Reduction Techniques for Training Support Vector Machines

by

Kuan-ming Lin

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer Science and Information Engineering)
in National Taiwan University
2002

ABSTRACT

Recently two kinds of reduction techniques which aimed at saving training time for SVM problems with nonlinear kernels were proposed. Instead of solving the standard SVM formulation, these methods explicitly alter the SVM formulation, and solutions for them are used to classify data. The first approach, reduced support vector machine (RSVM) [21], preselects a subset of data as support vectors and solves a smaller optimization problem. The second approach [11] uses imcomplete Cholesky factorization (ICF) to obtain a low-rank approximation of the kernel matrix. Therefore, an easier optimization problem is obtained. We find that several issues of their practical uses have not been fully discussed yet. For example, we do not know if they possess comparable generalization ability as the standard SVM. In addition, we would like to see for how large problems they outperform SVM on training time. In this thesis we show that the formulation of each technique is already in a form of linear SVM and discuss several suitable implementations. Experiments indicate that in general the test accuracy of both techniques is a little lower than that of the standard SVM. In addition, for problems with up to tens of thousands of data, if the percentage of support vectors is not high, existing implementations for SVM is quite competitive on the training time. Thus, the two techniques will be mainly useful for either larger problems or those with many support vectors. Experiments in this thesis also serve as comparisons of (1) different implementations for linear SVM; (2) standard SVM using linear and quadratic cost functions; and (3) two ICF algorithms for positive definite dense matrices.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Chih-Jen Lin. He introduced me to the subject of Support Vector Machines. The software implementation is extended from the work of Chih-Chung Chang and Chih-Wei Hsu. I also wish to express my gratitude to M. Heiler and Y.-J. Lee for many helpful comments.

To my parents I give my appreciation for their support and *love* over the years. Actually they know nothing about what I do at school, but they do everything for me at home.

This work was supported in part by the National Science Council of Taiwan via the grant NSC 90-2213-E-002-111.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER I

Introduction

The support vector machine (SVM) is a new and promising technique for data classification and regression. After the recent development, it has become an important topic in machine learning and pattern recognition. Not only it has a better theoretical foundation, practical comparisons have also shown that it is competitive with existing methods such as neural networks and decision trees (e.g. [20, 3, 9]).

Existing surveys and books on SVM are, for example, [7, 47, 48, 4, 41, 8]. The number of applications of SVM is dramatically increasing, for example, object recognition [36], combustion engine detection [39], function estimation [44], text categorization [18], chaotic system [31], handwritten digit recognition [29], and database marketing [1].

Unfortunately, there are some difficulties when one tries to scale up SVM to large data sets. First, training time increases dramatically with the size of the training data set; second, memory requirements may increase quadratically with data; third, prediction time is scaled up with the number of support vectors. A number of researchers have studied these issues and developed possible solutions. In this thesis we concentrate on methods which reduce the memory requirement by modifying the SVM formulations. We propose various implementations for these methods. Our

experiments shows that these methods slightly downgrade the testing accuracy, and some implementations show time speedup in addition to memory saving. Therefore, these methods will be mainly useful when we get trouble in solving traditional SVM with insufficient memory or time resources.

This thesis is structured as follows. In Chapter II, we briefly introduce the concepts of SVM. In Chapter III, we introduce the reduced support vector machines (RSVM) [21] by outlining the key modifications from standard SVM to RSVM. In Chapter IV, we detail the SSVM method used in [21], and apply four more techniques, the Least Square SVM (LS-SVM) [45], the Lagrangian SVM (LSVM) [28], and the decomposition method to solve the RSVM problem. Numerical experiments and comparisons are presented in Chapter V. The first part of this thesis is based on the paper [25].

In Chapter VI, we introduce the other reduction technique: the incomplete Cholesky factorization (ICF) kernel approximation [11]. There we compare the formulation, implementation, and performance with the original SVM and RSVM. Finally, we have some discussions and conclusions in Chapter VII.

CHAPTER II

Basic Concepts of SVM

The SVM technique was first developed by Vapnik [48] and his group in former AT&T Bell Laboratories. The original idea of SVM is to use a linear separating hyperplane to separate training data in two classes. Given training vectors $x_i, i = 1, \ldots, l$ of length n, and a vector y defined as follows

$$y_i = \begin{cases} 1 & \text{if } x_i \text{ in class 1,} \\ -1 & \text{if } x_i \text{ in class 2,} \end{cases}$$

the support vector technique tries to find the separating hyperplane with the largest margin between two classes, measured along a line perpendicular to the hyperplane. For example, in Figure 2.1, two classes could be fully separated by a dotted line $w^T x + b = 0$. We would like to decide the line with the largest margin. In other words, intuitively we think that the distance between two classes of training data should be as large as possible. That means we want to find a line with parameters w and b such that the distance between $w^T x + b = \pm 1$ is maximized. As the distance between $w^T x + b = \pm 1$ is $2/\|w\|$ and maximizing $2/\|w\|$ is equivalent to minimizing

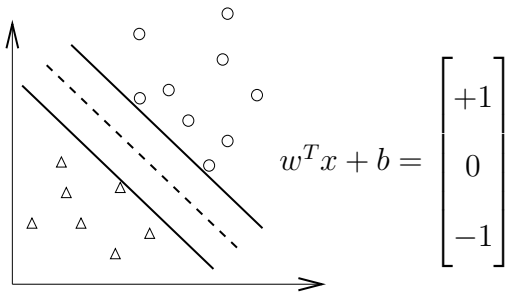

Figure 2.1: Separating hyperplane

$w^T w/2$, we have the following problem:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} w^T w \\ & y_i(w^T x_i + b) \geq 1, i = 1, \ldots, l. \end{aligned} \tag{2.1}$$

The constraint $y_i(w^T x_i + b) \geq 1$ means

$$\begin{aligned} w^T x_i + b \geq 1 \quad & \text{if } y_i = 1, \\ w^T x_i + b \leq -1 \quad & \text{if } y_i = -1. \end{aligned}$$

That is, data in the class 1 must be on the upper half space of $w^T x + b = 0$ while data in the other class must be on the left-hand side. Note that the reason of maximizing the distance between $w^T x + b = \pm 1$ is based on Vapnik's Structural Risk Minimization [48].

However, practically problems may not be linearly separable where an example is in Figure 2.2. SVM uses two methods to handle this difficulty [2, 7]: First, it allows training errors. Second, SVM non-linearly transforms the original input space into

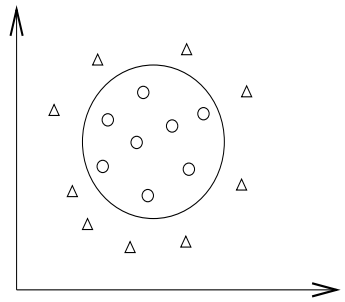

Figure 2.2: An example which is not linearly separable

a higher dimensional feature space by a function ϕ:

$$\min_{w,b,\xi} \quad \frac{1}{2}w^T w + C\sum_{i=1}^{l}\xi_i \tag{2.2}$$

$$y_i(w^T\phi(x_i)+b) \geq 1-\xi_i, \tag{2.3}$$

$$\xi_i \geq 0,\ i=1,\ldots,l. \tag{2.4}$$

A penalty term $C\sum_{i=1}^{l}\xi_i$ is added to the objective function and training errors are allowed. That is, constraints (2.3) allow that training data may not be on the correct side of the separating hyperplane $w^T x + b = 0$ while we minimize the training error $\sum_{i=1}^{l}\xi_i$ in the objective function. Hence if the penalty parameter C is large enough and the data is linearly separable, problem (2.2) reduces to (2.1) as all ξ_i go to zero [22].

Occasionally another kind of penalty term $C\sum_{i=1}^{l}\xi_i^2$ is used instead. An advantage is that the nonnegativity constraint (2.4) is redundant in this case. That is, any optimal solution for the following minimization problem will satisfy (2.4):

$$\min_{w,b,\xi} \quad \frac{1}{2}w^T w + C\sum_{i=1}^{l}\xi_i^2 \tag{2.5}$$

$$y_i(w^T\phi(x_i)+b) \geq 1-\xi_i,\ i=1,\ldots,l. \tag{2.6}$$

For the generalized problems, that training data x is mapped into a vector in a higher (possibly infinite) dimensional space:

$$\phi(x) = (\phi_1(x), \phi_2(x), \ldots).$$

Hence (2.5) (and also (2.2)) can be a problem in an infinite dimensional space which is not easy to solve. Currently the main procedure is to solve a dual formulation of (2.5). It needs a closed form of $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ which is usually called the kernel function. Some popular kernels are, for example, RBF kernel: $e^{-\gamma \|x_i - x_j\|^2}$ and polynomial kernel: $(x_i^T x_j/\gamma + \delta)^d$, where γ, δ, and d are parameters. Here is the dual formulation:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T (Q + \frac{I}{2C})\alpha - e^T \alpha \\ \text{subject to} \quad & y^T \alpha = 0, \\ & 0 \le \alpha_i, \ i = 1, \ldots, l, \end{aligned} \tag{2.7}$$

where e is the vector of all ones, Q is an l by l positive semi-definite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$.

After the dual problem is solved, w is constructed by α and training vectors. The decision function is written as

$$f(x) = \text{sign}(w^T \phi(x) + b). \tag{2.8}$$

In other words, for a test vector x, if $w^T \phi(x) + b > 0$, we classify it to be in the class 1. Otherwise, we think it is in the second class. Only some of $x_i, i = 1, \ldots, l$ are used to construct w and b and they are important data called support vectors. In general, the number of support vectors is not large. Therefore we can say SVM is

used to find important data (support vectors) from training data.

For large-scale problems, the difficulty of solving SVM problems results from the fully dense matrix Q (arising in the dual problem (2.7)), which cannot be saved in the main memory. Thus traditional optimization algorithms like Newton or Quasi-Newton cannot be directly used. Currently one major method is the decomposition method (for example, [34, 37, 17]) which solves a sequence of smaller-sized problems so the memory difficulty is avoided. However, for huge problems with many support vectors, the decomposition method still suffers from slow convergence. As a result, we survey two types of reduction techniques. Both of them reduce the size of the kernel matrix to any acceptable size, and then there exists suitable algorithms (some are exact solution for linear systems, while others are iterative approximations) to solve the reduced problem.

CHAPTER III

The RSVM Method

Recently [21] proposed to restrict the number of support vectors by solving the reduced support vector machines (RSVM). The main characteristic of this method is to reduce the matrix Q from $l \times l$ to $l \times m$, where m is the size of a randomly selected subset of training data considered as support vectors. The smaller matrix can be stored in memory, so optimization algorithms such as Newton method can be applied directly.

