Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning

by

Ryan Michael Rifkin

Submitted to the Sloan School of Management Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

© Massachusetts Institute of Technology 2002. All rights reserved.

Author . . . .
Sloan School of Management Science
August 1, 2002

Certified by . . . .
Tomaso Poggio
Uncas and Helen Whitaker Professor of Brain and Cognitive Sciences
Thesis Supervisor

Accepted by . . . .
James B. Orlin
Edward Pennell Brooks Professor of Operations Research
Co-director, MIT Operations Research Center

Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning

by

Ryan Michael Rifkin

Submitted to the Sloan School of Management Science
on August 1, 2002, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

This thesis shows that several old, somewhat discredited machine learning techniques are still valuable in the solution of modern, large-scale machine learning problems. We begin by considering Tikhonov regularization, a broad framework of schemes for binary classification. Tikhonov regularization attempts to find a function which simultaneously has small empirical loss on a training set and small norm in a Reproducing Kernel Hilbert Space. The choice of loss function determines the learning scheme. Using the hinge loss gives rise to the now well-known Support Vector Machine algorithm. We present SvmFu, a state-of-the-art SVM solver developed as part of thesis. We discuss the design and implementation issues involved in SvmFu, present empirical results on its performance, and offer general guidance on the use of SVMs to solve machine learning problems.

We also consider, and advocate in many cases, the use of the more classical square loss, giving rise to the Regularized Least Squares Classifiation algorithm. RLSC is "trained" by solving a single system of linear equations. While it is widely believed that the SVM will perform substantially better than RLSC, we note that the same generalization bounds that apply to SVMs apply to RLSC, and we demonstrate empirically on both toy and real-world examples that RLSC's performance is essentially equivalent to SVMs across a wide range of problems, implying that the choice between SVM and RLSC should be based on computational tractability considerations. We demonstrate the empirical advantages and properties of RLSC, discussing the tradeoffs between RLSC and SVMs. We also prove leave-one-out bounds for RLSC classification.

Next, we turn to the problem of multiclass classification. Although a large body of recent literature exists suggesting the use of sophisticated schemes involving joint optimization of multiple discriminant functions or the use of error-correcting codes, we instead advocate a simple "one-vs-all" approach in which one classifier is trained to discriminate each class from all the others, and a new point is classified according to which classifier fires most strongly. We present empirical evidence of the strength of this scheme on real-world problems, and, in the context of RLSC as the base classifier, compelling arguments as to why the simple one-vs-all scheme is hard to beat. We also present leave-one-out bounds for multiclass classification where the base learners belong to a broad class of Tikhonov regularizers.

Finally, we consider algorithmic stability, a relatively new theory that results in very elegant generalization bounds for algorithms which are "stable". We compare and contrast Tikhonov regularization, to which algorithmic stability applies, with Ivanov regularization, the form of regularization that is the basis for structural risk minimization and its related generalization bounds. We present some interesting examples which highlight the differences between the Tikhonov and Ivanov approaches, showing that the Tikhonov form has a much stronger stability than the Ivanov form. We show that a simple bagging scheme in which the training set is divided into equal-sized chunks, and the resulting classifiers averaged together, gives generalization bounds of the same order as Tikhonov regularization, even if the underlying binary classifiers are relatively unstable. Finally, we presult stability results for multiclass classification.

Thesis Supervisor: Tomaso Poggio
Title: Uncas and Helen Whitaker Professor of Brain and Cognitive Sciences

Acknowledgments

Thanks to Laura Baldwin for all of the LaTeX help. This thesis wouldn't have happened without her.

I would like to thank all the residents of Random Hall for making my five years of Graduate Resident Tutor so wonderful. If it weren't for you, I might have finished earlier, but more likely, I wouldn't have finished at all.

I would like to thank Professor Poggio for all the support and advice he has offered over the years. I would also like to thank all the past and current members of CBCL with which I've had so many enjoyable interactions: Sayan, Stan, Gene, Alex, Martin, Tony, Vinay, Constantine, Massimiliano, Gadi, Alessandro, and Theos.

Thanks to my mom and dad, my brother Sam and sister Rachel for all the support.

I would like to thank my friends for good times and support: Erica, Bronwen, Dianne, Danielle, Max, Dave, Laura, Ben, Amanda, Robyn, Mark, and of course and especially Anna.

I would like to thank everyone I should have thanked that I've forgotten to thank already.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of this introduction is to help you, the reader, understand what this thesis is about, and what you will learn if you choose to read it. The introduction will be informal and conversational in tone, and I will attempt to give a picture not only of the research itself, but of the choices and developments that led to this research. All the material discussed in this introduction is revisited at greater length in the thesis itself, and I will provide the references there.

To a large extent, this thesis is about Support Vector Machines and related algorithms. A Support Vector Machine is a machine learning algorithm for solving binary classification problems. When I first began the research that led to this thesis, in the fall of 1997, the SVM landscape was substantially different (and simpler) than it is now. Edgar Osuna's decomposition algorithm for Support Vector Machines had only recently been developed, and required the use of an expensive, third-party quadratic programming solver. Neither John Platt's SMO algorithm nor Thorsten Joachim's SVMLight implementation had been developed. Vapnik's seminal 1998 book (not to mention the flood of more recent books) had not yet been published, although an earlier, informal 1995 book was available. In my own lab, the Center for Biological and Computational Learning at MIT, Constantine Papageorgiou had successfully used Osuna's SVM implementation to find faces and pedestrians in images. There was a sense that SVMs outperformed a wide variety of other binary classification algorithms, such as neural networks, classification trees, Fisher discriminants, and so forth. There was also a sense that SVMs rested on a strong theoretical foundation, based on Vapnik's theory of VC-dimension.

One of the primary problems with SVMs was the lack of a good, fast, freely-distributable implementation. Osuna's code could often solve large problems, but it was hard to work with, and we could not freely distribute it because it used an expensive third-party library to solve the quadratic programming problems arising as intermediate steps. In 1998, John Platt generated a lot of excitement in the SVM community when he published his Sequential Minimal Optimization algorithm, or SMO for short. By solving subproblems consisting of only two examples at a time, which could themselves be solved analytically, the need for a third-party QP solver was obviated. Platt made very impressive claims for his algorithm, stating that for linear SVMs, SMO can be "1500 times faster than the Projected Conjugate Gradient chunking algorithm." Reading his paper, it was difficult to tell precisely what this PCG chunking algorithm was, how carefully it was implemented, and so forth; Platt was unable to release any of his code, as it was proprietary to Microsoft, where Platt is a researcher. In my own attempts to replicate his results, I quickly discovered a primary problem with SMO — as written, it suggested recomputing kernel products between pairs of points whenever they were needed. For high-dimensional data sets, this quickly became the crippling bottleneck for the algorithm. When I implemented the SMO algorithm myself and tested it on a face recognition task, it was substantially *slower* than the (already slow) code Osuna had written. This was my first encounter with the problems that plague the field of machine learning — extreme hype, sloppy experimental techniques, and irreproducible results.[1]

[1]The hype lives on. In the May/June 2002 issue of MIT's Technology Review Magazine, one of their five "patents to watch" is US Patent #6327581, granted to Microsoft for "Methods and Apparatus for Building a Support Vector Machine Classifier." From the Technology Review blurb (page 73):

> Computer scientists have long looked to machine learning methods as a way to "teach" computers new tasks; algorithms called "support vector machines" can be trained to produce programs that sort objects into different categories. But the algorithms have been too slow to be practically applied to very large problems like sorting text files. John Platt, the leader of Microsoft Research's Signal Processing group, created an original way to speed up support vector machines by a factor of 1,000, making it

I did not feel that SMO "solved" the problem of fast SVM training, and turned to the task of building a better SVM solver. I wanted an implementation that I could freely distribute, so I was unwilling to use a non-free quadratic programming solver. At the time, I was not aware of any industrial strength freely available QP solvers, and I did not want to write my own. I was therefore attracted to the idea of using Plat's SMO algorithm in some way. Eventually, I came up with the basic idea that became SvmFu, which is a major part of this thesis: to decompose the problem into subproblems as Osuna did, but to use an SMO-style algorithm to solve the resulting subproblems.

During the process of designing, implementing and reimplementing SvmFu, a number of other powerful SVM solvers became available, notably Thorsten Joachims' SVMLight system and Ronan Collobert's SVMTorch system. These systems both, in their fashion, solved the problems that remained in SVM training. They are both decomposition algorithms, and they both include powerful caching systems to avoid recomputation of kernel products. Neither of these systems takes SvmFu's specific approach of solving relatively large subproblems using an SMO-style algorithm. I attempted to take the best ideas from each of these applications and incorporate them into my own work. At the end of development, SvmFu is as fast or faster than these systems, but not so much more powerful as to be an "enabling technology"; roughly speaking, if one of the three systems will train an SVM in a reasonable amount of time, all three will.

The problem of training SVMs can now be considered "solved" in the following sense: although engineering and algorithmic enhancements may improve training speed somewhat, it is unlikely that further improvements will be substantial. With this in mind, I decided to focus on other *problems*, different from the standard SVM training problem — the goal being to find an approach that was as or more accurate as SVMs, while simultaneously being much much faster. I was inspired by an influential paper by Theodoros Evgeniou, Massimiliano Pontil, and my advisor Tomaso Poggio,

practical for the first time to tackle such problems.

which considered general regularization problems in the following form, known as Tikhonov regularization:

$$\sum_{i=1}^{\ell} V(f(\mathbf{x_i}), y_i) + \lambda ||f||_K^2 \tag{1.1}$$

Here, V is a user-supplied *loss function*, λ is a user-supplied *regularization constant*, and $||f||_K^2$ is the norm of f squared measured in a Reproducing Kernel Hilbert Space (see Appendix A). This equation is one of the central ones of this thesis, and will be explored in many different ways in the succeeding chapters. One major of contribution of the Evgeniou et al. paper was to notice that different learning schemes could be obtained simply by varying V. The "ideal" choice of V would be the 0-1 loss:

$$V(f(\mathbf{x}), y) = \begin{cases} 0 & \text{if } \text{sign}(f(\mathbf{x})) = y \\ 1 & \text{otherwise} \end{cases} \tag{1.2}$$

This is of course what we are really interested in, making as few mistakes as possible. Unfortunately, trying to optimize the 0-1 loss directly leads to nonconvex, intractable problems, so we instead consider *smooth* loss functions V that upper-bound the 0-1 loss. Choosing V to be a function known as the *hinge loss* leads to (almost) standard Support Vector Machines. I was also inspired by Olvi Mangasarian's work on Proximal Support Vector Machines, where he chose V to be the *square loss* $(y - f(\mathbf{x}))^2$. Mangasarian presented a method for solving *linear* PSVMs extremely rapidly. I was a bit put off by the name Proximal Support Vector Machines. There are no support vectors, so the name makes no sense. In the context of regression problems, the exact same algorithm has been in use since at least the late 1990's. In fact, Tikhonov developed his approach to regularization *assuming* that the loss function was the square loss (although he did not use the norm in an RKHS as his "stabilizer"). The primary contributions of Mangasarian's work were not to invent the algorithm (or, equivalently, to pioneer the use of the square loss), but to show that the linear problem could be solved very quickly, and to demonstrate on a number of benchmark datasets that the two loss functions produced very similar results. I renamed the algorithm Regularized Least-Squares Classification, or RLSC for short. I present a simpler analysis of RLSC than the one given by Mangasarian, who needlessly introduces a dual form (possibly in order to make the analogy to SVMs stronger). I introduce additional methods for solving the linear problem quickly, and in particular, I show how to use Conjugate Gradient to solve the linear problem quickly for text classification problems, where the input data are very high-dimensional but sparse.

I then turned to an analysis of nonlinear RLSC. Here, the situation is reversed, and solving an SVM is actually much faster than solving a general nonlinear RLSC problem. For this reason, I considered a number of different *approximations* to the full RLSC problem. I found that an approach known as the *Nystrom approximation*, suggested by Williams and Seeger in the context of Gaussian process classification, did not work well for me (I was entirely unable to replicate the extremely strong results they reported for the method), and developed a different approximation based on choosing a subset of the data in advance that I would use to "represent" my function. This approximation worked very well, and may represent a useful alternative to standard SVMs for nonlinear problems. It has several advantages over standard SVMs, including the fact that the precise amount of memory and computation required can be stated in advance, and that solving multiclass classification problems is only slightly more expensive than solving a single binary problem (see below). In this part of the thesis, I also prove leave-one-out bounds for Regularized Least Squares Classification; these bounds have a similar feel to a very general set of bounds for kernel classifiers developed by Jaakkola and Haussler, but the details are somewhat different. Additionally, we are able to exploit the strong geometrical constraints inherent in the RLSC formulation to give a particularly appealing form to the bounds, that makes explicit the connection between leave-one-out bounds and notions of algorthmic stability.

I next turned my attention to the problem of multiclass classification. The most obvious approach to combining binary classifiers (such as SVMs or RLSC) into a multiclass classification system is a simple "one-vs-all", or OVA approach: for each class, we train a classifier to discriminate that class from all other classes, and classify new points according to which of the classifiers fires most strongly. This approach has been "invented" numerous times by differing researchers. In the past few years, there has been a small cottage industry of papers describing newer, more sophisticated methods for multiclass classification. In my own work on multiclass text document and cancer classification, I consistently found that these sophisticated approaches performed no better than the OVA scheme, but reviewers often seemed surprised by this, and referred me to the relevant papers "showing" that other approaches offered superior performance compared to OVA. For this reason, this thesis includes a substantive review of the multiclass classification literature, considering in detail a number of "sophisticated" methods for multiclass classification. Without exception, we find that either there is no experimental evidence to support a substantial advantage over a simple OVA scheme, or that the experiments are poorly controlled and do not the support the conclusions made; in some cases, we run our own experiments to illustrate this point. We then try to understand *why* the OVA scheme performs so well in comparison to other schemes. We find that for many of the other schemes, the theory points to their performing well *if the outputs from various binary classifiers are "decorrelated"* in a certain sense. Running experiments, we find that the underlying binary classifiers that are combined in these multiclass schemes are extremely highly correlated, indicating that the complicated schemes cannot use decorrelation to improve on OVA. Additionally, in the specific context where RLSC is the underlying binary learner, we present some intriguing theoretical arguments that exploit the *linearity* inherent in RLSC to illustrate why other schemes are unlikely to outperform OVA, and, directly connected to this, show how we can solve a multiclass RLSC problem using approximately the same amount of computation as a single binary problem. Finally, we present leave-one-out bounds for multiclass classification.

In the final chapter, we turn to the notion of algorithmic stability, an elegant new method of proving generalization bounds for algorithms. Traditional approaches to generalization bounds rely on the notion of "structure": they show that the space of functions an algorithm selects from is not too large in a certain sense, and that therefore, with high probability, *all* functions in that space will have empirical performance close to their true performance on unseen future data. Given this, it makes sense to pick the function in the space with the best empirical performance. The algorithmic stability approach, on the other hand, shows that if the particular function selected by an algorithm is *stable*, in the sense that the function will not change much when the dataset is perturbed slightly, then the function will have close agreement between empirical and future performance (with high probability). While the "structural" approach applies directly to Ivanov regularization algorithms (a different form of regularzation discussed in Chapter 5), the algorithms we actually use (SVMs and RLSCs) are Tikhonov regularization algorithms, and algorithmic stability applies directy to them. We use the theory of algorithmic stability to explore the connections between Tikhonov and Ivanov regularization. In particular, we show that Tikhonov algorithms automatically live in a "stucture", so structural bounds can apply to them. In contrast, we show that Ivanov regularization algorithms are not necessarily stable. We also show that certain simple bagging algorithms are stable, yielding a nice explanation for the power of bagging methods. Finally, we present algorithmic stability-based bounds for multiclass classification that operate by combining the outputs from stable binary classifiers.

The remainder of this thesis will be written in a more technical, formal style, with the use of "we" rather than "I."

Chapter 2

Theory and Implementation of Support Vector Machines

2.1 Introduction

We introduce SvmFu, a new algorithm for training Support Vector Machines. SvmFu was designed for high performance on large datasets, providing both a fast algorithm and the flexibility to easily adapt its operating characteristics to individual problems. Conceptually, SvmFu can be viewed as a unification of ideas presented by Osuna [83], Platt [85], and Joachims [57], with additional extensions. We describe and justify our approach, and present experimental results exploring SvmFu's performance and comparing SvmFu to other popular codes.

In recent years, the Support Vector Machine (SVM) has become an important technique in machine learning [20, 115, 116]. Loosely speaking, an SVM for classification attempts to find a hyperplane that maximizes the margin between positive and negative examples, while simultaneously minimizing training set misclassifications (for excellent introductions, see [16] or [27]).

SVMs are motivated by strong theoretical arguments, and their observed empirical performance is extremely good. However, training an SVM requires the solution of a quadratic program in as many variables as there are data points in the training set. Worse still, the quadratic program has a dense (and often full rank) Hessian matrix. For an SVM on 20,000 data points, representing the entire Hessian would require $1.6 * 10^8$ (1.6 Gigabytes) of RAM, assuming each entry was stored as a 4-byte floating point number. The memory requirements of a direct approach to training an SVM make solving the QP directly with standard, off-the-shelf QP codes [80, 113] impossible. Instead, we must look to specialized methods that avoid representing the entire Hessian explicitly.

The contributions of this chapter are as follows. In Section 2.2, we review the mathematics associated with Support Vector Machines, presenting a particularly clear derivation of SVMs from the standpoint of regularization theory, and comparing and contrasting this approach to the more standard geometric derivation. In Section 2.3, we discuss previous approaches to training large-scale SVMs. In Section 2.4, we present the SvmFu algorithm, contrasting it with previous methods. In Section 2.5, we present experimental results on SvmFu and other algorithms. In Section 2.6, we present accumulated "folk wisdom" and "rules of thumb" relating to SVMs — this knowledge is quite valuable to people who actually want to use SVMs in practice or are interested in writing their own large-scale SVM implementations.

2.2 Support Vector Machines

This is an overview of the mathematics of Support Vector Machines, focussing on the details necessary to understand the algorithms used to train SVMs. More details can be found in wide variety of references [16, 27, 82, 116, 98, 53]. Whereas most developments start from a geometric viewpoint emphasizing separating hyperplanes and "margin", we begin with the idea of regularization, which allows us to easily develop both the primal and dual formulations in a very clean and general way.

2.2.1 From Regularization Theory to Quadratic Programs

We are given ℓ examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_\ell)$, with $\mathbf{x}_i \in \mathbf{R}^n$ and $y_i \in \{-1, 1\}$ for all i. As we discussed in Chapter 1, the problem of learning a function that will generalize well on new examples is ill-posed. The classical approach to restoring well-posedness to learning is regularization theory [48, 31, 28, 111, 119, 8, 9]. This leads to the following regularized learning problem:

$$\min_{f \in \mathcal{H}} \frac{1}{\ell} \sum_{i=1}^{\ell} V(y_i, f(\mathbf{x}_i)) + \lambda ||f||_K^2. \tag{2.1}$$

Here, $||f||_K^2$ is the norm in a Reproducing Kernel Hilbert Space $\mathcal{H}$ defined by a positive definite kernel function K (see Appendix A), V is a *loss function* indicating the penalty we pay for guessing $f(\mathbf{x}_i)$ when the true value is y, and λ is a regularization parameter quantifying our willingness to trade off accuracy of classification for a function with small norm in the RKHS $\mathcal{H}$.

The classical SVM arises by considering the specific loss function[1]

$$V(f(\mathbf{x}), y) \equiv (1 - yf(\mathbf{x}))_+, \tag{2.2}$$

where

$$(k)_+ \equiv \max(k, 0). \tag{2.3}$$

This loss function, shown in Figure 2-1, is often referred to as the *hinge loss*. The hinge loss has a simple but compelling justification. Remembering that for each example i, y_i is either 1 or -1, a reasonable goal is to find a function such that $f(\mathbf{x_i})$ is large and positive when $y_i = 1$ and is large and negative when $y_i = -1$. The hinge loss enforces this goal. If $y_i f(\mathbf{x_i})$ is at least 1, we pay no penalty for point i. If $y_i f(\mathbf{x_i}) < \mathbf{1}$, we pay a penalty linear in the amount by which we fail to satisfy the constraint. The quantity $y_i f(\mathbf{x_i})$ is also known as the *margin* (see below). The now-classical SVM algorithm as developed by Vapnik et al. [12, 115, 116] uses the hinge loss, and this shall be our focus for most of present chapter.

[1] Since we are interested in solving binary classification problems, the most natural loss would be the *0-1 loss*, which simply counts the number of misclassifications. Using this loss function directly in the training problem leads to an NP-complete, and therefore intractable, optimization problem. Some care must be taken when deriving generalization bounds — we need these bounds to apply to the 0-1 loss, not to the hinge loss we used to train the SVM. See Chapter 5 for additional discussion of this point.

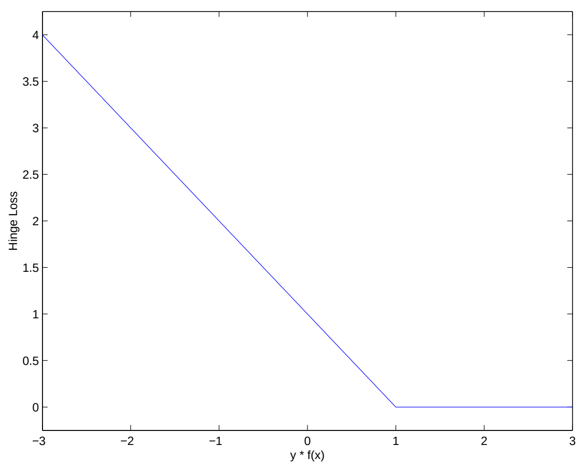


Figure 2-1: The hinge loss $V(f(x), y) = (1 - yf(x))_+$.

Using the hinge loss, our regularization problem becomes

$$\min_{f \in \mathcal{H}} \frac{1}{\ell} \sum_{i=1}^{\ell} (1 - y_i f(\mathbf{x}_i))_+ + \lambda ||f||_K^2. \tag{2.4}$$

In order to make the problem easier to work with, we introduce slack variables ξ_i, corresponding to the penalty we pay at point i. With this definition, our problem becomes:

$$\begin{aligned} &\min_{f \in \mathcal{H}} && \tfrac{1}{\ell} \textstyle\sum_{i=1}^{\ell} \xi_i + \lambda ||f||_K^2 && && (2.5) \\ &\text{subject to :} && y_i f(\mathbf{x}_i) \geq 1 - \xi_i && i = 1, \ldots, \ell && (2.6) \\ & && \xi_i \geq 0 && i = 1, \ldots, \ell && (2.7) \end{aligned}$$

Under quite general conditions (see Appendix B) it can be shown that the solution f^* to the above regularization problem has the form

$$f^*(\mathbf{x}) = \sum_{i=1}^{\ell} c_i K(\mathbf{x}, \mathbf{x}_i). \tag{2.8}$$

Therefore, in order to find f^*, we need only find the optimal c_i. Using basic facts about RKHS, defining $\mathbf{c} \equiv [c_1 \ldots c_\ell]^T$, and defining the matrix K as the ℓ-by-ℓ matrix satisfying $K_{ij} \equiv K(\mathbf{x}_i, \mathbf{x}_j)$, we arrive at a constrained quadratic programming problem:

$$\min_{\mathbf{c}\in\mathbf{R}^\ell} \quad \tfrac{1}{\ell}\textstyle\sum_{i=1}^{\ell}\xi_i + \lambda \mathbf{c}^T K \mathbf{c} \tag{2.9}$$

$$\text{subject to}: \quad y_i \textstyle\sum_{j=1}^{\ell} c_j K(x_i, x_j) \geq 1 - \xi_i \quad i = 1, \dots, \ell \tag{2.10}$$

$$\xi_i \geq 0 \qquad i = 1, \dots, \ell \tag{2.11}$$

As we will see, this problem is essentially identical to a nonlinear version of the standard "primal" problem derived in most references. The only difference is that the standard SVM includes an additional nonregularized "bias" term b (see Subsection 2.6.2 for a discussion of the role of the bias parameter), so that the function f^* takes the form

$$f^*(\mathbf{x}) = \sum_{i=1}^{\ell} c_i K(\mathbf{x}, \mathbf{x}_i) + b. \tag{2.12}$$

With this change, the quadratic program becomes

$$\min_{\mathbf{c}\in\mathbf{R}^\ell, \xi\in\mathbf{R}^\ell} \quad \tfrac{1}{\ell}\textstyle\sum_{i=1}^{\ell}\xi_i + \lambda \mathbf{c}^T K \mathbf{c} \tag{2.13}$$

$$\text{subject to}: \quad y_i (\textstyle\sum_{j=1}^{\ell} c_j K(x_i, x_j) + b) \geq 1 - \xi_i \quad i = 1, \dots, \ell \tag{2.14}$$

$$\xi_i \geq 0 \qquad i = 1, \dots, \ell \tag{2.15}$$

We derive the Wolfe dual quadratic program using Lagrange multiplier techniques [6, 10, 75, 33]. Because the primal is a feasible convex quadratic programming problem, strong duality holds — the dual problem will also be feasible and convex, and the optimal objective values of the primal and dual problem will be identical [125, 4]. Proceeding, our Lagrangian is

$$L(\mathbf{c}, \xi, b, \alpha, \zeta) = \frac{1}{\ell}\sum_{i=1}^{\ell}\xi_i + \lambda \mathbf{c}^T K \mathbf{c} - \sum_{i=1}^{\ell}\alpha_i (y_i \{\sum_{j=1}^{\ell} c_j K(x_i, x_j) + b\} - 1 + \xi_i) - \sum_{i=1}^{\ell}\zeta_i \xi_i, \tag{2.16}$$

which we want to minimize with respect to $\mathbf{c}$, b, and ξ, and maximize with respect to α and ζ, subject to the constraints of the primal problem and nonnegativity constraints on α and ζ. We will eliminate the primal variables by taking the derivatives with respect to each of them in turn and setting these derivatives to zero, thereby deriving additional dual constraints. We first take the derivatives with respect to b and ξ:

$$\frac{\partial L}{\partial b} = 0 \implies \sum_{i=1}^{\ell} \alpha_i y_i = 0 \tag{2.17}$$

$$\frac{\partial L}{\partial \xi_i} = 0 \implies \frac{1}{\ell} - \alpha_i - \zeta_i = 0 \tag{2.18}$$

$$\implies 0 \le \alpha_i \le \frac{1}{\ell} \tag{2.19}$$

The last equation was derived using the nonnegativity of α_i and ζ_i. By using these identities, we can write the following reduced Lagrangian in terms of the remaining variables:

$$L^R(\mathbf{c}, \alpha) = \lambda \mathbf{c}^T K \mathbf{c} - \sum_{i=1}^{\ell} \alpha_i (y_i \sum_{j=1}^{\ell} c_j K(x_i, x_j) - 1) \tag{2.20}$$

It is notationally simplest to take the derivative of L^R with respect to the entire $\mathbf{c}$ vector simultaneously. Defining Y to be the diagonal matrix with $Y_{ii} equiv y_i$:

$$\frac{\partial L^R}{\partial \mathbf{c}} = 0 \implies 2\lambda K \mathbf{c} - KY\alpha = 0 \tag{2.21}$$

$$\implies \mathbf{c} = \frac{Y\alpha}{2\lambda}. \tag{2.22}$$

The last equation assumed that the kernel was positive definite and that the matrix K was invertible. If K is not invertible, there will in general exist multiple solutions, but $c = \frac{\alpha \odot \mathbf{y}}{2\lambda}$ will always be a solution.[2] Substituting in our expression for $\mathbf{c}$, we are left with the following dual program:

$$\max_{\alpha \in \mathbf{R}^{\ell}} \quad \textstyle\sum_{i=1}^{\ell} \alpha_i - \frac{1}{(2\lambda)^2} \alpha^T Q \alpha \tag{2.23}$$

$$\text{subject to :} \quad \textstyle\sum_{i=1}^{\ell} y_i \alpha_i = 0 \tag{2.24}$$

$$0 \le \alpha_i \le \tfrac{1}{\ell} \qquad i = 1, \ldots, \ell \tag{2.25}$$

[2]This point was somewhat misunderstood in Osuna's thesis [82]. Osuna was attempting to find additional solutions to the SVM problem which used fewer support vectors (see below). He started with the dual program, and rederived a "nonlinear primal" which is essentially the QP we started with. He then remarked (correctly) that the c_i in this program do not necessarily have the same geometric interpretation as the α_i in the dual, but failed to note that if the matrix were invertible, there is no freedom in choosing the c_i given the α_i. (Rob Freund and Gert Cauwenbachs, personal communication.)

Here, Q is the matrix defined by the relationship

$$Q = YKY \Longleftrightarrow Q_{ij} = y_i y_j K(x_i, x_j). \tag{2.26}$$

2.2.2 Standard Notation

We derived our primal and dual quadratic programs starting from a formulation which is standard in regularization theory:

$$\min_{f \in \mathcal{H}} \frac{1}{\ell} \sum_{i=1}^{\ell} V(y_i, f(\mathbf{x}_i)) + \lambda ||f||_K^2. \tag{2.27}$$

In most of the SVM literature, these equations are reparametrized. Instead of the regularization parameter λ, regularization is controlled via a parameter C, defined using the relationship

$$C = \frac{1}{2\lambda\ell}. \tag{2.28}$$

Using this definition, the basic regularization problem becomes

$$\min_{f \in \mathcal{H}} C \sum_{i=1}^{\ell} V(y_i, f(\mathbf{x}_i)) + \frac{1}{2} ||f||_K^2. \tag{2.29}$$

The parameter C also controls the tradeoff between classification accuracy and the norm of the function. The derivation of the primal and dual programs is nearly identical. The first version is more natural from the standpoint of regularization theory (see Chapter 5), but the version using C is much more prevalent in the SVM literature. Additionally, the latter version has the nice property that $\mathbf{c} = Y\alpha$ is a solution, so the equivalence between $\mathbf{c}$ and α is even closer. In this form, the primal problem becomes

$$\begin{array}{lll}
\min\limits_{\mathbf{c} \in \mathbf{R}^\ell, \xi \in \mathbf{R}^\ell} & C \sum_{i=1}^{\ell} \xi_i + \frac{1}{2} \mathbf{c}^T K \mathbf{c} & \qquad (2.30) \\
\text{subject to}: & y_i(\sum_{j=1}^{\ell} c_j K(x_i, x_j) + b) \geq 1 - \xi_i \quad i = 1, \ldots, \ell & \qquad (2.31) \\
 & \xi_i \geq 0 \quad i = 1, \ldots, \ell & \qquad (2.32)
\end{array}$$

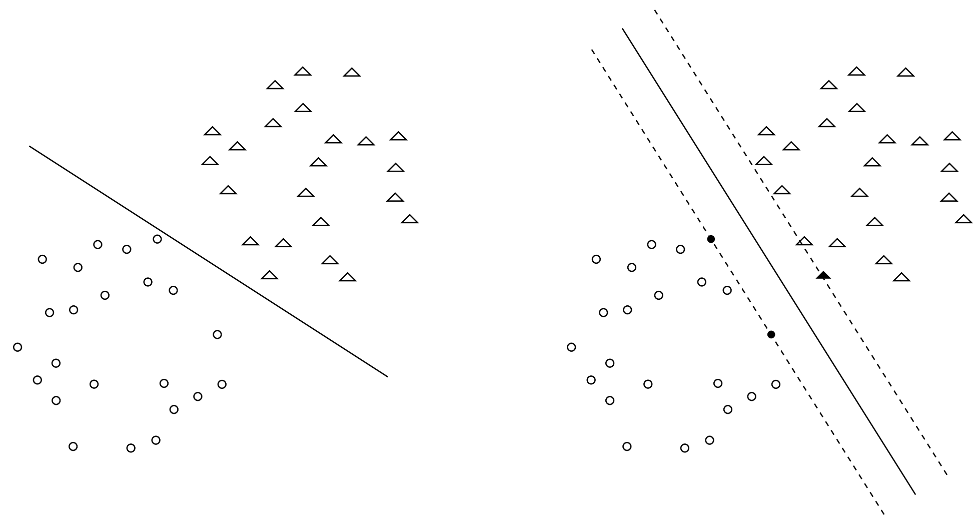


Figure 2-2: Two hyperplanes with different margin. Intuitively, the large margin hyperplane (b) seems likely to perform better on future examples than the much smaller margin hyperplane (a).

The dual problem becomes

$$\max_{\alpha \in \mathbf{R}^\ell} \quad \textstyle\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2}\alpha^T Q \alpha \tag{2.33}$$

$$\text{subject to :} \quad \textstyle\sum_{i=1}^{\ell} y_i \alpha_i = 0 \tag{2.34}$$

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, \ell \tag{2.35}$$

2.2.3 Geometric Interpretation

The "traditional" approach to developing the mathematics of SVM is to start with the concepts of *separating hyperplanes* and *margin.* The theory is usually developed in a linear space, beginning with the idea of a perceptron [92, 79], a linear hyperplane that separates the positive and the negative examples. Defining the margin as the distance from the hyperplane to the nearest example, the basic observation is that intuitively, we expect a hyperplane with larger margin to generalize better than one with smaller margin (Figure 2-2).

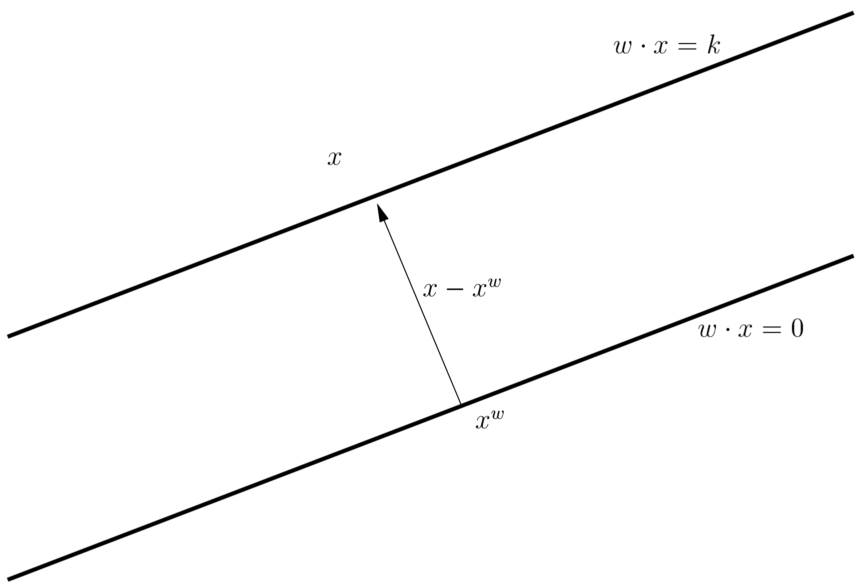


Figure 2-3: Derivation of the optimal separating hyperplane w.

We denote our hyperplane by $\mathbf{w}$, and we will classify a new point $\mathbf{x}$ via the function

$$f(x) = \text{sign}\ (\mathbf{w} \cdot \mathbf{x}). \tag{2.36}$$

Given a separating hyperplane $\mathbf{w}$ (geometrically, $\mathbf{w}$ is *normal* to the separating hyperplane, but we refer to $\mathbf{w}$ as the hyperplane), we let $\mathbf{x}$ be a data point closest to $\mathbf{w}$, and we let $\mathbf{x}^\mathbf{w}$ be the unique point on $\mathbf{w}$ that is closest to x. Obviously, finding a maximum margin $\mathbf{w}$ is equivalent to maximizing $||\mathbf{x} - \mathbf{x}^\mathbf{w}||$. We have, for some value of k[3] (see Figure 2-3),

$$\mathbf{w} \cdot \mathbf{x} = k \tag{2.37}$$

$$\mathbf{w} \cdot \mathbf{x}^\mathbf{w} = 0 \tag{2.38}$$

$$\implies \mathbf{w} \cdot (\mathbf{x} - \mathbf{x}^\mathbf{w}) = k \tag{2.39}$$

[3]We assume without loss of generality that $k > 0$; if $k < 0$, the same argument applies with slight modifications.

Noting that the vector $\mathbf{x} - \mathbf{x}^{\mathbf{w}}$ is parallel to the normal vector w,

$$\begin{aligned}
\mathbf{w} \cdot (\mathbf{x} - \mathbf{x}^{\mathbf{w}}) &= \mathbf{w} \cdot \left(\frac{||\mathbf{x} - \mathbf{x}^{\mathbf{w}}||}{||\mathbf{w}||} \mathbf{w} \right) && (2.40) \\
&= ||\mathbf{w}||^2 \frac{||\mathbf{x} - \mathbf{x}^{\mathbf{w}}||}{||\mathbf{w}||} && (2.41) \\
&= ||\mathbf{w}|| \; ||\mathbf{x} - \mathbf{x}^{\mathbf{w}}|| && (2.42) \\
&\Longrightarrow ||\mathbf{w}|| \; ||(\mathbf{x} - \mathbf{x}^{\mathbf{w}})|| = k && (2.43) \\
&\Longrightarrow ||\mathbf{x} - \mathbf{x}^{\mathbf{w}}|| = \frac{k}{||\mathbf{w}||} && (2.44)
\end{aligned}$$

The parameter k is a nuisance parameter. Given any $\mathbf{w}$, if we consider instead $c\mathbf{w}$ for a positive constant c, $||\mathbf{x} - \mathbf{x}^{\mathbf{w}}||$ will remain unchanged, but $||\mathbf{w}||$, $\mathbf{w} \cdot \mathbf{x}$, and k will all change by a factor of c. Therefore, without loss of generality, we may fix k to 1.[4] Essentially, the only choice we can make geometrically is the direction of the normal to the separating hyperplane. Once we fix k to be 1, we see that maximizing $||\mathbf{x} - \mathbf{x}^{\mathbf{w}}||$ is equivalent to maximizing $\frac{1}{||w||}$, which in turn is equivalent to minimizing $||w||$. We can now define the margin as the distance between the hyperplanes $\mathbf{w} \cdot \mathbf{x} = 0$ and $\mathbf{w} \cdot \mathbf{x} = 1$.

The SVM introduced by Vapnik includes an unregularized bias term b, leading to classification via a function of the form:

$$f(x) = \text{sign } (\mathbf{w} \cdot \mathbf{x} + b). \qquad (2.45)$$

This b term has become a part of the "classic" SVM formulation, and will be included throughout most of this chapter. See Sections 2.6.2 for further discussion of this point.

In practice, we want to be able to work with datasets that are not necessarily linearly separable, so we introduce slack variables ξ_i just as in the previous section. Unfortunately, this makes the interpretation of the "margin" somewhat difficult, which is a primary advantage of explaining and deriving SVMs in terms of regularization theory rather than geometrically. We can still define the margin as the distance be-

[4]Vapnik calls the set of hyperplanes for which $k = 1$ the set of *canonical separating hyperplanes.*

tween the hyperplanes $\mathbf{w} \cdot \mathbf{x} = 0$ and $\mathbf{w} \cdot \mathbf{x} = 1$, but this is no longer particularly satisfying.

With slack variables, the primal SVM problem becomes

$$\min_{\mathbf{w} \in \mathbf{R}^n, b \in \mathbf{R}} \quad C \sum_{i=1}^{\ell} \xi_i + \frac{1}{2}||\mathbf{w}||^2 \tag{2.46}$$

$$\text{subject to :} \quad y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 - \xi_i \quad i = 1, \ldots, \ell \tag{2.47}$$

$$\xi_i \geq 0 \qquad i = 1, \ldots, \ell \tag{2.48}$$

Historically, most developments began with this form, derived a dual program which was identical to the dual we derived above, and only then observed that the dual program required only dot products and that these dot products could be replaced with a kernel function (see Appendix A). In fact, Osuna's thesis [82] was perhaps the first to take the dual of the dual, recovering a formulation essentially identical to the primal we easily derived from a regularization theory standpoint.

In the linearly separable case, there is another geometric viewpoint for the optimal separating hyperplane. Consider all vectors connecting pairs of points in opposite class. Let v be the shortest such vector, and let x^+ and x^- be its positive and negative endpoints, respectively. Then, geometrically, the optimal separating hyperplane has a normal vector parallel (or anti-parallel) to v, and passes through $\frac{x^+ + x^-}{2}$. An algorithm based on this observation was derived by Vapnik in the 1970's [114], and another such algorithm with extensions to the nonseparable case with squared slack variables has recently been proposed by Keerthi et al. [60].

2.2.4 The SVM Dual

The SVM dual problem is substantially easier to solve than the primal, because the constraint structure is much simpler — whereas the primal has a dense inequality constraint for every data point, the dual has only a single dense equality constraint and a set of box constraints. Therefore, we shall now focus on the SVM dual problem, and particularly on aspects that are relevant for solving it..

For convenience, we restate the dual problem we shall focus on for most of the remainder of the chapter:

$$\begin{aligned} \max \quad & \textstyle\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j Q_{ij} && (2.49) \\ \text{subject to :} \quad & \textstyle\sum_{i=1}^{\ell} y_i \alpha_i = 0 && (2.50) \\ & 0 \leq \alpha_i \leq C, \ \forall i && (2.51) \end{aligned}$$

Here, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, where K is a user-supplied positive semidefinite *kernel function* [3, 120]. Choosing a kernel function is equivalent to choosing a (possibly high- or infinite-dimensional) *feature space* in which to embed the examples. Kernels commonly in use include linear, polynomial and radial basis function kernels (see Appendix A).

The solution of the SVM training problem is the vector of optimal α's. These induce a real-valued classification function[5], given by:

$$f(x) = \sum_{i=1}^{\ell} \alpha_i y_i K(x, x_i) + b \tag{2.52}$$

Only the α_i's which are nonzero appear in this expansion.[6] The corresponding data points are known as *support vectors*. It is obvious that the removal of non-support vectors from the training set does not affect the solution. For any i, if α_i is strictly between 0 and C, then point i is designated an *unbounded support vector* (USV).

The *bias term* b can be found using any α_i whose value is strictly between 0 and C:[7]

$$b = y_i - f(x_i) \tag{2.53}$$

In practice, most authors find b by averaging the value of b imputed by many different unbounded support vectors. Our software makes this choice as well, although we

[5]In actual classification problems, we use $sign(f(x))$ to get 0/1 classifications.

[6]In essence, we are solving for the c_i that solve the regularization problem introduced in Section 2.2.1; because we have parametrized the problem using C instead of λ, a valid solution is $c_i = \alpha_i y_i$.

[7]Under extremely mild conditions, such a point is guaranteed to exist (see [15] or [91]).

usually find that the standard deviation of the imputed values of b are all extremely close (10^{-8} or less), indicating that this choice is not especially important if the α's are being found with a high degree of accuracy.

C is a user-supplied regularization constant with a simple geometric interpretation. We are attempting to classify each point with a margin of at least 1 — i.e., we would like to satisfy, for each i, the condition

$$y_i f(x_i) \geq 1. \tag{2.54}$$

C is then the unit cost of misclassification of an example; if $y_i f(x_i) = m_i < 1$, we pay a penalty of $C(1 - m_i)$.

2.2.5 Optimality Conditions

The primal (Equations 2.31-2.32) and dual (Equations 2.50-2.51) are both convex quadratic programs. Therefore, optimal solutions to the primal and dual will satisfy *complementary slackness* conditions [6, 10, 75, 33]. Complementary slackness tells us important information about optimal solutions to the primal and dual problems. Consider the dual variable associated with a primal inequality constraint. At optimality, complementary slackness dictates that either the inequality constraint is satisfied as an equality, or the corresponding dual variable is zero. Combining this with the feasibility conditions for both problems, we can derive a complete set of conditions that a pair of optimal solutions to the primal and dual problem must satisfy:

$$\sum_{j=1}^{\ell} c_j K(\mathbf{x_i}, \mathbf{x_j}) - \sum_{j=1}^{\ell} y_i \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) = 0 \qquad i = 1, \ldots, \ell \tag{2.55}$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0 \tag{2.56}$$

$$C - \alpha_i - \zeta_i = 0 \qquad i = 1, \ldots, \ell \tag{2.57}$$

$$y_i(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) + b) - 1 + \xi_i \geq 0 \qquad i = 1, \ldots, \ell \tag{2.58}$$

$$\alpha_i[y_i(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) + b) - 1 + \xi_i] = 0 \qquad i = 1, \ldots, \ell \tag{2.59}$$

$$\zeta_i \xi_i = 0 \qquad i = 1, \ldots, \ell \tag{2.60}$$

$$\xi_i, \alpha_i, \zeta_i \geq 0 \qquad i = 1, \ldots, \ell \tag{2.61}$$

If we have $\mathbf{c}$, ξ, b, α and ζ satisfying the above conditions, we know that they represent optimal solutions to the primal and dual problems. These optimality conditions are also known as the Karush-Kuhn-Tucker (KKT) conditons.

We can simplify the KKT conditions substantially by examining the problem more closely. Consider a point i which is not a support vector. Then

$$\begin{aligned} \alpha_i = 0 \quad &\Longrightarrow \quad \zeta_i = C & (2.62)\\ &\Longrightarrow \quad \xi_i = 0 & (2.63)\\ &\Longrightarrow \quad y_i f(\mathbf{x_i}) \geq 1, & (2.64) \end{aligned}$$

where we have as usual defined

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} y_i \alpha_i K(\mathbf{x}, \mathbf{x_i}) + b. \tag{2.65}$$

Conversely, for a point i which satisfies $y_i f(\mathbf{x_i}) > 1$, we have

$$\begin{aligned} y_i f(\mathbf{x_i}) > 1 \quad &\Longrightarrow \quad \xi_i = 0 & (2.66)\\ &\Longrightarrow \quad \alpha_i[y_i(\sum_{j=1}^{\ell} y_i \alpha_i K(\mathbf{x_i}, \mathbf{x_j}) + b) - 1] = 0 & (2.67)\\ &\Longrightarrow \quad \alpha_i = 0. & (2.68) \end{aligned}$$

Now consider a point i which is a support vector at its upper bound, with $\alpha_i = C$:

$$\begin{aligned} \alpha_i = C \quad &\Longrightarrow \quad y_i(\sum_{j=1}^{\ell} y_i \alpha_i K(\mathbf{x_i}, \mathbf{x_j}) + b) - 1 + \xi_i = 0 & (2.69)\\ &\Longrightarrow \quad y_i f(\mathbf{x_i}) \leq 1. & (2.70) \end{aligned}$$

Also,

$$\begin{aligned} y_i f(\mathbf{x_i}) < 1 \implies & \xi_i > 0 & (2.71)\\ \implies & \zeta_i = 0 & (2.72)\\ \implies & \alpha_i = C & (2.73) \end{aligned}$$

Finally, we consider an unbounded support vector, a point i satisfying $0 < \alpha_i < C$. Using the same argument as when $\alpha_i = C$ (the only relevant fact about this argument was $\alpha_i > 0$; the fact that it was equal to C was not used), we again find that $y_i f(\mathbf{x_i}) \leq 1$. Using the same argument as when $\alpha_i = 0$, we again find that $\zeta_i > 0$, $\xi_i = 0$, and $y_i f(\mathbf{x_i}) \geq 1$. Combining these arguments, we find that for unbounded support vectors, $y_i f(\mathbf{x_i}) = 1$. There is no converse argument: knowing that $y_i f(\mathbf{x_i}) = 1$ does not necessarily tell us anything about α_i.

Putting together these results, we have the following "simplified" optimality conditions for the SVM training problem:

$$\begin{aligned} \alpha_i = 0 \implies & y_i f(x_i) \geq 1 & (2.74)\\ 0 < \alpha_i < C \implies & y_i f(x_i) = 1 & (2.75)\\ \alpha_i = C \implies & y_i f(x_i) \leq 1 & (2.76) \end{aligned}$$

Our goal is to find a set of α's (and the induced b) that satisfy these equations. In practice, we accept a solution vector of α's as optimal when all the optimality conditions are satisfied to within some tolerance τ.

We note the slight asymmetry in the arguments used to derive the simplified optimality conditions. For example, $y_i f(\mathbf{x_i}) < 1$ implies $\alpha_i = C$, but $\alpha_i = C$ only implies $y_i f(\mathbf{x_i}) \leq 1$. This is a consequence of the possibility of *degeneracy* in the solution; in particular, the fact that a point that satisfies the margin requirement exactly has no constraint on its possible α value.

It is in interpreting these reduced optimality conditions that the geometric development of SVMs becomes particularly valuable. Figure 2-4 provides an illustration

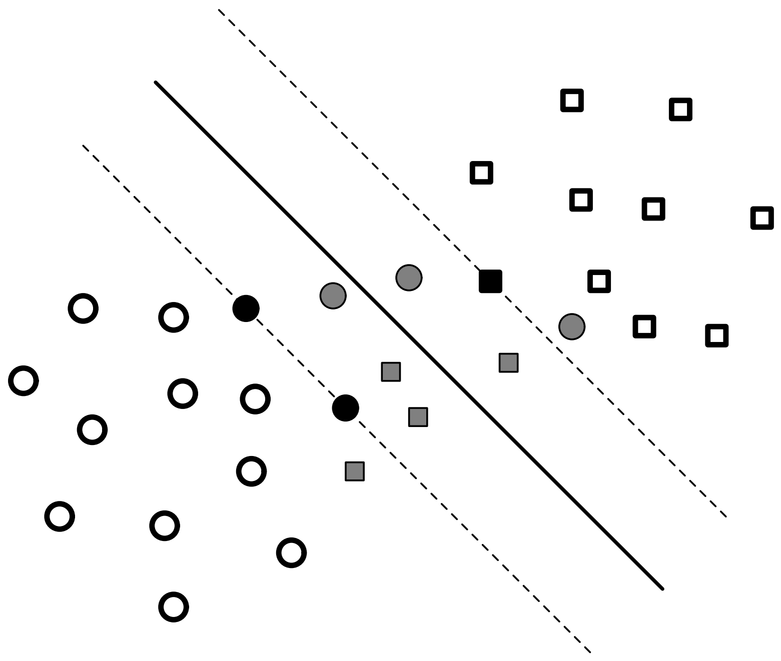


Figure 2-4: Geometric illustration of the reduced optimality conditions. The circles and the squares represent the two classes. The outlined points are not support vectors, and satisfy Equation 2.74. The solid points are unbounded support vectors, and satisfy Equation 2.75. The grey shaded points are support vectors at their upper bound C, and satisfy Equation 2.76.

of this principle. In this figure, the circles and the squares represent the two different classes. The outlined points are the points which satisfy the margin requirement. They satisfy Equation 2.74: $y_i f(x_i) \geq 1$, $\alpha_i = 0$, and they lie outside the margin on the correct side of the hyperplane. The solid black points are the unbounded support vectors. They satisfy the margin requirement with equality. For these points, Equation 2.75 holds, $y_i f(x_i) = 1$, and $0 < \alpha_i < C$. Finally, the grey shaded points are the bounded support vectors. They do not satisfy the margin requirement. For these points, Equation 2.76, and $\alpha_i = C$. It is important to note (see Figure) that the points which satisfy Equation 2.76 are not necessarily errors. They are points which fail to satisfy the margin requirement, but they may still lie on the correct side of the hyperplane, inside the margin. However, the converse is true — all points which are training errors satisfy Equation 2.76. For this reason, the number of bounded support vectors can be lower bounded by the number of training errors, a point we shall return to later.

2.3 Previous Work

We now consider previous attempts to solve the SVM problem in the form discussed above (Equations 2.50-2.51). Many modifications to this form have been proposed, often in order to simplify the problem. These include but are by no means limited to removing the bias term [37], penalizing the bias term (or, equivalently, adding an extra dimension of all 1's to the data [73]), and adding an extra dimension for each data point, thereby making the problem separable [36, 61]. Evaluating whether these alternatives work as well or better than the standard formulation is an open research question; Olvi Mangasarian has made progress in addressing these issues [74]. Chapter 3 is devoted entirely to studying a different particularly promising alternative form.

The coverage in this section is not meant to be exhaustive, but is meant to point out several key developments and landmark papers. In addition, we discuss some papers that we feel are of interest, but have not seemed to have a big impact in the field.

