur for the *ISVMP* algorithm.

The accuracy of *SSVMP* was measured by comparing the objective value obtained at each breakpoint with that obtained by running a conventional SVM training algorithm at the same $\lambda = \frac{1}{C}$ values. The *SVM-RSQP* algorithm was used and configured with a stopping tolerance of 10^{-6}. The relative precision was measured as $| \left(o_{SSVMP} - o_{ref} \right) / o_{ref} |$, where o_{ref} is the objective function value for the reference algorithm, and $o = \frac{1}{2\lambda} \boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha}$, similar to as done in [67]. Figures 6.9, 6.10, 6.11, and 6.12 show the results for the case of the linear kernel and RBF kernel, respectively. In all cases, the precision is maintained to within 10^{-6} with higher precision maintained for the RBF kernel, on average. The worst-case accuracy was seen for the *web-1a* dataset where the accuracy just began to exceed the 10^{-6} precision during the last few iterations. As a comparison, the worst-case relative accuracy reported in [67] for *ISVMP* was on the order of 10^{-3}.

It was found that the regularization path algorithm can begin to fail when attempting to compute the path for $\lambda < 10^{-3}$. This was the case for both *ISVMP* and *SSVMP*. Failures begin to occur when $\lambda^\ell - \lambda^{\ell+1}$ approaches the algorithm tolerance, in this case 10^{-6}.
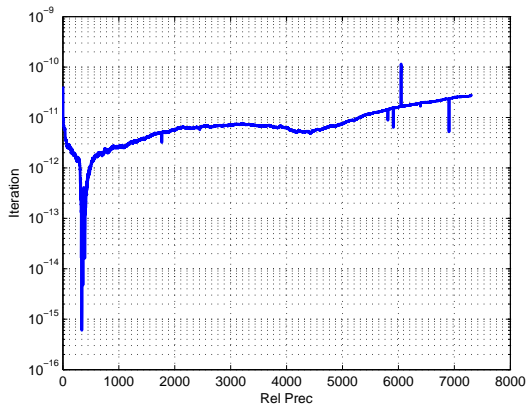
(a) abalone
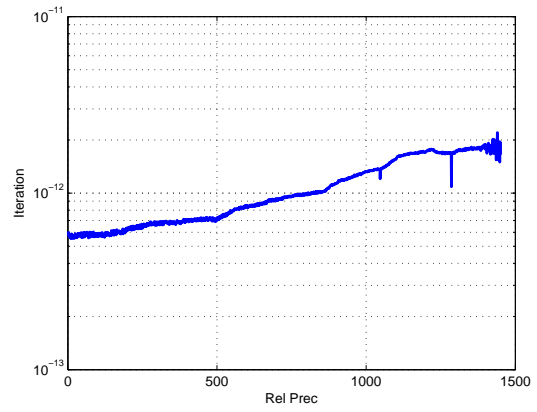
(b) adult
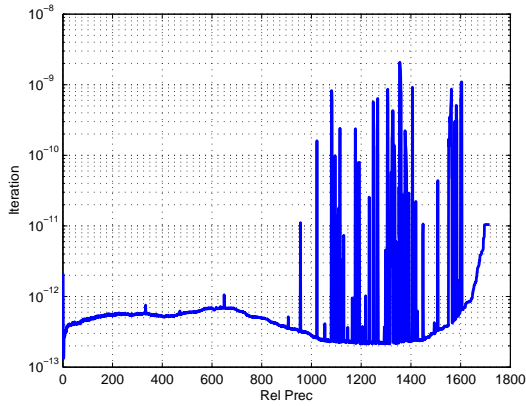
(c) australian
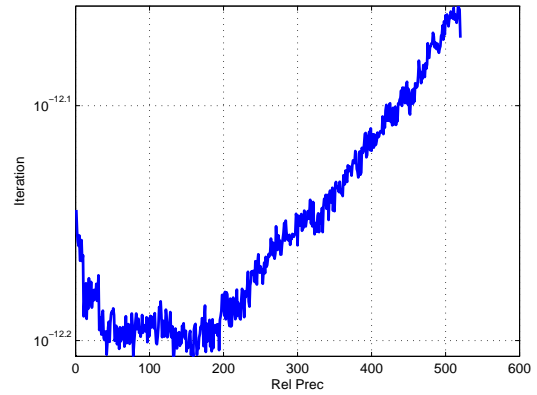
(d) diabetes

(e) german

(f) heart

Figure 6.9: Simple SVMPath path accuracy for the linear kernel
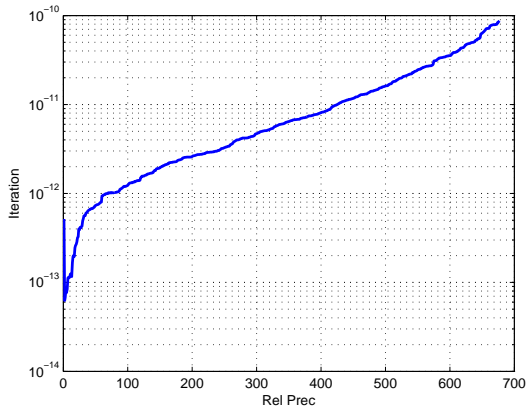
(a) ionosphere
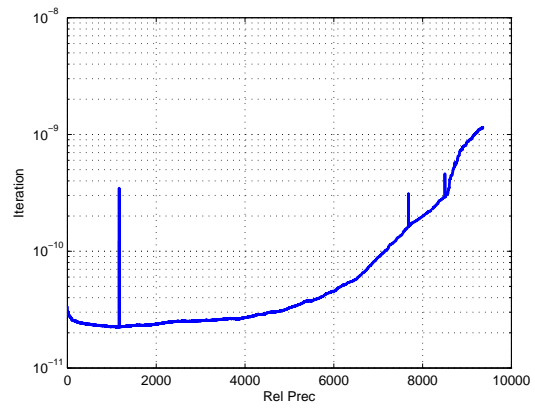
(b) spam
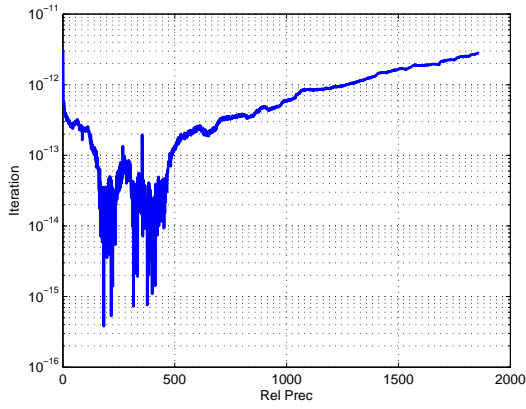
(c) splice
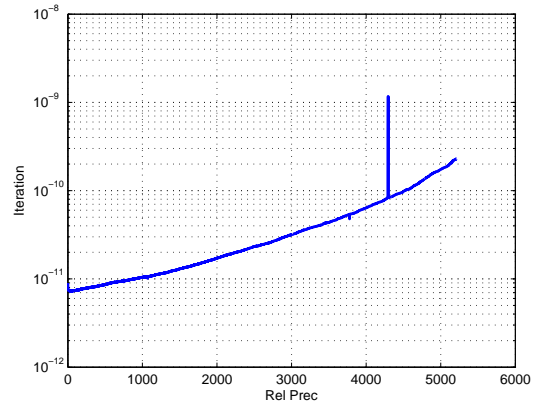
(d) svmguide1
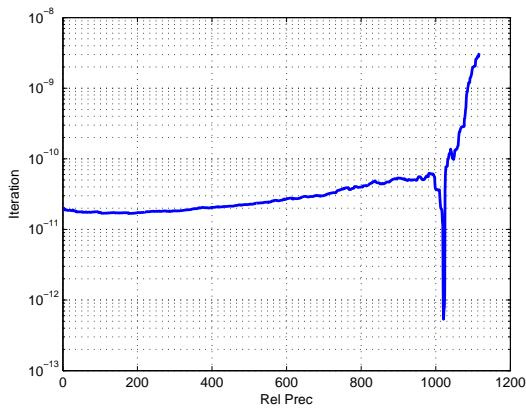
(e) wdbc
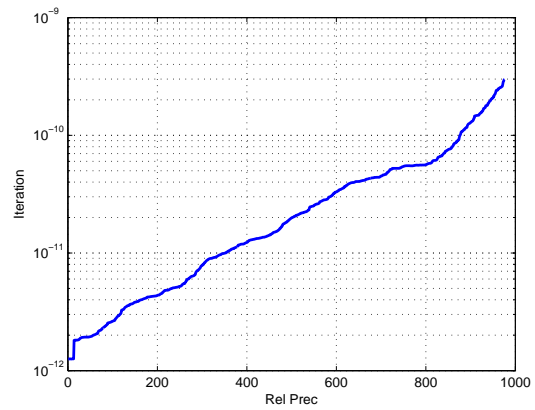
(f) web-1a

Figure 6.10: Simple SVMPath path accuracy for the linear kernel
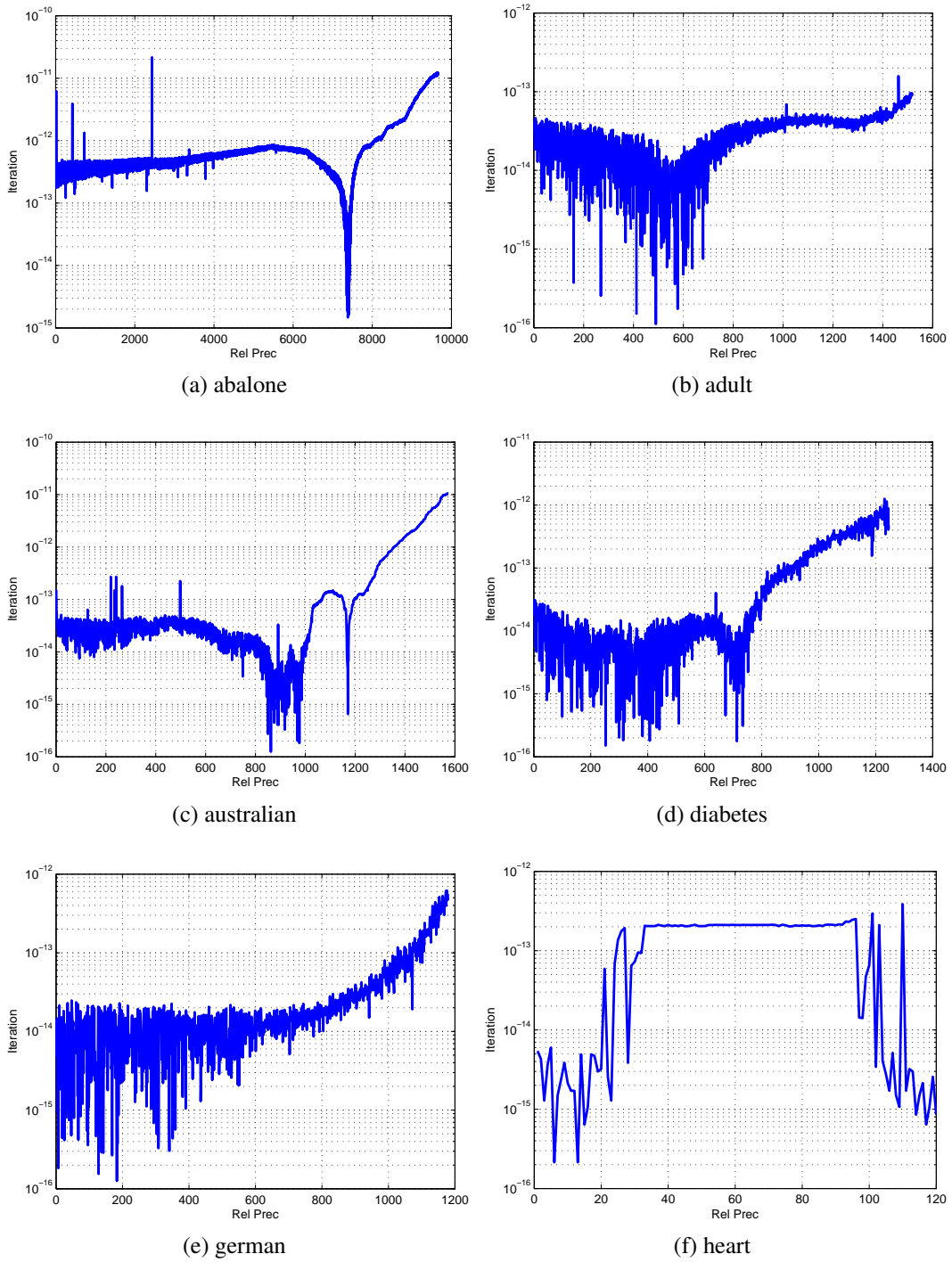
Figure 6.11: Simple SVMPath path accuracy for the RBF kernel

138
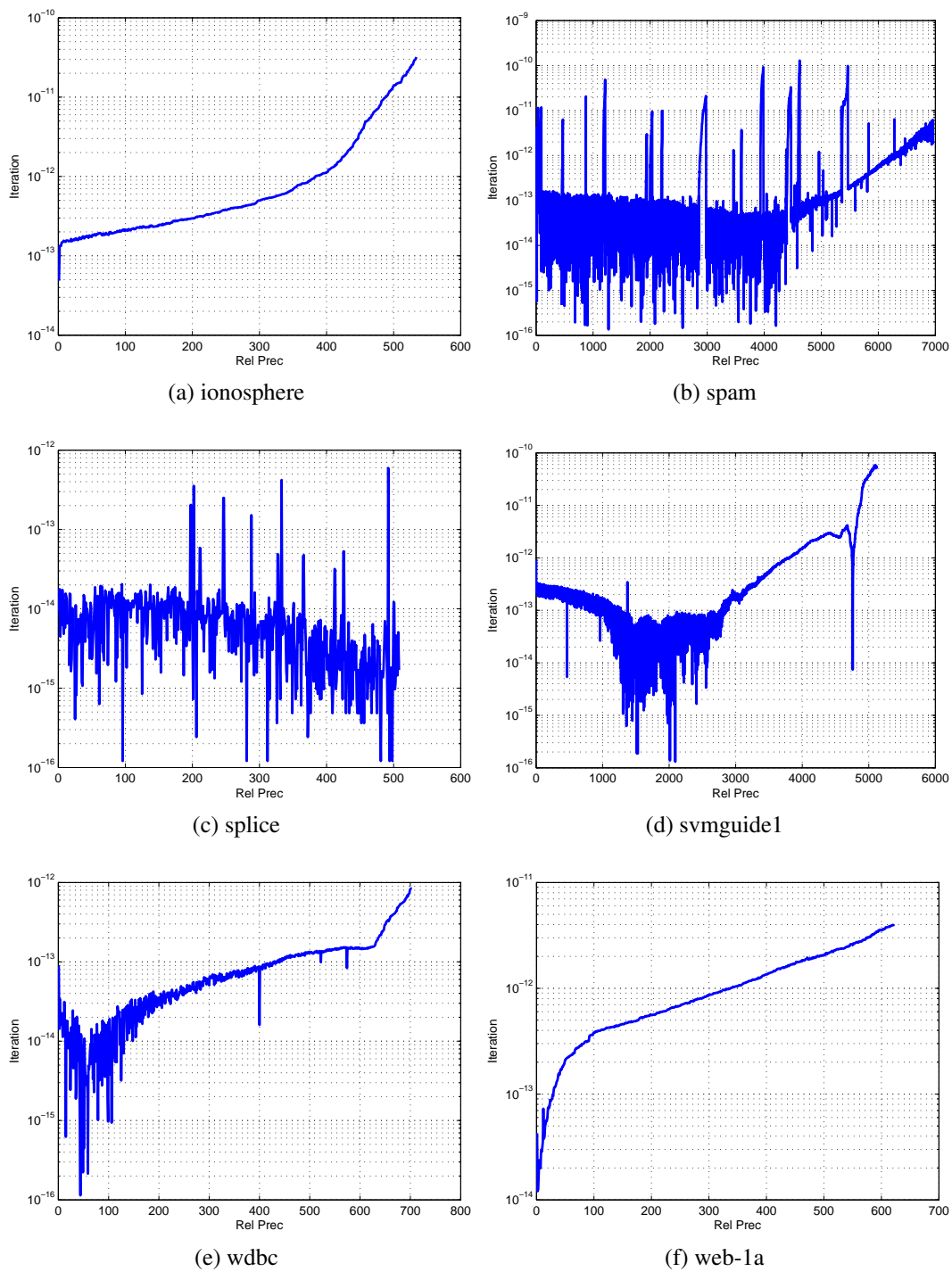
Figure 6.12: Simple SVMPath path accuracy for the RBF kernel