Here we outline the key modifications from standard SVM to RSVM. They started from adding an additional term $b^2/2$ to the objective function of (2.5):

$$\min_{w,b,\xi} \quad \frac{1}{2}(w^T w + b^2) + C \sum_{i=1}^{l} \xi_i^2 \tag{3.1}$$

$$\text{subject to} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i. \tag{3.2}$$

The dual form becomes a simpler bound-constrained problem:

$$\min_{\alpha} \quad \frac{1}{2}\alpha^T(Q + \frac{I}{2C} + yy^T)\alpha - e^T\alpha$$

$$\text{subject to} \quad 0 \leq \alpha_i,\ i = 1, \ldots, l. \tag{3.3}$$

The approach of adding $b^2/2$ and then obtaining a bounded dual was proposed in [12] and [27]. This is equivalent to adding a constant feature to the training data and

finding a separating hyperplane passing through the origin. Numerical experiments in [16] show that the accuracy does not vary much when $b^2/2$ is added.

It is known that, at the optimal solution, w is a linear combination of training data:

$$w = \sum_{i=1}^{l} y_i \alpha_i \phi(x_i). \tag{3.4}$$

If we substitute w into (3.1), using

$$\begin{aligned} y_i w^T \phi(x_i) &= \sum_{j=1}^{l} y_i y_j \alpha_j \phi(x_j)^T \phi(x_i) \\ &= \sum_{j=1}^{l} Q_{ij} \alpha_j = (Q\alpha)_i, \text{ and} \qquad (3.5) \\ w^T w &= \sum_{i=1}^{l} y_i \alpha_i \phi(x_i)^T w \\ &= \sum_{i=1}^{l} \alpha_i (Q\alpha)_i = \alpha^T Q \alpha, \qquad (3.6) \end{aligned}$$

we obtain a new optimization problem:

$$\begin{aligned} \min_{\alpha,b,\xi} \quad & \frac{1}{2}(\alpha^T Q \alpha + b^2) + C \sum_{i=1}^{l} \xi_i^2 \\ \text{subject to} \quad & Q\alpha + by \geq e - \xi. \end{aligned} \tag{3.7}$$

Though (3.7) is different from (3.3), the dual problem, we can show that for any optimal α of (3.7), the corresponding w defined by (3.4) is also an optimal solution of (3.1), so we can solve (3.7) instead of (3.3). The details are in Appendix A. Actually such reformulation of (3.1) to (3.7) also appeared in [35].

The idea of RSVM is to reduce the number of support vectors. It is achieved by

randomly selecting a subset of m samples for constructing w:

$$w = \sum_{i \in R} y_i \alpha_i \phi(x_i), \tag{3.8}$$

where R contains indices of this subset. Substituting (3.8) into (3.2), we can get a similar problem as (3.7), with the number of major variables (i.e. α) reduced to m:

$$\begin{aligned} \min_{\bar{\alpha},b,\xi} \quad & \frac{1}{2}(\bar{\alpha}^T Q_{RR} \bar{\alpha} + b^2) + C \sum_{i=1}^{l} \xi_i^2 \\ \text{subject to} \quad & Q_{:,R} \bar{\alpha} + by \geq e - \xi, \end{aligned} \tag{3.9}$$

where $\bar{\alpha}$ is the collection of all $\alpha_i, i \in R$. Note that now m is the size of R. We use $Q_{:,R}$ to represent the sub-matrix of columns corresponding R. Thus, there are still l constraints, and hence solving RSVM is not close to directly solving smaller SVM problems with m training data, for the formulation of the latter (changing l to m in (3.7)) has only m constraints.

Following the generalized SVM by Mangasarian [26], [21] simplified the term $\frac{1}{2}\bar{\alpha}^T Q_{RR} \bar{\alpha}$ to $\frac{1}{2}\bar{\alpha}^T \bar{\alpha}$ so RSVM solves

$$\begin{aligned} \min_{\bar{\alpha},b,\xi} \quad & \frac{1}{2}(\bar{\alpha}^T \bar{\alpha} + b^2) + C \sum_{i=1}^{l} \xi_i^2 \\ \text{subject to} \quad & Q_{:,R} \bar{\alpha} + by \geq e - \xi. \end{aligned} \tag{3.10}$$

Later on we will address more about this simplification.

The idea of using (3.8) is similar to the Radial Basis Function (RBF) networks [32] which either select a subset of training data or generate some "centers" in order to construct a decision function. RBF networks directly start from a form similar to (3.8) and the regularization term used is $\bar{\alpha}^T \bar{\alpha}/2$. If the inequality (3.10) is replaced by an equality, (3.10) is in a form of the RBF networks. There are already comparisons between SVM and RBF networks. For example, [42] shows that SVM performs

better as RBF network uses less information. Therefore, we are curious whether similar scenarios apply to SVM and RSVM, which will be one of the main aims of this thesis.

CHAPTER IV

Different Implementations for RSVM

In [21], the authors used Smooth SVM (SSVM) to solve RSVM, which basically approximates the original problem by an unconstrained problem. However, we observe that (3.10) is already in the primal form of the original linear SVM. Therefore, existing methods which focus on solving linear SVM can be applied here. To see this, suppose R contains indices $\{j_1, \ldots, j_m\}$, in the following we point out the relation between (3.1) and (3.10):

SVM	RSVM
n	m
w	$\bar{\alpha}$
x_i	$[y_i Q_{ij_1}, \ldots, y_i Q_{ij_m}]^T$

Therefore, if we consider $[y_1 Q_{ij_1}, \ldots, y_m Q_{ij_m}]^T$, $i = 1, \ldots, l$ as training data, then the number of attributes is m and $(\bar{\alpha}, b)$ are coefficients of the separating hyperplane. In other words, it is like that we are training a linear SVM with l data. For linear SVM the number of features is usually much smaller than the number of data so there are methods which possess this property. However, for nonlinear SVM where the dimensionality of the feature space is large, such methods may not be applicable so usually the dual problem is considered. Now we choose $m \ll l$ so we are safe to

apply methods which were originally mainly suitable for linear SVM.

In Section 4.1 we will discuss the SSVM originally used in [21] while in Sections 4.2- 4.4, we consider three methods which were suitable for linear SVM: Least Square SVM (LS-SVM) [45], Lagrangian SVM (LSVM) [28], and the decomposition method for linear SVM.

Before describing different implementations, for the sake of convenience, with following substitution

$$\widetilde{Q} = \begin{bmatrix} Q_{:,R} & y \end{bmatrix}, \widetilde{\alpha} = \begin{bmatrix} \bar{\alpha} \\ b \end{bmatrix},$$

we consider a simpler form:

$$\begin{aligned} \min_{\widetilde{\alpha},\xi} \quad & \frac{1}{2}\widetilde{\alpha}^T\widetilde{\alpha} + C\sum_{i=1}^{l}\xi_i^2 \\ \text{subject to} \quad & \widetilde{Q}\widetilde{\alpha} \geq e - \xi. \end{aligned} \tag{4.1}$$

4.1 Using SSVM to implement RSVM

Define $(\cdot)_+ \equiv \max(\cdot, 0)$, and apply the property that whenever $\xi_i > 0$, the ith constraint (4.1) must be active, then the constrained problem (4.1) can be transformed to an unconstrained one:

$$\min_{\widetilde{\alpha}} \quad \frac{1}{2}\widetilde{\alpha}^T\widetilde{\alpha} + C\sum_{i=1}^{l}((e - \widetilde{Q}\widetilde{\alpha})_i)_+^2 \tag{4.2}$$

If the objective function is differentiable (or twice differentiable), we can use general methods (for example, Newton method, quasi-Newton method, etc.) to find an optimal solution. Unfortunately $((e - \widetilde{Q}\widetilde{\alpha})_i)_+$ is not differentiable so SSVM approximates the function $(t)_+$ by

$$P_\beta(t) \equiv t + \beta^{-1}\log(1 + \exp(-\beta t)),$$

where β is called the smoothing parameter. When β approaches to infinity, $P_\beta(t)$ converges to $(t)_+$.

We observe that if C is large, sometimes huge objective values of (4.2) may cause numerical difficulties. For example, if a method for solving (4.2) uses the absolute difference on objective values of two successive iterations as the stopping criterion, the large objective values may cause very long iterations before reaching a specified tolerance. Hence, we divide the objective function by C:

$$\min_{\widetilde{\alpha}} f(\tilde{\alpha}) = \frac{1}{2C}\widetilde{\alpha}^T\widetilde{\alpha} + \sum_{i=1}^{l} P_\beta(e - \widetilde{Q}\widetilde{\alpha})_i^2 \tag{4.3}$$

We solve (4.3) by a Newton's method implemented in the software TRON [23], which uses both the information of the first derivative (gradient) and the second derivative (Hessian).

While [21] did not give the explicit form of the gradient and the Hessian, we list them here as a reference. By defining

$$V \equiv \exp(-\beta(e - \widetilde{Q}\widetilde{\alpha})),$$

then

$$\nabla f(\widetilde{\alpha}) = -2\widetilde{Q}^T(\mathrm{diag}(P_\beta(e - \widetilde{Q}\widetilde{\alpha}))\mathrm{diag}(e+v)^{-1})e + \frac{1}{C}\widetilde{\alpha} \text{ and} \tag{4.4}$$

$$\nabla^2 f(\widetilde{\alpha}) = 2\widetilde{Q}^T\mathrm{diag}(e+v)^{-2}(I + \beta\mathrm{diag}(v)\mathrm{diag}(P_\beta(e - \widetilde{Q}\widetilde{\alpha})))\widetilde{Q} + \frac{I}{C}, \tag{4.5}$$

where $\mathrm{diag}(v)$ is a diagonal matrix with elements of v on the diagonal. Details on deriving them can be found in Appendix B.

TRON was originally designed for large sparse problems. Now the Hessian is dense so we use the modification in [16] to solve (4.3). As a Newton's method is used, TRON possesses the property of quadratic convergence.

Time complexity analysis: Now the Hessian is m by m so the Newton's

method takes $O(m^3)$ operations in each iteration for inverting or factorizing the Hessian. However, in order to obtain the Hessian, in (4.5), we need more operations: $O(lm^2)$. Hence $O(lm^2)$ is complexity of each iteration.

4.2 Using Least-Square SVM to implement RSVM

Least-square SVM (LS-SVM), first introduced in [45], can solve linear SVM efficiently. It changes the inequality constraints to equalities, and solve the resulting linear system. Taking (3.1) as an example, they change the primal inequalities (3.2) to $y_i(w^T\phi(x_i) + b) = 1 - \xi_i$ and remove constraints $\xi_i \geq 0$. Thus substituting ξ into the objective function we get an unconstrained problem:

$$\min_{w,b} \quad \frac{1}{2}(w^T w + b^2) + C\sum_{i=1}^{l}(1 - y_i(w^T\phi(x_i) + b))^2 \tag{4.6}$$

Of course changing inequalities into equalities may affect the meaning of separating hyperplanes though this is not the topic we would like to discuss here.