There exist polynomial time algorithms for solving large classes of convex programming problems, which include convex quadratic prorgramming problems as a special case [81]. For many quadratic programming problems, interior point methods have led to the most efficient algorithms in practice as well as theory. However, the quadratic programming problem arising from Support Vector Machines does not fit this pattern, for two main reasons. The first reason is that for large problems, the matrix Q is too large to store in memory, and is in general dense. The second reason is that the solution to an SVM is in general quite sparse — there are few nonzero coefficients relative to the number of data points. This is because every point which lives outside the margin on the correct side is guaranteed not to be a support vector. For this reason, we will prefer algorithms that start with an all-zero solution and attempt to modify α values as necessary. Interior point methods, on the other hand, will start with a solution that contains no zeros (satisfies $0 < \alpha_i < C$ for all i). These "no-zero" solutions are much more expensive to work with.

For the reasons just mentioned, it has long been recognized that direct attempts to solve the QP problem associated with SVMs will not scale to large problem sizes, primarily due to the size of the Q matrix. Vapnik [114, 115] first observed that because only the support vectors (those points with non-zero α_i's) appeared in the solution, remaining points could be removed from the training set. This led to the *method of portraits*, in which the large QP was solved as a series of smaller QPs, each one containing the support vectors from the previous iteration, as well as some new points that violated the optimality conditions. This had the advantage of not needing to store the entire Q matrix; however, one still had to store a submatrix of size equal to the number of support vectors squared. Recall that every point which does not meet its margin requirements is a support vector. For large datasets, particularly if there is noise in the labels, this is still likely to be too large to store.

2.3.1 Osuna's Decomposition Approach

Osuna [83] introduced a key innovation for solving large-scale SVMs. He proved that one could use a *fixed size* working set and optimize with respect to just the working set variables, adding variables that violate the optimality conditions to the working set (and removing other variables) at the end of each iteration, and that the overall objective function would increase at each iteration. With this approach, SVM problems of arbitrary size could be solved using a fixed amount of memory. While his argument that the solution would asymptotically converge to the optimal solution was incorrect, and convergence of this algorithm to the optimal solution has not been formally established, the algorithm seems to converge on all real-world examples. Attempts to study the convergence of SVM algorithms have been made: Chang et al. [17] study general algorithms, but rely on the algorithm finding the examples which maximize the step size at each iteration. Keerthi and Gilbert [59] provide a general convergence proof for SMO-style algorithms (see below), in which the working set consists of only two examples at each iteration. Neither of these conditions are satisfied by Osuna's decomposition algorithm. Nevertheless, it seems extremely likely that the algorithm converges, assuming some minor regularity conditions such as the requirement that all training points are eventually checked for membership in the working set. This algorithm became known as the *decomposition* algorithm, and is also sometimes referred to as the chunking algorithm.[8]

Specifically, assume that we have partitioned our dataset into two subsets, a *working set* W and the remaining points R. Assuming without loss of generality that the working set of size k contains the points numbered $1, \ldots, k$, and that R consists of points $k+1, \ldots, \ell$, thereby inducing the partitioning $\alpha = [\alpha_W \ \alpha_R]$, we can rewrite the dual problems as

$$\max_{\alpha_W \in \mathbf{R}^{|W|},\ \alpha_R \in \mathbf{R}^{|R|}} \quad \sum_{\substack{i=1 \\ i \in W}}^{\ell} \alpha_i + \sum_{\substack{i=1 \\ i \in R}} \alpha_i - \tfrac{1}{2}[\alpha_{\mathbf{W}} \ \alpha_{\mathbf{R}}] \begin{bmatrix} Q_{WW} & Q_{WR} \\ Q_{RW} & Q_{RR} \end{bmatrix} \begin{bmatrix} \alpha_{\mathbf{W}} \\ \alpha_{\mathbf{R}} \end{bmatrix} \tag{2.77}$$

$$\text{subject to :} \quad \sum_{i \in W} y_i \alpha_i + \sum_{i \in R} y_i \alpha_i = 0 \tag{2.78}$$

$$0 \leq \alpha_i \leq C, \ \forall i \tag{2.79}$$

Suppose we have a feasible solution α to the dual problem. We can proceed by treating the $\alpha_{\mathbf{W}}$ as variable and the $\alpha_{\mathbf{R}}$ as constant. Doing so, and eliminating terms which are constant, and combining remaining terms leads us to the reduced dual problem

$$\max_{\alpha_W \in \mathbf{R}^{|W|}} \quad (\mathbf{1} - Q_{WR}\alpha_{\mathbf{R}})\alpha_W - \tfrac{1}{2}\alpha_{\mathbf{W}} Q_{WW} \alpha_{\mathbf{W}} \tag{2.80}$$

$$\text{subject to :} \quad \sum_{i \in W} y_i \alpha_i = -\sum_{i \in R} y_i \alpha_i \tag{2.81}$$

$$0 \leq \alpha_i \leq C, \ \forall i \in W \tag{2.82}$$

This subproblem is of a fixed size, and can be solved using an off-the-shelf QP solver. Osuna used the MINOS [80] solver [80] in his implementation, although other choices are possible — Joachims [57] used the free LOQO software [113] to solve essentially the same problem.

Osuna solved a large-scale SVM problem by solving a subsequence of problems of the form given by Equations 2.81-2.82. His method for choosing what points to

[8]The earlier method of portraits is also sometimes referred to as a chunking algorithm. The names have become somewhat confused over time.

put in the next subproblem was extremely simple. He simply cycled through the data sequentially, looking for points which violated the reduced optimality conditions (Equations 2.74-2.76) and adding them to the working set. He used a heuristic strategy to determine the number of points to add, which works with two parameters, `lookahead` and `newlimit`. The `lookahead` parameter defined the maximum number of points to check optimality conditions for in a "normal" iteration — if no violating points have been out after `lookahead` points have been examined, the system continued until either a single violating point had been found or all points had been checked. At each iteration, at most `newlimit` points could be added to the working set, so the algorithm stops examining points and solves a new subproblem once `newlimit` points have been found. Although Osuna provided experimental results in his thesis, the values of these parameters are not specified. One assumes that a set of heuristics are in place to deal with situations such as when there are fewer than `newlimit` non-violators available to remove from the working set, although again, these are not specified. Additional information relating to this issue is presented in Section 2.6.

Osuna defined a tolerance τ; a violation of an optimality condition by less than τ was not considered a violation. Although all SVM implementations proceed in this matter, different SVM implementations use slightly different parameterizations of the optimality conditions, which makes a comparison of training speeds across implementations somewhat difficult (see Section 2.5). In practice, these differences do not seem to be very important. Generally, solving the KKT conditions to a relatively modest accuracy such as $\tau = 10^{-4}$ is sufficient; many researchers have found that using much higher accuracies does not make a difference, although counterexamples exist.

It is worth noting that Osuna's implementation did not store the entire dataset in memory, but instead simply made as many passes as necessary through the file containing the dataset. This had the advantage of allowing him to solve problems where the individual data points were quite large,[9] and the entire dataset was too big

[9]One of the primary applications of the Osuna implementation has been (and continues to be)

to fit in memory. On the other hand, this approach has two major disadvantages. The first is the large amounts of time spent reading the dataset from disk. The second is that leaving the data on disk makes it extremely difficult to consider accessing the data points in a nonsequential order, which is a major feature of more recent approaches to solving large-scale SVMs.

It is also worth noting that many QP solvers, including MINOS, require the Q matrix be given explicitly. Therefore, each time Osuna's code solved a subproblem, it first computed and stored the entire Q matrix for that subproblem. Entries from previous subproblems were saved across entries to avoid redundant computation. An important additional speedup came from exploiting the fact that

$$\begin{aligned}
f(x_i) &= \sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) && (2.83)\\
&= \sum_{j=1, \alpha_j \neq 0}^{\ell} y_j \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) && (2.84)\\
&= \sum_{j=1, 0<\alpha_j<C}^{\ell} y_j \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) + \sum_{j=1, \alpha_j = C}^{\ell} y_j \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) && (2.85)\\
&= \sum_{j=1, 0<\alpha_j<C}^{\ell} y_j \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) + C \sum_{j=1, \alpha_j = C}^{\ell} y_j K(\mathbf{x_i}, \mathbf{x_j}). && (2.86)
\end{aligned}$$

For each data point, Osuna maintained a term corresponding to the second term in the Equation 2.86. When Osuna removed points from the working set with $\alpha_i = C$ (these points could be removed because they did not violate the *optimality conditions*, although they of course violated the *margin requirement*), he updated this term. Then, when he wanted to check whether a point not currently in the working set violated the optimality condition, he only had to compute the first term in Equation 2.86, corresponding to the unbounded support vectors.

Osuna's thesis and implementation included an extension to the case where two different values of C were used for the two classes. Such an approach is (in theory) useful when the number of examples from the two classes is very different, particu-

detection of faces and pedestrians in images. These images are sometimes as large 64 by 128 pixels, and there are often many thousands of them in the training set.

larly if these numbers are not proportional to the class membership percentages in the target population. Although this idea was relatively straightforward and widely known in the literature, it was nevertheless "rediscovered" and analyzed extensively by Lin, Lee and Wahba [69, 70]

2.3.2 Platt's Sequential Minimal Optimization

Platt [84, 85] extended Osuna's ideas with the additional intriguing observation that one could set the working set size in the decomposition algorithm to 2. The resulting two point QPs could be solved analytically, avoiding the need for a QP solver, which is generally either expensive or difficult to implement. This made it possible to solve large problems using an extremely simple code. This algorithm is known as Sequental Minimal Optimization (SMO), and is also often referred to as Platt's algorithm.

A complete derivation of the mathematics associated with Platt's paper is beyond the scope of thesis (see [85] for a complete mathematical derivation), but we will introduce some key equations that will be relevant to the SvmFu algorithm.

At a given step, suppose that we are interested in optimizing α_i and α_j while leaving all other α's constant. Using Equations 2.81 to 2.82, we can write a reduced two-element dual problem. This dual problem will contain the single equality constraint

$$y_i\alpha_i + y_j\alpha_j = -\sum_{\substack{i=1\\ i\notin\{j,k\}}}^{\ell} y_k\alpha_k \equiv \gamma \tag{2.87}$$

$$\implies \alpha_j^{\text{new}} = \frac{\gamma - y_i\alpha_i}{y_j}. \tag{2.88}$$

Using this relationship, we can eliminate α_j from the problem, leaving ourselves with a quadratic equation in α_i. We first find the unconstrained minimum of this problem, then "clip" the solution so that the resulting new values of α_i and α_j are feasible.

The unconstrained value of α_i^{new} is given by

$$\alpha_i^{\text{new}} = \alpha_i - \frac{y_i(E_j - E_i)}{2K_{ij} - K_{ii} - K_{jj}}, \tag{2.89}$$

where E_i is defined to be $f(\mathbf{x_i}) - y_i$, the "error" at point i.

Platt's algorithm uses a set of ad-hoc heuristics to choose what pair of examples to optimize at each step. There are separate heuristics for choosing the first and second element of the pair. The heuristic for the first pair alternates making single passes over the entire training set, then multiple passes over the unbounded support vectors, in order to concentrate effort on the examples most likely to need reoptimization. The second heuristic attempts to maximize the step size taken. SMO keeps a cached value of E_i for each USV i, and, if i is the element chosen by the first choice heuristic, will choose as the second example the j maximizing $|E_i - E_j|$. There also exist a collection of additional heuristics to handle rare cases, primarily relating to how to cope with the problem of multiple identical examples in the training set.

SMO uses an additional technique to greatly speed-up the training of linear SVMs. Noting that in the linear case,

$$\mathbf{w} = \sum_i y_i \alpha_i \mathbf{x_i}, \tag{2.90}$$

SMO maintains this $\mathbf{w}$ vector explicitly, and updates it using

$$\mathbf{w}^{\text{new}} = \mathbf{w} + y_i(\alpha_i^{\text{new}} - \alpha_i)\mathbf{x_i}. \tag{2.91}$$

When SMO needs to evaluate the optimality conditions at a new point which is not currently a USV, instead of having to compute a kernel product (dot product) with all the support vectors, it need only compute a single inner product, which represents a large savings.

Platt's algorithm generated a lot of excitement in the SVM community when it was initially introduced. One reason was that it represented a way to solve large-scale SVMs without any dependence on a third-party quadratic program solver, and seemed quite easy to code. Another reason was the impressive timing results Platt claimed compared to other methods. However, the computational experiments are somewhat suspect. Platt compared his algorithm to a "standard" chunking algorithm using "projected conjugate gradient" as the QP solver. Few details were available, and Platt was unable to release his code due to restrictions from Microsoft Research (personal communication). In retrospect, it appears that Platt's algorithm as written is not particularly fast, and that the timing results spring from a poor implementation of the alternate "chunking" method (see below). This is not surprising, given that as written, Platt's algorithm calls for recomputing each kernel product as it is needed. More modern, publicly released SVM implementations such as SVMLight [57] or SvmFu depend crucially on a cache system to cache frequently used kernel products. Indeed, in informal experiments, this author as well as other researchers have coded up the SMO algorithm as written and found it to be quite slow.

As written, SMO needs to examine the current values of $f(\mathbf{x_i})$ at each step. When an example is updated, if the example is a USV after updating, it imputes a value of b. If only one of the two examples is a USV, the imputed value of b is used. If both of the two examples are USVs, the average of the imputed b's is used. Keerthi et al. [61] show that because the algorithm takes a step by examining $E_i - E_j$, the b's will cancel, and therefore, knowledge of b is not needed to take a step. Somewhat strangely, instead of suggesting that we simply implement the algorithm in such a way that b is not estimated until the end of the optimization, when all the α's are known (the approach which SvmFu took, see below), they provide a new algorithm that maintains b^{up} and b^{down}, the largest and smallest imputed valid imputed values of b. When these two values become sufficiently close, the algorithm has converged.

2.3.3 Joachims' SVMLight

Thorsten Joachims' SVMLight system [55, 57] introduced several additional key innovations. His system is publicly available, and is quite popular.

Like Osuna's system, Joachims solved a large SVM problem by decomposing it into a series of smaller problems. Joachims' solved the resulting problems using a primal-dual interior point method based on LOQO [113], although SVMLight is now set up to use any of several different solvers.

Joachims introduced a much more principled way of selecting the working set than Osuna used.[10] The idea behind his method was that we want to select a feasible direction vector d that has both steep descent and at most q non-zero entries. Note that the gradient vector for the dual problem is

$$\mathbf{1} - Q\alpha \equiv g(\alpha). \tag{2.92}$$

The direction of steepest ascent is of course $\mathbf{1} - Q\alpha^T$, but this direction is in general neither sparse nor feasible. Instead, we examine the following maximization problem:

$$\begin{aligned}
\max_{\mathbf{d}\in\mathbf{R}^\ell} \quad & g(\alpha)^T\mathbf{d} && && (2.93)\\
& \mathbf{y}^T\mathbf{d} = 0 && && (2.94)\\
\text{subject to :} \quad & d_i \geq 0 && \text{for } i : \alpha_i = 0 && (2.95)\\
& d_i \leq 0 && \text{for } i : \alpha_i = C && (2.96)\\
& -\mathbf{1} \leq \mathbf{d} \leq \mathbf{1} && && (2.97)\\
& |\{d_i : d_i \neq 0\}| = q && && (2.98)
\end{aligned}$$

The first three constraints ensure that the direction is feasible. The fourth constraint normalizes the problem, making it well-posed, and the fifth and final constraint enforces sparsity. Since there aren't actually any interactions between the d_i, this problem is extremely simple to solve; in fact, it can be solved in $\ell \log \ell$ time. All elements in the optimal $\mathbf{d}$ will be 0, 1, or -1. Define $\omega_i \equiv y_i g(\alpha)_i$, and sort ω into increasing order. Successively choose pairs of elements from opposing classes off either the top or the bottom of this list, in such a way that the pair satisfies the constraints and adds as much as possible to the objective value. Continue until $\frac{q}{2}$ pairs have been added. Implemented properly, the runtime of this approach is dominated by the time required to sort the ω vector.

Because the LOQO software included with SVMLight is not especially robust, and requires storage of the entire chunk submatrix, SVMLight supports fairly small chunk

[10]In SMO, because we are only selecting two examples at each step, and each step is so minimal, these issues are entirely handled by the heuristics for example selection.

sizes — 2 to 400, with a default of 10, as opposed to the Osuna implementation or SvmFu, which often use chunk sizes of 2,000 or greater.

Another important innovation introduced by Joachims was *shrinking*. The essential idea is that if a point is not an unbounded support vector, has not been for a long time, and shows no prospects of becoming one, we may with high probability remove it from further consideration, and avoid examining it further to see if it violates the optimality conditions. Joachims controlled shrinking via two parameters.[11] The first parameter controlled how many times a point would be looked at before it was eligible for shrinking. The second parameter controlled whether or not points that were shrunk would be checked again at the end for optimality. Without this additional check, the Joachims implentation could no longer guarantee (approximate) optimality. On the other hand, the check at the end rarely yielded violating points, and cost substantial amounts of time. The user could control how aggressively shrinking was pursued, trading off her desire for an accurate and fast solution, although the control offered was not very direct.

The third and final major innovation in SVMLight was the introduction of a caching strategy. The cache size c (given here in number of entries for convenience) is specified by the user, and consists of $\lfloor \frac{c}{\ell} \rfloor$ columns. The kernel products of active variables with all other examples are stored in the columns. The cache uses a least-recently-used (LRU) replacement strategy — when the cache is full and a new element needs to be added, the element that was used least recently is jettisoned. When shrinking occurs, the effective size of the training set is reduced, and the cache is reorganized with more columns and fewer rows. This caching strategy greatly reduces the number of kernel computations required, especially if a relatively large cache is used.

[11]Examination of the available source code appears to indicate that there are additional non-user-controlled parameters inside the code, such as the maximum percentage of points to shrink during an iteration.

2.4 The SvmFu Algorithm

The SvmFu algorithm can be viewed as a synthesis of Osuna's, Platt's, and Joachims' approaches, combining the advantages of each with some additional features of our own. With SvmFu, we solve a large problem as a sequence of subproblems small enough that their associated Hessians can fit in memory. However, because we don't precompute the entire Hessian for these problems, the subproblems can be much larger than those used by SVMLight while still being efficiently solvable. The subproblems themselves are then solved by using a variant of Platt's algorithm.[12]

We first discuss our approach to solving subproblems. We next present how we choose which examples to place in the working set for each subproblem. Finally, we discuss our approach to implementation, which allows for more flexibility and efficiency than previously available packages.

2.4.1 Solving Subproblems

In this section, we consider solving a "medium-sized" SVM problem: the problem may have many examples, but we assume that we have enough memory to store the entire Hessian matrix. This algorithm can be used by itself to solve small- to medium-problems (approximately 7,000 points or less on today's machines), or can be used (with a very slight modification described below) to solve each of the sequence of subproblems generated in solving a larger problem. In the latter case, within this subsection, the term "entire dataset" refers only to all the data associated with the current subproblem.

As in Platt's algorithm, we will solve this problem by adjusting pairs of coefficients until approximate optimality is reached. At each step, we consider a "working set", consisting of either the entire dataset or of the unbounded support vectors (USVs) only. We maintain, for each example in the working set, the gradient of the objective

[12]Strictly speaking, the only thing this "variant" has in common with Platt's original algorithm is that two variables are optimized over at each step, and that these minimal optimizations are solved analytically. It has become common in the SVM literature to refer to any approach that operates in this manner as "Platt's SMO Algorithm."

function with respect to that example:

$$g_i = 1 - (Q\alpha)_i \tag{2.99}$$

Noting that $f(\mathbf{x_i}) = g_i + b$, we see that knowing g_i and g_j allows us to trivially compute the E_i and E_j necessary to optimize α_i and α_j. The analytic technique used for optimizing a pair of examples is precisely the method introduced by Platt.

We then choose the pair of examples that *maximizes* the local first derivative of the objective function, and optimize with respect to that pair. This can easily be done in time linear in the number of examples in the working set. We simply find, for each class, the example with the largest gradient that is not currently at its upper bound, and the example with the smallest (i.e., most negative) gradient that is not currently at its lower bound. We then optimize over one of the pair of positive examples, the pair of negative examples, the positive example with the largest gradient and the negative example with the largest gradient, or the positive example with the smallest gradient and the negative example with the smallest gradient; the pair which has the largest combined gradient in the direction of movement is chosen. This is extremely similar to the method Joachims uses to choose the variables in his "working set", specialized to a working set of size 2.

The optimization is performed analytically, as in Platt's algorithm. The gradients can all be updated in linear time. Therefore, the total amount of time required to take a step (ignoring time to compute kernel computations) is linear in the number of examples in the working set.

We allocate enough memory to store the entire kernel matrix, but we do not precompute it. Instead, when α_i, the coefficient associated with the i'th example, first becomes nonzero, we compute the entire ith row of the kernel matrix. Entries which have already been computed as parts of other rows are copied, not recomputed. This scheme has the property that no kernel element is computed more than once, and kernel products between pairs of elements neither of whose coefficients are ever nonzero are never computed. Additionally, because we have enough memory to store the entire matrix if necessary, we can use a simple array data structure, and lookups are very fast.

We experimented with using a variant that exploited the symmetry of the kernel matrix. Although in principle, this could save as much as half the memory, in practice, the amount of memory used was close to equal, as it still need to compute and store $\ell - i$ entries when computing the $i + 1$'st row of a symmetric matrix (as opposed to computing $\ell - i$ and storing ℓ in the symmetric case. Unless the majority of the Hessian has been computed, the memory savings from exploiting the symmetry are not substantial. However, if we consider the worst case, when the entire Hessian is needed (all the points have nonzero coefficients at some point in the computation), we save half the memory. More important than the memory considerations, the symmetric version ended up being slower, because whenever the algorithm updates an α_i, it needs to access every entry of row i of the kernel matrix in order to update the gradient vector. The time to reconstruct the row or traverse the symmetric matrix was substantially longer than the time required to traverse the consecutive memory locations of a row in the full matrix, due to hardware cache issues. Nevertheless, this should still be considered a possible strategy if it is desirable to choose a subproblem size where the full matrix would be too large to store, but the symmetric matrix could fit in memory.

Initially, the working set is the entire dataset. We continue taking steps over the entire dataset until we have taken some prescribed number of steps without going outside the set of USVs (currently, we use a value of 100), at which point the working set becomes the set of current USVs, referred to as the *reduced working set.* The reduced working set may not consist entirely of USVs, as some of the coefficients may be adjusted to be at bounds while we are working with the reduced working set. If we cannot take a step and the working set is the USVs, we reset the working set to the full dataset; if the working set is already the full dataset, we are done. Because the time required to update a single pair of coefficients is linear in the size of the working set, and the number of USVs is usually a small fraction of the total size of the dataset, this reduced working set technique greatly speeds up the algorithm.

There is no computational cost associated with switching from the full dataset to the reduced working set. To switch back from the reduced working set to the full dataset requires operations proportional to the size of the reduced working set multiplied by the size of the remainder of the dataset — each example not in the reduced working set must have its gradient updated, and in general, the α's for all points in the reduced working set will have shifted.

Our technique for choosing examples to optimize over is quite different from Platt's technique. In the vast majority of cases, Platt's algorithm chooses the first example by either cycling through the working set or the entire dataset, and chooses the second example as the USV that optimizes the absolute value of the two-example gradient $|E_i - E_j|$, which involves searching through the set of USVs and performing a small constant number of operations on each one. Once the example is chosen and the new α_i and α_j are determined, Platt updates the error value (equivalently, the gradient) for all the USVs. If the first example is a USV, it's E_i is already known, so the total number of steps is the number of USVs. If the first example is not a USV, E_i needs to be computed from scratch, which takes time equal to the total number of support vectors. Therefore, the total time required to perform an update is either the total number of support vectors or (more frequently) the number of USVs.

In our algorithm, we require the E_i for every point in the working set, and we update all these values. Therefore, the time required per iteration is either the size of the reduced working set or the size of the entire dataset. The majority of the steps are taken using the reduced working set. Therefore, we take essentially the same amount of time per optimization step as Platt's algorithm. However, because each step maximizes the absolute value of the two-example gradient (either in the reduced working set or the full dataset), we expect to take many fewer steps than Platt's SMO algorithm. Informal experiments agree with this assessment.

In summary, we have created and implemented an "SMO variant" that is substantially different from Platt's original Sequential Minimization Optimization algorithm. We have replaced Platt's heuristics with a more principled choice, allowing many fewer steps at essentially the same cost per iteration, and we have added an intelli-

```
Let working set **W** be the entire dataset
Let **done** == false
Let **fullWSteps** = 0
**while done** = false **do**
  Let $i, j$ be the arguments maximizing $|g_i - g_j|$, subject to the constraint that $i$
  and $j$ can move in the directions indicated by their gradients
  Let $E = |g_i - g_j|$
  **if** $E >$ tolerance **then**
    Take step on examples $i$ and $j$, update $\alpha$'s and gradients.
    **if** $W$ == entire dataset **then**
      **fullWSteps = fullWSteps + 1**
      **if fullWSteps == 100 then**
        Set **W** = reduced working set of current USVs only.
      **end if**
    **end if**
  **else if W** = reduced working set **then**
    **W** = entire dataset
  **else**
    **done** = true
  **end if**
**end while**
```

Figure 2-5: SvmFu's algorithm for solving subproblems.

gent system for computing only the portion of the Hessian matrix that is needed while avoiding redundant kernel computations. An overview of the algorithm for solving subproblems is given in pseudocode in Figure 2-5.

2.4.2 Choosing the Next Subproblem

We now consider the problem of modifying the working set to generate a new subproblem. We first determine the maximum number of points to be removed from the old working set. This number is the size of the chunk, minus the number of USVs, minus some additional user-definable constant (currently, we use 200). If this number is less than some minimum value (100 in our work), we set it to the minimum value. This balances several competing interests. We do not want to remove USVs from the working set, as their values are extremely likely to change (see Section 2.6). We want to ensure that a reasonable number of new points enter the working set at each iteration. Additionally, we want to maintain some additional non-USV points which are "almost" USVs in the working set.

When a subproblem is solved, all points in the working set satisfy the optimality conditions (Equations 2.74-2.76). For each point in the working set, we determine the amount by which they satisfy the optimality conditions, and remove points which satisfy the conditions with the largest margin first. Note that points which are USVs satisfy the conditions exactly, and will not usually be removed. Additionally, the points which are "almost" USVs, and therefore are most likely to cross the margin line in future iterations, stay in the working set.

We maintain a priority queue of all points not in the current working set, sorted by the amount by which that point violated or failed to violate the KKT conditions the last time it was looked at. (Initially, $w = 0$ and $b = 0$, so all points not in the working set are placed in the queue with a priority of -1.) When we need to select new points for the working set, we begin pulling points off the queue, examining them, and moving them into the working set if they violate the KKT conditions, and otherwise holding them to be placed back on the queue once the new working set is established. We continue this process until we have examined all the points, we have found as many violating points as we are willing to add to the working set, or, if the first violating point we have found was the k'th point we examined, we have examined at least $k + 200$ points without finding a violating example.

This procedure has the effect that points which satisfy the KKT conditions by a large margin (i.e., are far from the separating hyperplane), will have low priority, and will be looked at only infrequently. This is an implicit, soft form of the "shrinking" idea forwarded by Joachims. We also incorporate an additional, explicit "shrinking": if a point is looked at a certain number of times in a row and never violates the KKT conditions, that point is removed from further consideration and not looked at again. Using explicit shrinking, we no longer have an optimality guarantee on our final solution.[13] However, we expect explicit shrinking, if not applied too aggressively,

[13]Strictly speaking, because the SVM is always solved only to some numerical tolerance, we never have a true optimality *guarantee* on the final solution. Nevertheless, although we conjecture that

to have little effect on the optimal solution. This problem is faced by other codes which use shrinking as well [55, 19].

Additionally, we take a sophisticated approach to caching in an attempt to avoid redundant kernel computations. There are essentially two kinds of cache in SvmFu (although they are both part of the same C++ object) — a square chunk cache which stores all the computed kernel entries in the current chunk, and an additional rectangular chunk cache which stores kernel products between members of the working set and additional points. The idea of the extra cache is that it will save time on recomputing the gradient for points which are not in the working set but are always relatively close to violating the optimality conditions. The kernel products of these points with the working set points will be stored in the extra cache, making the relatively frequent checks of these points much faster than they would be in the absence of the extra cache.

We can use exactly the same code described in Subsction 2.4.1 to solve the subproblems, with one slight modification. Looking at the form of the reduced problem (Equations 2.81-2.82), we see that the gradient of the objective function is no longer defined by

$$\mathbf{g} = \mathbf{1} - (Q\alpha), \tag{2.100}$$

but instead by

$$g_i = \mathbf{1} - Q_{WR}\alpha_{\mathbf{R}} - Q_{WW}\alpha_{\mathbf{W}}. \tag{2.101}$$

Essentially, this change is allowing for a modified objective function taking into account the (linear) interactions between the variable $\alpha_{\mathbf{W}}$ and the temporarily constant (over the life of a subproblem) $\alpha_{\mathbf{R}}$. This slight change is easily affected by allowing the outer loop of SvmFu to modify the gradient vector before solving a subproblem.

We believe that this "two-tiered" approach of solving a sequence of larger surproblems, each of which is itself solved via a two-at-a-time optimization, is a strong approach with several advantages. The two-at-a-time models have a key advantage

the basic algorithm, if applied with infinite precision, would converge to the true optimum, it is easy to construct examples where this does not occur if explicit shrinking is used.

over more standard quadratic programming codes which do not exploit the structure of the SVM: they do not require explicit precomputation of the kernel matrix for the entire chunk, and indeed, large parts of this matrix are often never computed. At the same time, solving the entire problem at once using our two-at-a-time approach is infeasible because the cost to take a single step is proportional to the size of the chunk, and we could no longer assume that we could store the entire chunk kernel matrix if necessary, a crucial assumption in making the code simple and efficient.

2.4.3 Flexibility and Efficiency

SvmFu includes a number of engineering choices designed to make it easy to solve large problems quickly.[14] In particular, SvmFu was written as a parametrized C++ library with templates, with as few assumptions as possible. By instantiating the library in different ways, it is possible to take advantage of the particular form of your data. Specifically, the user can easily choose the data type of both the input data and the kernel values without recompiling. For instance, if the data consists of integers between 0 and 256, you can represent each data point as an array of `char`'s. Kernel values can be computed in single or double precision, letting the user take advantage of tradeoffs between speed and accuracy. The kernel functions are part of the client, not the library, so it's easy for the user to include new kernel functions and data types.

A more important advantage is that the structure of a "data point" is also a client side decision: the SVM library itself takes a pointer to an array of data points, and pointers to a kernel function which can operate on these data points, without making assumptions about the structure of the data. This makes it extremely easy to allow the SVM to operate on either dense or sparse data: the user may store the data in whatever format she likes, so long as she provides a pointer to an appropriate kernel function. It would even be easy for the user to define an SVM that operated on data where the "data points", were not stored in array form, but instead were stored in

[14]In this section, we assume some familiarity with C and C++ programming.

an object of arbitrary type.[15]

Additionally, because of SvmFu's client and library based interface, SvmFu is an extremely flexible system that can be used in a wide variety of applications. A client that needs to access an SvmFu, can simply link to the SvmFu libraries, and then train its own svms. Writing various multiclass classification systems (see Chapter 4) was done using exactly this technique.

There are some disadvantages to using parameterized types, and putting so much of the power on the client side. Chief among these is that the types of the data must be chosen at compile time (the current client instantiates the SVM library a number of times corresponding to various commonly used data types, but the addition of new types requires a recompile). The use of templated functions presented an extreme problem in ease-of-programming and portability. Templates are not currently handled uniformly by all C++ compilers. SvmFu was developed using the popular, open-source Gnu compiler `g++`. However, the associated `gdb` debugger does not fully support templates, making it essentially impossible to use advanced debugging tools on SvmFu and requiring the antiquated technique of debugging via "print" statements. Even worse, it will take a major amount of effort to get SvmFu to compile under Windows Visual C++, for which the author receives a steady stream of requests. However, although some education and care is required in the use of SvmFu, and some of the engineering choices made the code more difficult to implement and less portable, we believe that the advantages of our approach far outweigh the problems.

2.5 Experiments

In this section, we discuss and examine the performance of SvmFu. We seek to avoid going too far in this direction, for the following reasons. When we first began working on SvmFu, the alternative was to solve SVMs using Osuna's implementation

[15]Easy here should not be taken to mean "trivial". The user would still have to write her own client program, or modify the `svmfutrain` client included with SvmFu.

— neither Platt's SMO nor Joachims' SvmLight were known. There was a real need for an SVM solver that was substantially faster than what was then available. In the interim, a wide variety of high-powered SVM solvers capable of solving SVM problems have appeared. We believe that SvmFu is likely to be as fast or somewhat faster than these solvers across a wide variety of problems. However, it is unlikely to be so much faster as to be an *enabling technology* — it is not so much faster that there is a wide class of problems that can be solved in a "reasonable" amount of time by SvmFu that cannot be solved in a reasonable amount of time by any other method.

The task of benchmarking SVM software is difficult and thankless. The difficulty lies in the fact that each implementation has its own parameters which must be explored. Even assuming we fix the kernel and the value of C, we are still left with the problem of ensuring the two systems to be compared have access to the same total amount of memory, and choosing the chunk sizes for each of the two systems, noting that the optimal choice of chunk size may not be the same. Additionally, the systems may not represent the data in the same way, which can cause a substantial constant-time difference in the performance of different systems. For example, SvmLight stores all data in a "sparse" format, where for each example, a list of integer values of the non-zero dimensions is stored, along with the floating-point value of these dimensions. SvmFu, on the other hand, has the capability to store data in either a dense or sparse format, and the values themselves can be integer, single- or double-precision floating point, or even user-defined. In this context, it is very difficult to know what constitutes a "fair" comparison. The situation only becomes more complicated when we try to deal with options like explicit shrinking which no longer guarantee the optimal solution. Keerthi et al. [61] have reasonably suggested that the number of kernel evaluations be used as a measure of performance, because the majority of the time in SVM training in spent evaluating kernel products. However, the majority does not mean "almost all" — for SvmFu, we often found that about $\frac{2}{3}$ of the total time was spent evaluating kernel products. Since the differences in time performance between different systems are often smaller than $\frac{1}{3}$, the number of kernel evaluations is not necessarily a reliable predictor of which system will solve a given training problem faster.

The most important reason to avoid spending too much effort study the performance of different SVM systems very precisely is the fact that all the modern SVM systems perform fairly closely to "optimally". To see this, we consider the following question: if we had an SVM training problem, and an oracle gave us the optimal α vector at no computational cost, how much would it cost us to *verify* that the purported solution was correct? It seems reasonable that we would have to check the optimality conditions (Equations 2.74-2.76). How long would this take? We would have to compute the kernel product of all the support vectors with all the data points. Intuitively, this is a useful *lower bound* on how much computation is required to train an SVM. As pointed out by Joachims [57], for large but solvable problem sizes, SVM-Light is already within a small constant factor of this amount of computation. We conclude that the scope for improvement is relatively limited — we may decrease the time required to find a solution by a factor of 2 or even in some cases a factor of 5, but probably not a factor of 10 and certainly not a factor of 100. This is contrast to what we see in Chapter 3, where we find that a different formulation leads to a much simpler problem that can be solved much faster.

We benchmark SvmFu against the popular, state-of-the-art solver SVMLight. We benchmark the two systems on two different tasks. The first problem is a database of 31,022 faces and non-faces. Each example is a 361 dimensional dense real-valued vector. The second task is the so-called "adult" database, consisting of 32,561 examples. Each example is a 122 dimensional sparse binary-valued vector (containing no more than 13 1's), and a target vector indicating whether the example's yearly income is more or less than \$50,000.

The systems were benchmarked on a computer with a Pentium III running at 500 MHz with 256 Megabytes of RAM running Linux. There were no other heavy jobs in the system, so the results are fairly accurate. However, the machines were not put in single process mode, so the results should only be viewed as accurate to within a few percentage points. In all cases, we allow both systems to use 120 Megabytes of memory, we set the SvmFu chunk size to 3,000 data points, we set the

SVMLight chunk size to 400 data points (the default). We run SVMLight with the default settings for explicit shrinking, and try it both with and without a final check which guarantees the optimal solutions. The results are shown over a range of nested datasets of increasing size in Figure 2-6.

As we see, SvmFu is as fast or faster than SVMLight on both tasks. For the faces, it is substantially faster, about a factor of 4 faster when using the entire data set. A large part of this difference can be attributed to SVMLight's inefficient sparse representation for this dense dataset. In the case of the adult dataset, the performance of the two systems is essentially identical. We conclude that SvmFu exhibits state-of-the-art performance, performing as well or better than a well-respected system on two different tasks. Nevertheless, these results should be taken with a grain of salt — the tests were not exhaustive, it was difficult to control for all the variables. Most important is still the observation that there are many systems capable of solving large problems, and that SvmFu's primary advantage lies not in speed, but in its library structure, flexibility and adaptability to a variety of related tasks.

In early versions of SvmFu, there was only a single cache, corresponding to the square matrix needed to store the chunk for a subproblem. This chunk matrix was preserved and modified from subproblem to subproblem, leading to effective reuse of kernel computations. However, as the subproblems get bigger, each subproblem takes longer to solve (remember that the time per step within a subproblem is often proportional to the subproblem size), and as the chunk size grows large, beyond a certain point the total training time required increases. Unfortunately, this led to the "problem" that allowing SvmFu additional memory could actually cause the total time required to solve the problem to increase, because this additionally memory was directly linked to a larger chunk size, which slowed the whole system down. Other state of the art systems do not have this difficulty. This was the rationale behind the additional caching system. As is shown in Figure 2-7, for any given chunk size, the performance of the system generally improves (there is one unexplained exception for a chunk size of 3,000) as the total amount of memory used increases.

Although this two-tiered caching system is largely successful, it is complicated

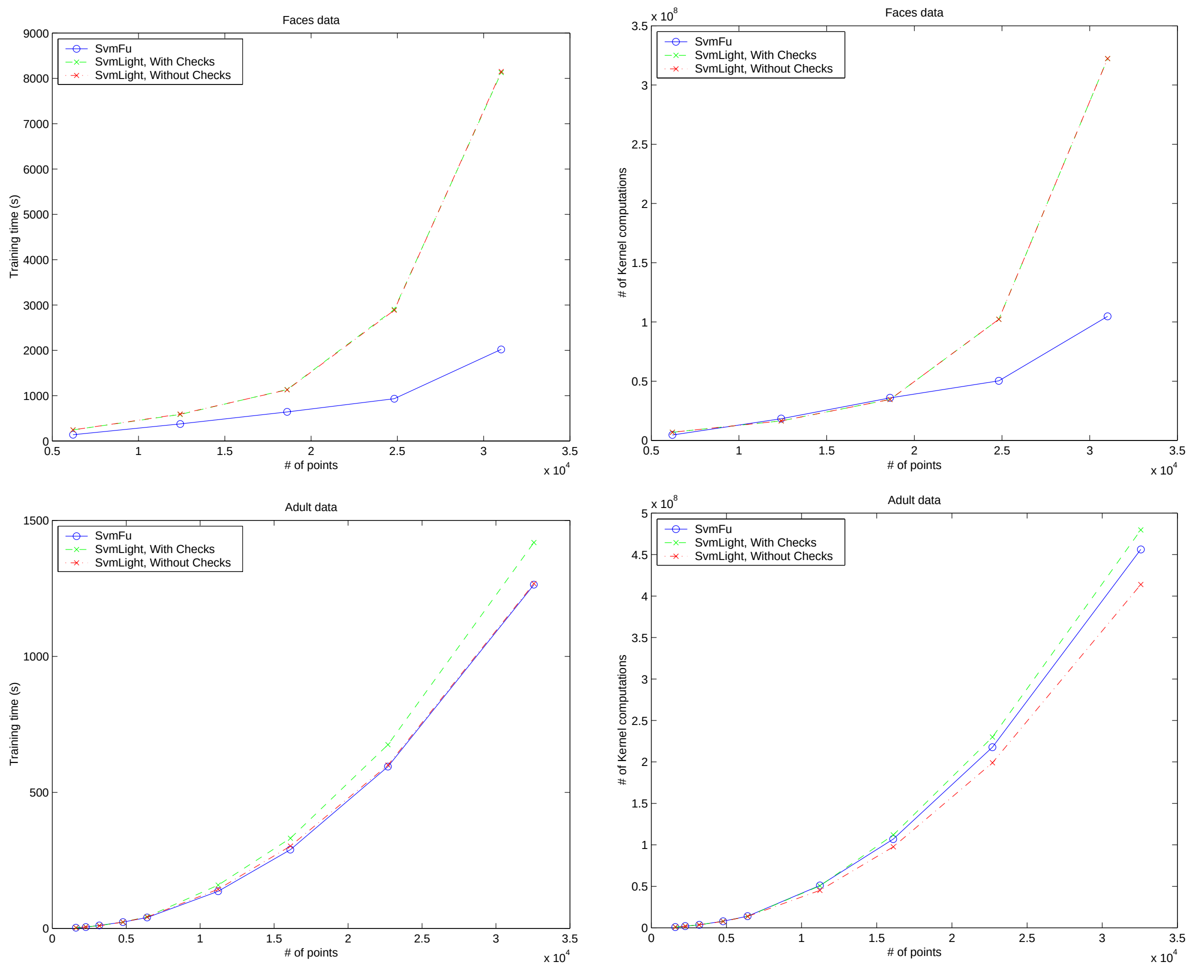


Figure 2-6: Experiments comparing SvmFu and SVMLight.

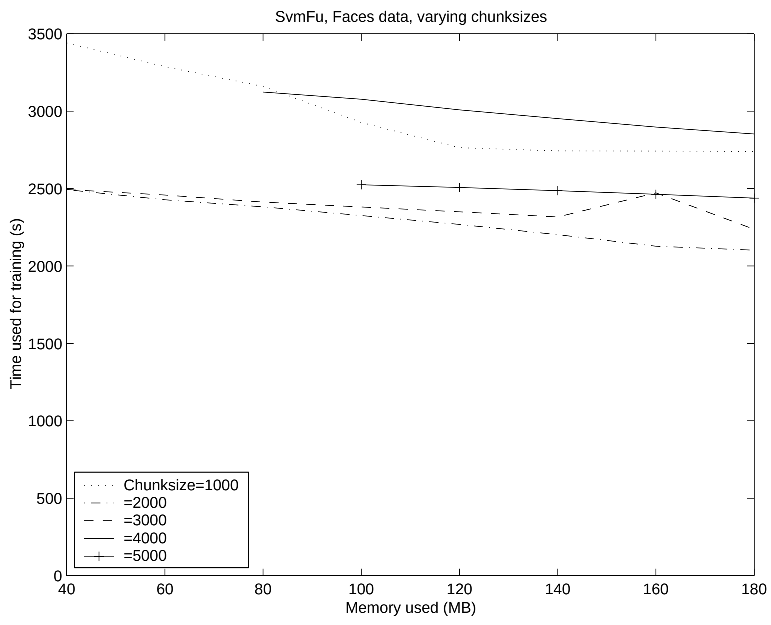


Figure 2-7: The time required to solve the SVM for varying chunk sizes and total cache sizes.

and difficult to modify and work with. If we were reimplementing the system from scratch, with the urge to squeeze every last ounce of speed out of SVM training, we would probably implement a caching system more similar to SVMLight's, incorporating explicit shrinking, with an automatic mechanism to determine the cache size. However, SvmFu is an adequate tool for solving large scale SVMs, and the additional advantages of such an approach are small. Combined with the fact that other formulations present more interest (see Chapter 3), making another version of SvmFu does not seem worth the effort.

2.6 Folk Wisdom

2.6.1 The Kernel, C, and Scaling

People often wonder what kernel to use with an SVM. Unfortunately, there is no easy answer to this question. The best approach is to try a variety of different kernels, and validate their performance on an independent cross-validation set. Many users try a number of different kernels, but only report results on the kernel that performed best on their test set. This is not justifiable scientifically, as choosing the kernel parameter that maximizes performance on a test set is a form of "overfitting" — we lose confidence that performance on the test set is a good predictor of future performance when we try a large number of kernels (technically, more than one) on the same training and test set pair.

However, there are some rules of thumb. One good rule is to start with a linear kernel, since it is the simplest kernel and often gives good performance. Another key is to remember that performance on the training and test set will be different, and to be sure to remember to validate your model, rather than just choosing a kernel with the lowest training error (which can always be driven to zero in any case).

Empirically, we have found in several examples, notably face and pedestrian dectection, that a Gaussian kernel with appropriate settings of the width parameter σ will usually produce results that are as good or better than any other kernel. However, the Gaussian kernel will generally have more support vectors than the polynomial kernel, and will therefore be much slower to use at testing time. It is also worth paying attention to how the σ term in the Gaussian relates to the number of support vectors. Recall that the Gaussian kernel is defined by

$$K(\mathbf{x_1}, \mathbf{x_2}) = \exp\left(-\frac{||\mathbf{x_1} - \mathbf{x_2}||^2}{2\sigma^2}\right) \tag{2.102}$$

For any σ, $K(\mathbf{x}, \mathbf{x})$, the kernel product of a point with itself, is always one. Now, consider the effect of sweeping over all possible values of σ. As $\sigma \to \infty$, for all pairs of points i and j,

$$K(\mathbf{x_1}, \mathbf{x_2}) = exp\left(-\frac{||\mathbf{x_1} - \mathbf{x_2}||^2}{2\sigma^2}\right) \to 1. \tag{2.103}$$

In other words, as $\sigma \to \infty$, all the points will begin to "look alike" and we will be unable to build a usable classification function. In terms of SvmFu, this problem will manifest in terms of a very poor *training* error, and in extreme cases, an inability to find an unbounded support vector to determine b (see below).

On the other hand, as $\sigma \to 0$, $K(\mathbf{x_i}, \mathbf{x_j})$ will go to 0 for $i \neq j$, while remaining 1 for $i = j$. Conceptually, when we set α_i to a nonzero value, we are placing a (positive or negative, depending on y_i) spherical Gaussian centered at $\mathbf{x_i}$, with standard deviation σ and maximum height proportional to α_i.[16] If σ is close to zero ("close" here is relative to the scaling of the data), then we have a situation where the Gaussian around any point becomes very very close to zero before reaching any other point, and therefore cannot contribute to the correct classification of any other point. In SvmFu, this will manifest itself with essentially every point being a support vector, and if C is sufficiently large, an unbounded support vector. Additionally, the training time will become extremely large. Geometrically, we can only adequately classify the training set by placing a very tightly centered Gaussian around every single point, an obviously unsatisfactory situation that will fail to generalize.

These difficulties can be observed and corrected without recourse to a test set — if σ is much too big, the training performance will be terrible, and if σ is much too small, almost every training point will be a support vector. Finding the *best* value of sigma requires cross-validation, but a reasonable range of σ's to test can be found via careful application of the above advice. Fortunately, in many problems, the best value of σ is reasonably robust over a fairly wide range.

Additionally, how to set C presents something of a challenge. Geometrically, C controls the tradeoff between how hard we try to have a large margin and how hard we try to avoid training error. As C gets sufficiently large, we will converge to the solution with minimium training error (measured using the hinge loss). Further increases in C beyond this point will have no effect. This situation will usually result in a very small margin, as well as an extremely large training time. This point cannot be overemphasized — *training SVMs with an excessively large value of C can increase the required training time by several orders of magnitude.* On the other hand, if C is too small, this will manifest itself in a solution with no unbounded support vectors, and therefore no ability to determine the b term [91]. Fortunately, we again find that the solution is often not especially sensitive to the choice of the C parameter within a fairly wide range. In particular, especially for the Gaussian kernel, we find that

[16]It is not a completely standard Gaussian, as the Gaussian kernel is generally used in an *unnormalized* fashion, so it does not integrate to 1 over the entire space.

$C = 1$ is often a good choice: by setting $\alpha_i = 1$, any given training point can be classified correctly using only its own contribution to its classification.

Although this is not widely known or practiced in the SVM literature, stability theory and the regularization view of SVMs (see Chapter 5) indicate that we should scale C as $\frac{1}{\ell}$. This has the effect of maintaining a given weighting between the norm of the function (the $||w||^2$ term), and the *total* misclassification error on the training set. However, the issue is actually more complicated than this [28, 102], and in practice, one is often presented with only a single training set, so the issue of how to scale C with ℓ is not important.

Finally, a word about scaling is in order. Kernel products generally rely on either the dot product $\mathbf{x_i} \cdot \mathbf{x_j}$ of two data points (linear and polynomial kernels) or the difference $\mathbf{x_i} - \mathbf{x_j}$ of two data points (the Gaussian and related radial kernels). In the former case, dimensions whose absolute values are larger will contribute disproportionately to the kernel product. In the latter case, dimensions whose variance is higher will contribute disproportionately to the kernel product. There is nothing inherently wrong with this, and indeed, if information is known about the relative importance of the variables this should definitely be used. However, this situation is often introduced unknowingly — different variables are measured in different units, and without knowing anything particular about the problem domain, the variables have very different means and variances. For this reason, when no information is known about the relative importance of the variables, we generally scale all the variables to have mean 0 and variance 1.

Some additional care must be taken with linear or especially polynomial kernels, which can potentially generate very large values (the Gaussian kernel's values are bounded between 0 and 1, so they present less of a problem in this respect). For technical reasons relating to the fact that our code operates in finite precision floating point arithmetic and that we compute the desired α_i^{new} using

$$\alpha_i^{\text{new}} = \alpha_i - \frac{y_i(E_j - E_i)}{2K_{ij} - K_{ii} - K_{jj}}, \tag{2.104}$$

we want to avoid situations where the kernel values become extremely large. If there are a large number (many thousands) of variables, in addition to setting each variable to have mean 0 and variance 1, it is often useful to divide all values by a constant such as the number of variables.

2.6.2 b

In the standard SVM formulation which we focus on in this chapter, there is a bias term b which is entirely unregularized. If we study the way that the dual problem is derived from the primal, we see that the equality constraint

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0 \tag{2.105}$$

results directly from this unregularized b term.

If we require that $b = 0$ this constraint disappears from the dual problem. Alternatively, in the linear case (without using kernels), we can add a term of the form $\frac{1}{2}b^2$ to the objective function. This is equivalent to adding an extra dimension to the data which contains a constant term 1 for every example. In this case, the constraint will also disappear — in deriving the dual, we can obtain $b = \sum_{i=1}^{\ell} y_i \alpha_i$, and substitute this term into the Lagrangian.

The absence of the b term removes the equality constraint from the dual, making the problem somewhat easier to solve. In particular, we could then allow algorithms that updated a single α_i at a time, rather than updating pairs of α's simultaneously in order to satisfy the equality constraint. However, the main difficulty in SVMs arises not from the equality constraint (which after all represents merely a projection in one dimension), but in the box constraints, so for general, nonlinear SVMs, we do not expect the absence of the b term to make a huge difference. In the linear case the situation is different — however, in Chapter 3 we will show that in the linear case, there is another formulation which is generally superior to standard SVMs.

In some sense, the use of the b term in SVMs is a historical accident made at the time of their introduction [115, 21, 20]. In classical regression, it was considered desirable to have enough representational power to closely represent any function in L_2 (i.e., we want our RKHS $\mathcal{H}$ to be dense in L_2. Strictly positive definite kernels such as the Gaussian allowed this. However, certain kernels based on splines were slightly degenerate, and required the addition of one or more constant or polynomial terms in order to be dense in L_2. In [87], we provide an extensive discussion of this point. The discussion has the following practical implications of this term for SVMs. In the case of a strictly positive definite kernel such as the Gaussian, the b term is unnecessary for density of $\mathcal{H}$ in L_2. For the finite dimensional feature spaces induced by linear or polynomial kernels, the $\mathcal{H}$ will not be dense in L_2 either with or without the b term. The question seems to therefore come down to whether one feels that zero is a "priviliged" value of the bias term b, and that deviations from zero should be paid for or even disallowed. From this standpoint, it seems that the inclusion or omission of b is a facet of the choice of kernel. However, as we show in Chapter 5, the inclusion of even a single unregularized bias term has substantial implications for stability based approaches to generalization bounds, lending credence to the idea that the unregularized b term is actually a poor choice from a theory perspective. In practice, it doesn't seem to make very much difference whether a b term is used or not, although one can certainly construct (pathological) examples where it is either helpful or harmful.

