LIBSVM: a Library for Support Vector Machines (Version 2.31)

Chih-Chung Chang and Chih-Jen Lin*

September 7, 2001

Abstract

LIBSVM is a library for support vector machines (SVM). Its goal is to help users can easily use SVM as a tool. In this document, we present all its implementation details.

1 Introduction

LIBSVM is a library for support vector classification (SVM) and regression. Its goal is to let users can easily use SVM as a tool. In this document, we present all its implementation details.

In Section 2, we show formulations used in LIBSVM: C-support vector classification (C-SVC), ν-support vector classification (ν-SVC), distribution estimation (one-class SVM), ϵ-support vector regression (ϵ-SVR), and ν-support vector regression (ν-SVR). We discuss the implementation of solving quadratic problems in Section 3. Section 4 describes two implementation techniques: shrinking and caching. Then in Section 5 we discuss the implementation of multi-class classification. We now also support different penalty parameters for unbalanced data. Details are in Section 6

2 Formulations

2.1 C-Support Vector Classification (Binary Case)

Given training vectors $x_i \in R^n, i = 1, \ldots, l$, in two classes, and a vector $y \in R^l$ such that $y_i \in \{1, -1\}$, C-SVC (Cortes & Vapnik, 1995; Vapnik, 1998) solves the following primal problem:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \\ & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{2.1}$$

*Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan (cjlin@csie.ntu.edu.tw).

Its dual is

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ & 0 \le \alpha_i \le C, \qquad i = 1, \dots, l, \\ & y^T\alpha = 0, \end{aligned} \tag{2.2}$$

where e is the vector of all ones, C is the upper bound, Q is an l by l positive semidefinite 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. Here training vectors x_i are mapped into a higher (maybe infinite) dimensional space by the function ϕ.

The decision function is

$$f(x) = sign(\sum_{i=1}^{l} y_i\alpha_i K(x_i, x) + b).$$

2.2 ν-Support Vector Classification (Binary Case)

The ν-support vector classification (Schölkopf et al., 2000) uses a new parameter ν which let one control the number of support vectors and errors. The parameter $\nu \in (0, 1]$ is an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.

Details of the algorithm implemented in LIBSVM can be found in (Chang & Lin, 2001a). Given training vectors $x_i \in R^n, i = 1, \dots, l$, in two classes, and a vector $y \in R^l$ such that $y_i \in \{1, -1\}$, the primal form considered is:

$$\begin{aligned} \min_{w,b,\xi,\rho} \quad & \frac{1}{2}w^T w - \nu\rho + \frac{1}{l}\sum_{i=1}^{l}\xi_i \\ & y_i(w^T\phi(x_i) + b) \ge \rho - \xi_i, \\ & \xi_i \ge 0, i = 1, \dots, l, \rho \ge 0. \end{aligned}$$

The dual is:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha \\ & 0 \le \alpha_i \le 1/l, \qquad i = 1, \dots, l, \\ & e^T\alpha \ge \nu, \\ & y^T\alpha = 0. \end{aligned} \tag{2.3}$$

where $Q_{ij} \equiv y_i y_j K(x_i, x_j)$.

The decision function is:

$$f(x) = sign(\sum_{i=1}^{l} y_i\alpha_i(K(x_i, x) + b)).$$

In (Crisp & Burges, 1999; Chang & Lin, 2001a), it has been shown that $e^T\alpha \geq \nu$ can be replaced by $e^T\alpha = \nu$. With this property, in LIBSVM, we solve a scaled version of (2.3):

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha \\ & 0 \leq \alpha_i \leq 1, \qquad i = 1, \ldots, l, \\ & e^T\alpha = \nu l, \\ & y^T\alpha = 0. \end{aligned}$$

We output α/ρ so the computed decision function is:

$$f(x) = sign(\sum_{i=1}^{l} y_i(\alpha_i/\rho)(K(x_i, x) + b))$$

and then two margins are

$$y_i(w^T\phi(x_i) + b) = \pm 1$$

which are the same as those of C-SVC.

2.3 Distribution Estimation (One-class SVM)

One-class SVM was proposed by Schölkopf et al. (1999) for estimating the support of a high-dimensional distribution. Given training vectors $x_i \in R^n, i = 1, \ldots, l$ without any class information, the primal form in (Schölkopf et al., 1999) is:

$$\begin{aligned} \min_{w,b,\xi,\rho} \quad & \frac{1}{2}w^T w - \rho + \frac{1}{\nu l}\sum_{i=1}^{l}\xi_i \\ & w^T\phi(x_i) \geq \rho - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{aligned}$$

The dual is:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha \\ & 0 \leq \alpha_i \leq 1/(\nu l), i = 1, \ldots, l, \\ & e^T\alpha = 1, \end{aligned} \tag{2.4}$$

where $Q_{ij} = K(x_i, x_j) \equiv \phi(x_i)^T\phi(x_j)$.