Table 6.5: Linear Kernel Results

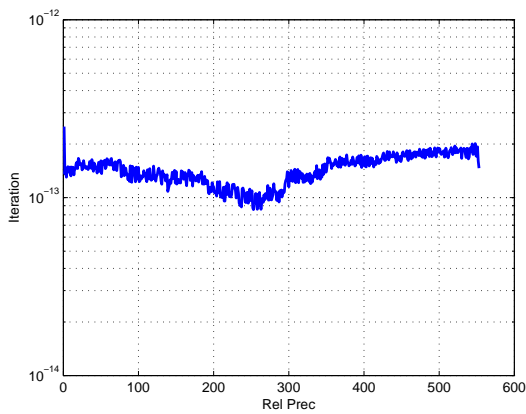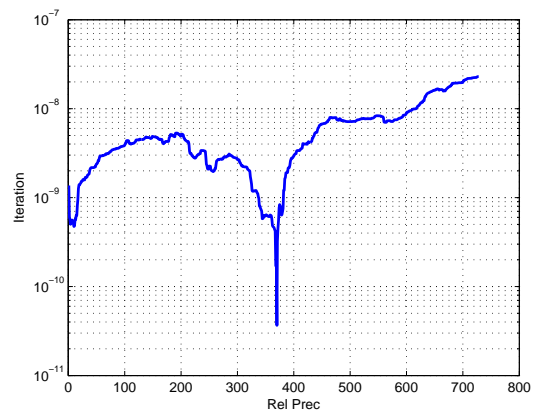|	SSVMP				ISVMP			
Dataset	Path Time (sec)	Init Time (sec)	Num Events	Num Repeat	Path Time (sec)	Init Time (sec)	Num Events	Num BR
abalone	54.63	0.79	7301	451	80.98	1.27	6803	0
adult-1a	8.74	14.86	1452	7	19568.18	3.32	1428	14
australian	2.93	0.52	1719	3	5.70	0.09	1762	0
diabetes	1.03	1.71	522	0	1.74	0.36	522	1
german	1.66	5.07	555	0	60.53	0.86	507	9
heart[a]	0.67	0.22	730	2	1.67	0.03	776	0
ionosphere	0.79	0.42	679	2	2.13	0.07	677	1
sonar	0.48	0.12	442	0	2.01	0.01	522	0
spam	107.19	37.67	9359	338	984.27	24.52	9125	9
splice	5.27	0.28	1862	18	13.26	0.26	1844	0
svmguide1	34.40	23.21	5213	182	55.79	17.03	5149	1
wdbc	1.58	0.81	1119	15	3.11	0.13	1105	0
web-1a	16.30	66.70	978	35	2855.38	7.48	983	37

[a] Used $\rho = .1$ for initialization.

Figures 6.13 and 6.14 depicts the behavior of $\lambda^{\ell} - \lambda^{\ell+1}$ as the path algorithm progresses for all of the datasets for the linear kernel and RBF kernel, respectively. As can be seen, $\Delta\lambda$ tends to zero as progress is made along the path. Note that the algorithm experiences a number of repeat events with the *heart* dataset and RBF kernel, which results in $\Delta\lambda$ approaching 10^{-12}. Repeat events do not result in path breakdown. In general, as $\Delta\lambda$, approaches the algorithm tolerance and disregarding repeat events, the path following algorithm may begin to fail. This suggests there is a practical limit on the minimum λ, which can be computed due to numerical precision.

Table 6.6: RBF Kernel Results, $\gamma = 0.1$

	SSVMP				ISVMP			
Dataset	Path Time (sec)	Init Time (sec)	Num Events	Num Repeat	Path Time (sec)	Init Time (sec)	Num Events	Num BR
abalone	136.79	0.81	9671	30	269.39	1.18	9672	1
adult-1a	117.36	15.17	1520	10	280.95	3.81	1525	11
australian	5.66	0.50	1575	0	13.82	0.09	1575	1
diabetes	16.43	1.85	1248	0	33.18	0.36	1246	1
german	69.74	5.36	1181	0	116.88	0.90	1184	11
heart	1.55	0.61	122	56	1.06	0.04	2[a]	1
ionosphere	1.00	0.40	536	1	2.94	0.07	535	1
spam	3671.02	47.23	6959	229	24266.74	29.96	7152	15
sonar	0.83	0.07	158	0	2.49	0.01	170	2
splice	98.31	20.05	510	21	131.40	0.39	486	2
svmguide1	34.61	25.42	5120	87	51.31	18.33	5030	1
wdbc	3.37	0.83	705	0	7.57	0.14	704	1
web-1a	81.54	104.98	624	12	545.87	4.41	674	11

[a] Failed to compute path

A potential concern with the *SSVMP* algorithm is that enforcing single transitions may result in a large number of repeat events and resultant inefficiency. In most cases, the number of repeat events, where $\lambda^{\ell} = \lambda^{\ell+1}$, was a small fraction of the total number of events-typically less than 6 percent of the total events. In one case for the RBF kernel and *heart* dataset, nearly half of the 122 events were repeat events. This appears to be a highly pathological case as evidenced by the fact that *ISVMP* fails to converge. On the other hand, *SSVMP* does not experience cycling and converges to an accurate solution even in this case. Overall, the occurrence of repeat events does not prevent convergence, as expected, and does not appear to affect the overall efficiency in terms of training times.

Figure 6.13: Depiction of the quantity $\Delta\lambda$ as the path following algorithm progresses for several datasets where the linear kernel is used. Log scale is used for both axes. Smoothing has been applied to the raw data to highlight trends. The path stops when $\lambda = 10^{-3}$.

The initialization method was successful in all cases as indicated by the worst-case precision reported for all of the datasets. Failures are also detected when points begin to transition out of $\mathcal{R}^*$ having the same label as the artificial data points prior to all of the artificial data points leaving $\mathcal{R}^*$. While $\rho = 0.01$ was used in the majority of cases, this was changed to $\rho = 1.0$ for the *heart* dataset and linear kernel. Overall, the initialization method appears to be robust in terms of setting ρ. In general, ρ should be chosen as high as possible; however, setting ρ to high can result in numerical scaling issues and instability.

Figure 6.14: Depiction of the quantity $\Delta\lambda$ as the path following algorithm progresses for several datasets where the RBF kernel is used. Log scale is used for both axes. Smoothing has been applied to the raw data to highlight trends. The path stops when $\lambda = 10^{-3}$.

In most cases, the initialization algorithm is found to be slower than the SMO algorithm used by *ISVMP* for initialization. This is not entirely unexpected since the SMO algorithm is known to be highly efficient and possesses speed-up strategies such as shrinking. The time required for initialization is governed by the number of events occurring during initialization, which is on the order of $|N_- - N_+|$ and observed to be between 4 and 5 times $|N_- - N_+|$. For example, the *adult-1a* dataset has 395 positive examples out of a total of 1601 data points while the *abalone* dataset has 2081 positive examples out of 4177 total data points. As expected, the initialization time required for the *adult-1a* dataset is much higher. However, the initialization time is generally a small fraction

of the total time, and the results indicate that total training time, including initialization, is generally faster for the *SSVMP* algorithm in all cases.

A key advantage to the proposed initialization method is that the true initial solution is found, that is, the solution described by Lemma 2 [38]. The *ISVMP* algorithm, instead, starts with a solution for some arbitrarily large value of λ. If not selected large enough, a part of the regularization path may not be computed. Another key advantage, of course, is that the *SSVMP* algorithm is completely self-contained and does not rely on an external solver for initialization. This may be particularly important when considering numerical precision. For example, the *ISVMP* algorithm must "interpret" the result obtained from the SMO algorithm when determining how to assign variables to $\mathcal{E}$, $\mathcal{L}$, and $\mathcal{R}$. This is done by comparing each component α_i with a set of tolerances. This can result in an overestimate of the number of components within $\mathcal{E}$ and unnecessarily results in large, singular KKT matrices. A similar issue arises each time the SMO algorithm is invoked as part of a backup routine. In summary, use of an external solver can result in loss of efficiency as well as loss of precision. On the other hand, initialization with the SMO algorithm may still be a desirable choice for very large datasets when there is a large class size imbalance and as initialization time becomes a significant fraction of the total training time.

CHAPTER 7: SOLVING THE APPROXIMATE PATH USING SVM-RSQP

The regularization path-following algorithms while successful, can still experience numerical difficulty as discussed in Chapter 6. In addition, recently introduced approximate or sub-optimal regularization path-following algorithms such as in Karasuyama *et al.* [44] and Giesen *et al.* [30] cannot be used with semi-definite kernels for the ℓ_1-loss SVM formulation. As an alternative, the conventional active set method can be modified, using warm starts, to allow incremental updates to the solution as the regularization path and/or kernel path is explored. Generally, this is done within the context of a grid search where a series of user-specified regularization or kernel parameter values are explored. In this scenario, for example, the solution at a previous regularization value is used as an initial solution for the next value as described in [84], [87], and [29]. A similar approach referred to as "alpha-seeding" is also described by DeCoste *et al.* [21] where LIBSVM and SVMLight are modified to allow warm-starts. A criticism against the grid search approach is that the solution behavior between arbitrarily selected regularization values is unknown [30], [44]; therefore, there is no guarantee on optimality of the regularization parameter chosen via grid search.

However, rather than simply solving the regularization and/or kernel path across a series of user-specified values, the conventional active set method, with warm-starts, can be modified to compute the approximate regularization path. The approximate path can be efficiently computed since the same KKT matrix already maintained by the active set method is employed to compute the path. Note that while algorithms such as SMO and SVMLight support warm-starts, it is not clear whether the approximate path can be efficiently computed in these methods since the KKT system is not maintained. The conventional active set method has the advantage of being able to use semi-definite kernels, possesses improved numerical stability, and benefits from the applicability of speed-up mechanisms such as shrinking and sprinting as well as kernel caching.

In this research, the SVM-RSQP algorithm is modified to compute the approximate regularization path and kernel path. The SVM-RSQP algorithm is also modified to incorporate low-rank kernel approximations as a speed-up mechanism. This represents the first known implementation of an approximate regularization path algorithm for semi-definite kernels (for the ℓ_1-loss SVM), implementation of an approximate kernel path algorithm, and implementation of a low-rank kernel approximation within an active set method. In the latter case, while low-rank kernel approximation methods have been widely reported (e.g., see [27], [43]), active set implementations have most likely been avoided due to difficulties with semi-definite kernels.

The remainder of this chapter will discuss warm start strategies for the revised simplex method (SVM-RSQP), the approximate regularization path, the approximate kernel path, low-rank kernel approximations, and limiting behaviors of the regularization and kernel path. The chapter ends with detailed results and discussion.

Warm Start for the Revised Simplex Method

The active set method is advantageous, in general, since it allows warm-starts. In this instance, the goal is to employ warm-starts to allow faster exploration of the regularization parameter and kernel parameter spaces. The revised simplex method for quadratic programming (SVM-RSQP) requires an initial solution that is not only feasible but also satisfies the complementary conditions. The goal, then, is to find a way of scaling or adjusting the solution between regularization or kernel parameter values so that the initial solution in each case is feasible and satisfies the complementary conditions.

In the case of the regularization parameter C, given a solution at some C_ℓ one possible method for initializing the solution at $C_{\ell+1}$ is to simply scale all of the components α_i such that $\alpha' = \alpha \frac{C_{\ell+1}}{C_\ell}$.

This automatically satisfies the feasibility conditions including the equality constraint $\mathbf{y}^T\boldsymbol{\alpha} = 0$. This is quite simple to implement and does not require adjustment of the sets $\mathcal{I}_o, \mathcal{I}_s, \mathcal{I}_c$. Most importantly, the set of non-bound support vectors, $\mathcal{I}_s$, remains constant, and, therefore, there is no required book-keeping or updates to the Cholesky factorization.

However, the complementary conditions become violated after this scaling. Specifically, the pricing variables or components $\boldsymbol{\delta}_s$, corresponding to the set of non-bound support vectors, are no longer zero. This needs to be corrected before the active set method can continue. It is straight-forward to show that the following descent direction will correct the complementary conditions

$$
\begin{pmatrix} Q_{ss} & \mathbf{y}_s \\ \mathbf{y}_s^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ g \end{pmatrix} = \begin{pmatrix} -\boldsymbol{\delta}_s \\ 0 \end{pmatrix}
\tag{7.1}
$$

where the quantity $\boldsymbol{\gamma} = Q\mathbf{h} - g\mathbf{y}$ represents the descent direction for $\boldsymbol{\delta}_s$. Updates are computed as $\boldsymbol{\alpha}_s \leftarrow \boldsymbol{\alpha}_s - \theta\mathbf{h}$, $\beta \leftarrow \beta - \theta g$, and $\boldsymbol{\delta}_s \leftarrow \boldsymbol{\delta}_s - \theta\boldsymbol{\gamma}$. The computed direction is followed until $\boldsymbol{\delta}_s = 0$ or some variable α_i in the set of non-bounds support vectors $i \in \mathcal{I}_s$ becomes infeasible. That is, θ is found as follows

$$
\theta = \min\left\{ \frac{\alpha_i}{h_i} \middle| h_i > 0, \frac{C - \alpha_i}{-h_i} \middle| h_i < 0, 1.0 \right\}
\tag{7.2}
$$

If $\theta = 1.0$, the desired solution has been reached; otherwise, a variable is removed from $\mathcal{I}_s$, and the quantities $\boldsymbol{\alpha}_s$, $\boldsymbol{\delta}_s$, and β are updated using the update formulas and θ from above. The process is then repeated until $\max |\delta_i| < \epsilon, i \in \mathcal{I}_s$, where ϵ is a floating point tolerance. The complete algorithm for adjusting the complementary conditions is listed in Algorithm 4.

Similarly, a warm-start can be provided for a new kernel parameter value. In this case, the components, δ_i, are updated according to the new kernel parameter. This requires a reset of the kernel cache, if in use, and a full recompute of the pricing variables δ_i. Therefore, initialization at a new kernel parameter is more costly than for a new regularization parameter.