2.6.3 Chunk Sizes and Memory

There are two parameters that must be chosen to control SvmFu's usage, the size of the problems and the total amount of memory used. In general, it will be advatangeous to use as much as memory as possible (see Section 2.5). The choice of the chunk size is somewhat more delicate. An overly large chunk size will result in a somewhat slower system overall. However, the reverse problem is much more of an issue. If the chunk size is too small, then the system will be forced to remove unbounded support vectors from the system. This is almost certain to lead to disaster. Unbounded support vectors need to satisfy the equation:

$$y_i(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x_i}, \mathbf{x_j})) = 1. \tag{2.106}$$

Therefore, if an unbounded support vector is removed from the working set, it is almost certainly going to violate its optimality condition when it is reexamined after the subproblem is solved. It will then have to be readded to the chunk, necessitating the removal of additional unbounded support vectors. This will lead to churning of the working set, resulting in an explosion of the number of subproblems necessary to solve the overall problem.

For this reason, it is important to choose the chunk size to be substantially bigger than the number of unbounded support vectors. Unfortunately, the number of USVs cannot be known a priori. Therefore, we advise starting with a relatively large chunk size (say, 2,000), and increasing it as necessary if it seems too small.

This leads directly to a crucial observation about SVMs. Given sufficient memory, any of the modern large-scale SVM solvers will be able to solve the problem. Their solution times may differ by a small constant factor (even as much as a factor of 5), but it is not the case that one implementation makes problems solvable which are in general unsolvable by other implementations. On the other hand, if a given training problem has enough unbounded support vectors that we cannot afford to store the kernel product between all pairs of USVs, we are unlikely to be able to solve the problem, using SvmFu or any other method. For this reason, we argue that the number of unbounded support vectors is the primary indicator of how "difficult" an SVM training problem is.

Chapter 3

Regularized Least-Squares Classification

3.1 Introduction

A general approach to learning is Tikhonov regularization, where we try to find a function that simultaneously has small empirical error and small norm in a Reproducing Kernel Hilbert Space K. Specifically, the Tikhonov regularization problem [111, 119, 31] is stated as

$$\min_{f \in \mathcal{H}} \frac{1}{\ell} \sum_{i=1}^{\ell} V(y_i, f(\mathbf{x}_i)) + \lambda ||f||_K^2. \tag{3.1}$$

By varying our choice of V and K, we can arrive at a wide variety of popular learning schemes [31]. In Chapter 2, we considered the loss function $V(f(\mathbf{x}, y)) = (1 - yf(\mathbf{x}))_+$, and showed how this loss function led directly to the well-known Support Vector Machine algoritm. In this chapter, instead of the hinge loss, we consider the square loss $V(f(\mathbf{x}, y)) = (y - f(\mathbf{x}))^2$ (Figure 3-1).

We call the resulting algorithm Regularized Least-Squares Classification (RLSC

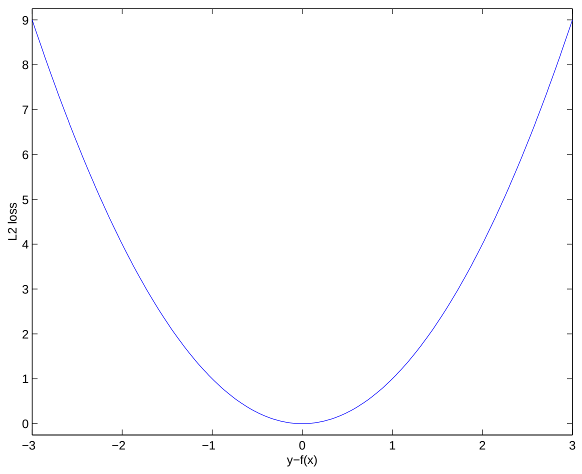


Figure 3-1: The square loss $V(f(x), y) = (y - f(x))^2$.

for short):[1]

$$\min_{f\in\mathcal{H}} \frac{1}{\ell}\sum_{i=1}^{\ell}(y_i - f(\mathbf{x}_i))^2 + \lambda||f||_K^2. \tag{3.2}$$

We will see that a Regularized Least-Squares Classification problem can be solved by solving a single system of linear equations. This is in direct contrast to the Support Vector Machine algorithm, which requires the solution of an inequality constrained convex optimization problem. At first glance, it seems that we should be able to solve RLSC much more rapidly. However, the situation is more complicated than this, because RLSC does not permit the decomposition approach that made large-scale SVMs tractable. Nevertheless, we will see that RLSC represents a highly viable alternative to SVMs, especially for linear problems.

The contributions of this chapter are as follows. In Section 3.2, we derive the essential mathematics of the Regularized Least-Squares Classification Algorithm. In Section 3.3, we derive a bound on the leave-one-out error of RLSC. In Section 3.4, we discuss the intuition behind the algorithm, and demonstrate on simple toy examples that its performance is essentially equivalent to that of the SVM. In Section 3.5, we discuss previous work on this problem, placing this work in the proper historical con-

[1]The author is not the inventor of this algorithm. This algorithm is a straightforward application of the work of Tikhonov and Arsenin [111], although it has recently been rediscovered by a number of authors (see Section 3.5 for more details). The name Regularized Least-Squares Classification is the author's.

text and comparing it to our own approach. In Section 3.6, we begin our study of the efficiency of RLSC by considering a specialization to linear kernels (functions of the form $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$), and showing that in this case, the algorithm can be implemented extremely efficiently. We also present a large-scale text classification example, and show that on a large dataset, RLSC gives a huge speedup in training time while offering identical quality of the learned functions. In Section 3.7, we consider more general, nonlinear RLSC. Here, it is much more difficult to make nonlinear RLSC efficient — to implement RLSC directly, we need to store the entire kernel matrix, which is intractable for large problems. Instead of solving the problem directly, we consider several approximation schemes, including bagging smaller classifiers, and constructing low rank approximations to the kernel matrix. We find that these schemes offer a viable alternative to SVMs, although the situation is much less clear-cut than in the linear case. Finally, we see that some of these ideas allow us to construct an interesting new algorithm that uses the SVM's hinge loss.

3.2 The Mathematics of Regularized Least-Squares Classification

In this section we review the mathematics associated with Regularized Least-Squares Classification. Because the square loss function is differentiable everywhere, the mathematics ends up being substantially simpler than in the Support Vector Machine case.

Our RLSC problem can be stated as

$$\min_{f \in \mathcal{H}} \frac{1}{\ell} \sum_{i=1}^{\ell} (y_i - f(\mathbf{x}_i))^2 + \lambda ||f||_K^2. \tag{3.3}$$

Again, we can use the classical Representer Theorem to show that the solution f^* to the above regularization problem has the form

$$f^*(\mathbf{x}) = \sum_{i=1}^{\ell} c_i K(\mathbf{x}, \mathbf{x}_i), \tag{3.4}$$

so our function finding problem is reduced to the ℓ-dimensional problem of finding the c_i. Substituting this into our regularization functional, and again defining $\mathbf{c} \equiv [c_1 \ldots c_\ell]^T$, and K as the ℓ-by-ℓ matrix satisfying $K_{ij} \equiv K(\mathbf{x}_i, \mathbf{x}_j)$, we can rewrite our problem as

$$\min_{\mathbf{c} \in \mathbf{R}^\ell} \frac{1}{\ell}(\mathbf{y} - K\mathbf{c})^T(\mathbf{y} - K\mathbf{c}) + \lambda \mathbf{c}^T K \mathbf{c}. \tag{3.5}$$

For convenience, we divide our functional by 2, which will obviously not change the minimizer:

$$\min F(\mathbf{c}) = \min_{\mathbf{c} \in \mathbf{R}^\ell} \frac{1}{2\ell} \sum_{i=1}^{\ell} (\mathbf{y} - K\mathbf{c})^T(\mathbf{y} - K\mathbf{c}) + \frac{\lambda}{2} \mathbf{c}^T K \mathbf{c}. \tag{3.6}$$

Because we now have a convex differentiable function, we can find the minimzer simply by taking the derivative with respect to $\mathbf{c}$:

$$\begin{aligned} \nabla F_{\mathbf{c}} &= \frac{1}{\ell}(\mathbf{y} - K\mathbf{c})^T(-K) + \lambda K\mathbf{c} && (3.7) \\ &= -\frac{1}{\ell}K\mathbf{y} + \frac{1}{\ell}K^2\mathbf{c} + \lambda K\mathbf{c}. && (3.8) \end{aligned}$$

Because the kernel matrix K was positive semidefinite, $\nabla F_{\mathbf{c}} = 0$ is a necessary and sufficient condition for optimality of a candidate solution $\mathbf{c}$. By multiplying through by K^{-1} if K is invertible, or reasoning that we only need a single solution if K is not invertible, the optimal K can be found by solving

$$(K + \lambda \ell I)\mathbf{c} = \mathbf{y}, \tag{3.9}$$

where I denotes an appropriately-sized identity matrix. We see that a Regularized Least-Squares Classification problem can be solved by solving a single system of linear equations. Using a standard algorithm [49] for solving this system will require $O(\ell^2 d)$ time and $O(\ell^2)$ storage to generate and store the kernel matrix, and $O(\ell^3)$ time to find the solution given the kernel matrix K. For large problems, the $O(\ell^2)$ memory required to store the kernel matrix will render a direct approach intractable. We will revisit this subject in Section 3.6, where we see that for the case of linear RLSC, these problems disappear through the appropriate use of linear algebra, and in Section 3.7, where we explore approximations to the generally intractable nonlinear, kernelized, RLSC.

An important, intimately related difference between SVMs and RLSC is to the issue of sparsity. For Support Vector Machines, only those points which live on the margin, or fail to meet the margin requirements, are support vectors, and only those points will have nonzero coefficients c_i in the derived function. In RLSC, it should be clear that there is no reason to expect any sparsity in the solution. In general, all the coefficients will be nonzero, and therefore, in order to use a kernel-based RLSC, we expect to compute the kernel products of every training point with every test point. This has substantial computational implications, implying that nonlinear RLSC, applied directly, will be much more computationally expensive to apply to new data than a nonlinear SVM. This lack of sparsity is directly linked to the $O(\ell^2)$ storage requirement mentioned in the previous paragraph — the fact that the final solution is sparse is what allows an SVM algorithm to avoid examining the majority of the kernel matrix.

For Support Vector Machines, we used the hinge loss function $(1-yf(\mathbf{x}))_+$, which was not differentiable at $yf(\mathbf{x}) = 1$. To put the problem into a form that was easy to work with (a quadratic programming problem with linear constraints), we introduced slack variables. This led to the standard SVM primal. We then introduced dual variables associated with the constraints, and used Lagrangian duality to derive a dual program by taking the derivative of the Lagrangian with respect to the primal variables and eliminating them from the formulation. In the case of RLSC, this is not necessary: the functional $L(\mathbf{c})$ is differentiable everywhere, and the solution can be found directly via differentiation. However, we will see in Section 3.7 that introducing slack variables and deriving a "dual" as we did for SVMs allows us to make some interesting connections with other ideas from the literature. Proceeding along these lines, we rewrite our problem as:

$$\min_{\mathbf{c}\in\mathbf{R}^\ell} \quad \frac{1}{2\ell}\xi^T\xi + \frac{\lambda}{2}\mathbf{c}^T K\mathbf{c} \tag{3.10}$$

$$\text{subject to :} \qquad K\mathbf{c} - \mathbf{y} = \xi \tag{3.11}$$

We introduce a vector of dual variables $\mathbf{u}$ associated with the equality constraints, and form the Lagrangian:

$$L(\mathbf{c}, \xi, \mathbf{u}) = \frac{1}{2\ell}\xi^T\xi + \frac{\lambda}{2}\mathbf{c}^T K\mathbf{c} - \mathbf{u}^T(K\mathbf{c} - \mathbf{y} - \xi). \tag{3.12}$$

As in Chapter 2, we want to minimize the Lagrangian with respect to $\mathbf{c}$ and ξ, and maximize it with respect to $\mathbf{u}$. We can derive a "dual" by eliminating $\mathbf{c}$ and ξ from the problem. We take the derivative with respect to each in turn, and set it equal to zero:

$$\frac{\partial L}{\partial \mathbf{c}} = \lambda K\mathbf{c} - K\mathbf{u} = 0 \implies \mathbf{c} = \frac{\mathbf{u}}{\lambda} \tag{3.13}$$

$$\frac{\partial L}{\partial \xi} = \frac{1}{\ell}\xi + \mathbf{u} = 0 \implies \xi = -\ell\mathbf{u} \tag{3.14}$$

Unsurprisingly, we see that both $\mathbf{c}$ and ξ are simply expressible in terms of the dual variables $\mathbf{u}$. Substituting these expressions into the Lagrangian, we arrive at the reduced Lagrangian

$$L^R(\mathbf{u}) = \frac{\ell}{2}\mathbf{u}^T\mathbf{u} + \frac{1}{2\lambda}\mathbf{u}^T K\mathbf{u} - \mathbf{u}^T(K\frac{\mathbf{u}}{\lambda} - \mathbf{y} + \ell\mathbf{u}) \tag{3.15}$$

$$= -\frac{\ell}{2}\mathbf{u}^T\mathbf{u} - \frac{1}{2\lambda}\mathbf{u}^T K\mathbf{u} + \mathbf{u}^T\mathbf{y}. \tag{3.16}$$

We are now faced with a differentiable concave maximization problem in $\mathbf{u}$, and we can find the maximum by setting the derivative with respect to $\mathbf{u}$ equal to zero:

$$\nabla L^R_{\mathbf{u}} = -\ell\mathbf{u} - \frac{1}{\lambda}K\mathbf{u} + y = 0 \Longrightarrow (K + \lambda\ell I)\mathbf{u} = \lambda\mathbf{y}. \tag{3.17}$$

After we solve for $\mathbf{u}$, we can recover $\mathbf{c}$ via Equation 3.13. It is trivial to check that the resulting $\mathbf{c}$ satisfies Equation 3.9. While the exercise of deriving the dual may seem somewhat pointless, its value will become clear in later sections, when we use this dual approach to make connections to other ideas. Additionally, some authors [41, 40] who have published on this topic derive the dual in order to find the solution, possibly in order to make their least-squares formulation look as much like the standard SVM formulation as possible. In that context, it seems that this clear explanation of how the identical solution can be derived directly is useful.

3.3 Leave-One-Out Bounds for RLSC

An accurate method for evaluating the performance of a learning system on future data is to train ℓ different classifiers, each on a subset of $\ell - 1$ training points, and to test each classifier on the single "held out" training point. This procedure is known as *leave-one-out cross validation*, and it was shown by Luntz and Brailovsky in 1969 [72] that the leave-one-out error was an (almost) unbiased estimate of the generalization error. Although the leave-one-out error is a very desirable quantity to compute from this viewpoint, it is computationally extremely expensive, and often prohibitive. Therefore, it is useful to have a *bound* on the leave-one-out error, preferably one that is easy to compute.

Throughout this section, we denote f_S to be the function obtained when RLSC is solved on the entire dataset, and f_{S^i} to be the function obtained when the i'th point is removed from the training set.

For RLSC, we shall see that the leave-one-out error can be computed exactly without actually solving ℓ problems of size $\ell - 1$. Recall that the solution to an RLSC problem is of the form

$$f_S(\mathbf{x}) = \sum_{i=1}^{\ell} c_i K(\mathbf{x}, \mathbf{x}_i), \tag{3.18}$$

where the vector $\mathbf{c}$ is obtained by solving a system of linear equations:

$$(K + \lambda \ell I)\mathbf{c} = \mathbf{y}, \tag{3.19}$$

or, formally,

$$\mathbf{c} = (K + \lambda \ell I)^{-1}\mathbf{c} = \mathbf{y}. \tag{3.20}$$

Throughout this section, we assume λ fixed, and define $G \equiv (K + \lambda \ell I)^{-1}$ for notational convenience. For RLSC, the following equation holds:

$$y_i - f_{S^i}(\mathbf{x}_i) = \frac{y_i - f_S(\mathbf{x_i})}{1 - G_{ii}} \tag{3.21}$$

The above equation tells us that for RLSC, we can compute the leave-one-out values knowing only the function values and the diagonals of the inverse matrix G, without actually solving $\ell - 1$ training problems. The above statement is not too difficult to prove; we present here an argument given in the very readable monograph of Green and Silverman [50].

Define $\mathbf{f}^i$ to be the vector whose jth coordinate is $f_{S^i}(\mathbf{x_j})$, the value of the function obtained by training on a dataset with the ith point removed, applied to the jth point. Define $\mathbf{Y}^i$ to be a vector whose jth entry is y_j if $j \neq i$, and whose ith entry is $f_{S_i}(\mathbf{x_i})$. Then

$$\mathbf{f}^i = GY^i. \tag{3.22}$$

To see this, we simply note that for any f in the RKHS,

$$\begin{aligned}
\sum_{j=1}^{\ell}(\mathbf{Y}_j^i - f(\mathbf{x}_j))^2 + \lambda||f||_K^2 \quad &\geq \quad \sum_{j \neq i}(\mathbf{Y}_j^i - f(\mathbf{x}_j))^2 + \lambda||f||_K^2 && (3.23)\\
&\geq \quad \sum_{j \neq i}(\mathbf{Y}_j^i - f_{S^i}(\mathbf{x}_j))^2 + \lambda||f_{S^i}||_K^2 && (3.24)\\
&= \quad \sum_{j=1}^{\ell}(\mathbf{Y}_j^i - f_{S^i}(\mathbf{x}_j))^2 + \lambda||f_{S^i}||_K^2, && (3.25)
\end{aligned}$$

where the second inequality follows by the definition of f_{S^i} as the minimizer of a Tikhonov regularization problem with the ith point removed and the final inequality follows because $f_{S^i}(\mathbf{x}_i) = \mathbf{Y}_i^i$ by definition. This shows that f_{S^i} is the optimal solution to a modified problem where the right-hand-side is replaced with Y^i, as desired.

Knowing that $\mathbf{f}^i = G\mathbf{Y}^i$, we can prove our desired relationship by considering just the ith linear equation:

$$y_i - f_{S^i}(\mathbf{x}_i) \quad = \quad y_i - (G\mathbf{Y}^i)_i \tag{3.26}$$

$$
\begin{aligned}
&= y_i - \sum_{j=1}^{\ell} G_{ij}\mathbf{Y}_j^i && (3.27)\\
&= y_i - \sum_{j \neq i} G_{ij} y_j - G_{ii} f_{S^i}(\mathbf{x_i}) && (3.28)\\
&= y_i - \sum_{j=1}^{\ell} G_{ij} y_j + G_{ii}(y_i - f_{S^i}(\mathbf{x_i}))) && (3.29)\\
&= y_i - f_S(\mathbf{x_i}) + G_{ii}(y_i - f_{S^i}(\mathbf{x_i})). && (3.30)
\end{aligned}
$$

Combining terms and simple rearrangement gives Equation 3.21. The above equation tells us how to compute the leave-one-out error exactly for RLSC, in far less time than it would take to solve ℓ problems of size $\ell - 1$. However, this computation requires the diagonals of the matrix $G = (K + \lambda\ell I)^{-1}$, which for large problems is impossible to obtain; as we will see below, we solve large problems (especially in the linear case) using substantially less time and (more importantly) vastly less memory than the computation of G would require. Therefore, this relationship, while exact, does not represent a practical approach to computing the leave-one-out error for large RLSC problems.

An alternative approach, introduced by Craven and Wahba [26], is known as the *generalized approximate cross-validation*, or GACV for short. Instead of actually using the entries of the inverse matrix G directly, an approximation to the leave-one-out value is made using the *trace* of G:

$$
y_i - f_{S^i}(\mathbf{x}_i) \approx \frac{y_i - f_S(\mathbf{x_i})}{1 - \frac{1}{\ell}\mathrm{tr}(G)} \qquad (3.31)
$$

This approach, while being only an approximation rather than an exact calculation, has an important advantage. The trace of a matrix is the sum of its eigenvalues. If the eigenvalues of K are known, the eigenvalues of G can be computed easily for any value of λ, and we can then easily select the value of λ which minimizes the leave-one-out bound. Unfortunately, computing all the eigenvalues of K is in general much too expensive, so again, this technique is not practical for large problems.

Jaakkola and Haussler introduced an interesting class of simple leave-one-bounds [54] for kernel classifiers. In particular, they considered classifiers that minimize an expression $J(\alpha)$ of the form

$$J(\alpha) = -\frac{1}{2}\alpha^T Y K Y \alpha + \sum_{i=1}^{\ell} F(\alpha_i), \tag{3.32}$$

where $0 \leq \alpha_i \leq 1$ and $F(\alpha_i)$ is continuous. For these systems, Jaakkola and Haussler provide a simple and elegant proof that the number of leave-one-out errors of a classifier in the above framework is bounded above by

$$|\mathbf{x_i} : -y_i \sum_{j \neq i} \alpha_j y_j K(\mathbf{x_i}, \mathbf{x_j}) \leq 0| \tag{3.33}$$

In words, their bound says that if a given training point $\mathbf{x}$ can be classified correctly by f_S *without using the contribution of* $\mathbf{x}$ *to* f_S, then $\mathbf{x}$ cannot be a leave-one-out error — $f_{S^i}(\mathbf{x_i}) >= 0$.

This class of problems includes Support Vector Machines (without an unregularized bias term, although Joachims extended the bound to include SVMs with a free "b" term, at the expense of an algebraically complicated derivation [56, 58]), but does not include Regularized Least Squares Classification, where the signs of the coefficients α_i are unconstrained. The proof given by Jaakkola and Haussler does depend crucially on the sign of α_i. However, using a slightly more involved argument, we are able to obtain a similar bound for RLSC, as we now see.

Our first goal is to show that points which live "outside" the hyperplanes, for which $y_i f(\mathbf{x}_i) > 1$, cannot be leave-one-out errors (they will still be classified correctly if they are removed from the training set). To this end, we introduce a parametrized family of functionals.

$$\begin{aligned} F_{\alpha,i}(f) &= \frac{1}{\ell}\sum_{j \neq i}(y_j - f(\mathbf{x}_j)^2) + \frac{\alpha}{\ell}(y_i - f(\mathbf{x_i}))^2 + \lambda||f||_K^2 & (3.34) \\ &\equiv R(f) + \frac{\alpha}{\ell}(y_i - f(\mathbf{x_i}))^2, & (3.35) \end{aligned}$$

where α is between 0 and 1. At $\alpha = 1$, we have the standard RLSC problem. As α decreases from one to zero, we are weighting the ith data point less and less. When

$\alpha = 0$, we have the reduced problem in which the ith training point has been removed entirely.

Consider two values of α, with $\alpha_1 > \alpha_2$. Let f_1 and f_2 be the minimizers of $F_{\alpha_1,i}$ and $F_{\alpha_2,i}$, respectively. The following two equations hold:

$$R(f_1) + \frac{\alpha_1}{\ell}(y_i - f_1(\mathbf{x_i}))^2 \leq R(f_2) + \frac{\alpha_1}{\ell}(y_i - f_2(\mathbf{x_i}))^2 \tag{3.36}$$

$$R(f_2) + \frac{\alpha_2}{\ell}(y_i - f_2(\mathbf{x_i}))^2 \leq R(f_1) + \frac{\alpha_2}{\ell}(y_i - f_1(\mathbf{x_i}))^2 \tag{3.37}$$

Summing these two equations, subtracting $R(f_1) + R(f_2)$ from both sides, and collecting terms appropriately, we easily arrive at

$$\frac{\alpha_1 - \alpha_2}{\ell}(y_i - f_1(\mathbf{x_i}))^2 \leq \frac{\alpha_1 - \alpha_2}{\ell}(y_i - f_2(\mathbf{x_i}))^2. \tag{3.38}$$

Dividing through by the (positive) quantity $\frac{\alpha_1 - \alpha_2}{\ell}$, we see that

$$(y_i - f_1(\mathbf{x_i}))^2 \leq (y_i - f_2(\mathbf{x_i}))^2. \tag{3.39}$$

We see that as we decrease α from one to zero, the "training error" at $\mathbf{x_i}$ is monotonically nondecreasing. Additionally, it is easy (though somewhat tedious) to show that the solution $\mathbf{c}$ is a continuous function of α. Taken together, we can see that if $y_i f(\mathbf{x_i}) > 1$, then as α decreases from one to zero, the training error $(y_i - f(\mathbf{x_i})^2)$ will increase continuously, and $\mathbf{x_i}$ will stay on the same side of the hyperplane $y_i f(\mathbf{x_i}) = 1$, so $\mathbf{x_i}$ cannot be a leave-one-out error.

We turn now to those points for which $y_i f(\mathbf{x_i}) \leq 1$, and we show that in this case, we can derive a bound using an argument essentially identical to the one used to prove the Jaakkola-Haussler bound. For convenience, we will work in terms of the dual variables $\mathbf{u}$. Recall that $\mathbf{c}$ can be computed from $\mathbf{u}$ using $\mathbf{c} = \frac{\mathbf{u}}{\lambda}$ (Equation 3.13). Also, recall that the optimal solution $\mathbf{u}^*$ is found by minimizing the following expression (renamed from $L^R(\mathbf{u})$ to $L(\mathbf{u})$ for convenience):

$$L(\mathbf{u}) = \frac{\ell}{2}\mathbf{u}^T\mathbf{u} + \frac{1}{2\lambda}\mathbf{u}^T K\mathbf{u} - \mathbf{u}^T\mathbf{y}. \tag{3.40}$$

Suppose we remove the ℓ'th data point (without loss of generality), solve the resulting problem $L_{-\ell}$ of size $\ell - 1$, and the resulting solution is $\mathbf{u}^\dagger$. Now suppose we add the ℓ'th point back to the data set, and considering setting $u_\ell = u_\ell^*$, and performing a joint optimization over the remaining variables only. Our goal is then to minimize the reduced problem

$$\begin{aligned} L^R(u_1, \ldots, u_{\ell-1}) &= \frac{\ell}{2}\sum_{i=1}^{\ell-1}(u_i^2 - u_i y_i) + \frac{1}{2\lambda}\sum_{i,j\neq\ell} u_i u_j K_{ij} + \frac{1}{\lambda}\sum_{i\neq\ell} u_i u_\ell^* K_{i\ell} \quad (3.41) \\ &= L_{-\ell}(u_1, \ldots, u_{\ell-1}) + \frac{1}{\lambda}\sum_{i\neq\ell} u_i u_\ell^* K_{i\ell} \quad (3.42) \end{aligned}$$

We denote the vector $[u_1^*, \ldots, u_{\ell-1}^*]'$ by $u_{-\ell}^*$. Then, by definition of the associated optimization problems, $L_{-\ell}(\mathbf{u}^\dagger) \leq L_{-\ell}(\mathbf{u}_{-\ell}^*)$, and $L^R(\mathbf{u}_{-\ell}^*) \leq L^R(\mathbf{u}^\dagger)$. Expanding this out, we find that

$$\begin{aligned} L^R(\mathbf{u}_{-\ell}^*) &\leq L^R(\mathbf{u}^\dagger) \quad (3.43) \\ \Longrightarrow L_{-\ell}(\mathbf{u}_{-\ell}^*) + \frac{1}{\lambda}\sum_{i\neq\ell} u_i^* u_\ell^* K_{i\ell} &\leq L_{-\ell}(\mathbf{u}^\dagger) + \frac{1}{\lambda}\sum_{i\neq\ell} u_i^\dagger u_\ell^* K_{i\ell} \quad (3.44) \\ \Longrightarrow \frac{1}{\lambda}\sum_{i\neq\ell} u_\ell^* \sum_{i\neq\ell} u_i^* K_{i\ell} &\leq \frac{1}{\lambda}\sum_{i\neq\ell} u_\ell^* \sum_{i\neq\ell} u_i^\dagger K_{i\ell} \quad (3.45) \\ \Longrightarrow c_\ell^* \sum_{i\neq\ell} c_i^* K_{i\ell} &\leq c_\ell^* \sum_{i\neq\ell} c_i^\dagger K_{i\ell}, \quad (3.46) \end{aligned}$$

where the last equation follows from Equation 3.13 and inducing the c^* and $c^\dagger$ from u^* and $u^\dagger$ in the obvious way.

Fortunately, we are able to reason about the sign of c_ℓ^* using a simple geometric argument. Recall that we are only interested in the case where $y_\ell f(\mathbf{x}_\ell) < 1$, as we showed above that this is a prerequisite for a point to be a leave-one-out error. Suppose that $y_\ell = 1$. Then $f(\mathbf{x}_\ell) < 1$, which implies that the slack variable $\xi_\ell^* < 0$ by the definition of the slack variables, which in turn implies that $c_\ell^* > 0$ by Equations 3.13 and 3.14. Therefore, dividing Equation 3.46 by c_ℓ^*, we find

$$\sum_{i\neq\ell} c_i^* K_{i\ell} \leq c_i^\dagger K_{i\ell}. \quad (3.47)$$

This is precisely the result we are after. The left-hand side is a quantity that we can easily compute given only the solution the original problem c^*, and the right-hand side is the function value for the leave-one-out problem, bounded below for the case where $y_\ell = 1$.

If $y_\ell = -1$, then $f(\mathbf{x}_\ell) > -1$ implies $\xi_\ell^* > 0$ and $c_\ell^* < 0$, implying that

$$\sum_{i \neq \ell} c_i^\dagger K_{i\ell} \leq c_i^* K_{i\ell}, \tag{3.48}$$

again giving the desired leave-one-out bound.

Combining these results, we can bound the number of leavel-one-out errors for leave-one-out classification by

$$|\mathbf{x_i} : y_i f(\mathbf{x_i}) < 1 \wedge y_i(\sum_{j \neq i} c_j K(\mathbf{x_i}, \mathbf{x_j})) \leq 0|. \tag{3.49}$$

Further examination reveals that we can actually remove the condition $y_i f(\mathbf{x_i}) < 1$, because points which fail to satisfy this condition will automatically fail the second condition as well. This is because if $y_i f(\mathbf{x_i}) > 1$, then c_i and y_i are of opposite signs, and, using the fact that $K(\mathbf{x_i}, \mathbf{x_i}) > 0$ for any kernel satisfying Mercer's condition, we can see that the deletion of the ith point will in fact always *increase* $y_i f(\mathbf{x_i})$ (assuming $y_i f(\mathbf{x_i}) > 1$ to begin with). So in fact, the standard Jaakola-Haussler bound holds for RLSC as well (although their proof does not), and we can bound the number of leave-one-out errors of RLSC via

$$|\mathbf{x_i} : y_i(\sum_{j \neq i} c_j K(\mathbf{x_i}, \mathbf{x_j})) \leq 0| \tag{3.50}$$

$$\implies |\mathbf{x_i} : y_i(f(\mathbf{x_i}) - c_i K(\mathbf{x_i}, \mathbf{x_i})) \leq 0| \tag{3.51}$$

We can simplify the bound further using the precise geometric definitions of the variables. Looking again at Equations 3.13 and 3.14, we can easily combine them to derive

$$\xi_i = -\ell \lambda c_i. \tag{3.52}$$

Combining this with the definition of the ξ_i,

$$\xi_i = f(\mathbf{x_i}) - y_i, \tag{3.53}$$

we find that

$$c_i = \frac{y_i - f(\mathbf{x}_i)}{\ell\lambda}. \tag{3.54}$$

Using this, we can eliminate c_i from the bound, arriving at

$$|\mathbf{x_i} : y_i \left(f(\mathbf{x_i}) - \left(\frac{y_i - f(\mathbf{x_i})}{\ell\lambda} \right) K(\mathbf{x_i}, \mathbf{x_i}) \right) \leq 0| \tag{3.55}$$

$$\implies \quad |\mathbf{x_i} : y_i \left((1 + \frac{K(\mathbf{x_i}, \mathbf{x_i})}{\ell\lambda}) f(\mathbf{x_i}) - \frac{y_i}{\ell\lambda} K(\mathbf{x_i}, \mathbf{x_i}) \right) \leq 0|. \tag{3.56}$$

Supposing $y_i = 1$, for some i, the condition reduces to

$$(1 + \frac{K(\mathbf{x_i}, \mathbf{x_i})}{\ell\lambda}) f(\mathbf{x_i}) \leq \frac{1}{\ell\lambda} K(\mathbf{x_i}, \mathbf{x_i}) \tag{3.57}$$

$$\implies \quad f(\mathbf{x_i}) \leq \frac{K(\mathbf{x_i}, \mathbf{x_i})}{\ell\lambda + K(\mathbf{x_i}, \mathbf{x_i})}. \tag{3.58}$$

Reasoning similarly for the case where $y_i = -1$, we find that we can bound the number of leave-one-out errors for RLSC via

$$|\mathbf{x_i} : y_i f(\mathbf{x_i}) \leq \frac{K(\mathbf{x_i}, \mathbf{x_i})}{K(\mathbf{x_i}, \mathbf{x_i}) + \ell\lambda}|. \tag{3.59}$$

This form of the bound is particularly appealing, as it makes explicit the connection between leave-one-out bounds and algorithmic stability (discussed in Section 5). In particular, we see that if we fix λ and increase ℓ, asymptotically, the requirement on $y_i f(\mathbf{x})$ in order for $\mathbf{x}$ not to be a leave-one-out error is $O(\frac{1}{\ell})$. This meshes well with the notion of *strong stability*, discussed in Chapter 5; intuitively, we expect bounds of this form because Tikhonov regularization is a strongly stable algorithm.

3.4 The Intuition of RLSC

In this section, we present a collection of toy examples comparing Support Vector Machines and Regularized Least-Squares Classification. SVMs have a well-deserved reputation as a robust and accurate method for binary classification. A priori, it is not obvious how the RLSC algorithm will perform. RLSC has two features that seem somewhat "perverse" to the casual observer, although we can plausibly argue that these features are not very problematic.

The first seemingly odd characteristic of RLSC is that large positive values are as bad as large negative values — we pay $(f(\mathbf{x})-y)^2$. For example, if we classify a positive example with $f(\mathbf{x}) = 5$, we pay a penalty of 16. This seems less desirable than an SVM, which pays no penalty in this case. However, what we imagine (and find in the examples) is that this simply has the effect of pulling the "margin" functions $f(\mathbf{x}) = 1$ and $f(\mathbf{x}) = -1$ farther apart, without changing the direction of the hyperplane.

The second seemingly odd characteristic of RLSC compared to SVMs is the squaring of the penalty function. This seems to imply more dependancy on outliers for RLSC than for SVMs. However, the squared dependance on the distance from a correct output value must be balanced by the fact that we pay a penalty for all ℓ training points, not just for the misclassified or almost-misclassified points as we do for SVMs. This has a strong mitigating effect on the ability of outliers to influence the solution.

For our examples, we consider Gaussian data generated from two classes (with additional outliers in some examples) and linear separating hyperplanes. For ease of comparison, we use a variant of SVMs in which there is no bias term b — the hyperplanes for both SVMs and RLSC pass through the origin.[2] We show one data set per page, and the top, middle, and bottom figures correspond to $\lambda = .1$, $\lambda = .01$, and $\lambda = .001$ respectively. In all figures, the solid lines correspond to the SVM solution $\mathbf{w} \cdot \mathbf{x} = 0$ (the two fainter lines correspond to the hyperplanes $\mathbf{w} \cdot \mathbf{x} = 1$ and

[2]We note in passing that if the SVM hyperplane is forced to pass through the origin, there may not be unbounded support vectors directly at the edge of the margin in both classes.

$\mathbf{w} \cdot \mathbf{x} = -1$), and the dashed lines correspond to the RLSC solution.

In the first four examples, we generate 100 training examples and 1000 test examples for each class. In Figure 3-2, we consider a situation in which the two Gaussian have symmetric (unit) variance, and the two distributions are well-separated. We observe that the SVM and RLSC algorithms find nearly identical hyperplanes, that the margin is insensitive to the choice of λ, and that a test error rate of zero is achieved by both classifiers. In Figure 3-3, we again consider well-separated datasets, but with a nondiagonal covariance matrix. Here we find that the SVM solution is more sensitive to the choice of λ than the RLSC solution, but that the two functions perform identically on the test set, with only a single test error (almost certainly an extreme outlier in one of the Gaussian distributions). In Figure 3-4, we again consider a diagonal covariance matrix, but move the datasets closer together, so that there is more overlap between the two distributions. Here we find that the margin of the SVM solution is very sensitive to the choice of λ, although its direction and test performance are insensitive. The RLSC solution is insensitive to the choice of λ. Finally, we consider Figure 3-5, which is the non-diagonal covariance version of Figure 3-4. Again we find that the SVM margin is the only quantity which is sensitive to λ, and the functions returned by RLSC and SVM are essentially identical. In all four of these figures, for all chosen values of λ, the test performance of RLSC and SVM is identical to within a single example.

We now consider some examples in which outliers are added to the data. In these examples, we add five outliers to one or both classes, and observe the effects on the solution and the test error rate. We consider both the test rate on data drawn from the original distribution, and the test rate on data with 50 outliers drawn from the same distribution as the training outliers. In other words, we consider the testing performance separately under the assumptions that the outliers do and do not reflect the "true" test distribution.

In Figure 3-6, we add "opposite" outliers to the negative class, centered on the other side of the positive class from the negative class. Comparing to Figure 3-5, we see that the RLSC is basically unaffected, and that for the SVM, the added outliers

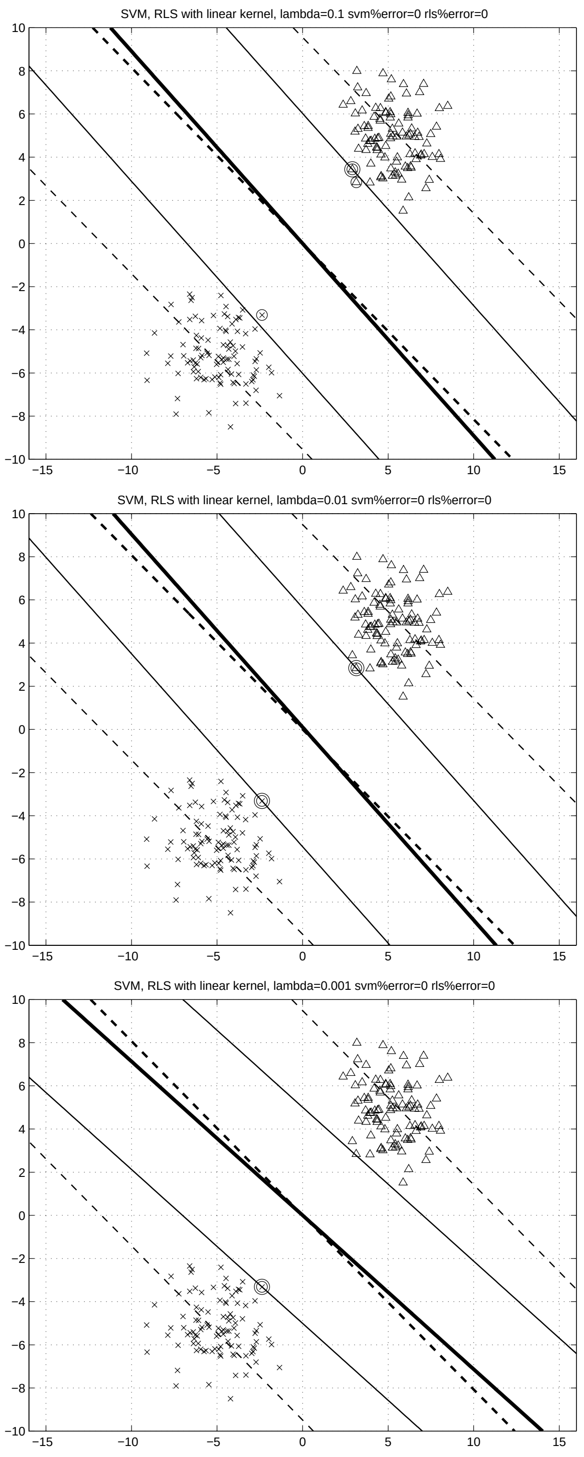


Figure 3-2: RLSC and SVM: Symmetric, well-separated datasets.

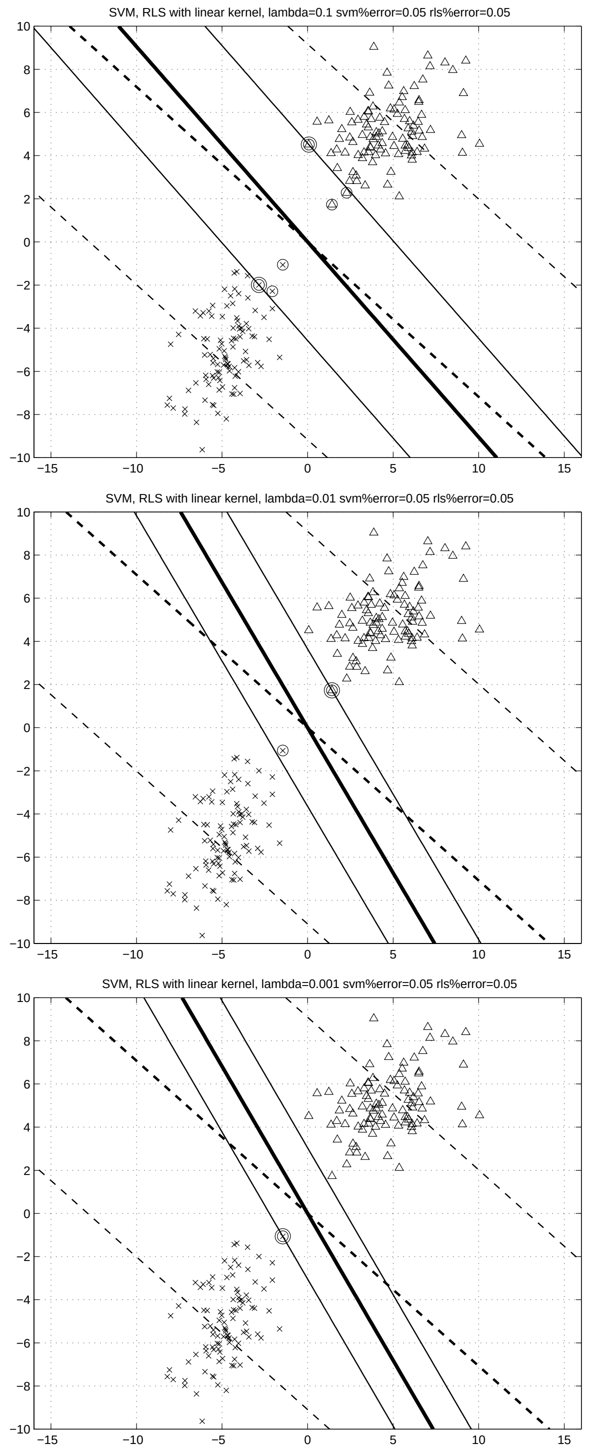


Figure 3-3: RLSC and SVM: Unsymmetric, well-separated datasets.

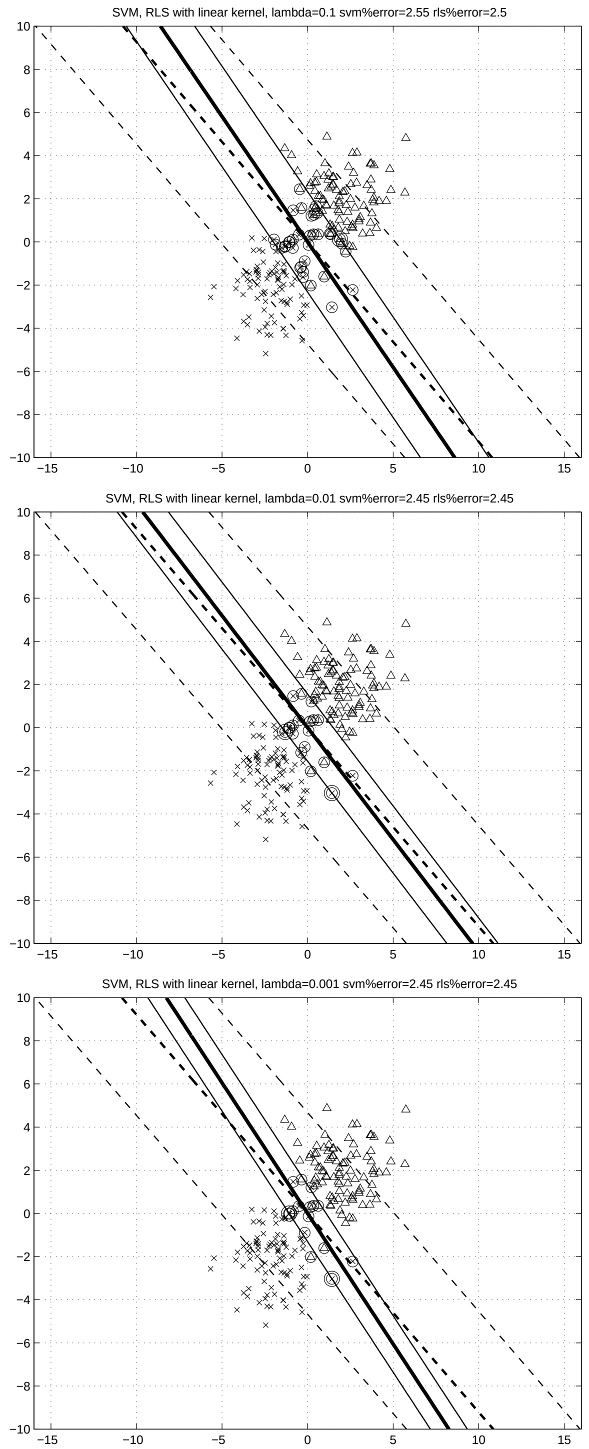


Figure 3-4: RLSC and SVM: Symmetric, closely-spaced datasets.

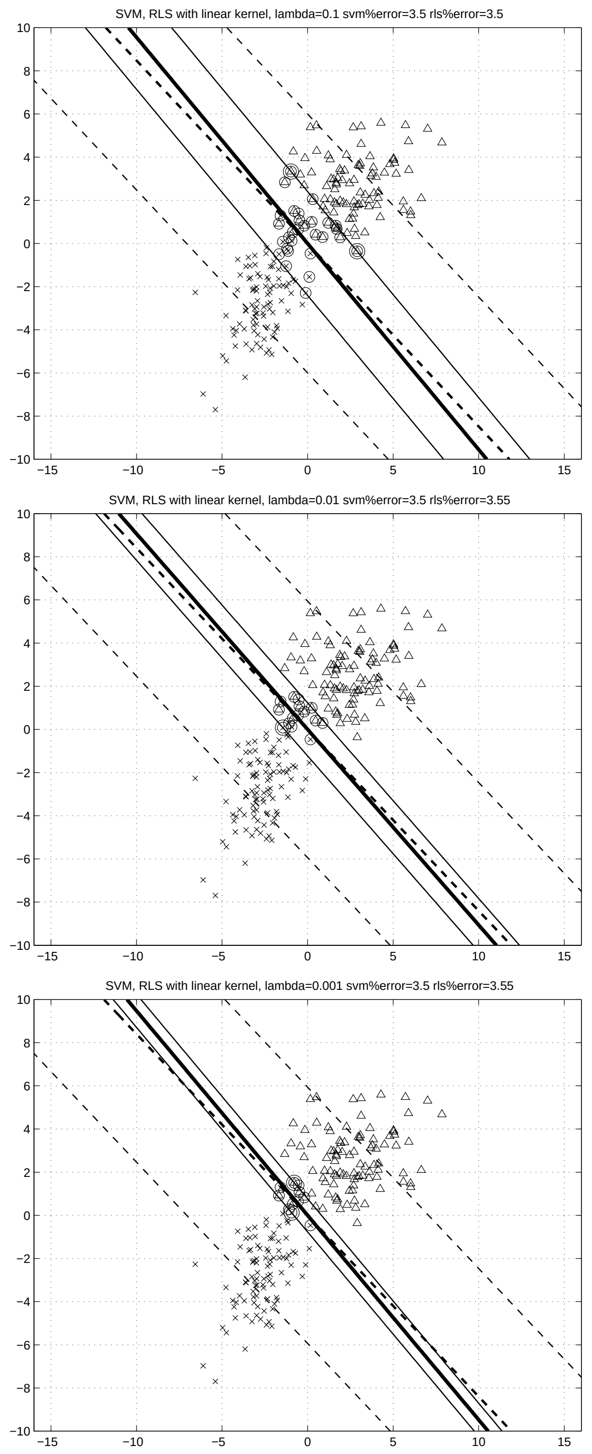


Figure 3-5: RLSC and SVM: Unsymmetric, closely-spaced datasets.

seem to actually *improve* the robustness: because the new examples cannot possibly be classified correctly, shrinking the margin has less effect on the training error rate, and the sensitivity to λ is therefore reduced. In Figure 3-7, we add "opposite" examples to both classes, and observe an essentially identical effect.

In Figure 3-8, we add "twisting" outliers to the negative class, which are placed off to the left of the data in an attempt to twist the separating hyperplane. We find that both the RLSC and the SVM functions are twisted substantially. The RLSC function found is much less sensitive to λ than the SVM function; however, the best performance is found for the SVM, at $\lambda = .1$. In this case, we find the SVM outperforms RLSC by .75% (if we also include outliers in the test data, the difference declines to .4%). At smaller values of λ, RLSC outperforms the SVM, and RLSC's performance is insensitive to the choice of λ. Figure 3-9, in which the "twisting" outliers are added to both classes, tells a similar story.

To sum up, across a wide range of reasonable examples, Support Vector Machines and Regularized Least-Squares Classification perform very similarly. Both the classification functions and their performance are essentially identical (it is not obvious that *any* of the differences reported are statistically significant). Both SVMs and RLSC are somewhat susceptible to outliers, and it is not clear that RLSC are more susceptible. These experiments do not conclusively "prove" that RLSC is just as good as (or better than) SVMs, but they do indicate that for many reasonable examples, the two algorithms will perform similarly. Further evidence is given by RLSC's performance on real-world datasets, as demonstrated in the work of Fung and Mangasarian [40] (see also the next section), and later in this chapter in experiments on text classification and face recognition.