In LIBSVM we solve a scaled version of (2.4):

$$\begin{aligned} \min \quad & \frac{1}{2}\alpha^T Q\alpha \\ & 0 \leq \alpha_i \leq 1, \qquad i = 1, \ldots, l, \\ & e^T\alpha = \nu l. \end{aligned}$$

The decision function is

$$f(x) = sign(\sum_{i=1}^{l} \alpha_i K(x_i, x) - \rho).$$

2.4 ϵ-Support Vector Regression (ϵ-SVR)

Given a set of data points, $\{(x_1, z_1), \ldots, (x_l, z_l)\}$, such that $x_i \in R^n$ is an input and $z_i \in R^1$ is a target output, the standard form of support vector regression (Vapnik, 1998) is:

$$\begin{aligned} \min_{w,b,\xi,\xi^*} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i + C \sum_{i=1}^{l} \xi_i^* \\ & z_i - w^T \phi(x_i) - b \le \epsilon + \xi_i, \\ & w^T \phi(x_i) + b - z_i \le \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \ge 0, i = 1, \ldots, l. \end{aligned}$$

The dual is:

$$\begin{aligned} \min_{\alpha,\alpha^*} \quad & \frac{1}{2}(\alpha - \alpha^*)^T Q(\alpha - \alpha^*) + \epsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} z_i(\alpha_i - \alpha_i^*) \\ & \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) = 0, 0 \le \alpha_i, \alpha_i^* \le C, i = 1, \ldots, l, \end{aligned} \tag{2.5}$$

where $Q_{ij} = K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$.

The approximate function is:

$$f(x) = \sum_{i=1}^{l} (-\alpha_i + \alpha_i^*) K(x_i, x) + b.$$

2.5 ν-Support Vector Regression (ν-SVR)

Similar to ν-SVC, for regression, (Schölkopf et al., 2000) use a parameter ν to control the number of support vectors. However, unlike ν-SVC where C is replaced by ν here ν replaces the parameter ϵ of ϵ-SVR. The primal form is

$$\begin{aligned} \min_{w,b,\xi,\xi^*,\epsilon} \quad & \frac{1}{2} w^T w + C(\nu\epsilon + \frac{1}{l} \sum_{i=1}^{l} (\xi_i + \xi_i^*)) \\ & (w^T \phi(x_i) + b) - z_i \le \epsilon + \xi_i, \\ & z_i - (w^T \phi(x_i) + b) \le \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \ge 0, i = 1, \ldots, l, \ \epsilon \ge 0. \end{aligned} \tag{2.6}$$

and the dual is

$$\begin{aligned}\min_{\alpha,\alpha^*} \quad & \frac{1}{2}(\alpha-\alpha^*)^T Q(\alpha-\alpha^*) + z^T(\alpha-\alpha^*) \\ & e^T(\alpha-\alpha^*) = 0,\ e^T(\alpha+\alpha^*) \le C\nu, \\ & 0 \le \alpha_i, \alpha_i^* \le C/l, \qquad i = 1,\dots,l, \end{aligned} \tag{2.7}$$

Similarly the inequality $e^T(\alpha+\alpha^*) \le C\nu$ can be replaced by an equality. In LIBSVM, we consider $C \leftarrow C/l$ so the dual problem solved is:

$$\begin{aligned}\min_{\alpha,\alpha^*} \quad & \frac{1}{2}(\alpha-\alpha^*)^T Q(\alpha-\alpha^*) + z^T(\alpha-\alpha^*) \\ & e^T(\alpha-\alpha^*) = 0,\ e^T(\alpha+\alpha^*) = Cl\nu, \\ & 0 \le \alpha_i, \alpha_i^* \le C, \qquad i = 1,\dots,l. \end{aligned} \tag{2.8}$$

Then the decision function is

$$f(x) = \sum_{i=1}^{l}(-\alpha_i + \alpha_i^*)K(x_i, x) + b,$$

the same as that of ϵ-SVR.

3 Solving the Quadratic Problems

3.1 The Decomposition Method for C-SVC, ϵ-SVR, and One-class SVM

We consider the following general form of C-SVC, ϵ-SVR, and one-class SVM:

$$\begin{aligned}\min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + p^T\alpha \\ & y^T\alpha = \Delta, \\ & 0 \le \alpha_t \le C, t = 1,\dots,l, \end{aligned} \tag{3.1}$$

where $y_t = \pm 1, t = 1,\dots,l$. It can be clearly seen that C-SVC and one-class SVM are already in the form of (3.1). For ϵ-SVR, we consider the following reformulation of (2.5):

$$\begin{aligned}\min_{\alpha,\alpha^*} \quad & \frac{1}{2}\begin{bmatrix}\alpha^T, (\alpha^*)^T\end{bmatrix}\begin{bmatrix}Q & -Q \\ -Q & Q\end{bmatrix}\begin{bmatrix}\alpha \\ \alpha^*\end{bmatrix} + \begin{bmatrix}\epsilon e^T + z^T, \epsilon e^T - z^T\end{bmatrix}\begin{bmatrix}\alpha \\ \alpha^*\end{bmatrix} \\ & y^T\begin{bmatrix}\alpha \\ \alpha^*\end{bmatrix} = 0, 0 \le \alpha_t, \alpha_t^* \le C, t = 1,\dots,l, \end{aligned} \tag{3.2}$$

where y is a $2l$ by 1 vector with $y_t = 1, t = 1,\dots,l$ and $y_t = -1, t = l+1,\dots,2l$.

The difficulty of solving (3.1) is the density of Q because Q_{ij} is in general not zero. In LIBSVM, we consider the decomposition method to conquer this difficulty. Some work on this method are, for example, Osuna et al. (1997b), Joachims (1998), Platt (1998), and Saunders et al. (1998).

Algorithm 3.1 (Decomposition method)

1. *Given a number $q \leq l$ as the size of the working set. Find α^1 as the initial solution. Set $k = 1$.*

2. *If α^k is an optimal solution of (2.2), stop. Otherwise, find a working set $B \subset \{1, \ldots, l\}$ whose size is q. Define $N \equiv \{1, \ldots, l\} \backslash B$ and α_B^k and α_N^k to be sub-vectors of α^k corresponding to B and N, respectively.*

3. *Solve the following sub-problem with the variable α_B:*

$$\begin{aligned} \min_{\alpha_B} \quad & \frac{1}{2}\alpha_B^T Q_{BB}\alpha_B + (p_B + Q_{BN}\alpha_N^k)^T\alpha_B \\ & 0 \leq (\alpha_B)_t \leq C, t = 1, \ldots, q, \\ & y_B^T\alpha_B = \Delta - y_N^T\alpha_N^k, \end{aligned} \tag{3.3}$$

   where $\begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix}$ is a permutation of the matrix Q.

