A study on reduced support vector machines

Article *in* IEEE Transactions on Neural Networks · February 2003

DOI: 10.1109/TNN.2003.820828 · Source: PubMed

2 authors:

Kuan-Ming Lin
National Taiwan University
6 PUBLICATIONS **303** CITATIONS

SEE PROFILE

Chih-Jen Lin
National Taiwan University
154 PUBLICATIONS **70,233** CITATIONS

SEE PROFILE

A Study on Reduced Support Vector Machines

Kuan-Ming Lin and Chih-Jen Lin

Department of Computer Science and

Information Engineering

National Taiwan University

Taipei 106, Taiwan (cjlin@csie.ntu.edu.tw)

Abstract

Recently the Reduced Support Vector Machine (RSVM) was proposed as an alternate of the standard SVM. Motivated by resolving the difficulty on handling large data sets using SVM with nonlinear kernels, it preselects a subset of data as support vectors and solves a smaller optimization problem. However, several issues of its practical use have not been fully discussed yet. For example, we do not know if it possesses comparable generalization ability as the standard SVM. In addition, we would like to see for how large problems RSVM outperforms SVM on training time. In this paper we show that the RSVM formulation is already in a form of linear SVM and discuss four RSVM implementations. Experiments indicate that in general the test accuracy of RSVM are 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, from this empirical study, RSVM will be mainly useful for either larger problems or those with many support vectors. Experiments in this paper also serve as comparisons of (1) different implementations for linear SVM; and (2) standard SVM using linear and quadratic cost functions.

I. Introduction

We consider the support vector machine (SVM) with a quadratic cost function [25]: Given a training set of instance-label pairs $(x_i, y_i), i = 1, \ldots, l$ where $x_i \in R^n$ and $y_i \in \{1, -1\}$, the support vector technique solves the following optimization problem:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C(\sum_{i=1}^{l} \xi_i^2) \\ \text{subject to} \quad & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i. \end{aligned} \tag{1}$$

Since $\phi(x)$ maps x into a higher (maybe infinite) dimensional space, practically we solve its dual, a quadratic programming problem with the number of variables equal to l :

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

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

For large-scale problems, the difficulty of solving (2) results from the fully dense matrix $(Q + \frac{I}{2C})$, 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, [18], [20], [8]) 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. On the other hand, if the linear kernel (i.e. $K(x_i, x_j) = x_i^T x_j$) is used, Q is directly the product of a rectangular matrix and its transpose so we can exploit more possibilities to design efficient algorithms. Thus, here we mainly consider the more difficult case by using nonlinear kernels.

Recently [10] 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 candidates of support vectors. The smaller matrix can be stored in memory, so optimization algorithms such as Newton method can be applied. RSVM is different from directly solving smaller SVM problems with a subset of training because as the l constraints in the primal problem (1) are still kept during the optimization process. In [10] the authors showed the performance (testing accuracy) is as good as the regular SVM. As RSVM essentially uses less information than the regular SVM, we feel there is a need to conduct some serious comparisons. This is the first aim of the paper. Our experiments show that in general the testing accuracy of RSVM is a little lower than SVM.

Moreover, as the training efficiency is the main motivation of RSVM, we would like to discuss its different implementations, and compare their training time with the regular SVM. In addition, we investigate for how large problems, RSVM becomes an efficient alternate of SVM. Results show that for problems with up to tens of thousands of data, if the percentage of support vectors is not high, existing implementations for SVM are quite competitive on the training time. Thus, from this empirical study, RSVM will be mainly useful for either larger problems or those with many support vectors. We note that results here are mainly empirical and we do not conduct further theoretical analysis about the generalization error.