For linear SVM (i.e. x_i instead of $\phi(x_i)$ is used), we can solve (4.6) directly by a linear system of $(n + 1)$ variables where n is the number of attributes as well as the length of w. For nonlinear SVM where $n \gg l$, we must substitute $w = \sum_i y_i\alpha_i\phi(x_i)$ into (4.6) and solve a linear system with variables α and b. Thus, when l is large, the same problem of conventional SVM formulations occur: the $l \times l$ matrix cannot be placed in memory, so it is not easy to solve a large dense linear system. As a result, LS-SVM so far is more suitable for linear SVM problems.

Following the same procedure, here we change the inequality (4.1) to an equality

$$\widetilde{Q}\widetilde{\alpha} = e - \xi, \tag{4.7}$$

and substitute ξ into (4.1). Similar to the case of SSVM, we divide the objective

function by C and get an unconstrained problem where $\widetilde{\alpha}$ is the only variable:

$$\min_{\widetilde{\alpha}} \quad f(\widetilde{\alpha}) = \quad \frac{1}{2C}\widetilde{\alpha}^T\widetilde{\alpha} + \sum_{i=1}^{l}(e - \widetilde{Q}\widetilde{\alpha})_i^2 \tag{4.8}$$

With

$$f(\widetilde{\alpha}) = \frac{1}{2C}\widetilde{\alpha}^T\widetilde{\alpha} + \widetilde{\alpha}^T\widetilde{Q}^T\widetilde{Q}\widetilde{\alpha} - 2e^T\widetilde{Q}\widetilde{\alpha} + e^Te,$$

we minimize f by finding the solution of $\frac{\partial f}{\partial \widetilde{\alpha}_i} = 0, i = 1, \ldots, m$:

$$\begin{aligned} \frac{1}{C}\widetilde{\alpha} + 2\widetilde{Q}^T\widetilde{Q}\widetilde{\alpha} - 2\widetilde{Q}^Te = 0, \\ (\widetilde{Q}^T\widetilde{Q} + \frac{I}{2C})\widetilde{\alpha} = \widetilde{Q}^Te, \end{aligned} \tag{4.9}$$

a positive definite linear system of size m. As $m \ll l$, (4.9) is small enough. Thus we can use direct methods such as Gaussian elimination or Cholesky factorization to efficiently solve it.

Time complexity analysis: The cost of Gaussian elimination, $O(m^3)$, is less than that for computing the matrix multiplication $\widetilde{Q}^T\widetilde{Q}$ which costs $O(lm^2)$. Hence the total time complexity is $O(lm^2)$.

4.3 Using Lagrangian SVM to implement RSVM

Here we discuss Lagrangian SVM (LSVM) [28], another technique to solve SVM problems. LSVM is an iterative algorithm which solves the dual form (3.3). Define $H \equiv Q + \frac{I}{2C} + yy^T$, then the Karush-Kuhn-Tucker (KKT) optimality conditions of (3.3) are

$$\begin{aligned} H\alpha - e \geq 0, \alpha \geq 0, \\ (H\alpha - e)^T\alpha = 0. \end{aligned}$$

Again we denote $\max(\cdot,\, 0)$ by $(\cdot)_+$. The authors of [28] use the following identity between any two vectors a and b:

$$a \geq 0, b \geq 0, a^T b = 0 \Leftrightarrow a = (a - \beta b)_+, \forall \beta > 0, \tag{4.10}$$

so the optimality conditions is equivalent to

$$H\alpha - e = (H\alpha - e - \beta\alpha)_+, \forall \beta > 0. \tag{4.11}$$

Thus, an optimal α is a solution of the fixed-point equation and they apply the following iterative scheme:

$$\alpha^{k+1} = H^{-1}(e + (H\alpha^k - e - \beta\alpha^k)_+). \tag{4.12}$$

[28] showed that LSVM algorithm is linearly convergent if β is chosen so that

$$0 < \beta < \frac{2}{C}.$$

Though in each iteration, we must invert the $l \times l$ dense matrix H, for linear SVMs, the Sherman-Morrison-Woodbury (SMW) identity can be used. But it cannot be applied to nonlinear kernels where the input vectors are mapped into high (maybe infinite) dimensional spaces. Thus, LSVM method so far is not practical for large-scale problems using nonlinear kernels. An earlier comparison with LIBSVM, a decomposition method for standard SVM, is in [14], which shows that LSVM is slower when using nonlinear kernels.

We have mentioned earlier that RSVM is equivalent to a linear SVM problem, so the Lagrangian SVM algorithm can be used here. First we write down the dual

form of (4.1):

$$\begin{aligned} \min_{\hat{\alpha}} \quad & \frac{1}{2}\hat{\alpha}^T(\widetilde{Q}\widetilde{Q}^T + \frac{I}{2C})\hat{\alpha} - e^T\hat{\alpha} \\ \text{subject to} \quad & 0 \le \hat{\alpha}_i,\ i = 1, \ldots, l. \end{aligned} \tag{4.13}$$

Let

$$H \equiv (\widetilde{Q}\widetilde{Q}^T + \frac{I}{2C}).$$

Lagrangian SVM solves

$$\begin{aligned} \hat{\alpha}^{k+1} &= H^{-1}(e + ((H\hat{\alpha}^k - e) - \beta\hat{\alpha}^k)_+) \\ &= H^{-1}(e + ((\widetilde{Q}\widetilde{Q}^T + \frac{I}{2C} - \beta I)\hat{\alpha}^k - e)_+). \end{aligned}$$

Here H^{-1} is calculated using the SMW identity:

$$H^{-1} = (\frac{I}{2C} + \widetilde{Q}\widetilde{Q}^T)^{-1} = 2C(I - \widetilde{Q}(\frac{I}{2C} + \widetilde{Q}^T\widetilde{Q})^{-1}\widetilde{Q}^T).$$

In order to get $\widetilde{\alpha}$, now the primal variable, we apply the relationship (3.4):

$$\widetilde{\alpha} = \widetilde{Q}^T\hat{\alpha}.$$

Time complexity analysis: The generation and then inversion of the $m \times m$ matrix $(\frac{I}{2C} + \widetilde{Q}^T\widetilde{Q})$ costs $O(lm^2)$, where the main task is the matrix multiplication. This can be done before all iterations. Then in each iteration, the main cost is several matrix-vector multiplications which costs $O(lm)$ operations. So the total time complexity is $(\#iterations) \times O(lm) + O(lm^2)$. Compared with SSVM, LSVM method seems need more iterations since it guarantees only linear convergence.

4.4 Using decomposition methods to implement RSVM

Originally the decomposition method [34, 17, 37] was proposed to handle the nonlinear SVM whose kernel matrices are fully dense and cannot be stored. It is an iterative process where in each iteration the index set of variables is separated to two sets B and N, where B is the working set. Then in that iteration variables corresponding to N are fixed while a sub-problem on variables corresponding to B is minimized.

If $q \ll l$ is the size of the working set B and further techniques such as caching and shrinking are not used, in each iteration q columns of the kernel matrix are used for calculating $Q\Delta\alpha$, where Q is the kernel matrix and $\Delta\alpha$ is a vector with at most q nonzero modifications of one iteration. Therefore, the total complexity is

$$\#iterations \times O(lqn),$$

where n is the number of attributes and we assume each kernel evaluation costs $O(n)$. However, for linear SVM, Q is in a form of XX^T and X, an $l \times n$ matrix, contains all l training data. Thus,

$$Q\Delta\alpha = X(X^T\Delta\alpha) \tag{4.14}$$

costs only $O(nq) + O(ln) = O(ln)$ operations.

Therefore, for large problems where caching recently used Q_{ij} is not very useful, in each iteration, using (4.14) can be q times faster than the regular decomposition method. This trick has been implemented in, for example, SVM^{light} [17] and BSVM (version 2.03 and after) [16]. Of course the selection of q is still an issue. It should be larger than those usually used for nonlinear SVM but also cannot be too large. Otherwise solving the sub-problem which has q variables may cost more than $O(ln)$

for (4.14).

For RSVM, since $m \ll l$, we consider RSVM as a linear SVM and apply a decomposition method to solve its dual. However, we consider the formulation with linear cost function $C\sum_{i=1}^{l}\xi_i$ so instead of (4.13), the dual problem is

$$\begin{aligned} \min_{\hat{\alpha}} \quad & \frac{1}{2}\hat{\alpha}^T\widetilde{Q}\widetilde{Q}^T\hat{\alpha} - e^T\hat{\alpha} \\ \text{subject to} \quad & 0 \le \hat{\alpha}_i \le C, i = 1, \ldots, l. \end{aligned} \tag{4.15}$$

The reason is that we will use the software BSVM which solves (4.15) but not (4.13).

4.5 Stopping criteria

In order to compare the performance of different methods, we should use comparable stopping criteria for them. However, so many differences among these methods prohibit us from finding precisely equivalent stopping criteria for all methods. Still we try to use reasonable criteria which will be described below in detail.

In the next Chapter we will use LIBSVM as the representative for solving the regular nonlinear SVM so we first explain its stopping criterion. LIBSVM is a decomposition method which solves the dual form of the standard SVM with a linear cost function:

$$\begin{aligned} \min_{\alpha} \quad & f(\alpha) = \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{subject to} \quad & y^T\alpha = 0, \\ & 0 \le \alpha_i \le C, i = 1, \ldots, l. \end{aligned} \tag{4.16}$$

It is shown in [6] that if $C > 0$, the KKT condition of (4.16) is equivalent to

$$\begin{aligned} m(\alpha) &= \max(\max_{\alpha_i<C, y_i=1} -\nabla f(\alpha)_i, \max_{\alpha_i>0, y_i=-1} \nabla f(\alpha)_i) \\ &\leq \min(\min_{\alpha_i<C, y_i=-1} \nabla f(\alpha)_i, \min_{\alpha_i>0, y_i=1} -\nabla f(\alpha)_i) = M(\alpha). \end{aligned} \tag{4.17}$$

For practical implementation, a tolerance ϵ is set, and the stopping criterion can be

$$m(\alpha) \leq M(\alpha) + \epsilon, \tag{4.18}$$

where we choose $\epsilon = 0.001$.

To compare the performance of using linear and quadratic cost functions, we also modify LIBSVM to solve (2.7). It is easy to see that a similar stopping criterion can be used. Another decomposition software BSVM is used for solving RSVM as described in Section 4.4. It also has a similar stopping criterion.