4. *Set α_B^{k+1} to be the optimal solution of (3.3) and $\alpha_N^{k+1} \equiv \alpha_N^k$. Set $k \leftarrow k + 1$ and goto Step 2.*

The basic idea of the decomposition method is that in each iteration, the indices $\{1, \ldots, l\}$ of the training set are separated to two sets B and N, where B is the working set and $N = \{1, \ldots, l\} \backslash B$. The vector α_N is fixed so the objective value becomes $\frac{1}{2}\alpha_B^T Q_{BB}\alpha_B - (p_B - Q_{BN}\alpha_N)^T\alpha_B + \frac{1}{2}\alpha_N^T Q_{NN}\alpha_N - p_N^T\alpha_N$. Then a sub-problem with the variable α_B, i.e. (3.3), is solved. Note that B is updated in each iteration. To simplify the notation, we simply use B instead of B^k. The strict decrease of the objective function holds and the theoretical convergence will be discussed in Section 3.3.

3.2 Working Set Selection and Stopping Criteria for C-SVC, ϵ-SVR, and One-class SVM

An important issue of the decomposition method is the selection of the working set B. The Karush-Kuhn-Tucker (KKT) condition of (3.1) shows that there is a scalar b and two nonnegative vectors λ and μ such that

$$Q\alpha + p + by = \lambda - \mu$$
$$\lambda_i\alpha_i = 0, \mu_i(C - \alpha)_i = 0, \lambda_i \geq 0, \mu_i \geq 0, i = 1, \ldots, l.$$

Then the above can be rewritten as

$$\begin{aligned} Q\alpha + p + by \quad & \geq 0 \quad \text{if } \alpha = 0, \\ & = 0 \quad \text{if } 0 < \alpha < C, \\ & \leq 0 \quad \text{if } \alpha = C. \end{aligned}$$

By using $y_i = \pm 1, i = 1, \ldots, l$, this further implies that

$$\begin{array}{lcll} y_t = 1, \alpha_t < C & \Rightarrow & (Q\alpha + p)_t + b \geq 0 & \Rightarrow \quad b \geq -(Q\alpha + p)_t = -\nabla f(\alpha)_t, \\ y_t = -1, \alpha_t > 0 & \Rightarrow & (Q\alpha + p)_t - b \leq 0 & \Rightarrow \quad b \geq (Q\alpha + p)_t = \nabla f(\alpha)_t, \\ y_t = -1, \alpha_t < C & \Rightarrow & (Q\alpha + p)_t - b \geq 0 & \Rightarrow \quad b \leq (Q\alpha + p)_t = \nabla f(\alpha)_t, \\ y_t = 1, \alpha_t > 0 & \Rightarrow & (Q\alpha + p)_t + b \leq 0 & \Rightarrow \quad b \leq -(Q\alpha + p)_t = -\nabla f(\alpha)_t, \end{array} \tag{3.4}$$

where $f(\alpha) \equiv \frac{1}{2}\alpha^T Q\alpha + p^T\alpha$ and $\nabla f(\alpha)$ is the gradient of $f(\alpha)$ at α. We consider

$$i \equiv \text{argmax}(\{-\nabla f(\alpha)_t \mid y_t = 1, \alpha_t < C\}, \{\nabla f(\alpha)_t \mid y_t = -1, \alpha_t > 0\}), \tag{3.5}$$

$$j \equiv \text{argmin}(\{\nabla f(\alpha)_t \mid y_t = -1, \alpha_t < C\}, \{-\nabla f(\alpha)_t \mid y_t = 1, \alpha_t > 0\}). \tag{3.6}$$

We then use $B = \{i, j\}$ as the working set for the sub-problem (3.3) of the decomposition method. Here i and j are the two elements which violate the KKT condition the most. The idea of using only two elements for the working set are from the Sequential Minimal Optimization (SMO) by Platt (1998). The main advantage is that an analytic solution of (3.3) can be obtained so there is no need to use an optimization software. Note that (3.5) and (3.6) are a special case of the working set selection of the software SVM^{light} by Joachims (1998). To be more precise, in SVM^{light}, if α is the current solution, the following problem is solved:

$$\begin{aligned} \min_d \quad & \nabla f(\alpha)^T d \\ & y^T d = 0, \ -1 \leq d \leq 1, \qquad (3.7) \\ & d_t \geq 0, \text{ if } \alpha_t = 0, \ d_t \leq 0, \text{ if } \alpha_t = C, \\ & |\{d_t \mid d_t \neq 0\}| = q. \qquad (3.8) \end{aligned}$$

Note that $|\{d_t \mid d_t \neq 0\}|$ means the number of components of d which are not zero. The constraint (3.8) implies that a descent direction involving only q variables is obtained. Then components of α with non-zero d_t are included in the working set B which is used to construct the sub-problem (3.3). Note that d is only used for identifying B but not as a search direction.