This paper is organized as follows. In Section II, we outline the key modifications from standard SVM to RSVM. In Section III, we detail the Smooth SVM (SSVM) method used in [11], and apply three more techniques, the Least Square SVM (LS-SVM) [23], the Lagrangian SVM (LSVM) [15], and the decomposition method to solve the RSVM problem. Issues related to practical implementations such as stopping criteria and approaches for solving multi-class problems are in Section IV. Numerical experiments are in Section V which present that the accuracy of RSVM is usually lower than that

of SVM. We also discuss that different techniques for solving RSVM have some minor differences on the accuracy, but using LS-SVM method is the fastest on training. In Section VII, there is an attempt to make modifications on the original RSVM formulation. Finally we have some discussions and conclusions in Section VIII.

II. Reduced Support Vector Machine Formulation

Here we outline the key modifications from standard SVM to RSVM in [10]. They start from adding an additional term $b^2/2$ to the objective function of (1):

$$\min_{w,b,\xi} \quad \frac{1}{2}(w^T w + b^2) + C(\sum_{i=1}^{l} \xi_i^2) \tag{3}$$

$$\text{subject to} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i. \tag{4}$$

Its dual form becomes a simpler bound-constrained problem:

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

$$\text{subject to} \quad 0 \leq \alpha_i, i = 1, \ldots, l. \tag{5}$$

The approach of adding $b^2/2$ and then obtaining a bounded dual was proposed in [4] and [14]. This is equivalent to adding a constant feature to the training data and finding a separating hyperplane passing through the origin. Numerical experiments in [7] 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:

$$w = \sum_{i=1}^{l} y_i \alpha_i \phi(x_i). \tag{6}$$

If we substitute w into (3), using

$$\begin{aligned} y_i w^T \phi(x_i) &= \sum_{j=1}^{l} y_i y_j \alpha_j \phi(x_j)^T \phi(x_i) \\ &= \sum_{j=1}^{l} Q_{ij} \alpha_j = (Q\alpha)_i, \text{ and} \end{aligned} \tag{7}$$

$$\begin{aligned} w^T w &= \sum_{i=1}^{l} y_i \alpha_i \phi(x_i)^T w \\ &= \sum_{i=1}^{l} \alpha_i (Q\alpha)_i = \alpha^T Q \alpha, \end{aligned} \tag{8}$$

we obtain a new optimization problem:

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

$$\text{subject to} \quad Q\alpha + by \geq e - \xi. \tag{9}$$

Though (9) is different from (5), the dual problem, we can show that for any optimal α of (9), the corresponding w defined by (6) is also an optimal solution of (3), so we can solve (9) instead of (5). We put the details in Appendix A. The reformulation of (3) to (9) was first used in [19].

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:

$$w = \sum_{i \in R} y_i \alpha_i \phi(x_i), \tag{10}$$

where R contains indices of this subset. By substituting (10) into (4), we can get a similar problem as (9), with the number of major variables (i.e. α) reduced to m:

$$\begin{aligned} \min_{\bar{\alpha}, b, \xi} \quad & \frac{1}{2}(\bar{\alpha}^T Q_{RR} \bar{\alpha} + b^2) + C(\sum_{i=1}^{l} \xi_i^2) \\ \text{subject to} \quad & Q_{:,R}\bar{\alpha} + by \geq e - \xi, \end{aligned} \tag{11}$$

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 to R. Thus, there are still l constraints. Following the generalized SVM by Mangasarian [13], [10] 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

$$\begin{aligned} \min_{\bar{\alpha}, b, \xi} \quad & \frac{1}{2}(\bar{\alpha}^T \bar{\alpha} + b^2) + C(\sum_{i=1}^{l} \xi_i^2) \\ \text{subject to} \quad & Q_{:,R}\bar{\alpha} + by \geq e - \xi. \end{aligned} \tag{12}$$

Later on we will address more about this simplification. Note that throughout the discussion we omit writing the constraint $\xi_i \geq 0$ as it is always satisfied at the optimal solution.