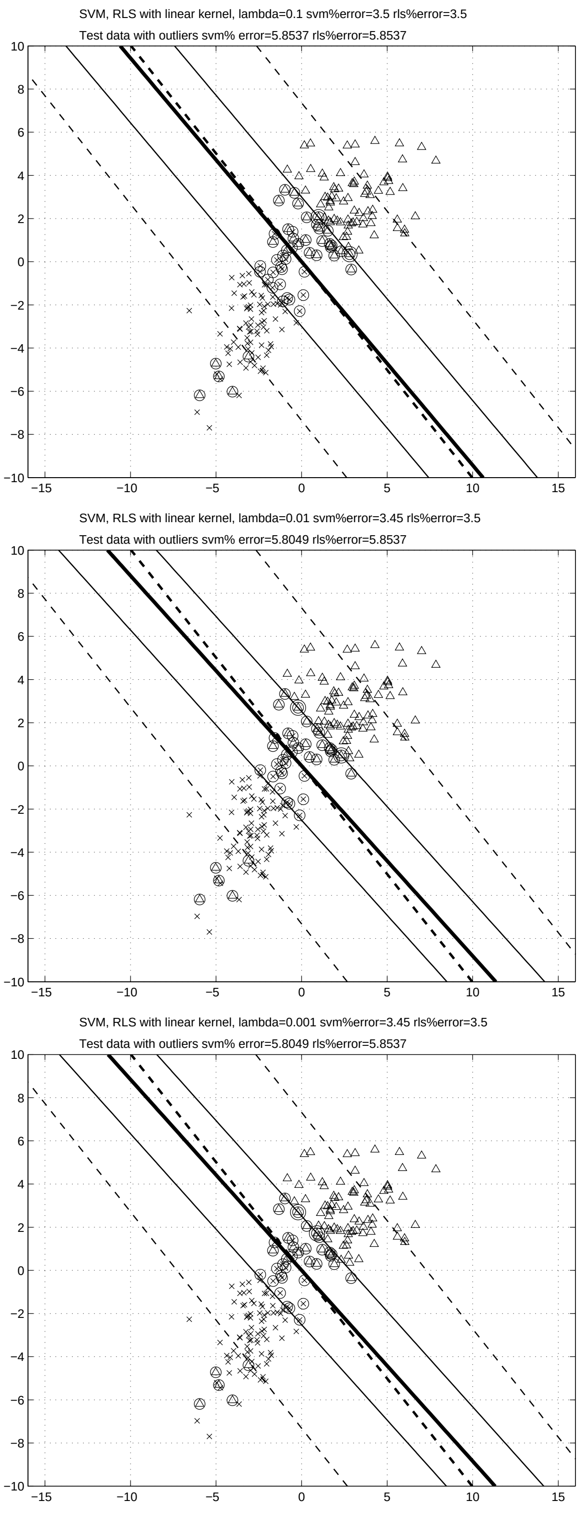
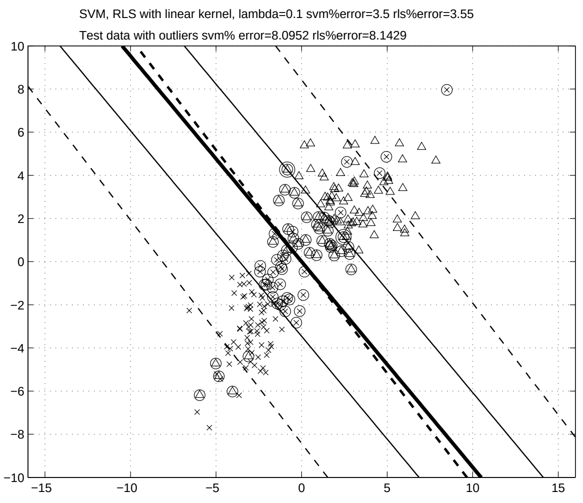
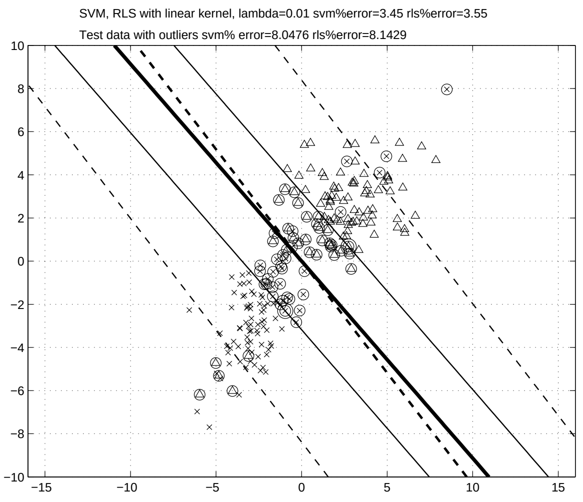


Figure 3-6: RLSC and SVM: Unsymmetric, closely-spaced datasets, “opposite” outliers in one class.





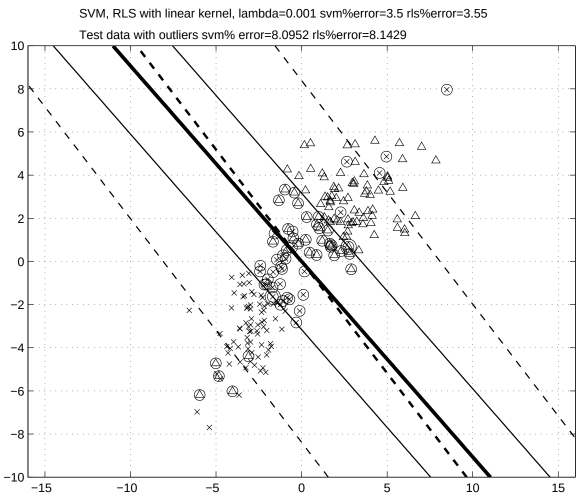
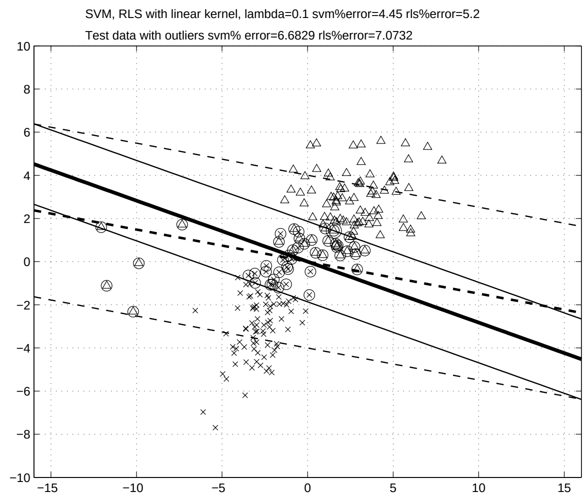
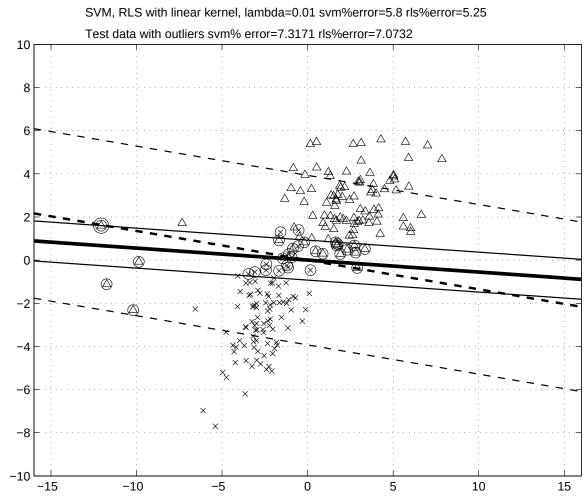


Figure 3-7: RLSC and SVM: Unsymmetric, closely-spaced datasets, "opposite" outliers in both classes.





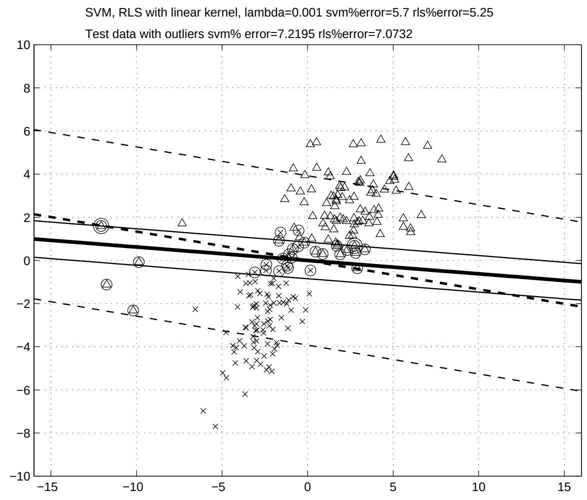


Figure 3-8: RLSC and SVM: Unsymmetric, closely-spaced datasets, “twisting” outliers in one class.

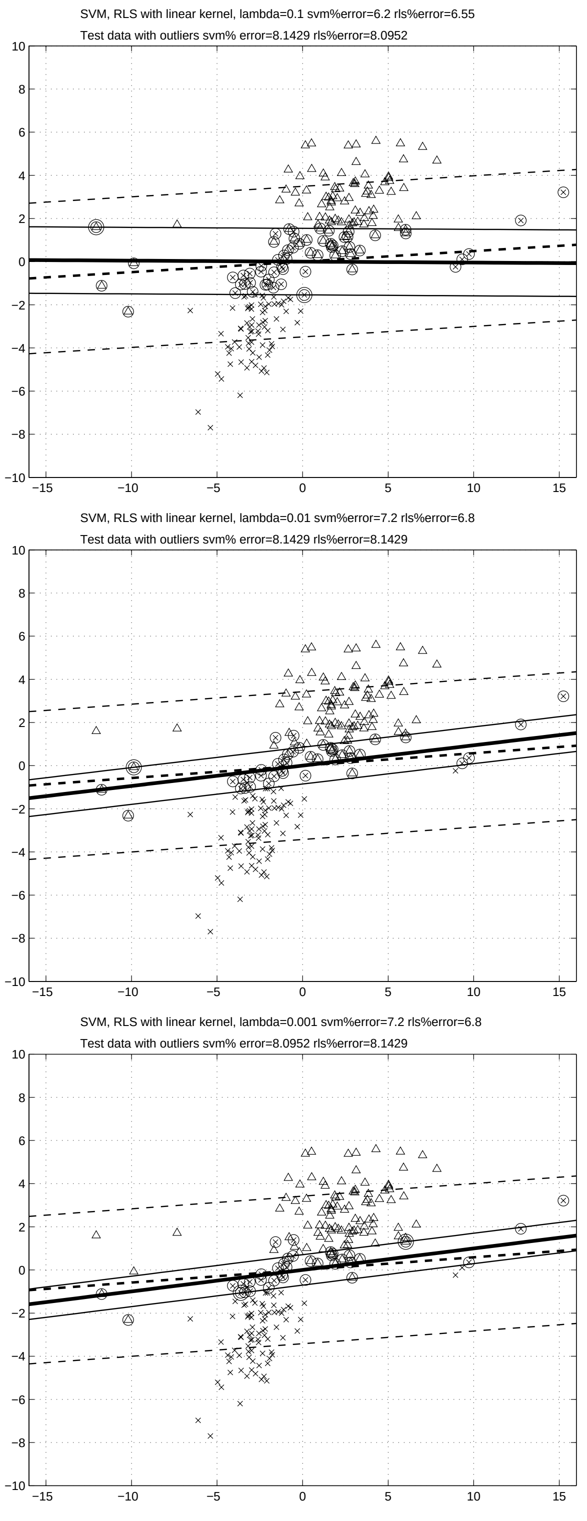


Figure 3-9: RLSC and SVM: Unsymmetric, closely-spaced datasets, “twisting” outliers in both classes.

3.5 Previous Work

3.5.1 1990 and Before

The idea of Regularized Least-Squares Classification is not a new one. All the pieces of it have existed in the literature for at least ten years, and several of these pieces have often been combined. For many problems, the square loss is the most natural, intuitive and mathematically tractable loss function. The mathematics of finding a linear function that minimizes the square loss over a set of data was first derived by Gauss in the 1800's [43]. The idea of regularization is apparent in the work of Tikhonov and Arsenin [111], who used least-squares regularization to restore well-posedness to ill-posed problems. Schönberg's seminal article on smoothing splines [99] also used regularization. These authors considered regression problems rather than classification, and did not use Reproducing Kernel Hilbert Spaces as regularizers.

In 1989, Girosi and Poggio [48, 86] considered regularized classification and regression problems with the square loss. They used pseudodifferential operators as their stabilizers; these are essentially equivalent to using the norm in an RKHS.

In 1990, Wahba [119] considered square-loss regularization using the norm in a Reproducing Kernel Hilbert Space as a stabilizer. Wahba considered only regression problems.

As we see, by 1990, the pieces of Regularized Least-Squares Classification were already well-known in various places of the literature. When we perform RLSC, we are essentially treating the classification problem as a regression problem, with the y values restricted to 1 or -1. Viewed in this light, the RLSC algorithm is an obvious extension of literature that is at least a decade old. Nevertheless, the algorithm has been at least twice "discovered" independently in recent years, in both cases as a modification of the Support Vector Machine.

3.5.2 Fung and Mangasarian

In 2001, Fung and Mangasarian rederived the Regularized Least Squares Classification formulation, naming the resulting machines Proximal Support Vector Machine Classifiers [41, 40]. The later paper, published in conference proceedings, acknowledges in passing in the abstract that the algorithm "can also be interpreted as regularized least squares and considered in the much more general context of regularized networks", in response to personal communication from Tomaso Poggio. However, this paper begins from a more geometric approach to SVMs. Viewing both the standard and proximal SVMs as attempts to classify data using large margin hyperplanes, the standard SVM penalizes points which do not meet the margin requirement, whereas the proximal SVMs instead penalize points according to their (lack of) proximity to the appropriate hyperplane. This is obviously exactly equivalent to RLSC. They mention that "our specific formulation leads to a strongly convex objective function which is not always the case in" [regularization networks], but it is unclear what is meant by this. No intuition as to why RLSC is about as good as SVMs is provided.

They begin by deriving a linear version of RLSC. Possibly in order to make their formulation look more like an SVM, they start with a formulation that is essentially identical to Equation 3.10, where slack variables have already been introduced. They also explicitly include a bias term b,[3] but because the square of this bias term is penalized, this is identical to adding an extra dimension of all ones to the data, and not using a b. In this sense, adding the b term makes the notation slightly more confusing, but doesn't add any expressive power to the model.

The two main contributions of this paper are a very fast approach to solving linear RLSC, and extensive computational results showing that in nonlinear and linear RLSC produce results of essentially equal quality to SVMs on benchmark datasets. We focus here on the first point, inviting the reader to peruse the papers for more information on the second point. The presentation here is notationally different from

[3]Denoted as γ in their work. Inexplicably, they choose to call the slack variables (our ξ_i) the y_i, flying in the face of standard machine learning practice — generally, we think of trying to learn a function from the data $\mathbf{x}$ to labels y.

Fung and Mangasarian's (we rephrase everything in our notation, and suppress the b term), but the argument is identical. We recall that the goal of RLSC is to solve a system of the form

$$(K + \lambda \ell I)\mathbf{c} = \mathbf{y}. \tag{3.60}$$

In the case of a linear kernel, if we denote by A the ℓ by d data matrix in which the data points are rows and the dimensions are columns, we have the relation

$$K = AA^T, \tag{3.61}$$

and the system we want to solve becomes

$$(AA^T + \lambda \ell I)\mathbf{c} = \mathbf{y}. \tag{3.62}$$

Noting that $AA^T + \lambda \ell I$ is positive definite and therefore invertible (AA^T is positive semidefinite, and $\lambda \ell I$ is positive definite), we can write

$$\mathbf{c} = (AA^T + \lambda \ell I)^{-1}\mathbf{y}. \tag{3.63}$$

Solving this system directly involves finding the inverse of an ℓ-by-ℓ matrix, which is not in general tractable. However, we can express the inverse by using the Sherman-Morrison-Woodbury formula [49]:

$$(M + BCD^T)^{-1} = M^{-1} - M^{-1}B(C^{-1} + D^T M^{-1} B)^{-1} D^T M^{-1}. \tag{3.64}$$

In our case, $M = \lambda \ell I$, $M^{-1} = \frac{1}{\lambda \ell} I$, $B = D' = A$, and $C = C^{-1} = I$:

$$\begin{aligned}
(\lambda \ell I + AA^T) &= \frac{I}{\lambda \ell} - \frac{I}{\lambda \ell} A(I + A^T \frac{I}{\lambda \ell} A)^{-1} A^T \frac{I}{\lambda \ell} && (3.65)\\
&= \frac{I}{\lambda \ell} - \frac{I}{\lambda \ell} A(\lambda \ell I + A^T A)^{-1} (\frac{I}{\lambda \ell})^{-1} A^T \frac{I}{\lambda \ell} && (3.66)\\
&= \frac{I}{\lambda \ell} - \frac{I}{\lambda \ell} A(\lambda \ell I + A^T A)^{-1} A^T && (3.67)\\
&= \frac{1}{\lambda \ell}(I - A(\lambda \ell I + A^T A)^{-1} A^T). && (3.68)
\end{aligned}$$

Fung and Mangasarian suggest explicitly computing the inverse of the d-by-d matrix (recall d is the number of dimensions) $(\lambda\ell I + A^T A)$, and then using this matrix to solve Equation 3.63 for $\mathbf{c}$, without forming any ℓ-by-ℓ matrices. Specifically,

$$\begin{aligned}
\mathbf{c} &= (AA^T + \lambda\ell I)^{-1}\mathbf{y} && (3.69)\\
&= \frac{1}{\lambda\ell}(I - A(\lambda\ell I + A^T A)^{-1}A^T)\mathbf{y} && (3.70)\\
&= \frac{\mathbf{y}}{\lambda\ell} - \frac{1}{\lambda\ell}A(((\lambda\ell I + A^T A)^{-1})^{-1}(A^T\mathbf{y})), && (3.71)
\end{aligned}$$

where we may have made powerful use of the associativity of matrix multiplication, allowing the multiplication of the vector $\mathbf{y}$ by an ℓ-by-ℓ matrix without actually forming the matrix.

This appears to represent the first instance in machine learning theory (the technique is "standard" in numerical linear algebra) of directly using the low-rank nature of the linear kernel matrix to turn a problem in the number of datapoints into a problem in the number of dimensions. However, the complicated algebraic nature of the Sherman-Morrison-Woodbury formula seems to somewhat obscure the essential observation, which is precisely that the matrix AA^T, while having size ℓ-by-ℓ, has rank at most d. Once we understand this, we immediately see that any factorization of AA^T that exploits its low-rank nature will allow us to achieve a similar result. See Section 3.6 for more on this point.

Fung and Mangasarian also suggest nonlinear proximal support vector machines, but here their case is much weaker, and they seem to make an odd error in the derivation. As we have formulated it, the RLSC problem as stated in Equation 3.6 already includes the general nonlinear approach, and includes linear RLSC as a special case. Fung and Mangasarian start from the linear case, where

$$\mathbf{w} = \sum_{i=1}^{\ell} c_i K(x, x_i), \tag{3.72}$$

which in turn implies that

$$||f||_K^2 \equiv \mathbf{c}^T K \mathbf{c} \equiv \mathbf{w}^T \mathbf{w}. \tag{3.73}$$

Their linear proximal support vector machine therefore (correctly) minimizes the weighted sum $\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{\ell}\sum_{i=1}^{\ell}(y_i - f(\mathbf{x}_I))^2$. However, in the nonlinear case, instead of minimizing the sum of $\frac{\lambda}{2}\mathbf{c}^T K\mathbf{c}$ and the average training error, they (inexplicably) minimize $\frac{\lambda}{2}\mathbf{c}^T\mathbf{c}$ directly. This leads to terms of the form KK^T in the dual problem, another indication of the oddness of this approach. In any case, the method requires solving an ℓ-by-ℓ system of equations, which is not tractable for large problems. They suggested using a "reduced kernel method", which amounts to choosing a small subset of the data points at random and only allowing those points to have nonzero coefficients (this was previously suggested by Lee and Mangasarian for a different SVM-like formulation in [68]). This is in fact a good idea, and has been suggested independently by several authors in different contexts. However, because of the initial error in their derivation of the nonlinear approach, their method is needlessly mathematically complicated. In Section 3.7, we discuss in detail this "reduced set" method, extensively exploring the literature on the subject and applicability of this idea to nonlinear RLSC.

3.5.3 Hans Suykens

Hans Suykens has also rederived the RLSC algorithm, under the name of Least Squares Support Vector Machines. He has been extremely prolific on the topic, publishing literally dozens of papers; the papers most relevant to the topics discussed in this thesis are [109, 110, 107, 108, 118, 105, 44, 106]. However, these papers do not seem to advance either the theory or practice of RLSC. Like the work of Fung and Mangasarian, Suykens derives RLSC as a modification of Support Vector Machines, rather than directly as an implementation of Tikhonov regularization with a particular loss function, and seems somewhat ignorant of previous work in approximation theory. Suykens uses RLSC with an unregularized bias term, leading to slightly more complicated algorithms. Although the speed of a variety of different methods for solving RLSC are examined, no real attempt to handle large-scale problems (problems for which it is infeasible to store and work with ℓ-by-ℓ matrices) is made. Empirical work and examples are given, but the least-squares SVM is generally compared to nonregularized algorithms such as trees, nearest neighbors, and linear discriminants, and the examples are often relatively small benchmark datasets such as two-spiral or n-spiral problems. Generalizations to multiclass formulations are made in the obvious ways (see Chapter 3).

One simple but relatively interesting technique suggested in [106] and [108] is the idea of restoring sparsity to the solution via an iterative pruning approach. At each iteration, a small fraction of points corresponding to the smallest multipliers are removed, and the system is retrained. Pruning is continued until validation performance is observed to decline. While this will have the effect of sparsifying the final function, it is important to note that at each iteration, the points for which $(y - f(\mathbf{x}))^2$ are minimal (the points closest to satisfying $|f(\mathbf{x})| = 1$) will be removed. This will have the effect of increasing the dependence on outliers. No real empirical evaluation of the approach is given; it would be interesting to see if the technique was of practical use.

3.5.4 Conclusions

In our own work, we do not claim to have "invented" the RLSC algorithm. Instead, our primary focus is to show how to use RLSC for very large problems, in both the linear and nonlinear cases. The two cases are quite different. As we shall see, in the linear case, by cleverly applying simple knowledge of matrix algebra and algorithms, we can obtain an extremely fast algorithm. In the nonlinear case, we will see that we need to modify the basic algorithm to make the problem tractable. This difficulty is mostly ignored in the work of Fung and Mangasarian and Suykens, but we recognize it as a major obstacle to the widespread adoptions of RLSC and address it directly.

3.6 Linear RLSC

Training an RLSC system involves the solution of an ℓ-by-ℓ system of linear equations. In the specific case of a linear kernel, $K = AA^T$, where A is the ℓ-by-d matrix whose rows are the data points. As we saw in the previous section discussing the work of Fung and Mangasarian, it is possible to use the Sherman-Morrison-Woodbury formula to transform the problem into the solution of a d-by-d system of equations. This is certainly a reasonable approach in some circumstances. However, we instead prefer to solve the system

$$(AA^T + \lambda\ell I)\mathbf{c} = \mathbf{y} \tag{3.74}$$

using the Conjugate Gradient algorithm (see Appendix C), which solves a linear system of the form $Mx = b$ by repeatedly multiplying a candidate solution x' by the vector M. In the case of linear RLSC, we can easily form the matrix-vector product in $O(2\ell d + \ell)$ time by taking advantage of the associativity of matrix multiplication:

$$\begin{aligned}
(AA^T + \lambda\ell I)\mathbf{x} &= AA^T\mathbf{x} + \lambda\ell I\mathbf{x} && (3.75)\\
&= A(A^T\mathbf{x}) + \lambda\ell I\mathbf{x} && (3.76)
\end{aligned}$$

It is difficult to say whether or not this approach is faster than using the Sherman-Morrison-Woodbury formula, because the total time required by the algorithm is proportional to the number of Conjugate Gradient iterations required, which in turn depends both on the matrix A and the accuracy required. However, the Conjugate Gradient approach is applicable in an important situation where the Sherman-Morrison-Woodbury formula approach is not, which is the case where d (the number of dimensions is very large), but is sparse. This is a situation that often arises in text classification problems. Here, $d >> \ell$, and both AA^T and A^TA are dense matrices, so the Sherman-Morrison-Woodbury formula is not feasible. However, we can still form a matrix-vector product $(AA^T + \lambda\ell I)\mathbf{x}$ in $O(\bar{d}\ell)$, where $\bar{d}$ is the average number of nonzero entries per example. We turn now to just such an application.

3.6.1 Linear RLSC Application: Text Categorization

In text categorization, we are given a corpus of documents and a collection of categories. The goal is to place each document into the correct category. SVM and RLSC are both algorithms for solving *binary* classification tasks — we address multiclass classification in detail in Chapter 4, and explain the text classification task in more detail. We introduce this application here as it is a prime example of a case where linear RLSC provides an impressive computational advantage over linear SVMs.

In this example, we consider two standard datasets. The `20newsgroups` data consists of 19,997 documents in 20 categories, which we randomly split into 15,935 training documents and 3,993 test documents. The dimensionality of the `20newsgroups` data is 62,021. The `sector105` data consists of 9,649 documents in 105 categories, which we randomly split into 4,797 training documents and 4,822 test documents. The dimensionality of the `sector105` data is 55,197. For details on the preprocessing performed, see [90].

Rennie and Rifkin considered the multiclass text classification problem [90], and showed that on both the datasets considered here, a multiclass SVM was able to achieve results that were substantially better than any previously reported results. Here we consider the use of RLSC instead of SVMs. What we find is that RLSC's are essentially identically as accurate as SVMs, and, on the `20newsgroups` data, they are substantially faster.

For all the experiments considered in this section, we ran RLSC with $\lambda\ell = 1$. The multiclass classification schemes run a number of binary classifiers, and then combine the results to make a multiclass prediction — for details, see Chapter 4 and [90]. The OVA scheme trains as many binary classifiers as there are classes, and the BCH 63 scheme trains 63 binary classifiers.

Tables 3.1 and 3.2 compare the accuracies of SVM and RLSC on this task. We also considered subsampling the training set, using fewer than the maximal number of examples per class, in order to see how this affects the accuracy. We find that across all the training set sizes and all the schemes, RLSC performs as well or slightly

	800		250		100		30	
	SVM	RLSC	SVM	RLSC	SVM	RLSC	SVM	RLSC
OVA	0.131	0.129	0.167	0.165	0.214	0.211	0.311	0.309
BCH 63	0.125	0.129	0.164	0.165	0.213	0.213	0.312	0.316

Table 3.1: A comparison of SVM and RLSC accuracy on the `20newsgroups` multiclass classification task. The top row indicates the number of documents/class used for training. The left column indicates the multiclass classification scheme. Entries in the table are the fraction of misclassified documents.

	52		20		10		3	
	SVM	RLSC	SVM	RLSC	SVM	RLSC	SVM	RLSC
OVA	0.072	0.066	0.176	0.169	0.341	0.335	0.650	0.648
BCH 63	0.067	0.069	0.176	0.178	0.343	0.344	0.653	0.654

Table 3.2: A comparison of SVM and RLSC accuracy on the `sector105` multiclass classification task. The top row indicates the number of documents/class used for training. The left column indicates the multiclass classification scheme. Entries in the table are the fraction of misclassified documents.

better than SVMs, indicating that from an accuracy perspective, RLSC is certainly a feasible choice for a text classification task.

Table 3.3 shows the relative time required to train the SVM and RLSC multiclass system. We immediately note that for the `20newsgroup` dataset, which is the larger of the two datasets, the total time to train the RLSC system is much less than the time required for the SVM system for both the OVA and BCH schemes. For the smaller, `sector105` dataset, the SVM system is faster for OVA, and the RLSC is somewhat faster for BCH. We also note that the time required per classifier varies greatly between the BCH and OVA schemes for SVM, but is nearly constant for RLSC. This is consistent with our expectations — the BCH problems involve separating a subset of the classes from the remaining classes, which is likely to be a much more difficult task than simply separating one class from the remainder as is required in the OVA scheme. This makes the training time much slower for the SVM. For RLSC, the Conjugate Gradient algorithm is known to have relatively strong invariance to the choice of right hand side vector, which is all we are changing in the different classifiers.

A few additional remarks about Table 3.3 are in order. The total times given are

Dataset/Scheme	Classifiers	Size	SVM	S/C	RLSC	S/C
20newsgroups, OVA	200	15,935	10,045	50	1,309	6.5
20newsgroups, BCH 63	630	15,935	104,027	165	3,913	6.2
sector105, OVA	1,050	4,797	1,706	1.6	3,655	3.5
sector105, BCH 63	630	4,797	4,620	7.3	2,857	4.5

Table 3.3: Relative training times of SVM and RLSC. Classifiers indicates the total number of binary classifiers trained (for the full training set). Size indicates the size of the training set. The SVM and RLSC columns indicate the total time required (in seconds) to train the classifiers. The associated S/M columns measure the numbers of seconds per classifier.

the times required to train at all data sizes shown in Tables 3.1 and 3.2, not just the time for training on the "full" datset. As such, the columns measuring the amount of time required per classifier are not entirely accurate, although the time required to train on the "full" dataset strongly dominates (at a guess, roughly 90% in all cases) the total training time. More importantly, the SVM system, which was implemented using `SvmFu`, reuses kernel products between machines. If the dataset is sufficiently small that the kernel matrix can be stored in memory (as is the case for `sector105`), this will provide huge savings, and the time to train n machines will be much less than linear in n. If the dataset is large, this effect will be negligible, and we will have to use about the same amount of time to train each machine. RLSC, as implemented here, does not have this property: each classifier is trained separately, no information is preserved across the classifiers. This explains why RLSC gives a large speedup over SVMs for the `20newsgroups` data, but not for the `sector105` data.

Additionally, it is worth noting that in order to keep the estimate of the relative speed of RLSC compared to SVMs conservative, we included the time required to load the data in the above figures. SvmFu, implemented in `C++` appears to take only a few seconds to read each dataset, whereas RLSC, which is implemented in `Matlab`, takes about 30 seconds to read each dataset from disk. Therefore, if we were to remove the data loading times, RLSC would be about 300 seconds faster on all tasks, whereas SVM would be only 100 seconds faster.

3.6.2 Linear RLSC Application: Tumor Classification

We also mention briefly an application to multiclass molecular tumor classification. In [89], the author (among others) considered multiclass tumor classification. The data consisted of 190 tumors divided into 14 classes. Each example is the 16,063 dimensional output of a DNA microarray. The best results reported in [89] were obtained using a one-vs-all linear SVM system with $C = .1$. In a leave-one-out framework, the system attained an accuracy of 156 out of 190 samples. If instead we used linear RLSC with $\lambda\ell = 10$, we achieve an accuracy of 158 out of 190 samples. It is worth mentioning that because we have a small number of examples in a large number of dimensions, we solve the problem by storing the kernel matrix AA^T explicitly and solving $(K + \lambda\ell I)\mathbf{c} = \mathbf{y}$ directly.

3.7 Nonlinear RLSC

3.7.1 Low-Rank Kernel Approximations

Several authors have suggested the use of low-rank approximations to the kernel matrix in order to avoid explicit storage of the entire kernel matrix [103, 68, 122, 45, 32]. These techniques can be used in a variety of methods, including Regularized Least Squares Classification, Gaussian Process regression and classification, interior point approaches to Support Vector Machines.

The approaches all rely on choosing a subset of m of the training points (or a subset of size m, abusing notation slightly) representing those points exactly in the Hilbert space, and representing the remaining points approximately as a linear combination of the points in m. The methods differ in the approach to choosing the subset, and in the matrix math used to represent the hypothesis. Both Smola and Schölkopf [103] and Williams and Seeger [122] suggest the use of the approximate kernel matrix

$$\tilde{K} = K_{\ell m} K_{mm}^{-1} K_{m\ell}. \tag{3.77}$$

The assumption is that m is small enough so that K_{mm} is invertible. If we use a method that does not need the entries of $\tilde{K}$, but only needs to multiply vectors by $\tilde{K}$, we can form the matrix-vector produce by taking advantage of the $\tilde{K}$'s representation as three separate matrices:

$$\begin{aligned} \tilde{K}\mathbf{x} &= (K_{\ell m}K_{mm}^{-1}K_{m\ell})\mathbf{x} & (3.78) \\ &= (K_{\ell m}(K_{mm}^{-1}(K_{m\ell}\mathbf{x}))). & (3.79) \end{aligned}$$

This approximation is known as the *Nystrom approximation* and is justified theoretically in [122]. If we approximate the kernel matrix using a subset of size m, we can show that we are actually approximating the first m eigenfunctions of the integral operator induced by the kernel (evaluated at all the data points). The quality of this approximation will of course depend on the rate of decay of the eigenvalues of the kernel matrix.

The two sets of authors differ in their suggestions as to how to choose the subset. Williams and Seeger simply pick their subset randomly. Smola and Schölkopf suggest a number of greedy approximation methods that greedily, iteratively reduce some measure of difference between K and $\tilde{K}$, such as the trace of $K - \tilde{K}$. They also suggest approximations to the full greedy approach in order to speed up their method. However, all their methods are likely to involve computing nearly all of the entries of K over the course of their operation (for reasonable subset sizes), implying that forming a matrix approximation in this manner will already be slower than training an SVM.

Referring to n as the set of points not in m (the remaining training points), it is easy to show that $\tilde{K}$ has the property that $\tilde{K}_{mm} = K_{mm}$, $\tilde{K}_{m\ell} = K_{m\ell}$, and (trivially by symmetry) $\tilde{K}_{\ell m} = K_{\ell m}$. In other words, a the approximated kernel product $\tilde{K}(\mathbf{x_1}, \mathbf{x_2})$ for a pair of examples $\mathbf{x_1}$ and $\mathbf{x_2}$ will be exact if either of $\mathbf{x_1}$ or $\mathbf{x_2}$ are in m. It is also easy to show that $\tilde{K}_{nn} = K_{\ell m}K_{mm}^{-1}K_{m\ell}$.

Inverting the matrix K_{mm} takes $O(m^3)$ steps formally, and in practice, performing the inversion may be ill-conditioned, so this approach does not not seem to be a good choice for real-world applications.

Fine and Scheinberg [32] instead suggest the use of an incomplete Cholesky factorization. Noting that a positive semidefinite kernel matrix can always be represented as $K = R^T R$ where R is an upper-triangular matrix, the incomplete Cholesky factorization is formed by using only the first m rows of R. Fine and Scheinberg cleverly choose the subset on the fly — at each iteration, they pivot a pair of rows of the matrix so that the largest diagonal element becomes the next element in the subset. The total running time of their approach is only $O(mk^2)$[4]. In terms of the feature space, this pivoting technique is equivalent to, at each step, adding to the subset the data point which is most poorly represented by the current subset. Although the final matrix contains only m nonzero rows, it is still upper-triangular, and we write

$$\tilde{K} = R_m^T R_m. \tag{3.80}$$

However, there is a serious problem with this approach. Although Fine and Scheinberg claim that their method "takes the full advantage of the greedy approach for for the best reduction in the approximation bound $\mathrm{tr}(\Delta Q)$", this is not the case. To reduce the approximation bound, we need to consider which element not in the basis will best represent (the currently unrepresented portion of) all remaining non-basis elements. This is what Smola and Schölkopf attempt in [103], at too large a computational cost. The Fine and Scheinberg approach, in contrast, adds to the basis the element which is most poorly represented by the current basis element. If we believe that the trace of $K - \tilde{K}$ is a useful measure of the quality of the approximation, this turns out to be a very poor choice, at least for Gaussian kernels. In particular, on two different datasets (see Section 3.7.3 below), we find that the portion of the trace accounted for by the Fine and Scheinberg approximation is consistently smaller than the portion of the trace accounted for by selecting the subset at random. This is not too hard to explain. Because the Fine and Scheinberg approach adds the most

[4]We have not formally defined "operations" (is a multiplication more expensive than an addition or a memor access?), but it is clear that the constant involved in this approach is small compared to the methods suggested in [103].

poorly represented element to the basis, under the Gaussian kernel, it will tend to add outliers — data points that are far from any other data point. The trace would be reduced much more by adding elements which are not as far away, but which are themselves close to a large number of additional points. Speaking informally, we want to add the points which are centers of clusters to the basis, not point which are outliers.

It is not hard to show that given the same subset and ordering of the points, the two representations are identical. The incomplete Cholesky approach is likely to be preferable in practice, as it is numerically stable, fast, and chooses the optimal greedy subset.

We now consider the use of a low-rank approximation, obtained by any of the above means, in RLSC.

3.7.2 Low-Rank Approximations for RLSC

The most obvious approach (and indeed, the one suggested in [122] in the context of Gaussian process classification and [32] in the context of Support Vector Machine classification) is to simply use the matrix $\tilde{K}$ in place of the original K, resulting in the system of linear equations:

$$(\tilde{K} + \lambda \ell I) = \mathbf{y}. \tag{3.81}$$

These equations can be solved using the conjugate gradient method, taking advantage of the factorization of $\tilde{K}$ to avoid having to store an ℓ-by-ℓ matrix. From a machine learning standpoint, this approach consists of taking a subset of the points, estimating the kernel values at the remaining points, then treating the estimate as correct and solving the original problem over all the data points. In practice, we found that this worked quite poorly, because the matrix eigenvalues do not decay sufficiently quickly (see Section 3.7.3).

Instead, we consider the following modified *algorithm*, alluded to (among other places) in [103]. We minimize the empirical risk over all points, but allow the c_i to be nonzero only at a specific subset of the points (identical to the points used in the low-rank kernel approximation). This leads to the following modified problem:

$$\min F(\mathbf{c_m}) \tag{3.82}$$

$$= \min_{\mathbf{c_m}\in\mathbf{R}^m} \quad \frac{1}{\ell}(\mathbf{y} - K_{\ell m}\mathbf{c_m})^T(\mathbf{y} - K_{\ell m}\mathbf{c_m}) + \lambda \mathbf{c_m}^T K \mathbf{c_m} \tag{3.83}$$

$$= \min_{\mathbf{c_m}\in\mathbf{R}^m} \quad \frac{1}{2\ell}\left(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T K_{\ell m}\mathbf{c_m} + \mathbf{c}^T K_{m\ell}K_{\ell m}\mathbf{c}\right) + \frac{\lambda}{2}\mathbf{c_m}^T K \mathbf{c_m}. \tag{3.84}$$

We take the derivative with respect to $\mathbf{c_m}$ and set it equal to zero:

$$\nabla F_{\mathbf{c_m}} = \frac{1}{\ell}\left(K_{m\ell}\mathbf{y} + K_{m\ell}K_{\ell m}\mathbf{c_m}\right) + \lambda K_{mm}\mathbf{c_m} = 0 \tag{3.85}$$

$$\Longrightarrow \quad (K_{m\ell}K_{\ell m} + K_{mm}\lambda\ell)\mathbf{c_m} = K_{m\ell}\mathbf{y}. \tag{3.86}$$

We see that when we only allow a subset of size m points to have nonzero coefficients in the expansion, we can solve a m by m system of equations rather than an ℓ-by-ℓ system. As with the standard full RLSC, we were able to find the system of equations that defines the optimal solution by setting the derivative equal to zero. Again, suppose for the sake of argument we decide to derive a "dual" problem. We introduce a vector of dual variables $\mathbf{u}$ — it is important to note that this vector is of length ℓ, not m. We form the Lagrangian:

$$L(\mathbf{c_m}, \xi, \mathbf{u}) = \frac{1}{2\ell}\xi^T\xi + \frac{\lambda}{2}\mathbf{c_m}^T K_{mm}\mathbf{c_m} - \mathbf{u}^T(K_{\ell m}\mathbf{c_m} - \mathbf{y} + \xi). \tag{3.87}$$

We take the derivative with respect to ξ:

$$\frac{\partial L}{\partial \xi} = \frac{1}{\ell}\xi - \mathbf{u} = 0 \tag{3.88}$$

$$\Longrightarrow \quad \xi = \ell\mathbf{u} \tag{3.89}$$

We take the derivative with respect to $\mathbf{c_m}$:

$$\frac{\partial L}{\partial \mathbf{c_m}} = \lambda K_{mm}\mathbf{c_m} - K_{m\ell}\mathbf{u} = 0 \tag{3.90}$$

$$\implies \quad \mathbf{c_m} = \frac{1}{\lambda} K_{mm}^{-1} K_{m\ell} \mathbf{u}. \tag{3.91}$$

Substituting these equations into L, we reduce the Lagrangian to:

$$\begin{aligned}
& L(\mathbf{u}) && (3.92)\\
= \; & \frac{\ell}{2}\mathbf{u}^T\mathbf{u} + \frac{1}{2\lambda}\mathbf{u^T} K_{\ell m} K_{mm}^{-1} K_{m\ell}\mathbf{u} - \frac{1}{\lambda}\mathbf{u^T} K_{\ell m} K_{mm}^{-1} K_{m\ell}\mathbf{u} + \mathbf{u}^T\mathbf{y} - \ell\mathbf{u}^T\mathbf{u} && (3.93)\\
= \; & \frac{\ell}{2}\mathbf{u}^T\mathbf{u} + \frac{1}{2\lambda}\mathbf{u^T}\tilde{K}\mathbf{u} - \frac{1}{\lambda}\mathbf{u^T}\tilde{K}\mathbf{u} + \mathbf{u}^T\mathbf{y} - \ell\mathbf{u}^T\mathbf{u} && (3.94)\\
= \; & -\frac{\ell}{2}\mathbf{u}^T\mathbf{u} - \frac{1}{2\lambda}\mathbf{u^T}\tilde{K}\mathbf{u} + \mathbf{u}^T\mathbf{y}. && (3.95)
\end{aligned}$$

Setting the derivative to zero yields

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{u}} &= -\ell\mathbf{u} - \frac{1}{\lambda}\tilde{K}\mathbf{u} + \mathbf{y} = 0 && (3.96)\\
\implies \quad & (\tilde{K} + \lambda\ell I)\mathbf{u} = \lambda\mathbf{y} && (3.97)
\end{aligned}$$

If we solve this system for $\frac{\mathbf{u}}{\lambda}$, *we are solving exactly the same system of equations as if we had used the Nystrom approximation at the subset m directly.* However, in order to find the optimal solution, we need to recover the vector $\mathbf{c_m}$ using the equation

$$\mathbf{c_m} = \frac{1}{\lambda} K_{mm}^{-1} K_{m\ell} \mathbf{u}. \tag{3.98}$$

To summarize, we suggest that instead of using the matrix $\tilde{K}$ directly in place of K, we consider a modified problem in which we require our function to be expressed in terms of the points in the subset m. Used directly, this leads to an algorithm that doesn't directly involve $\tilde{K}$, but uses only the component matrices K_{mm}, $K_{\ell m}$ and $K_{m\ell}$. Although it is not strictly necessary, we can take the Lagrangian dual of this problem, at which point we solve a system that is identical the original, full RLSC problem with K replaced with $\tilde{K}$. However, we do not use the resulting $\mathbf{u}$ vector directly, instead recovering the c_m by means of Equation 3.98.

A New Approach to SVMs

A development very similar to that made for RLSC in this chapter was made by Fine and Scheinberg for SVMs [32]. Standard approaches to solving SVMs use active set methods — they start with the all zero solution, and add non-zero variables as needed, maintain dual feasibility at each iteration. For many other linear and quadratic programming problems, interior point methods have become the method of choice, based on their fast performance and theoretical polynomial-time guarantees. Interior point methods work with dense (all non-zero) candidate solutions, and it is known based on geometric reasoning that for SVMs, the majority of the dual variables will be zero at optimality. Because the kernel matrix for SVMs is dense, this implies that each iteration of an interior point method would require multiplying a vector by this dense kernel matrix, which would have to be computed in full. Because an SVM is generally trained in less time than it takes simply to compute all the entries of the kernel matrix, it seemed that interior point methods were not useful for the SVM problem.

Fine and Scheinberg handled this problem by using low-rank approximation to the kernel matrix. Just as we did in this development, they noted that the linear and nonlinear cases are very different. For the linear case (in fewer dimensions than the number of data points), they used a product-form Cholesky factorization (see [7, 34, 46]) of the ℓ-by-ℓ kernel matrix AA^T, turning each iteration of the interior point method from an $O(\ell^2)$ operation to an $O(d^2\ell)$ operation[5]. For linear SVMs, this approach seems to work quite effectively. By taking advantage of the low-rank nature of the kernel matrix, interior point methods become a highly effective method for solving Support Vector Machines. The experiments presented in the Fine and Scheinberg paper are somewhat difficult to follow, with many possible confounding factors, but the method shows promise for linear SVMs. Of course, in the linear case,

[5]Fine and Scheinberg also considered using the Sherman-Morrison-Woodbury formulation instead of a product-form Cholesky factorization. They remark that the SMW update has sometimes been found to be numerically unstable, and give an example and an explanation of this behavior. However, the examples seem somewhat contrived, especially in light of the fact that in SVM applications, only a moderate amount of solution accuracy is required.

we can also solve learning problems extremely quickly and effectively using RLSC, as we showed in this chapter.

In the nonlinear case, the authors appeal to the body of work in machine learning suggesting that for many kernels, the eigenvalues of kernel operators are rapidly (exponentially) decaying, which implies that the kernel matrix will have rapid eigenvalue decay [103, 122, 123, 124]. If the eigenvalues of the matrix K are rapidly decaying, then we can accurately approximate K with a low-rank matrix $\tilde{K}$. The authors show how to perform an incomplete Cholesky decomposition with a pivot at every iteration so that the next "basis" data point chosen is one which is most poorly represented by the elements already used in the approximation. In a more traditional approach to Cholesky factorization (as described for example in [49]), we are interested in forming the complete factorization, and the pivoting is necessary if we encounter a column which is a linear combination of already examined columns. The end result of this factorization is an approximate kernel matrix

$$\tilde{K} = R^T R, \tag{3.99}$$

where R is an upper triangular matrix of which only the first m rows are nonzero. As we mentioned before, given the same subset m and the same ordering of the points, the $\tilde{K}$ obtained can also be written as

$$\tilde{K} = K_{\ell m} K_{mm}^{-1} K_{m\ell}, \tag{3.100}$$

although the $R^T R$ representation avoids some possible numerical instability issues associated with inverting K_{mm}, as well as providing an efficient mechanism for finding the "greedy optimal" subset m.

Once the matrix $\tilde{K}$ is (implicitly) determined, Fine and Scheinberg suggest substituting it directly for K in the dual SVM problem, resulting in the following modified SVM dual:

$$\max_{\alpha \in \mathbf{R}^\ell} \quad \textstyle\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2}\alpha^T \tilde{Q} \alpha \tag{3.101}$$

$$\text{subject to :} \quad \textstyle\sum_{i=1}^{\ell} y_i \alpha_i = 0 \tag{3.102}$$

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, \ell \tag{3.103}$$

Here, $\tilde{Q} \equiv Y \tilde{K} Y$, where Y is again a diagonal matrix with $Y_{ii} \equiv y_i$. As described earlier, this suggestion amounts to approximating the kernel products between pairs of points not in the subset m, then treating these approximations as correct. If $K - \tilde{K}$ and $Q - \tilde{Q}$ are very small, we expect the solution to this system to be essentially equivalent to the solution of the original SVM. However, as we show in the next section (in the context of RLSC), for reasonable subset sizes, $K - \tilde{K}$ is not very small. The eigenvalues of the kernel matrix do decay, but they don't decay very rapidly.

Suppose that, instead of using $\tilde{K}$ as a direct proxy for K, we again consider a modified version of the primal SVM problem, in which we require the function we find to have nonzero coefficients only for those data points in the subset m:

$$\min_{\mathbf{c} \in \mathbf{R}^\ell, \xi \in \mathbf{R}^\ell} \quad C \textstyle\sum_{i=1}^{\ell} \xi_i + \frac{1}{2}\mathbf{c}^T K \mathbf{c} \tag{3.104}$$

$$\text{subject to :} \quad y_i(\textstyle\sum_{j=1}^{\ell} c_j K(x_i, x_j) + b) \geq 1 - \xi_i \quad i = 1, \ldots, \ell \tag{3.105}$$

$$\xi_i \geq 0 \qquad i = 1, \ldots, \ell \tag{3.106}$$

$$c_i = 0 \qquad i \notin m \tag{3.107}$$

By defining the vector $\mathbf{c_m}$ to be the entries of $\mathbf{c}$ corresponding to the subset m, we can rewrite this problem as:

$$\min_{\mathbf{c_m} \in \mathbf{R}^m, \xi \in \mathbf{R}^\ell} \quad C \textstyle\sum_{i=1}^{\ell} \xi_i + \frac{1}{2}\mathbf{c_m}^T K_{mm} \mathbf{c_m} \tag{3.108}$$

$$\text{subject to :} \quad Y(K_{\ell m}\mathbf{c_m} + b) \geq \mathbf{1} - \xi \tag{3.109}$$

$$\xi \geq 0 \tag{3.110}$$

Taking the Lagrangian,

$$L(\mathbf{c_m}, \xi, b, \alpha, \zeta) = C\mathbf{1}^T\xi + \frac{1}{2}\mathbf{c_m}^T K_{mm}\mathbf{c_m} - \alpha^T(Y(K_{\ell m}\mathbf{c_m} + b) - 1 + \xi) - \zeta^T\xi. \quad (3.111)$$

Identically to the case of the standard SVM developed in Chapter 2, taking the derivatives with respect to b yields a constraint of the form $\sum_{i=1}^{\ell} y_i\alpha_i = 0$, and taking the derivative with respect to ξ yields $C\mathbf{1} - \xi - \zeta = 0$. We again write a reduced Lagrangian in terms of the remaining variables:

$$L^R(\mathbf{c_m}, \alpha) = \frac{1}{2}\mathbf{c_m}^T K_{mm}\mathbf{c_m} - \alpha^T(Y(K_{\ell m}\mathbf{c_m}) - 1) \quad (3.112)$$

Taking the derivative of L^R with respect to $\mathbf{c_m}$ and setting it equal to zero yields

$$K_{mm}\mathbf{c_m} = K_{m\ell}Y\alpha, \quad (3.113)$$

or, assuming K_{mm} is invertible (which is guaranteed to be the case, at least formally, if K is a positive definite kernel such as the Gaussian),

$$\mathbf{c_m} = K_{mm}^{-1}K_{m\ell}Y\alpha. \quad (3.114)$$

Substituting this into the reduced Lagrangian, we find that:

$$\begin{aligned} \frac{1}{2}\mathbf{c_m}^T K_{mm}\mathbf{c_m} - \alpha^T(Y(K_{\ell m}\mathbf{c_m}) - 1) &= \quad (3.115) \\ \frac{1}{2}\alpha^T Y K_{\ell m}K_{mm}^{-1}K_{mm}K_{mm}^{-1}K_{m\ell}Y\alpha - \alpha^T(Y(K_{\ell m}K_{mm}^{-1}K_{m\ell}Y\alpha)) + \alpha^T\mathbf{1} &= \quad (3.116) \\ -\frac{1}{2}\alpha^T Y K_{\ell m}K_{mm}^{-1}K_{m\ell}Y\alpha + \alpha^T\mathbf{1} & \quad (3.117) \end{aligned}$$

Adding in the constraints we obtained when we took the derivative of the Lagrangian with respect to b and ξ, we arrive at the following dual problem:

$$\max_{\alpha \in \mathbf{R}^\ell} \quad \sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2}\alpha^T\tilde{Q}\alpha \quad (3.118)$$

$$\text{subject to :} \quad \sum_{i=1}^{\ell} y_i\alpha_i = 0 \quad (3.119)$$

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, \ell \tag{3.120}$$

where we have defined $\tilde{Q} \equiv Y\tilde{K}Y \equiv YK_{\ell m}K_{mm}^{-1}K_{\ell m}Y$. We see that in this case, the quadratic programming problem we solve is identical to the one obtained if we simply use $\tilde{Q}$ in place of Q. However, in order to obtain the primal solution (which is actually the solution to our problem, as we have defined our desired function as $f(x) = \sum_{i \in m} c_i K(\mathbf{x}, x_i) + b$), instead of just setting $c_i = y_i \alpha_i$ as in the standard SVM, we need to solve

$$K_{mm}\mathbf{c_m} = K_{m\ell}Y\alpha. \tag{3.121}$$

In the context of active set methods such as SvmFu, this approach is not valuable. We never require the entire kernel matrix, and working with $\tilde{Q}$ is no less costly than working with Q directly. For an interior point method such as the one outlined by Fine and Scheinberg, this technique represents a promising modification to their approach. In the context of RLSC (see the next section), using the modified problem in which we require the coefficients to be zero at points not in the subset works much better (results in more accurate classifiers) than simply using $\tilde{K}$ directly. Additionally, we see that these two approaches solve the *same* dual problem, and that a modification of the relationship between the dual and primal is the difference between the two approaches. Therefore, in order for Fine and Scheinberg to modify their interior point SVM to take advantage of this technique, they only need to post process the α vector into the $\mathbf{c_m}$ vector — the rest of their code remains identical. At the time of this writing, we are in contact with Dr. Scheinberg, and are considering collaborating on a joint project.

The above developments highlight the key point that we must keep in mind the learning problem we are actually trying to solve. If we simply modify the dual, we are changing the underlying learning problem in an unpredictable way. Many researchers treat the dual problems as "basic". By treating the original primal learning problem as basic, we were able to modify the problem in a way that seems intuitively reasonable, and, as we shall see in the next section, gives good results.

3.7.3 Nonlinear RLSC Application: Image Classification

To test our ideas on nonlinear RLSC, we compare SVMs and various nonlinear RLSC approaches on two different datasets. The first dataset is the US Postal Service handwritten database, used in [122] and communicated to us by the authors, and referred to here as `usps`. This dataset consists of 7,291 training and 2,007 testing points; the task we consider is to discriminate images of the digit "4" from images of other digits. The training set contains 6,639 negative examples and 652 positive examples. The testing set contains 1,807 negative examples and 200 positive example. The second data set is a face recognition data set that has been used numerous times at the Center for Biological and Computational Learning at MIT [52, 2], referred to here as `faces`. The training set contains 2,429 faces and 4,548 non-faces. The testing set contains 472 faces and 23,573 non-faces.