It can be clearly seen that if $q = 2$, the solution of (3.7) is

$$\begin{aligned} i &= \text{argmin}\{\nabla f(\alpha)_t d_t \mid y_t d_t = 1; d_t \geq 0, \text{ if } \alpha_t = 0; d_t \leq 0, \text{ if } \alpha_t = C.\}, \\ j &= \text{argmin}\{\nabla f(\alpha)_t d_t \mid y_t d_t = -1; d_t \geq 0, \text{ if } \alpha_t = 0; d_t \leq 0, \text{ if } \alpha_t = C.\}, \end{aligned}$$

which is the same as (3.5) and (3.6). We also notice that this is also the modification 2 of the algorithms in (Keerthi et al., 2001).

We then define

$$g_i \equiv \begin{cases} -\nabla f(\alpha)_i & \text{if } y_i = 1, \alpha_i < C, \\ \nabla f(\alpha)_i & \text{if } y_i = -1, \alpha_i > 0, \end{cases} \tag{3.9}$$

and

$$g_j \equiv \begin{cases} -\nabla f(\alpha)_j & \text{if } y_j = -1, \alpha_j < C, \\ \nabla f(\alpha)_j & \text{if } y_j = 1, \alpha_j > 0. \end{cases} \tag{3.10}$$

From (3.4), we know

$$g_i \leq -g_j \tag{3.11}$$

implies that α is an optimal solution of (2.2). Practically the stopping criteria can be written and implemented as:

$$g_i \leq -g_j + \epsilon, \tag{3.12}$$

where ϵ is a small positive number.

3.3 Convergence of the Decomposition Method

The convergence of decomposition methods was first studied in Chang et al. (2000) but algorithms discussed there do not coincide with existing implementations. In this section we will discuss only convergence results related to the specific decomposition method in Section 3.2.

From Keerthi and Gilbert (2002) we have

Theorem 3.2 *Given any $\epsilon > 0$, after a finite number of iterations (3.12) will be satisfied.*

This theorem establishes the so-called "finite termination" property so we are sure that after finite steps the algorithm will stop.

For asymptotic convergence from Lin (2001a) we have

Theorem 3.3 *If $\{\alpha^k\}$ is the sequence generated by the decomposition method in Section 3.2, the limit of any its convergent subsequence is an optimal solution of (3.1).*

Note that Theorem 3.2 does not imply Theorem 3.3 as if we consider g_i and g_j in (3.12) as functions of α, they are not continuous. Hence we cannot take limit on both sides of (3.12) and claim that any convergent point has already satisfies the KKT condition.

Theorem 3.3 was first proved as a special case of general results in (Lin, 2001b) where some assumptions are needed. Now the proof in (Lin, 2001a) does not require any assumption.

3.4 The Decomposition Method for ν-SVC and ν-SVR

Both ν-SVC and ν-SVR can be considered as the following general form:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + p^T\alpha \\ & y^T\alpha = \Delta_1, \\ & e^T\alpha = \Delta_2, \\ & 0 \le \alpha_t \le C, t = 1, \dots, l. \end{aligned} \tag{3.13}$$

The decomposition method is the same as Algorithm 3.1 but the sub-problem is different:

$$\begin{aligned} \min_{\alpha_B} \quad & \frac{1}{2}\alpha_B^T Q_{BB}\alpha_B + (p_B + Q_{BN}\alpha_N^k)^T\alpha_B \\ & y_B^T\alpha_B = \Delta_1 - y_N^T\alpha_N^k, \\ & e_B^T\alpha_B = \Delta_2 - e_N^T\alpha_N^k, \\ & 0 \le (\alpha_B)_t \le C, t = 1, \dots, q. \end{aligned} \tag{3.14}$$

Now if only two elements i and j are selected but $y_i \ne y_j$, then $y_B^T\alpha_B = \Delta_1 - y_N^T\alpha_N^k$ and $e_B^T\alpha_B = \Delta_2 - e_N^T\alpha_N^k$ imply that there are two equations with two variables so (3.14) has only one feasible point. Therefore, from α^k, the solution cannot be moved any more. On the other hand, if $y_i = y_j$, $y_B^T\alpha_B = \Delta_1 - y_N^T\alpha_N^k$ and $e_B^T\alpha_B = \Delta_2 - e_N^T\alpha_N^k$ become the same equality so there are multiple feasible solutions. Therefore, we have to keep $y_i = y_j$ while selecting the working set.

The KKT condition of (3.13) shows

$$\begin{aligned} \nabla f(\alpha)_i - \rho + by_i &= 0 \text{ if } 0 < \alpha_i < C, \\ &\ge 0 \text{ if } \alpha_i = 0, \\ &\le 0 \text{ if } \alpha_i = C. \end{aligned}$$

Define

$$r_1 \equiv \rho - b,\ r_2 \equiv \rho + b.$$

If $y_i = 1$ the KKT condition becomes

$$\begin{aligned} \nabla f(\alpha)_i - r_1 &\ge 0 \text{ if } \alpha_i < C, \\ &\le 0 \text{ if } \alpha_i > 0. \end{aligned} \tag{3.15}$$

On the other hand, if $y_i = -1$, it is

$$\begin{aligned} \nabla f(\alpha)_i - r_2 &\ge 0 \text{ if } \alpha_i < C, \\ &\le 0 \text{ if } \alpha_i > 0. \end{aligned} \tag{3.16}$$

Hence, indices i and j are selected from either

$$\begin{aligned} i &= \text{argmin}_t\{\nabla f(\alpha)_t | y_t = 1, \alpha_t < C\}, \\ j &= \text{argmax}_t\{\nabla f(\alpha)_t | y_t = 1, \alpha_t > 0\}, \end{aligned} \tag{3.17}$$

or

$$\begin{aligned} i &= \text{argmin}_t\{\nabla f(\alpha)_t | y_t = -1, \alpha_t < C\}, \\ j &= \text{argmax}_t\{\nabla f(\alpha)_t | y_t = -1, \alpha_t > 0\}, \end{aligned} \tag{3.18}$$

depending on which one gives a smaller $\nabla f(\alpha)_i - \nabla f(\alpha)_j$ (i.e. larger KKT violations).