Algorithm 4 Fixing the Complementary Condition

1: **while** $\max |\delta_i| > \epsilon \quad \forall i \in \mathcal{I}_s$ **do**

2:     solve $\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_s} \\ g_\beta \end{pmatrix} = \begin{pmatrix} -\boldsymbol{\delta}_s \\ 0 \end{pmatrix}$

     $\boldsymbol{\gamma} \leftarrow Q_{ss}\mathbf{h}_s - g\mathbf{y}_s$

3:     $\theta = \min \left\{ \frac{\alpha_i}{h_i} \middle| h_i > 0, \frac{C-\alpha_i}{-h_i} \middle| h_i < 0, 1 \right\}$

4:     **if** $\theta < 1$ **then**

5:        Adjust the sets $\mathcal{I}_s, \mathcal{I}_c, \mathcal{I}_o$.

6:     **end if**

7:     $\boldsymbol{\delta}_s \leftarrow \delta_s - \theta\boldsymbol{\gamma}$

8:     $\beta \leftarrow \beta - \theta g_\beta$

9:     $\boldsymbol{\alpha_s} \leftarrow \alpha - \theta\mathbf{h_s}$

10: **end while**

Here, again, the complementary conditions become violated when updating the kernel parameter and must be corrected; the same procedure described in Algorithm 4 is used here as well. Algorithm 5 describes the warm-start strategy when updating the kernel parameter.

Algorithm 5 Computing Warm Start for a New Kernel Parameter

1: Reset or recompute kernel cache

2: Compute $\boldsymbol{\delta}_s = -\mathbf{1} - \beta\mathbf{y}_s + Q_{os}\boldsymbol{\alpha}_s + Q_{oc}\boldsymbol{\alpha}_c$

3: Fix the complementary condition according to Algorithm 4

4: Update $\delta_i, i \in \mathcal{I}_o, \mathcal{I}_c$

Computing the Approximation Regularization Path

Instead of providing a series of predetermined C values, the goal of the approximate path method is to find a series of C values where optimality is maintained within some user-specified tolerance between consecutive C values. In essence, the revised simplex method will be used to solve the approximate regularization path. Similar to the regularization path-following algorithm, given a solution at a particular value C value, the solution at a new value C' is easily found, given the set of active constraints do not change. Recall that given a set of basic variables $\mathbf{x}_B$ where $x_i > 0$ for

$i \in B$, the solution is computed as

$$\begin{pmatrix} Q_B & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}_B \\ \boldsymbol{\pi} \end{pmatrix} = \begin{pmatrix} \mathbf{p}_B \\ \mathbf{b} \end{pmatrix} \tag{7.3}$$

For the SVM problem, the quantity $\mathbf{p}_B$ is constant while $\mathbf{b}$ is a function of C. Therefore, $\mathbf{h}$ and g are found as

$$\begin{pmatrix} Q_B & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{b}(\mathbf{C'}) - \mathbf{b}(\mathbf{C}) \end{pmatrix} \tag{7.4}$$

For the SVM optimization problem, it is easy to show this becomes

$$\begin{pmatrix} Q_{ss} & \mathbf{y}_s \\ \mathbf{y}_s^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h}_s \\ g_\beta \end{pmatrix} = \begin{pmatrix} Q_{sc}\mathbf{1} \\ -\mathbf{y}_c^T\mathbf{1} \end{pmatrix} (C' - C) \tag{7.5}$$

where $\boldsymbol{\alpha}_s \leftarrow \boldsymbol{\alpha}_s + \mathbf{h}$, $\beta \leftarrow \beta + g_\beta$. The above equation can be efficiently solved since the Cholesky factorization of the matrix is already maintained by the revised simplex method. Recall the optimality conditions for the SVM formulation,

$$\mathbf{y}^T\boldsymbol{\alpha} = 0 \tag{7.6}$$

$$0 \leq \alpha_i \leq C \ \forall i \tag{7.7}$$

$$\delta_i > 0 \ \forall i \tag{7.8}$$

$$\delta_i \alpha_i = 0 \tag{7.9}$$

The goal is to find the next C value for which the optimality conditions are violated by some tolerance τ. In this case, since the active set does not change, the constraint $\mathbf{y}^T\boldsymbol{\alpha} = 0$ will be satisfied as well as the complementary condition $\alpha_i\delta_i = 0$ regardless of C. The τ-optimal conditions are

then,

$$-\tau \leq \alpha_i \leq C + \tau \ \ \forall i \in \mathcal{I}_s \tag{7.10}$$

$$\delta_i > -\tau \ \ \forall i \in \mathcal{I}_c \cup \mathcal{I}_o \tag{7.11}$$

where τ is referred to as the path tolerance. Using the expression in Equation (7.5), for the variables in $\mathcal{I}_s$, the relationship for α_i as a function of $C' - C$ becomes

$$\alpha_i \leftarrow \alpha_i + (C' - C)h_i \tag{7.12}$$

For components where $h_i > 0$, violation of optimality occurs when $\alpha_i + (C' - C)h_i > C' + \tau$. Note that when $h_i < 1$ then $\alpha_i + (C' - C)h_i \leq C'$ for all C'. Consider, for example, the case where $\alpha_i = C$, then $C + (C' - C)h_i \leq C'$ or $(C' - C)h_i \leq (C' - C)$ is always true. Therefore, given $h_i > 1$, a violation of optimality occurs when

$$C' > \frac{\alpha_i - Ch_i}{1 - h_i} \tag{7.13}$$

and, for the case where $h_i < 0$, a violation occurs when

$$C' > \frac{Ch_i - \alpha_i}{h_i}. \tag{7.14}$$

The quantity δ_i for $i \in \mathcal{I}_s \cup \mathcal{I}_c$ varies with C according to

$$\boldsymbol{\delta}_o(C') - \boldsymbol{\delta}_o(C) = (C' - C)\left(Q_{os}\mathbf{h}_s + Q_{oc}\mathbf{1} - \mathbf{y}_o g_\beta\right) \tag{7.15}$$

$$\boldsymbol{\delta}_c(C') - \boldsymbol{\delta}_c(C) = (C' - C)\left(-Q_{cs}\mathbf{h}_s - Q_{cc}\mathbf{1} + \mathbf{y}_c g_\beta\right) \tag{7.16}$$

Define $\mathbf{s}$ such that $\mathbf{s}_o = (Q_{os}\mathbf{h}_s + Q_{oc}\mathbf{1} - \mathbf{y}_o g_\beta)$ and $\mathbf{s}_c = (-Q_{cs}\mathbf{h}_s - Q_{cc}\mathbf{1} + y_c g_\beta)$. A violation

150

of optimality occurs when, for $s_i < 0$,

$$-\tau > \delta_i + (C' - C) s_i \tag{7.17}$$

$$C' s_i < -\tau - \delta_i + C s_i \tag{7.18}$$

$$C' > \frac{-\tau - \delta_i + C s_i}{s_i} \tag{7.19}$$

Therefore, the following is computed

$$r_i = \begin{cases} \frac{\alpha_i - C h_i - \tau}{1 - h_i}, h_i > 1, i \in \mathcal{I}_s \\[2ex] \frac{-\alpha_i - \tau + C h_i}{h_i}, h_i < 0, i \in \mathcal{I}_s \\[2ex] \frac{-\tau - \delta_i + C s_i}{s_i}, s_i < 0, i \in \mathcal{I}_c \cup \mathcal{I}_o \\[2ex] r_i = \infty, \text{otherwise} \end{cases} \tag{7.20}$$

The new regularization parameter is found as follows where the tolerance ϵ ensures forward progress.

$$C' = \max \left(\min_i r_i, C + \epsilon \right) \tag{7.21}$$

Computing the Kernel Path

The approximate kernel path can also be solved within the context of the revised simplex method. Again, given a set of basic variables, $\mathbf{x}_B$, if the Hessian depends upon a parameter λ, the solution is

$$\begin{pmatrix} Q_B(\lambda) & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}_B(\lambda) \\ \boldsymbol{\pi}(\lambda) \end{pmatrix} = \begin{pmatrix} \mathbf{p}_B \\ \mathbf{b} \end{pmatrix} \tag{7.22}$$

151

In general, the matrix $Q_B(\lambda)$ is a non-linear function of λ. A first-order Taylor's series approximation can be used to find $\mathbf{x}_B(\lambda), \boldsymbol{\pi}(\lambda)$ as a linear function of λ as follows

$$\begin{pmatrix} \mathbf{x}_B(\lambda) \\ \boldsymbol{\pi}(\lambda) \end{pmatrix} \approx \begin{pmatrix} \mathbf{x}_B(\lambda_0) \\ \boldsymbol{\pi}(\lambda_0) \end{pmatrix} + (\lambda - \lambda_0) \frac{\partial}{\partial \lambda} \left[\begin{pmatrix} Q_B(\lambda_0) & B^T \\ B & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{p}_B \\ \mathbf{b} \end{pmatrix} \right] \tag{7.23}$$

Using the fact that $\frac{\partial A^{-1}}{\partial t} = -A^{-1} \frac{\partial A}{\partial t} A^{-1}$, when A depends on the parameter t, the expression simplifies to

$$\begin{pmatrix} \mathbf{x}_B(\lambda) \\ \boldsymbol{\pi}(\lambda) \end{pmatrix} \approx \begin{pmatrix} \mathbf{x}_B(\lambda_0) \\ \boldsymbol{\pi}(\lambda_0) \end{pmatrix} - (\lambda - \lambda_0) \begin{pmatrix} Q_B(\lambda_0) & B^T \\ B & 0 \end{pmatrix}^{-1} \begin{pmatrix} \frac{\partial}{\partial \lambda} \mathbf{x}_B \\ \mathbf{0} \end{pmatrix} \tag{7.24}$$

and, upon defining $h \triangleq \mathbf{x}_B(\lambda) - \mathbf{x}_B(\lambda_0)$, $g \triangleq \boldsymbol{\pi}(\lambda) - \boldsymbol{\pi}(\lambda_0)$ the following system of equations is solved to find the direction taken as λ varies

$$\begin{pmatrix} Q_B & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ \mathbf{g} \end{pmatrix} = (\lambda - \lambda_0) \begin{pmatrix} -\frac{\partial Q_B}{\partial \lambda} \mathbf{x}_B(\lambda_0) \\ \mathbf{0} \end{pmatrix} \tag{7.25}$$

It is straight forward to show that with the SVM formulation (or within the SVM-RSQP implementation) this becomes

$$\begin{pmatrix} -Q_{ss} & \mathbf{y}_s \\ \mathbf{y}_s^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h}_s \\ g_\beta \end{pmatrix} = \begin{pmatrix} \frac{\partial Q_{ss}}{\partial \lambda} \boldsymbol{\alpha}_s + \frac{\partial Q_{sc}}{\partial \lambda} \mathbf{1} C \\ 0 \end{pmatrix} \tag{7.26}$$

The pricing variables δ_i vary non-linearly, as well, according to λ. Therefore, the Taylor series approximation is used, again as follows

$$\delta_i(\lambda) \approx \delta_i(\lambda_0) + \frac{\partial \delta_i(\lambda_0)}{\partial \lambda}(\lambda - \lambda_0) \tag{7.27}$$

152

yielding, for $\boldsymbol{\delta}_o$, for example,

$$\frac{\partial \boldsymbol{\delta}_o}{\partial \lambda} = \frac{\partial Q_{os} \boldsymbol{\alpha}_s}{\partial \lambda} + \frac{\partial Q_{oc} \boldsymbol{\alpha}_c}{\partial \lambda} - \frac{\partial \beta \mathbf{y}_o}{\partial \lambda} \tag{7.28}$$

which upon expanding and removing constant terms yields

$$\frac{\partial \boldsymbol{\delta}_o}{\partial \lambda} = \frac{\partial Q_{os}}{\partial \lambda} \boldsymbol{\alpha}_s + Q_{os} \frac{\partial \boldsymbol{\alpha}_s}{\partial \lambda} + \frac{\partial Q_{oc}}{\partial \lambda} \boldsymbol{\alpha}_c - \mathbf{y}_o \frac{\partial \beta}{\partial \lambda} \tag{7.29}$$

From earlier

$$\begin{pmatrix} \frac{\partial \boldsymbol{\alpha}_s}{\partial \lambda} \\ \frac{\partial \beta}{\partial \lambda} \end{pmatrix} = \begin{pmatrix} -Q_{ss} & \mathbf{y}_s \\ \mathbf{y}_s^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} \frac{\partial Q_{ss}}{\partial \lambda} \boldsymbol{\alpha}_s + \frac{\partial Q_{sc}}{\partial \lambda} \mathbf{1} C \\ 0 \end{pmatrix} \tag{7.30}$$

or $\frac{\partial \boldsymbol{\alpha}_s}{\partial \lambda} = \mathbf{h}_s$ and $\frac{\partial \beta}{\partial \lambda} = g_\beta$. Therefore, the final result becomes

$$\frac{\partial \boldsymbol{\delta}_o}{\partial \lambda} = \frac{\partial Q_{os}}{\partial \lambda} \boldsymbol{\alpha}_s + Q_{os} \mathbf{h}_s + \frac{\partial Q_{oc}}{\partial \lambda} \boldsymbol{\alpha}_c - \mathbf{y}_o g_\beta \tag{7.31}$$

and for points in $\mathcal{I}_c$

$$\frac{\partial \boldsymbol{\delta}_c}{\partial \lambda} = -\frac{\partial Q_{cs}}{\partial \lambda} \boldsymbol{\alpha}_s - Q_{cs} \mathbf{h}_s - \frac{\partial Q_{cc}}{\partial \lambda} \boldsymbol{\alpha}_c + \mathbf{y}_c g_\beta \tag{7.32}$$

Define $s_i \triangleq \frac{\partial \delta_i}{\partial \lambda}$ in this case. Finally, similar to the case for the approximate path in terms of the regularization path for C, the next path in the approximate path can be found by computing

$$r_i = \begin{cases} \frac{C + \tau - \alpha_i + \lambda_0 h_i}{h_i}, & h_i > 0, i \in \mathcal{I}_s \\[2ex] \frac{-\alpha_i - \tau + \lambda_0 h_i}{h_i}, & h_i < 0, i \in \mathcal{I}_s \\[2ex] \frac{-\tau - \delta_i + \lambda_0 s_i}{s_i}, & s_i < 0, i \in \mathcal{I}_c \cup \mathcal{I}_o \\[2ex] r_i = \infty, & \text{otherwise} \end{cases} \tag{7.33}$$

The new kernel parameter is found as follows

$$\lambda' = \max\left(\min_i r_i, \lambda_0 + \epsilon\right) \tag{7.34}$$

where, once again, a tolerance ϵ is added to ensure forward progress. Note that while this method can be implemented efficiently similar to the case for the approximate regularization path, additional computation and storage is required for the derivative of the kernel matrix. This is unless the form of the kernel function allows some other simplification. The kernel caching scheme can be simply modified to store both the original kernel computations along with the derivative since the access pattern is similar. As a result, storage requirements and computations simply double. Furthermore, similar to the case where the quantities $Q_{oc}\boldsymbol{\alpha}_c$ and $Q_{cc}\boldsymbol{\alpha}_c$ are stored and updated as variables enter/leave the set $\mathcal{I}_c$, the quantities $(\partial Q_{oc}/\partial\lambda)\boldsymbol{\alpha}_c$ and $(\partial Q_{cc}/\partial\lambda)\boldsymbol{\alpha}_c$ can also be stored and updated.