There is an important difference between the two datasets. The `usps` dataset was divided into a training and test set at random. The `faces` dataset, on the other hand, was collected from several sources: the training faces were collected for [104], the training non-faces were collected for [52], and the testing data is a subset of the CMU Test Set #1 [93] selected according to procedures described in [52]. For this reason, it is plausible to speculate that the training and test portions of this dataset are not drawn from identical distributions, ignoring the obvious discrepancy caused by grossly varying proportions of the two classes in the training and test sets.

We first considered the question of the relative performance of SVMs and RLSC on each dataset taken as a whole. This involved storing and working with a matrix with as many entries as the size of the database squared. We stored the entries in the kernel matrix as floating points (four bytes per entry), so this required approximately 200 Megabytes of storage. Running Conjugate Gradient on the full problems was therefore resource intensive and time-consuming, particularly in comparison to the SVMs: the time required simply to compute and store the matrix is substantially more than the time required to completely solve the SVM (the SVM only ever requires a small fraction of the kernel matrix, corresponding to kernel products between pairs

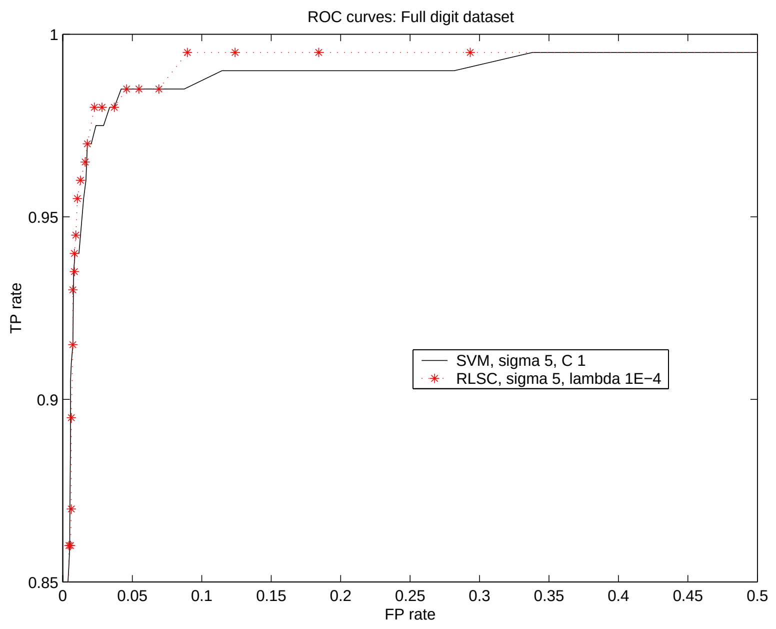


Figure 3-10: ROC curves for SVM and RLSC on the `usps` dataset.

of points which become support vectors at some time during the computation), and overall, RLSC on these problems takes more than an order of magnitude more time to train than an SVM. Additionally, the SVM solution is sparse, so RLSC functions will be much slower to test on new datapoints. However, for the moment, we are interested only in overall performance, ignoring issues of computational efficiency. Figure 3-10 shows the performance on the `usps` data. We first optimized the parameters (C and σ) for the SVM, then optimized the λ parameter of RLSC using the same σ ($\sigma = 5$). In Figure 3-10, we see that the full RLSC performed as well or better than the full SVM across the entire range of the ROC curve.

When we made a similar figure for the `faces` data (Figure 3-11), we found that the SVM performed substantially better than an RLSC with the same σ. Changing the σ from 5 to 2 for RLSC allowed RLSC to perform approximately as well as the SVM (performance for the SVM with $\sigma = 2$ was also quite similar, but slightly worse than, the SVM $\sigma = 5$ performance). This was surprising, in light of our intuitive experiments in Section 3.4 and the experimental results on real-world problems, which indicated that in general RLSC performed very well, and was not especially sensitive to parameter settings, so we decided to investigate this behavior using crossplots that compare the outputs of both SVM and RLSC on a point-by-point basis for both the training and testing sets. Figure 3-12 shows a crossplot for the `usps` data; here we find that the RLSC and SVM outputs are very well correlated, and the training and test crossplots look quite similar (the test crossplot is somewhat noisier). Figure 3-13 shows a crossplot for SVM with $\sigma = 2$ and RLSC with $\sigma = 2$, and Figure 3-14 shows a crossplot for SVM with $\sigma = 2$ and RLSC with $\sigma = 5$. These figures are in marked contrast to Figure 3-12. Whereas the `usps` dataset had very similar performances (for both algorithms) on the training and test set, for the `faces` dataset both algorithms perform much more poorly on the test set than the training set, even when tuned to have their best performance. In particular, none of the positive points attain an output value of $+1$ or greater for either algorithm, indicating that none of the test faces, if added to the training set, would be accepted by the SVM without changing the solution. This is not too surprising in light of the fact that the training and test sets were drawn from different sources; however, one might have expected that the test faces would look (to the algorithms) more like training faces than they do. The *relative* performance of SVM and RLSC remains somewhat mysterious, although a reasonable hypothesis from looking at the crossplots is that the choice of $\sigma = 5$ is forcing the function values to be spread out more, and in particular, bringing the two distributions closer together; however, it is not immediately clear why this translates into a lower test error. It is also worth remembering that the test set contains only 472 faces and 23,573 non-faces, so the differences between the ROC curves are not based on very many actual face examples. We also considered a modified version of the `faces` dataset, where we removed 25% of the training data and made it into a test set. A crossplot and ROC curves for this modified dataset are given in Figures 3-15 and 3-16 respectively. These figures show that both systems are learning the distribution represented by the *training data* extremely accurately without overfitting, and the difficulties are caused by the differences between the training and test distributions. Although it is tempting (based on this example) to speculate that SVMs are more robust to modification of the distribution than RLSC, it is impossible to justify this assumption without some model of how the distribution is being modified, which is

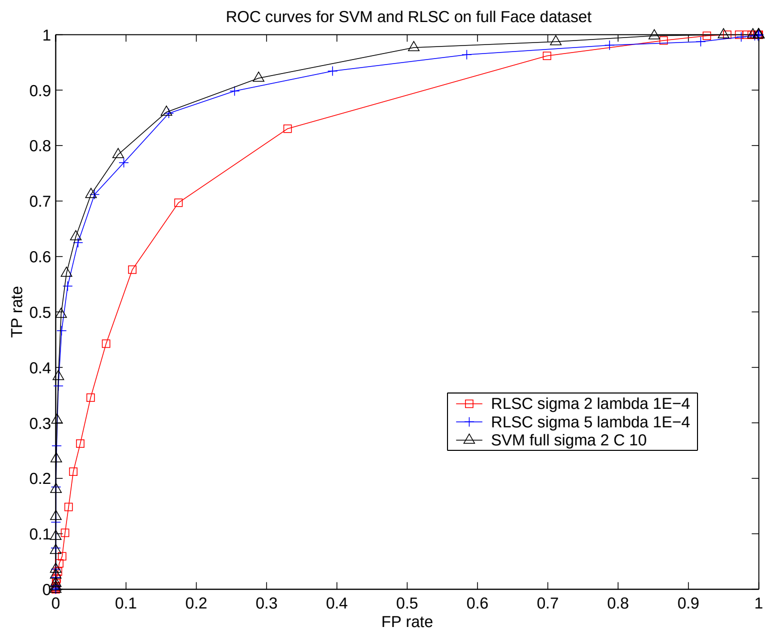


Figure 3-11: ROC curves for SVM and RLSC on the `faces` dataset.

beyond the scope of this thesis.

We have seen that running RLSC allows to recover solutions with very similar power to the SVM solution on both datasets (possibly using a different value of σ). We now turn to the question of whether an *approximation* to RLSC results can also produce results that are essentially equivalent to SVM. We consider three different approximation methods. The first method, which we call `subset`, involves merely selecting (at random) a subset of the points, and solving a reduced RLSC problem on only these points. The second method, which we call `rectangle`, is the primary algorithm discussed in Section 3.7.2: we choose a subset of the points, allow only those points to have nonzero coefficients in the function expansion, but minimize the loss over all points simultaneously. The third method is the `nystrom` method, also discussed briefly in Section 3.7.2 and presented more extensively in [122] (and in a slightly different context in [32]): in this method we choose a subset of the points, use those points to approximate the entire kernel matrix, and then solve the full problem using this approximate kernel matrix. In all experiments, we try four different subset sizes (1,024, 512, 256, and 128 for the `usps` dataset, and 1,000, 500, 250 and 125 for the `faces` dataset), and the results presented are the average of ten independent

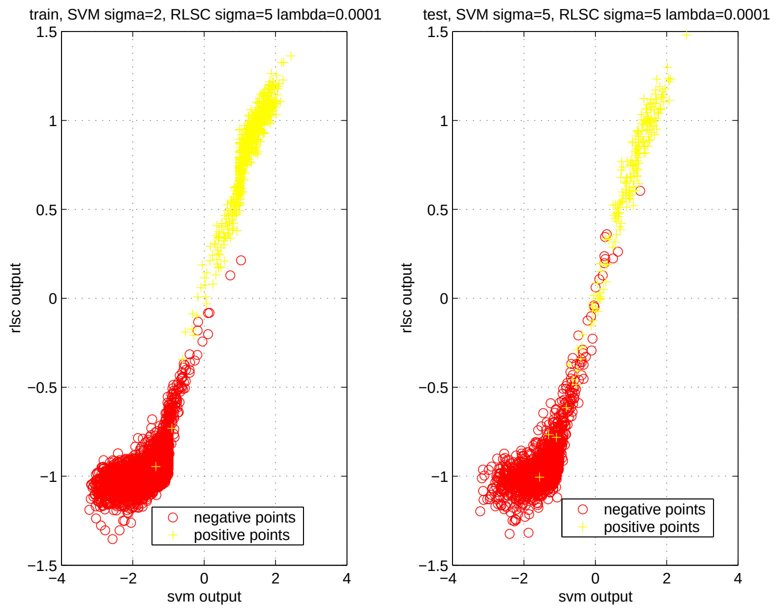


Figure 3-12: A crossplot for the `usps` dataset.

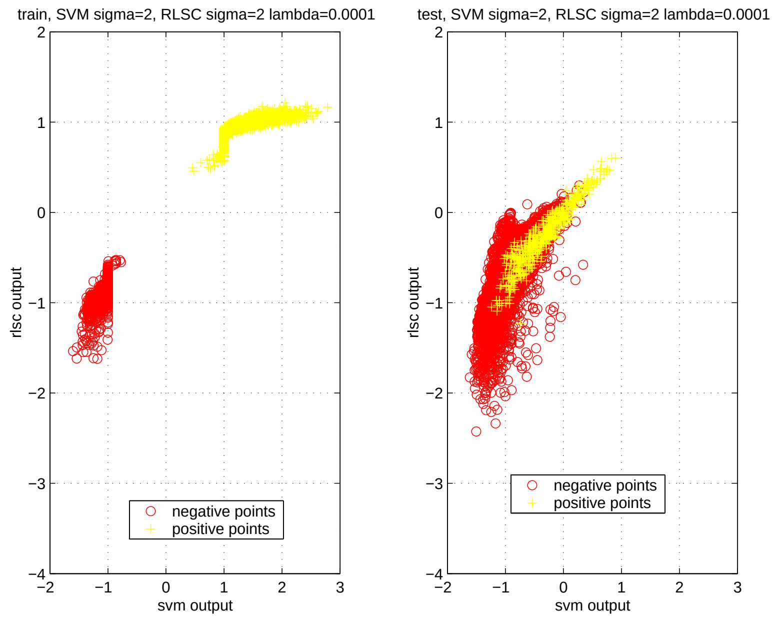


Figure 3-13: A crossplot for the `faces` dataset using $\sigma = 2$ for RLSC.

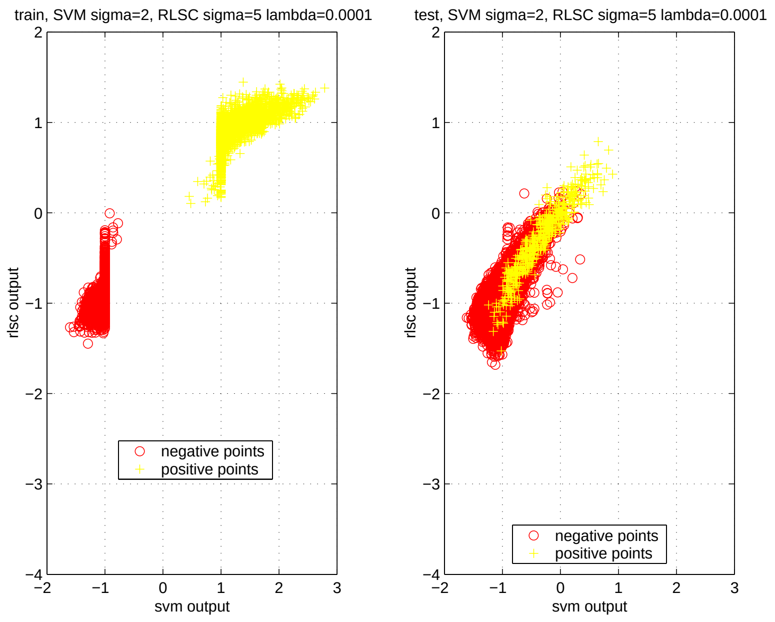


Figure 3-14: A crossplot for the `faces` dataset using $\sigma = 5$ for RLSC.

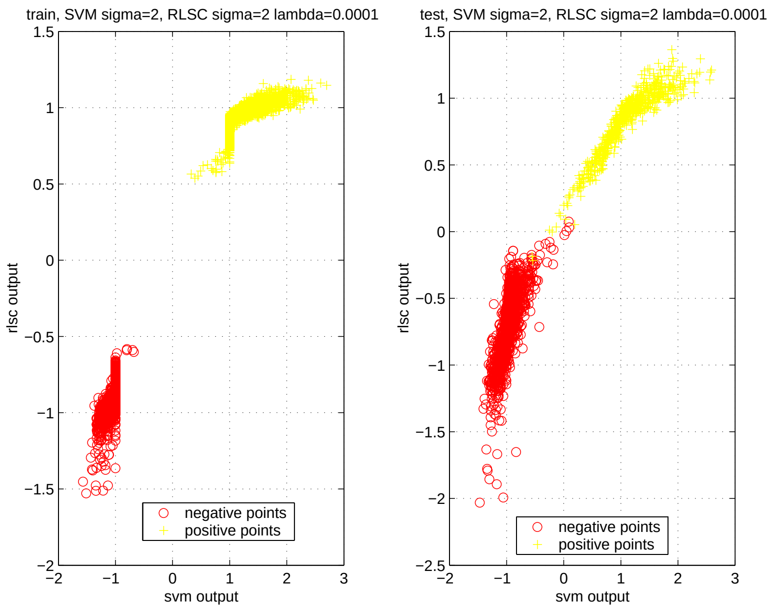


Figure 3-15: A crossplot for the modified `faces` dataset using $\sigma = 2$ for RLSC. The training data is 75% of the original training set (chosen at random), the test data is the remaining 25%. We see that the system is able to “learn” the pattern described by the training data, and generalize well to an independent test sample from the same distribution.

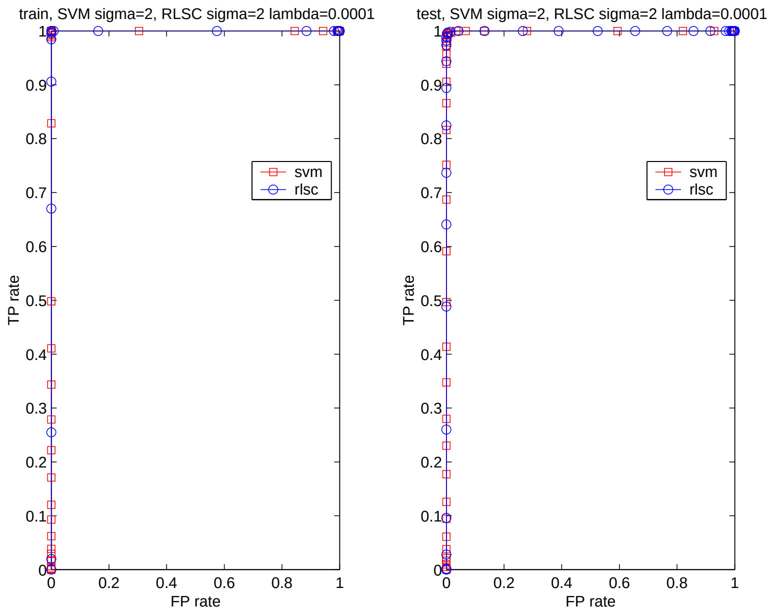


Figure 3-16: ROC curves for SVM and RLSC on the modified `faces` dataset using $\sigma = 2$ for RLSC. The training data is 75% of the original training set (chosen at random), the test data is the remaining 25%.

independent runs.

The results for the `usps` data set are given in Figure 3-17. We see that `rectangle` performs best, followed by `subset`, followed by `nystrom`. We suggest in passing that the extremely good results reported for `nystrom` in [122] may be a consequence of looking only at the error rate (no ROC curves are provided) for a problem with a highly skewed distribution (1,807 negative examples, 200 positive examples). For `rectangle` performance is very good at both the 1,024 and 512 sizes, but degrades noticeably for 256 samples. The results for the `faces` dataset, shown in Figure 3-18, paint a very similar picture. Note that the overall accuracy rates are much lower for this dataset, which contains many difficult test examples.

We also considered a bagged RLSC, in which we averaged the ten different functions derived in the previous experiment together into a single classifier. The results are given in Figure 3-19 for the `usps` dataset, and in Figure 3-20 for the `faces` dataset. In general, bagging improved performance for the weaker algorithms, particularly the `nystrom` method at small sample sizes. However, the best algorithms did not improve with bagging, and the relative strength of the algorithms was unchanged; if our goal

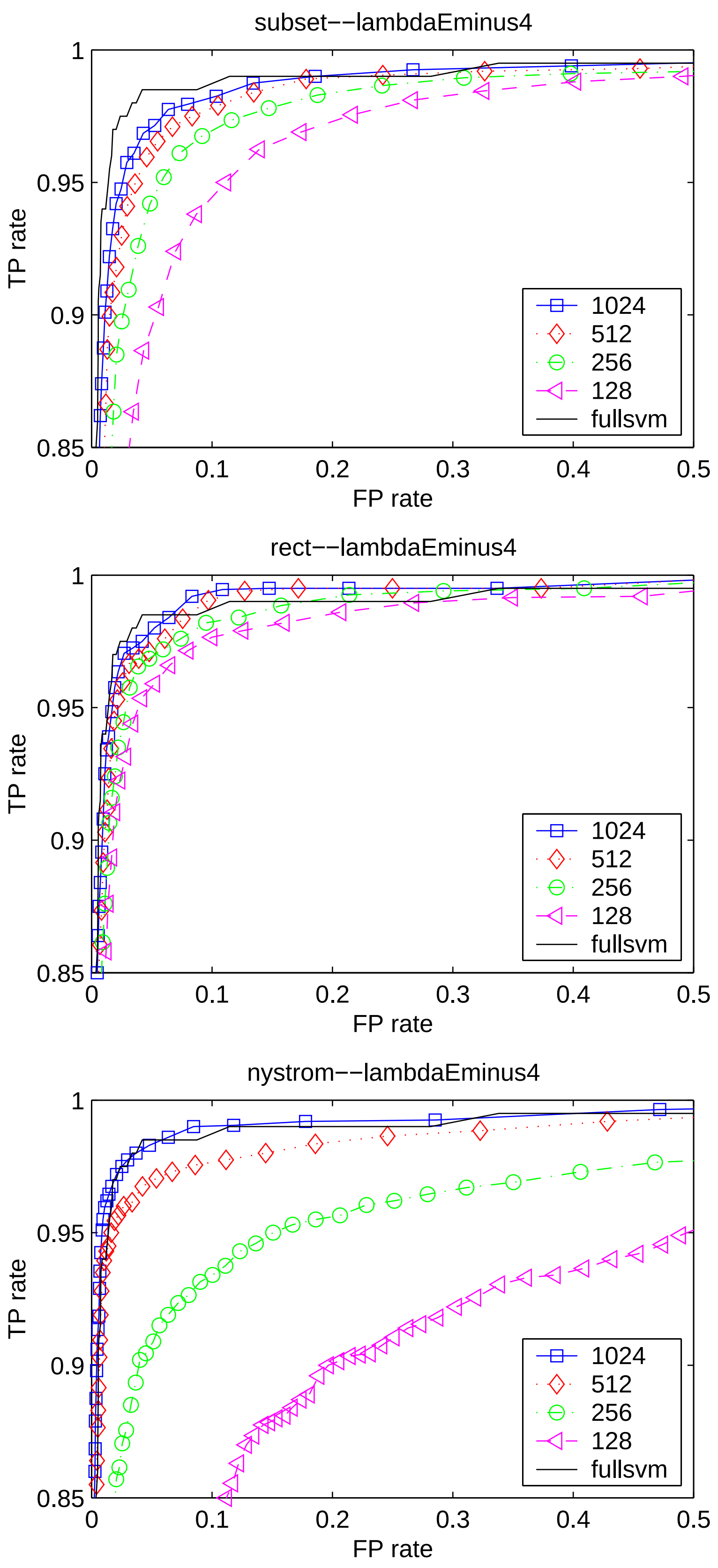


Figure 3-17: A comparison of the `subset`, `rectangle` and `nystrom` approximations to RLSC on the `usps` dataset. Both SVM and RLSC used $\sigma = 5$.

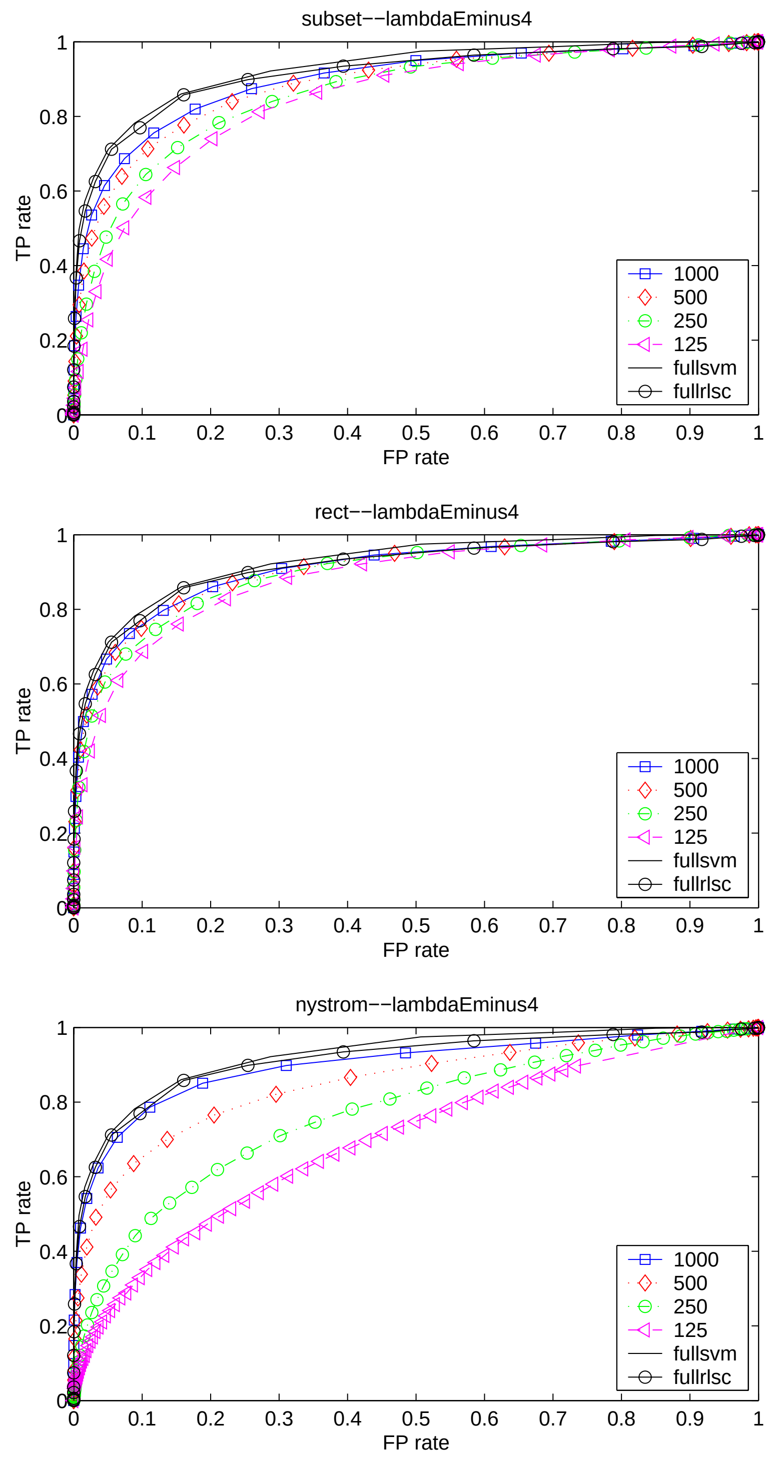


Figure 3-18: A comparison of the `subset`, `rectangle` and `nystrom` approximations to RLSC on the `faces` dataset. SVM used $\sigma = 5$, RLSC used $\sigma = 2$.

is to solve the problem as accurately as possible, it does not appear that bagging is helpful.

In all the experiments described so far, the subset of points was selected at random. Fine and Scheinberg [32] suggests selecting the points iteratively using an optimal greedy heuristic: at each point, the example is selected which will minimize the trace of the difference between the approximate matrix and the true matrix. Because the method simply amounts to running an incomplete Cholesky factorization for some number of steps (with pivoting at each step), we call this method `ic`. As mentioned earlier, if we believe that smaller values of $\mathrm{tr}(K - \tilde{K})$ are indicative of a better approximation, this method appears to produce a worse approximation than choosing the subset at random (see Figure 3-21 and Figure 3-22). As pointed out by Scheinberg in personal communication, a smaller value of $\mathrm{tr}(K - \tilde{K})$ is not necessarily indicative of a better matrix for learning, so we conducted experiments comparing `ic` to choosing a subset of the data at random randomly. Figures 3-23, 3-24, 3-25 show the results for the `usps` data, and Figures 3-26, 3-27, and 3-28 show the results for the `faces` data. In all these experiments, we compare the average performance of the ten classifiers trained on a random subset to the performance of a single classifier trained on the subset of the size obtained using the `ic` method. In general, it does not seem that selecting the "optimal" subset using the `ic` method improves performance on learning problems, although it seems more plausible that it is helping on the smallest subsets. One possible explanation of this phenomenon is that the best representation for reconstructing the kernel matrix is not the best representation for learning. Reconstructing the kernel matrix as accurately as possible with a Gaussian kernel will involve choosing many points which are far away from other points — training outliers will be the first points to be chosen. However, since we are interested in getting the bulk of the points right, at the expense of the outliers, this may not be a good strategy.

In summary, we have found that for nonlinear problems, full RLSC will perform about as well as an SVM, albeit possibly with different kernel settings. Furthermore, it seems that using the `rectangle` approximation to the full RLSC problem with a

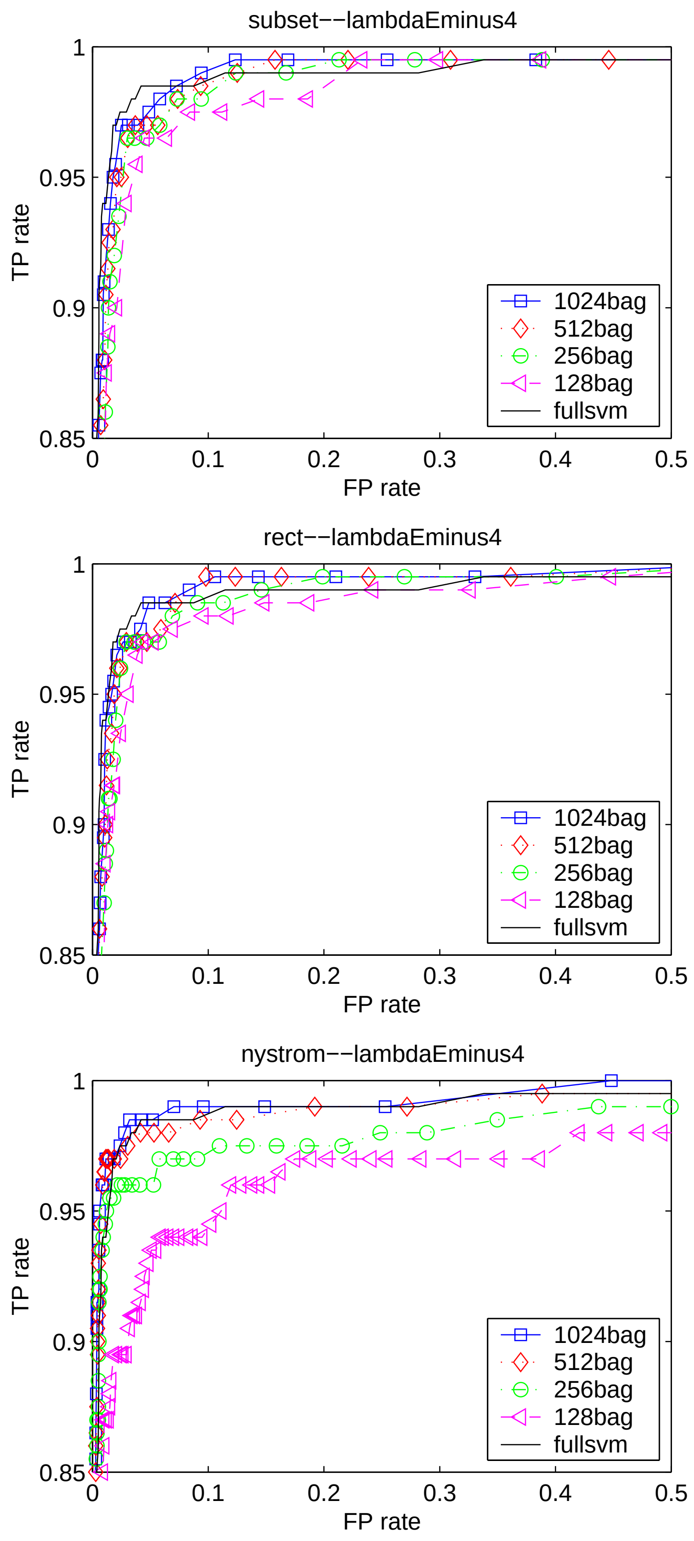


Figure 3-19: A comparison of bagged classifiers trained using the `subset`, `rectangle` and `nystrom` approximations to RLSC on the `usps` dataset. Both SVM and RLSC used $\sigma = 5$.

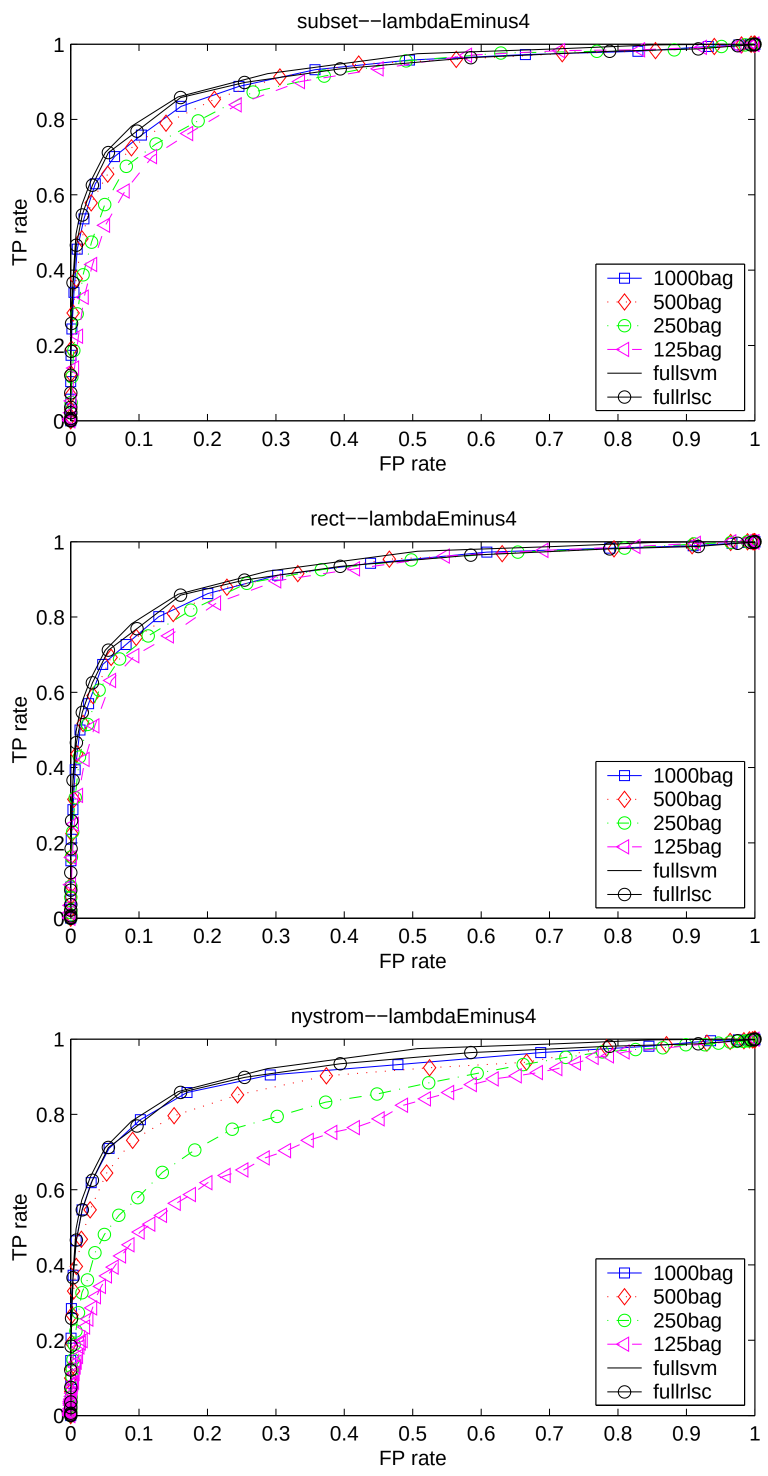


Figure 3-20: A comparison of bagged classifiers trained `subset`, `rectangle` and `nystrom` approximations to RLSC on the `faces` dataset. SVM used $\sigma = 5$, RLSC used $\sigma = 2$.

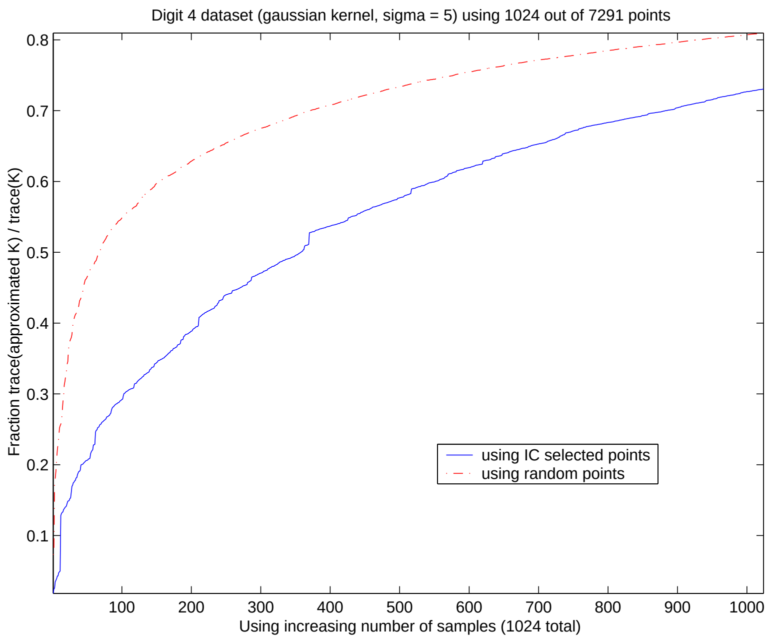


Figure 3-21: A comparison between the portion of the trace of K accounted for by using a subset of the points chosen using `ic` and a random subset, for the `usps` dataset.

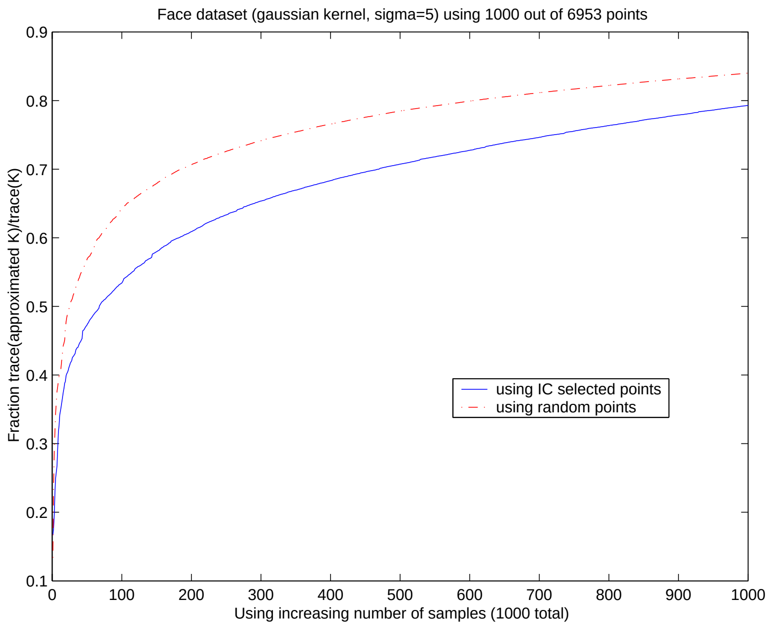


Figure 3-22: A comparison between the portion of the trace of K accounted for by using a subset of the points chosen using `ic` and a random subset, for the `faces` dataset.

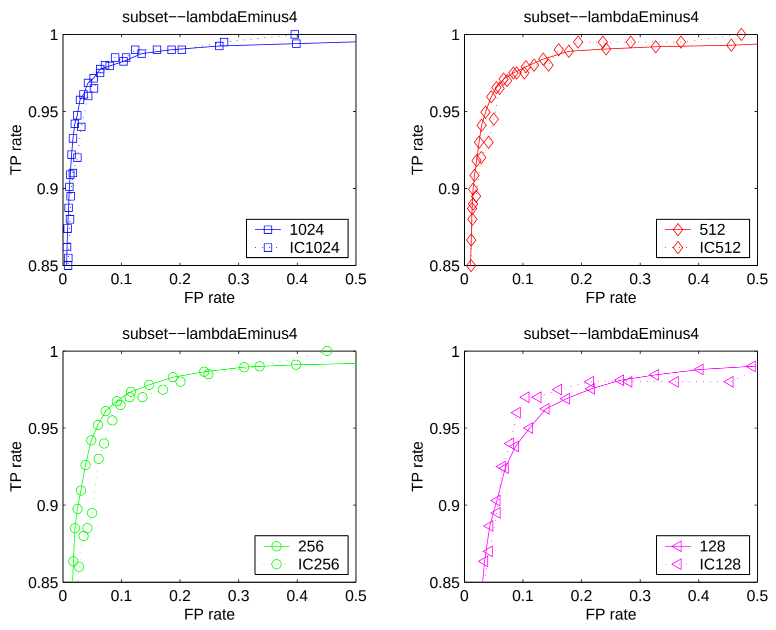


Figure 3-23: A comparison between random subset selection and the `ic` method, classifying the `usps` dataset using the `subset` method.

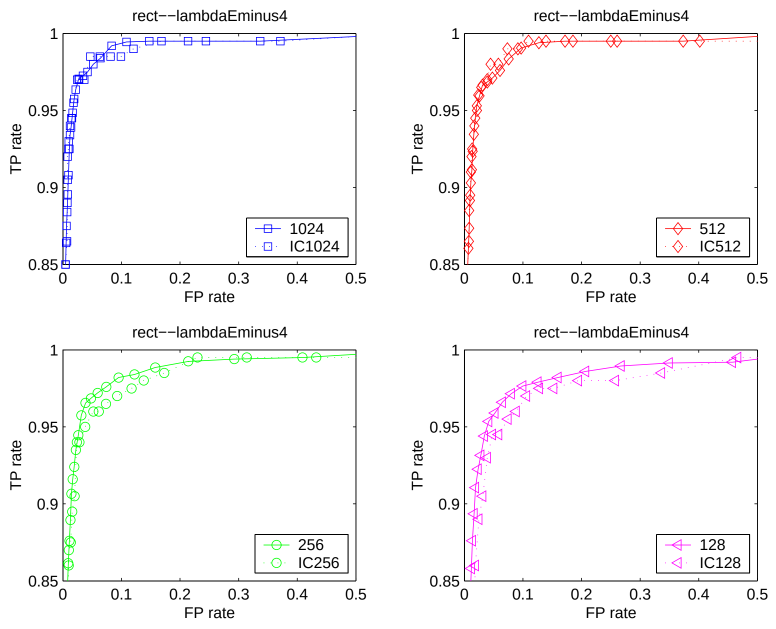


Figure 3-24: A comparison between random subset selection and the `ic` method, classifying the `usps` dataset using the `rectangle` method.

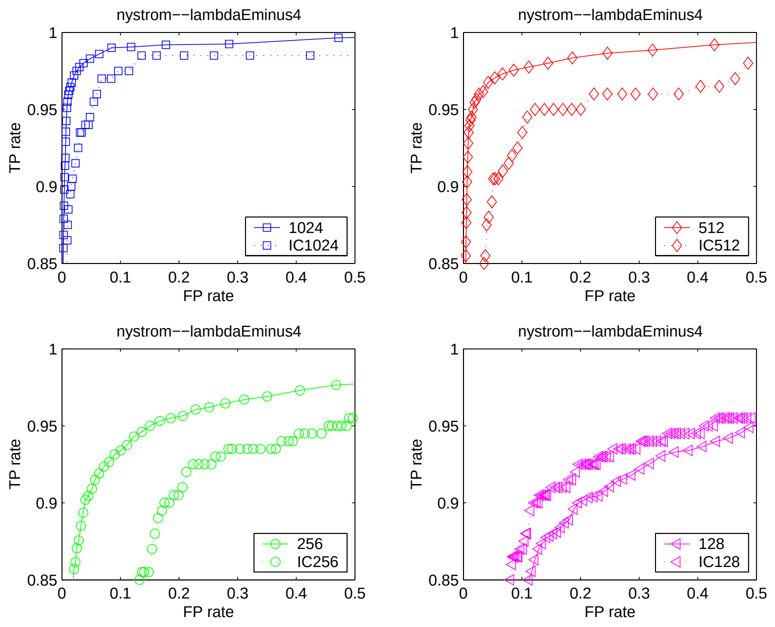


Figure 3-25: A comparison between random subset selection and the `ic` method, classifying the `usps` dataset using the `nystrom` method.

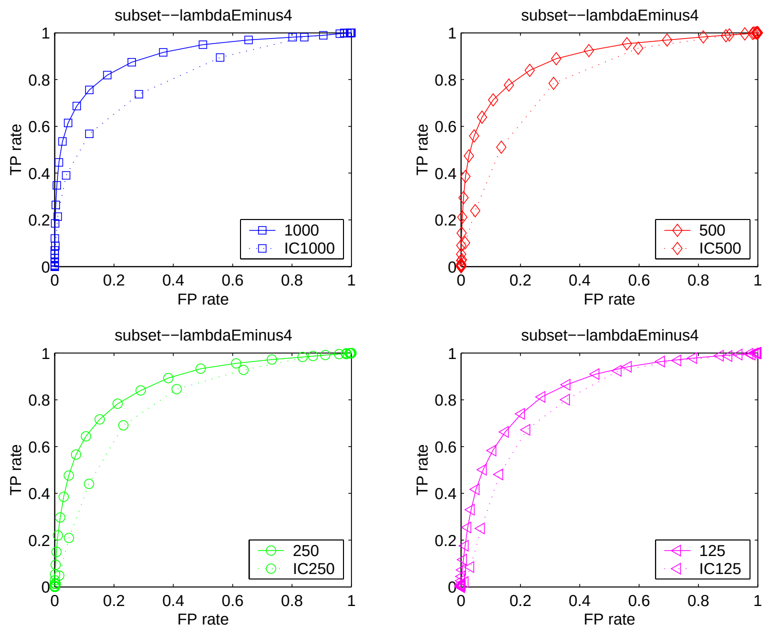


Figure 3-26: A comparison between random subset selection and the `ic` method, classifying the `faces` dataset using the `subset` method.

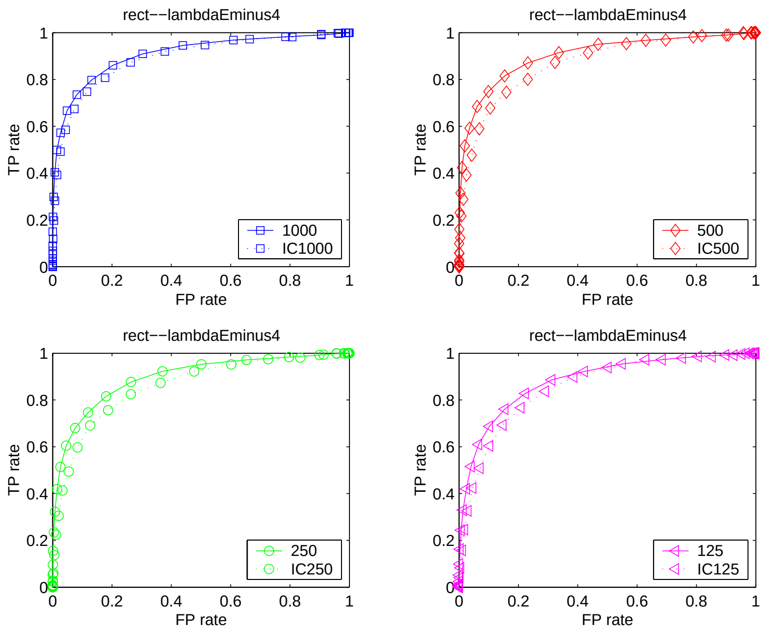


Figure 3-27: A comparison between random subset selection and the `ic` method, classifying the `faces` dataset using the `rectangle` method.

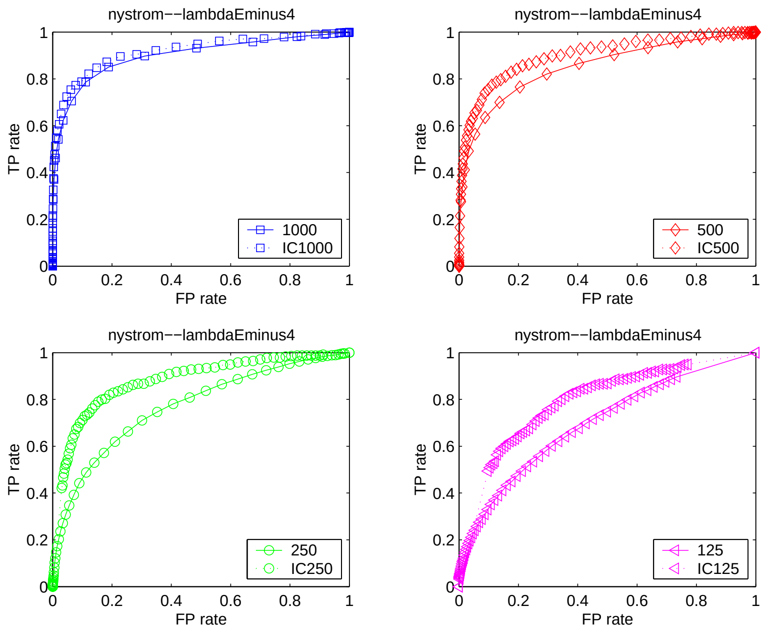


Figure 3-28: A comparison between random subset selection and the `ic` method, classifying the `faces` dataset using the `nystrom` method.

moderate sized subset will also produce very powerful results. We found that the `rectangle` method consistently produced better results than either the `subset` or `nystrom` methods. We found that bagging did not improve the performance (of the best methods), and that choosing the subset using `ic` rather than randomly did not improve performance.

If we approximate a dataset of size ℓ with a subset of size m, we decrease our storage requirements by a factor of $O(\frac{m}{\ell})$, and the time to run each iteration of Conjugate Gradient will be reduced from $O(\ell^2)$ to $O(2\ell m + m^2) = O(\ell m)$, so the computational requirements will also be reduced by a factor of $O(\frac{m}{\ell})$, assuming (reasonably) an approximately equal number of iterations are required for the original problem and the approximation. One possible advantage of a nonlinear approximate RLSC system over an SVM system is that for the former, it is possible to easily specify in advance the total amount of memory required to solve the problem. In contrast, the amount of memory required to effectively solve an SVM problem will depend on the number of support vectors, which is not known in advance. One can also easily control the total time commitment of RLSC by limiting the number of iterations of Conjugate Gradient; however, one may also (at greater difficulty) implement a similar computational cap on an SVM solution, stopping with whatever has been achieved at that point.

Chapter 4

Multiclass Classification

4.1 Introduction

The Support Vector Machine and Regularized Least Squares classifiers discussed in Chapters 2 and 3 are inherently binary classifiers: they classify an example as being positive or negative. In contrast, many problems we are interested in solving are multiclass problems — there are more than 2 classes, and our job is to pick the single class to which a data point belongs.[1] SVMs and RLSCs have been shown to perform very well for binary problems, and it is desirable to extend their capabilities into multiclass domains. In order to deal with multiclass problems, it is necessary to somehow adapt or combine the binary classifiers provided by SVMs or RLSCs.

Throughout this chapter, N refers to the number of classes in a multiclass classification problem. Given a data point $\mathbf{x}$, $y_{\mathbf{x}} \in \{1, \ldots, N\}$ refers to the class of x. Abusing notation slightly, given an index $i \in \{1, \ldots, \ell\}$, $y_i = y_{\mathbf{x}_i}$.

Probably the simplest multiclass classification scheme is to train N different binary classifiers, each one trained to distinguish the examples in a single class from the examples in all remaining classes. When it is desired to classify a new example, the N classifiers are run, and the classifier which outputs the largest (most positive)

[1]Note that we wish to distinguish this from the case when there are more than 2 classes, but a given example can be a member of more than one class simultaneously. In this case, the problem very naturally decomposes into N unlinked binary problems (N is the number of classes), where the ith binary learner simply learns to distinguish whether or not an example is in class i.

value is chosen. This scheme will be referred to as the "one-vs-all" or OVA scheme throughout this chapter. The one-vs-all scheme is conceptually simple, and has been independently invented numerous times by different researchers. One might well argue that OVA is the first thing one might think of when asked to come up with an approach for combining binary classifiers to solve multiclass problems. Although it is simple and obvious, a primary thesis of this chapter is that the OVA scheme is extremely powerful, producing results that are often as or more accurate than other methods. This thesis seems to be somewhat in opposition to a large body of recent literature on multiclass classification, in which a number of more complicated methods have been developed and their superiority over OVA claimed. For this reason, a substantial portion of this chapter shall be spent reviewing and discussing this literature in detail. What we will find is that although this literature develops a wide array of more sophisticated methods for multiclass classification, experimental evidence of the superiority of these methods is either lacking or sloppy.

One reason for preferring the OVA approach is its implementational simplicity and relatively fast running time. Training an OVA-system simply requires training N binary systems, each the size of the data set. For RLSC using Conjugate Gradient, where the running time is relatively independent of the labelling, this implies that training an OVA system will take approximately N times as long as training a single classifier; furthermore, this work can be trivially parallelized N ways on N machines. In contrast, many of the methods discussed below require training far more than N classifiers on the full data set, or solving a single quadratic program with $O(\ell N)$ variables. Because solving a quadratic program generally takes time superlinear in the number of variables, these methods represent a much larger computational burden than OVA schemes. Furthermore, many of these methods require sophisticated software which is not commonly available (such as in-house implementations not released by the researchers who have published) in order to perform well. Therefore, there is reason to prefer an OVA system unless a different scheme offers some compelling advantage in performance.