This was first proposed in (Keerthi & Gilbert, 2002). Some details for classification and regression can be seen in (Chang & Lin, 2001a, Section 4) and (Chang & Lin, 2001b), respectively.

The stopping criterion will be described in Section 3.6.

3.5 Analytical Solutions

Now (3.3) is a simple problem with only two variables:

$$\begin{aligned} \min_{\alpha_i, \alpha_j} \quad & \frac{1}{2} \begin{bmatrix} \alpha_i & \alpha_j \end{bmatrix} \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ji} & Q_{jj} \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + (Q_{i,N}\alpha_N - 1)\alpha_i + (Q_{j,N}\alpha_N - 1)\alpha_j \\ & y_i\alpha_i + y_j\alpha_j = \Delta_1 - y_N^T\alpha_N^k, \\ & 0 \le \alpha_i, \alpha_j \le C. \end{aligned} \tag{3.19}$$

Platt (1998) substituted $\alpha_i = y_i(\Delta_1 - y_N^T\alpha_N - y_j\alpha_j)$ into the objective function of (3.3) and solved an unconstrained minimization on α_j. The following solution is obtained:

$$\alpha_j^{new} = \begin{cases} \alpha_j + \frac{-G_i - G_j}{Q_{ii} + Q_{jj} + 2Q_{ij}} & \text{if } y_i \ne y_j, \\ \alpha_j + \frac{G_i - G_j}{Q_{ii} + Q_{jj} - 2Q_{ij}} & \text{if } y_i = y_j, \end{cases} \tag{3.20}$$

where

$$G_i \equiv \nabla f(\alpha)_i \text{ and } G_j \equiv \nabla f(\alpha)_j.$$

If this value is outside the possible region of α_j (that is, exceeds the feasible region of (3.3)), the value of (3.20) is clipped into the feasible region and is assigned as the new α_j. For example, if $y_i = y_j$ and $C \le \alpha_i + \alpha_j \le 2C$, α_j^{new} must satisfy

$$L \equiv \alpha_i + \alpha_j - C \le \alpha_j^{new} \le C \equiv H$$

as the largest value α_i^{new} and α_j^{new} can be is C. Hence if

$$\alpha_j + \frac{G_i - G_j}{Q_{ii} + Q_{jj} + 2Q_{ij}} \le L,$$

we define $\alpha_j^{new} \equiv L$ and then

$$\alpha_i^{new} = \alpha_i + \alpha_j - \alpha_j^{new} = C. \tag{3.21}$$

However, numerically the last equality of (3.21) may not hold. The floating-point operation will cause that

$$\begin{aligned} & \alpha_i + \alpha_j - \alpha_j^{new} \\ = \; & \alpha_i + \alpha_j - (\alpha_i + \alpha_j - C) \\ \neq \; & C. \end{aligned}$$

Therefore, in most SVM software, a small tolerance ϵ_a is specified and all $\alpha_i \geq C - \epsilon_a$ are considered to be at the upper bound and all $\alpha_i \leq \epsilon_a$ are considered to be zero. This is necessary as otherwise some data will be wrongly considered as support vectors. In addition, the calculation of the bias term b also need correct identification of those α_i which are free (i.e. $0 < \alpha_i < C$).

In (Hsu & Lin, 2002b), it has been pointed out that if all bounded α_i obtain their values using direct assignments, there is no need of using an ϵ_a. To be more precise, for floating-point computation, if $\alpha_i \leftarrow C$ is assigned somewhere, a future floating-point comparison between C and C returns true as they both have the same internal representation. More details about its implementation will be presented in Section 6. Note that though this involves a little more operations, as solving the analytic solution of (3.19) takes only a small portion of the total computational time, the difference is negligible.

Another minor problem is that the denominator in (3.20) is sometime zero. When this happens,

$$Q_{ij} = \pm(Q_{ii} + Q_{ij})/2$$

so

$$\begin{aligned} & Q_{ii}Q_{jj} - Q_{ij}^2 \\ = \; & Q_{ii}Q_{jj} - (Q_{ii} + Q_{jj})^2/4 \\ = \; & -(Q_{ii} - Q_{ij})^2/4 \leq 0. \end{aligned}$$

Therefore, we know if Q_{BB} is positive definite, the zero denominator in (3.20) never happens. Hence this problem happens only if Q_{BB} is a 2 by 2 singular matrix. We discuss some situations where Q_{BB} may be singular.

1. The function ϕ does not map data to independent vectors in a higher-dimensional space so Q is only positive semidefinite. For example, using the linear or low-degree polynomial kernels. Then it is possible that a singular Q_{BB} is picked.

2. Some kernels have a nice property that $\phi(x_i), i = 1, \ldots, l$ are independent if $x_i \neq x_j$. Thus Q as well as all possible Q_{BB} are positive definite. An example is the RBF kernel (Micchelli, 1986). However, for many practical data we have encountered, some of $x_i, i = 1, \ldots, l$ are the same. Therefore, several rows (columns) of Q are exactly the same so Q_{BB} may be singular.