The idea of using (10) is similar to the Radial Basis Function (RBF) networks [17] which either select a subset of training data or generate some "centers" in order to construct a decision function. The RBF networks directly start from a form similar to (10) and the regularization term used is $\bar{\alpha}^T\bar{\alpha}/2$. If the inequality (12) is replaced by an equality, (12) is in a form of the RBF networks. There are already comparisons between SVM and RBF networks. For example, [22] shows that SVM performs better as the RBF networks use less information. Therefore, we are curious whether similar scenarios apply to SVM and RSVM, which will be one of the main aims of this paper.

III. Different Implementations for RSVM

In [10], the authors used Smooth SVM (SSVM) to solve RSVM, which basically approximates the original problem by an unconstrained one. However, we observe that (12) 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) and (12):

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 the 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 take this property. However, for nonlinear SVM where the dimensionality of the feature space is large, such methods may not be applicable. Thus, usually the dual problem is considered instead. Now we choose $m \ll l$ so we are safe to apply methods which were originally mainly suitable for linear SVM.

In Section III-A we will discuss the SSVM originally used in [10] while in Sections III-B- III-D, we consider three methods which were suitable for linear SVM: Least Square SVM (LS-SVM) [23], Lagrangian SVM (LSVM) [15], 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:

$$\begin{aligned} \min_{\widetilde{\alpha}, \xi} \quad & \frac{1}{2}\widetilde{\alpha}^T \widetilde{\alpha} + C(\sum_{i=1}^{l} \xi_i^2) \\ \text{subject to} \quad & \widetilde{Q}\widetilde{\alpha} \geq e - \xi. \end{aligned} \tag{13}$$

A. Using SSVM

Define $(\cdot)_+ \equiv \max(\cdot, 0)$, and apply the property that whenever $\xi_i > 0$, the ith constraint (13) must be active, the constrained problem (13) can be transformed to the unconstrained one:

$$\min_{\widetilde{\alpha}} \quad \frac{1}{2}\widetilde{\alpha}^T \widetilde{\alpha} + C(\sum_{i=1}^{l} ((e - \widetilde{Q}\widetilde{\alpha})_i)_+^2) \tag{14}$$

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 $(e - \widetilde{Q}\widetilde{\alpha})_+$ 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 (14) may cause numerical difficulties. For example, if a method for solving (14) 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:

$$\min_{\widetilde{\alpha}} f(\widetilde{\alpha}) = \frac{1}{2C}\widetilde{\alpha}^T\widetilde{\alpha} + \sum_{i=1}^{l} P_\beta(e - \widetilde{Q}\widetilde{\alpha})_i^2 \tag{15}$$

We solve (15) by a Newton's method implemented in the software TRON [12], which uses both the information of the first derivative (gradient) and the second derivative (Hessian).

While [10] 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

$$\nabla f(\widetilde{\alpha}) = -2\widetilde{Q}^T(\text{diag}(P_\beta(e - \widetilde{Q}\widetilde{\alpha}))\text{diag}(e+v)^{-1})e + \frac{1}{C}\widetilde{\alpha} \text{ and} \tag{16}$$

$$\nabla^2 f(\widetilde{\alpha}) = 2\widetilde{Q}^T\text{diag}(e+v)^{-2}(I + \beta\text{diag}(v)\text{diag}(P_\beta(e - \widetilde{Q}\widetilde{\alpha})))\widetilde{Q} + \frac{I}{C}, \tag{17}$$

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

TRON was originally designed for large sparse problems. Now the Hessian is dense so we use the modification in [7] to solve (15). 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 (17), we need more operations: $O(lm^2)$. Hence $O(lm^2)$ is the complexity of each iteration.