<center>Limiting Behaviors of the Regularization and Kernel Path</center>

It is of interest to explore the limiting behaviors of the SVM solution for both the regularization path as well as kernel path. This provides some insight into why the regularization-path following algorithm may be more efficient than the traditional active set method in some cases. This also highlights potential improvements that might be made to further improve efficiency in the future. Of particular interest is the limiting behaviors in terms of the regularization parameter C as well as the kernel parameter γ for the RBF kernel.

First, note that the limiting behaviors described by Hastie *et al.* are general and just as applicable when exploring the solution behavior as C is varies. For thoroughness, the following corollaries show how the solution varies in terms of C for the conventional ℓ_1-loss formulation where $C \triangleq \frac{1}{\lambda}$

<center>154</center>

and λ is the regularization parameter used in the typical regularization path-following algorithm. Recall that the problem was reformulated and λ used in the path-following algorithms as it somewhat simplifies the expressions for computing the next event. The following depicts the expected solution for the conventional SVM formulation when C is small in the case of equal class sizes or $n_+ = n_-$

Corollary 2. *Given $n_+ = n_-$, then for sufficiently small C, all the $\alpha_i = C$.*

This simply follows the result reported in Hastie *et al.* [Lemma 1, [38]] using the fact that $\alpha_i = C\alpha_i'$ where α_i' represent the dual variables for the SVM formulation in the SVM path following algorithm reported by Hastie. The following is shown, similarly and without loss of generality, for the case where $n_+ > n_-$.

Corollary 3. *If $n_- > n_+$, then for sufficiently small C, $\alpha_i = C \ \forall i \in \mathcal{I}_-$ and $\alpha_i \in [0, C] \ \forall i \in \mathcal{I}_+$, $\sum_{i \in \mathcal{I}_+} \alpha_i = Cn_-$*

The following lemma is of interest since it highlights the fact that the active set method will be efficient in terms of both memory consumption and training time for small C.

Lemma 3. *As C approaches zero, it will be the case that $|\mathcal{I}_s| = 1$ for all iterations for the SVM-RSQP algorithm.*

This is easily shown by noting that when a new data point is added to $\mathcal{I}_s$ from $\mathcal{I}_c$ or $\mathcal{I}_o$, then θ, which represents the maximum distance that can be taken along the descent direction, approaches zero as C approach zero. That is, given a single variable, α_j is in $\mathcal{I}_s$, it is easy to show that upon choosing a variable α_i to enter $\mathcal{I}_s$, the quantity h_j computed for the variable in $\mathcal{I}_s$ according to the SVM-RSQP algorithm is ± 1 (see Algorithm 1). Further, given θ is computed as $\theta = \frac{\alpha_j - C[h_j < 0]}{h_j}$ the quantity $\alpha_j - C[h_j < 0] \to 0$ as $C \to 0$ and, therefore, $\theta \to 0$. This implies the variable α_j

will always be removed from $\mathcal{I}_s$ before adding the variable α_i. Given only single transitions occur, then the size of $\mathcal{I}_s$ can never be more than one.

Remarkably, this shows that the SVM-RSQP algorithm (and most active set implementations) can be very efficient at finding the solution for sufficiently small C since a large number of cheap iterations are performed. In fact, the revised simplex method behaves similarly to the SMO algorithm since only two variables are optimized during each iteration (one variable is entering $\mathcal{I}_s$ while another is leaving and the α_i components are adjusted).

Although not proven here, it seems reasonable that $|\mathcal{I}_s|$ will grow as C increases since data points are less likely to leave $\mathcal{I}_s$ when adding a new variable as C is increased. This is supported by the results shown in Figure 7.1 where the number of non-bound support vectors increase in an asymptotic fashion as C is increased. The maximum number of non-bound support vectors is limited by the effective rank of the kernel matrix, which will be discussed later.

This is of practical importance from the perspective of computing the regularization path with warm-starts. For example, consider the problem possessing a large fraction of bound support vectors even for high values of C. In that case, the majority of bound support vectors can be found at low values of C where the algorithm is more efficient. Therefore, the active set method can be sped up, in this case, by first finding the solution at a small C value and using that solution as a warm-start for some large C value.

The following proposition further indicates that the maximum size of $|\mathcal{I}_s|$ is limited by the rank of the kernel or Gram matrix Q.

Lemma 4. *Given $rank(Q) = m$, then $|\mathcal{I}_s| \leq m + 1$ for the SVM-RSQP algorithm.*
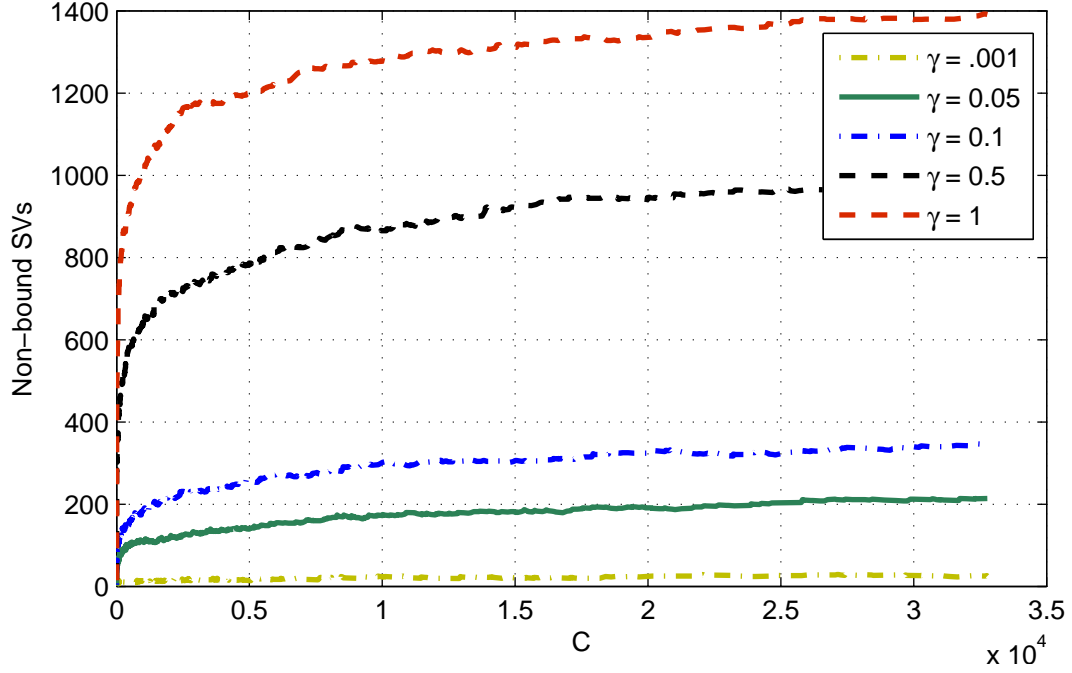
Figure 7.1: Depiction of how the number of non-bound support vectors varies as a function of C and kernel parameters. Results are shown for the abalone dataset and Gaussian kernel. As the kernel parameter γ increases, the effective rank and, therefore, maximum number of non-bound support vectors increases. The maximum number of non-bound support vectors are reached asymptotically as C increases.

This follows from the fact that the KKT matrix

$$
\begin{pmatrix}
-Q_{ss} & \mathbf{y}_s \\
\mathbf{y}_s^T & 0
\end{pmatrix}
\tag{7.35}
$$

is non-singular. In general, given a KKT matrix

$$
\begin{pmatrix}
Q & A^T \\
A & 0
\end{pmatrix}
\tag{7.36}
$$

where $Q \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{p \times n}$, the following statements are equivalent (a) the KKT matrix is non-

singular (b) $Z^T Q Z$ is positive definite where $AZ = 0$ and $Z \in \mathbb{R}^{n \times n-p}$, and (c) $\mathcal{N}(Q) \cap \mathcal{N}(A) = \{0\}$ or the null space of Q and A do not intersect (see Nocedal [66], Boyd [6]). In addition, the KKT matrix will not be singular if Q is positive definite [6]. Of use here is also the fact that $rank(AB) \leq \min(rank(A), rank(B))$ from which it is clear that $rank(Z^T Q Z) \leq \min(rank(Q), rank(Z))$. Since $rank(Z) = n - p$, then $rank(Z^T Q Z) \leq \min(rank(Q), n - p)$. Given $Z^T Q Z \in \mathbb{R}^{n-p \times n-p}$, the KKT system must be singular if $rank(Q) < n - p$ or $|\mathcal{I}_s| \geq m + 2$ given $rank(Q) = m$, and therefore, it will always be true that $|\mathcal{I}_s| \leq m + 1$. For $|\mathcal{I}_s| \leq m + 1$, the KKT matrix will be non-singular as long as $\mathcal{N}(Q_{ss}) \cap \mathcal{N}(\mathbf{y}_s^T) = \{0\}$.

For the case of a linear kernel, the rank of the Gram matrix is determined by the data dimensionality or number of features. Therefore, $|\mathcal{I}_s| \leq d + 1$ where d is the data dimensionality. Unfortunately, when using a kernel, the dimensionality may be even infinite-dimensional. The maximum rank of the Gram matrix is n where n is the dataset size, therefore, $|\mathcal{I}_s| \leq n$ for any general kernel function. Practically, the effective rank may be somewhat less than n where effective rank is measured by the number of singular values above a certain threshold. The effective rank of the Gram matrix associated with a kernel may depend upon the kernel parameters; for example, the Gaussian kernel, defined as $k_{ij} = exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2)$ generates a rank-one matrix as γ approaches zero ($k_{ij} = 1 \; \forall i, j$) and generates a diagonal matrix in the limit as γ tends to infinity. In another example, the Gram matrix associated with the polynomial kernel approaches a rank-one matrix as the degree, d, approaches zero. The effects of this behavior are clearly seen in Figure 7.1 where the maximum value for $|\mathcal{I}_s|$ increases as γ increases. As γ approaches 0 for the Gaussian kernel; the number of non-bound support vectors approaches one and, again, the computational complexity per iteration approaches that of the SMO algorithm. As γ increases, exponentially worse training times can be expected as $|\mathcal{I}_s|$ begins to increase, approaching n as γ increases to infinity.

This again suggests a rational for the efficiency of the regularization path and kernel path algorithms. That is, in some cases, if the solution found at some small value of C or γ is close in some

sense to the solution at a larger value, then significant speed-ups can be realized. While not fully realized in this work, this suggests the possibility of dynamically adjusting C and γ (effectively rescaling the active set problem) as a speed-up mechanism.

Low Rank Approximations of the Kernel Matrix

As shown previously, the maximum size of the KKT matrix is governed by the effective rank of the kernel matrix. Therefore, an effective means for limiting both memory usage and computational complexity for large datasets is to limit the rank of the Gram matrix. Fine *et al.* [27] suggested employing an incomplete Cholesky factorization for deriving a low-rank kernel approximation. The goal is to find a set of factors such that $\tilde{L}\tilde{L}^T \approx K$. The factor $\tilde{L}$ is of size $n \times m$ and results in a rank- m approximation and a storage requirement of $O(mn)$. The computational complexity of the incomplete Cholesky is shown to be $O(m^2n)$ [27], [37]. The incomplete Cholesky factorization with pivoting is shown below (see [37]).

Note that this implementation returns an approximation $L^T L \approx K$ where L is $\mathbb{R}^{n \times n}$ with only m rows containing non-zero entries. The algorithm has been modified to take a reference to the kernel function $k(i, j)$, instead of the entire kernel matrix. This "kernelization" provides additional efficiency since the algorithm only needs to access $O(nm)$ plus $O(n)$ diagonal entries from the kernel matrix. It is a straight-forward process to modify the algorithm to store the factor, L, in a more compact form $L \in \mathbb{R}^{m \times n}$. The incomplete Cholesky factorization stops when the trace of the approximation error or $tr(\Delta K)$ is less than ϵ where $\Delta K = K - \tilde{K}$ and $\tilde{K} = LL^T$. It is straightforward to show $tr(\Delta K) = \sum_j d_j$ (see [27]).

Implementation consists of computing and storing the incomplete Cholesky factorization prior to running the active set method.

Algorithm 6 Incomplete Cholesky Factorization with Pivoting

Input: Kernel function $k(i,j)$, desired approximation error, ϵ
Output: Cholesky factorization $L \in \mathbb{R}^{n \times n}$, diagonal $d \in \mathbb{R}^n$
 1: Set $d_j = k(j,j) \forall j$
 2: Set $\pi = \{1, 2, ..., n\}$
 3: Set err = $\|d\|_1$
 4: $m \leftarrow 1$
 5: **while** $err < \epsilon$ **do**
 6:     $j = \arg\max_{j \in \{m,...,n\}} d_{\pi_j}$
 7:     Swap π_j and π_m
 8:     $L_{m,\pi_m} = \sqrt{d_{\pi_m}}$
 9:     **for** $i = m + 1 : n$ **do**
10:        $L_{m,\pi_i} = k(\pi_m, \pi_i)$
11:        **for** $j = 1 : m - 1$ **do**
12:            $L_{m,\pi_i} = L_{m,\pi_i} - L_{j,\pi_m} L_{j,\pi_i}$
13:        **end for**
14:        $L_{m,\pi_i} = L_{m,\pi_i} / L_{m,\pi_m}$
15:        $d_{\pi_i} = d_{\pi_i} - L_{m,\pi_i} L_{m,\pi_i}$
16:     **end for**
17:     $d_{\pi_m} = 0$
18:     $err = \sum_{j=m+1}^{n} d_{\pi_j}$
19:     $m \leftarrow m + 1$
20: **end while**

For the SVM-RSQP algorithm, kernel caching is also used; however, the cache entries are formed from the Cholesky factors instead of computing the kernel function. While kernel caching is not necessary, this provides some additional savings since $O(m)$ operations are still required to compute a kernel entry from the Cholesky factors.

Results and Discussion

Approximate Regularization Path

Table 7.1: Datasets Used for Regularization and Kernel Path Experiments

Dataset	N	d
adult-1a	1605	123
abalone	4177	10
australian	690	14
diabetes	768	8
german	1000	24
heart	270	13
ionosphere	351	34
ocr-0	7291	256
sonar	208	60
spam	4601	57
splice	1000	60
web-1a	2477	300

Testing was done to compare performance of the SVM-RSQP algorithm using the traditional grid search methodology versus a grid search with incremental training and the proposed approximate path method. The grid search with incremental training is referred to as the "fast grid search". Experiments were performed on a Windows NT Server with 8 CPUs and 32 GBytes of RAM. The SVM-RSQP algorithm, with the proposed modifications, was compiled as a MATLAB MEX function. A stopping tolerance of 10^{-6} was employed, and testing was done for both the linear kernel and RBF kernel. Table 7.1 describes the datasets used for this study. Figures 7.2 and 7.3 depict the approximate path performance for the linear kernel and Figures 7.4 and 7.5 depict the results for the RBF kernel. Tables 7.2 and 7.3 summarize the results for the complete path search.

161

A grid search is performed with $\log_2(C)$ varying from -15 to 15 in increments of 2. This grid search is consistent with that seen in the literature, e.g. for use with LIBSVM [25]. The grid search was performed both with and without incremental training and individual times as well as cumulative training times are plotted for the case with full re-training at each C value. The cumulative times are plotted for the approximate path with path tolerances of 10^{-3}, 10^{-1}, and 10.