In the majority of cases discussed below, the authors present a sophisticated method but offer no substantial experimental evidence to help judge whether the method is useful or not. In those cases where they do, we often find that the evidence is grossly overstated due to sloppy experimental conditions. Unfortunately, there are numerous difficulties associated with trying to decide whether two different techniques are equally good, and there is no simple statistical test that will answer this question. Additionally, in order to truly be useful, a test would not have to simply say whether or not two systems were equally good, but if they were different, would have to quantify this difference: if I have to use one of two systems, and I know that one approach will *always* perform .001% better, I will choose the approach which is easier to implement. Many authors choose to take the opposite approach, building a table with one column per method and one row per dataset, putting the best result for each data set in bold, and declaring as "best" the classifier with the most boldface entries in its column, ignoring the fact that the differences are often miniscule. This is especially troublesome considering that the differences involved are often smaller than those associated with variations in parameters that are generally not even reported, such as the particular choice of tolerance for the optimality conditions in an algorithm.

Our main interest is to help the reader choose a system for multiclass classification which will be both fast to train on large data sets and have a high accuracy relative to other available choices, and our claim is that using SVMs or RLSC in a simple OVA scheme meets this goal. In Section 4.2, we will discuss in great detail alternative approaches to multiclass classification, showing that in all these papers, the authors either do not provide experimental evidence as to the value of their method or that the experimental work is sloppy. In Section 4.3, we will bolster our claim by presenting the result of our own experiments on applying multiclass methods to real-world data sets; some of these results were also discussed in the context of RLSC in Chapter 3. Finally, in Section 4.4, we will prove leave-one-out bounds for multiclass classification, provide some theoretical and intuitive arguments as to why error-correcting code approaches seem unlikely to improve on the performance of the simple one-vs-all scheme.

4.2 Previous Work

The central thesis of this chapter is that one-vs-all classification using SVMs or RLSC is an excellent choice for multiclass classification. In the past few years, many papers have been presented that claim to represent an advance on this technique. We will review these papers in detail, pointing out specifics of these papers in order to demonstrate the weakness of their claims.

The papers claiming to offer more advanced techniques for multiclass classification fall into two main categories. The first category attempts to solve a single optimization problem rather than combine the solutions to a collection of binary problems. The second category attempts to use the power of error correcting codes to improve multiclass classification. We deal with these two approaches separately.

4.2.1 Single Machine Approaches

Vapnik, Weston and Watkins

The single machine approach was introduced simultaneously in the 1998 book of Vapnik [116] and a technical report by Weston and Watkins [121]. The formulations introduced in these two sources are essentially identical. The approach is a multiclass generalization of Support Vector Machines. A standard SVM finds a function

$$f(x) = \sum_{j=1}^{\ell} c_j K(x, x_j) + b \tag{4.1}$$

The multiclass SVM of Weston and Watkins finds N functions $f_1, \ldots, f_N$ simultaneously, where

$$f_i(x) = \sum_{j=1}^{\ell} c_{ij} K(x, x_j) + b_i. \tag{4.2}$$

The basic idea behind the multiclass support vector machine of Weston and Watkins (as well as all other single machine approaches, with slight modifications, as we shall see) is that instead of paying a penalty for each machine separately based on whether each machine satisfies its margin requirements for a given point, we pay a penalty based on the *relative* values output by the different machines. More concretely, given a single data point x belonging to class i, in the one-vs-all scheme we pay a penalty for machine i if $f_i(x) < 1$, and for all other classes j we pay a penalty if $f_j(x) > -1$. In the Weston and Watkins scheme, for each pair $i \neq j$, we pay a penalty if $f_i(x) < f_j(x)+2$. If $f_i(x) < 1$, we may not pay a penalty, as long as $f_j(x)$ is sufficiently small for $i \neq j$; similarly, if $f_j(x) > 1$, we will not pay a penalty for x is $f_i(x)$ is sufficiently large. To facilitate this, we will use $\ell(N-1)$ slack variables ξ_{ij}, where $i \in \{1, \ldots, \ell\}$ and $j \in \{1, \ldots, N\} \backslash y_i$. Using these slack variables, the optimization problem being solved can be expressed using our notation as

$$\min_{\mathbf{f_1},\ldots,\mathbf{f_N} \in \mathcal{H}, \xi \in \mathbf{R}^{\ell(\mathbf{N-1})}} \quad \textstyle\sum_{i=1}^{N} ||f_i||_K^2 + C \sum_{i=1}^{\ell} \sum_{j \neq y_i} \xi_{ij} \tag{4.3}$$

$$\text{subject to}: \quad f_{y_i}(\mathbf{x_i}) + b_{y_i} \geq f_j(\mathbf{x_i}) + b_j + 2 - \xi_{ij} \tag{4.4}$$

$$\xi_{ij} \geq 0 \tag{4.5}$$

where the constraints all run over $i \in \{1, \ldots, \ell\}$ and $j \in \{1, \ldots, N\} \backslash y_i$. As we did for SVMs and RLSCs in Chapters 2 and 3, we can write for each f_i

$$||f_i||_K^2 = \mathbf{c_{i,\cdot}}^T K \mathbf{c_{i,\cdot}}, \tag{4.6}$$

where $\mathbf{c_{i,\cdot}}$ is the vector whose jth entry is c_{ij}. Doing so leads to a single quadratic programming problem with $N\ell$ function defining variables c_{ij}, $(N-1)\ell$ slack variables ξ_{ij} variables, and N bias terms b_i. The dual of this problem can be taken using the standard Lagrangian approach. Weston and Watkins define α_{ij} to be the dual variables associated with the first set of constraints (including "dummy" variables α_{i,y_i}), and β_{ij} to be the dual variables associated with the second set of constraints. Introducing the notation

$$A_i = \sum_{j=1}^{N} \alpha_{ij}, \tag{4.7}$$

and defining $c_{ij} = 1$ if $y_i = j$ and 0 otherwise. Skipping the intermediate algebra, the dual problem derived by Weston and Watkins is

$$\max_{\alpha \in \mathbf{R}^{\ell N}} \quad 2\sum_{ij} \alpha_{ij} + \sum_{i,j,k} \left[-\tfrac{1}{2} c_{j,y_i} A_i A_j + \alpha_{i,k} \alpha j, y_i - \tfrac{1}{2} \alpha_{i,k} \alpha_{j,k} \right] K(\mathbf{x_i}, \mathbf{x_j}) \tag{4.8}$$

$$\text{subject to :} \quad \sum_{i=1}^{\ell} \alpha_{ij} = \sum_{i=1}^{\ell} c_{ij} A_i \tag{4.9}$$

$$0 \le \alpha_{ij} \le C \tag{4.10}$$

$$\alpha_{i,y_i} = 0 \tag{4.11}$$

The first set of constraints holds for $j \in \{1, \ldots, N\}$, the second over $i \in \{1, \ldots, \ell\}$ and $j \in \{1, \ldots, N\}$, and the third over $i \in \{1, \ldots, \ell\}$.

It is not clear whether this is useful or not, as it is unknown whether the resulting dual problem can be decomposed in any useful way. Weston and Watkins mention in passing that "decomposition techniques can be used, as in the usual SV case," but provide no mathematical derivation or implementation. Unlike the SVM, which has box constraints and a single equality constraint over all the variables, this system has N equality constraints, where the equality constraint for class j involves $\ell + JN$ terms, where J is the number of points in class j. The relative complexity of the constraints makes it likely that the decomposition algorithm would have to be substantially more complicated to maintain feasibility of the generated solutions at each iteration. Also, unlike the SVM scenario, the "dual" problem does not succeed in fully eliminating the primal variables c_{ij}.

Weston and Watkins take the dual for a different reason — they appear to be unaware of how to use kernels directly in the primal formulation, and write the primal formulation only for the linear case, directly in terms of the N hyperplanes $\mathbf{w_1}, \ldots, \mathbf{w_N}$. They then proceed through several pages of rather tedious algebra to derive a dual problem, and note that in the dual problem, only the dot products of training points appear, and that these dot products $\mathbf{x_i} \cdot \mathbf{x_j}$ can be replaced with kernel products $K(\mathbf{x_i}, \mathbf{x_j})$.

Weston and Watkins perform two different sorts of experiments. In the first set of experiments, they work with toy examples where several classes in the plane are classified using their algorithm. They show some examples which are both separable and nonseparable, but they do not compare their algorithm to any other method, so these experiments only serve as a proof of concept that the algorithm works in some vaguely reasonable way. Since any good solution to the one-vs-all problem is automatically a good solution to this problem, this is fairly obvious.

In the second set of experiments, they compare their algorithm to a one-vs-all scheme on five data sets from the UCI repository [77] (the data sets used were `iris`, `wine`, `glass`, `soy`, and `vowel`). They find that on two of the five data sets, the multiclass support vector machine performs substantially better, and on the remaining three, the performance is about the same. However, they state that "to enable comparison, for each algorithm $C = \infty$ was chosen (the training data must be classified without error)," implying that they are simply performing empirical risk minimization rather than regularization, making the formulations extremely difficult to compare. They also note that the total number of support vectors appearing in the one-vs-all scheme is more than in the single machine scheme, but they do not compare the total number of support vectors corresponding to different data points, so it is also impossible to draw any conclusions about which formulation is faster at testing new points.

Lee, Lin and Wahba

In [66] and [67], Lee, Lin and Wahba present a substantially different single-machine approach to multiclass classification. The work has its roots in an earlier paper by Lin [71] on the asymptotic properties of Support Vector Machine regularization for binary classification. If we define $p(\mathbf{x})$ to be the probability that a data point located at $\mathbf{x}$ is in class 1, Lin proved using elegant elementary arguments that the minimizer of $E[(1 - yf(\mathbf{x})_+)]$ is $f(\mathbf{x}) = \text{sign}(p(\mathbf{x}) - \frac{1}{2})$. In other words, if we consider solving an SVM problem and let the number of data points tend to infinity, the minimizer of the loss functional (ignoring the regularization term $\lambda||f||_K^2$, and the fact that the functional f has to live in the RKHS $\mathcal{H}_K$) tends to $\text{sign}(p(\mathbf{x}) - \frac{1}{2})$. Lee refers to this function as the Bayes-optimal solution.

Considering this to be a useful property, they set out to design a multiclass classification technique with similar behavior. They begin by noting that a standard one-vs-all SVM approach does not have this property. In particular, defining $p_i(\mathbf{x})$ to be the probability that a point located at $\mathbf{x}$ belongs to class i, the results of Lin show that $f_i(\mathbf{x}) \rightarrow \text{sign}(p_i(\mathbf{x}) - \frac{1}{2})$ as $\ell \rightarrow \infty$. For all points for which $\arg\max_i p_i(\mathbf{x}) \geq \frac{1}{2}$, we will recover the correct result: asymptotically, $f_i(\mathbf{x}) = 1$, and $f_j(\mathbf{x}) = -1$ for $j \neq i$. However, if $\arg\max_i p_i(\mathbf{x}) < \frac{1}{2}$, then asymptotically, $f_i(\mathbf{x}) = -1\ \forall i$, and we will be unable to recover the correct class. They note that for other formulations such as the one of Weston and Watkins [121], the asymptotic behavior is hard to analyze.

Lee, Lin and Wahba proceed to derive a multiclass formulation with the desired correct asymptotic behavior. For $1 \leq i \leq N$, they define v_i to be an N dimensional vector with a 1 in the ith coordinate and $\frac{1}{N-1}$ elsewhere.[2] The v_i vector plays the role of a "target" for points in class i — we try to get function outputs that are very close to the entries of v_i. However, for technical reasons, instead of worrying about all N functions, they only worry about $f_j(\mathbf{x_i})$ for $j \neq y_{\mathbf{x}_i}$, and ensure (approximate) correctness of $f_i(\mathbf{x_i})$ by requiring that for all $\mathbf{x}$, $\sum_{i=1}^N f_i(\mathbf{x}) = 0$. This leads to the following optimization problem:

$$\min_{f_1,\ldots,f_N \in \mathcal{H}_K} \quad \frac{1}{\ell}\sum_{i=1}^{\ell}\sum_{j=1,j\neq y_i}^{N}(f_j(\mathbf{x}_i) + \tfrac{1}{N-1})_+ + \lambda\sum_{j=1}^{C}||f_j||_K^2 \qquad (4.12)$$

$$\text{subject to}: \quad \sum_{j=1}^{C} f_j(\mathbf{x}) = 0 \qquad \forall\mathbf{x} \qquad (4.13)$$

Using arguments along the same lines of those in [71], it is shown that the asymptotic solution to this regularization problem (again ignoring the λ term and the fact that the functions must live in the RKHS) is $f_i(\mathbf{x}) = 1$ if $i = \arg\max_{j=1,\ldots,N} p_j(\mathbf{x})$ and $f_i(\mathbf{x}) = -\frac{1}{N-1}$ otherwise; $f_i(\mathbf{x})$ is one if and only if i is the most likely class for a point located at $\mathbf{x}$. They point out that this is a natural generalization of binary SVMs, if we view binary SVMs as producing two functions, one for each class, constrained so that $f_1(\mathbf{x}) = f_{-1}(\mathbf{x})$ for all $\mathbf{x}$. They also prove a variant of the Representer

[2]We have taken some liberties with the notation in order to shorten the presentation and keep notation consistent with the rest of the thesis.

Theorem (the "standard" Representer Theorem is discussed in Appendix B), which is instrumental in showing that the above mathematical optimization problem can be (reasonably) feasibly solved. A Lagrangian dual is derived, and it is noted that the approach retains some of the sparsity properties of binary SVMs. The resulting optimization problem is approximately $N-1$ times as large as a single SVM problem, and no decomposition method is provided.

Although this approach is interesting, there are a number of problems with it. The primary difficulty is that the analysis is entirely asymptotic, holding only as the number of data points goes to infinity and the regularization term is ignored. In this framework, any method which asymptotically estimates densities accurately will also perform optimally. However, such density estimation methods have been shown to be grossly inferior to discriminative methods such as SVMs in real-world classification tasks using limited amounts of data. Therefore, it is difficult to argue the superiority of a method based only on its asymptotic behavior. In the Lee, Lin and Wahba analysis, no information is provided about the rate of convergence to the Bayes-optimal solution. In order to arrive at this Bayes-optimal solution, we must also let $\lambda \to 0$ and $\ell \to \infty$; although this is a reasonable requirement, no information about rates is provided. Additionally, comparing this method to one-vs-all SVMs, the only points $\mathbf{x}$ for which this approach (asymptotically) makes a difference are points for which $\arg\max_i p_i(\mathbf{x}) < \frac{1}{2}$. In other words, if a single class is more than 50% likely at a given $\mathbf{x}$, this approach and the computationally much simpler one-vs-all approach will make the same prediction (asymptotically). We expect this to be the case for the vast majority of the probability mass of many real-world problems, although this is an intuition rather than a known fact.

If the class densities are highly overlapping in some region (one class is only slightly more likely than all the others), there are two additional problems. The first is that classification accuracy is inherently limited to the likelihood of the most likely class, indicating that our problem is too difficult to be usefully solved or that we have not represented or measured our data in a manner that allows good classification. There may be exceptions to this, such as problems involving financial data (which are notoriously hard to achieve high rates of accuracy on), but in general, we are most interested in problems which can be solved quite accurately. The second difficulty is that in high dimensions, if two class densities are similar over a region, we expect that we will need a large number of points (exponentially many in the dimensionality) to accurately capture this distinction.

Another intimately related problem with this approach is that it ignores the fundamental goal of regularization theory, which is the attempt to find *smooth* functions that fit the data accurately. Although the optimization problem suggested does include a regularization term, the analysis of the technique is completely dependent on ignoring the regularization term. The Bayes-optimal asymptotic solution can be arbitrarily non-smooth, and convergence to it relies on the use of an RKHS that is dense in L_2 (such as the one obtained when the kernel K is Gaussian).

Two toy examples illustrating the method are presented. In one example, the method is illustrated graphically, and no comparisons to other methods are made. In the other example, a comparison to one-vs-all is made. The training data consists of 200 one-dimensional points (in the interval $[0, 1]$) from three overlapping classes, and the test data consists of $10,000$ independent test points from the distribution. The distributions are chosen so that class 2 never has a conditional probability of more than 50%. In the example, the method of Lee, Lin and Wahba is able to predict class 2 over the region where it is more likely than any other class, and a one-vs-all system is not. On the test set, the one-vs-all system has an error rate of .4243 and the Lee, Lin and Wahba method has an error of .389. However, it is difficult to understand how the parameter settings were chosen — it is mentioned (and the equations given support this hypothesis) that they were selected to optimize the performance of the Lee, Lin and Wahba method, and that they were selected using the characteristic of the test distribution (for (not very much) more information see [66]), possibly indicating that different parameter settings would help the one-vs-all system. Nevertheless, this experiment is somewhat interesting, and it would be good to see a number of better-controlled experiments on more realistic data sets.

Some additional insights can be gained from taking another look at the original

Lin paper [71]. The Lee, Lin and Wahba paper was based on Lin's results for the SVM hinge loss function: $V(f(\mathbf{x}), y) = (1 - yf(\mathbf{x}))_+$. The Lin paper also includes easily proved results stating that for any $q > 1$, if the loss function is either $(1 - yf(\mathbf{x})_+)^q$ or $|y - f(\mathbf{x})|^q$, then the asymptotic minimizer is given by (recalling that $p(\mathbf{x})$ is the conditional probability of a point at $\mathbf{x}$ being in class 1):

$$f(\mathbf{x}) = \frac{(p(\mathbf{x}))^{\frac{1}{q-1}} - (1 - p(\mathbf{x}))^{\frac{1}{q-1}}}{(p(\mathbf{x}))^{\frac{1}{q-1}} + (1 - p(\mathbf{x}))^{\frac{1}{q-1}}} \tag{4.14}$$

In the specific case of Regularized Least Squares Classification discuss in Chapter 3, $V(f(\mathbf{x}, y)) = (y - f(\mathbf{x}))^2$, so the asymptotic discrimination function is

$$f(\mathbf{x}) = \frac{(p(\mathbf{x}))^{\frac{1}{2}} - (1 - p(\mathbf{x}))^{\frac{1}{2}}}{(p(\mathbf{x}))^{\frac{1}{2}} + (1 - p(\mathbf{x}))^{\frac{1}{2}}} \tag{4.15}$$

Now, instead of SVM, let's consider the use of RLSC in a one-vs-all framework. We will (asymptotically) arrive at N functions, where

$$f_i(\mathbf{x}) = \frac{(p_i(\mathbf{x}))^{\frac{1}{2}} - (1 - p_i(\mathbf{x}))^{\frac{1}{2}}}{(p_i(\mathbf{x}))^{\frac{1}{2}} + (1 - p_i(\mathbf{x}))^{\frac{1}{2}}} \tag{4.16}$$

Now assume $p_i(\mathbf{x}) > p_j(\mathbf{x})$. We will show that this implies that $f_i(\mathbf{x}) > f_j(\mathbf{x})$. Specifically, we consider the notationally simplified quantity

$$R(p) = \frac{p^{\frac{1}{2}} - (1 - p)^{\frac{1}{2}}}{p^{\frac{1}{2}} + (1 - p)^{\frac{1}{2}}}, \tag{4.17}$$

and show that $R(p)$ is increasing as a function of p, for $p \in [0, 1]$. We first note that $R(0) = -1$ and $R(1) = 1$. Next, for $p \in (0, 1)$, we find that

$$\frac{dR}{dp} \tag{4.18}$$

$$= \frac{\frac{d(p^{\frac{1}{2}} - (1-p)^{\frac{1}{2}})}{dp}(p^{\frac{1}{2}} + (1 - p)^{\frac{1}{2}}) - (p^{\frac{1}{2}} - (1 - p)^{\frac{1}{2}})\frac{d(p^{\frac{1}{2}} + (1-p)^{\frac{1}{2}})}{dp}}{(p^{\frac{1}{2}} + (1 - p)^{\frac{1}{2}})^2} \tag{4.19}$$

$$= \frac{\frac{1}{2}\left[(p^{-\frac{1}{2}} + (1 - p)^{-\frac{1}{2}})(p^{\frac{1}{2}} + (1 - p)^{\frac{1}{2}})\right]}{(p^{\frac{1}{2}} + (1 - p)^{\frac{1}{2}})^2} \tag{4.20}$$

$$
\begin{aligned}
&+ \quad \frac{\frac{1}{2}\left[(p^{\frac{1}{2}}-(1-p)^{\frac{1}{2}})(p^{-\frac{1}{2}}-(1-p)^{-\frac{1}{2}})\right]}{(p^{\frac{1}{2}}+(1-p)^{\frac{1}{2}})^2} && (4.21)\\
&= \quad \frac{\frac{1}{2}\left[1+(\frac{1-p}{p})^{\frac{1}{2}}+(\frac{p}{1-p})^{\frac{1}{2}}+1-1+(\frac{1-p}{p})^{\frac{1}{2}}+(\frac{p}{1-p})^{\frac{1}{2}}-1\right]}{(p^{\frac{1}{2}}+(1-p)^{\frac{1}{2}})^2} && (4.22)\\
&= \quad \frac{(\frac{1-p}{p})^{\frac{1}{2}}+(\frac{p}{1-p})^{\frac{1}{2}}}{(p^{\frac{1}{2}}+(1-p)^{\frac{1}{2}})^2} && (4.23)\\
&> \quad 0. && (4.24)
\end{aligned}
$$

In other words, $R(p)$ is a strictly increasing function of p (Figure 4-1 shows both $R(p)$ and $\frac{dR}{dP}$ as a function of p), which implies that if class i is more likely than class j at point $\mathbf{x}$, $f_i(\mathbf{x}) > f_j(\mathbf{x})$. This in turn implies that if we use a one-vs-all RLSC scheme, and classify test points using the function with the largest output value (which is of course the common procedure in one-vs-all classification), the error of our scheme will asymptotically converge to the Bayes error, just as the multiclass SVM of Lee, Lin and Wahba does. Put differently, *the need for a single-machine approach with a sum-to-zero constraint on the functions in order to asymptotically converge to the Bayes function was a specific technical requirement associated with the use of the SVM hinge loss.* When we change the loss function to the square loss, another commonly used loss function, the one-vs-all approach has precisely the same asymptotic convergence properties.

We are not claiming that this analysis is a strong argument in favor of the one-vs-all RLSC scheme as opposed to the one-vs-all SVM. The argument is an asymptotic one, applying only in the limit of infinitely many data points. There are a large number of schemes that will work equivalently with infinite amounts of data, and it is something of a technical oddity that the one-vs-all SVM appears not to be one of them. However, we do not believe that this asymptotic analysis tells us anything especially useful about the performance of a multiclass scheme on finite, limited amounts of high-dimensional data. In this regime, both a one-vs-all SVM scheme and a one-vs-all RLSC scheme have been demonstrated to behave quite well empirically. However, the fact that the one-vs-all RLSC scheme has equivalent asymptotic behavior to the Lee, Lin and Wahba scheme casts further doubt on the idea that their scheme will

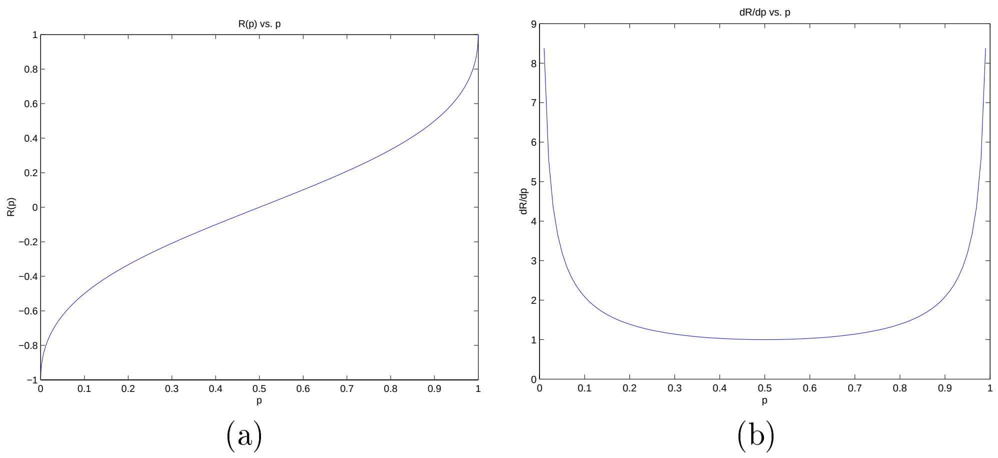


Figure 4-1: An analysis of the quantity $R(p)$. (a): $R(p)$ vs. p. (b): $\frac{dR}{dp}$ vs. p. We see that $R(p)$ is a strictly increasing function of p, implying that if class i is more likely than class j at point $\mathbf{x}$, then, asymptotically, $f_i(\mathbf{x}) > f_j(\mathbf{x})$.

prove superior to one-vs-all on real applications.

Bredensteiner and Bennett

Bredensteiner and Bennett [14] also suggest a single-machine approach to multiclass classification. Like Weston and Watkins, they begin by stating the invariant that they want the functions generated by their multiclass system to satisfy:

$$\mathbf{w_{y_i}}^T \cdot \mathbf{x_i} + b_i \geq \mathbf{w_j}^T \cdot \mathbf{x_i} + b_j + 1 - \xi_{ij}, \tag{4.25}$$

where $\mathbf{x_i}$ is a member of class y_i and $j \neq y_i$. They rewrite this equation as

$$(\mathbf{w_{y_i}} - \mathbf{w_j})^T \cdot \mathbf{x_i} \geq (b_j - b_i) + 1 - \xi_{ij}. \tag{4.26}$$

They then (rather inexplicably) argue that a good measure of the separation between class i and j is $\frac{2}{||w_i - w_j||}$, and suggest maximizing this quantity by minimizing $||w_i - w_j||$ over all pairs i and j. They also (with no additional justification) add the regularization term $\frac{1}{2}\sum_{i=1}^{N} ||w_i||^2$ to the (supposedly already regularized) objective function. The resulting optimization problem (where we have adjusted the notation substantially to fit with our development) is:

$$\min_{(\mathbf{w_i}, b_i \in \mathbf{R}^{d+1})} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}||w_i - w_j||^2 + \frac{1}{2}\sum_{i=1}^{N}||w_i||^2 + C\sum_{i=1}^{\ell}\sum_{j\neq y_i}\xi_{ij} \tag{4.27}$$

$$\text{subject to :} \quad \mathbf{w_{y_i}} - \mathbf{w_j}^T \cdot \mathbf{x_i} \geq (b_j - b_i) + 1 - \xi_{ij} \tag{4.28}$$

$$\xi_{ij} \geq 0 \tag{4.29}$$

Using standard but rather involved techniques, they derive the Lagrangian dual problem, and observe that the dot products can be replaced with kernel products.

Two sets of experiments were performed. The first set involved two data sets from the UCI repository [77] (`wine` and `glass`). Ten-fold cross validation was performed on each data set, and polynomials of degree one through five are used as models. On both data sets, the highest performance reported is for a one-vs-all SVM system rather than the multiclass system they derived (their multiclass SVM does perform better than an unregularized system which merely finds an arbitrary separating hyperplane).

In the second set of experiments, two separate subsets of the `USPS` data (no reference for this data set was provided) were constructed. Subsets of the training data were used, because training their multiclass method on the full data set was not computationally feasible.[3] On both data sets, a one-vs-all SVM system performs (slightly) better than their single-machine system.

Crammer and Singer

Crammer and Singer consider a similar but not identical single-machine approach to multiclass classification [24]. This work is a specific case of a general method for solving multiclass problems, discussed in [23, 22, 25] and in Section 4.2.2 below. The method can be viewed as a simple modification of the approach of Weston and Watkins [121]. Weston and Watkins start from the idea that if a point $\mathbf{x}$ is in class i, we should try to make $f_i(\mathbf{x}) \geq f_j(\mathbf{x}) + 2$ for $i \neq j$, and arrive at the following

[3]For example, their method takes over 10,000 seconds to train on a data set of 1,756 examples in three classes. SvmFu routinely solves problems on 5,000 or more points in 30 seconds or less.

formulation:

$$\min_{\mathbf{f_1},\ldots,\mathbf{f_N}\in\mathcal{H},\xi\in\mathbf{R}^{\ell(\mathbf{N-1})}} \quad \textstyle\sum_{i=1}^{N}||f_i||_K^2 + C\sum_{i=1}^{\ell}\sum_{j\neq y_i}\xi_{ij} \tag{4.30}$$

$$\text{subject to :} \quad f_{y_i}(\mathbf{x_i}) + b_{y_i} \geq f_j(\mathbf{x_i}) + b_j + 2 - \xi_{ij} \tag{4.31}$$

$$\xi_{ij} \geq 0 \tag{4.32}$$

Crammer and Singer begin with the same condition, but instead of paying for *each* class $j \neq i$ for which $f_i(\mathbf{x}) < f_j(\mathbf{x}) + 1$, [4] they pay only for the *largest* $f_j(\mathbf{x})$. This results in a *single* slack variable for each data point, rather than the $N-1$ slack variables per point in the Weston and Watkins formulation. The resulting mathematical programming problem is (as usual, placing the formulation into our own notations for consistency):

$$\min_{\mathbf{f_1},\ldots,\mathbf{f_N}\in\mathcal{H},\xi\in\mathbf{R}^{\ell}} \quad \textstyle\sum_{i=1}^{N}||f_i||_K^2 + C\sum_{i=1}^{\ell}\sum_{j\neq y_i}\xi_i \tag{4.33}$$

$$\text{subject to :} \quad f_{y_i}(\mathbf{x_i}) \geq f_j(\mathbf{x_i}) + 1 - \xi_i \tag{4.34}$$

$$\xi_i \geq 0 \tag{4.35}$$

The majority of the paper is devoted to the development of an efficient algorithm for solving the above formulation. The Lagrangian dual is taken, and the standard observations that the dot products can be replaced with kernels are made. A rather elegant dual decomposition algorithm is developed, in which a single data point is chosen at each step (we note in passing that the Crammer and Singer formulation does not include unregularized bias terms) and an iterative algorithm is used to solve the reduced N-variable quadratic programming problem associated with the chosen data point. A number of additional implementation tricks are used, including using the KKT conditions for example selections, caching kernel values, maintaining an

[4]The choice of 1 rather 2 as a "required difference" is arbitrary. Crammer and Singer quite reasonably choose 1 for simplicity. The choice of 2 in Weston and Watkins seems to be motivated from a desire to make the system as like standard binary SVMs as possible, where we require a margin of 1 for points in the positive class and -1 for points in the negative class. This choice is arbitrary: if we required 1 for points in the positive class and 0 for points in the negative class, the details of the algorithm would change, but the function found by the algorithm would not.

active set from which the example to be optimized in a given iteration is chosen, and cooling of the accuracy parameter (all of these techniques except the last are utilized in SvmFu, see Chapter 2).

In the experiments section of the paper, Crammer and Singer make claims regarding both speed and accuracy of their method. For speed, they claim that their method is orders of magnitude faster than a one-vs-all approach, but the comparison is not fair: their system uses 2Gb of memory and a sophisticated caching scheme, and they compare it to Platt's 1998 results that used no caching whatsoever. Furthermore, the paper states that the fastest version has "two more technical improvements which are not discussed here but will be documented in the code that we will shortly make available"; as of this writing, fourteen months after the paper was submitted to the Journal of Machine Learning Research, the code has not been made available, so this claim is impossible to evaluate.

As far as accuracy, Crammer and Singer considered a number of data sets from the UCI repository. They produce a chart showing the difference in error rate between their one-machine system and an OVA system, but not the actual error rates. There are two datasets for which the difference beween their system and OVA seems to be large: `satimage` with a difference of approximately 6.5% in performance, and `shuttle` with a difference of approximately 3% in performance. In personal communiation, Crammer indicated that the actual error rates for his system on these two datasets were 8.1% and 0.1%, respectively. In our own one-vs-all experiments on this dataset, we achieved error rates of 7.9% for the `satimage` data and 0.35% on the `shuttle` data.[5] The extra 2 tenths of Crammer's system on the `shuttle` data, given that the problem can be currently solved to such high accuracy, is intriguing, although not obviously statistically significant. In any case, these numbers are in sharp contast to the numbers reported in the paper; we speculate that the kernel parameters were optimized for the Crammer and Singer scheme, and not for the one-vs-all scheme.

[5]The `satimage` experiments are discussed in greater detail later in the chapter. For the `shuttle` data, the results were obtained with $\sigma = 16$ and $C = 8$; these parameters were selected in the same way as for the `satimage` data, described later.

Summary

When we apply the one-vs-all strategy, we solve N separate optimization problems, where N is the number of classes. The single machine approaches solve a single optimization problem, finding all N functions simultaneously. Although these approaches may have theoretical interest to some, it does not appear that they offer any advantages in the solution of multiclass classification problems. In particular, the methods are generally complicated to implement and slow to train, while giving no better performance than a simple one-vs-all scheme.

4.2.2 Error Correcting Code Approaches

We now turn to error-correcting code approaches, a second major approach to combining binary classifiers into a multiclass classification system.

Dietterich and Bakiri

Dietterich and Bakiri [29] first popularized the idea of using error-correcting codes for multiclass classification. We will describe the method using notation introduced later by Allwein et al. ([1], see below) in order to simply the presentation.

Dietterich and Bakiri suggested the use of a $\{-1, 1\}$-valued matrix M of size N by F ($M \in \{-1, 1\}^{N \times F}$), where N is the number of classes and F is the number of binary classifiers to be trained. To avoid excessive subscripting, we let M_{ij} refer to the entry in the ith row and the jth column of M. The ith column of the matrix induces a partition of the classes into two "metaclasses", where a point $\mathbf{x_i}$ is placed in the positive metaclass for the jth if and only if $M_{y_i j} = 1$. In our framework, in which the binary classifiers implement Tikhonov regularization, the jth machine solves the following problem:

$$\min \sum_{i=1}^{\ell} V(f_j(\mathbf{x_i}), M_{y_i j}) + \lambda ||f_j||_K^2. \tag{4.36}$$

When faced with a new test point $\mathbf{x}$, we compute $f_1(\mathbf{x}), \ldots, f_F(\mathbf{x})$, take the signs of these values, and then compare the Hamming distance between the resulting vector and each row of the matrix, choosing the minimizer:

$$f(\mathbf{x}) = \arg \min_{r \in 1,\dots,N} \sum_{i=1}^{F} \left(\frac{1 - \operatorname{sign}(M_{ri} f_i(\mathbf{x}))}{2} \right). \tag{4.37}$$

This representation had been previously used by Sejnowski and Rosenberg [100], but in their case, the matrix M was chosen so that a column of M corresponded to the presence or absence of some specific feature across the given classes. For example (taken from [29]), in a digit recognizer, one might build a binary classifier that learned whether or not the digit contained a vertical line segment, placing the examples in classes 1, 4, and 5 in the positive metaclass for this classifier, and the remaining classes in the negative metaclass.

Dietterich and Bakiri take their cue from the theory of error-correcting codes [11], and suggest that the M matrix be constructed to have good error-correcting properties. The basic observation is that if the minimum Hamming distance between rows of M is d, then the resulting multiclass classification will be able to correct any $\lfloor \frac{d-1}{2} \rfloor$ errors. They also note that good *column* separation is important when using error-correcting codes for multiclass classification; if two columns of the matrix are identical (or are opposites of each other, assuming an algorithm that treats positive and negative examples equivalently), they will make identical errors.

After these initial observations, the majority of the paper is devoted to experimental results. A number of data sets from various sources are used, including several data sets from the UCI Machine Learning Repository [77], and a subset of the NETtalk data set used in [100]. Two learning algorithms were tested: decision trees using a modified version of the C4.5 algorithm [88], and feed-forward neural networks. The parameters were often tuned extensively to improve performance. In some cases, the algorithms were modified for individual data sets. They considered four different methods for constructing good error-correcting codes: an exhaustive method, a method that selects a subset of the columns generated by the exhaustive method, a method based on randomized hill climbing, and a method based on using BCH codes or a subset of BCH codes (sometimes selected using manual intervention).

In summary, although the description of the experimental work is quite lengthy, it would be essentially impossible to replicate the work exactly.

A large variety of experimental results are reported. It appears that in general, with the data sets and algorithms tried, the error-correcting code approach performs better than a one-vs-all approach. However, the difference is often small, and it is difficult to know how good the underlying binary classifiers are. In many instances, only relative performance results are given — the difference between the error-correcting and a one-vs-all method is given, but the actual performance numbers are not given, making comparison to alternate approaches (such as a one-vs-all SVM scheme) impossible.

Allwein, Schapire and Singer

In 2000, Allwein, Schapire and Singer [1] extended the earlier work of Dietterich and Bakiri in several directions. They were specifically interested in *margin-based* classifiers, where the underlying classifier is attempting to minimize (while also possibly trying to minimize a regularization term) an expression of the form

$$\frac{1}{\ell}\sum_{i=1}^{\ell} L(y_i f(\mathbf{x_i})), \tag{4.38}$$

where L is an arbitrary (chosen) function. The quantity $y_i f(\mathbf{x_i})$ is referred to as the *margin*, and is notated as z, so that we are interested in minimizing

$$\frac{1}{\ell}\sum_{i=1}^{\ell} L(z), \tag{4.39}$$

In the case of the SVM, $L(yf(\mathbf{x})) = (1 - yf(\mathbf{x_i}))_+ \equiv (1 - z)_+$ and in the case of RLSC, $L(yf(\mathbf{x})) = (1 - yf(\mathbf{x}))^2 = (y - f(\mathbf{x}))^2 \equiv (1 - z)^2$; we see that both SVM and RLSC are margin-based classifiers, and that we can easily relate the margin loss function $L(yf(\mathbf{x}))$ to the loss function $V(f(\mathbf{x}), y)$ we considered in Chapters 2 and 3. Allwein et al. are also very interested in the AdaBoost algorithm [35, 96], which builds a function $f(\mathbf{x})$ that is a weighed linear combination of base hypothesis h_t:

$$f(\mathbf{x}) = \sum_t \alpha_t h_t(\mathbf{x}), \tag{4.40}$$

where the h_t are selected by a (weak) base learning algorithm, and reference numerous papers indicating that AdaBoost is approximately greedily minimizing

$$\sum_{i=1}^{\ell} e^{-y_i f(\mathbf{x_i})}, \tag{4.41}$$

demonstrating that AdaBoost is a margin-based classifier with $L(yf\mathbf{x}) = e^{-y_i f(\mathbf{x_i})}$. It is important to note that this notion of "margin" is more general and not necessarily entirely equivalent to the idea as used in the context of SVMs, where (at least in the separable case) there is an obvious geometric meaning associated with the term "margin". In particular, note that we do not always prefer to achieve "large margin": for RLSC, we want to force the margin at each example to be as close to 1 as possible, rather than trying to maximize the margin.

Allwein et al. chose the matrix $M \in \{-1, 0, 1\}^{N \times F}$, rather than only allowing 1 and -1 as entries in the matrix as Dieterrich and Bakiri did. If $M_{y_i j} = 0$, then example i is simply not used when the jth classifier is trained. With this extension, they were able to place one-vs-all classification, error-correcting code classification schemes, and all-pairs classification schemes [51] in a single theoretical framework.

If the classifiers are combined using Hamming decoding (taking the signs of the real values output by the classifiers, then finding the closest match among the rows of M), we again have

$$f(\mathbf{x}) = \arg \min_{r \in 1, \ldots, N} \sum_{i=1}^{F} \left(\frac{1 - \text{sign}(M_{ri} f_i(\mathbf{x}))}{2} \right), \tag{4.42}$$

where it is now understood that if $M_{ri} = 0$ (class r was not used in the ith classifier), class r will contribute $\frac{1}{2}$ to the sum. Allwein et al. note that the major disadvantage of Hamming decoding is that completely ignores the magnitude of the predictions, which can often be interpreted as a measure of "confidence" of a prediction. If the underlying classifiers are margin-based classifiers, they suggest using the loss function L instead of the Hamming distance. More specifically, they suggested that the prediction for a point $\mathbf{x}$ should be the class r that minimizes the total loss of the binary predictions under the assumptions that the label for point $\mathbf{x}$ for the ith machine is M_{ri}:

$$f(\mathbf{x}) = \arg \min_{r \in 1,\ldots,N} \sum_{i=1}^{F} L(M_{ri} f_i(\mathbf{x})) \tag{4.43}$$

This procedure is known as *loss-based decoding.* If the matrix M represents a one-vs-all coding scheme ($M_{ri} = 1$ if $r = i$, $M_{ri} = -1$ otherwise), the above equation simplifies to

$$\begin{aligned} f(\mathbf{x}) &= \arg \min_{r \in 1,\ldots,N} \sum_{i=1}^{F} L(M_{ri} f_i(\mathbf{x})) & (4.44) \\ &= \arg \min_{r \in 1,\ldots,N} L(f_r(\mathbf{x})) - \sum_{i \neq r}^{F} L(-f_i(\mathbf{x})). & (4.45) \end{aligned}$$

It is easy to check that for all the loss functions considered here (SVM, RLSC, and AdaBoost), the prediction in the one-vs-all scheme will be chosen so that

$$f(\mathbf{x}) = \arg \max_{r} f_r(\mathbf{x_i}). \tag{4.46}$$

Allwein et al. provide an elegant analysis of the training error of multiclass error-correcting code based systems using both Hamming decoding and loss-based decoding. Some notation is required to state these bounds. They define ε to be the average loss of the underlying binary classifiers:

$$\varepsilon \equiv \frac{1}{F\ell} \sum_{i=1}^{F} \sum_{j=1}^{\ell} L(M_{y_j i} f_i(\mathbf{x_j})). \tag{4.47}$$

They define ρ to be the minimum distance between a pair of rows of the matrix M, using a generalized notion of Hamming distance in which, $d(i,j) = \frac{1}{2}$ if either i or j is zero, and $d(i,j)$ is the standard Hamming distance between i and j otherwise. Using this notation, ρ can be expressed as

$$\rho = \min_{c_1, c_2 \in 1,\ldots,N, c_1 \neq c_2} \sum_{i=1}^{N} d(M_{c_1 i}, M_{c_2 i}) \tag{4.48}$$

$$= \min_{c_1, c_2 \in 1,\ldots,N, c_1 \neq c_2} \sum_{i=1}^{N} \frac{1 - M_{c_1 i} M_{c_2 i}}{2}. \tag{4.49}$$

They begin by considering loss-based decoding. They make the (mild, and satisfied by all loss functions considered) requirement on the loss function that

$$\frac{L(z) + L(-z)}{2} \geq L(0) \geq 0. \tag{4.50}$$

Under these assumptions, they are able to show that the training error of a multiclass system (the number of mistakes made on the training set) that uses loss-based decoding is bounded above by

$$\frac{\ell N \varepsilon}{\rho L(0)}. \tag{4.51}$$

For Hamming decoding, assuming instead that L satisfies

$$z < 0 \Longrightarrow L(z) \geq L(0) \geq 0, \tag{4.52}$$

they are able to bound the training error by

$$\frac{2\ell N \varepsilon}{\rho L(0)}. \tag{4.53}$$

Some additional discussion notes that these bounds implicitly depend on the fraction of zero entries in the matrix, and modified versions of the bounds that make this dependence explicit are provided.

Allwein et al. next turn to an analysis of the generalization performance of multiclass loss-based schemes in the particular case when the underlying binary classifier is AdaBoost. The arguments are extensions of those given by Schapire et al. in [95], but are beyond the scope of this thesis.

The remainder of the paper is devoted to experiments on both toy and UCI

Repository data sets, using AdaBoost and SVMs as the base learners. The two stated primary goals of the experiments are to compare Hamming and loss-based decoding and to compare the performance of different output codes.

The toy experiment considers $100k$ one-dimensional points generated from a single normal distribution, selecting the class boundaries so that each class contains 100 training points. AdaBoost is used as the weak learner, and comparisons are made between Hamming and loss-based decoding, and a one-vs-all code and a complete code. The authors find that the loss-based decoding substantially outperforms the Hamming decoding, and that the one-vs-all and complete codes perform essentially identically.

Allwein et al. next consider experiments on a number of data sets from the machine learning repository. For SVMs, eight data sets are used: `dermatology`, `satimage`, `glass`, `ecoli`, `pendigits`, `yeast`, `vowel`, and `soybean`. Five different codes are considered: the one-vs-all code, the all-pairs code (omitted when there were too many classes), the complete code (omitted when there were too many classes), and two types of random codes. The first type had $\lceil 10log_2(N)\rceil$ columns, and each entry was chosen to be 1 or -1 with equal probabilities. The codes were picked by considering $10,000$ random matrices, and picking the one with the highest value of ρ which did not have any identical columns. These codes were refereed to as *dense* codes. They also considered *sparse* codes, which had $\lceil 15log_2(N)\rceil$ columns, and each entry was 0 with probability $\frac{1}{2}$, and 1 or -1 with probability $\frac{1}{4}$ each. Again, 10,000 random matrices were considered, and the one with the best ρ with no identical columns and no columns or rows containing only zeros was chosen.

Direct numerical results of the experiments are presented, as well as bar graphs showing the relative performance of the various codes. The authors conclude that "For SVM, it is clear that the widely used one-against-all code is inferior to all the other codes we tested." However, this conclusion is somewhat premature. All the SVM experiments were performed using a polynomial kernel of degree 4, and no justification for this choice of kernel was given. Additionally, the regularization parameter used (i.e., λ) was not specified by the authors. Looking at the bar graphs comparing relative performance, we see that there are two data sets on which the one-vs-all SVMs seem to be doing particularly badly compared to the other codes: `satimage` and `yeast`. We performed our own SVM experiments on this data, using a Gaussian kernel with σ and C tuned separately for each scheme (for actual parameter values, see Section 4.3). [6]

The results are summarized in Table 4.1 and Table 4.2. We find that while other codes do sometimes perform (slightly) better than one-vs-all, that none of the differences are large. This is in stark contrast to the gross differences reported by Allwein et al. Although we do not test the other data sets from the UCI repository, this experiment strongly supports the hypothesis that the differences observed by Allwein et al. result from a poor choice of kernel parameters, which makes the SVM a much weaker classifier than it would be with a good choice of kernel. In this regime, it is plausible the errors from the different classifiers will be somewhat decorrelated, and that a scheme with better error-correcting properties than the one-vs-all scheme will be superior. However, given that our goal is to solve the problem as accurately as possible, it appears that choosing the kernel parameters to maximize the strength of the individual binary classifiers, and then using a one-vs-all multiclass scheme, performs as well as the other coding schemes, and, in the case of the `satimage` data, substantially better than *any* of the coding schemes when the underlying classifiers are weak.[7]

[6]We replicated the dense and sparse random codes as accurately as possible, but the information in Allwein et al. is incomplete. For both codes, we added the additional constraint that each column had to contain at least one $+1$ and at least one -1; one assumes that Allwein et al. had this constraint but did not report it, as without it, the individual binary classifiers could not be trained. For the sparse random code, the probability that a random column of length six (the number of classes in the `satimage` data set) generated according to the probabilities given fails to contain both a $+1$ and a -1 is more than 35%, and the procedure as defined in the paper fails to generate a single usable matrix. Personal communication with Allwein et al. indicate that it is likely that columns not satisfying this constraint were thrown out immediately upon generation. Additionally, there are only 601 possible length six columns containing at least one $+1$ and one -1 entry, and if these columns were chosen at random, only 28% of the matrices generated (the matrices have $\lceil 15 log_2(6) \rceil = 39$ columns) would not contain duplicate columns. Because there was no mention in either case of avoiding columns which were opposites of each other (which is equivalent to duplication if the learning algorithms are symmetric), we elected to allow duplicate columns in our sparse codes, in the belief that this would have little effect on the quality of the outcome.

[7]In this context, weak is not use in a formal sense, but merely as a stand-in for "badly tuned."

	One-vs-All	All-Pairs	Complete	Dense	Sparse
Allwein et al.	40.9	27.8	13.9	14.3	13.3
Rifkin	7.9	7.9	8.0	8.2	8.2

Table 4.1: Multiclass classification results for the `satimage` data set. Allwein et al. used a polynomial kernel of degree four and an unknown value of C. Rifkin used a Gaussian kernel tuned separately for each scheme. We see that with the Gaussian kernel, overall performance is much stronger, and the differences between coding schemes disappear. Entries in the table are the percentage of errors; all results are rounded to the nearest tenth of a percent.

	One-vs-All	All-Pairs	Complete	Dense	Sparse
Allwein et al.	72.9	40.9	40.4	39.7	47.2
Rifkin	39.0	39.3	38.5	39.3	38.8

Table 4.2: Multiclass classification results for the `yeast` data set. Allwein et al. used a polynomial kernel of degree four, an unknown value of C, and ten-fold cross-validation. Rifkin used a Gaussian kernel tuned separately for each scheme, and leave-one-out cross-validation. We see that with the Gaussian kernel, the performance of the one-vs-all scheme jumps substantially, and the differences between coding schemes disappear. Entries in the table are the percentage of errors; all results are rounded to the nearest tenth of a percent.

Crammer and Singer

In [23, 22, 25], Crammer and Singer develop a formalism for multiclass classification using *continuous* output coding. This formalism includes the single-machine approach discussed in [24] and above as a special case.

The Crammer and Singer framework begins by assuming that a collection of binary classifiers $f_1, \ldots, f_F$ is provided. The goal is then to *learn* the N-by-F error-correcting code matrix M. Crammer and Singer show (under some mild assumptions) that finding an optimal discrete code matrix is an NP-complete problem, so they relax the problem and allow the matrix M to contain real-valued entries. Borrowing ideas from regularization, they argue that we would like to find a matrix M that has good performance on the training set but also has a small norm. To simplify the presentation, we introduce the following notation. We let $\bar{f}(\mathbf{x})$ denote the vector $f_1(\mathbf{x}), \ldots, f_F(\mathbf{x})$, and we let M_i denote the ith row of the matrix M. Given a matrix M, we let $K(\bar{f}(\mathbf{x}), M_i)$ denote our confidence that point x is in class i; here K is an arbitrary positive definite kernel function satisfying Mercer's Theorem. Then, the

Crammer and Singer approach is:

$$\min_{M \in \mathbf{R}^{N \times F}} \quad \lambda ||M||_p + \textstyle\sum_{i=1}^{\ell} \xi_i \tag{4.54}$$

$$K(\bar{f}(\mathbf{x_i}), M_{y_i}) \geq K(\bar{f}(\mathbf{x_i}), M_r) + 1 - \xi_i \tag{4.55}$$

In the above formulation, the constraints range over all points $\mathbf{x_i}$, and all classes $r \neq y_i$. Simply put, we try to find a matrix with small norm so that the confidence for the correct class is greater by at least one than the confidence for any other class. Note that as in [24] above, there is only a single slack variable ξ_i for each data point, rather than $N-1$ slack variables per data point as in many of the other formulations we discuss.

In their general formulation, the norm in which the matrix is measured ($||M||_p$) is left unspecified. Crammer and Singer briefly show that if $p = 1$ or $p = \infty$ and $K(x, y) = x \cdot y$, the resulting formulation is a linear program. They spend the majority of the paper considering $p = 2$ (technically, they penalize $||M||_2^2$, not $||M||_2$), and showing that this choice results in a quadratic programming problem. They take the dual (in order to introduce kernels; again, they start with the linear formulation in the primal), and indicate some algorithmic approaches to solving the dual problem.

In an interesting twist, Crammer and Singer also show that we can derive the one-machine multiclass SVM formulation in [24] (and discussed in a previous section) by taking $\bar{f}(\mathbf{x}) = \mathbf{x}$. In this case, the implicit assumption is that our "given" binary classifiers are d (the dimensionality of the input space) machines, where $f_i(\mathbf{x})$ is equal to the value of the ith dimension at point x. In the linear case ($K(x, y) = x \cdot y$), the code matrix M becomes the N separating hyperplane functions $w_1, w_2, \ldots, w_N$. This formulation is discussed in greater detail in the previous section.