The LSVM method, another implementation for RSVM, solves problems in the dual form (4.13). Hence we can use a similar stopping criterion: Now

$$\nabla f(\hat{\alpha}) = (\tilde{Q}\tilde{Q}^T + \frac{I}{2C})\hat{\alpha} - e.$$

The KKT condition of (4.13) shows that if

$$\min \nabla f(\hat{\alpha})_i \geq \max_{\hat{\alpha}_i>0} \nabla f(\hat{\alpha})_i,$$

then $\hat{\alpha}$ is an optimal solution. Thus, an stopping criterion with a tolerance ϵ can be

$$\max_{\hat{\alpha}_i\geq 0} \nabla f(\hat{\alpha}) \leq \min \nabla f(\hat{\alpha}) + \epsilon. \tag{4.19}$$

We also set $\epsilon = 0.001$ here.

For the SSVM, we simply employ the original stopping criterion of TRON:

$$\|\nabla f(\widetilde{\alpha}^k)\|_2 \leq \epsilon \|\nabla f(\widetilde{\alpha}^1)\|_2, \tag{4.20}$$

where $f(\widetilde{\alpha})$ is defined in (4.3), $\widetilde{\alpha}^1$ is the initial solution, $\widetilde{\alpha}^k$ is the current solution, and $\epsilon = 10^{-5}$.

Note that for LS-SVM implementation, we use direct methods to solve the linear system, and therefore no stopping criterion is needed.

CHAPTER V

Experiments on RSVM

In this chapter we conduct experiments on some commonly used problems. At first we extend RSVM to multi-class problems. Several methods have been proposed for SVM multi-class classification. A common way is to consider a set of binary SVM problems, while some authors also proposed methods that consider all classes at once. These methods in general can also be applied to RSVM.

There have been some comparisons of methods for multi-class SVM. In [15] we compare different decomposition implementations. Results indicate that diffenent strategies for multi-class SVM achieve similar testing accuracy, but the one-against-one method performs faster in practice. The comparison in [46] for LS-SVM also prefers the one-against-one method. So we use it for our implementation here. Suppose there are k classes of data, this method constructs $k(k-1)/2$ classifiers where each one trains data from two classes. In classification we use a voting strategy: each binary classification is considered to be a voting where votes can be cast for all data points x - in the end point is designated to be in a class with maximum number of votes.

5.1 Problems and settings

We choose large multi-class datasets from the Statlog collection: dna, satimage, letter, and shuttle [30]. We also consider mnist [19], an important benchmark for handwritten digit recognition. The problem ijcnn1 is from the first problem of IJCNN challenge 2001 [38]. Note that we use the winner's transformation of raw data [5]. The last one, protein, is a data set for protein secondary structure prediction [49]. Except problems dna, ijcnn1, and protein whose data values are already in a small range, we scale all training data to be in $[-1, 1]$. Then test data, available for all problems, are adjusted to $[-1, 1]$ accordingly. Note that for problem mnist, it takes too much training time if the whole 60,000 training samples are used, so we consider the training and testing data together (i.e. 70,000 samples) and then cut the first 30% for training and test the remaining 70%. Also note that for the problem satimage, there is one missing class. That is, in the original application there is one more class but in the data set no examples are with this class. We give problem statistics in Table 5.1. Some problems with a large number of attributes may be very sparse. For example, for each instance of protein, only 17 of the 357 attributes are nonzero.

Table 5.1: Problem statistics

Problem	#training data	#testing data	#class	#attribute
dna	2000	1300	3	180
satimage	4435	2000	6	36
letter	15000	5000	26	16
shuttle	43500	14500	7	9
mnist	21000	49000	10	780
ijcnn1	49990	45495	2	22
protein	17766	6621	3	357

We compare four implementations of RSVM discussed in Chapter IV with two implementations of regular SVM: linear and quadratic cost functions (i.e., (2.2) and

(2.5)). For regular SVM with the linear cost function, we use the software LIBSVM which implements a simple decomposition method. We can easily modify LIBSVM for using the quadratic cost function, which we will refer to as LIBSVM-q in the rest of this thesis. However, we will not use it to solve RSVM as LIBSVM implements an SMO type algorithm where the size of the working set is restricted to two. In Section 4.4 we have shown that larger working sets should be used when using decomposition methods for linear SVM.

The computational experiments for this section were done on a Pentium III-1000 with 1024MB RAM using the gcc compiler. For three of the four RSVM methods (SSVM, LS-SVM, and LSVM), the main computational work are some basic matrix operations so we use ATLAS to optimize the performance [50]. This is very crucial as otherwise a direct implementation of these matrix operations can at least double or triple the computational time. For decomposition methods where the kernel matrix cannot be fully stored we allocate 500MB memory as the cache for storing recently used kernel elements. Furthermore, LIBSVM, LIBSVM-q, and BSVM all use a shrinking technique so if most variables are finally at bounds, it solves smaller problems by considering only free variables. Details on the shrinking technique are in [6, Section 4].

Next we discuss the selection of parameters in different implementations. It is of course a tricky issue on selecting m, the size of the subset of RSVM. This depends on practical situations such as how large the problem is. Here in most cases we fix m to be 10% of the training data, which was also considered in [21]. For multi-class problems, we cannot use the same m for all binary problems as the data set may be highly unbalanced. Hence we choose m to be 10% of the size of each binary problem so a smaller binary problem use a smaller m. In addition, for problems shuttle, ijcnn1,

and protein, binary problems may be too large for training. Thus, we set $m = 200$ for these large binary problems. This is similar to how [21] deals with large problems. Once the size is determined, for all four implementations, we select the same subset for each binary RSVM problem. Note that $n \ll m$ for most problems we consider, so the time of kernel evaluations is $O(lmn)$ which is much less than that of each implementations for RSVM.

We need to decide some additional parameters prior to experiments. For SSVM method, the smoothing parameter β is set to 5 because the performance is almost the same for $\beta \geq 5$. For LSVM method, we set $\beta = \frac{1.9}{C}$ which is the same as that in [28]. We do not conduct cross validation for selecting them as otherwise there are too many parameters. For BSVM which is used to solve linear SVM arising from RSVM, we use all default settings. In particular, the size of the working set is 10.

The most important criterion for evaluating the performance of these methods is their accuracy rate. However, it is unfair to use only one parameter set and then compare these methods. Practically for any method we have to find the best parameters by performing the model selection. This is conducted on the training data where the test data are assumed unknown. Then the best parameter set is used for constructing the model for future testing. To reduce the search space of parameter sets, here we consider only the RBF kernel $K(x_i, x_j) \equiv e^{-\gamma\|x_i - x_j\|^2}$, so the parameters left for decision are kernel parameter γ and cost parameter C. In addition, for multi-class problems we consider that C and γ of all $k(k-1)/2$ binary problems via the one-against-one approach are the same.

For each problem, we estimate the generalized accuracy using different $\gamma = [2^4, 2^3, 2^2, \ldots, 2^{-10}]$ and $C = [2^{12}, 2^{11}, 2^{10}, \ldots, 2^{-2}]$. Therefore, for each problem we try $15 \times 15 = 225$ combinations. For each pair of (C, γ), the validation perfor-

mance is measured by training 70% of the training set and testing the other 30% of the training set. Then we train the whole training set using the pair of (C, γ) that achieves the best validation rate and predict the test set. The resulting accuracy is presented in the "rate" columns of Table 5.2. Note that if several (C, γ) have the same accuracy in the validation stage, we apply all of them to the test data and report the highest rate. If there are several parameters resulting in the highest testing rate, we report the one with the minimal training time.

5.2 Results

Table 5.2 shows the result of comparing LIBSVM, LIBSVM-q, and the four RSVM implementations. We present the optimal parameters (C, γ) and the corresponding accuracy rates. It can be seen that optimal parameters (C, γ) are in various ranges for different implementations so it is essential to test so many parameter sets. We observe that LIBSVM and LIBSVM-q have very similar accuracy. This does not contradict the current understanding in SVM area as we have not seen any report which shows one has higher accuracy than the other. Except ijcnn1, the difference of the four RSVM implementations is also small. This is reasonable as essentially they solve (3.10) with minor modifications.

For all problems, LIBSVM and LIBSVM-q perform better than RSVM implementations. We can expect this because for RSVM the support vectors are randomly chosen in advance, therefore we cannot ensure that the support vectors are important representatives of the training data. This seems imply that if problems are not too large, we would like to stick on the original SVM formulation. We think this situation is like the comparison between RBF networks and SVM [42] where as RBF networks select only several centers, it may not extract enough information.

In general the optimal C of RSVM is much larger than that of the regular SVM. As RSVM is indeed a linear SVM with a lot more data than the number of attributes, it tends to need a larger C so that data can be correctly separated. How this property affects its model selection remains to be investigated.

We also observe that the accuracy of LS-SVM is a little lower than SSVM and LSVM. In particular, for problem ijcnn1 the difference is quite large. Note that ijcnn1 is an unbalanced problem where 90% of the data have the same label. Thus the 91.7% accuracy by LS-SVM is quite poor. We suspect that the change of inequalities to equalities for LS-SVM may not be suitable for some problems.

Table 5.2: A comparison on RSVM: testing accuracy

	SVM				RSVM							
	LIBSVM		LIBSVM-q		SSVM		LS-SVM		LSVM		Decomposition	
Problem	C,γ	rate	C,γ	rate	C,γ	rate	C,γ	rate	C,γ	rate	C,γ	rate
dna	2^4,2^{-6}	95.447	2^2,2^{-6}	95.447	2^{12},2^{-10}	92.833	2^4,2^{-6}	92.327	2^5,2^{-7}	93.002	2^9,2^{-6}	92.327
satimage	2^4,2^0	91.3	2^3,2^0	91.9	2^{12},2^{-1}	89.8	2^{12},2^{-3}	89.9	2^2,2^{-1}	90	2^{11},2^{-1}	90
letter	2^4,2^2	97.98	2^5,2^1	97.88	2^{11},2^{-1}	95.9	2^{12},2^{-2}	95.14	2^{12},2^{-1}	95.42	2^{12},2^{-1}	92.76
shuttle	2^{11},2^3	99.924	2^{11},2^3	99.938	2^{12},2^4	99.78	2^{12},2^4	99.58	2^{10},2^3	99.814	2^{12},2^4	99.772
mnist	2^6,2^{-5}	97.753	2^7,2^{-5}	97.753	2^7,2^{-6}	96.833	2^9,2^{-6}	96.48	2^4,2^{-5}	96.578	2^{12},2^{-5}	96.129
ijcnn1	2^1,2^1	98.76	2^0,2^0	98.692	2^{12},2^{-3}	95.949	2^{-2},2^{-2}	91.676	2^{12},2^{-3}	96.813	2^{12},2^{-1}	96.11
protein	2^1,2^{-3}	69.97	2^1,2^{-3}	69.793	2^1,2^{-5}	65.957	2^2,2^{-6}	66.244	2^0,2^{-5}	65.957	2^{11},2^{-6}	66.138

Table 5.3: A comparison on RSVM: number of support vectors

	SVM		RSVM			
	LIBSVM	LIBSVM-q	SSVM	LS-SVM	LSVM	Decomposition
Problem	#SV		#SV (all same)			
dna	973	1130	372			
satimage	1611	1822	1826			
letter	8931	8055	13928			
shuttle	285	652	4982			
mnist	8333	8364	12874			
ijcnn1	4555	9766	200			
protein	14770	16192	596			

In Table 5.3 we report the number of "unique" support vectors for each method.