As expected cumulative training times for the approximate path algorithm increases greatly as the path tolerance approaches 10^{-3}. For the linear kernel case, the cumulative approximate path time is generally less than the cumulative time for the traditional grid search in all cases. When C is less than approximately 2^{-5}, the approximate path algorithms, regardless of tolerance, tend to be faster than the conventional grid search. The SVM solution does not vary much in this region (for the given datasets and scaling) and the traditional grid search is forced to retrain at all C values specified in the grid search while the approximate methods proceed directly to the C value where changes begin to occur. In essence, the approximate method finds the initial C value where the solution begins to change.

In all cases of the linear kernel and for $C < 2^5$, the approximate path method with a tolerance of 10^{-3} is slower than the traditional grid search by a factor of nearly 4 to 5 in some cases. However, the approximate path, in this case, typically computes several thousand solutions compared to less than 10 solutions for the traditional grid search. On the other hand, beyond $C = 2^{10}$, the approximate path is, in some cases faster than the traditional grid search by several factors. For high values of C, the active set, again, tends to remain constant and the approximate method avoids computing the solutions at these higher values of C while the grid search method is forced to re-train. This is especially beneficial since training times for the conventional active set method typically increase, sometimes exponentially, at higher values of C due to the affects of scaling.

162

Figure 7.2: Cumulative training times for the regularization path search and linear kernel. Individual grid search training times are also shown.

(a) ionosphere

(b) ocr-0

(c) sonar

(d) spam

(e) splice

(f) web-1a

Figure 7.3: Cumulative training times for the regularization path search and linear kernel. Individual grid search training times are also shown.
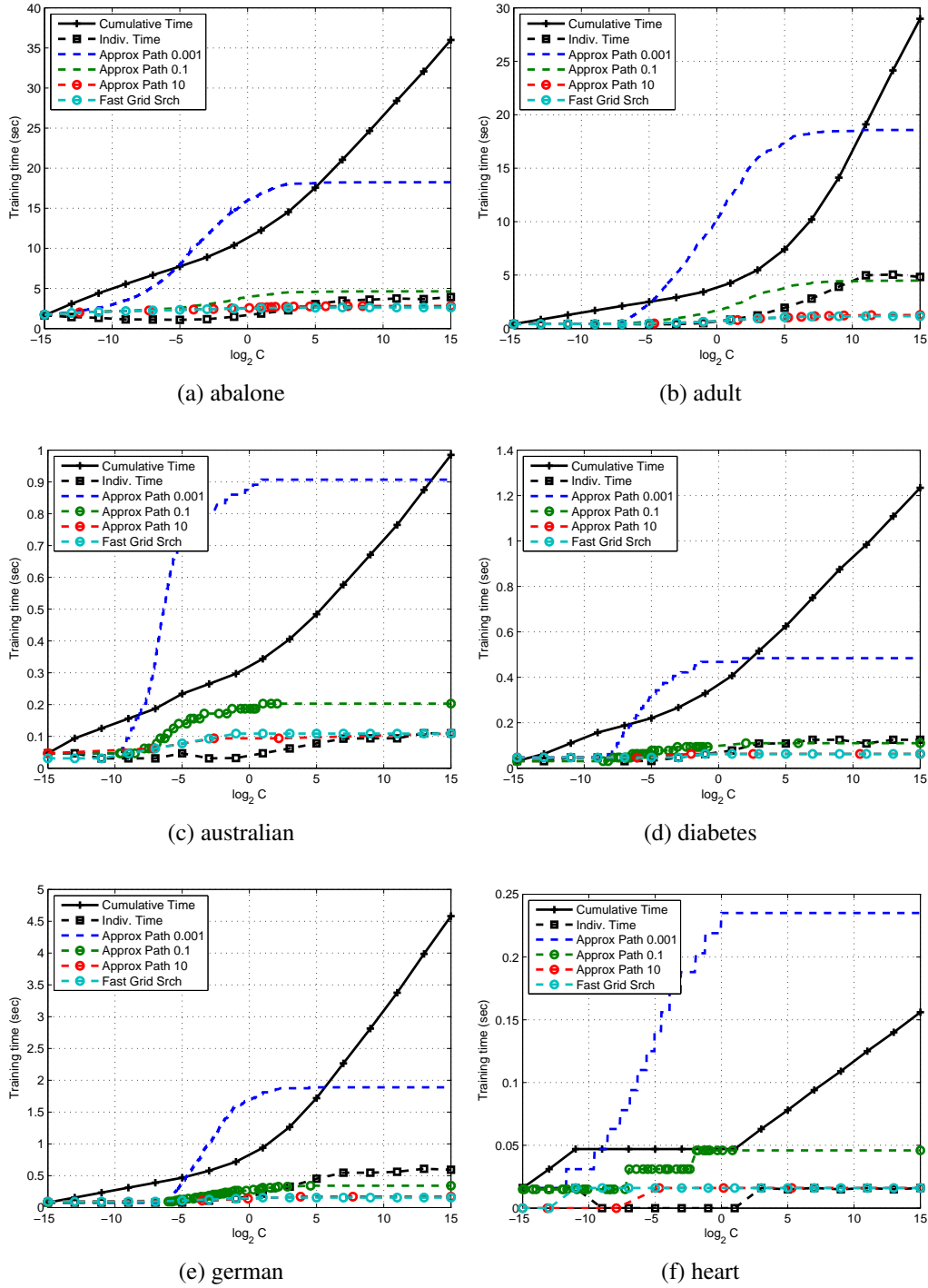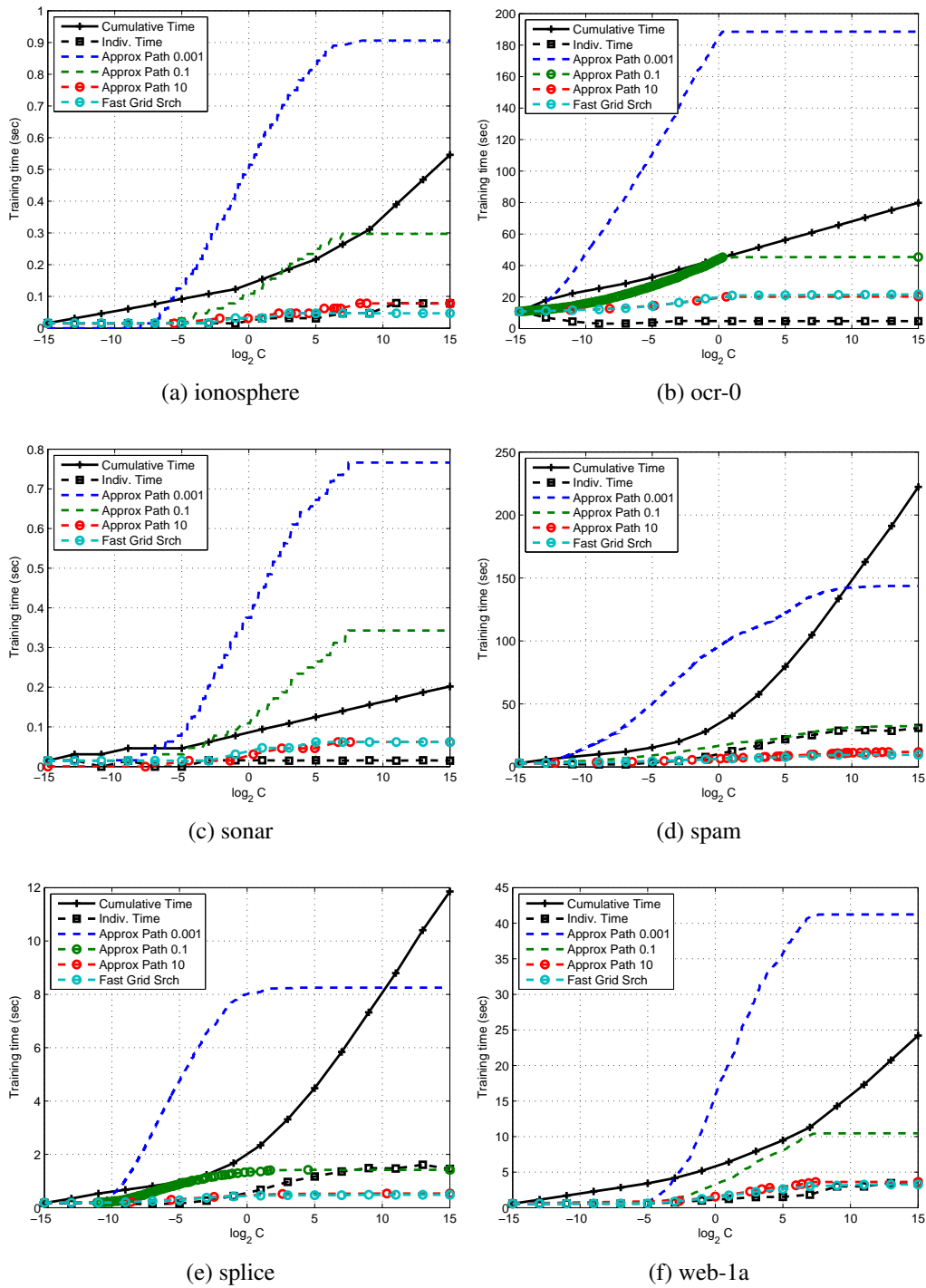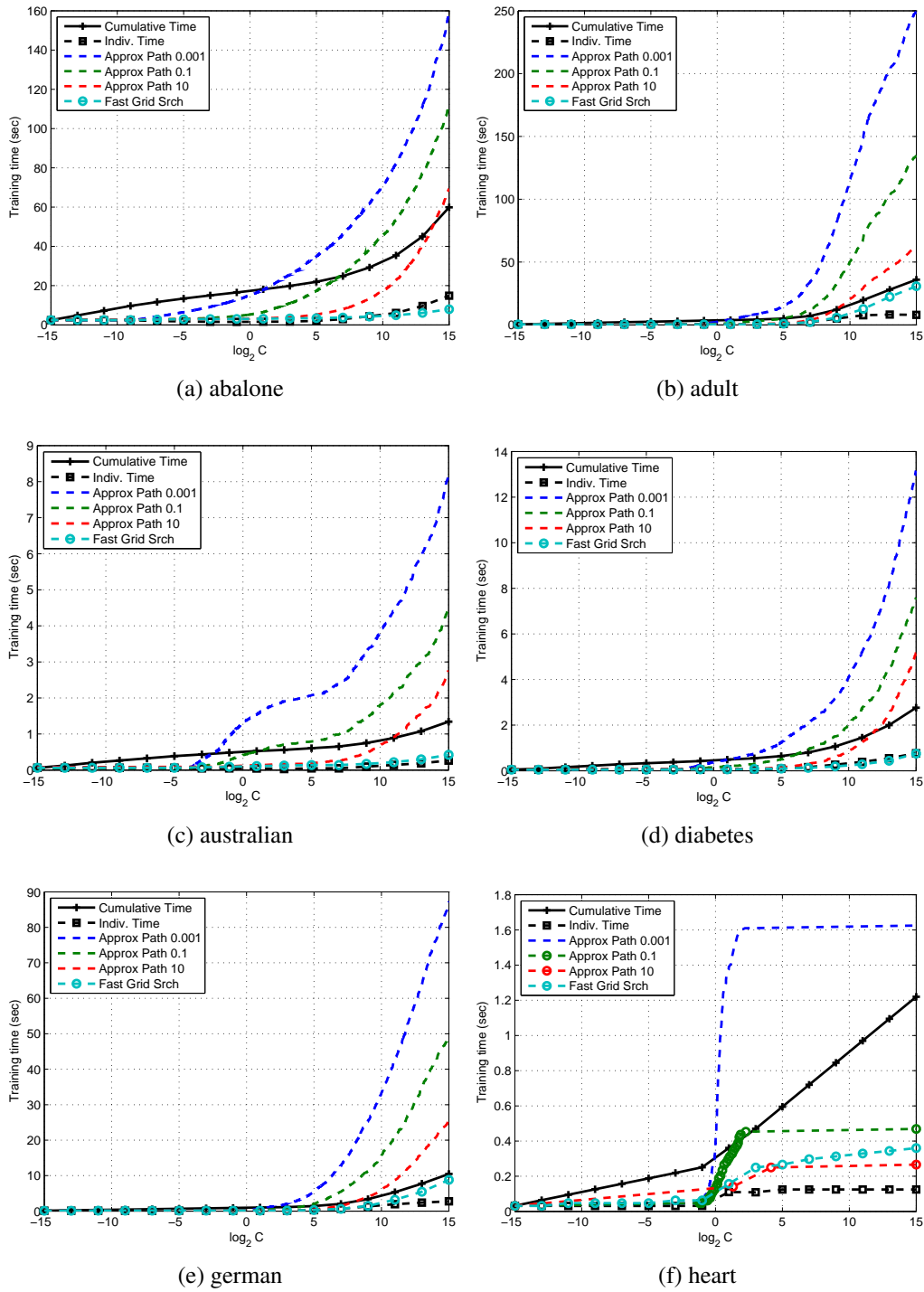
Figure 7.4: Cumulative training times for the regularization path search and RBF kernel. Individual grid search training times are also shown.
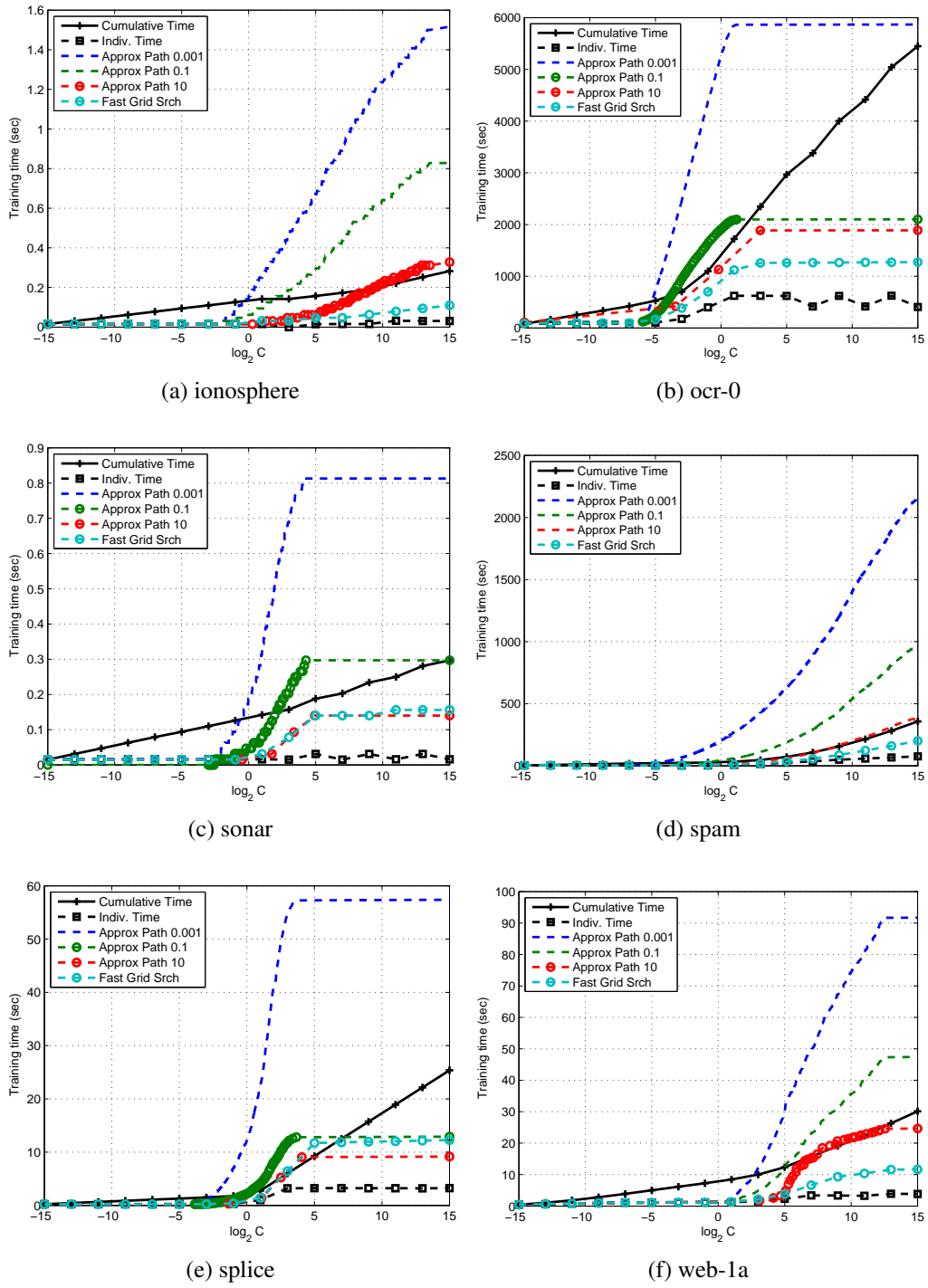
Figure 7.5: Cumulative training times for the regularization path search and RBF kernel. Individual grid search training times are also shown.