However, even if the denominator of (3.20) is zero, there are no numerical problems: From (3.12), we note that

$$g_i + g_j \geq \epsilon$$

during the iterative process. Since

$$\begin{aligned} g_i + g_j &= \pm(-G_i - G_j) \text{ if } y_i \neq y_j, \text{ and} \\ g_i + g_j &= \pm(G_i - G_j) \text{ if } y_i = y_j, \end{aligned}$$

the situation of $0/0$ which is defined as NaN by IEEE standard does not appear. Therefore, (3.20) returns $\pm\infty$ if the denominator is zero which can be detected as special quantity of IEEE standard and clipped to regular floating point number.

3.6 The Calculation of b or ρ

After the solution α of the dual optimization problem is obtained, the variables b or ρ must be calculated as they are used in the decision function. Here we simply describe the case of ν-SVC and ν-SVR where b and ρ both appear. Other formulations are simplified cases of them.

The KKT condition of (3.13) has been shown in (3.15) and (3.16). Now we consider the case of $y_i = 1$. If there are α_i which satisfy $0 < \alpha_i < C$, then $r_1 = \nabla f(\alpha)_i$. Practically to avoid numerical errors, we average them:

$$r_1 = \frac{\sum_{0<\alpha_i<C, y_i=1} \nabla f(\alpha)_i}{\sum_{0<\alpha_i<C, y_i=1} 1}.$$

On the other hand, if there is no such α_i, as r_1 must satisfy

$$\max_{\alpha_i=C, y_i=1} \nabla f(\alpha)_i \leq r_1 \leq \min_{\alpha_i=0, y_i=1} \nabla f(\alpha)_i,$$

we take r_1 the midpoint of the range.

For $y_i = -1$, we can calculate r_2 in a similar way.

After r_1 and r_2 are obtained,

$$\rho = \frac{r_1 + r_2}{2} \text{ and } -b = \frac{r_1 - r_2}{2}.$$

Note that the KKT condition can be written as

$$\max_{\alpha_i>0, y_i=1} \nabla f(\alpha)_i \le \min_{\alpha_i<C, y_i=1} \nabla f(\alpha)_i$$

and

$$\max_{\alpha_i>0, y_i=-1} \nabla f(\alpha)_i \le \min_{\alpha_i<C, y_i=-1} \nabla f(\alpha)_i.$$

Hence practically we can use the following stopping criterion: The decomposition method stops if the iterate α satisfies the following condition:

$$\max(\quad - \min_{\alpha_i<C, y_i=1} \nabla f(\alpha)_i + \max_{\alpha_i>0, y_i=1} \nabla f(\alpha)_i, \tag{3.22}$$

$$- \min_{\alpha_i<C, y_i=-1} \nabla f(\alpha)_i + \max_{\alpha_i>0, y_i=-1} \nabla f(\alpha)_i) < \epsilon, \tag{3.23}$$

where $\epsilon > 0$ is a chosen stopping tolerance.

4 Shrinking and Caching

Since for many problems the number of free support vectors (i.e. $0 < \alpha_i < C$) is small, the shrinking technique reduces the size of the working problem without considering some bounded variables (Joachims, 1998). Near the end of the iterative process, the decomposition method identifies a possible set A where all final free α_i may reside in. Then instead of solving the whole problem (2.2), the decomposition method works on a smaller problem:

$$\begin{aligned} \min_{\alpha_A} \quad & \frac{1}{2}\alpha_A^T Q_{AA}\alpha_A - (p_A - Q_{AN}\alpha_N^k)^T \alpha_A \\ & 0 \le (\alpha_A)_t \le C, t = 1, \dots, q, \\ & y_A^T \alpha_A = \Delta - y_N^T \alpha_N^k, \end{aligned} \tag{4.1}$$

where $N = \{1, \dots, l\} \backslash A$.

Of course this heuristic may fail if the optimal solution of (4.1) is not the corresponding part of that of (2.2). When that happens, the whole problem (2.2) is reoptimized starting from a point α where α_B is an optimal solution of (4.1) and α_N are bounded variables identified before the shrinking process. Note that while solving the shrinked problem (4.1), we only know the gradient $Q_{AA}\alpha_A + Q_{AN}\alpha_N + p_A$ of (4.1). Hence when problem (2.2) is reoptimized we also have to reconstruct the whole gradient $\nabla f(\alpha)$, which is quite expensive.

Many implementations began the shrinking procedure near the end of the iterative process, in LIBSVM however, we start the shrinking process from the beginning. The procedure is as follows:

1. After every $\min(l, 1000)$ iterations, we try to shrink some variables. Note that during the iterative process

$$\begin{aligned} & \min(\{\nabla f(\alpha_k)_t \mid y_t = -1, \alpha_t < C\}, \{-\nabla f(\alpha_k)_t \mid y_t = 1, \alpha_t > 0\}) = -g_j \\ < \quad & g_i = \max(\{-\nabla f(\alpha_k)_t \mid y_t = 1, \alpha_t < C\}, \{\nabla f(\alpha_k)_t \mid y_t = -1, \alpha_t > 0\}), \end{aligned} \tag{4.2}$$

as (3.11) is not satisfied yet.

We conjecture that for those

$$g_t = \begin{cases} -\nabla f(\alpha)_t & \text{if } y_t = 1, \alpha_t < C, \\ \nabla f(\alpha)_t & \text{if } y_t = -1, \alpha_t > 0, \end{cases} \tag{4.3}$$

if

$$g_t \le -g_j, \tag{4.4}$$

and α_t resides at a bound, then the value of α_t may not change any more. Hence we inactivate this variable. Similarly, for those

$$g_t \equiv \begin{cases} -\nabla f(\alpha)_t & \text{if } y_t = -1, \alpha_t < C, \\ \nabla f(\alpha)_t & \text{if } y_t = 1, \alpha_t > 0, \end{cases} \tag{4.5}$$

if

$$-g_t \ge g_i, \tag{4.6}$$

and α_t is at a bound, it is inactivated. Thus the set A of activated variables is dynamically reduced in every $\min(l, 1000)$ iterations.