B. Using Least-Square SVM

Least-square SVM (LS-SVM), first introduced in [23], can solve linear SVM efficiently. It changes the inequality constraints to equalities, and solve the resulting linear system. Taking (3) as an example, they change the primal inequalities (4) 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:

$$\min_{w,b} \qquad \frac{1}{2}(w^Tw + b^2) + C(\sum_{i=1}^{l}(1 - y_i(w^T\phi(x_i) + b))^2) \tag{18}$$

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 (18) 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 (18) and solve a linear system with variables α and b. Thus, when l is large, the same problem of conventional SVM formulations occurs: the $l \times l$

matrix cannot be placed in the main 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 (13) to an equality

$$\widetilde{Q}\widetilde{\alpha} = e - \xi, \tag{19}$$

and substitute ξ into (13). 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:

$$\min_{\widetilde{\alpha}} \quad f(\widetilde{\alpha}) = \quad \frac{1}{2C}\widetilde{\alpha}^T\widetilde{\alpha} + \sum_{i=1}^{l}(e - \widetilde{Q}\widetilde{\alpha})_i^2 \tag{20}$$

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 $\frac{\partial f}{\partial \widetilde{\alpha}_i} = 0, i = 1, \ldots, m$:

$$\begin{aligned} \frac{1}{C}\widetilde{\alpha} + 2\widetilde{Q}^T\widetilde{Q}\widetilde{\alpha} - 2\widetilde{Q}^Te = 0, \\ (\widetilde{Q}^T\widetilde{Q} + \frac{I}{2C})\widetilde{\alpha} = \widetilde{Q}^Te, \end{aligned} \tag{21}$$

a positive definite linear system of size m. As $m \ll l$, (21) 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 $O(lm^2)$.

C. Using Lagrangian SVM

Here we discuss Lagrangian SVM (LSVM) [15], another technique to solve SVM problems. LSVM is an iterative algorithm which solves the dual form (5). Define $H \equiv Q + \frac{I}{2C} + yy^T$, then the Karush-Kuhn-Tucker (KKT) optimality conditions of (5) are

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

Again we denote max(·, 0) by $(\cdot)_+$. The authors of [15] used the following identity between any two vector a and b:

$$a \geq 0, b \geq 0, a^Tb = 0 \Leftrightarrow a = (a - \beta b)_+, \forall \beta > 0, \tag{22}$$

so the optimality conditions are equivalent to

$$H\alpha - e = (H\alpha - e - \beta\alpha)_+, \forall \beta > 0. \tag{23}$$

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

$$\alpha^{k+1} = H^{-1}(e + (H\alpha^k - e - \beta\alpha^k)_+). \tag{24}$$

[15] 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 [5], 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 (13):

$$\begin{aligned} \min_{\hat{\alpha}} \quad & \frac{1}{2}\hat{\alpha}^T(\widetilde{Q}\widetilde{Q}^T + \frac{I}{2C})\hat{\alpha} - e^T\hat{\alpha} \\ \text{subject to} \quad & \hat{\alpha}_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{25}$$

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 (6):

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

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.

D. Using Decomposition Methods for Linear SVM

Originally the decomposition method [18], [8], [20] was proposed to handle the nonlinear SVM whose kernel matrix is 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 in 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,

$$Q\Delta\alpha = X(X^T\Delta\alpha) \tag{26}$$

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 (26) can be q times faster than the regular decomposition method. This trick has been implemented in, for example, SVM^{light} [8] and BSVM (version 2.03 and after) [7]. Of course the selection of q becomes an issue. It should be larger than the one 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 (26).

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 (25), the dual problem is

$$\begin{aligned} \min_{\hat{\alpha}} \quad & \frac{1}{2}\hat{\alpha}^T\widetilde{Q}\widetilde{Q}^T\hat{\alpha} - e^T\hat{\alpha} \\ \text{subject to} \quad & 0 \le \hat{\alpha}_i \le C, i = 1, \ldots, l. \end{aligned} \tag{27}$$

The reason is that we will use the software BSVM which solves (27) but not (25).