Table 7.2: Approximate Path Timing for the Linear Kernel

| Dataset | Grid Srch.[a] Time (sec) | Fast Grid[a] Time (sec) | Approx Path | | | | | |
| | | | tol = 0.001 | | tol = 0.1 | | tol = 10 | |
			(sec)	Sol.	(sec)	Sol.	(sec)	Sol.
adult-1a	28.97	1.16	18.58	911	4.48	133	1.26	12
abalone	36.00	2.62	18.25	2184	4.64	210	2.81	19
spam	222.30	9.64	143.67	2938	32.37	391	11.87	43
ocr-0	79.83	21.68	188.56	1065	45.39	95	20.28	7
web-1a	24.20	3.28	41.23	598	10.46	102	3.62	11
australian	0.99	0.11	0.91	442	0.2	31	0.11	5
diabetes	1.23	0.06	0.48	327	0.11	33	0.06	6
german	4.58	0.16	1.89	361	0.34	39	0.17	6
heart	0.16	0.02	0.24	348	0.05	42	0.02	7
ionosphere	0.55	0.05	0.91	499	0.30	116	0.08	15
sonar	0.2	0.06	0.77	407	0.34	118	0.06	11
splice	11.86	0.48	8.25	786	1.42	61	0.53	7

[a] Grid search performed across $\log_2(C) = (-15 : 2 : 15)$ for a total of 16 solutions.

As a result, it may be best to perform the fast grid search for high values of C even if only the solution at the last value of C is desired.

For path tolerances greater than 0.1, the approximate path-following method begins to be consistently faster than the traditional approach. Only in the case of the *sonar* dataset and linear kernel is the approximate path slower than the traditional grid search for a path tolerance equal to 0.1. At a path tolerance of 10.0 the approximate path is consistently faster than the traditional approach on all datasets and appears to produce a number of solutions on the order of the traditional grid search (16 solutions).

Table 7.3: Approximate Path Timing for the RBF Kernel

Dataset	Grid Srch.[a] Time (sec)	Fast Grid[a] Time (sec)	Approx Path					
			tol = 0.001		tol = 0.1		tol = 10	
			(sec)	Sol.	(sec)	Sol.	(sec)	Sol.
adult-1a	35.98	30.81	249.17	1783	134.7	664	61.84	140
abalone	59.91	7.97	158.266	5334	111.14	2126	69.27	777
spam	357.39	200.86	2159.77	4941	965.64	1190	384.78	210
ocr-0	5447.04	1273.20	5866.26	1215	2101.20	59	1888.52	5
web-1a	30.14	11.67	91.72	596	47.40	237	24.64	34
australian	1.34	0.42	8.19	1218	4.48	444	2.73	159
diabetes	2.77	0.75	13.16	1039	7.58	476	5.14	209
german	10.48	8.81	87.27	1416	49.59	583	25.72	165
heart	1.22	0.36	1.63	135	0.47	28	0.27	4
ionosphere	0.28	0.11	1.52	629	0.83	267	0.33	58
sonar	0.30	0.16	0.81	271	0.30	55	0.14	6
splice	25.36	12.27	57.38	934	12.91	60	9.17	6

[a] Grid search performed across $\log_2(C) = (-15 : 2 : 15)$ for a total of 16 solutions.

Surprisingly, the fast grid-search as well as the approximate path (tolerance of 10.0) are capable of computing the solution for several values of C in a time faster than the time it takes to solve a single value of C with full re-training. As an example, the time required for computing the solution for the *spam* dataset and linear kernel, from scratch, at $C = 2^{15}$ was 32 seconds while both the fast grid search and approximate path computed all the solutions up to $C = 2^{15}$ in less than 12 seconds. This type of behavior can occur when there are a large number of bound support vectors in the final solution at large values of C. The majority of these bound support vectors may be found at low values of C where iterations are cheaper. With incremental training, these bound support vectors do not have to be discovered again.

Table 7.4: Regularization Path Test Accuracy for the Linear Kernel

Dataset	Approx Path				Grid Srch
	tol = 0.001	tol = 0.1	tol = 1	tol = 10	
adult-1a	87.29	87.29	86.92	87.1	86.92
abalone	78.52	78.52	78.45	78.45	78.45
web-1a	98.18	98.18	97.94	97.7	97.94
australian	88.7	88.7	87.39	87.39	88.7
diabetes	80.86	80.86	80.47	80.08	80.47
diabetes	80.86	80.86	80.47	80.08	80.47
german	74.47	74.47	74.17	73.87	74.17
heart	83.33	83.33	83.33	82.22	82.22
ionosphere	92.31	92.31	92.31	92.31	92.31
sonar	81.16	79.71	79.71	79.71	79.71
splice	83.78	83.48	83.48	82.58	83.48
ocr-0	99.26	99.26	99.22	99.26	99.18
spam	93.8	93.8	93.8	93.8	93.74

For example, this scenario also occurs for the *adult-1a* and *abalone* datasets where there are 485 out of 1605 bound support vectors and 2234 out of 4177 bound support vectors for the largest value of C, respectively. This shows that in some cases (especially when there are a large fraction of bound support vectors) it may be advantageous to perform a grid search with incremental updates even when the solution at a single value of C is desired. In those cases, the fast grid search can be seen as a speed-up technique for the conventional active set method.

Testing was also performed with the RBF kernel ($\gamma = 0.01$). Here, the approximate regularization path method underperformed the traditional grid search in terms of total training time in nearly all cases while the fast grid search method out-performed the traditional grid search in all cases.

Table 7.5: Regularization Path Test Accuracy for the RBF Kernel

Dataset	Approx Path				Grid Srch
	tol = 0.001	tol = 0.1	tol = 1	tol = 10	
adult-1a	84.3	84.3	84.3	83.93	84.3
abalone	80.96	80.96	80.96	80.96	80.6
web-1a	97.7	97.7	97.58	97.58	97.58
australian	89.13	88.7	88.26	87.83	88.26
diabetes	76.56	76.56	76.56	76.56	76.17
german	78.38	78.38	78.38	78.38	78.08
heart	61.11	61.11	60.0	60.0	61.11
ionosphere	97.44	97.44	97.44	97.44	96.58
sonar	89.86	89.86	89.86	88.41	89.86
splice	86.79	86.79	86.49	86.49	86.79
ocr-0	98.81	98.81	98.81	98.81	98.81
spam	93.93	93.93	93.93	93.93	93.54

In those cases where the approximate path is slower than the traditional grid search (at a tolerance of 10.0), the number of computed solutions or events is typically several orders of magnitude higher. For example, while the approximate path took nearly twice as long as the traditional grid search for the *adult-1a* dataset, nearly 10 times as many solutions were computed. In a few cases, the fast grid search exhibited cumulative training times that were on-par if not faster than the time required to compute a solution from scratch. For example, the fast grid search computed all solutions 50 percent faster than the time taken to compute a single solution at $C = 2^{15}$ for the *abalone* dataset.

Overall, these results suggest that a substantial speed-up can be achieved by performing incremental updates to the solution while performing a grid search. In many cases, the cumulative training time associated with a full grid search is improved by more than a factor of 10. Performance was

170

best for the linear the kernel. Performance was not as significant for the RBF kernel; however, the fast grid search was nearly a factor of 5 faster than the traditional grid search for the *ocr-0* dataset, for example. A path tolerance of 10.0 appears to most closely approximate the traditional grid search in terms of the number of computed solutions as well as speed, especially for the linear kernel. For the non-linear kernel, the approximate path appears to be less predictable with the number of computed solutions for a path tolerance of 10.0 varying between 4 for the *heart* dataset and 777 for the *abalone* dataset. Additional research is needed, here, to understand the relationship between path tolerance, which is really a measure of algorithm tolerance, versus classifier accuracy.

An additional set of experiments were performed to compare test accuracy for the conventional grid search versus the approximate regularization path with several path tolerances. For each dataset, $1/3^{rd}$ of the data was reserved for testing and training was performed on the remaining $2/3^{rd}$s of the data. Tables 7.4 and 7.5 summarize these results. Surprisingly, the results suggest that a low path tolerance provides little benefit in terms of maximizing the performance of the classifier. The largest disparity in performance between the standard grid search and the approximate path search is found for the *sonar* dataset and linear kernel. Here, the approximate path search with the lowest tolerance yielded a 1.4 percent improvement in accuracy over that achieved with the standard grid search. In addition, even the highest tolerance case (tolerance of 10.0) yielded a test accuracy within 1.5 percent of that achieved with the lowest tolerance.

In this testing, a tolerance of 10^{-3} is found to be overly cautious while incurring significant training costs. A path tolerance of 1.0 or higher appears to provide reasonable accuracy while attaining a significant speed-up in terms of training time. The results suggest the fast grid search is the best option in terms of speed and precision. Remarkably, in fact, the logarithmic grid search, which is often cited within the literature, performs surprisingly well at finding the best performing classifier. While the grid search cannot provide guarantees on the degree to which a discovered solution is optimal, it seems the method can still be expected to perform well from a practical

standpoint. Additionally, classifier performance, in terms of test accuracy, is not linearly related to path tolerance. Test accuracy remained relatively stable even as path tolerance was increased by several orders of magnitude. Further research is needed to understand the bounds on accuracy for a given path tolerance. In the end, the approximate path still provides guarantees on optimality and is the recommended approach.

A question arises as to whether the traditional regularization path algorithm (SSVMP) is faster than the proposed approximate path search using SVM-RSQP. A set of experiments were performed with a MATLAB version of SVM-RSQP for comparison with SSVMP. The maximum C value for the approximate path search was set to 1000.0, which is equivalent to the $\lambda = 10^{-3}$ stopping condition for SSVMP. Reported times do not include times required for kernel caching. Tables 7.6 and 7.7 show results for the linear and RBF kernel, respectively.

For the linear kernel, the SVM-RSQP approximate path algorithm appears to be competitive with SSVMP. At a path tolerance of 10^{-6}, the number of events for the SVM-RSQP path-following algorithm appears to be "on par" with the SSVMP algorithm. This seems reasonable since an algorithm tolerance of 10^{-6} is used with the SSVMP algorithm. Significant differences in the number of events occur only for the *spam* and *abalone* datasets. As the path tolerance is increased, significant speed-ups can be achieved. For example, at a tolerance of 1.0, the approximate path is computed 12 times faster for the *web-1a* algorithm when compared to the SSVMP algorithm. It is important to note that the time for computing the approximate path does not necessarily continue to decrease as path tolerance is increased. This is due to the fact that as path tolerance is increased, the number of iterations required to update the solution between events also increases. Therefore, there is a non-linear relationship between training times and path tolerance. In fact, it is possible to experience increased training times with increased path tolerance, which is explained by the observation that the regularization path-following algorithm can be seen as a speed-up mechanism in some instances.

172

For the RBF kernel, small inefficiencies of the SVM-RSQP algorithm compared to SSVMP become more apparent. A path tolerance of 10^{-6}, once again, appears to provide similar results to the SSVMP algorithm in terms of the number of events being consistent for nearly all datasets (only the *heart* dataset experienced fewer events). However, training times for the SVM-RSQP algorithm are nearly double that of SSVMP in some cases. The poorer performance for the RBF kernel is due to the fact that the KKT matrix is larger as a result of higher effective rank of the kernel matrix. The source of the inefficiency is that between events, the SSVMP algorithm requires, typically, only one solution of the KKT matrix while the SVM-RSQP algorithm must solve the KKT matrix to find the next event, fix the complementary conditions, as well as adjust the active set. Therefore, as many as 2 to 3 additional solutions involving the KKT matrix may be required between each event, at a path tolerance of 10^{-6}, when compared to the SSVMP algorithm. Despite this, training times are reduced as the path tolerance is increased, and, as a result, training times for the approximate path can be reduced to nearly half of that associated with the SSVMP algorithm. A possible topic of future research is to explore methods for further improving the efficiency of the SVM-RSQP algorithm when computing the regularization path.

In summary, the SVM-RSQP algorithm is competitive with the SSVMP algorithm when computing the "exact" path, especially for the linear kernel. Additionally, the results demonstrate the ability of the SVM-RSQP algorithm to realize speed-ups associated with computing the approximate path.

Approximate Kernel Path

A MATLAB implementation of the SVM-RSQP algorithm was modified to incorporate the approximate kernel path algorithm. The MATLAB implementation fully implements kernel caching as well as rank-one updates to a Cholesky factorization as described in Chapter 5.

Table 7.6: Approximate Path Performance versus SSVMP with the Linear Kernel

Dataset	tol = 10^{-6}		tol = 10^{-3}		tol = .1		tol = 1		SSVMP	
	(sec)	Evts	(sec)	Evts	(sec)	Evts	(sec)	Evts	(sec)	Evts
adult-1a	5.86	1454	4.65	902	2.66	129	2.15	40	12.55	1454
abalone	17.46	4103	13.10	2074	8.10	211	7.38	63	29.23	7301
australian	3.05	1423	1.95	435	1.30	31	1.01	12	2.29	1719
diabetes	1.26	511	1.04	325	0.60	29	0.54	10	1.57	522
heart	1.11	583	0.83	340	0.45	37	0.41	13	0.55	650
german	1.66	552	1.37	360	0.84	38	0.78	13	3.91	555
ionosphere	1.43	672	1.23	497	0.71	115	0.54	35	0.85	679
sonar	1.28	514	1.16	404	0.71	117	0.53	34	0.47	521
splice	5.64	1596	3.93	761	2.34	60	1.78	14	3.96	1862
web-1a	8.02	904	7.16	599	3.93	97	3.07	34	36.44	1020
spam	33.01	5278	24.28	2689	14.54	370	12.25	105	69.47	9270

The kernel cache was modified to cache both columns of the kernel matrix as well as columns of the derivative of the kernel matrix. All routines were executed on a Windows 2003 Server workstation with 8 CPU cores and 16 GBytes of RAM. MATLAB was configured to enable multi-core support. The same datasets described in the previous section were used here. A grid search, fast grid search, as well as approximate grid searches were performed for γ ranging between 2^{-15} and 2^{15}. One third of the data was reserved for testing in all cases. The results focus on training speed as well as accuracy in all cases.