Table 5.4: A comparison on RSVM: training time and testing time (in seconds)

	SVM				RSVM				
	LIBSVM		LIBSVM-q		SSVM	LS-SVM	LSVM	Decomposition	
Problem	training	testing	training	testing	training	training	training	training	testing
dna	7.09	4.65	8.5	5.39	5.04	2.69	23.4	7.59	1.52
satimage	16.21	9.04	19.04	10.21	23.77	11.59	141.17	43.75	11.4
letter	230	89.53	140.14	75.24	193.39	71.06	1846.12	446.04	149.77
shuttle	113	2.11	221.04	3.96	576.1	150.59	3080.56	562.62	74.82
mnist	1265.67	4475.54	1273.29	4470.95	1464.63	939.76	4346.28	1913.86	7836.99
ijcnn1	492.53	264.58	2791.5	572.58	57.87	19.42	436.46	16152.54	6.36
protein	1875.9	687.9	9862.25	808.68	84.21	64.6	129.47	833.35	35

We say "unique" support vectors because for the one-against-one approach, one training data may be a support vector of different binary classifiers. Hence, we report only the number of training data which corresponds to at least one support vector of a binary problem. Note that as we specified, all four RSVM implementations have the same number of support vectors.

For letter and mnist the RSVM approach has a large number of support vectors. A reason is that subsets selected for all binary problems are quite different. This is unlike standard SVM where important vectors may appear in different binary problems so the number of unique support vectors is not that large. An alternative way for RSVM may be to select a subset of all data first and then for each binary problem support vectors are elements in this subset which belong to the two corresponding classes. This will reduce the testing time as [15] has discussed that if the number of classes is not huge, it is proportional to the number of unique support vectors. On the contrary, training time will not be affected much as the size of binary problems is still similar.

Though the best parameters are different, roughly we can see LIBSVM-q requires more support vectors than LIBSVM. This is consistent with the common understanding that the quadratic cost function leads to more data with small nonzero ξ_i. For

protein, LIBSVM and LIBSVM-q both require many training data as support vectors. Indeed most of them are "free" support vectors (i.e. corresponding dual variables α_i not at the upper bound). Such cases are very difficult for decomposition methods and their training time will be discussed later.

We report the training time and testing time in Table 5.4. For four RSVM implementations, we find their testing time is quite close so we post only that of the decomposition implementation. Note that here we report only the time for solving the optimal model. Except the LS-SVM implementation for RSVM which solves a linear equation for each parameter set, the training time of all other approaches depend on parameters. In other words, their number of iterations may vary for different parameter sets. Thus, results here cannot directly imply which method is faster as the total training time depends on the model selection strategy. However, generally we can see for problems with up to tens of thousands of data, the decomposition method for the traditional SVM is still competitive. Therefore, RSVM will be mainly useful for larger problems. In addition, RSVM possesses more advantage on solving large binary problems as for multi-class data sets we can afford to use decomposition methods to solve several smaller binary problems.

Table 5.4 also indicates that the training time of decomposition methods for SVM strongly depends on the number of support vectors. For the problem ijcnn1, compared to the standard LIBSVM the number of support vectors using LIBSVM-q is doubled and then the training time is a lot more. This has been known in the literature as starting from the zero vector as the initial solution, the smaller the number of support vectors (i.e. nonzero elements at an optimal solution) is, the fewer variables may need to be updated in the decomposition iterations. As discussed earlier, protein is a very challenging case for LIBSVM and LIBSVM-q due to the high percentage of

training data as support vectors. For such problem RSVM is a promising alternate as using very few data as support vectors, the computational time is largely reduced and the accuracy does not suffer much.

Among the four RSVM implementations, LS-SVM method is the fastest in the training stage. This is obvious as its cost is just that of one iteration of the SSVM method. Therefore, from the training time of both implementations we can roughly see the number of iterations of the SSVM method. As expected, the Newton's method converges quickly, usually in less than 10 iterations. On the other hand, LSVM which is cheaper for each iteration, needs hundreds of iterations. For quite a few problems it is the slowest.

We observe that for ijcnn1 and protein the testing time of RSVM is much less than that of traditional SVM. On the other hand, for mnist the behavior reverses. We have mensioned how the testing time grows with the number of support vectors during the comparison of support vectors. As a result, for ijcnn1 and protein we reduce the number of support vectors to a few hundreds, so the testing time is extremely little; for mnist our selection strategy results in a large number of support vectors in toto, so the testing time is huge.

It is interesting to see that the decomposition method, not originally designed for linear SVM, performs well on some problems. However, it is not very stable as for some difficult cases, the number of training iterations is prohibitive. Actually, we observe that for problem ijcnn1, the number of nonzero dual variables $\hat{\alpha}$ is extremely large (more than 20000). For such situation, it is very hard for the decomposition implementation to solve the linear SVM dual problem.

5.3 Some modifications on RSVM and their performances

In this section, two types of modifications are applied to the original RSVM formulation. The first is about the regularization term. Remember we mentioned in Chapter III that following the generalized SVM the authors of [21] replace $\frac{1}{2}\bar{\alpha}^T Q_{RR}\bar{\alpha}$ in (3.9) by $\frac{1}{2}\bar{\alpha}^T\bar{\alpha}$ and solve (3.10). So far we only see that for the LSVM implementation, if without doing so we may have troubles to obtain and use the dual problem. For SSVM and LS-SVM, $\frac{1}{2}\bar{\alpha}^T Q_{RR}\bar{\alpha}$ can be kept and the same methods can still be applied. As the change leads to the loss of the property on maximizing the margin, we are interested in whether the performance (testing accuracy) is worsen or not.

Without changing the $\frac{1}{2}\bar{\alpha}^T Q_{RR}\bar{\alpha}$ term in (4.8), LS-SVM formulation is

$$\min_{\widetilde{\alpha}} \quad f(\widetilde{\alpha}) = \quad \frac{1}{2C}\widetilde{\alpha}^T Q_{RR}\widetilde{\alpha} + \sum_{i=1}^{l}(e - \widetilde{Q}\widetilde{\alpha})_i^2 \tag{5.1}$$

Thus we will solve a different linear system:

$$(\widetilde{Q}^T\widetilde{Q} + \frac{1}{2C}Q_{RR})\widetilde{\alpha} = \widetilde{Q}^T e. \tag{5.2}$$

(4.9) is a positive definite linear system because $I/2C$ is positive definite. However, (5.2) does not share this property, since in some cases, $Q_{RR}/2C$ is only positive semi-definite. An example when using the RBF kernel is that some training data are at the same point (e.g. dna in our experiments). Therefore, Cholesky factorization which we used to solve (4.9) can fail in solving (5.2). Thus, LU factorization is used. This change affects little to the training time according to the time complexity analysis in Section 3.2. A comparison on the testing accuracy between solving (4.9) and (5.2) is in Table 5.5. We can see that their accuracy is very similar. Therefore, we conclude that the use of a simpler quadratic term in RSVM is basically fine.

The second modification is changing the random selection of support vectors.

Table 5.5: A comparison on modified versions of RSVM: testing accuracy

	LS-SVM								Decomposition			
	(4.9)		(5.2)		$\sum K_{:,i}$+(4.9)		ICF+(4.9)		(4.15)		loosen+(4.15)	
Problem	C,γ	rate	C,γ	rate	C,γ	rate	C,γ	rate	C,γ	rate	C,γ	rate
dna	$2^4,2^{-6}$	92.327	$2^4,2^{-10}$	93.086	$2^2,2^{-6}$	92.833	$2^3,2^{-8}$	91.737	$2^9,2^{-6}$	92.327	$2^{11},2^{-5}$	89.713
satimage	$2^{12},2^{-3}$	89.9	$2^8,2^{-3}$	89.7	$2^{11},2^{-3}$	88.45	$2^3,2^{-1}$	90.35	$2^{11},2^{-1}$	90	$2^{12},2^0$	90.2
letter	$2^{12},2^{-2}$	95.14	$2^{12},2^{-3}$	94.62	$2^{12},2^{-1}$	94.38	$2^{12},2^{-2}$	96.02	$2^{12},2^{-1}$	92.76	$2^{12},2^0$	96.44
shuttle	$2^{12},2^4$	99.58	$2^7,2^4$	99.7	$2^3,2^3$	91.572	$2^{12},2^4$	99.697	$2^{12},2^4$	99.772	$2^{12},2^4$	99.814
mnist	$2^9,2^{-6}$	96.48	$2^7,2^{-6}$	96.484	$2^{12},2^{-7}$	95.043	$2^7,2^{-6}$	96.571	$2^{12},2^{-5}$	96.129	$2^{12},2^{-5}$	97.025
ijcnn1	$2^{-2},2^{-2}$	91.676	$2^{-2},2^{-6}$	90.962	$2^3,2^{-1}$	91.738	$2^{-2},2^{-1}$	92.082	$2^{12},2^{-1}$	96.11	$2^{10},2^{-1}$	92.322

The random selection is efficient as it costs $O(lmn)$ time where n is the number of attributes in the training vector. However, we suspect that a more careful selection might improve the testing accuracy. We try several heuristics to select a subset of training data which we consider more important as support vectors, and then use the LS-SVM implememtation to solve the reduced problem. They are listed as follows:

1. For each column of K, we calculate the sum of all entries in that column. Then we take the m vectors corresponding to the columns with larger sums. Since RBF kernel is used, all entries are positive, and so are the sums. We think columns with larger sums might be more important. The main work for this strategy is to obtain all entries of K, which costs $O(l^2n)$ time. This selection strategy is denoted as "$\sum K_{:,i}$" in Table 5.5.