In the following we summarize the time complexity of the four approaches discussed earlier:

Method	Time complexity
SSVM	$\#iterations \times O(lm^2)$
LS-SVM (w/ direct method)	$O(lm^2)$
LSVM	$\#iterations \times O(lm) + O(lm^2)$
Decomposition	$\#iterations \times O(lm)$

IV. Implementation Issues

A. 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 section 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:

$$\begin{aligned} \min_{\alpha} \quad & f(\alpha) = \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{subject to} \quad & y^T\alpha = 0, \\ & 0 \le \alpha_i \le C, i = 1, \ldots, l. \end{aligned} \tag{28}$$

It is shown in [3] that if $C > 0$, the KKT condition of (28) is equivalent to

$$\begin{aligned} m(\alpha) &= \max(\max_{\alpha_i<C, y_i=1} -\nabla f(\alpha)_i, \max_{\alpha_i>0, y_i=-1} \nabla f(\alpha)_i) \\ &\le \min(\min_{\alpha_i<C, y_i=-1} \nabla f(\alpha)_i, \min_{\alpha_i>0, y_i=1} -\nabla f(\alpha)_i) = M(\alpha). \end{aligned} \tag{29}$$

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

$$m(\alpha) \le M(\alpha) + \epsilon, \tag{30}$$

where we choose $\epsilon = 0.001$.

To check the performance of using linear and quadratic cost functions, we also modify LIBSVM to solve (2). 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 III-D. It also has a similar stopping criterion.

The LSVM method, another implementation for RSVM, solves problems in the dual form (25), hence a stopping criteria can be similar: Now

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

The KKT condition of (25) shows that if

$$\min \nabla f(\hat{\alpha})_i \ge \max_{\hat{\alpha}_i>0} \nabla f(\hat{\alpha})_i \text{ and } \hat{\alpha}_i \ge 0, \forall i,$$

then $\hat{\alpha}$ is an optimal solution. Note that during LSVM iterations, the $\hat{\alpha}_i \ge 0$ constraints are not maintained. Thus, we consider a stopping criterion as follows:

$$\max_{\hat{\alpha}_i>\epsilon/100} \nabla f(\hat{\alpha}) \le \min \nabla f(\hat{\alpha}) + \epsilon \text{ and } \hat{\alpha}_i \ge -\epsilon/100, \forall i. \tag{31}$$

We also set $\epsilon = 0.001$ here.

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

$$\|\nabla f(\widetilde{\alpha}^k)\|_2 \leq \epsilon \|\nabla f(\widetilde{\alpha}^1)\|_2, \tag{32}$$

where $f(\widetilde{\alpha})$ is defined in (15), $\widetilde{\alpha}^1$ is the initial solution, and $\epsilon = 10^{-5}$.

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

B. Multi-class RSVM

Although SVM was originally designed for binary classification, several methods have been proposed to solve multi-class classification. A common way is by considering 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 [6] we compare different decomposition implementations, and in [24] some comparisons of methods for multi-class LS-SVM are given. They both indicate that the one-against-one method performs well in practice. 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.

The implementation of all methods mentioned above is available upon request.

V. EXPERIMENTS

In this section we conduct experiments on some commonly used problems. We choose large multi-class datasets from the Statlog collection: dna, satimage, letter, and shuttle [16]. We also consider mnist [9], an important benchmark for handwritten digit recognition. The problem ijcnn1 is from the first problem of IJCNN challenge 2001 [21]. Note that we use the winner's transformation of raw data [2]. Problem protein, is a data set for protein secondary structure prediction [26]. Finally, the problem adult, from the UCI "adult" data set [1] and compiled by Platt [20], is also included. For the adult dataset, there are several realizations. Here, we only consider the realization with the smallest training set; the full dataset with training data (including duplicated ones) removed is taken as the test set. We choose this dataset because it has been tested in the RSVM paper [10]. Our preparation of the training and test sets are also similar to theirs. Except problems dna, ijcnn1, protein, and adult whose data values are either binary or 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 V.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 V.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
adult	1605	32561	2	123