2. Of course the above shrinking strategy may be too aggressive. Since the decomposition method has a very slow convergence and a large portion of iterations are spent for achieving the final digit of the required accuracy, we would not like those iterations are wasted because of a wrongly shrinked problem (4.1). Hence when the decomposition method first achieves the tolerance

$$g_i \le -g_j + 10\epsilon,$$

where ϵ is the specified stopping criteria, we reconstruct the whole gradient. Then based on the correct information, we use criteria like (4.3) and (4.5) to inactivate some variables and the decomposition method continues.

Therefore, in LIBSVM, the size of the set A of (4.1) is dynamically reduced. To decrease the cost of reconstructing the gradient $\nabla f(\alpha)$, during the iterations we always keep

$$\bar{G}_i = \sum_{\alpha_j = C} Q_{ij}, i = 1, \ldots, l.$$

Then for the gradient $\nabla f(\alpha)_i, i \notin A$, we have

$$\nabla f(\alpha)_i = \sum_{j=1}^{l} Q_{ij}\alpha_j = C\bar{G}_i + \sum_{0<\alpha_j<C} Q_{ij}\alpha_j.$$

For ν-SVC and ν-SVR, as the stopping condition (3.23) is different from (3.12), the shrinking strategies (4.4) and (4.6) must be modified. From (4.2), now we have to separate two cases: $y_t = 1$ and $y_t = -1$. For $y_t = 1$, (4.2) becomes

$$\begin{aligned} & \min\{-\nabla f(\alpha)_t \mid y_t = 1, \alpha_t > 0\} = -g_j \\ < \quad & g_i = \max\{-\nabla f(\alpha)_t \mid y_t = 1, \alpha_t < C\} \end{aligned}$$

so we inactivate those α_t with

$$-\nabla f(\alpha)_t \leq -g_j \text{ if } y_t = 1 \text{ and } \alpha_t < C,$$

and

$$-\nabla f(\alpha)_t \geq g_i \text{ if } y_t = 1 \text{ and } \alpha_t > 0.$$

The case for $y_t = -1$ is similar.

Another technique for reducing the computational time is caching. Since Q is fully dense and may not be stored in the computer memory, elements Q_{ij} are calculated as needed. Then usually a special storage using the idea of a cache is used to store recently used Q_{ij} (Joachims, 1998). Hence the computational cost of later iterations can be reduced.

In LIBSVM, we implement a simple least-recent-use strategy for the cache. We dynamically cache only recently used columns of Q_{AA} of (4.1).

5 Multi-class classification

We use the "one-against-one" approach (Knerr et al., 1990) in which $k(k-1)/2$ classifiers are constructed and each one trains data from two different classes. The first use of this strategy on SVM was in (Friedman, 1996; Kreßel, 1999). For training data from the ith

and the jth classes, we solve the following binary classification problem:

$$\begin{aligned}\min_{w^{ij},b^{ij},\xi^{ij}} \quad & \frac{1}{2}(w^{ij})^T w^{ij} + C(\sum_t (\xi^{ij})_t) \\ & ((w^{ij})^T \phi(x_t)) + b^{ij}) \geq 1 - \xi_t^{ij}, \text{ if } x_t \text{ in the } i\text{th class}, \\ & ((w^{ij})^T \phi(x_t)) + b^{ij}) \leq -1 + \xi_t^{ij}, \text{ if } x_t \text{ in the } j\text{th class}, \\ & \xi_t^{ij} \geq 0.\end{aligned}$$

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. In case that two classes have indentical votes, thought it may not be a good strategy, now we simply select the one with the smaller index.

Another method for multi-class classification is the "one-against-all" approach in which k SVM models are constructed and the ith SVM is trained with all of the examples in the ith class with positive labels, and all other examples with negative labels. We did not consider it as some research work (e.g. (Weston & Watkins, 1998; Platt et al., 2000)) have shown that it does not perform as good as "one-against-one"

In addition, though we have to train as many as $k(k-1)/2$ classifiers, as each problem is smaller (only data from two classes), the total training time may not be more than the "one-against-all" method. This is one of the reasons why we choose the "one-against-one" method. Some detailed comparisons are in (Hsu & Lin, 2002a).

6 Unbalanced Data

For some classification problems, numbers of data in different classes are unbalanced. Hence some researchers (e.g. (Osuna et al., 1997a)) have proposed to use different penalty parameters in the SVM formulation: For example, C-SVM becomes

$$\begin{aligned}\min_{w,b,\xi} \quad & \frac{1}{2}w^T w + C_+ \sum_{y_i=1} \xi_i + C_- \sum_{y_i=-1} \xi_i \\ & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l.\end{aligned}$$

Its dual is

$$\begin{aligned}\min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha - e^T \alpha \\ & 0 \leq \alpha_i \leq C_+, \text{ if } y_i = 1 \qquad (6.1) \\ & 0 \leq \alpha_i \leq C_-, \text{ if } y_i = -1 \qquad (6.2) \\ & y^T \alpha = 0,\end{aligned}$$

Note that by replacing C with different $C_i, i = 1, \ldots, l$, most of the analysis earlier are still correct. Now using C_+ and C_- is just a special case of it. Therefore, the implementation is almost the same. A main difference is on the solution of the sub-problem (3.19). Now it becomes:

$$\begin{aligned} \min_{\alpha_i, \alpha_j} \quad & \frac{1}{2} \begin{bmatrix} \alpha_i & \alpha_j \end{bmatrix} \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ji} & Q_{jj} \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + (Q_{i,N}\alpha_N - 1)\alpha_i + (Q_{j,N}\alpha_N - 1)\alpha_j \\ & y_i\alpha_i + y_j\alpha_j = \Delta_1 - y_N^T\alpha_N^k, \\ & 0 \le \alpha_i \le C_i, 0 \le \alpha_j \le C_j, \end{aligned} \tag{6.3}$$

where C_i and C_j can be C_+ or C_- depending on y_i and y_j.