2. Conducting incomplete Cholesky factorization with symmetric pivoting on K (This will be detailed in Chapter VI), and the first m pivoted indices are used for selecting support vectors. This is a by-product of a factorization process for K. We use an algorithm to perform imcomplete Cholesky factorization in $O(lm^2)$ time, and denote it as "ICF" strategy in Table 5.5.

3. Train the original SVM using conventional decomposition methods with a very

loose stopping tolerance (here BSVM with $\epsilon = 0.1$ is used). BSVM solves a problem of the form (4.15), and those dual variables equal to C are called bounded dual variables. Since they are nonzero, the corresponding training data are all support vectors, but a notable property states that all misclassified training data are included in these support vectors. For RSVM case where only a few support vectors are selected, if we select all these bounded support vectors, we find that RSVM is severely affected by these points and constructs a bad model. Thus, we mainly select the free support vectors (corresponding to dual variables not equal to C) of BSVM. If there are less than m free support vectors, we randomly select other training data, including bounded support vectors. We find that loosing the stopping tolerance reduces not much of the training time for BSVM so overall this strategy is slower than BSVM with default tolerance. However, from this strategy we can see whether an approximated solution of decomposition methods can help to select support vectors of RSVM. We denote it as "loosen" strategy in Table 5.5.

The result is also shown in Table 5.5. For some problems, "pivot" and "loosen" improve the training accuracy very little. " $\sum K_{:,i}$ " in general does not improve the accuracy, and for shuttle the result is poor. Compared with Table 5.2, all improvements do not make RSVM competable with the original SVM, so we suggest the original simple and fast random selection.

CHAPTER VI

ICF Approximated Kernel Representation for SVM

As we have mentioned before, the difficulty for solving the quadratic programming problem (2.7) is the dense matrix Q, which is so large that we can neither store it in memory nor efficiently compute its inverse. RSVM has circumvented this problem by modifying the problem so only some columns of Q are used during the optimization. This is an example of approximating the large dense Q by its sub-matrix, and there exists other approximations. For example, a greedy approximation [43] approximates Q by an $l \times m$ basis matrix which is linear combinations of some columns of Q. Their algorithms, like RSVM, involve random selections as well. Here, we study another technique which approximates Q by its incomplete Cholesky factorization (ICF). This idea was proposed in [11, Section 4], and in this chapter we will detail the concept and implementation for this kind of reduction, and compare it with RSVM.

6.1 Incomplete Cholesky factorization (ICF)

Linear systems can be solved in two popular schemes: direct methods such as the Gaussian elimination, and iterative methods, which approximate the actual solution by repeatedly solving simpler problems. The conjugate gradient method (CG) is

one major approach of iterative methods, but a problem of CG is that it sometimes converges very slowly. A modern approach suggested in the survey [10] is the preconditioned CG method. When solving the linear system $Qx = b$ for x, the first step of this approach is obtaining a preconditioner V from Q. Then we use CG to solve the system $(V^{-1}QV^{-T})y = V^{-1}b$ for y, and finally solve $V^T x = y$ for x. If the preconditioner V is chosen well, CG converges fast and the linear system is solved efficiently. If Q is symmetric positive definite, a perfect choice is the Cholesky factorization satisfying $Q = VV^T$, which transforms Q to the identity matrix. However, obtaining the Cholesky factorization is usually as hard as solving the linear system. We then search for V which can be obtained more easily and VV^T approximates Q. Such a matrix V is called the incomplete Cholesky factorization (ICF). ICF has been a general way for obtaining preconditioners, especially for solving sparse linear systems. However, in this Chapter we will focus on the use of ICF for handling problems involving dense matrices.

Currently most ICF methods are modifications on traditional Cholesky factorization algorithms. The modifications may be, for example, discarding too small entires of Q or use fast but inaccurate operations in ICF algorithms. Based on different orders of loops and indexes, there are several versions of Cholesky factorization [33, Appendix]. However, since we do not store the dense matrix but calculate each element during the factorization, only some special forms are usable. For example, we consider the popular outer product form [13, Section 10.3.2], which is based on the following formula:

$$\begin{bmatrix} \alpha & v^T \\ v & B \end{bmatrix} = \begin{bmatrix} \sqrt{\alpha} & 0 \\ \frac{v}{\sqrt{\alpha}} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & B - \frac{vv^T}{\alpha} \end{bmatrix} \begin{bmatrix} \sqrt{\alpha} & \frac{v^T}{\sqrt{\alpha}} \\ 0 & I \end{bmatrix}. \tag{6.1}$$

In the beginning, the left-hand side is the whole matrix Q. The vector $\begin{bmatrix} \sqrt{\alpha} \\ \frac{v}{\sqrt{\alpha}} \end{bmatrix}$ is the leftmost column of V, and the same procedure is recursively applied to the sub-matrix $B - \frac{vv^T}{\alpha}$.

In each iteration of the Cholesky factorization procedure, since we are not able to store B (it can be as large as Q), $B - \frac{vv^T}{\alpha}$ is also not stored. This means that we must delay the operation of subtracting $\frac{vv^T}{\alpha}$ until the respective column is considered. In other words, if v is the jth column, elements of the jth column of Q are not computed until the jth step. The jth column of V is computed by using the 1st to $(j-1)$st columns of V.

For the ICF, each column of V can also be computed by this strategy. In the next section we will consider two of such ICF methods.

6.2 ICF algorithms

The first method discussed here is proposed in [24], which examines all entries of Q, but in each step only the largest m elements of v are stored as a column of V. When we compute the later v's, we use the previous stored columns of V for updating them. [24] showed that in practice, the time complexity of this algorithm is $O(lm^2)$. The memory requirement is $O(lm)$ so it can be pre-allocated conveniently. This situation is similar to that of RSVM. Since v is not wholly stored as a column of V, the updating is not exact. Therefore, after a number of steps, negative diagonals may occur. That is, in (6.1) the entry α is negative. The process cannot continue since we must take the square root of α. When this algorithm encounters such a problem, it adds βI to Q and restarts itself. If unfortunately the same problem occurs again, β is iteratively increased. We observe that for some problems, β becomes so large that the resulting Q is dominated by βI. Then the ICF may not be a good approximation

of Cholesky factorization.

for $j = 1 : m$
  /* begin{symmetric pivoting} */
  for $i = j : l$
    $G_{ii} = Q_{ii}$
    for $k = 1 : j - 1$
      $G_{ii} -= G_{ik}^2$
    end{for k}
  end{for i}
  Find $i^* = argmax_{i=j:l} G_{ii}$.
  $G_{j:l,l} = Q_{i^*:l,l}$
  Swap $G_{j,1:j}$ and $G_{i^*,1:j}$.
  /* end{symmetric pivoting} */
  for $k = 1 : j - 1$
    $G_{j+1:l,j} -= G_{j+1:l,k} G_{j,k}$
  end{for k}
  $G_{jj} = \sqrt{G_{jj}}$
  $G_{j+1:l,j} /= G_{jj}$
end{for j}

Algorithm VI.1: Column-wise ICF with symmetric pivoting

The second algorithm is proposed in [11, Section 4], which is an early stopping version of the Cholesky factorization. It stops generating the columns of the Cholesky factorization when the trace of the remaining submatrix B is smaller than a stopping tolerance. In each step, the updating procedure is exactly the same as the Cholesky factorization, so it inherits the good property that negative diagonals never occur. Moreover, while [24] does not perform pivoting, [11] implements symmetric pivoting to improve numerical stability. This is done by swapping the column and row with

the largest diagonal to current column and row. Here instead of checking the trace of the remaining submatrix, we stop the algorithm after m columns are generated. This also results in looking $O(lm)$ entries of Q, which is also the same as RSVM. We put the modified procedure in Algorithm VI.1. Each iteration consists of symmetric pivoting and one column update, and both cost $O(lm)$ time. Hence, the total time complexity is $O(lm^2)$.

6.3 Using ICF in SVM

In [11] the authors proposed to use interior point method (IPM) to solve the SVM dual problem (4.16) with a low rank kernel matrix Q. The need for a low rank kernel matrix results from the most time-consuming operation on every step of IPM, that is, calculating $(D+Q)^{-1}u$ where D is a diagonal matrix and u is some vector. If we already have the low rank representation $Q = VV^T$ where V is an $l \times m$ matrix with $m \ll l$, the calculation is efficient. Actually we have introduced one approach, the SMW identity (4.3), which can invert $(D+VV^T)$ in $O(lm^2)$ time. Another approach is detailed in [11, Section 3.2] with the same order of time.

We observe that if linear kernel is used, for example $Q = diag(y)XX^Tdiag(y)$ in (4.16), then $V = diag(y)X$ is a low-rank matrix, as long as X has more rows than columns. That is, in SVM input, the number of training data vectors l is more than the number of attributes n. However, for other situations, such as using the RBF kernel, Q has full rank, so the low rank representation of Q does not exist. Such problem is overcome in [11, Section 4] by introducing the ICF approximation. A low rank ICF VV^T is obtained, then $\tilde{Q} = VV^T$ substitutes Q in the IPM procedure. The resulting optimal dual solution $\hat{\alpha}$ is considered as an approximate solution of the original dual and is used to construct $w = \sum_{i=1}^{l} y_i\hat{\alpha}_i\phi(x_i)$. Also, the decision function

is the same as the original SVM (compared with (3.4) and (2.8)). Note that every x_i corresponding to nonzero $\hat{\alpha}_i$ is saved as the support vectors. This is different from RSVM as the latter fixes the support vectors in advance. Consequently, for the ICF approximation method, it is interesting to see whether the support vectors will be sparse or not.

Here we show that except IPM, other methods which focus on solving linear SVM are possible to solve the SVM problem with an ICF kernel representation. We change the kernel matrix Q in the dual problem (4.15) to its ICF $Q = VV^T$:

$$\begin{aligned} \min_{\hat{\alpha}} \quad & \frac{1}{2}\hat{\alpha}^T(VV^T + yy^T)\hat{\alpha} - e^T\hat{\alpha} \\ \text{subject to} \quad & 0 \leq \hat{\alpha}_i \leq C, i = 1, \ldots, l. \end{aligned} \tag{6.2}$$

We observe that (6.2) is already in the dual form of a linear SVM with more data points than attributes. Therefore, all methods suitable for the dual problem of linear SVM can be applied without modification.

6.4 Implementations

We have described several techniques for solving linear SVM problems in Chapter IV. The solvers focusing on the dual form (LSVM and decomposition) can be immediately applied for (6.2). Here we construct the two ICF described in Section 6.2, then respectively solve the linear SVM problem (6.2) by the decomposition method. We denote them by ICFSVM in our experiments. The total time complexity for ICFSVM is $O(lm^2) + \#iterations \times O(lm)$.