Experiments are performed on seven different data sets from the UCI repository, as well as (a subset of) the MNIST data set. The experiments compare the performance of the continuous output codes to discrete output codes, including the one-vs-all code, BCH codes, and random codes. Personal communication with Crammer indicates that the "base learners" for the continuous codes are linear SVMs. Seven different kernels are tested, although their identities are not disclosed (although I receieved personal communication that they were homogeneous and nonhomogeneous polynomials of degree one through three, and a Gaussian kernel with a sigma that was not recorded). No performance results for individual experiments are given. Instead, for each dataset, we find the improvement in performance for the "best kernel" (presumably the kernel with the largest difference in performance), and the average improvement in performance across the seven kernels. It is important to note that his comparison was NOT against an OVA system, but against using the error-correcting coding approach directly with *linear SVMs* as the underlying binary classifiers. Therefore, he was comparing the classification ability of nonlinear and linear systems on datasets for which we have already seen that Gaussian classifiers perform very strongly. In this context, his results are unsurprising.

Crammer communicated to us personally the actual performance numbers, which allows us to compare his continuous codes to an OVA approach. For the `satimage` data, the *best* error rate he achieved (over all seven kernels and three coding schemes) was 9.8%, compared to 7.9% for OVA in our own experiments. For the `shuttle` data, his best error rate was 0.5%, as opposed to 0.35% for OVA in our own experiments. We see that although his nonlinear system did greatly outperform a linear system, it did not allow us to actually achieve better multiclass classification error rates, which is of course our actual goal.

Fürnkranz

Very recently, Fürnkranz published a paper on Round Robin Classification [42], which is his 2002 name for all-vs-all classification. He used Ripper [18], a rule-based learner, as his underlying binary learner. He experimentally found that an all-vs-all system had improved performance compared to a one-vs-all scheme. His data sets included the `satimage` and `yeast` data sets studied here, and the round robin system achieved error rates of 10.4% and 41.8% on the two data sets (as opposed to 7.9% and 39.0% for one-vs-all in our own experiment with SVM as the base learner). These numbers support the hypothesis that Ripper is not as effective a binary learner as SVMs, and are therefore able to benefit from an error-correcting scheme such as all-pairs; however, the scheme does not enable Fürnkranz to obtain better results than an OVA SVM scheme.

Summary

Dietterich and Bakiri were fully aware of both the promise and the difficulty of this approach, which obviously relies heavily on the errors produced by different binary classifiers being (at least somewhat) decorrelated. In a companion paper, they address this issue for the specific case where the underlying binary classifiers are decision trees [62]. We believe (and show experimentally in Section 4.3) that when the underlying classifiers are appropriately tuned regularization systems such as SVM or RLSC, that the errors produced by different binary classifiers will be extremely highly correlated, implying that a one-vs-all scheme will be at least as effective as an error-correcting code scheme. Furthermore, we will show that across several data sets, using SVM (or RLSC) in a one-vs-all framework yields extremely strong results.

4.3 Experimental Results

In this section we provide experimental results on multiclass classification using the schemes suggested by Allwein et al. [1], with well-tuned SVMs or RLSCs as the underlying binary classifiers. We will observe, across a number of different data sets, that a simple OVA system performs approximately as well as any other system. We will also see that the systems, and the underlying classifiers, produce results which are very highly correlated, making it unlikely that the more complicated error-correcting code schemes will outperform OVA.

For clarity, we briefly review the error-correcting code schemes here; more details can be found in our discussion of the Allwein et al. paper (Section 4.2.2). The one-vs-all code, referred to as OVA, builds one classifier per class, trying to separate a single class from all other classes. The all-vs-all or all-pairs code, referred to as AVA, builds $\frac{N(N-1)}{2}$ classifiers, each trying to separate a pair of classes. The complete code, referred to as COMP, builds $2^{N-1}-1$ classifiers, where each classifier tries to separate a nontrivial subset of the classes from the remaining classes. The dense code, also referred to as DEN, builds $\lceil 10 log_2(N) \rceil$ classifiers, each of which separates a random subset of the classes from the remaining classes. The sparse code, also referred to as SPA, builds $\lceil 15 log_2(N) \rceil$ classifiers, and is similar to the dense code except that each classifier only involves approximately half the classes. For more details on the dense and sparse codes, see Section 4.2.2.

In this section, we will consider results on four data sets: the `satimage` and `yeast` data considered (among other data sets) by Allwein et. al, the `20newsgroups` text classification data set considered by Rennie and Rifkin [90], and the `gcm` cancer classification data set explored by the author in joint work with members of the Cancer Genomics Center at the Whitehead Institute, published in [127] and [89].

4.3.1 Preprocessing and Parameter Settings

The `satimage` data consists of 4,435 training points and 2,000 testing point in six classes, obtained from the UCI Machine Learning Repository [77]. For numerical stability, we divided all the numbers by 100, and also renumbered the classes (in the data set as distributed, the classes are numbered 0, 1, 2, 3, 4, and 6). We tested all five schemes using SVMs with a Gaussian kernel as the binary learner. For each scheme, we tuned the σ for the Gaussian and the regularization parameter C individually. We tuned each parameter by starting it at 1, and then increasing it and decreasing it by powers of 2 until a local minima was found. Because the dependence on σ was much stronger than on C, we tuned σ first and then C. Table 4.3 describes the optimal settings for each scheme. We note that neither dependency seemed very strong — often several powers of two in each direction induced performance differences of less than one percent.

The `yeast` data, also from the UCI Machine Learning Repository, consists of 1,484 points in 10 classes. Because no separate testing set was provided, we tested our schemes using leave-one-out cross-validation. SVMs were again used as a binary learner. We tested all five schemes using SVMs with a Gaussian kernel. For the

Scheme	σ	C
OVA	.25	2
AVA	.25	8
COMP	.25	2
DEN	.25	2
SPA	.25	1

Table 4.3: The optimal σ and C settings for each of the five multiclass schemes used on the `20Newsgroups` data set.

Scheme	σ	C
OVA	.125	1
AVA	.25	8
COMP	.125	1
DEN	.125	1
SPA	.125	2

Table 4.4: The final σ and C settings for each of the five multiclass schemes used on the `20Newsgroups` data set. The settings were tuned for OVA, AVA, DEN and SPA, and chosen by the author for COMP.

OVA, AVA, DEN and SPA code, we tuned the parameters as above. For the COMP code, because there were 511 classifiers and the experiments took a long time to run, and because none of the other systems showed a very strong dependence on the parameters, we simply chose a single set of parameters that seemed likely to be effective. Table 4.4 describes the final settings for each scheme. Again, the dependence on the parameters was not strong.

The Global Cancer Map, or `gcm` data in this study, is a collection of tumors collected to explore the task of automatic cancer classification.[8] There are 190 samples in 14 classes, and each example is a 16,063 dimensional vector. For numerical stability we divided all raw values by 5,000. We tested each scheme using leave-one-one cross validation. We used a linear SVM with C tuned individually for each scheme. We found that $C = .125$ was best for the OVA and DEN schemes, and $C = .25$ was best

[8]The data is available online from the Cancer Genomics website, at `http://www-genome.wi.mit.edu/cgi-bin/cancer/data sets.cgi.`

for the AVA and SPA schemes. We did not use the COMP scheme as we would have needed $2^{19} - 1$ classifiers.

The `20Newsgroups` data is a set of documents divided into twenty classes; the task is to choose which of 20 different newsgroups a document came from.[9] There are a total of 19,998 documents in 20 classes; perform ten-fold cross-validation, creating ten training sets of size 15,935 and 10 test sets of size 3,993. We used linear RLSC with $\lambda\ell$ tuned individually for each scheme. We found that $\lambda\ell = 1$ worked best for OVA and DEN, and $\lambda\ell = .5$ worked best for AVA and SPA, although the sensitivity to the precise choice of $\lambda\ell$ within a wide range was quite low. We did not use the COMP scheme as we would have needed $2^{19} - 1$ binary classifiers.[10]

4.3.2 Results

Table 4.5 presents the accuracy rates for the different classification schemes. We see immediately that across the data sets, the performance of the OVA scheme is quite strong. On the `satimage` and `20Newsgroups` data, OVA is strongest overall, and on the `yeast` data set, it is within one half of one percent of the strongest performance. On the `gcm` data, it is 2.6% worse than the DENSE scheme; however, this data set consisted of only 190 points total (in 16,063 dimensions), with the difference in error being 5 samples out of 190, and the difference is not clearly statistically significant. These numbers are in stark contrast to the observations of Allwein et. al, who found that the OVA scheme was much worse than any of the error-correcting schemes. Their work used polynomial kernels, and it seems that this resulted in poorly tuned SVMs that had errors that were somewhat decorrelated, allowing the error-correcting schemes to improve performance. However, the best performance they report on the `satimage` and `yeast` data set were lower than what we found, so it appears that a good way to get the best possible performance is simply to use an OVA scheme with well-tuned binary classifiers.

[9]The data is available online from `http://www.ai.mit.edu/people/jrennie/ecoc-svm`.

[10]In Section 4.4, below, we show that for RLSC, the predictions for the COMP scheme will be *identical* to those of the OVA scheme, providing another good reason not to use it.

	OVA	AVA	COMP	DEN	SPA
`satimage`	7.85	7.85	7.95	8.2	8.15
`yeast`	39.0	39.4	38.5	39.3	38.8
`gcm`	18.9	30.5	—	16.3	22.1
`20Newsgroups`	12.8	14.8	—	14.2	13.3

Table 4.5: Results for multiclass classification using five different coding schemes on four different data sets. Entries in the tables are percentages of misclassified points.

4.3.3 Correlation Analysis

For the `satimage` and `yeast` data sets, we performed additional experiments in order to further illustrate the behavior of the different schemes. Intuitively (but naively), it seems that the OVA scheme may be inferior, because it is unable to correct for any errors. The other four schemes all possess an innate ability to correct numerous errors, and indeed, this is the primary motivation for their use. However, the value of the ability to correct some number of errors depends crucially on there being some amount of *decorrelation* between the classifiers. In particular, the error-correction of these schemes works when a relatively small number of incorrect binary classifiers are "outvoted" by the remaining classifiers. If the classifiers all make essentially identical errors, the code's ability to correct will be of no value. We see in this section that this is exactly what occurs.

Tables 4.6 and 4.7 show, for each pair of codes used, the fraction of points which were predicted to be in the same class by both codes in the pair. We see immediately that the numbers are very high, indicating that on both data sets, the different systems are not just getting the same points correct, but *they are making the same mistakes* — if one scheme incorrectly decides that a point in class i is instead in class j, then the other systems are likely to also mistakenly place the point in class j. Indeed, fully 95.5% of the test points in the `satimage` data set and 84% of the points in the `yeast` data set are placed into the same class by all five schemes. The numbers are lower for the `yeast` then the `satimage` data (likely related to the fact that the overall error rate is much higher), but the correlation is very high for both data sets.

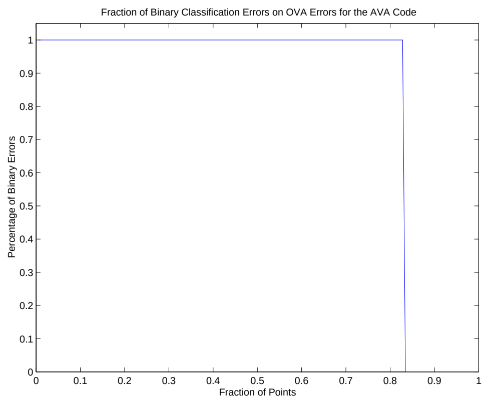
OVA	AVA	COMP	DEN	SPA	
1.0	.987	.995	.995	.980	OVA
	1.0	.986	.986	.979	AVA
		1.0	.997	.980	COMP
			1.0	.980	DEN

Table 4.6: Overlaps of predictions between classifiers on the `satimage` data set. Each entry in the table gives the fraction of identical class predictions for two different multiclass schemes. The five schemes are one-vs-all (OVA), all-pairs (AVA), a complete code (COMP), a dense code (DEN), and a sparse code (SPA). See the text for details of the parameter settings and code definitions.

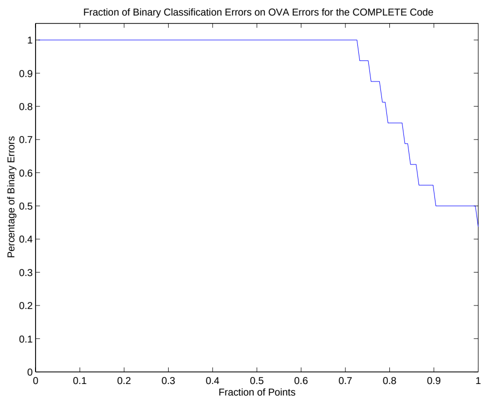
OVA	AVA	COMP	DEN	SPA	
1.0	.888	.970	.920	.950	OVA
	1.0	.906	.894	.870	AVA
		1.0	.923	.967	COMP
			1.0	.924	DEN

Table 4.7: Overlaps of predictions between classifiers on the `yeast` data set. Each entry in the table gives the fraction of identical class predictions for two different multiclass schemes. The five schemes are one-vs-all (OVA), all-pairs (AVA), a complete code (COMP), a dense code (DEN), and a sparse code (SPA). See the text for details of the parameter settings and code definitions.

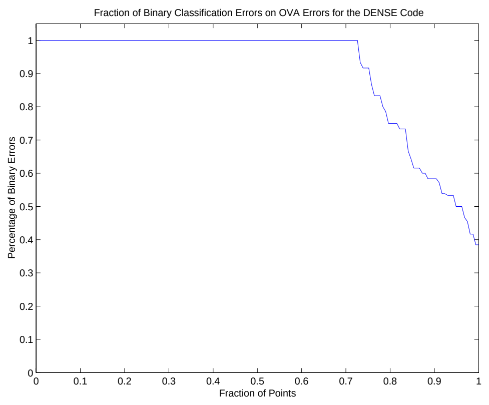
To further understand the behavior of the systems, we performed an additional experiment. For each point $\mathbf{x}$ that belonged to class i but was mistakenly placed in class j, we considered (separately for each scheme) the behavior at $\mathbf{x}$ for each binary classifier that placed classes i and j into different metaclasses. The idea is that if an error-correcting scheme is going to avoid making the same mistake on a point that the OVA scheme makes, it must be able to construct a majority of binary classifiers that successfully tell that the point is in a metaclass including class i rather than a metaclass including class j. This is precisely the opposite of what we see. Figures 4-2 and 4-3 show the results of the experiment for the `satimage` and `yeast` data sets, respectfully. Each figure shows the reverse cumulative distribution function of the percentage of binary errors made on points that were errors for the OVA system for classifiers that placed the true class and the class chosen by the OVA system into separate metaclasses. For example, looking at Figure 4-2c, we see that for the DENSE code, for 80% of the points which were OVA errors, 70% or more of the binary classifiers that tried to separate the true class from the mistakenly chosen OVA class failed to do so. Intuitively (this is not strictly correct, because predictions are turned into confidences via the loss function), we expect that we would expect an error-correcting approach to correct an OVA error only if more than half of the binary classifiers successfully avoided the error. As we see from the figures, this is only the case for a very small portion of the OVA errors. Indeed, a reasonable conclusion to draw from the figures is that *if the OVA system mistakenly places a point* $\mathbf{x}$ *that is in class i into class j, nearly every binary classifier that places classes i and j into different metaclasses will mistakenly place* $\mathbf{x}$ *into the wrong metaclass.* Viewed from this perspective, it is unsurprising that the error-correcting approaches fail to correct many errors — the OVA classifiers, when well-tuned, are extremely accurate at using the information that is present to tell whether a point belongs to a given class or not. When the OVA system makes an error on $\mathbf{x}$, it is because, as far as our classification systems can tell, the point "looks" more like it belongs to the incorrect class than the correct one. We can reassign additional classes to the metaclasses however we like, resulting in the AVA, COMPLETE, DENSE and SPARSE schemes, but this does



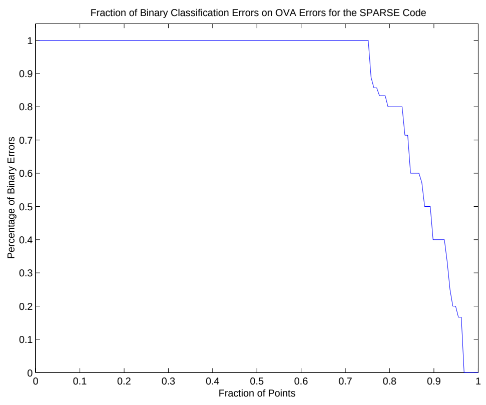
(a)



(b)



(c)



(d)

Figure 4-2: An analysis of the binary errors of the classifiers induced by the error-correcting coding schemes for the points which were OVA errors for the `satimage` data. See the text for a full explanation.

not seem to give any benefit, as these other schemes will make (most of) the same mistakes.

4.4 Theoretical Results

In this section, we explore two theoretical issues relating to multiclass classification. First, we derive leave-one-out bounds for multiclass classification with kernel methods (SVMs or RLSC, for example) as the underlying binary classifiers. Then, we consider some specific implications of the choice of RLSC as the underlying binary classifiers, amassing further arguments in support of using a simple one-vs-all classification

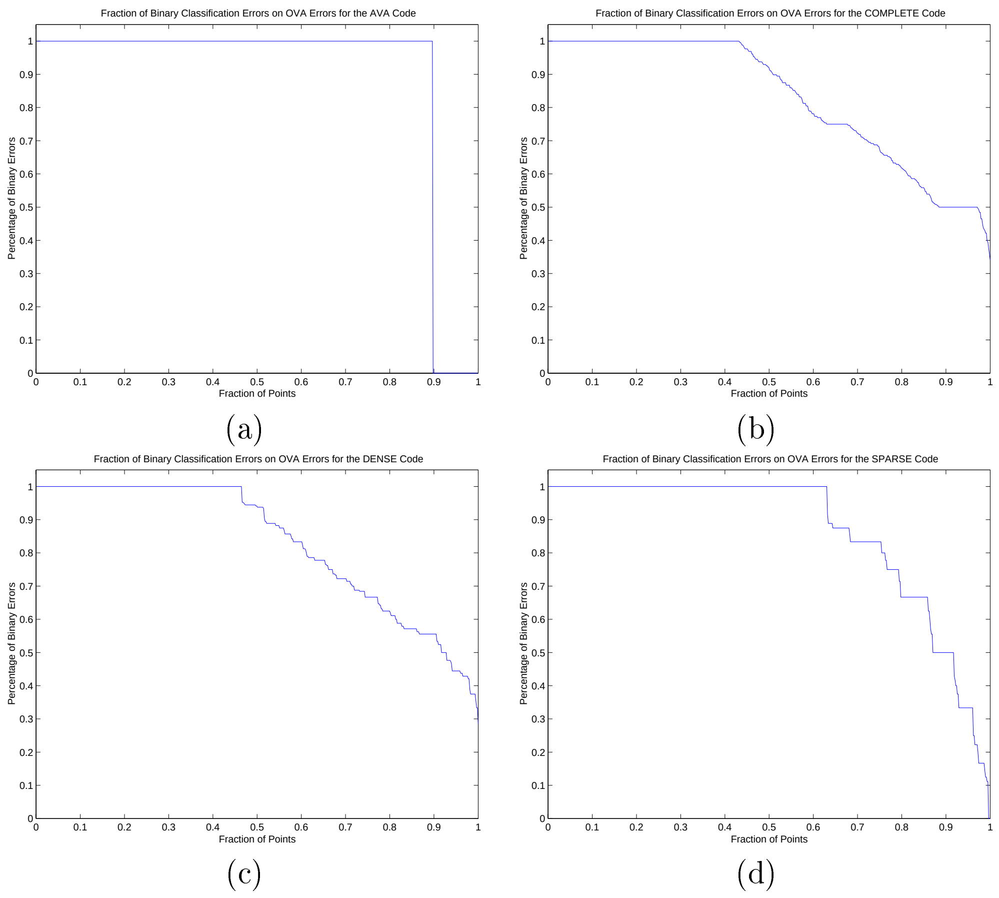


Figure 4-3: An analysis of the binary errors of the classifiers induced by the error-correcting coding schemes for the points which were OVA errors for the `yeast` data. See the text for a full explanation.

scheme.

4.4.1 Multiclass Leave-One-Out Bounds

Model validation is of primary importance in machine learning. The choice of an underlying binary classifier (with its associated kernel, kernel parameters and regularization parameters in the case of RLSC) and a scheme for combining the outputs of the binary classifiers into a multiclass prediction yields a multiclass classifier. We are interested in the performance of this classifier on new data. Often, we are not given enough data to hold out an independent validation set. In this case, a good approach is to the use the leave-one-out error: we train ℓ different classifiers, each on a subset of $\ell - 1$ training points, and test each classifier on the single "held out" training point. However, this procedure is often computationally expensive, and so it becomes useful to *bound* the number of errors that the leave-one-out procedure could produce. Throughout this section, we will assume for convenience that the underlying binary classifier is RLSC, but this is not restrictive; the family of classifiers considered by Jaakkola and Haussler ([54], also Section 3.3) could easily be used instead, with minor modifications to the arguments.

When the underlying classifiers are binary kernel classifiers that arise as the solution to Tikhonov minimization problems, leave-one-out bounds are easy to derive using the associated binary leave-one-bounds. Throughout this section, we will let f_i denote the ith binary classifier (corresponding to the ith column of the coding matrix), c_{ij} denote the coefficient associated with $\mathbf{x_j}$ in f_i, and f_i^j denote the classifier obtained when the jth example is removed from the training set. Also, we will use $[a, b]$ to denote the interval of numbers between $\min(a, b)$ and $\max(a, b)$, inclusive. For RLSC, we showed in Section 3.3 that for a training point $\mathbf{x_j}$ satisfying $y_j f_i(\mathbf{x_j}) < 1$, that $y_j f_i^j(\mathbf{x_j})$ satisfies:

$$f_i^j(\mathbf{x_j}) \in [f_i(\mathbf{x_j}), f_i(\mathbf{x_j}) - c_{ij} K(\mathbf{x_j}, \mathbf{x_j})], \tag{4.56}$$

If we look at the argument used to prove the above relation, we see that the same equation holds for points for which $y_j f_i^j(\mathbf{x_j}) > 1$.

Using these binary bounds, it is easy to derive leave-one-out bounds for multiclass classification. Recall that, in our current notation, the final multiclass function $f(\mathbf{x})$ is the class with minimal loss over the associated row of the coding matrix:

$$f(\mathbf{x}) = \arg \min_{r \in 1,\ldots,N} \sum_{i=1}^{F} L(M_{ri} f_i(\mathbf{x})) \tag{4.57}$$

Therefore, in order to see if a point $\mathbf{x_j}$ could possibly be a leave-one-out error, we calculate the "hypothetical" $f_i^j(\mathbf{x_j})$ in order to maximize the loss of y_j *relative to the other classes.* The idea is that for each classifier, we try to find the possible output (the set of possible outputs is given by the underlying binary leave-one-out bound) that will make y_j look least attractive to the multiclass system. For clarity, we will refer to this hypothetical worst-case set of outputs as $\hat{f_i^j}(\mathbf{x_j})$ (for $i \in 1, \ldots, F$). If we cannot construct a vector $\hat{\mathbf{f}}^{\mathbf{j}}(\mathbf{x_j})$ that the multiclass system will put into a class other than y_j, then $\mathbf{x_j}$ cannot be a multiclass leave-one-out error. In particular, (considering for the time being only a single point $\mathbf{x_j}$), for $r \neq y_j$, we define

$$\begin{aligned} \hat{D}_r &= \sum_{i=1}^{F} L(M_{ri}\hat{f_i^j}(\mathbf{x_j})) - \sum_{i=1}^{F} L(M_{y_j i}\hat{f_i^j}(\mathbf{x_j})) & (4.58) \\ &= \sum_{i=1}^{F} \Big(L(M_{ri}\hat{f_i^j}(\mathbf{x_j})) - L(M_{y_j i}\hat{f_i^j}(\mathbf{x_j})) \Big). & (4.59) \end{aligned}$$

A point is a possible leave-one-out error only if we can construct a vector $\hat{f^j}(\mathbf{x_j})$ so that $min_r \hat{D}_r < 0$. Looking at the second equation above, we see that we can optimize this by adjusting each $\hat{f_i^j}(\mathbf{x_j})$ individually; there are no "cross terms."

We begin with the specific case where each binary classifier uses all the data (the coding matrix contains no zero entries); this includes the OVA, COMPLETE, and DENSE codes considered by Allwein et al., as well as the BCH codes considered by Rennie and Rifkin [90], but not Allwein et al.'s AVA or SPARSE codes. For each individual classifier f^i, there are two cases to consider. If $y_j f_i(\mathbf{x_j}) < 1$, then, by setting $\hat{f_i^j}(\mathbf{x_j}) = f_i(\mathbf{x_j}) - c_{ij}K(\mathbf{x_j}, \mathbf{x_j})$, we will decrease D_r for all r for which $M_{ri} \neq M_{y_j i}$, leaving D_r unchanged for the remaining r. If $|c_{ij}K(\mathbf{x_j}, \mathbf{x_j})|$ is small and $y_j f_i(\mathbf{x_j}) > -1$, then we will have $L(y_j f_i(\mathbf{x_j})) < L(y_j f_i^j(\mathbf{x_j}))$ and $L(-y_j f_i(\mathbf{x_j})) > L(-y_j f_i^j(\mathbf{x_j}))$. If $|c_{ij}K(\mathbf{x_j}, \mathbf{x_j})|$ is sufficiently large or $y_j f_i(\mathbf{x_j}) < -1$, we will have $L(y_j f_i(\mathbf{x_j})) < L(y_j f_i^j(\mathbf{x_j}))$ and $L(-y_j f_i(\mathbf{x_j})) > L(-y_j f_i^j(\mathbf{x_j}))$, but the loss for the "correct" labelling will always increase *more* than the loss for the incorrect labelling.

On the other hand, if $y_j f_i(\mathbf{x_j}) > 1$, we choose $\hat{f}_i^j(\mathbf{x_j}) = f_i(\mathbf{x_j})$, leaving D_r unchanged; any other choice of $\hat{f}_i^j(\mathbf{x_j})$ would cause the loss for y_j to increase, but would cause the loss for $-y_j$ to increase even more.

Proceeding in this manner, we can construct the entire vector $\hat{f}^j(\mathbf{x_j})$, calculate the minimal D_r, and decide whether $\mathbf{x_j}$ could possibly be a leave-one-out error. Defining $D_r(\mathbf{x_j})$ to be the vector D_r associated with leaving point $\mathbf{x_j}$ out of the computation, we can bound the number of leave-one-out errors by

$$|\mathbf{x_j} : \min_{r \neq y_j} D_r(\mathbf{x_j}) < 0| \equiv |\mathbf{x_j} : \arg \min_{r \in 1,\ldots,N} \sum_{i=1}^{F} L(M_{ri}\hat{f}_j^i(\mathbf{x_j})) \neq y_j|. \tag{4.60}$$

For the specific case of an OVA scheme, the bound can be simplified somewhat. Recall that for the OVA scheme, we will choose

$$f(\mathbf{x}) = \arg\max_r f_r(\mathbf{x_i}). \tag{4.61}$$

Also, recall that the possible values of the leave-one-out classifiers lie in the range

$$f_i^j(\mathbf{x_j}) \in [f_i(\mathbf{x_j}), f_i(\mathbf{x_j}) - c_{ij}K(\mathbf{x_j}, \mathbf{x_j})]. \tag{4.62}$$

Therefore, a point is a possible leave-one-out bound only if

$$\max_{r \neq j} \max(f_r(\mathbf{x_j}), f_r(\mathbf{x_j}) - c_{rj}K(\mathbf{x_j}, \mathbf{x_j})) \geq \min(f_{y_j}(\mathbf{x_j}), f_{y_j}(\mathbf{x_j}) - c_{y_j j}K(\mathbf{x_j}, \mathbf{x_j})). \tag{4.63}$$

We can also extend the technique to deal with code matrices that contain zeros, such as the AVA or SPARSE matrices. For a given point $\mathbf{x_j}$, for classifiers i such that $M_{y_j i} \neq 0$, we proceed exactly as before, setting $\hat{f}_i^j(\mathbf{x_j})$ to maximize the difference between the loss for class y_j (and all other classes c such that $M_{y_j i} = M_{ci}$) and all classes c such that $M_{ci} = -y_j$; the losses for classes c for which $M_{ci} = 0$ will be unaffected. For those machines for which $M_{y_j i} = 0$, we take a different approach. We let I be the set of machines for which $M_{y_j i} = 0$. For each class c, we attempt to set $\hat{f}^i_j(\mathbf{x_j})$ to minimize the loss for *c for all machines in I simultaneously.* In so doing, the loss of y_j will be unaffected, since $M_{y_j i} = 0 \ \forall i \in I$. We perform this operation for each class c individually, and if we are able to achieve lower loss for any c than for y_j, $\mathbf{x_j}$ is a possible leave-one-out error.

Informal experiments indicate that these bounds are not especially tight. However, the bounds may be *predictive*, in that models that lead to lower values of the bound may have lower actual leave-one-out error (and lower generalization error) than models that give higher values of the bound. This is an interesting avenue for future research.

Another possible idea for future research is to consider the use of *partial computation* to improve the bound. In particular, because the bounds above are not tight, we will find that they predict many more leave-one-out errors then the number of errors that actually occur. If we were to actually compute the leave-one-out *values* for some of the binary classifiers, we would reduce the amount of uncertainty. It is possible that with a moderate amount of computation, we could reduce the bound on the number of leave-one-out errors substantially.

4.4.2 Multiclass Classification with RLSC

In this section, we make some simple yet powerful arguments relating to multiclass classification with RLSC as the underlying binary classifier. The basic insight is very simple. Recall that to solve an RLSC problem, we solve a linear system of the form

$$(K + \lambda \ell I)\mathbf{c} = \mathbf{y}. \tag{4.64}$$

In particular, this is a *linear* system, which means that the vector $\mathbf{c}$ is a linear function of the right hand side $\mathbf{y}$. Suppose we define the vector $\mathbf{y^i}$, where

$$y^i_j = \begin{cases} 1 & \text{if } y_j = i \\ 0 & \text{otherwise} \end{cases} \tag{4.65}$$

Now, suppose that we solve N RLSC problems of the form

$$(K + \lambda \ell I)\mathbf{c^i} = \mathbf{y^i}, \tag{4.66}$$

and denote the associated functions $f^{c_1}, \ldots, f^{c_N}$, Now, for *any* possible right hand side $\mathbf{y}^*$ for which the y_i and y_j are equal whenever $\mathbf{x_i}$ and $\mathbf{x_j}$ are in the same class, we can calculate the associated $\mathbf{c}$ vector from the $\mathbf{c^i}$, without solving a new RLSC system. In particular, if we let m_i ($i \in \{1, \ldots, m_N\}$) be the y value for points in class i, then the associated solution vector $\mathbf{c}$ is given by

$$\mathbf{c} = \sum_{i=1}^{N} c^i y_{m_i}. \tag{4.67}$$

For any code matrix M not containing any zeros, we can define the output of the coding system using only the entries of the coding matrix and the outputs of the underlying one-vs-all classifiers. In particular, we do not need to actually train the classifiers associated with the coding matrix. We can simply use the appropriate linear combination of the "underlying" one-vs-all classifiers:

$$\begin{aligned} f(\mathbf{x}) &= \arg\min_{r \in 1,\ldots,N} \sum_{i=1}^{F} L(M_{ri} f_i(\mathbf{x})) && (4.68) \\ &= \arg\min_{r \in 1,\ldots,N} \sum_{i=1}^{F} L(M_{ri} \sum_{j=1}^{N} M_{ji} f^j(\mathbf{x})) && (4.69) \\ &= \arg\min_{r \in 1,\ldots,N} \sum_{i=1}^{F} (M_{ri} - \sum_{j=1}^{N} M_{ji} f^j(\mathbf{x}))^2 && (4.70) \\ &= \arg\min_{r \in 1,\ldots,N} \sum_{i=1}^{F} (1 - M_{ri} \sum_{j=1}^{N} M_{ji} f^j(\mathbf{x}) + \sum_{j=1}^{N} \sum_{k=1}^{N} M_{ji} M_{ki} f^j(\mathbf{x}) f^k(\mathbf{x})) && (4.71) \end{aligned}$$

$$
\begin{aligned}
&= \arg\min_{r\in 1,\ldots,N} \sum_{i=1}^{F}(-M_{ri}\sum_{j=1}^{N} M_{ji}f^j(\mathbf{x})) && (4.72)\\
&= \arg\min_{r\in 1,\ldots,N} \sum_{i=1}^{F}(-f^r(\mathbf{x}) - M_{ri}\sum_{\substack{j=1\\ j\neq r}} M_{ji}f^j(\mathbf{x})) && (4.73)\\
&= \arg\min_{r\in 1,\ldots,N} -Ff^r(\mathbf{x}) - \sum_{i=1}^{F}\sum_{\substack{j=1\\ j\neq r}}^{N} M_{ri}M_{ji}f^j(\mathbf{x}) && (4.74)\\
&= \arg\min_{r\in 1,\ldots,N} -Ff^r(\mathbf{x}) - \sum_{\substack{j=1\\ j\neq r}}^{N} f^j(\mathbf{x})\sum_{i=1}^{F} M_{ri}M_{ji} && (4.75)\\
&= \arg\min_{r\in 1,\ldots,N} -Ff^r(\mathbf{x}) - \sum_{\substack{j=1\\ j\neq r}}^{N} C_{rj}f^j(\mathbf{x}) && (4.76)\\
&= \arg\min_{r\in 1,\ldots,N} -F\sum_{j=1}^{N} f^r(\mathbf{x}) + \sum_{\substack{j=1\\ j\neq r}}^{N}(F - C_{rj})f^j(\mathbf{x}) && (4.77)\\
&= \arg\min_{r\in 1,\ldots,N} \sum_{\substack{j=1\\ j\neq r}}^{N}(F - C_{rj})f^j(\mathbf{x}), && (4.78)
\end{aligned}
$$

where in the last three equations we define $C_{rj} \equiv \sum_{i=1}^{F} M_{ri}M_{ji}$. In terms of the coding matrix M, C_{rj} is the inner product of rows r and j of the matrix. We note that $F - C_{rj}$ is guaranteed to be positive under the basic assumption that no two rows of the coding matrix are identical.

We define a coding matrix to be *class-symmetric* if it treats all classes symmetrically: in particular, a coding-matrix is class symmetric if, whenever it contains a columns containing k 1's and $n-k$ -1's, all $\frac{N!}{k!(N-k)!}$ such columns are included. The OVA and COMPLETE schemes are class-symmetric, while the DENSE scheme is not (the AVA and SPARSE schemes include zeros in the coding matrix, and are not addressed by this analysis).

For *class-symmetric* schemes, C_{rj} is independent of the choice of r and j, and can be denoted simply as C^*. For these schemes,

$$
\begin{aligned}
f(\mathbf{x}) &= \arg\min_{r\in 1,\ldots,N}(F - C^*)\sum_{\substack{j=1\\ j\neq r}}^{N} f^j(\mathbf{x}) && (4.79)\\
&= \arg\max_{r\in 1,\ldots,N} f^r(\mathbf{x}), && (4.80)
\end{aligned}
$$

where the second equation follows by noting that $C^* < F$ (assuming the matrix contains both 1's and -1's), and the final equation follows by noting that the sum in the third equation will be minimized when the *largest* term, $\arg\max_{r \in 1,\dots,N} f^r(\mathbf{x})$, is not included. We have shown that *when RLSC is used as the underlying binary learner for class-symmetric coding matrices containing no zeros, the predictions generated are identical to those of the one-vs-all scheme.*

For matrices which are not class-symmetric, we cannot use the above argument directly. However, we believe that Equation 4.78 is still highly indicative. If the C_{rj} are all close to equal (which is generally the case for the matrices generated), then we can see that if $f^i(\mathbf{x})$ is substantially larger than $f^j(\mathbf{x})$ for $j \neq i$, that i will be chosen by the multiclass RLSC system.

Proceeding differently, we note that in the Allwein et al. framework, the coding matrices are generated without looking at the data. Therefore, we can view the assignment of the classes to the labels $\{1, \dots, N\}$ as random. Suppose we consider a single test point $\mathbf{x}$, and before training, we permute the labels of the classes via a permutation τ (a point which was "originally" in class i is placed into class $\tau(i)$). Given a permutation τ, the coding scheme will now classify a point according to

$$f_\tau(\mathbf{x}) = \arg\min_{r \in 1,\dots,N} \sum_{\substack{j=1 \\ j \neq r}}^{N} (F - C_{\tau(r)\tau(j)}) f^j(\mathbf{x}), \tag{4.81}$$

Suppose that $\arg\max_{r \in 1,\dots N} f^r(\mathbf{x}) = i$, and there exists a permutation τ_1 such that

$$f_{\tau_1}(\mathbf{x}) = j \neq i, \tag{4.82}$$

and the system makes a different prediction than the OVA scheme. Consider a new permutation τ_2, in which the roles of classes i and j are reversed: $\tau_2(i) = \tau_1(j), \tau_2(j) = \tau_1(i)$, and $\tau_2(k) = \tau_1(k)$ for $k \neq \{i, j\}$. For convenience, we define $D_{ij} \equiv (F - C_{ij})$, and define

$$\Omega_{\tau,r} = \sum_{\substack{k=1 \\ k \neq r}} D_{\tau(r)\tau(k)} f^k(\mathbf{x}). \tag{4.83}$$

Using this notation, and recalling that $D_{\tau_1(i)\tau_1(j)} = D_{\tau_2(i)\tau_2(j)}$, $D_{ij} > 0$ for all i and j, and $f^i(\mathbf{x}) > f^j(\mathbf{x})$ (by assumption), we find that:

$$
\begin{aligned}
\Omega_{\tau_2,i} &= \Omega_{\tau_1,j} - D_{\tau_1(j)\tau_1(i)} f^i(\mathbf{x}) + D_{\tau_2(i)\tau_2(j)} f^j(\mathbf{x}) && (4.84)\\
&= \Omega_{\tau_1,j} - D_{\tau_1(i)\tau_1(j)}(f^i(\mathbf{x}) - f^j(\mathbf{x})) && (4.85)\\
&= \Omega_{\tau_1,j} - D_{\tau_1(i)\tau_1(j)}(f^i(\mathbf{x}) - f^j(\mathbf{x})) && (4.86)\\
&< \Omega_{\tau_1,j} && (4.87)\\
&< \Omega_{\tau_1,i} && (4.88)\\
&= \Omega_{\tau_2,j} - D_{\tau_2(j)\tau_2(i)} f^i(\mathbf{x}) + D_{\tau_1(i)\tau_1(j)} f^j(\mathbf{x}) && (4.89)\\
&= \Omega_{\tau_2,j} - D_{\tau_1(i)\tau_1(j)}(f^i(\mathbf{x}) - f^j(\mathbf{x})) && (4.90)\\
&< \Omega_{\tau_2,j}. && (4.91)
\end{aligned}
$$

We see that if we replace τ_1 with τ_2, the scheme prefers class i to class j. For every permutation τ that chooses a class j over class i, we can construct a corresponding modified permutation that chooses class i over class j. It would be nice to extend this argument and show that under τ_2, i is the chosen class, but we have not so far been able to show this as of yet — it remains possible that there exists an example where a third class is chosen under τ_2. This is an open question for future research.

It is important to note that the above arguments *only* apply to schemes in which all the data is used for every classifier, and there are no zeros in the coding matrix. This includes the OVA, DENSE, and COMPLETE schemes of Allwein et al., but not their AVA or SPARSE schemes. However, in practice, we have not found the addition of zeros to the matrix to be helpful.

4.5 Computational Concerns

In this section, we briefly discuss some of the issues and tradeoffs involved in comparing the resource requirements of different approaches to multiclass classification. Unfortunately, we find that it is somewhat difficult to derive hard and fast rules, as the computational requirements are often dependent on the specific algorithm.

We begin by considering "naive implementations" of error-correcting coding methods in the framework of Allwein et al. [1]. We define an implementation as naive if we consider all of the induced binary problems as being entirely separate, and do not use the computations or results from one binary learner to help us with another binary learner. From this naive standpoint, an OVA system will take N times as long to train, and be N times as slow to test. An AVA system, assuming that the underlying binary classifiers are superlinear in the training set size, will actually be faster to train than an OVA system; although there are $O(N^2)$ classifiers as opposed to N for the OVA system, they use only $O(\frac{1}{N})$ as much data on average.[11]. However, the naive AVA system will be $O(N)$ slower at test time, as we will have to test each example on all $\frac{N(N-1}{2}$ classifiers. If N is large, even the storage of all the classifiers for an AVA scheme may become computationally prohibitive. For example, if we tried to run an AVA scheme on the `Sector105` text classification data set discussed in Chapter 3 and [90], where each classifier was a hyperplane in 55,197 dimensional space, simply storing all the classifiers would have required approximately 1.2 Gigabytes of memory, rather than the 23 Megabytes required by the OVA scheme. Similar analysis for other naive schemes, depending on the number of binary classifiers desired, are easy to perform.

Considering kernel methods in particular, in which we induce a number of binary learners of the form

$$f_i(\mathbf{x}) = \sum_{j=1}^{\ell} c_{ij} K(\mathbf{x_j}, \mathbf{x}), \tag{4.92}$$

there are two primary ways to improve substantially on the naive method, assuming that the same kernel is used for all the binary learners. The first is, during training, to cache the kernel products computed while training one binary learner for use while computing additional binary learners. The SvmFu implementation currently includes this strategy. If the data is high-dimensional, and the time to compute the kernel

[11]This fact was stated by Friedman in 1996 [39] and proved at great length by Fürnkranz in 2002 [42]

products dominates the training time, this can have a substantial effect on the total computation time. Unfortunately, because the different binary learners will in general have very different sets of support vectors, if the data set is very large, we will likely not be able to store the appropriate kernel products at each step, and we will perform a large amount of recomputation, mitigating the benefit.

The second key improvement for kernel machines is at testing time, where we find that a given kernel product is often used in many different binary learners $f_i(\mathbf{x})$. This strategy is also currently used by SvmFu. By caching these values, we can avoid recomputing them. If the data is very high-dimensional relative to the number of classes, this will tend to have a levelling effect on the testing time requirements of the different schemes, as the time to test a point under any of the schemes will be dominated by the time required to compute the kernel products. It is important to note that if the final classifiers are represented as linear hyperplanes, this advantage does not apply; in this case, an AVA scheme will be $O(N)$ times as expensive at test time (in both time and memory requirements) as an OVA scheme.

Finally, for the particular case where RLSC is the underlying binary learner and all the classifiers make use of all the data, there is a crucial additional speedup that can be used. Recall from Chapter 3 that training an RLSC system involves solving a system of the form

$$(K + \ell\lambda I)\mathbf{c} = \mathbf{y}. \tag{4.93}$$

Formally, if we derive $(K+\ell\lambda I)^{-1}$, which is an $O(\ell^3)$ operation, we can solve for a new right hand side $\mathbf{y}$ by performing a single $O(\ell^2)$ matrix multiplication. So if $\ell >> N$, the cost of building an OVA system is essentially equivalent to the cost of building a single binary classifier.[12] This technique was not used in the RLSC experiments described in this chapter; for the text classification task, in which the matrix is large and sparse, the Conjugate Gradient algorithm is the only effective method for solving

[12]Practically speaking, we would not invert $(K + \ell\lambda I)$, but would instead perform a Cholesky decomposition on $K + \ell\lambda I$, both for numerical stability and efficiency reasons. If we chose to use a subset of the data parametrized by m as in Section 3.7.2, we would perform a Cholesky decomposition on $(K_{m\ell}K_{\ell m} + K_{mm}\lambda\ell)$. In the linear case, where $(K + \ell\lambda I) = (AA^T + \ell\lambda I)$, we can make use of the Morrison-Woodbury formula.

RLSC, and for the tumor classification task, the algorithm ran very quickly because the data set was so small.

In conclusion, we find that a host of factors can affect the computational performance of a multiclass scheme, relating to the number of underlying binary classifiers and the specific algorithm used to train those classifiers. As an example, Crammer and Singer [24] claim that their single-machine multiclass system is *faster* than a one-vs-all system. Ignoring any other difficulties with their comparison (see Section 4.2.1), they assume that the competing SVM implementation is naive, and multiply the time required to solve a single SVM by the number of classes. The reader is advised to approach such comparisons with caution.

Chapter 5

Stability Theory and its Implications

In this brief, final chapter, we explore some extensions and implications of stability theory for learning algorithms. This form of stability theory (there are, of course, many things called "stability theory" in other fields) is a relativley recent development, popularized by the landmark 2002 paper of Elisseef and Bousquet [13]. The basic idea is to show that if an algorithm is *stable* in a certain sense — the function generated by the algorithms cannot change very much when we modify the training set slightly — then the algorithm will have good generalization error.

Section 5.1 presents the essential definitions and theorems. In Section 5.2, we explore the relationship between different approaches to proving generalization error bounds and different forms of regularization. In Section 5.3, we show that a simple form of bagging gives rise to stable algorithms, even when the original underlying learners are not stable. Finally, in Section 5.4, we prove stability bounds for multiclass classification.

5.1 Definitions and Theorems

We begin by introducing the necessary background, notation and definitions, then state the key theorems of stability theory. Given an input space $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ and an output space $y \in \mathcal{Y} \subseteq \mathbb{R}$, a training set of ℓ identically distributed independent samples

$$S = \{z_1 = (\mathbf{x}_1, y_1), ..., z_\ell = (\mathbf{x}_\ell, y_\ell)\}, \tag{5.1}$$

is drawn from an unknown distribution D over $(\mathcal{X} \times \mathcal{Y})$. We will often need to refer to a set

$$S^{i,u} = \{z_1, ..., z_{i-1}, u, z_{i+1}, \ldots, z_\ell\} = (S \setminus z_i) \cup \mathbf{u}. \tag{5.2}$$

in which the the ith point z_i is removed from S and replaced with an arbitrary new point u.

An algorithm is a mapping from a training set S to a function $f_S : \mathbf{x} \to y$. The hope is that f_S will be successful at mapping future $\mathbf{x}$ values to their associated y values. We will work with a *loss function* $V(f(\mathbf{x}), y)$, which gives the cost when we see $\mathbf{x}$ and predict $f(\mathbf{x})$, but the actual associated value is y. Abusing notation slightly, we also write this loss as $V(f, z)$. The empirical error of f_S is defined to be

$$\hat{R}(f_S) \equiv \frac{1}{\ell} \sum_{i=1}^{\ell} V(f_S(\mathbf{x_i}), y_i), \tag{5.3}$$

and the *generalization error* (or "true" error) is defined to be

$$R(f_S) = \int_{\mathcal{Z}} V(f_S, z) P(z)\ dz \equiv \mathbb{E}_z\ V(f_S, z). \tag{5.4}$$

We now introduce the basic notion of stability.

Definition 5.1.1 (Bousquet and Elisseeff, 2002)[13] *An algorithm has stability β with respect to the loss function V if*

$$\forall S, S^{i,u} \in \mathcal{Z}^\ell, \forall z \in \mathcal{Z}, \quad |V(f_S, z) - V(f_{S^{i,u}}, z)| \leq \beta.$$

In words, this notion of stability states that an algorithm has a stability of β, if, for any possible training set yielding f_S, and any possible single-point modification of that training set yielding $f_{S^{i,u}}$, the difference in the losses of f_S and $f_{S^{i,u}}$ at any possible new point z can be no more than β. Note that β will in general depend on

ℓ, so we could more precisely define stability as a function from the integers to the reals, but the usage will be clear from context. This definition of stability is known as *uniform* stability. It is a very restrictive condition, as it needs to hold on all possible training sets, even training sets that can only occur with probability 0. For β stable algortihms, Bousquet and Elisseef proved the following theorem:

Theorem 5.1.1 (Bousquet and Elisseeff, 2002) [13] *Let A be a β-stable learning algorithm satisfying $0 \leq V(f_S, z) \leq M$ for all training sets S and for all $z \in \mathcal{Z}$. For all $\varepsilon > 0$ and all $\ell \geq 1$,*

$$\mathbb{P}_S \left\{ |\hat{R}(f_S) - R(f_S)| > \varepsilon + 2\beta_\ell M \right\} \leq \exp\left(-\frac{2\ell\varepsilon^2}{(4\ell\beta_\ell + M)^2} \right).$$

A simple manipulation of the theorem gives us that with probability $1 - \delta$,

$$R(f_S) \leq \hat{R}(f_S) + 2\beta + (4\ell\beta + M)\sqrt{\frac{\ln \frac{1}{\delta}}{2\ell}}, \tag{5.5}$$

In general we are mainly interested in the case where $\beta_\ell = O\left(\frac{1}{\ell}\right)$; we will call an algorithm that possesses this property *strongly* β-stable. A key result of Elisseef and Bousquet was to show that Tikhonov regularization,

$$\min_{f \in \mathcal{H}} \frac{1}{\ell} \sum_{i=1}^{\ell} V(y_i, f(\mathbf{x}_i)) + \lambda ||f||_K^2, \tag{5.6}$$

is a strongly β-stable algorithm for a large choice of loss functions V. In particular, they define a σ-admissible loss function:

Definition 5.1.2 (Bousquet and Elisseeff, 2002)[13] *A loss function V is σ-admissible if it is convex with respect to its first argument and the following condition holds:*

$$\forall y_1, y_2 \in \mathcal{D}, \forall y' \in \mathcal{Y}, |V(y_1, y') - V(y_2, y')| \leq \sigma |y_1 - y_2|, \tag{5.7}$$

where $\mathcal{D} = \{y : \exists f \in \mathcal{F}, \exists x \in \mathcal{X}, f(\mathbf{x}) = y\}$ is the domain of the first argument of V.

This definition is closely related to the standard notion of a Lipschitz constant for a function f, which bounds how much $f(\mathbf{x})$ can change for a given change in $\mathbf{x}$; in particular, if our loss function V has a Lipshitz constant k, it is also σ-admissible with $\sigma = k$. However, this definition allows to work with functions (such as the square loss) that do not necessarily have a Lipschitz constant k. In simple terms, the definition states that a loss function is σ-admissible if the function has Lipshitz constant σ over all possible values of all possible functions generated by the algorithm.

Bousquet and Elisseef go on to show that Tikhonov regularization with a σ-admissible loss function is strongly β-stable with

$$\beta = \frac{\sigma^2 \kappa^2}{2\lambda\ell}, \tag{5.8}$$

where $\kappa \equiv \sup_{x \in \mathcal{X}} \sqrt{K(\mathbf{x}, \mathbf{x})}$.

5.2 Tikhonov and Ivanov Regularization

The use of stability to prove generalization bounds for learning algorithms is a relatively recent phenomenon. A much more widely used tactic is the *structural approach*, which can be used to prove bounds for Ivanov regularization, also referred to as *regularized empirical risk minimization*:

$$\begin{aligned} \min_{f \in \mathcal{H}} \quad & \tfrac{1}{\ell} \textstyle\sum_{i=1}^{\ell} V(f(\mathbf{x}_i), y_i) && (5.9) \\ \text{subject to}: \quad & ||f||_K^2 \leq C && (5.10) \end{aligned}$$

The constraint $||f||_K^2 \leq C$ limits the norm of the function in the RKHS. Functional analysis arguments can be used to show that the "space" $||f||_K^2 \leq C$ has finite covering number[1] [28]. In turn, this can be used to prove uniform convergence:

$$\mathbb{P}\left\{ \sup_{f: ||f||_K^2 \leq C} |R(f) - \hat{R}(f)| \geq \epsilon \right\} \leq \delta, \tag{5.11}$$

[1]Loosely speaking, this means that although there are infinitely many functions in the set $||f||_K^2 \leq C$, there are only finitely many "effectively different" functions.

where, given the covering number $\mathcal{N}$, we can express δ as

$$\delta = \mathcal{N} e^{\frac{\ell \epsilon^2}{C}}. \tag{5.12}$$

In a nutshell, the structural approach tells us that, if our allowable function set $||f||_K^2 \leq C$ has finite capacity (finite covering number), then, with high probability, as ℓ grows larger, *every function satisfying* $||f||_K^2 \leq C$ *behaves similarly on the training set* S *and on future data.* In particular, with δ defined as above, for every $\tau \in (0, 1)$, with probability $1 - \tau$ over the choice of training set S,

$$\sup_{f:||f||_K^2 \leq C} |R(f) - \hat{R}(f)| \leq O(\frac{1}{\sqrt{\ell}}) \tag{5.13}$$

The structural approaches matches very naturally with the regularized empirical risk minimization algorithm — because we have a bound on the difference between the empirical risk and the true risk for every function in the set, and we are interested in minimizing true risk, it makes sense to choose the function in the set with the smallest empirical risk. Vapnik [115, 116] also suggests the idea of *structural risk minimization*, where a series of nested structures are constructed by choosing $C_1 < C_2 < \ldots$. For each C_i, we will have a bound on the difference between empirical risk and true risk (the bounds will get weaker as C_i increases), and we will have a (measured) empirical risk $\hat{R}_{C_i}$ (which will decrease as C_i increases), and we choose the C_i that gives the best bound (the bounds get slightly weaker when we consider multiple structures in this fashion).