Timing results are listed in Table 7.8. For the approximate path searches, all cases where the path tolerance was greater than 10.0 resulted in faster training times than either the standard or fast grid approaches. However, in most cases, the number of computed solutions were also fewer (20 solutions are computed in the grid search).

Table 7.7: Approximate Path Performance versus SSVMP with the RBF Kernel ($\gamma = 0.1$)

Dataset	tol = 10^{-6}		tol = 10^{-3}		tol = .1		tol = 1		SSVMP	
	(sec)	Evts	(sec)	Evts	(sec)	Evts	(sec)	Evts	(sec)	Evts
adult-1a	110.09	1515	94.59	974	59.69	106	60.36	28	67.90	1520
abalone	95.54	8623	71.53	4215	52.14	1226	43.89	552	86.85	9671
australian	8.76	1597	7.47	965	4.92	214	4.11	85	5.37	1575
diabetes	19.14	1267	19.25	1029	13.10	272	11.04	111	12.37	1248
heart	0.79	36	0.71	17	0.65	3	0.64	2	1.54	122
german	72.05	1186	63.82	894	33.06	143	30.01	41	40.78	1181
ionosphere	1.71	539	1.49	415	0.88	82	0.70	27	1.27	536
sonar	1.17	168	0.93	110	0.56	13	0.54	4	0.68	170
splice	82.30	478	53.79	148	40.03	12	38.48	3	55.82	510
web-1a	67.84	617	51.74	328	40.93	55	48.72	17	94.57	624
spam	2957.51	6620	2253.70	2597	1541.56	264	1475.22	85	1554.37	6960

In the case of the *abalone* dataset, 31 solutions were computed in slightly less time than for the full grid search and 20 solutions were computed in 40 percent less time. For a path tolerance of 0.1, training time is slower than for the grid search in many cases. However, consider that for the *abalone* dataset, 928 different kernel values are solved versus 20 in the grid search while taking only 3 times as long.

The fast grid search method is also faster than the traditional grid search in all cases with the exception of the *abalone* dataset where the fast grid search was only slightly slower. In many cases, the fast grid search is 3 to 5 times faster than the traditional grid search. An interesting result is observed for the *abalone* dataset where the approximate search path with a path tolerance of 100.0 computes 20 solutions in far less time than even the fast grid search, which also computes 20 solutions. Similar behavior is observed for the *german* dataset and a path tolerance of 1.0.

Table 7.8: Approximate Kernel Path for $C = 1$

Dataset	Grid Srch.[a] Time (sec)	Fast Grid[a] Time (sec)	Approx Path							
			N	100 (sec)	N	10 (sec)	N	1 (sec)	N	.1 (sec)
adult-1a	512.75	301.65	4	99.48	7	147.98	21	212.60	93	539.0
abalone	3628.88	3890.44	20	2239.33	31	3442.80	214	4689.23	928	9083.38
web-1a	2287.47	1521.48	4	527.20	6	610.08	16	974.5	33	1348.92
australian	29.75	14.73	6	8.25	12	10.84	67	17.55	236	33.63
diabetes	44.16	16.40	5	7.25	9	10.77	30	13.18	122	26.89
german	103.43	48.06	5	20.95	9	28.34	20	40.29	72	79.03
heart	7.03	1.48	5	0.79	9	1.03	24	1.57	65	2.45
ionosphere	9.04	3.76	5	1.60	10	2.52	32	3.77	104	6.24
sonar	5.38	1.33	4	0.61	7	0.73	17	1.22	50	2.31
splice	83.47	68.39	5	14.06	8	49.04	31	83.71	129	161.34

[a] Grid search performed across $\log_2(\gamma) = (-15 : 2 : 15)$ for a total of 20 solutions.

This suggests a potential advantage of the approximate kernel path method compared to the fast grid search in terms of training time, perhaps due to the effect of problem scaling at each of the kernel parameters chosen by the approximate path versus the grid search. Unfortunately, the number of solutions computed for a given path tolerance is not easily predicted for a given dataset, a topic of future research.

In addition to training times, the classifier best-case test accuracy is reported for each of the path tolerances in Table 7.9. In summary, a path tolerance less than or equal to 10.0 was found to be acceptable in terms of providing an optimal solution. Surprisingly, again, the standard grid search results in a peak test accuracy within 1 percent of the lowest tolerance case.

Table 7.9: Kernel Path Test Accuracy for the RBF Kernel ($C = 1$)

Dataset	Approx Path				Grid Srch
	tol = 0.1	tol = 1	tol = 10	tol = 100	
adult-1a	82.24	82.24	82.06	80.37	81.68
abalone	78.88	78.88	78.74	78.59	78.88
web-1a	97.94	97.94	97.94	97.94	97.94
australian	87.39	86.96	86.96	86.96	86.96
diabetes	76.95	76.95	76.17	74.22	76.17
german	78.38	78.08	78.08	74.17	77.48
heart	70.00	70.00	70.00	70.00	70.00
ionosphere	96.58	96.58	96.58	94.87	95.73
sonar	84.06	82.61	82.61	84.06	84.06
splice	90.69	89.79	87.39	88.29	90.09

The approximate path with the highest tolerance (100.0) achieved a test accuracy within 4 percent of the best accuracy and a tolerance of 10.0 achieved an accuracy within 1 percent of the best case and appeared to perform most similarly to the grid search. The *splice* dataset represents an exception where a path tolerance of 10.0 resulted in a test accuracy 3.3 percent less than the best case.

In summary, the approximate kernel path can provide a significant speed-up in terms of training time, nearly a factor of 5 speed up in some cases, while providing test accuracy well within the 3 percent of the best case (for a tolerance of 10.0). With the lowest tolerance, the kernel path is no more than 3 times slower than the conventional grid search while providing up to 50 times as many solutions.

Table 7.10: Kernel Path with Incomplete Cholesky Kernel Approximation ($C = 1$)

Dataset	rank = 50 Time (sec)	Acc. %	100 Time (sec)	Acc. %	150 Time (sec)	Acc. %	200 Time (sec)	Acc. %	250 Time (sec)	Acc. %
adult-1a	25.86	84.30	39.73	84.30	55.87	84.30	79.37	84.49	98.62	84.49
abalone	100.32	79.81	131.03	79.74	165.44	80.46	210.76	80.46	305.74	80.39
web-1a	25.24	96.97	53.24	96.85	85.12	96.97	118.85	96.97	155.73	96.97
australian	12.43	85.65	16.69	86.52	22.27	86.09	26.55	87.39	31.70	86.52
diabetes	13.30	79.30	18.99	79.30	25.32	79.30	31.99	79.30	37.86	79.30
german	17.04	77.78	24.15	77.18	33.40	76.88	45.50	76.58	58.21	76.58
heart	5.33	63.33	7.21	63.33	9.09	63.33	8.72	63.33	8.84	63.33
ionosphere	5.65	89.74	7.98	93.16	10.55	94.02	11.73	94.02	11.79	94.02
sonar	3.91	79.71	5.07	84.06	6.42	85.51	5.60	85.51	5.57	85.51
splice	21.33	80.78	31.97	82.28	37.71	83.78	46.34	84.38	56.67	85.29
spam	110.90	89.04	155.05	91.39	210.47	92.11	272.95	92.76	346.25	93.02
ocr-0	123.27	97.12	209.24	98.19	309.01	98.35	421.93	98.35	566.78	98.48

Incomplete Cholesky Kernel Approximation

A MATLAB implementation of SVM-RSQP was again modified to incorporate the incomplete Cholesky factorization technique for low-rank kernel approximation. A grid search was performed on the RBF kernel parameter with $\log_2 (\gamma)$ varying between -15 and 15 in steps of 2. Cumulative training time as well as maximum grid search test accuracy are reported. The incomplete Cholesky method was configured to terminate when the trace of the error matrix reaches 10^{-1} or once the maximum rank is achieved. The maximum rank was tested at values of 50, 100, 150, 200, and 250. Results are listed in Table 7.10.

As expected, significant speed-ups can be achieved with the kernel approximation. For example,

when compared to the traditional grid search, a worst-case factor of 10 speed-up is observed for the *abalone* dataset. Note that the reported results are for a conventional grid search. The potential benefit gained from performing incremental updates to the solution for the kernel path are minimized by the need to recompute the incomplete Cholesky factorization for each kernel parameter value. While the kernel path is reported here, more significant speed-ups are expected for the standard regularization path since the incomplete Cholesky factorization need only be computed once.

In terms of test accuracy, performance is acceptable in most cases even when the maximum rank of the approximation is 50. In many cases, the improvement in accuracy gained from increasing the rank of the kernel approximation from 50 to 250 is less than 1 percent. The *spam* and *ocr-0* datasets appear to benefit the most from extending the rank to 250. These results are consistent with those reported in Fine *et al.* [27].

It is instructive to consider the performance of the *ocr-0* dataset in more detail. Figure 7.6 shows that while peak accuracy does not vary significantly as the rank of the approximation varies (less than 5 percent variation), test accuracy can vary significantly at individual points along the path. In this case, test accuracy is degraded by more than 50 percent in one case where the lowest rank approximation is employed and by more than 10 percent at the highest rank approximation. Generally, the low-rank kernel approximation is expected to perform well for small γ values where the effective rank of the kernel matrix is low and will become less accurate as γ is increased. At some point as γ is further increased, the off-diagonal entries of the kernel matrix approach zero. In this case, the low-rank approximation will again accurately model the off-diagonal elements although errors along the diagonal will be large. In summary, approximation errors will tend to be the greatest at some mid-point value of γ where the kernel matrix possesses high effective rank along with a large number of non-zero off-diagonal entries.
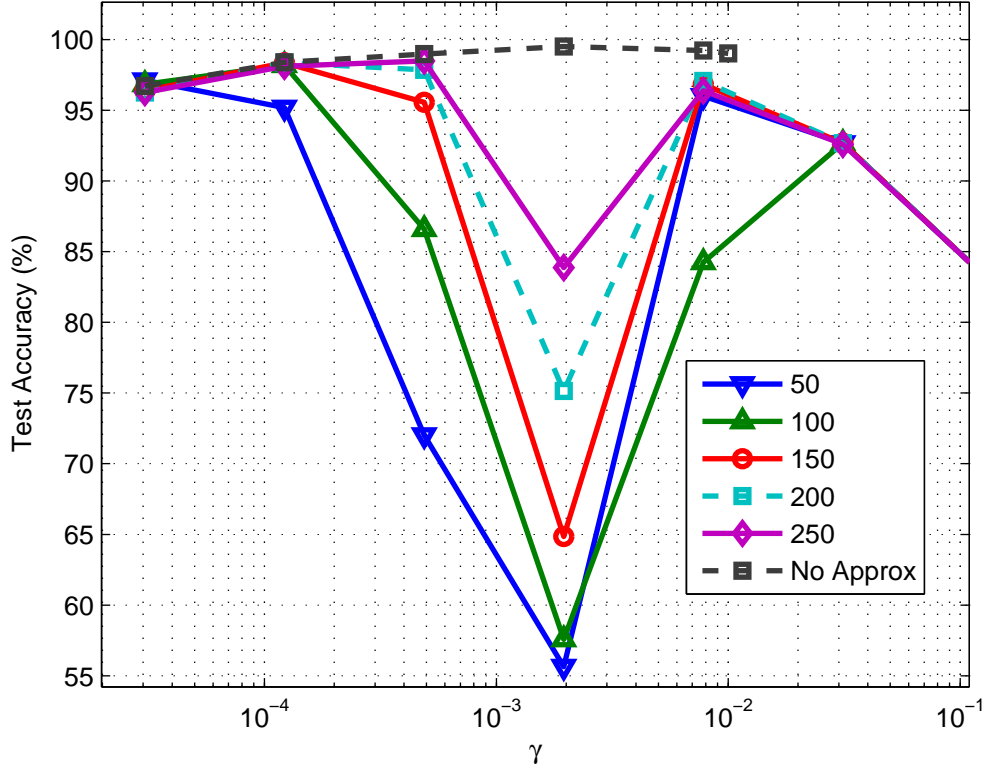
Figure 7.6: Example of the test accuracy for the *ocr-0* dataset during a kernel grid search ($C = 1$) using the incomplete Cholesky factorization for low-rank kernel approximation. Each curve depicts a different rank of the kernel approximation. The dashed black curve shows the test accuracy without approximation.

Restricting the rank of the approximation, rather than strictly relying on the trace of the error matrix to determine the rank, is desirable in cases where memory and/or computational constraints are considered. More importantly, as the RBF kernel approaches a diagonal matrix, a low-rank approximation with low error cannot be found with the Cholesky factorization, and, therefore, a nearly full rank matrix will result since the Cholesky factorization of a diagonal matrix is itself a diagonal matrix. Therefore, an important topic of future research is to study the kernel approximation from the context of path-searching that takes into consideration the nature of the kernel matrix with varying kernel parameters.

CHAPTER 8: CONCLUSIONS

This research largely addresses the issues and practical application of the active set method for SVM training with semi-definite kernels. This research is shown to be important since the linear kernel, a semi-definite kernel, is one of the most widely-used kernels, the kernel matrix can become semi-definite even when the kernel function is positive definite when, for example, duplicate data points exist, and there is a need to employ low-rank kernel approximations for large datasets. The conventional active set method was shown to encounter difficulty with the semi-definite kernel when a chosen active set results in a singular KKT system. This research shows that a practical active set method can be implemented that automatically avoids singular KKT systems. This is done without needing to explicitly detect the rank of the KKT matrix and adjusting the solution method or explicitly detecting whether a particular pivot will result in a singular KKT matrix.

The revised simplex method for semi-definite quadratic programming was efficiently implemented for SVM training (SVM-RSQP) and shown to out-perform existing active set implementations for semi-definite kernels as well as remain competitive with decomposition methods such as SVM-Light and SMO, in terms of training time. The active set method has recently gained additional importance with the regularization path-following algorithms. Therefore, inspired by the revised simplex method for semi-definite programming, a new algorithm was developed, Simple SVM-Path (SSVMP), for solving the regularization path with semi-definite kernels. This is a fully self-contained implementation in that it does not need an external solver. For example, the SVMPath algorithm [38] requires an external solver for initialization with unequal class sizes and the ISVMP algorithm [67] requires an external solver for initialization, to compute the next event, and as a backup recovery mechanism when a breakdown in the path computation is found. The SSVMP algorithm also implements a new initialization method for unequal class sizes, which relies on the ability to use semi-definite kernels. The SSVMP algorithm was found to out-perform compet-

ing algorithms in terms of both training time and numerical stability. Extensive analysis proves correctness and convergence of the algorithm as well as reveals previously unknown issues with existing methods that are solved with SSVMP. As a result, the SSVMP algorithm represents a practical implementation of a path-following algorithm that can be reliably applied to a broad class of problems including those with sparse, semi-definite kernels.

Finally, the SVM-RSQP algorithm was modified to compute the approximate regularization path and approximate kernel path and suggested as a practical alternative to existing path-following methods. Further, due to the fact that semi-definite kernels can be reliably handled, low-rank kernel approximations can be successfully applied to speed-up the active set method for large datasets. While the SVM-RSQP algorithm was found to be slightly less efficient than the SSVMP algorithm when computing the "exact" regularization path for the RBF kernel, significant speed-ups were realized when computing the approximate path with higher tolerances. Surprisingly, the entire approximate regularization path is computed in a time faster than that required to perform full training at a single regularization value in some cases. As a result, the approximate path method, in addition to computing solutions at several values of the regularization path, may be seen to represent a speed-up method for conventional active set methods even when the solution at a single value of the regularization parameter is desired. This is especially true for large values of the regularization parameter, C, where training times tend to increase exponentially with increasing C due to poor scaling of the problem.