We compare four implementations of RSVM discussed in Section III with two implementations of the regular SVM: linear and quadratic cost functions ((28) and (2)). For regular SVM with the linear cost function, we use the software LIBSVM which implements a simple decomposition method. We can easily modify LIBSVM to solve the formulation with a quadratic cost function, which we will refer to as LIBSVM-q in the rest of this paper. However, we will not use it for solving RSVM as LIBSVM implements an SMO type algorithm where the size of the working set is restricted to two. In Section III-D we have shown that larger working sets should be used when applying decomposition methods to 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 is some basic matrix operations so we use ATLAS to optimize the performance [27]. 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, they solve smaller problems by considering only free variables. The details on the shrinking technique are in [3, 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 [10]. 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 [10] deals with large problems. Once the size is determined, for all four implementations, we select the same subset for each binary RSVM problem.

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 [15]. 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 performance 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 V.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 that achieve the same highest testing rate, we report the one with the minimal training time.

VI. Results

A. Accuracy

Table V.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

TABLE V.2

A COMPARISON ON 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}	96.767	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
adult	2^4,2^{-6}	83.869	2^{-2}, 2^{-4}	83.974	2^{11}, 2^{-8}	83.872	2^{11}, 2^{-7}	83.788	2^{11}, 2^{-8}	83.842	2^{12},2^{-9}	83.899

TABLE V.3

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			
adult	642	1137	160			

not contradict the current understanding in this 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 (12) 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 [22] since RBF networks select only several centers, it may not extract enough information.

TABLE V.4

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
adult	1.89	15.2	2.67	39.89	8.43	1.37	27.91	16.67	3.73

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.

B. Number of Support Vectors

In Table V.3 we report the number of "unique" support vectors for each method. 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 three RSVM implementations have the same number of support vectors.

For letter, shuttle 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 [6] 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's. 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 in the next subsection.

C. Training and Testing Time

We report the training time and testing time in Table V.4. For the four RSVM implementations, we find their testing time is quite close so we post only that of the LSVM 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 depends 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 advantages on solving large binary problems because for multi-class data sets we can afford to use decomposition methods to solve several smaller binary problems.

Table V.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: 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, the 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.

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.

VII. Modifications on the RSVM Formulation

Remember we mentioned in Section II that following the generalized SVM the authors of [10] replace $\frac{1}{2}\bar{\alpha}^T Q_{RR}\bar{\alpha}$ in (11) by $\frac{1}{2}\bar{\alpha}^T\bar{\alpha}$ in (12). 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 sacrificed or not.

Without changing the $\frac{1}{2}\bar{\alpha}^T Q_{RR}\bar{\alpha}$ term in (20), LS-SVM formulation is

$$\min_{\widetilde{\alpha}} \quad f(\widetilde{\alpha}) = \quad \frac{1}{2C}\widetilde{\alpha}^T Q_{RR}\widetilde{\alpha} + \sum_{i=1}^{l}(e - \widetilde{Q}\widetilde{\alpha})_i^2 \tag{33}$$

Thus we will solve a different linear system:

$$(\widetilde{Q}^T\widetilde{Q} + \frac{1}{2C}Q_{RR})\widetilde{\alpha} = \widetilde{Q}^T e. \tag{34}$$

Though (34) is nearly the same as (21), it is only a positive semi-definite but not positive definite system. Thus, LU factorization instead of Cholesky factorization is used. This change affects very little to the training time according to the time complexity analysis in Section III-B. A comparison on the testing accuracy between solving (21) and (34) is in Table VII.1. 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.