Although most of the *theory* of generalization bounds (prior to the use of algorithmic stability) was based on Ivanov regularization, both the Support Vector Machine and the Regularized Least Squares Classifier discussed in this thesis are Tikhonov Regularization algorithms. The algorithmic stability bounds apply directly to, and give primacy to, the Tikhonov form. Additionally, we will see below that Ivanov regularization is *not* necessarily strongly β-stable.

There exist "equivalences" between the Tikhonov and Ivanov forms [117]. In particular, suppose f^* is the unique solution to a Tikhonov minimization problem. Then, it is clear that f^* is also the solution to an Ivanov minimization problem with $C = ||f^*||_K^2$. Conversely, suppose that f^* is the unique solution to an Ivanov problem. Then (under some "smoothness" assumptions) there will exist a λ that makes f^* the solution of an associated Tikhonov minimization problem. It is crucial to remember that these equivalences are *a posteriori* — we cannot use them to *solve* a Tikhonov problem by solving a single Ivanov problem or vice versa. Also extremely important is that if the constraint is not tight in the solution to an Ivanov problem (i.e., $||f^*||_K^2 \neq C$), then $\lambda = 0$ in the associated Tikhonov problem, and in both sets of problems, we simply find the function in the RKHS that minimizes the training error; the Ivanov regularization has no effect, and therefore the associated Tikhonov problem will have no regularization. From this perspective, even though there are equivalences between the forms, they are fundamentally different: in the Ivanov form, less smooth functions cost no more than smoother functions, until the limit C is reached, whereas in the Tikhonov form, we pay continuously as we go. This intuition is reflected in the fact that the Tikhonov approach has much stronger algorithmic stability properties than the Ivanov approach, as we will see below.

We note that the definition of σ-admissibility above makes explicit reference to a "space of functions" $\mathcal{F}$ that the algorithm works in, although it is not immediately obvious that such a space exists. However, for Tikhonov regularization, we *can* found the space, as follows. Let f^* be the optimal solution to a Tikhonov regularization problem. Let $\mathbf{0}$ denote the constant all-zero function (this function is in every RKHS K, and has norm zero). Assuming that the loss function $V(f(\mathbf{x}_i), y_i)$ is always non-negative,

$$\begin{aligned}
\lambda||f^*||_K^2 &\leq \frac{1}{\ell}\sum_{i=1}^{\ell} V(f^*(\mathbf{x}_i), y_i) + \lambda||f^*||_K^2 &\quad (5.14)\\
&\leq \frac{1}{\ell}\sum_{i=1}^{\ell} V(\mathbf{0}(\mathbf{x_i}), y_i) + \lambda||\mathbf{0}||_K^2 &\quad (5.15)\\
&= \frac{1}{\ell}\sum_{i=1}^{\ell} V(0, y_i) &\quad (5.16)
\end{aligned}$$

$$\implies ||f^*||_K^2 \leq \frac{\sup_{y\in\mathcal{Y}} V(0,y)}{\lambda}, \tag{5.17}$$

which gives a bound on the RKHS norm of f^*. Defining $B \equiv \sup_{y\in\mathcal{Y}} V(0,y)$, the largest possible penalty we could pay when we "predict" $f(\mathbf{x}) = 0$, we see that for Tikhonov regularization, $||f^*||_K^2 \leq \frac{B}{\lambda}$. This immediately allows us to state that the Tikhonov minimization algorithm is working in the bounded space $||f^*||_K^2 \leq \frac{B}{\lambda}$, which in turn immediately allows us to bound the covering number of the space and obtain the associated generalization bounds in the form of 5.13. We reiterate that bounds in the form of 5.13 depend only on the size (covering number) of the space of functions, and *not* on the algorithm used to choose a function from this space. In particular, if we consider a Tikhonov problem with regularization constant λ and an "associated" Ivanov problem with $C = \frac{B}{\lambda}$, the problems can be viewed as operating in the *same* space of functions and we can obtain an *identical* bound of the form 5.13. In practice, we might expect the Ivanov approach to lead to better bounds because it would in general find a function with lower training error — for a given *fixed* bound on the RKHS norm, it makes more sense to find the function that *minimizes* the empirical risk in the constrained space, as the Ivanov algorithm does, rather than take a pay-as-you-go approach a la Tikonov. Using $\frac{B}{\lambda}$ to bound the size of the space in this manner, and thereby prove *structural* bounds for Tikhonov regularization, is obvious in retrospect, and does not seem to lead to better bounds than the Ivanov approach; however, we were unable to find any previous mention of this approach in the literature.

Using the notion of σ-admissibility, we can derive stability results for various instantiations of Tikhonov regularization.[2] For the case of the L_1 loss $V(f(\mathbf{x}), y) = |f(\mathbf{x}) - y|$ (Figure 5-1), or the ϵ-insensitive loss $V(f(\mathbf{x}), y) = (|f(\mathbf{x}) - y| - \epsilon)_+$ (Figure 5-2), it is clear that they are both Lipschitz with Lipschitz constant 1, and therefore they are both σ-admissible with $\sigma = 1$. This implies that for both these loss functions,

[2]The theory as stated above holds only for *regression* algorithms; for classification, where the loss is zero or one, an extension is required. This is discussed in Section 5.4, where we present this extension and extend it further in order to prove stability bounds for multiclass classification.

Tikhonov regularization is strongly β-stable with

$$\beta = \frac{\kappa^2}{2\lambda\ell}. \tag{5.18}$$

For the case of the square loss $V(f(\mathbf{x}), y) = (y - f(\mathbf{x}))^2$ (Figure 5-3), we must be careful. The difficulty is that the square loss is not Lipschitz for any Lipschitz constant; it grows steeper and steeper as $(y - f(\mathbf{x}))$ increases. However, we can still derive a σ-admissibility constant, as follows. We recall that the Tikhonov algorithm can be viewed as operating in the space $||f||_K^2 \leq \frac{B}{\lambda}$. However, this also implies a bound on the L_∞ norm of f (see Appendix A):

$$||f||_\infty \leq \kappa\sqrt{\frac{B}{\lambda}}. \tag{5.19}$$

This in turn implies that that $\kappa\sqrt{\frac{B}{\lambda}}$ is the *largest* absolute value that can be generated by any function satisfying $||f||_K^2 \leq \frac{B}{\lambda}$. Now we can bound the loss at any point $y \in \mathcal{Y}$ as

$$M \equiv (\sqrt{\frac{B}{\lambda}} + sup_{y \in \mathcal{Y}}|y|)^2 \tag{5.20}$$

which in turn implies that the loss function is σ-admissible with $\sigma = M$.[3] Note that the smoothness of the loss function, and in turn the quality of the bound, now depends on λ

We now demonstrate by means of a counterexample that Ivanov regularization does not enjoy the same strong stability properties as Tikhonov regularization. For both examples, we consider the $\mathcal{X}$ and and $\mathcal{Y}$ domains to be the closed interval $[0, 1]$, and we consider our RKHS to be the set of lines passing through the origin. Our Ivanov constraint is that we consider functions with slope less than or equal to one.

We recall that an algorithm is strongly β-stable only if, for *every* possible training set generating a function f_S, and *every* possible one point modification of that set

[3]We note in passing that the Bousquet and Elisseef paper [13], which states that the squares loss is $2B$-admissible, is incorrect.

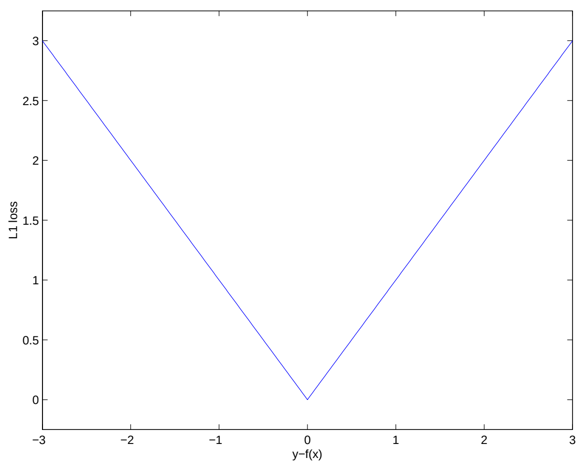


Figure 5-1: The hinge loss $V(f(\mathbf{x}), y) = |f(\mathbf{x}) - y|$.

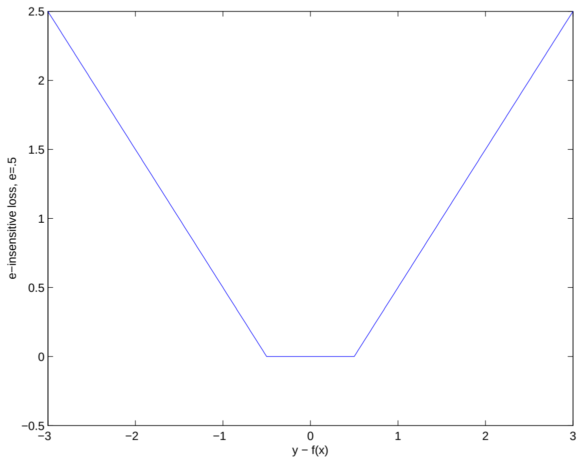


Figure 5-2: The ϵ-insensitive loss $V(f(\mathbf{x}), y) = (|f(\mathbf{x}) - y| - \epsilon)_+$

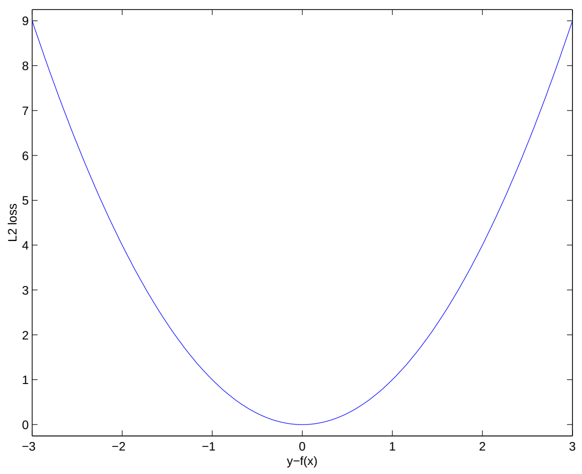


Figure 5-3: The square loss $V(f(\mathbf{x}), y) = (y - f(\mathbf{x}))^2$.

generating a function $f_{S^{i,u}}$, we can state that

$$sup_{\mathbf{x}\in\mathcal{X},y\in\mathcal{Y}}||V(f_S(\mathbf{x}),y) - V(f_{S^{i,u}}(\mathbf{x}),y)|| \leq O(\frac{1}{\ell}). \tag{5.21}$$

We consider the following (admittedly pathological, but not disallowed) case. We generate a training set S where $\ell - 1$ points are placed directly at the origin, and the ℓth point is placed at $(1,0)$. We let V be the square loss (a number of different losses could be used in this example). Now we modify S by deleting the ℓth point and replacing it with a point at $(1,1)$. It is clear that f_S will be a line passing through the origin with slope 0, and that $f_{S^{i,u}}$ will be a line passing through the origin with slope 1, *independent of ℓ*. Therefore, the stability does not (necessarily) increase at all as we increase ℓ, and we can conclude that in the worst case, Ivanov regularization has only *constant* stability. This highlights the difference between Tikhonov and Ivanov regularization. Tikhonov regularization, which forces us to pay a price for the RKHS norm "as we go", is a *stronger* form of regularization than Ivanov regularization, where we pay nothing for the RKHS norm until we reach a hard limit.

In very recent work [65, 64, 63], Kutin and Niyogi explore a number of different *probabilistic* generalizations of stability. These notions of stability are somewhat weaker than the one used by Bousquet and Elisseef, in that they assert (in various ways) that the change when a single example is deleted is *usually*, but not always small. An interesting problem that is possibly solved by this work is to characterize the strongest form(s) of stability satisfied by Ivanov regularization. Many of these weaker forms of stability do lead to bounds which behave asymptotically as $O(\frac{1}{\sqrt{\ell}})$, like the Bousquet and Elisseef bounds, but the derivations and the bounds themselves are extremely complicated. In any case, it is clear that Tikhonov regularization *does* satisfy the extremely stringent stability definition of Bousquet and Elisseef, whereas Ivanov regularization does not, indicating that (at least in some sense) Tikhonov regularization is a more stable form of regularization than Ivanov.

5.3 Bagging Regularizes

We begin by introducing a more primary notion of stability.

Definition 5.3.1 *An algorithm has α-stability with respect if*

$$\forall S, S^{i,u} \in \mathcal{Z}^\ell, \forall x \in \mathcal{X}, \quad |f_S(\mathbf{x}) - f_{S^{i,u}}(\mathbf{x})| \leq \alpha. \tag{5.22}$$

Whereas β stability refers to closeness of the functions $V(f_S, \cdot)$ and $V(f_S^{i,u}, \cdot)$ (i.e., closeness of f_S and $f_S^{i,u}$ after being "passed through" the loss function V), α stability refers directly to the proximity of f_S and $f_S^{i,u}$.[4] It is clear that, for a σ-admissibile loss function, strong α-stability implies strong β-stability. The converse is *in general* false: stability with respect to the loss function does not imply stability of the functions, even for σ-admissible loss functions. For example, we can consider the square loss, and the case where $f_1(x) = y + K$ and $f_2(x) = y - K$. The loss of the two functions is identical, but their L_∞ norms differ by $2K$. However, published proofs of β-stability of algorithms (such as the ones in the Bousquet and Elisseef paper) first prove α-stability (not explicitly defined as such) of the algorithms then use the σ-admissibility of the loss function to obtain the β-stability result, which is the one that is necessary for proving generalization bounds.

We now turn to the notion of bagging, or averaging, of regressors. The point of the above discussion is that we can easily prove bounds on a simple bagging scheme by using α-stability directly, because α-stability can be "averaged" in a way that β-stability cannot. In particular, consider the following simple bagging scheme. The training set of size ℓ is divided into J disjoint subsets, each of size p; for simplicity we assume ℓ is a multiple of p. Each data point will appear in exactly one of the J subsets. As we increase the size of the training set, we keep p fixed and let J increase, which implies that

$$J = \frac{\ell}{p}. \tag{5.23}$$

[4]This notion of stability is also called "classification stability" by Bousquet and Elisseef, but they do not give much emphasis to this form of stability, viewing it as a mathematical convenience for extending stability arguments from regression to classification algorithms.

We assume that we are training an algorithm with a σ-admissible loss function. For each of the J subsets, we obtain a function f_i. Our base learning algorithm has α-stability α_p (as it operates on a subset of the data of size p), and β-stability $\beta_p = \sigma\alpha_p$. Note that we do not assume that the base learning algorithm is strongly stable; the stability α_p may be as bad as *constant* in p. Now, we consider the "bagged" function

$$f(\mathbf{x}) = \frac{1}{J}\sum_{i=1}^{J} f_i(\mathbf{x}) \tag{5.24}$$

We analyze the α-stability of this scheme. When we remove a single point from the training set and replace it, only one of the J subsets, say subset d, is affected. We obtain a new function f_d^N satisfying $||f_d - f_d^N||_\infty \leq \alpha_p$. The new combined function f^N satisfies

$$\begin{aligned} ||f - f^N||_\infty &= ||\frac{1}{J}(f_d - f_d^N)||_\infty && (5.25)\\ &\leq \frac{1}{J}\alpha_p && (5.26)\\ &= \frac{p}{\ell}\alpha_p. && (5.27)\end{aligned}$$

This simple result shows that if we average together regressors which are not necessarily strongly stable, we obtain a strongly stable ensemble of regressors, with the same asymptotic rate of convergence as Tikhonov regularization. In exchange for weaker stability, it is also possible to allow the subset size p to increase with ℓ; the details are straightforward. We must also note that although this argument indicates that Tikhonov regularization and bagging of non-strongly stable classifiers will have approximately the same differential between training and generalization error, this argument tells us nothing about the *training error* of either scheme, and so does not directly tell us which algorithm is better. (Intuitively, we expect training a single Tikhonov problem on the entire dataset, which explicitly tries (as one of its goals) to *minimize* the training error, will of course provide a lower training error than simply averaging together very many classifiers trained on small fixed-size samples.)

It is also worth noting that if the original base regressors are already strongly stable (for instance, they are found by solving Tikhonov regularization problems), then bagging them in this manner does not increase the stability. This is somewhat interesting, given that bagging (and boosting) algorithms have rarely been shown to increase accuracy when the underlying base learners are powerful algorithms such as Tikhonov regularization.

5.4 Stability for Multiclass Classification

In this section, we derive stability-based generalization bounds on multiclass classification where the underlying binary classifiers are themselves stable learners, for example the solutions of Tikhonov minimization problems. The idea is a simple generalization of the approach used to prove stability bounds for binary classification, which is itself a generalization of the approach used to prove stability bounds for regression. The basic difficulty with using stability to prove generalization bounds for classification is that the L_0 loss

$$V(f(\mathbf{x}), y) = \begin{cases} 0 & \text{if } \mathrm{sign}(f(\mathbf{x})) = y \\ 1 & \text{otherwise} \end{cases} \tag{5.28}$$

is not σ-admissible for any σ (it is also non-convex, which is why we do not minimize it directly in learning schemes for classification). The basic solution is to introduce a new loss function L_N (L_N will be specified shortly) which is σ-admissible and which upper bounds the L_0 loss. Stability will allow us to relate the empirical and true "risk" under this new loss function, which we can in turn use to upper bound the true risk under the actual L_0 loss.

We now turn to an examination of multiclass classification with an error-correcting code matrix M. For simplicity, we consider only the case where the coding matrix M contains no zero entries, and we will as usual pay special attention to the simple one-vs-all case. Translating the margin-based notation $L(yf(\mathbf{x}))$ into the stability notation $V(f(\mathbf{x}), y)$, we recall that a multiclass classifier classifies according to

$$f(\mathbf{x}) = \arg \min_{r \in 1,\dots N} \sum_{i=1}^{F} V_b(f_i(\mathbf{x}), M_{ri}) \tag{5.29}$$

where M_{ri} is 1 or -1 depending on whether machine i places class r in the positive or negative "metaclass", and V_b denotes the loss function that is minimized by the underlying binary classifier. We will reserve V to denote the multiclass $0-1$ loss:

$$V(f(\mathbf{x}), y) = \begin{cases} 0 & \text{if } \arg\min_{r \in 1,\dots N} \sum_{i=1}^{F} V_b(f_i(\mathbf{x}), M_{ri}) = y \\ 1 & \text{otherwise} \end{cases} \tag{5.30}$$

For notational convenience, we define the total loss for a class at a given point:

$$L(\mathbf{x}, r) = \sum_{i=1}^{F} V_b(f_i(\mathbf{x}), M_{ri}). \tag{5.31}$$

We can now define the following multiclass loss function (parametrized over $\gamma > 0$):

$$V_\gamma(f(\mathbf{x}), y) = \begin{cases} 0 & \text{if } L(\mathbf{x}, y) < \min_{r \neq y} L(\mathbf{x}, r) \\ \frac{L(\mathbf{x}, y) - \min_{r \neq y} L(\mathbf{x}, r)}{\gamma} & \text{if } 0 \leq L(\mathbf{x}, y) - \min_{r \neq y} L(\mathbf{x}, r) \leq \gamma \\ 1 & \text{if } L(\mathbf{x}, y) - \min_{r \neq y} L(\mathbf{x}, r) > \gamma \end{cases} \tag{5.32}$$

It is clear that for any $\gamma > 0$, $V_\gamma(f(\mathbf{x}), y) \geq V(f(\mathbf{x}), y)$. Additionally, V_γ will be a σ-admissible (actually a Lipshitz) loss function, where σ will depend on the properties of the coding matrix M. We begin by considering a single machine f_i. When we remove and replace a single training point, it is clear that $V_\gamma(f(\mathbf{x}), y)$ will change by at most $\frac{2\beta_b}{\gamma}$, where β_b is the stability of the underlying binary learner: $L(\mathbf{x}, y)$ might increase by β_b, and $\min_{r \neq y} L(\mathbf{x}, y)$ might simultaneously decrease by β_b. So a naive bound on the stability (with respect to V_γ) of the multiclass system is $\frac{2F\beta_b}{\gamma}$. This bound can be strengthened by defining G to be the maximum number of positions in which any two rows of the coding matrix differ:

$$G = \max_{y_1, y_2 \in 1,\dots,N} \{i : M_{iy_1} \neq M_{iy_2}\} \tag{5.33}$$

It is clear that when we modify the training set by removing and replacing a data point, at most G of the classifiers can contribute to changing V_γ: for each class $r \neq y$, the remaining $F - G$ classifiers wil put r and y into the same metaclass, and therefore contribute nothing to V_γ. We conclude that we can actually bound the stability of V_γ by $\frac{2G\beta_b}{\gamma}$. Noting that $V_\gamma(f(\mathbf{x}), y)$ upper bounds $V(f(\mathbf{x}), y)$, and also that $0 \leq V_\gamma(f(\mathbf{x}), y) \leq 1$, we can use Equation 5.5 to derive the following bound on multiclass classification:

$$R \leq R_\gamma \leq \hat{R}_\gamma + 2\frac{2G\beta_b}{\gamma} + (4\ell\frac{2G\beta_b}{\gamma} + 1)\sqrt{\frac{\ln\frac{1}{\delta}}{2\ell}}. \tag{5.34}$$

It is also possible, using techniques developed by Bartlett [5] and used by Bousquet and Elisseeff in the context of bounds for binary classification, to extend this to a bound that holds simultaneously for all γ in the range $(0, B]$; the proof in the Appendix of [13] is valid without modifications.

We note that in the specific case of one-vs-all multiclass classification, $G = 2$. For other schemes (remembering that we only consider the case where the matrix contains no zero entries), $G > 2$, implying that from this perspective, stability theory actually yields stronger bounds for one-vs-all classification than for other schemes. We should be careful not to read very much into this; this worst-case analysis assumes that the classifiers are complete decorrelated in the worst possible way, and as we saw in Chapter 4, different binary classifiers for the same problem tend to be very highly correlated.

NEED THING ABOUT UNREGULARIZED BIAS TERM!!!

Appendix A

Reproducing Kernel Hilbert Spaces

In this appendix we present and discuss key facts about Reproducing Kernel Hilbert Spaces. More in-depth discussions can be found in a variety of sources such as [3, 120, 94, 76].

A Reproducing Kernel Hilbert Space (RKHS) is a Hilbert space of real-valued[1] functions on a compact domain $\mathcal{X}$ with the property that for all $x \in \mathcal{X}$ the evaluation functional $\mathcal{F}_x$ is a bounded linear functional:

$$\exists M \text{ s.t.} \forall x \in \mathcal{X},\ |\mathcal{F}_x[f]| = |f(x)| \leq M||f|| \tag{A.1}$$

If $\mathcal{H}$ is an RKHS, then, by the Riesz representation theorem, for each $x \in \mathcal{X}$, there exists a function $K_x \in \mathcal{H}$ satisfying the *reproducing property*:

$$\mathcal{F}_x[f] \equiv f(x) =< f(.), K_x(.) >_K \ \ \forall f \in \mathcal{H} \tag{A.2}$$

In other words, the reproducing property states that the inner product of a function f and the function K_x is $f(x)$ in the RKHS. Defining K as a two-variable function

$$K(x, y) = K_x(y), \tag{A.3}$$

[1]the theory extends easily to the complex-valued case, with a little more care in the use of complex conjugation.

it can be shown that K is a postive definite function. K is known as the *kernel* of the RKHS. It can be shown that there is one-to-one correspondance between positive definite kernel functions K and Reproducing Kernel Hilbert Spaces $\mathcal{H}$, and we will henceforth refer to an RKHS with its associated kernel as $\mathcal{H}_K$.

Mercer's theorem tells us that a kernel function K has the expansion

$$K(x,y) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x)\phi_i(y), \tag{A.4}$$

where the λ's and ϕ's are the eigenvalues and orthonormal eigenfunctions of the integral equation

$$\int_X K(x,y)\phi(x)dx = \lambda\phi(y). \tag{A.5}$$

If there are infinitely many solutions to the above equation, the corresponding RKHS is dense in L_2. The space $\mathcal{H}_K$ can also be represented as

$$\mathcal{H}_K = \{f | f(x) = \sum_{i=1}^{\infty} d_i^f \phi_i(x)\}. \tag{A.6}$$

$$f(x) = \sum_{i=1}^{\infty} d_i \phi_i(x), \tag{A.7}$$

Using this expansion, it is easy to see that

$$\begin{aligned} < f, g >_K &= \left(\sum_{i=1}^{\infty} d_i^f \phi_i(x), \sum_{j=1}^{\infty} d_j^g \phi_j(x) \right)_K && \text{(A.8)} \\ &= \sum_{i=1}^{\infty} \frac{d_i^f d_i^g}{\lambda_i} && \text{(A.9)} \end{aligned}$$

and

$$\begin{aligned} ||f||_K^2 &= \left(\sum_{i=1}^{\infty} d_i^f \phi_i(x), \sum_{j=1}^{\infty} d_j^f \phi_j(x) \right)_K && \text{(A.10)} \\ &= \sum_{i=1}^{\infty} \frac{(d_i^f)^2}{\lambda_i} && \text{(A.11)} \end{aligned}$$

The RKHS kernel K is often interpreted as performing a mapping of the data into a high- or even infinite-dimensional *feature space*:

$$\Phi : x \to \Phi(x) \equiv \{\sqrt{\lambda_1}\phi_1(x), \sqrt{\lambda_2}\phi_2(x), \ldots\}. \tag{A.12}$$

Using this definition, the inner product between $\Phi(x)$ and $\Phi(y)$ in feature space is $K(x, y)$, via an application of Mercer's Theorem.

If we consider sets of the form

$$S_A \equiv \{f : f \in \mathcal{H}_K,\ ||f||_K \leq A\}, \tag{A.13}$$

the parameter A serves to control the size of the hypothesis space. As A gets larger, S_A gets larger: its covering number grows, and it contains functions which are less smooth. This has well-known implications for learning theory, as discussed in Chapter 5.

There is also an important relation between the L_∞ norm of a function and its RKHS norm. If we restrict our input space to a set $\mathcal{X}$ such that

$$\kappa \equiv \sup_{x \in \mathcal{X}} \sqrt{K(\mathbf{x}, \mathbf{x})}, \tag{A.14}$$

exists, then it can be shown that

$$\sup_{x \in calX} f(x) \equiv ||f||_\infty \leq \kappa ||f||_K. \tag{A.15}$$

Appendix B

The Representer Theorem

In this appendix we discuss and prove the so-called Representer Theorem, showing that under rather general conditions, the form of the solution f^* to the regularization problem

$$\min_{f \in \mathcal{H}} \frac{1}{\ell} \sum_{i=1}^{\ell} V(y_i, f(\mathbf{x}_i)) + \lambda ||f||_K^2. \tag{B.1}$$

can be written as

$$f^*(\mathbf{x}) = \sum_{i=1}^{\ell} c_i K(\mathbf{x}, \mathbf{x}_i). \tag{B.2}$$

In other words, the solution to the regularization problem can be expressed as a vector (hyperplane) in feature space corresponding to a certain linear combination of the projections of the training examples into feature space. This theorem underlies all the algorithms discussed in thsis paper, because it implies that instead of having to search an entire RKHS $\mathcal{H}$ to find the solution to a Tikhonov regularization problem, we need only search an ℓ-dimensional subspace parametrized by the coefficients c_i. In other words, the inherently infinite-dimensional problem of finding the best function in an RKHS is transformed into a tractable problem of finding ℓ parameters c_i.

The proof we present here is due to Schölkopf et al. [97], although the proof is "implicit" in much earlier work of Wahba [119].

Recall from Appendix A that we define $\Phi(x_i)$ to be the projection of x_i into a high (possibly infinite) dimensional feature space, and that all functions in the RKHS can be viewed as "hyperplanes" in thisis feature space. From this viewpoint, the Representer Theorem states that the solution to the Tikhonov problem has the form

$$f^* = \sum_{i=1}^{\ell} c_i \Phi(\mathbf{x_i}). \tag{B.3}$$

The proof proceeds by contradiction. Suppose that we *cannot* express f^* in this form. Then we can express it as

$$f^* = \sum_{i=1}^{\ell} c_i \Phi(\mathbf{x_i}) + v, \tag{B.4}$$

where v is orthogonal to $\Phi(\mathbf{x_1}), \ldots, \Phi(\mathbf{x_\ell})$. Define

$$\hat{f} \equiv \sum_{i=}^{\ell} c_i \Phi(\mathbf{x_i}) = f^* - v \tag{B.5}$$

Consider the application of this function f^* to an arbitrary *training point* $\mathbf{x_j}$:

$$\begin{aligned} f^*(\mathbf{x_j}) &= (\sum_{i=1}^{\ell} c_i \Phi(\mathbf{x_i}) + v, \Phi(\mathbf{x_j})) \qquad (B.6) \\ &= (\sum_{i=1}^{\ell} c_i \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j})) \qquad (B.7) \\ &= \hat{f}(\mathbf{x_j}), \qquad (B.8) \end{aligned}$$

by our orthogonality condition. But this shows that $f^*(\mathbf{x_j})$ is independent of v — for any training point, the choice of v will not affect the value of f^* at that point. (We are not stating that f^* and $f^* - v$ are the same function, only that they have the same value at all of the training points.) In particular,

$$\sum_{i=1}^{\ell} V(f^*(\mathbf{x_i}), y_i) = \sum_{i=1}^{\ell} V(\hat{f}(\mathbf{x_i}), y_i). \tag{B.9}$$

In addition,

$$\begin{aligned} ||f^*||_K^2 &= ||\sum_{i=1}^{\ell} c_i \Phi(\mathbf{x_i}) + v||_K^2 \qquad (B.10) \\ &= ||\sum_{i=1}^{\ell} c_i \Phi(\mathbf{x_i})||_K^2 + ||v||_K^2 \qquad (B.11) \end{aligned}$$

$$\geq \ ||\sum_{i=1}^{\ell} c_i \Phi(\mathbf{x_i})||_K^2 \tag{B.12}$$

$$= \ ||\hat{f}||_K^2, \tag{B.13}$$

and the function $\hat{f}$ has RKHS norm strictly less than f^* if $v \neq 0$. Therefore, by choosing $v = 0$, we can reduce the RKHS norm of f^* without affecting the loss at any of the training points, thereby obtaining a better solution to the Tikhonov regularization problem. This contradicts the notion that f^* was optimal; at any optimal solution, $v = 0$, and the solution has the desired form.

The above proof is extremely general (and can easily be made even more general, allowing for a loss function that depends on all the function values simultaneously), but is nonconstructive: it tells us the *form* of the solution, but does not tell us how to actually find the c_i. In the specific case of the square loss, where $V(f(\mathbf{x}), y) = (y - f(\mathbf{x})^2)$, Girosi [47] includes a constructive proof of the form of the c_i; such a proof is very similar to the derivation of the RLSC algorithm given in Chapter 3, although there we begin by assuming the form of the solution.

Using the above equations, it is easy to show that if we have a function in the form:

$$f = \sum_{i=a1}^{\ell} c_i \Phi(\mathbf{x_i}), \tag{B.14}$$

then

$$||f||_K^2 = \mathbf{c}^T K \mathbf{c}, \tag{B.15}$$

where K is now the *kernel matrix* defined via $K_{ij} = \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j}) = K(\mathbf{x_i}, \mathbf{x_j})$.

Appendix C

The Conjugate Gradient Algorithm

The Conjugate Gradient algorithm is an algorithm for solving systems of linear equations $A\mathbf{x} = \mathbf{b}$, where A is positive semidefinite (or, equivalently, for minimizing unconstrained quadratic forms $\frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x}$).

Although the Conjugate Gradient algorithm is simple to state, understanding its workings is quite difficult. For a good tutorial on the Conjugate Gradient, see [112] or especially [101]. Here, we will restrict ourselves to stating the algorithm, and discussing why it is especially useful in machine learning problems such as Regularized Least Squares Classification.

Conjugate Gradient is an *iterative* algorithm for solving system of equations. Unlike algorithms such as Gaussian elimination, which operate directly on the matrix and operate in a fixed number of floating-point operations, iterative methods begin with a candidate solution $\mathbf{x}$ (often the all-zero vector), and iteratively refine this estimate, generally through a combination of multiplying A by $\mathbf{x}$ and some additional computations. Gradient descent is another (simpler) iterative algorithm for solving $A\mathbf{x} = \mathbf{b}$ when A is positive semidefinite.

Because iterative algorithms operate primarily by forming matrix-vector products $A\mathbf{x}$, they are especially useful in situations where these products are easy to form, but operating directly on the matrix using row-operations is unwieldy or infeasible. A primary example of this is if the matrix A is large and sparse. It may be fast to form $A\mathbf{x}$, but if we begin performing Gaussian elimination, we will soon generate a

```
Let $i = 0$
Let $\mathbf{r} = \mathbf{b} - A\mathbf{x}$
Let $\mathbf{d} = \mathbf{r}$
Let $\delta_{\text{new}} = \mathbf{r}^T\mathbf{r}$
Let $\delta_0 = \delta_{\text{new}}$
while $i < i_{\max}$ and $\delta_new > \epsilon^2\delta_0$ do
    Let $\mathbf{q} = A\mathbf{d}$
    Let $\alpha = \frac{\delta_{\text{new}}}{\mathbf{d}^T\mathbf{q}}$
    Let $\mathbf{x} = \mathbf{x} + \alpha\mathbf{d}$
    if $i$ is divisible by $50$ then
        Let $\mathbf{r} = \mathbf{b} - A\mathbf{x}$
    else
        Let $\mathbf{r} = \mathbf{r} - \alpha\mathbf{q}$
    end if
    Let $\delta_{\text{old}} = \delta_{\text{new}}$
    Let $\delta_{\text{new}} = \mathbf{r}^T\mathbf{r}$
    Let $\beta = \frac{\delta_{\text{new}}}{\delta_{\text{old}}}$
    Let $\mathbf{d} = \mathbf{r} + \beta\mathbf{d}$
    Let $i = i + 1$
end while
```

Figure C-1: The Conjugate Gradient algorithm.

dense matrix. Another primary example, that appears several times in this thesis, is the case where $A = MM^T + kI$, where M is an ℓ-by-d matrix. If ℓ is large, MM^T may be too large to store, so we cannot work with A directly, but we can form the matrix-vector product $A\mathbf{x}$ in $O(\ell d)$ operations.

Conjugate gradient has been shown empirically and theoretically to have very fast convergence, and avoids many of the "zigzagging" and "overshooting" issues often associated with Steepest Descent. It generally converges in a relatively small number of iterations (compared to Steepest Descent). For solving large, positive semidefinite systems of equations, it is often the algorithm of choice. For completeness, we present the algorithm in Figure C-1; this algorithm, along with derivations, explanations and intuitions, can be found in [101].

Bibliography

[1] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Machine Learning Research*, pages 113–141, 2000.

[2] Mariano Alvira and Ryan Rifkin. An empirical comparison of snow and svms for face detection. Technical Report A. I. Memo No. 2001-004, C.B.C.L. Memo No. 193, MIT Center for Biological and Computational Learning, 2001.

[3] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

[4] K. J. Arrow and L. Hurwicz. Reduction of constrained-maxima to saddle-point problems. In J. Neyman, editor, *Proceedings of the Third Berkeley Symposium on Probability*, volume V, pages 1–20, Berkeley, CA, 1956. University of California Press.

[5] Peter L. Bartlett. For valid generalization, the size of the weights is more important than the size of the network. In *Advances in Neural Information Processing Systems 9, Proceedings of the 1996 Conference*, pages 134–140, London, UK, December 1996. MIT Press.

[6] Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. *Nonlinear programming: theory and algorithms.* Wiley-Interscience, 2nd edition, 1993.

[7] J. M. Bennet. Triangular factors of modified matrices. *Numerisches Mathematik*, 7:217–221, 1965.

[8] M. Bertero. Regularization methods for linear inverse problems. In C. G. Talenti, editor, *Inverse Problems*. Springer-Verlag, Berlin, 1986.

[9] M. Bertero, T. Poggio, and V. Torre. Ill-posed problems in early vision. *Proceedings of the IEEE*, 76:869–889, 1988.

[10] Dimitris Bertsimas and John Tsitsiklis. *Introduction to linear optimization.* Athena Scientific, Belmont, Mass, 1997.

[11] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error-correcting binary group codes. *Inofrmation and Control*, 3:68–79, 1960.

[12] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pages 144–152, 1992.

[13] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2002.

[14] Erin Bredensteiner and Kristin P. Bennett. Multicategory classification by support vector machines. In *Computational Optimizations and Applications*, volume 12, pages 53–79, 1999.

[15] C. J. C. Burges and D. J. Crisp. Uniqueness of the svm solution. In *Neural Information Processing Systems*, 1999.

[16] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[17] Chih-Chung Chang, Chih-Wei Hsu, and Chih-Jen Lin. The analysis of decomposition methods for support vector machines. In *Proceedings of IJCAI99, SVM workshop*, 1999.

[18] William W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[19] Ronan Collobert and Samy Bengio. Svmtorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1, 2001.

[20] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:1–25, 1995.

[21] Corinna Cortes. *Prediction of Generalization Ability in Learning Machines.* PhD thesis, University of Rochester, Rochester, New York, 1995.

[22] Koby Crammer and Yoram Singer. Improved output coding for classification using continuous relaxation. In *Proceedings of the thirteenth annual conference on neural information processing systems*, 2000.

[23] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. In *Computational Learning Theory*, pages 35–46, 2000.

[24] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-basd vector machines. *Journal of Machine Learning Research*, 2001.

[25] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 2002, to appear.

[26] P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numerical Mathematics*, 31:377–390, 1966.

[27] Nello Cristianini and John Shaw-Taylor. *An Introduction To Support Vector Machines.* Cambridge University Press, 2000.

[28] Felipe Cucker and Steve Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39:1–49, 2002.

[29] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[30] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[31] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio. Regularization networks and support vector machines. *Advanced In Computational Mathematics*, 13(1):1–50, 2000.

[32] Shai Fine and Katya Scheinberg. Efficient application of interior point methods for quadratic problems arising in support vector machines using low-rank kernel representation. *Submitted to Mathematical Programming*, 2001.

[33] R. Fletcher. *Practical Methods of Optimization, Second Edition.* John Wiley & Sons, 1987.

[34] R. Fletcher and M. J. D. Powell. On the modification of the ldl^t factorization. *Mathematics of Computation*, 28(128):1067–1087, 1974.

[35] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[36] Thilo-Thomas Frieβ. Support vector neural networks: The kernel adatron with bias and soft-margin. Unpublished., 1998.

[37] Thilo-Thomas Frieβ, Nello Cristianini, and Colin Campbell. The kernel-adatron algorithm: a fast and simple learning procedure for support vector machines. In Jude Shavlik, editor, *Machine Learning: Proceedings of the Fifteenth International Conference (ICML '98)*, pages 188–196. Morgan Kaufman, 1998.

[38] Thilo-Thomas Frieβ and Rob Harrison. The kernel adatron with bias unit: Analysis of the algorithm. Technical Report 729, University of Sheffield, October 1998.

[39] J. H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University, 1996.

[40] Glenn Fung and O. L. Mangasarian. Proximal support vector classifiers. In Provost and Srikant, editors, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 77–86. ACM, 2001.

[41] Glenn Fung and O. L. Mangasarian. Proximal support vector machine classifiers. Technical report, Data Mining Institue, 2001.

[42] Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[43] C. F. Gauss. Theoria combinationis obsevationum erroribus minimis obnoxiae. *Werke*, 1823.

[44] T. Van Gestel, J. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle. Benchmarking least squares support vector machine classifiers. Technical Report Internal Report 00-37, ESAS-SISTA, 2000.

[45] T. Van Gestel, J. Suykens, B. De Moor, and J. Vandewalle. Bayesian inference for ls-svms on large data sets using the nstrom method. In *International Joint Conference on Neural Networks*, 2002.

[46] P. E. Gill, W. Murray, and M. A. Saunders. Methods for computing and modifying the *ldv* factors of a matrix. *Mathematics of Computation*, 29:1051–1060, 1975.

[47] Federico Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 1998.

[48] Federico Girosi and Tomaso Poggio. Networks and the best approximation property. Technical Report A.I. Memo No. 1164, C.B.C.L Paper No. 45, Massachusetts Institute of Technology, Artificial Intelligence Laboratory and Center

for Biological and Computational Learning, Department of Brain and Cognitive Sciences, October 1989.

[49] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[50] P. J. Green and B. W. Silverman. *Nonparametric Regression and Generalized Linear Models*. Number 58 in Monographs on Statistics and Applied Probability. Chapman & Hall, 1994.

[51] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *The Annals of Statistics*, 26(2):451–471, 1998.

[52] Bernd Heisele, Tomaso Poggio, and Massimiliano Pontil. Face detection in still gray images. Technical Report A.I. Memo No. 2001-010, C.B.C.L. Memo No. 197, MIT Center for Biological and Computational Learning, 2000.

[53] Ralf Herbrich. *Learning Kernel Classifiers*. MIT Press, 2002.

[54] Tommi Jaakkola and David Haussler. Probabilistic kernel regression models. In *Advances in Neural Information Processing Systems 11*, 1998.

[55] Thorsten Joachims. Making large-scale svm learning practical. Technical Report LS VIII-Report, Universität Dortmund, 1998.

[56] Thorsten Joachims. Estimating the generalization performance of a svm efficiently. Technical Report LS8-Report 25, Universität Dortmund, 1999.

[57] Thorsten Joachims. Making large-scale svm learning practical. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, chapter 11, pages 169–185. MIT Press, 1999.

[58] Thorsten Joachims. Estimating the generalization performance of a svm efficiently. In *Proceedings of the International Conference on Machine Learning*, 2000.

[59] S. S. Keerthi and E. G. Gilbert. Convergence of a generalized smo algorithm for svm classifier design. Technical Report CD-00-01, Control Division, Dept. of Mechanical and Production Engineering, National University of Sinagpore, 2000.

[60] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. Technical Report TR-ISL-99-03, Intelligent System Lab, Department of Computer Science & Automation, Indian Institute of Science, 1999.

[61] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. Technical Report CD-99-14, Control Division, Department of Mechanical and Production Engineering, National University of Singapore, 1999.

[62] E. B. Kong and T. G. Dietterich. Why error-correcting output coding works with decision trees. Technical report, Department of Computer Science, Oregon State University, Corvallis, OS, 1995.

[63] S. Kutin. Extensions to mcdiarmid's inequality when differences are bounded with high probability. Technical Report TR-2002-04, University of Chicago, 2002.

[64] S. Kutin and P. Niyogi. Almost-everywhere algorithmic stability and generalization error. Technical report TR-2002-03, University of Chicago, 2001.

[65] S. Kutin and P. Niyogi. The interaction of stability and weakness in adaboost. Technical report TR-2001-30, University of Chicago, 2001.

[66] Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines. Technical Report 1043, Department of Statistics, University of Wisconsin, 2001.

[67] Yoonkyung Lee, Yi Lin, and Grace wahba. Multicategory support vector machines. In *Proceedings of the 33rd Symposium on the Interface*, 2001.

[68] Yuh-Jye Lee and Olvi Mangasarian. Rsvm: Reduced support vector machines. In *SIAM International Conference on Data Mining*, 2001.

[69] Y. Lin, Y. Lee, and G. Wahba. Support vector machines for classification in nonstandard situations. Technical Report TR 1016, University of Wisconsin, Madison, March 2000.

[70] Y. Lin, Y. Lee, and G. Wahba. Support vector machines for classification in nonstandard situations. *Machine Learning*, 46:191–202, 2002.

[71] Yi Lin. Support vector machines and the bayes rule in classification. Technical Report Technical Report Numberr 1014, Department of Statistics, University of Wisconsin, 1999.

[72] A. Luntz and V. Brailovsky. On estimation of characters obtained in statistical procedure of recognition (in russian). *Technicheskaya Kibernetica*, 3, 1969.

[73] O. L. Mangasarian and David R. Musicant. Successive overrelaxation for support vector machines. Technical Report Mathematical Programming Technical Report 98-14, Computer Sciences Department, University of Wisconsin Madison, 1998.

[74] Olvi L. Mangasarian. www.cs.wisc.edu/˜olvi/.

[75] Olvi L. Mangasarian. *Nonlinear programming.* McGraw-Hill, New York, 1969.

[76] László Máté. *Hilbert Spce Methods in Science and Engineering.* Adam Hilger, 1989.

[77] C. J. Merz and P. M. Murphy. Uci repository of machine learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html, 1998.

[78] Dan Roth Ming-Hsuan Yang and Narendra Ahuja. A snow-based face detector. In *Advances in Neural Information Processing Systems 12 (NIPS 12)*, 2000.

[79] Marvin L. Minsky and Seymour A. Papert. *Perceptrons.* MIT Press, 1988.

[80] B. A. Murtagh and M. A. Saunders. Minos 5.4 user's guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Stanford University, December 1983. (revised February 1995).

[81] Yurii Nesterov and Arkadii Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming*, volume 13 of *Studies In Applied Mathematics*. SIAM, 1994.

[82] Edgar Osuna. *Support Vector Machines: Training and Applications*. PhD thesis, Massachusetts Institute of Technology, 1998.

[83] Edgar Osuna, Robert Freund, and Federico Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII.*, pages 276–285, New York, NY, USA, 1997.

[84] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MST-TR-98-14, Microsoft Research, April 1998.

[85] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, chapter 12, pages 169–185. MIT Press, 1999.

[86] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, September 1990.

[87] Tomaso Poggio, Sayan Mukherjee, Ryan Rifkin, Alex Rakhlin, and Alessandro Verri. b. In *Uncertainty in Geometric Computations*, 2001.

[88] J. R. Quinlan. *C4.5: Programs for Empirical Learning.* Morgan Kaufmann, San Francisco, CA, 1993.

[89] Ramaswamy, Tamayo, Rifkin, Mukherjee, Yeang, Angelo, Ladd, Reich, Latulippe, Mesirov, Poggio, Gerlad, Loda, Lander, and Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *Proceedings of the National Academy of Science*, December 2001.

[90] Rennie and Rifkin. Improving multiclass classification with the support vector machine. Technical Report A.I. Memo 2001-026, C.B.C.L. Memo 210, MIT Artificial Intelligence Laboratory, Center for Biological and Computational Learning, 2001.

[91] Ryan Rifkin, Massimiliano Pontil, and Theodoros Evgeniou. A note on support vector machine degeneracy. Technical Report A.I. Memo No. 1661, C.B.C.L. Paper No. 177, MIT, 1998.

[92] Frank Rosenblatt. *Principles of Neurodynamics*. Spartan Books, 1962.

[93] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–28, 1998.

[94] Saburou Saitoh, Daniel Alpay, and Joseph Ball, editors. *Reproducing kernels and their applications*, volume 3 of *International Society for Analysis, Applications and Computation*. Kluwer Academic, 1999.

[95] R. E. Schapire, Y. Freudn, P. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[96] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[97] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In *Proceedings of the 14th Annual Conference on Computational Learning Theory*, pages 416–426, 2001.

[98] Bernhard Schölkopf and Alex Smola. *Learning with Kernels.* MIT Press, 2002.

[99] I. Schönberg. Spline functions and the problem of graduation. *Proceedings of the National Academy of Science*, pages 947–950, 1964.

[100] T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce english text. *Journal of Complex Systems*, 1(1):145–168, 1987.

[101] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. http://www-2.cs.cmu.edu/ jrs/jrspapers.html, 1994.

[102] S. Smale and D. Zhou. Estimating the approximation error in learning theory. preprint, 2001.

[103] Alex J. Smola and Bernhard Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the International Conference on Machine Learning*, 2000.

[104] K. K. Sung. *Learning and Example Selection for Object and Pattern Recognition.* PhD thesis, MIT, 1996.

[105] J. A. K. Suykens. Least squares support vector machines for classification and nonlinear modelling. *Neural Network World*, 10(1-2):29–48, January 2000.

[106] J. A. K. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle. Weighted least squares support vector machine: robustness and sparse approximation. Technical Report Internal Report 00-117, ESAT-SISTA, 2000.

[107] J. A. K. Suykens, L. Lukas, P. Van Dooren, B. De Moor, and J. Vandewalle. Least squares support vector machine classifiers: a large scale algorithm. In *Proceedings of the European Conference on Circuit Theory and Design*, 1999.

[108] J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse approximation using least squares support vector machines. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2000.

[109] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 1999.

[110] J. A. K. Suykens and J. Vandewalle. Multiclass least squares support vector machines. In *Proceedings of the International Joint Conference on Neural Networks*, 1999.

[111] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-posed problems.* W. H. Winston, Washington D.C., 1977.

[112] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 2000.

[113] Robert J. Vanderbei. Loqo user's manual – version 3.10. Technical Report SOR-97-03, Princeton University, December 1997.

[114] Vladimir N. Vapnik. *Estimation of Dependencies Based On Empirical Data.* Springer Verlag, 1979. (Russian. English translation in 1982.).

[115] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, 1995.

[116] Vladimir N. Vapnik. *Statistical Learning Theory.* John Wiley & Sons, 1998.

[117] V. V. Vasin. Relationship of several variational methods for the approximate solution of ill-posed problems. *Avtomatika i Telemekhanika*, 7:161–166, 1970.

[118] S. Viaene, B. Baesens, T. Van Gestel, J. Suykens, D. Dedene, B. De Moor, and J. Vanthienen. Least squares support vector machine classifiers: an empirical evaluation. In *Proceedings of the 12th Belgian-Dutch Artificial Intelligence Conference*, 2000.

[119] Grace Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics.* Society for Industrial & Applied Mathematics, 1990.

[120] Grace Wahba. Support vector machines, reproducing kernel hilbert spaces and the randomized gacv. Technical Report 984rr, University of Wisconsin, Department of Statistics, 1998.

[121] J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Royal Holloway, University of London, Department of Computer Science, 1998.

[122] Christopher K. I. Willams and Matthias Seeger. Using the nystrom method to speed up kernel machines. In *Neural Information Processing Systems*, 2000.

[123] Christopher K. I. Williams and Matthias Seeger. The effect of the input density distribution on kernel-based classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, pages 1159–1166, 2000.

[124] R. C. Williamson, A. J. Smola, and B. Schölkopf. Generalization perormance of networks and support vector machines via entropy numbers of compact operators. Technical Report NeuroCOLT NC-TR-98-019, Royal Holloway College, 1998.

[125] P. Wolfe. A duality theorem for nonlinear programming. *Quarterly of applied mathematics*, 19(3):239–244, 1961.

[126] Ming-Hsuan Yang and Narendra Ahuja. A geometric approach to train support vector machines. In *Proceedings of the 2000 IEEE Conference on Computer Vision and Pattern Recognition*, 2000.

[127] Yeang, Ramaswamy, Tamayo, Mukherjee, Rifkin, Angelo, Reich, Lander, Mesirov, and Golub. Molecular classification of multiple tumor types. In *Intelligent Systems in Molecular Biology*, 2001.