Surprisingly, the logarithmic grid search, often reported in the literature (e.g. [25], [80]), was found to be successful at finding a solution with reasonable test accuracy in most cases. This suggests that the logarithmic grid search may be an acceptable alternative to the regularization path algorithms in many cases. The fast grid search, where warm starts are employed to incrementally update solutions as the regularization or kernel parameter is adjusted, can be used to perform the logarithmic grid search, providing solutions with orders of magnitude faster training times.

However, the approximate path can be configured to provide an equivalent number of solutions as the logarithmic grid search while providing both a speed-up and guarantees on the optimality of the computed solution. In most cases, it is found that a high path tolerance is acceptable in terms of finding the optimal test accuracy. Lower values of the path tolerance may be used when higher test accuracy is required (< 1 percent) at the potential cost of longer training times.

Finally, the low-rank kernel approximation, via the incomplete Cholesky factorization, was studied as a mechanism for speeding-up the active set method for large datasets. The low-rank kernel approximation appears to be the best method for both reducing training times as well as reducing memory consumption. In fact, given a low-rank kernel approximation with rank m, the memory requirement becomes $O\left(mn + m^2\right)$. Orders of magnitude (> 10) speed-ups were reported with a rank of $m = 250$ for the low-rank approximation. The incomplete Cholesky factorization was found to work best, in terms of test accuracy, as expected when the kernel matrix has a low effective rank. However, cases were found where the incomplete Cholesky factorization begins to fail when the effective rank of the kernel matrix is high with a large proportion of non-zero off-diagonal entries. An important topic of future research remains in finding improved methods for approximating kernel matrices with low effective rank as well as those best approximated as full-rank diagonal matrices. Note that while suggested by others [95], this represented the first-known reported instance of using a low-rank kernel approximation with the active set method.

The approximate kernel path was also successfully implemented and shown to provide benefit in terms of training time relative to the traditional grid search where re-training occurs at each value. In many cases, training times were 2 or more times faster when the approximate kernel path was configured to compute a comparable number of solutions to the traditional grid search.

In summary, the active set method can be practically and efficiently applied to SVM training with semi-definite kernels. As a result, the active set remains a viable SVM training algorithm especially

for searching the solution space associated with the regularization and kernel parameters and when low-rank kernel approximations are employed for large-scale datasets.

LIST OF REFERENCES

[1] S. Abe and R. Yabuwaki. Convergence improvement of active set training for support vector regressors. In *Artificial Neural Networks, ICANN 2010*, volume 6353 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin / Heidelberg, 2010.

[2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[3] K. P. Bennett and E. J. Bredensteiner. Duality and geometry in svm classifiers. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 57–64, San Francisco, CA, USA, 2000.

[4] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, pages 103–107, 1977.

[5] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, New York, NY, USA, 1992. ACM.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[7] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[8] H. Byun and S.W. Lee. Applications of support vector machines for pattern recognition: A survey. *Lecture Notes in Computer Science*, pages 213–236, 2002.

[9] E. Byvatov and G. Schneider. Support vector machine applications in bioinformatics. *Applied Bioinformatics*, 2(2):67–77, 2003.

[10] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *NIPS*, pages 409–415, 2000.

[11] S. Chakrabarti, S. Roy, and M. V. Soundalgekar. Fast and accurate text classification via multiple linear discriminant projections. *The VLDB Journal*, 12:170–185, 2003.

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

[13] O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[14] Y. Chen, G. Wang, and S. Dong. Learning with progressive transductive support vector machine. In *Pattern Recognition Letters*, volume 24, pages 1845–1855. Elsevier, 2003.

[15] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[16] J. Dai, C. Chang, F. Mai, D. Zhao, and W. Xu. On the svmpath singularity. *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

[17] J. Dai and F. Mai. On the svmpath initialization. *Signal Processing*, 92(5):1258–1267, 2012.

[18] 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, October 2005.

[19] 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, 2006.

[20] G. B. Dantzig and P. Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, January 1960.

[21] D. DeCoste and K. Wagstaff. Alpha seeding for support vector machines. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 345–349, 2000.

[22] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(5):407–499, 2004.

[23] I. El-Naqa, Y. Yang, M. N. Wernick, N. P. Galatsanos, and R. M. Nishikawa. A support vector machine approach for detection of microcalcifications. *IEEE Transactions on Medical Imaging*, 21(12):1552–63, December 2002.

[24] R. Fan, X. Wang, and C.J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[25] R.E. Fan, P.H. Chen, and C.J. Lin. Working set selection using second order information for training support vector machines. *The Journal of Machine Learning Research*, 6:1889–1918, 2005.

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

[27] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *The Journal of Machine Learning Research*, 2:243–264, 2002.

[28] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, 1987.

[29] Jerome Friedman and Trevor Hastie. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, December 2007.

[30] J. Giesen, M. Jaggi, and Z. Eth. Approximate regularization paths for l2-loss support vector machines. 2009.

[31] P. E. Gill, W. Murray, and M. A. Saunders. User's guide for qpopt 1.0: A fortran package for quadratic programming, 1995.

[32] P. E. Gill, W. Murray, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization, 1997.

[33] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Inertia-Controlling Methods for General Quadratic Programming. *SIAM Review*, 33(1):1, 1991.

[34] T. Glasmachers and C. Igel. Maximum-gain working set selection for SVMs. *The Journal of Machine Learning Research*, 7:1437–1466, 2006.

[35] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, pages 1–33, 1983.

[36] G. H. Golub and Ch. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore and London, 3rd edition, 1996.

[37] H. Harbrecht, M. Peters, and R. Schneider. On the low-rank approximation by the pivoted Cholesky decomposition. *Applied Numerical Mathematics*, 62(4):428—-440, 2012.

[38] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *The Journal of Machine Learning Research*, 5(1):1391–1415, 2004.

[39] E. V. Haynsworth. Determination of the inertia of a partitioned hermitian matrix. *Linear Algebra and its Applications*, 1:73–81, 1968.

[40] J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, May 1994.

[41] T. Joachims. Making large-scale support vector machine learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[42] T. Joachims. Training linear SVMs in linear time. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[43] T. Joachims and C.N.J. Yu. Sparse kernel SVMs via cutting-plane training. *Machine Learning*, 76(2-3):179–193, July 2009.

[44] M. Karasuyama and I. Takeuchi. Suboptimal solution path algorithm for support vector machine. In *Proceedings of the 28th International Conference on Machine Learning ICML11*, pages 473–480, 2011.

[45] A. Kaylani, M. Georgiopoulos, M. Mollaghasemi, G. C. Anagnostopoulos, C. Sentelle, and M. Zhong. An adaptive multiobjective approach to evolving art architectures. *Trans. Neur. Netw.*, 21:529–550, April 2010.

[46] S.S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 2006.

[47] S.S. Keerthi and E.G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46(1):351–360, 2002.

[48] S.S. Keerthi, S.K. Shevade, and C. Bhattacharyya. A fast iterative nearest point algorithm for support vector machineclassifier design. *IEEE Transactions on Neural Networks*, 2000.

[49] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, 2001.

[50] Y. Koshiba and S. Abe. Comparison of L1 and L2 support vector machines. *Neural Networks, 2003. Proceedings of the*, 2003.

[51] B. Krishnapuram, J. Sichina, and L. Carin. Physics-based detection of targets in SAR imagery using support vector machines. *IEEE Sensors Journal*, 3(2):147–157, 2003.

[52] S. Lee and S.J. Wright. Decomposition Algorithms for Training Large-Scale Semiparametric Support Vector Machines. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, page 14. Springer, 2009.

[53] Y. Lee and O. L. Mangasarian. Rsvm: Reduced support vector machines. pages 5–7. SIAM, 2001.

[54] Y.J. Lee and O.L. Mangasarian. SSVM: A smooth support vector machine for classification. *Computational Optimization and Applications*, 20(1):5–22, 2001.

[55] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: a string kernel for svm protein classification. *Pacific Symposium on Biocomputing*, 575:564–75., January 2002.

[56] T. Liang and J. Gu. Active set iteration method for new l2 soft margin support vector machine. *Lecture Notes in Control and Information Sciences*, 345:663–669, 2006.

[57] C.J. Lin. Linear convergence of a decomposition method for support vector machines. Technical report, 2001.

[58] C.J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

[59] G. Loosli and S. Canu. Comments on the "Core Vector Machines: Fast SVM Training on Very Large Data Sets". *The Journal of Machine Learning Research*, pages 1–13, 2007.

[60] R. Lougee-Heimer. The common optimization interface for operations research. *IBM Journal of Research and Development*, 47(1):57–66, January 2003.

[61] O.L. Mangasarian and D.R. Musicant. Active support vector machine classification. *Advances in Neural Information Processing Systems*, 13:577 – 583, 2000.

[62] I. Maros. *Computational Techniques of the Simplex Method*. Kluwer Academic Publishers, Boston, 2003.

[63] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT Press, 2012.

[64] B. A. Murtagh, M.l A. Saunders, W. Murray, P. E. Gill, R. Raman, and E. Kalvelagen. Gams/minos: A solver for large-scale nonlinear optimization problems, 2002.

[65] D. R. Musciant and A. Feinberg. Active support vector machine classification. Technical Report 01-02, Department of Mathematics and Computer Science, Carleton College, Northfield, Minnesota, July 2002.

[66] J. Nocedal and S. J. Wright. *Numerical Optimization*. Operations Research. Springer, 1st edition, 1999.

[67] C.J. Ong, S. Shao, and J. Yang. An improved algorithm for the solution of the regularization path of support vector machine. *IEEE Transactions on Neural Networks*, 21(3):451–462, 2010.

[68] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *CVPR*, pages 130–136, San Juan, Puerto Rico, June 1997. 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97).

[69] J. Platt. Making large-scale support vector machine learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[70] J. Platt. *Microsoft Research*, 1998. Available at http://research.microsoft.com/~jplatt.

[71] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods-Support Vector Learning*, 1999.

[72] A. Rakotomamonjy and M. Davy. One-class SVM regularization path and comparison with alpha seeding. In *15th European Symposium on Artificial Neural Networks*, Bruges, Belgium, 2007.

[73] R. Rifkin. SVMFu: A fast, flexible architecture for training Support Vector Machines. *Signal Processing*, (September):1–2, 2001.

[74] D. Roobaert. Direct svm: a fast and simple support vector machine perceptron. In *Proceeding of IEEE, International Workshop on Neural Networks for Signal Processing*, pages 356–365, 2000.

[75] D. Roobaert. DirectSVM: A simple support vector machine perceptron. *The Journal of VLSI Signal Processing*, 32(1):147–156, 2002.

[76] M.H. Rusin. A revised simplex method for quadratic programming. *SIAM Journal on Applied Mathematics*, 20(2):143–160, 1971.

[77] K. Scheinberg. An efficient implementation of an active set method for SVMs. *Journal of Machine Learning Research*, 7:2237–2257, 2006.

[78] B. Scholkopf, J. C. Platt, J. Shawe-Taylor, J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–71, July 2001.

[79] C. Sentelle, G.C. Anagnostopoulos, and M. Georgiopoulos. A Fast Revised Simplex Method for SVM Training. *Pattern Recognition, 2008. ICPR 2008. 19th*, 2008.

[80] C. Sentelle, G.C. Anagnostopoulos, and M. Georgiopoulos. Efficient revised simplex method for svm training. *Neural Networks, IEEE Transactions on*, 22(10):1650–1661, 2011.

[81] 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.

[82] S. Shalev-Shwartz and Y. Singer. Efficient learning of label ranking by soft projections onto polyhedra. *The Journal of Machine Learning Research*, 7:1567–1599, 2006.

[83] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proceedings of the 24th international conference on Machine learning*, pages 807–814. ACM New York, NY, USA, 2007.

[84] A. Shilton, M. Palaniswami, D. Ralph, and A. C. Tsoi. Incremental training of support vector machines. *IEEE Transactions on Neural Networks*, 16(1):114–131, January 2005.

[85] A. Smola and B. Scholkopf. Sparse greedy matrix approximation for machine learning. In *Seventeenth International Conference on Machine Learning*, pages 911–918. Stanford, June 2000.

[86] A. J. Smola and S.V.N. Vishwanathan. LDL factorization for rank-k modifications of diagonal matrices.

[87] I. Steinwart, D. Hush, and C. Scovel. Training SVMs without offset. *The Journal of Machine Learning Research*, 12:141–202, 2011.

[88] F. E.H. Tay and L. Cao. Application of support vector machines in financial time series forecasting. *Omega: The International Journal of Management Science*, 29(4):309–317, 2001.

[89] I.W. Tsang, J.T. Kwok, and P.M. Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 2006.

[90] T. Van Gestel, J. Suykens, and D. Baestaens. Financial time series prediction using least squares support vector machines within the evidence framework. *IEEE Transactions on Neural Networks*, 12(4):809–821, 2001.

[91] R. J. Vanderbei. Loqo: An interior point code for quadratic programming. *Optimization Methods and Software*, 11(1):451 – 484, 1999.

[92] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer Series in Statistics. Springer-Verlag, New York, 1982.

[93] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer, 1st edition, 1995.

[94] T.L. Veldhuizen. Arrays in blitz++. In *Proceedings of the 2nd International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, pages 223–230. Springer-Verlag, 1998.

[95] S.V.N. Vishwanathan, A.J. Smola, and M.N. Murty. SimpleSVM. In *ICML*, pages 760–767, 2003.

[96] M. Vogt and V. Kecman. Active-set methods for support vector machines. In *Support Vector Machines: Theory and Applications*, volume 177 of *Studies in Fuzziness and Soft Computing*, pages 133–158. Springer Berlin Heidelberg, 2005.

[97] M. Vogt, U. Moissl, and J. Schaab. *Heart Rate Classification Using Support Vector Machines*, pages 716–723. Springer, 2006.

[98] X. Wang, P.K.H. Phua, and W. Lin. Stock market prediction using neural networks: Does trading volume help in short-term prediction? In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 4, pages 4pp2438–2442, 2003.

[99] T. Wen, A. Edelman, and D. Gorsich. A fast projected conjugate gradient algorithm for training support vector machines. *Joint Summer Research Conference on Fast Algorithms*, pages 1–19, 2003.

[100] K. Woodsend and J. Gondzio. Exploiting separability in large-scale linear support vector machine training. *Computational Optimization and Applications*, pages 1–29, 2009. 10.1007/s10589-009-9296-8.

[101] S. J. Wright. Optimization in machine learning. NIPS Workshop, December 2008.

[102] C. Wu, X. Wang, D. Bai, and H. Zhang. Fast svm incremental learning based on the convex hulls algorithm. *Computational Intelligence and Security, International Conference on*, 1:249–252, 2008.

[103] G. Zanghirati and L. Zanni. Large quadratic programs in training Gaussian support vector machines. *Technical Report*, 23:1–18, 2003.

[104] L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *J. Mach. Learn. Res.*, 7:1467–1492, 2006.

[105] X. Zhou, W. Jiang, Y. Tian, and Y. Shi. Kernel subclass convex hull sample selection method for svm on face recognition. *Neurocomput.*, 73:2234–2246, June 2010.

[106] M. Zhu and T. Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration, 2007.