TABLE VII.1

Testing accuracy: a comparison on solving (21) and (34)

	LS-SVM (21)		LS-SVM (34)	
Problem	(C,γ)	rate	(C,γ)	rate
dna	$(2^4, 2^{-6})$	92.327	$(2^4, 2^{-10})$	93.086
satimage	$(2^{12}, 2^{-3})$	89.9	$(2^8, 2^{-3})$	89.7
letter	$(2^{12}, 2^{-2})$	95.14	$(2^{12}, 2^{-3})$	94.62
shuttle	$(2^{12}, 2^4)$	99.58	$(2^7, 2^4)$	99.7
mnist	$(2^9, 2^{-6})$	96.48	$(2^7, 2^{-6})$	96.484
ijcnn1	$(2^{-2}, 2^{-2})$	91.676	$(2^{-2}, 2^{-6})$	90.962

VIII. Discussions and Conclusions

We have discussed four multi-class implementations for the RSVM formulations and have 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 to the primal form of SVM, restricting support vectors from a randomly selected subset still downgrades the performance. 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 the 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.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant NSC 90-2213-E-002-111. The authors thank M. Heiler, Y.-J. Lee, and X.-F. Lu for many helpful comments.

References

[1] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. Technical report, University of California, Department of Information and Computer Science, Irvine, CA, 1998. Available at `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

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

[3] 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`.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] Y. LeCun. MNIST database of handwritten digits. Available at `http://yann.lecun.com/exdb/mnist/`.

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

[11] Y.-J. Lee and O. L. Mangasarian. SSVM: A smooth support vector machine for classification. *Computational Optimization and Applications*, 20(1):5–22, 2001.

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

[13] 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.

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

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

[16] 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 `http://www.ncc.up.pt/liacc/ML/statlog/datasets.html`.

[17] 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`.

[18] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of CVPR'97*, pages 130–136, New York, NY, 1997. IEEE.

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

[20] 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.

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

[22] 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.

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

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

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

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

[27] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automatically tuned linear algebra software and the ATLAS project. Technical report, Department of Computer Sciences, University of Tennessee, 2000.

APPENDIX

The derivation from (7) and (8) to (9) shows only that if (w^*, b^*, ξ^*) and α^* are primal and dual optimal solutions, respectively, then (α^*, b^*, ξ^*) is feasible for (9). Hence, before using (9) in practice we want to make sure that any optimal (α, ξ, b) of (9) can construct an optimal w for the primal problem (3).

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). Since we have

shown that (α^*, b^*, ξ^*) is feasible for (9),

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

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

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

$$v \equiv \exp(-\beta(e - \widetilde{Q}\widetilde{\alpha})), \tag{35}$$

and then

$$P_\beta(e - \widetilde{Q}\widetilde{\alpha}) = (e - \widetilde{Q}\widetilde{\alpha}) + \beta^{-1}\log(1 + v). \tag{36}$$

The differentiations of v and P_β are:

$$\begin{aligned} \frac{\partial v_k}{\partial \widetilde{\alpha}_i} &= \frac{\partial}{\partial \widetilde{\alpha}_i}\exp(-\beta(e - \widetilde{Q}\widetilde{\alpha})_k) \\ &= -\beta(-\widetilde{Q}_{ki})\exp(-\beta(e - \widetilde{Q}\widetilde{\alpha})_k) \\ &= \beta\widetilde{Q}_{ki}v_k. \end{aligned}$$

$$\begin{aligned} \frac{\partial P_\beta(e - \widetilde{Q}\widetilde{\alpha})_k}{\partial \widetilde{\alpha}_i} &= \frac{\partial}{\partial \widetilde{\alpha}_i}(e - \widetilde{Q}\widetilde{\alpha} + \beta^{-1}\log(1 + v))_k \\ &= -\widetilde{Q}_{ki} + \beta^{-1}(1 + v_k)^{-1}\frac{\partial v_k}{\partial \widetilde{\alpha}_i} \\ &= -\widetilde{Q}_{ki} + \beta^{-1}(1 + v_k)^{-1}\beta\widetilde{Q}_{ki}v_k \\ &= -\widetilde{Q}_{ki} + \widetilde{Q}_{ki}(1 + v_k)^{-1}v_k \\ &= -\widetilde{Q}_{ki}(1 + v_k)^{-1}. \end{aligned}$$

Finally we obtain the gradient and Hessian of f:

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

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

where δ_{ij} is defined as

$$
\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}
$$

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

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

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