We want to see whether we can use the implementations for linear SVM in the

primal form. Clearly the primal form of (6.2) can be considered as a linear SVM:

$$\begin{aligned} \min_{\widetilde{\alpha},\xi} \quad & \frac{1}{2}\widetilde{\alpha}^T\widetilde{\alpha} + C(\sum_{i=1}^{l}\xi_i) \\ \text{subject to} \quad & V\widetilde{\alpha} \geq e - \xi, \\ & \xi \geq 0. \end{aligned} \tag{6.3}$$

Although solving (6.3) is as easy as solving a linear SVM problem, we have difficulties on explaining the solution. (6.3) is a linear SVM problem with "virtual" data points which are the rows of V. The solution $\widetilde{\alpha}$ is the normal vector of the hyperplane which separates the "virtual" data points. That is, for optimal value of $\hat{\alpha}$ we have $\widetilde{\alpha} = \sum_i \hat{\alpha}_i V_{i,:}$, so $\hat{\alpha}_i$ actually represents the weight of $V_{i,:}$, which is very different from $\phi(x_i)$. Thus in the prediction stage, since we do not know how to transform the test data into the "virtual" space, the separating hyperplane is not usable. To conclude, we find it impossible to make use of the solution of the primal problem (6.3). Therefore, we do not apply those implementations (SSVM, LS-SVM) which solve the primal form of SVM.

An inference from this observation is about one concluding remark made in [11] which believes that the set of nonzero dual variables (support vectors) obtained by ICFSVM is (almost) the same as the set obtained by solving standard SVM dual form. From the earlier discussion, we doubt this assertion since the resulting nonzero dual variables in (6.2) are only meaningful in constructing the separating hyperplane of the "virtual" data points in (6.3) , and we can hardly believe that the actual w will be better constructed by the linear combinations of $\phi(x_i)$, where each x_i is the training vector corresponding to a nonzero $\hat{\alpha}_i$. We also conduct some experiments on this issue in the following section.

An additional implementation is combining the ICFSVM technique with RSVM.

Some information from the ICF process can be used to select the support vectors for RSVM procedure. Here we use the data corresponding to the m chosen columns in Algorithm VI.1 as the support vectors, and then follow the regular RSVM routines. Experimental results using LS-SVM implementation have been shown in Table 5.5. It has similar accuracy with other RSVM modifications discussed in Section 5.3.

6.5 Experiments and results

We use the same target problems as RSVM experiments, with the same settings mentioned in Chapter V. In particular, here m controls the number of entries of the ICF matrix V, and it is selected in the same way as for RSVM: in most cases m is set to be 10% of the training data. However, since the number of dual variables is still l, the number of support vectors are not the same as RSVM; instead, it depends on the sparsity of the solution.

We compare five methods:

1. LIBSVM, the standard decomposition method for nonlinear SVM
2. Decomposition implementation for RSVM
3. ICFSVM using the first ICF algorithm (denoted as [24])
4. ICFSVM using the second ICF algorithm (denoted as (VI.1))
5. Re-training on the support vectors after 4.

To reduce the search space of parameter sets, here we consider only the RBF kernel $K(x_i, x_j) \equiv e^{-\gamma\|x_i - x_j\|^2}$, so the parameters left for decision are kernel parameter γ and cost parameter C. For all methods, we conduct model selection on the training data where the test data are assumed unknown. For each problem, we estimate the

generalized accuracy using different $\gamma = [2^4, 2^3, 2^2, \ldots, 2^{-10}]$ and $C = [2^{12}, 2^{11}, 2^{10}, \ldots, 2^{-2}]$. Therefore, for each problem we try $15 \times 15 = 225$ combinations. For each pair of (C, γ), the validation performance is measured by training 70% of the training set and testing the other 30% of the training set. The best parameter set is used for constructing the model for future testing. In addition, for multi-class problems we consider that C and γ of all $k(k-1)/2$ binary problems via the one-against-one approach are the same.

Table 6.1 shows the testing accuracy of all methods. ICFSVM using the second ICF algorithm achieves the highest rate. However, the best rate is similar to RSVM, so for moderate-sized training set ICFSVM is also not recommended. We also note that the optimal C parameter for ICFSVM is smaller than that of RSVM. This nice property helps the decomposition implementation since it is more stable for normal parameters, especially for moderately small C's. In addition, it is easier to apply different model selection techniques on ICFSVM than RSVM: many of them can work only on the region where C and γ are not too small or too large.

In Table 6.2 we report the number of "unique" support vectors at the optimal model of each method. We observe that the result of ICFSVM using the second algorithm is similar to that of the original SVM. Hence, we can say that such a method also gains the SVM property that support vectors are often sparse. The number of support vectors for RSVM is decided in advance, and is not similar to other techniques.

We report the training time and testing time for solving the optimal model in Table 6.3. Since ICFSVM has to perform ICF, an extra work before solving (6.2), the training time may be more than RSVM whose main work is solving (4.15), a QP similar to (6.2). Moreover, from the table we see that for some problems solving

Table 6.1: A comparison on ICFSVM: testing accuracy

	SVM		RSVM		ICFSVM(decomposition implementation)					
	LIBSVM		Decomposition		[24]		(VI.1)		(VI.1)+retrain	
Problem	C, γ	rate	C, γ	rate	C, γ	rate	C, γ	rate	C, γ	rate
dna	$2^4, 2^{-6}$	95.447	$2^9, 2^{-6}$	92.327	$2^{-1}, 2^{-4}$	66.273	$2^0, 2^{-9}$	92.917	$2^2, 2^{-4}$	87.436
satimage	$2^4, 2^0$	91.3	$2^{11}, 2^{-1}$	90	$2^{-1}, 2^1$	87.2	$2^6, 2^{-5}$	89.2	$2^1, 2^1$	72.65
letter	$2^4, 2^2$	97.98	$2^{12}, 2^{-1}$	92.76	$2^2, 2^2$	96.66	$2^5, 2^{-2}$	96.16	$2^0, 2^2$	94.06
shuttle	$2^{11}, 2^3$	99.924	$2^{12}, 2^4$	99.772	$2^{-1}, 2^4$	94.4	$2^9, 2^4$	99.862	$2^0, 2^{-1}$	94.331
mnist	$2^6, 2^{-5}$	97.753	$2^{12}, 2^{-5}$	96.129	N/A	N/A	$2^1, 2^{-7}$	96.045	$2^1, 2^{-5}$	95.259
ijcnn1	$2^1, 2^1$	98.76	$2^{12}, 2^{-1}$	96.11	$2^{-2}, 2^{-10}$	90.185	$2^9, 2^{-5}$	92.274	$2^{-2}, 2^0$	91.711
protein	$2^1, 2^{-3}$	69.97	$2^{11}, 2^{-6}$	66.138	N/A	N/A	$2^{-1}, 2^{-6}$	65.398	$2^{-2}, 2^{-5}$	65.926

N/A: training time too large to apply the model selection

Table 6.2: A comparison on ICFSVM: number of support vectors

	SVM	RSVM	ICFSVM		
	LIBSVM	Decomposition	[24]	(VI.1)	(VI.1)+retrain
Problem	#SV				
dna	973	372	1688	1389	1588
satimage	1611	1826	4022	1187	1507
letter	8931	13928	12844	5390	8953
shuttle	285	4982	43026	308	3714
mnist	8333	12874	N/A	5295	5938
ijcnn1	4555	200	49485	4507	8731
protein	14770	596	N/A	15049	15512

N/A: training time too large to apply the model selection

Table 6.3: A comparison on ICFSVM: training time and ICF time (in seconds)

	SVM	RSVM	ICFSVM					
	LIBSVM	Decomposition	[24]		(VI.1)		(VI.1)+retrain	
Problem	training	training	training	ICF	training	ICF	training	ICF
dna	7.09	7.59	440.41	427.18	9.62	5.45	33.77	5.52
satimage	16.21	43.75	558.23	467.48	48.49	28.37	61.59	28.32
letter	230	446.04	3565.31	2857.95	484.59	222.4	635.41	221.93
shuttle	113	562.62	70207.76	13948.14	1251.17	1184.63	1811.6	1265.51
mnist	1265.67	1913.86	N/A	N/A	2585.13	2021.64	2565.08	1866.9
ijcnn1	492.53	16152.54	21059.3	4680.63	5463.8	103.97	1579.73	102.52
protein	1875.9	833.35	N/A	N/A	217.53	92.52	3462.57	110.54

N/A: training time too large to apply the model selection

ICF occupies most of the training time. However, for the best model the training time of ICFSVM may not be more than RSVM. This is due to that ICFSVM has the best model at smaller C. Now they both use dcomposition methods which can be very slow if C is large (see discussions in [15]). Thus, if proper model selection techniques are used, the overall training time of ICFSVM can be competitive with that of RSVM. In addition, if in the model selection process, a line search procedure is need by fixing other parameters and varing C, we only have to do ICF once. This may be another advantage of ICFSVM. However, we have mentioned that RSVM can be solved through both primal and dual forms. So in this aspect, it is more flexible and efficient if only a single parameter set is considered.

The performance for re-training the support vectors obtained by ICFSVM is generally worse than the original ICFSVM. Moreover, the optimal parameters for all problems are quite different. This indicates that ICFSVM actually finds another set of support vectors, and re-training on this set will lead to quite different model.

The result for the first ICF method is even strange. Although for some problems it is comparable with other methods, for other problems we find the performance extremely bad, which shows that the trained model is quite far from the original SVM model. Moreover, we cannot provide the result for mnist and protein due to the huge training time for each model. The reason is that for these problems, the resulting ICF is not close to the kernel matrix, and we have the details in the discussion of this algorithm. As a consequence of the discussion, we can explain for the huge training time: the frequent restarting of the ICF process wastes much time. Moreover, as the identity matrix instead of the kernel matrix dominates the approximated problem, we find that many training models with different parameters are alike. That is, the optimal model are less sensitive for parameters C and γ.

CHAPTER VII

Discussions and Conclusions

In this thesis we first discuss four multi-class implementations for RSVM and compared them with two decomposition methods based on LIBSVM. Experiments indicate that in general the test accuracy of RSVM is a little worse than that of the standard SVM. Though RSVM keeps similar constraints as the primal form of SVM, restricting support vectors from a randomly selected subset still downgrades the performance. Moreover, we show that it is hard to improve the performance by changing the selection strategy. For the training time which is the main motivation of RSVM we show that based on the current implementation techniques, RSVM will be faster than regular SVM on large problems or some difficult cases with many support vectors. Therefore, for median-sized problems, standard SVM should be used but for large problems, as RSVM can effectively restrict the number of support vectors, it can be an appealing alternate. Regarding the implementation of RSVM, least-square SVM (LS-SVM) is the fastest among the four methods compared here though its accuracy is a little lower. Thus, for very large problems it is appropriate to try this implementation first.