Following the concept of assigning all bounded values explicitly, if $y_i \neq y_j$, we use the following segment of code:

```
if(y[i]!=y[j])
{
        double delta = (-G[i]-G[j])/(Q_i[i]+Q_j[j]+2*Q_i[j]);
        double diff = alpha[i] - alpha[j];
        alpha[i] += delta;
        alpha[j] += delta;

        if(diff > 0)
        {
                if(alpha[j] < 0)
                {
                        alpha[j] = 0;
                        alpha[i] = diff;
                }
        }
        else
        {
                if(alpha[i] < 0)
                {
                        alpha[i] = 0;
                        alpha[j] = -diff;
                }
        }
        if(diff > C_i - C_j)
        {
                if(alpha[i] > C_i)
                {
                        alpha[i] = C_i;
                        alpha[j] = C_i - diff;
                }
        }
        else
        {
                if(alpha[j] > C_j)
                {
```

```
                        alpha[j] = C_j;
                        alpha[i] = C_j + diff;
                    }
                }
            }
```

In the above code basically we think the feasible region as a rectangle with length C_i and width C_j. Then we consider situations where the line segment $\alpha_i - \alpha_j = y_i(\Delta_1 - y_N^T \alpha_N^k)$ goes to the outside of the rectangular region.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grants NSC 89-2213-E-002-013 and NSC 89-2213-E-002-106. The authors thank Chih-Wei Hsu and Jen-Hao Lee for many helpful discussions and comments. We also thank Ryszard Czerminski and Lily Tian for some useful comments.

References

Chang, C.-C., Hsu, C.-W., & Lin, C.-J. (2000). The analysis of decomposition methods for support vector machines. *IEEE Trans. Neural Networks*, *11*(4), 1003-1008.

Chang, C.-C., & Lin, C.-J. (2001a). Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, *13*(9), 2119-2147.

Chang, C.-C., & Lin, C.-J. (2001b). *Training ν-support vector regression: Theory and algorithms* (Tech. Rep.). Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

Cortes, C., & Vapnik, V. (1995). Support-vector network. *Machine Learning*, *20*, 273-297.

Crisp, D. J., & Burges, C. J. C. (1999). A geometric interpretation of ν-SVM classifiers. In *NIPS99.*

Friedman, J. (1996). *Another approach to polychotomous classification* (Tech. Rep.). Department of Statistics, Stanford University. (Available at `http://www-stat.stanford.edu/reports/friedman/poly.ps.Z`)

Hsu, C.-W., & Lin, C.-J. (2002a). A comparison of methods for multi-class support vector machines. *IEEE Trans. Neural Networks.* (To appear)

Hsu, C.-W., & Lin, C.-J. (2002b). A simple decomposition method for support vector machines. *Machine Learning*, *46*, 291–314.

Joachims, T. (1998). Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods - support vector learning.* Cambridge, MA: MIT Press.

Keerthi, S. S., & Gilbert, E. G. (2002). Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, *46*, 351–360.

Keerthi, S. S., Shevade, S., Bhattacharyya, C., & Murthy, K. (2001). Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, *13*, 637–649.

Knerr, S., Personnaz, L., & Dreyfus, G. (1990). Single-layer learning revisited: a stepwise procedure for building and training a neural network. In J. Fogelman (Ed.), *Neurocomputing: Algorithms, architectures and applications.* Springer-Verlag.

Kreßel, U. (1999). Pairwise classification and support vector machines. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods — support vector learning* (pp. 255–268). Cambridge, MA: MIT Press.

Lin, C.-J. (2001a). *Asymptotic convergence of an smo algorithm without any assumptions* (Tech. Rep.). Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

Lin, C.-J. (2001b). On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks.* (To appear)

Micchelli, C. A. (1986). Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, *2*, 11-22.

Osuna, E., Freund, R., & Girosi, F. (1997a). *Support vector machines: Training and applications* (AI Memo No. 1602). Massachusetts Institute of Technology.

Osuna, E., Freund, R., & Girosi, F. (1997b). Training support vector machines: An application to face detection. In *Proceedings of CVPR'97.*

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods - support vector learning.* Cambridge, MA: MIT Press.

Platt, J. C., Cristianini, N., & Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification. In *Advances in neural information processing systems* (Vol. 12, p. 547-553). MIT Press.

Saunders, C., Stitson, M. O., Weston, J., Bottou, L., Schölkopf, B., & Smola, A. (1998). *Support vector machine reference manual* (Tech. Rep. No. CSD-TR-98-03). Egham, UK: Royal Holloway, University of London.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (1999). *Estimating the support of a high-dimensional distribution* (Tech. Rep. No. 99-87). Microsoft Research.

Schölkopf, B., Smola, A., Williamson, R. C., & Bartlett, P. L. (2000). New support vector algorithms. *Neural Computation*, *12*, 1207 – 1245.

Vapnik, V. (1998). *Statistical learning theory.* New York, NY: Wiley.

Weston, J., & Watkins, C. (1998). *Multi-class support vector machines* (Tech. Rep. No. CSD-TR-98-04). Royal Holloway.