We then discuss the implementations for applying ICF approximated kernel to SVM problem. Experiments show that the testing accuracy is also a little lower

than that of traditional SVM. Compared with RSVM, we avoid the random selection of support vectors by the ICF strategy, but the approximation does not seem good enough for generating good models as SVM. Moreover, though RSVM and ICFSVM both can be implemented by linear SVM solvers, ICFSVM is restricted to those solvers focusing on the dual form. For those solvers focusing on the primal form, we explain why their solutions cannot be used for generating ICFSVM models. However, the optimal penalty parameter of ICFSVM is smaller than that of RSVM, which is usually a merit for model selection strategies. Also, we collect the set of support vectors obtained by ICFSVM, and then use SVM solvers to train on the set. Experimental results shows this extra work will downgrade the testing accuracy more.

BIBLIOGRAPHY

[1] K. P. Bennett, D. Hui, and L. Auslender. On support vector decision trees for database marketing. Department of Mathematical Sciences Math Report No. 98-100, Rensselaer Polytechnic Institute, Troy, NY 12180, Mar. 1998.

[2] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 1992.

[3] M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. S. Furey, M. Ares, Jr., and D. Haussler. Knowledge-based analysis of microarray gene expression data using support vector machines. *Proceedings of the National Academy of Science*, 97(1):262–267, 2000.

[4] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[5] C.-C. Chang and C.-J. Lin. IJCNN 2001 challenge: Generalization ability and text decoding. In *Proceedings of IJCNN*, 2001. Winner of IJCNN Challenge.

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

[7] C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

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

[9] D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46:161–190, 2002.

[10] A. Edelman. Large dense numerical linear algebra in 1993: The parallel computing influence. *International J. Supercomputer Appl.*, 7:113–128, 1993.

[11] S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machines Learning Research*, 2:243–264, 2001.

[12] T.-T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: a fast and simple learning procedure for support vector machines. In *Proceedings of 15th Intl. Conf. Machine Learning*. Morgan Kaufman Publishers, 1998.

[13] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, second edition, 1989.

[14] M. Heiler. Optimization criteria and learning algorithms for large margin classifiers. Master's thesis, University of Mannheim, Germany, Department of Mathematics and Computer Science, Computer Vision, Graphics, and Pattern Recognition Group, D-68131 Mannheim, Germany, 2001.

[15] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[16] C.-W. Hsu and C.-J. Lin. A simple decomposition method for support vector machines. *Machine Learning*, 46:291–314, 2002.

[17] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

[18] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of International Conference on Machine Learning*, 1999.

[19] Y. LeCun. MNIST database of handwritten digits. Available at `http://www.research.att.com/~yann/exdb/mnist`.

[20] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In F.Fogelman and P.Gallinari, editors, *International Conference on Artificial Neural Networks*, pages 53–60, Paris, 1995. EC2 & Cie., 1995.

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

[22] C.-J. Lin. Formulations of support vector machines: a note from an optimization point of view. *Neural Computation*, 13(2):307–317, 2001.

[23] C.-J. Lin and J. J. Moré. Newton's method for large-scale bound constrained problems. *SIAM J. Optim.*, 9:1100–1127, 1999.

[24] C.-J. Lin and R. Saigal. An incomplete Cholesky factorization for dense matrices. *BIT*, 40:536–558, 2000.

[25] K.-M. Lin and C.-J. Lin. A study on reduced support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2002.

[26] O. L. Mangasarian. Generalized support vector machines. In B. S. A. J. Smola, P. Bartlett and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 135–146. MIT Press, 2000.

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

[28] O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, 2001.

[29] N. Matic, I. Guyon, J. Denker, and V. Vapnik. Writer adaptation for on-line handwritten character recognition. In I. C. S. Press, editor, *In Second International Conference on Pattern Recognition and Document Analysis*, pages 187–191, Tsukuba, Japan, 1993.

[30] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Prentice Hall, Englewood Cliffs, N.J., 1994. Data available at anonymous ftp: `ftp.ncc.up.pt/pub/statlog/`.

[31] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 243–254, Cambridge, MA, 1999. MIT Press.

[32] M. Orr. Introduction to radial basis function networks. Technical report, Institute for Adaptive and Neural Computation, Edinburgh University, 1996. `http://www.anc.ed.ac.uk/#mjo/rbf.html`.

[33] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems*. Plenum Press, New York, 1988.

[34] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of CVPR'97*, New York, NY, 1997. IEEE.

[35] E. Osuna and F. Girosi. Reducing the run-time complexity of support vector machines. In *Proceedings of International Conference on Pattern Recognition*, 1998.

[36] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *International Conference on Computer Vision ICCV'98*, 1998.

[37] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

[38] D. Prokhorov. IJCNN 2001 neural network competition. Slide presentation in IJCNN'01, Ford Research Laboratory, 2001. `http://www.geocities.com/ijcnn/nnc_ijcnn01.pdf` .

[39] M. Rychetsky, S. Ortmann, and M. Glesner. Construction of a support vector machine with local experts. In *Workshop on Support Vector Machines at the International Joint Conference on Artificial Intelligence (IJCAI 99)*, 1999.

[40] B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors. *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, MA, 1998.

[41] B. Schölkopf and A. J. Smola. *Learning with kernels*. MIT Press, 2002.

[42] B. Schölkopf, K.-K. Sung, C. J. C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classiers. *IEEE Transactions on Signal Processing*, 45(11):2758–2765, 1997.

[43] A. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *International Conference on Machine Learning*, 2000.

[44] M. O. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, and J. Weston. Support vector regression with ANOVA decomposition kernels. In Schölkopf et al. [40], pages 285–292.

[45] J. Suykens and J. Vandewalle. Least square support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[46] J. Suykens and J. Vandewalle. Multiclass LS-SVMs: Moderated outputs and coding-decoding schemes. In *Proceedings of IJCNN*, Washington D.C., 1999.

[47] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY, 1995.

[48] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY, 1998.

[49] J.-Y. Wang. Application of support vector machines in bioinformatics. Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University, 2002.

[50] R. C. Whaley and J. J. Dongarra. Automatically tuned linear algebra software. Technical Report UT-CS-97-366, 1997.

APPENDICES

APPENDIX A

The Relation between (3.7) and (3.1)

The derivation from (3.5) and (3.6) to (3.7) shows only that if (w^*, b^*, ξ^*) and α^* are primal and dual optimal solutions, respectively, then (α^*, b^*, ξ^*) is feasible for (3.7). Hence, before using (3.7) in practice we want to make sure that any optimal (α, ξ, b) of (3.7) can construct an optimal w for the primal problem (3.1).

It is easy to see this. By using $w = \sum_{i \in B} y_i \alpha_i \phi(x_i)$, (w, b, ξ) is feasible for (3.1). Since we have shown that (α^*, b^*, ξ^*) is feasible for (3.7),

$$\begin{aligned} & \frac{1}{2}(w^*)^T w^* + (b^*)^2 + C \sum_{i=1}^{l} (\xi_i^*)^2 \\ = \; & \frac{1}{2}(\alpha^*)^T Q \alpha^* + (b^*)^2 + C \sum_{i=1}^{l} (\xi_i^*)^2 \\ \geq \; & \frac{1}{2}\alpha^T Q \alpha + b^2 + C \sum_{i=1}^{l} \xi_i^2 \\ = \; & \frac{1}{2} w^T w + b^2 + C \sum_{i=1}^{l} \xi_i^2. \end{aligned}$$

Thus, (w, b, ξ) is optimal for the primal.

APPENDIX B

Derivations of (4.4) and (4.5)

In order to simplify the complex $P_\beta(e - \widetilde{Q}\widetilde{\alpha})$ term, we define the vector

$$v \equiv \exp(-\beta(e - \widetilde{Q}\widetilde{\alpha})), \tag{B.1}$$

and then

$$P_\beta(e - \widetilde{Q}\widetilde{\alpha}) = (e - \widetilde{Q}\widetilde{\alpha}) + \beta^{-1}\log(1 + v). \tag{B.2}$$

The differentiations of v and P_β are:

Finally we obtain the gradient and Hessian of f:

$$
\begin{aligned}
& \frac{\partial^2 f}{\partial\widetilde{\alpha}_i \partial\widetilde{\alpha}_j} = \frac{\partial}{\partial\widetilde{\alpha}_j}(\frac{\partial f}{\partial\widetilde{\alpha}_i}) \\
= \; & \frac{\partial}{\partial\widetilde{\alpha}_j}(\frac{\partial}{\partial\widetilde{\alpha}_i}(-2(\sum_{k=1}^{l}\widetilde{Q}_{ki}P_\beta(e-\widetilde{Q}\widetilde{\alpha})_k(1+v_k)^{-1}))) + \frac{1}{C}\frac{\partial}{\partial\widetilde{\alpha}_j}\widetilde{\alpha}_i \\
= \; & -2(\sum_{k=1}^{l}\widetilde{Q}_{ki}\frac{\partial}{\partial\widetilde{\alpha}_j}(P_\beta(e-\widetilde{Q}\widetilde{\alpha})_k(1+v_k)^{-1})) + \frac{1}{C}\delta_{ij} \\
= \; & -2(\sum_{k=1}^{l}\widetilde{Q}_{ki}(\frac{\partial}{\partial\widetilde{\alpha}_j}(P_\beta(e-\widetilde{Q}\widetilde{\alpha})_k)(1+v_k)^{-1} + P_\beta(e-\widetilde{Q}\widetilde{\alpha})_k\frac{\partial}{\partial\widetilde{\alpha}_j}(1+v_k)^{-1})) + \frac{1}{C}\delta_{ij} \\
= \; & -2(\sum_{k=1}^{l}\widetilde{Q}_{ki}(-\widetilde{Q}_{kj}(1+v_k)^{-2} + P_\beta(e-\widetilde{Q}\widetilde{\alpha})_k(-1)(1+v_k)^{-2}\beta\widetilde{Q}_{kj}v_k)) + \frac{1}{C}\delta_{ij} \\
= \; & 2(\sum_{k=1}^{l}\widetilde{Q}_{ki}\widetilde{Q}_{kj}(1+v_k)^{-2}(1+\beta v_k P_\beta(e-\widetilde{Q}\widetilde{\alpha})_k)) + \frac{1}{C}\delta_{ij},
\end{aligned}
$$

where δ_{ij} is defined as

If we define $\mathrm{diag}(v) \equiv \begin{bmatrix} v_1 & & 0 \\ & \ddots & \\ 0 & & v_d \end{bmatrix}$ for vector v with dimension d, we obtain a more clear matrix form: