LIBSVM: a Library for Support Vector Machines

Chih-Chung Chang and Chih-Jen Lin [*]

Last updated: February 27, 2009

Abstract

LIBSVM is a library for support vector machines (SVM). Its goal is to help users to easily use SVM as a tool. In this document, we present all its implementation details. For the use of LIBSVM, the README file included in the package and the LIBSVM FAQ provide the information.

1 Introduction

LIBSVM is a library for support vector machines (SVM). Its goal is to let users can easily use SVM as a tool. In this document, we present all its implementation details. For using LIBSVM, the README file included in the package provides the information.

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. We also support different penalty parameters for unbalanced data. Details are in Section 5. Then Section 6 discusses the implementation of multi-class classification. Parameter selection is important for obtaining good SVM models. LIBSVM provides simple and useful tools, which are discussed in Section 7. Section 8 presents the implementation of probability outputs.

2 Formulations

2.1 C-Support Vector Classification

Given training vectors $\mathbf{x}_i \in R^n, i = 1, \ldots, l$, in two classes, and a vector $\mathbf{y} \in R^l$ such that $y_i \in \{1, -1\}$, C-SVC (Boser et al., 1992; Cortes and Vapnik, 1995) solves the

[*] Department of Computer Science, National Taiwan University, Taipei 106, Taiwan (http://www.csie.ntu.edu.tw/~cjlin). E-mail: cjlin@csie.ntu.edu.tw

following primal problem:

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

Its dual is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \mathbf{y}^T\boldsymbol{\alpha} = 0, \\ & 0 \leq \alpha_i \leq C, \qquad i=1,\ldots,l, \end{aligned} \tag{2.2}$$

where $\mathbf{e}$ is the vector of all ones, $C > 0$ is the upper bound, Q is an l by l positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, and $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ is the kernel. Here training vectors $\mathbf{x}_i$ are mapped into a higher (maybe infinite) dimensional space by the function ϕ.

The decision function is

$$\text{sgn}\left(\sum_{i=1}^{l} y_i\alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\right).$$

2.2 ν-Support Vector Classification

The ν-support vector classification (Schölkopf et al., 2000) uses a new parameter ν which controls the number of support vectors and training 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.

Given training vectors $\mathbf{x}_i \in R^n, i=1,\ldots,l$, in two classes, and a vector $\mathbf{y} \in R^l$ such that $y_i \in \{1,-1\}$, the primal form considered is:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} - \nu\rho + \frac{1}{l}\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b) \geq \rho-\xi_i, \\ & \xi_i \geq 0, i=1,\ldots,l, \rho \geq 0. \end{aligned}$$

The dual is:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq 1/l, \qquad i = 1, \ldots, l, \\ & \mathbf{e}^T\boldsymbol{\alpha} \geq \nu, \quad \mathbf{y}^T\boldsymbol{\alpha} = 0. \end{aligned} \tag{2.3}$$

where $Q_{ij} \equiv y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$.

The decision function is:

$$\operatorname{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\right).$$

In (Crisp and Burges, 2000; Chang and Lin, 2001), it has been shown that $\mathbf{e}^T\boldsymbol{\alpha} \geq \nu$ can be replaced by $\mathbf{e}^T\boldsymbol{\alpha} = \nu$. With this property, in LIBSVM, we solve a scaled version of (2.3):

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

We output $\boldsymbol{\alpha}/\rho$ so the computed decision function is:

$$\operatorname{sgn}\left(\sum_{i=1}^{l} y_i (\alpha_i/\rho)(K(\mathbf{x}_i, \mathbf{x}) + b)\right)$$

and then two margins are

$$y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) = \pm 1$$

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. (2001) for estimating the support of a high-dimensional distribution. Given training vectors $\mathbf{x}_i \in R^n, i = 1, \ldots, l$ without any class information, the primal form in (Schölkopf et al., 2001) is:

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

The dual is:

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

where $Q_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$.

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

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le 1, \qquad i = 1, \ldots, l, \\ & \mathbf{e}^T \boldsymbol{\alpha} = \nu l. \end{aligned} \tag{2.5}$$

The decision function is

$$\operatorname{sgn}(\sum_{i=1}^{l} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - \rho).$$

2.4 ϵ-Support Vector Regression (ϵ-SVR)

Given a set of data points, $\{(\mathbf{x}_1, z_1), \ldots, (\mathbf{x}_l, z_l)\}$, such that $\mathbf{x}_i \in R^n$ is an input and $z_i \in R^1$ is a target output, the standard form of support vector regression (Vapnik, 1998) is:

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

The dual is:

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T Q (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \epsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} z_i (\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) = 0, 0 \le \alpha_i, \alpha_i^* \le C, i = 1, \ldots, l, \end{aligned} \tag{2.6}$$

where $Q_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$.

The approximate function is:

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

In LIBSVM implementation, we store $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ together in an array. Note that $\boldsymbol{\alpha}^*$ comes first so the combined array in the code is actually $\begin{bmatrix}\boldsymbol{\alpha}^* & \boldsymbol{\alpha}\end{bmatrix}^T$.

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 ν replaces with C, here ν replaces with the parameter ϵ of ϵ-SVR. The primal form is

$$\begin{aligned} \min_{\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \epsilon} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\nu\epsilon + \frac{1}{l}\sum_{i=1}^{l}(\xi_i + \xi_i^*)) \qquad (2.7) \\ \text{subject to} \quad & (\mathbf{w}^T\phi(\mathbf{x}_i) + b) - z_i \leq \epsilon + \xi_i, \\ & z_i - (\mathbf{w}^T\phi(\mathbf{x}_i) + b) \leq \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0, i = 1, \ldots, l, \ \epsilon \geq 0. \end{aligned}$$

and the dual is

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T Q (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \mathbf{z}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) \\ \text{subject to} \quad & \mathbf{e}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) = 0, \ \mathbf{e}^T(\boldsymbol{\alpha} + \boldsymbol{\alpha}^*) \leq C\nu, \\ & 0 \leq \alpha_i, \alpha_i^* \leq C/l, \qquad i = 1, \ldots, l, \qquad (2.8) \end{aligned}$$

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

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T Q (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \mathbf{z}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) \\ \text{subject to} \quad & \mathbf{e}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) = 0, \ \mathbf{e}^T(\boldsymbol{\alpha} + \boldsymbol{\alpha}^*) = Cl\nu, \\ & 0 \leq \alpha_i, \alpha_i^* \leq C, \qquad i = 1, \ldots, l. \qquad (2.9) \end{aligned}$$

The decision function is

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

which is the same as that of ϵ-SVR.

2.6 Performance Measures

After models are trained by solving the above optimization problems, users can apply LIBSVM to predict labels (target values) of testing data. Let $\mathbf{x}_1, \ldots, \mathbf{x}_{\bar{l}}$ be the testing data and $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_{\bar{l}})$ be LIBSVM's predicted decision values (target values for regression). If the true labels (target values) of testing data are known and denoted as $y_i, \ldots, y_{\bar{l}}$, we evaluate the predictions by the following measures:

2.6.1 Classification

$$
\begin{aligned}
& \text{Accuracy} \\
= & \frac{\#\text{ correctly predicted data}}{\#\text{ total data}} \times 100\% \\
= & \frac{|\{i \mid y_i f(\mathbf{x}_i) > 0\}|}{\bar{l}} \times 100\%.
\end{aligned}
$$

2.6.2 Regression

LIBSVM outputs MSE (mean squared error) and r^2 (squared correlation coefficient):

$$
\begin{aligned}
\text{MSE} &= \frac{1}{\bar{l}} \sum_{i=1}^{\bar{l}} \left(f(\mathbf{x}_i) - y_i\right)^2, \\
r^2 &= \frac{\left(\bar{l} \sum_{i=1}^{\bar{l}} f(\mathbf{x}_i) y_i - \sum_{i=1}^{\bar{l}} f(\mathbf{x}_i) \sum_{i=1}^{\bar{l}} y_i\right)^2}{\left(\bar{l} \sum_{i=1}^{\bar{l}} f(\mathbf{x}_i)^2 - \left(\sum_{i=1}^{\bar{l}} f(\mathbf{x}_i)\right)^2\right) \left(\bar{l} \sum_{i=1}^{\bar{l}} y_i^2 - \left(\sum_{i=1}^{\bar{l}} y_i\right)^2\right)}.
\end{aligned}
$$

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:

$$
\begin{aligned}
\min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \mathbf{p}^T \boldsymbol{\alpha} \\
\text{subject to} \quad & \mathbf{y}^T \boldsymbol{\alpha} = \Delta, \\
& 0 \leq \alpha_t \leq C, t = 1, \ldots, l,
\end{aligned}
\tag{3.1}
$$

where $y_t = \pm 1, t = 1, \ldots, l$. 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 (2.6):

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}\left[\boldsymbol{\alpha}^T,(\boldsymbol{\alpha}^*)^T\right]\begin{bmatrix} Q & -Q \\ -Q & Q \end{bmatrix}\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} + \left[\epsilon\mathbf{e}^T+\mathbf{z}^T,\epsilon\mathbf{e}^T-\mathbf{z}^T\right]\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} \\ \text{subject to} \quad & \mathbf{y}^T\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} = 0, 0 \le \alpha_t, \alpha_t^* \le C, t = 1,\ldots,l, \end{aligned} \tag{3.2}$$

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

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., 1997a; Joachims, 1998; Platt, 1998). This method modifies only a subset of $\boldsymbol{\alpha}$ per iteration. This subset, denoted as the working set B, leads to a small sub-problem to be minimized in each iteration. An extreme case is the Sequential Minimal Optimization (SMO) (Platt, 1998), which restricts B to have only two elements. Then in each iteration one solves a simple two-variable problem without needing optimization software. Here we consider an SMO-type decomposition method proposed in Fan et al. (2005).

Algorithm 1 (An SMO-type Decomposition method in Fan et al. (2005))

1. Find $\boldsymbol{\alpha}^1$ as the initial feasible solution. Set $k = 1$.

2. If $\boldsymbol{\alpha}^k$ is a stationary point of (2.2), stop. Otherwise, find a *two-element* working set $B = \{i, j\}$ by WSS 1 (described in Section 3.2). Define $N \equiv \{1,\ldots,l\}\backslash B$ and $\boldsymbol{\alpha}_B^k$ and $\boldsymbol{\alpha}_N^k$ to be sub-vectors of $\boldsymbol{\alpha}^k$ corresponding to B and N, respectively.

3. If $a_{ij} \equiv K_{ii} + K_{jj} - 2K_{ij} > 0$

   Solve the following sub-problem with the variable $\boldsymbol{\alpha}_B$:

$$\begin{aligned} \min_{\alpha_i,\alpha_j} \quad & \frac{1}{2}\begin{bmatrix} \alpha_i & \alpha_j \end{bmatrix}\begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix}\begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + (\mathbf{p}_B + Q_{BN}\boldsymbol{\alpha}_N^k)^T\begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} \\ \text{subject to} \quad & 0 \le \alpha_i, \alpha_j \le C, \\ & y_i\alpha_i + y_j\alpha_j = \Delta - \mathbf{y}_N^T\boldsymbol{\alpha}_N^k, \end{aligned} \tag{3.3}$$

   else

Solve

$$\begin{aligned}\min_{\alpha_i,\alpha_j} \quad & \frac{1}{2}\begin{bmatrix}\alpha_i & \alpha_j\end{bmatrix}\begin{bmatrix}Q_{ii} & Q_{ij}\\ Q_{ij} & Q_{jj}\end{bmatrix}\begin{bmatrix}\alpha_i\\ \alpha_j\end{bmatrix} + (\mathbf{p}_B + Q_{BN}\boldsymbol{\alpha}_N^k)^T\begin{bmatrix}\alpha_i\\ \alpha_j\end{bmatrix}\\ & +\frac{\tau - a_{ij}}{4}((\alpha_i - \alpha_i^k)^2 + (\alpha_j - \alpha_j^k)^2) \qquad (3.4)\\ \text{subject to} \quad & \text{constraints of (3.3).}\end{aligned}$$

4. Set $\boldsymbol{\alpha}_B^{k+1}$ to be the optimal solution of (3.3) and $\boldsymbol{\alpha}_N^{k+1} \equiv \boldsymbol{\alpha}_N^k$. Set $k \leftarrow k+1$ and goto Step 2.

Note that B is updated at each iteration. To simplify the notation, we simply use B instead of B^k. If $a_{ij} \leq 0$, (3.3) is a concave problem. Hence we use a convex modification in (3.4).

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

The Karush-Kuhn-Tucker (KKT) optimality condition of (3.1) shows that a vector $\boldsymbol{\alpha}$ is a stationary point of (3.1) if and only if there is a number b[†] and two nonnegative vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ such that

$$\begin{gathered}\nabla f(\boldsymbol{\alpha}) + b\mathbf{y} = \boldsymbol{\lambda} - \boldsymbol{\mu},\\ \lambda_i\alpha_i = 0, \mu_i(C - \alpha_i) = 0, \lambda_i \geq 0, \mu_i \geq 0, i = 1, \ldots, l,\end{gathered}$$

where $\nabla f(\boldsymbol{\alpha}) \equiv Q\boldsymbol{\alpha} + \mathbf{p}$ is the gradient of $f(\boldsymbol{\alpha})$. This condition can be rewritten as

$$\nabla f(\boldsymbol{\alpha})_i + by_i \geq 0 \qquad \text{if } \alpha_i < C, \qquad (3.5)$$

$$\nabla f(\boldsymbol{\alpha})_i + by_i \leq 0 \qquad \text{if } \alpha_i > 0. \qquad (3.6)$$

Since $y_i = \pm 1$, by defining

$$\begin{aligned} I_{\text{up}}(\boldsymbol{\alpha}) &\equiv \{t \mid \alpha_t < C, y_t = 1 \text{ or } \alpha_t > 0, y_t = -1\}, \text{ and}\\ I_{\text{low}}(\boldsymbol{\alpha}) &\equiv \{t \mid \alpha_t < C, y_t = -1 \text{ or } \alpha_t > 0, y_t = 1\},\end{aligned} \qquad (3.7)$$

a feasible $\boldsymbol{\alpha}$ is a stationary point of (3.1) if and only if

$$m(\boldsymbol{\alpha}) \leq M(\boldsymbol{\alpha}), \qquad (3.8)$$

[†] From the primal-dual relationship, this b is the same as the one in the primal problem.

where

$$m(\boldsymbol{\alpha}) \equiv \max_{i \in I_{\text{up}}(\boldsymbol{\alpha})} -y_i \nabla f(\boldsymbol{\alpha})_i, \text{ and } M(\boldsymbol{\alpha}) \equiv \min_{i \in I_{\text{low}}(\boldsymbol{\alpha})} -y_i \nabla f(\boldsymbol{\alpha})_i.$$

From this we have the following stopping condition:

$$m(\boldsymbol{\alpha}^k) - M(\boldsymbol{\alpha}^k) \leq \epsilon. \tag{3.9}$$

About the selection of the working set set B, we consider the following procedure:

WSS 1

1. For all t, s, define

$$a_{ts} \equiv K_{tt} + K_{ss} - 2K_{ts}, b_{ts} \equiv -y_t \nabla f(\boldsymbol{\alpha}^k)_t + y_s \nabla f(\boldsymbol{\alpha}^k)_s > 0 \tag{3.10}$$

   and

$$\bar{a}_{ts} \equiv \begin{cases} a_{ts} & \text{if } a_{ts} > 0, \\ \tau & \text{otherwise.} \end{cases} \tag{3.11}$$

   Select

$$\begin{aligned} i &\in \arg\max_t \{-y_t \nabla f(\boldsymbol{\alpha}^k)_t \mid t \in I_{\text{up}}(\boldsymbol{\alpha}^k)\}, \\ j &\in \arg\min_t \left\{ -\frac{b_{it}^2}{\bar{a}_{it}} \mid t \in I_{\text{low}}(\boldsymbol{\alpha}^k), -y_t \nabla f(\boldsymbol{\alpha}^k)_t < -y_i \nabla f(\boldsymbol{\alpha}^k)_i \right\}. \end{aligned} \tag{3.12}$$

2. Return $B = \{i, j\}$.

Details of how we choose this working set is in (Fan et al., 2005, Section II).

3.3 Convergence of the Decomposition Method

See (Fan et al., 2005, Section III) or (Chen et al., 2006) for a detailed discussion of the convergence of Algorithm 1.

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:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \mathbf{p}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & \mathbf{y}^T \boldsymbol{\alpha} = \Delta_1, \\ & \mathbf{e}^T \boldsymbol{\alpha} = \Delta_2, \\ & 0 \leq \alpha_t \leq C, t = 1, \ldots, l. \end{aligned} \tag{3.13}$$

The KKT condition of (3.13) shows

$$\begin{aligned}\nabla f(\boldsymbol{\alpha})_i - \rho + b y_i &= 0 \text{ if } 0 < \alpha_i < C,\\ &\geq 0 \text{ if } \alpha_i = 0,\\ &\leq 0 \text{ if } \alpha_i = C.\end{aligned}$$

Define

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

If $y_i = 1$ the KKT condition becomes

$$\begin{aligned}\nabla f(\boldsymbol{\alpha})_i - r_1 &\geq 0 \text{ if } \alpha_i < C,\\ &\leq 0 \text{ if } \alpha_i > 0.\end{aligned} \tag{3.14}$$

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

$$\begin{aligned}\nabla f(\boldsymbol{\alpha})_i - r_2 &\geq 0 \text{ if } \alpha_i < C,\\ &\leq 0 \text{ if } \alpha_i > 0.\end{aligned} \tag{3.15}$$

Hence given a tolerance $\epsilon > 0$, the stopping condition is:

$$\max\left(m_p(\boldsymbol{\alpha}) - M_p(\boldsymbol{\alpha}), m_n(\boldsymbol{\alpha}) - M_n(\boldsymbol{\alpha})\right) < \epsilon, \tag{3.16}$$

where

$$\begin{aligned} m_p(\boldsymbol{\alpha}) &\equiv \max_{i \in I_{\text{up}}(\boldsymbol{\alpha}), y_i = 1} -y_i \nabla f(\boldsymbol{\alpha})_i, \qquad & M_p(\boldsymbol{\alpha}) &\equiv \min_{i \in I_{\text{low}}(\boldsymbol{\alpha}), y_i = 1} -y_i \nabla f(\boldsymbol{\alpha})_i, \text{ and}\\ m_n(\boldsymbol{\alpha}) &\equiv \max_{i \in I_{\text{up}}(\boldsymbol{\alpha}), y_i = -1} -y_i \nabla f(\boldsymbol{\alpha})_i, \qquad & M_n(\boldsymbol{\alpha}) &\equiv \min_{i \in I_{\text{low}}(\boldsymbol{\alpha}), y_i = -1} -y_i \nabla f(\boldsymbol{\alpha})_i.\end{aligned}$$

The working set selection is by extending WSS 1 to the following

WSS 2 (Extending WSS 1 to ν-SVM)

1. Find

   $i_p \in \arg m_p(\boldsymbol{\alpha}^k)$,

   $$j_p \in \arg\min_t \left\{ -\frac{b_{i_p t}^2}{\bar{a}_{i_p t}} \mid y_t = 1, \boldsymbol{\alpha}_t \in I_{\text{low}}(\boldsymbol{\alpha}^k), -y_t \nabla f(\boldsymbol{\alpha}^k)_t < -y_{i_p} \nabla f(\boldsymbol{\alpha}^k)_{i_p} \right\}.$$

2. Find

   $i_n \in \arg m_n(\boldsymbol{\alpha}^k)$,

   $$j_n \in \arg\min_t \left\{ -\frac{b_{i_n t}^2}{\bar{a}_{i_n t}} \mid y_t = -1, \boldsymbol{\alpha}_t \in I_{\text{low}}(\boldsymbol{\alpha}^k), -y_t \nabla f(\boldsymbol{\alpha}^k)_t < -y_{i_n} \nabla f(\boldsymbol{\alpha}^k)_{i_n} \right\}.$$

3. Return $\{i_p, j_p\}$) or $\{i_n, j_n\}$ depending on which one gives smaller $-b_{ij}^2/\bar{a}_{ij}$.

3.5 Analytical Solutions

Details are described in Section 5 in which we discuss the solution of a more general sub-problem.

3.6 The Calculation of b or ρ

After the solution $\boldsymbol{\alpha}$ of the dual optimization problem is obtained, the variables b or ρ must be calculated as they are used in the decision function.

We first describe the case of C-SVC, ϵ-SVR, and one-class SVM. For this case, b has the same role as $-\rho$ in one-class SVM, so we define $rho = -b$ and discuss how to find it. If there are α_i which satisfy $0 < \alpha_i < C$, then from the KKT condition (3.8), $\rho = y_i \nabla f(\boldsymbol{\alpha})_i$. Practically to avoid numerical errors, we average them:

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

On the other hand, if there is no such α_i, the KKT condition becomes

$$\begin{aligned} & -M(\boldsymbol{\alpha}) = \max\{y_i \nabla f(\boldsymbol{\alpha})_i \mid \alpha_i = 0, y_i = -1 \text{ or } \alpha_i = C, y_i = 1\} \\ \leq \; & \rho \\ \leq \; & -m(\boldsymbol{\alpha}) = \min\{y_i \nabla f(\boldsymbol{\alpha})_i \mid \alpha_i = 0, y_i = 1 \text{ or } \alpha_i = C, y_i = -1\}. \end{aligned}$$

We take ρ the midpoint of the range.

For the case of ν-SVC and ν-SVR, b and ρ both appear. The KKT condition of (3.13) has been shown in (3.14) and (3.15). Now we consider the case of $y_i = 1$. If there are α_i which satisfy $0 < \alpha_i < C$, then $r_1 = \nabla f(\boldsymbol{\alpha})_i$. Practically to avoid numerical errors, we average them:

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

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

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

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,

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

3.7 Initial Values

In the beginning of Algorithm 1, we must give a feasible $\boldsymbol{\alpha}$. For C-SVC and ϵ-SVR, the initial $\boldsymbol{\alpha}$ is simply zero. For ν-SVC, after scaling we have from (2.5) that

$$\alpha_i \leq 1 \text{ and } \sum_{i:y_i=1} \alpha_i = \frac{\nu l}{2}.$$

We therefore let the first $\frac{\nu l}{2}$ elements of α_i with $y_i = 1$ to have the value one. The situation for $y_i = -1$ is similar.

We use the same setting for one-class SVM and ν-SVR.

4 Shrinking and Caching

4.1 Shrinking

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. Indeed we can have the following theorem which shows that at the final iterations of the decomposition methods proposed in Section 3.2, only variables corresponding to a small set are still allowed to move:

Theorem 4.1 (Theorem IV in (Fan et al., 2005)) *Assume Q is positive semi-definite.*

1. *The following set is independent of any optimal solution $\bar{\boldsymbol{\alpha}}$:*

$$I \equiv \{i \mid -y_i \nabla f(\bar{\boldsymbol{\alpha}})_i > M(\bar{\boldsymbol{\alpha}}) \textit{ or } -y_i \nabla f(\bar{\boldsymbol{\alpha}})_i < m(\bar{\boldsymbol{\alpha}})\}. \tag{4.1}$$

Problem (2.2) has unique and bounded optimal solutions at $\alpha_i, i \in I$.

2. *Assume Algorithm 1 generates an infinite sequence $\{\boldsymbol{\alpha}^k\}$. There is $\bar{k}$ such that after $k \geq \bar{k}$, every $\alpha_i^k, i \in I$ has reached the unique and bounded optimal solution. It remains the same in all subsequent iterations and $\forall k \geq \bar{k}$:*

$$i \notin \{t \mid M(\boldsymbol{\alpha}^k) \leq -y_t \nabla f(\boldsymbol{\alpha}^k)_t \leq m(\boldsymbol{\alpha}^k)\}. \tag{4.2}$$

Hence instead of solving the whole problem (2.2), the decomposition method works on a smaller problem:

$$
\begin{aligned}
\min_{\boldsymbol{\alpha}_A} \quad & \frac{1}{2}\boldsymbol{\alpha}_A^T Q_{AA}\boldsymbol{\alpha}_A - (\mathbf{p}_A - Q_{AN}\boldsymbol{\alpha}_N^k)^T\boldsymbol{\alpha}_A \\
\text{subject to} \quad & 0 \le (\boldsymbol{\alpha}_A)_t \le C, t = 1, \ldots, q, \\
& \mathbf{y}_A^T\boldsymbol{\alpha}_A = \Delta - \mathbf{y}_N^T\boldsymbol{\alpha}_N^k,
\end{aligned}
\tag{4.3}
$$

where $N = \{1, \ldots, l\}\backslash A$ is the set of shrunk variables.

Of course this heuristic may fail if some elements are wrongly shrunk. When that happens, the whole problem (2.2) is reoptimized from a starting point $\boldsymbol{\alpha}$ where $\boldsymbol{\alpha}_A$ optimal for (4.3) and $\boldsymbol{\alpha}_N$ corresponds to shrunk bounded variables. Note that while solving the shrunk problem (4.3), we only know the gradient $Q_{AA}\boldsymbol{\alpha}_A + Q_{AN}\boldsymbol{\alpha}_N + \mathbf{p}_A$ of (4.3). Hence when problem (2.2) is reoptimized we must reconstruct the whole gradient $\nabla f(\boldsymbol{\alpha})$, which is discussed in Section 4.3.

Several SVM implementations begin the shrinking procedure near the end of the iterative process, but in LIBSVM, we start 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

$$
m(\boldsymbol{\alpha}^k) > M(\boldsymbol{\alpha}^k) \tag{4.4}
$$

as (3.9) is not satisfied yet. Following Theorem 4.1, we conjecture that variables in the following set can be shrunk:

$$
\begin{aligned}
&\{t \mid -y_t\nabla f(\boldsymbol{\alpha}^k)_t > m(\boldsymbol{\alpha}^k), t \in I_{\text{low}}(\boldsymbol{\alpha}^k), \alpha_t^k \text{ is bounded}\}\cup \\
&\quad \{t \mid -y_t\nabla f(\boldsymbol{\alpha}^k)_t < M(\boldsymbol{\alpha}^k), t \in I_{\text{up}}(\boldsymbol{\alpha}^k), \alpha_t^k \text{ is bounded}\} \\
=&\{t \mid -y_t\nabla f(\boldsymbol{\alpha}^k)_t > m(\boldsymbol{\alpha}^k), \alpha_t^k = C, y_t = 1 \text{ or } \alpha_t^k = 0, y_t = -1\}\cup \\
&\quad \{t \mid -y_t\nabla f(\boldsymbol{\alpha}^k)_t < M(\boldsymbol{\alpha}^k), \alpha_t^k = 0, y_t = 1 \text{ or } \alpha_t^k = C, y_t = -1\}.
\end{aligned}
\tag{4.5}
$$

Thus the set A of activated variables is dynamically reduced in every $\min(l, 1000)$ iterations. The problem (4.3) is thus constantly changed. Note that the above $m(\boldsymbol{\alpha}^k)$ and $M(\boldsymbol{\alpha}^k)$ are calculated using (4.3).

2. Of course the above shrinking strategy may be too aggressive. Since the decomposition method has 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 shrunk problem (4.3). Hence when the decomposition method first achieves the condition

$$m(\boldsymbol{\alpha}^k) \leq M(\boldsymbol{\alpha}^k) + 10\epsilon, \tag{4.6}$$

where ϵ is the specified stopping tolerance, we reconstruct the whole gradient. Details are in Section 4.3.

For ν-SVC and ν-SVR, as the stopping condition (3.16) is different from (3.9), the set (4.5) must be modified. For $y_t = 1$, we shrink elements in the following set

$$\begin{aligned}&\{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t > m_p(\boldsymbol{\alpha}^k), \alpha_t = C, y_t = 1\} \cup \\ &\quad \{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t < M_p(\boldsymbol{\alpha}^k), \alpha_t = 0, y_t = 1\}.\end{aligned}$$

For $y_t = -1$, we consider the following set:

$$\begin{aligned}&\{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t > m_n(\boldsymbol{\alpha}^k), \alpha_t = 0, y_t = -1\} \cup \\ &\quad \{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t < M_n(\boldsymbol{\alpha}^k), \alpha_t = C, y_t = -1\}.\end{aligned}$$

4.2 Caching

An effective 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. One can use a special storage called cache to store recently used Q_{ij} (Joachims, 1998). Then some kernel elements do not need to be recalculated.

Theorem 4.1 also supports the use of caching as in final iterations only some columns of the matrix Q are still needed. If the cache can contain these columns, we can avoid most kernel evaluations in final iterations.

In LIBSVM, we consider a simple least-recent-use strategy for the cache. We use a circular list of structures. Each structure corresponds to a kernel column and caches several elements of that column. As (4.3) is dynamically reduced, if a column i is needed but not in the cache, we calculate and store only $Q_{1,i}, \ldots, Q_{|A|,i}$. Therefore, columns cached in the computer memory may be in different length.

4.3 Reconstructing the Gradient

To decrease the cost of reconstructing the gradient $\nabla f(\boldsymbol{\alpha})$, during the iterations we maintain

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

Then for $\nabla f(\boldsymbol{\alpha})_i, i \notin A$, we have

$$\nabla f(\boldsymbol{\alpha})_i = \sum_{j=1}^{l} Q_{ij}\alpha_j + p_i = \bar{G}_i + p_i + \sum_{\substack{j:j\in A \\ 0<\alpha_j<C}} Q_{ij}\alpha_j. \tag{4.7}$$

We use the fact that if $j \notin A$, then $\alpha_j = 0$ or C.

To calculate (4.7), we need a two-level loop over i and j. Using i or j first may result in a very different number of Q_{ij} evaluations. We discuss their differences below. Note that in our implementation, we always swap elements of $\boldsymbol{\alpha}$ to maintain that $A = \{1, \ldots, |A|\}$.

1. For $|A| + 1 \leq i \leq l$, calculate $Q_{i,1:|A|}$. Though in (4.7), only $\{Q_{ij} \mid 0 < \alpha_j < C, j \in A\}$ are needed, due to our cache implementation, we must obtain all elements of $Q_{i,1:|A|}$ (i.e., $\{Q_{ij} \mid j \in A\}$). This method needs at most

   $$(l - |A|) \cdot |A| \tag{4.8}$$

   kernel evaluations.

2. Let $F = \{j \mid 1 \leq j \leq |A| \text{ and } 0 < \alpha_j < C\}$. For each $j \in F$, we obtain $Q_{1:l,j}$. Though only $Q_{|A|+1:l,j}$ is needed in calculating $\nabla f(\boldsymbol{\alpha})_i, i = |A|+1, \ldots, l$, due to our cache implementation, we must get the whole column. This method needs at most

   $$l \cdot |F| \tag{4.9}$$

   kernel evaluations.

We may choose a method by comparing (4.8) and (4.9). However, the decision depends on whether Q's elements have been cached. If the cache is large enough, then elements of Q's first $|A|$ columns tend to be in the cache as they have been recently used. Therefore, while method 1 may still require $(l-|A|)\cdot|A|$ kernel valuations, method 2 may need fewer evaluations than $l\cdot|F|$.

Therefore, as method 2 takes an advantage of the cache implementation, we consider the following rule:

```
If  $l \cdot |F| > 2 \cdot (l - |A|) \cdot |A|$
        use method 1
Else
        use method 2
```

This rule may not give the optimal choice as we do not take the cache contents into account. However, in the worst scenario, the selected method by the above rule is only slightly slower than the other method. Before giving some detailed explanations below, we make several assumptions:

- A LIBSVM training procedure involves two gradient reconstructions: The first is performed when the 10ϵ tolerance is achieved; see Eq. (4.6). The second is in the end of the training procedure.
- Our rule assigns the same method to perform the two gradient reconstructions. Moreover, these two reconstructions cost a similar amount of time.

We refer to "total training time of method 1" as the whole LIBSVM training time (where method 1 is used for reconstructing gradients), and "reconstruction time of method 1" as the time of one single gradient reconstruction via method 1. We then consider two situations:

1. Method 1 is chosen but method 2 is better.

   We have

$$\begin{aligned} & \text{Total time of method 1} \\ \leq\; & (\text{Total time of method 2}) + 2 \times (\text{Reconstruction time of method 1}) \\ \leq\; & 2 \times (\text{Total time of method 2}). \end{aligned} \tag{4.10}$$

   We explain the second inequality in detail. If method 2 is used for reconstructing gradients, during the training procedure, $Q_{:,j}, j \in F$ must have been calculated (either in regular iterations or reconstructing gradients). Since $l \cdot |F| > 2 \cdot (l - |A|) \cdot |A|$,

$$\text{(Reconstruction time of method 1)} \leq \frac{1}{2} \cdot \text{(Total time of method 2)}.$$

Hence (4.10) follows.

2. Method 2 is chosen but method 1 is better.

   We consider the worst situation where elements of Q's first $|A|$ columns are not in the cache. As $|A|+1, \ldots, l$ are indices of shrunk variables, most likely the remaining $l - |A|$ columns of Q are not in the cache either. Since $l \cdot |F| \leq 2 \cdot (l - |A|) \cdot |A|$,

   $$\text{(Reconstruction time of method 2)} \leq 2 \cdot \text{(Reconstruction time of method 1)}.$$

   Therefore,

   $$\begin{aligned} & \text{Total time of method 2} \\ \leq \quad & \text{(Total time of method 1)} + 2 \times \text{(Reconstruction time of method 1)}. \end{aligned}$$

Table 4.1 compares the number of kernel evaluations in reconstructing the gradient. We consider problems a7a and ijcnn1[‡]. Clearly, the proposed rule selects the better method for both problems. We implement this technique after version 2.88 of LIBSVM.

4.4 Is Shrinking Always Better?

We found that if the number of iterations is large, then shrinking can shorten the training time. However, if we stop the training procedure early (i.e., using a large ϵ as the stopping tolerance), the implementation without shrinking may be much faster. In this situation, the number of iterations is often small. The time spent on all iterations can be even smaller than one single gradient reconstruction.

In Table 4.1, we show the total training time without using shrinking. For a7a, we use the default $\epsilon = 0.001$. Under the parameters $C = 1$ and $\gamma = 4$, the number of

[‡] Available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`

Cache = 1000 MB		Cache = 10 MB				
Method 1	Method 2	Method 1	Method 2	$\lvert F\rvert$	$\lvert A\rvert$	l
0	21,470,526	45,213,024	170,574,272	10,597	12,476	16,100
0	0	45,213,024	171,118,048	10,630	12,476	16,100
102s	108s	341s	422s			
No shrinking: 111s		No shrinking: 381s				

(a) a7a: $C = 1, \gamma = 4, \epsilon = 0.001$.

Cache = 1000 MB		Cache = 10 MB				
Method 1	Method 2	Method 1	Method 2	$\lvert F\rvert$	$\lvert A\rvert$	l
274,297,840	5,403,072	275,695,536	88,332,330	1,767	43,678	49,990
263,843,538	28,274,195	264,813,241	115,346,805	2,308	6,023	49,990
189s	46s	203s	116s			
No shrinking: 42s		No shrinking: 87s				

(b) ijcnn1: $C = 16, \gamma = 4, \epsilon = 0.5$.

Table 4.1: The decomposition method reconstructs the gradient twice. We show in each row the number of kernel evaluations of a reconstruction. We check two cache sizes to reflect the situations with/without enough cache. The last row gives the total training time in seconds (where gradient constructions are only part of it). We consider the RBF kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$.

iterations is more than 30,000. Then shrinking is useful. However, for ijcnn1, we use a loose tolerance $\epsilon = 0.5$, so the number of iterations is only around 4,000. Since our shrinking strategy is quite aggressive, before the first gradient reconstruction, only $Q_{F,F}$ are in the cache, Then we need to obtain $Q_{:,F}$ in order to calculate the gradient.

If there are enough iterations, most elements in A should be free. In contrast, if the number of iterations is small (e.g., ijcnn1 in Table 4.1), we have $|F| \ll |A|$. Therefore, we can check the relation between $|F|$ and $|A|$ to conjecture if shrinking if useful. In LIBSVM, if shrinking is enabled and $2 \cdot |F| < |A|$ in reconstructing the gradient, we issue a warning message to indicate that the code may be faster without shrinking.

4.5 Computational Complexity

The discussion in Section 3.3 is about the asymptotic convergence of the decomposition method. Here, we discuss the computational complexity.

The main operations are on finding $Q_{BN}\boldsymbol{\alpha}_N^k + \mathbf{p}_B$ of (3.3) and the update of

$\nabla f(\boldsymbol{\alpha}^k)$ to $\nabla f(\boldsymbol{\alpha}^{k+1})$. Note that $\nabla f(\boldsymbol{\alpha})$ is used in the working set selection as well as the stopping condition. They can be considered together as

$$Q_{BN}\boldsymbol{\alpha}_N^k + \mathbf{p}_B = \nabla f(\boldsymbol{\alpha}^k) - Q_{BB}\boldsymbol{\alpha}_B^k, \tag{4.11}$$

and

$$\nabla f(\boldsymbol{\alpha}^{k+1}) = \nabla f(\boldsymbol{\alpha}^k) + Q_{:,B}(\boldsymbol{\alpha}_B^{k+1} - \boldsymbol{\alpha}_B^k), \tag{4.12}$$

where $Q_{:,B}$ is the sub-matrix of Q with column in B. That is, at the kth iteration, as we already have $\nabla f(\boldsymbol{\alpha}^k)$, the right-hand-side of (4.11) is used to construct the sub-problem. After the sub-problem is solved, (4.12) is employed to have the next $\nabla f(\boldsymbol{\alpha}^{k+1})$. As B has only two elements and solving the sub-problem is easy, the main cost is $Q_{:,B}(\boldsymbol{\alpha}_B^{k+1} - \boldsymbol{\alpha}_B^k)$ of (4.12). The operation itself takes $O(2l)$ but if $Q_{:,B}$ is not available in the cache and each kernel evaluation costs $O(n)$, one column indexes of $Q_{:,B}$ already needs $O(ln)$. Therefore, the complexity is:

1. #Iterations × $O(l)$ if most columns of Q are cached during iterations.

2. #Iterations × $O(nl)$ if most columns of Q are cached during iterations and each kernel evaluation is $O(n)$.

Note that if shrinking is incorporated, l will gradually decrease during iterations.

5 Unbalanced Data

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

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

Its dual is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C_+, \text{ if } y_i = 1, & (5.1) \\ & 0 \le \alpha_i \le C_-, \text{ if } y_i = -1, & (5.2) \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0. \end{aligned}$$

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 subproblem (3.3). Now it becomes:

$$\begin{aligned} \min_{\alpha_i, \alpha_j} \quad & \frac{1}{2} \begin{bmatrix} \alpha_i & \alpha_j \end{bmatrix} \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ji} & Q_{jj} \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + (Q_{i,N}\boldsymbol{\alpha}_N - 1)\alpha_i + (Q_{j,N}\boldsymbol{\alpha}_N - 1)\alpha_j \\ \text{subject to} \quad & y_i\alpha_i + y_j\alpha_j = \Delta - \mathbf{y}_N^T \boldsymbol{\alpha}_N^k, & (5.3) \\ & 0 \le \alpha_i \le C_i, 0 \le \alpha_j \le C_j, \end{aligned}$$

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

Let $\alpha_i = \alpha_i^k + d_i, \alpha_j = \alpha_j^k + d_j$ and $\hat{d}_i \equiv y_i d_i, \hat{d}_j \equiv y_j d_j$. Then (5.3) can be written as

$$\begin{aligned} \min_{d_i, d_j} \quad & \frac{1}{2} \begin{bmatrix} d_i & d_j \end{bmatrix} \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} + \begin{bmatrix} \nabla f(\boldsymbol{\alpha}^k)_i & \nabla f(\boldsymbol{\alpha}^k)_j \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} \\ \text{subject to} \quad & y_i d_i + y_j d_j = 0, & (5.4) \\ & -\alpha_i^k \le d_i \le C - \alpha_i^k, -\alpha_j^k \le d_j \le C - \alpha_j^k. \end{aligned}$$

Define a_{ij} and b_{ij} as in (3.10). Note that if $a_{ij} \le 0$, then a modification similar to (3.4). Using $\hat{d}_i = -\hat{d}_j$, the objective function can be written as

$$\frac{1}{2}\bar{a}_{ij}\hat{d}_j^2 + b_{ij}\hat{d}_j.$$

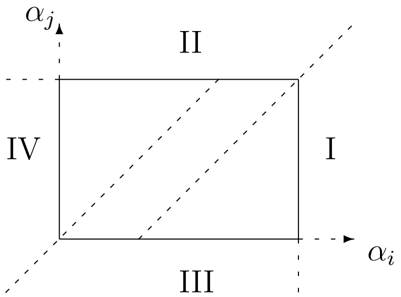
Thus,

$$\begin{aligned} \alpha_i^{\text{new}} &= \alpha_i^k + y_i b_{ij}/\bar{a}_{ij}, \\ \alpha_j^{\text{new}} &= \alpha_i^k - y_j b_{ij}/\bar{a}_{ij}. & (5.5) \end{aligned}$$

To modify them back to the feasible region, we first consider the case $y_i \neq y_j$ and write (5.5) as

$$\begin{aligned}\alpha_i^{\text{new}} &= \alpha_i^k + (-\nabla f(\boldsymbol{\alpha}^k)_i - \nabla f(\boldsymbol{\alpha}^k)_j)/\bar{a}_{ij},\\ \alpha_j^{\text{new}} &= \alpha_i^k + (-\nabla f(\boldsymbol{\alpha}^k)_i - \nabla f(\boldsymbol{\alpha}^k)_j)/\bar{a}_{ij}.\end{aligned}$$

If $\boldsymbol{\alpha}^{\text{new}}$ is not feasible, $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ is in one of the following four regions:



If it is in region I, α_i^{k+1} is set to be C_i first and then

$$\alpha_j^{k+1} = C_i - (\alpha_i^k - \alpha_j^k).$$

Of course we must check if it is in region I first. If so, we have

$$\alpha_i^k - \alpha_j^k > C_i - C_j \text{ and } \alpha_i^{\text{new}} \geq C_i.$$

Other cases are similar. Therefore, we have the following procedure to identify $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ in different regions and change it back to the feasible set.

```
if(y[i]!=y[j])
{
        double quad_coef = Q_i[i]+Q_j[j]+2*Q_i[j];
        if (quad_coef <= 0)
                quad_coef = TAU;
        double delta = (-G[i]-G[j])/quad_coef;
        double diff = alpha[i] - alpha[j];
        alpha[i] += delta;
        alpha[j] += delta;

        if(diff > 0)
        {
                if(alpha[j] < 0)  // in region III
                {
                        alpha[j] = 0;
```

```
                                alpha[i] = diff;
                        }
                }
                else
                {
                        if(alpha[i] < 0)  // in region IV
                        {
                                alpha[i] = 0;
                                alpha[j] = -diff;
                        }
                }
                if(diff > C_i - C_j)
                {
                        if(alpha[i] > C_i)  // in region I
                        {
                                alpha[i] = C_i;
                                alpha[j] = C_i - diff;
                        }
                }
                else
                {
                        if(alpha[j] > C_j)  // in region II
                        {
                                alpha[j] = C_j;
                                alpha[i] = C_j + diff;
                        }
                }
        }
```

6 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 two-class classification problem:

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

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 $\mathbf{x}$ - in the end point is designated to be in a class with maximum number of votes.

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

There are other methods for multi-class classification. Some reasons why we choose this “1-against-1” approach and detailed comparisons are in Hsu and Lin (2002).

7 Parameter Selection

LIBSVM provides a parameter selection tool using the RBF kernel: cross validation via parallel grid search. While cross validation is available for both SVC and SVR, for the grid search, currently we support only C-SVC with two parameters C and γ. They can be easily modified for other kernels such as linear and polynomial, or for SVR.

For median-sized problems, cross validation might be the most reliable way for parameter selection. First, the training data is separated to several folds. Sequentially a fold is considered as the validation set and the rest are for training. The average of accuracy on predicting the validation sets is the cross validation accuracy.

Our implementation is as follows. Users provide a possible interval of C (or γ) with the grid space. Then, all grid points of (C, γ) are tried to see which one gives the highest cross validation accuracy. Users then use the best parameter to train the whole training set and generate the final model.

For easy implementation, we consider each SVM with parameters (C, γ) as an independent problem. As they are different jobs, we can easily solve them in parallel.

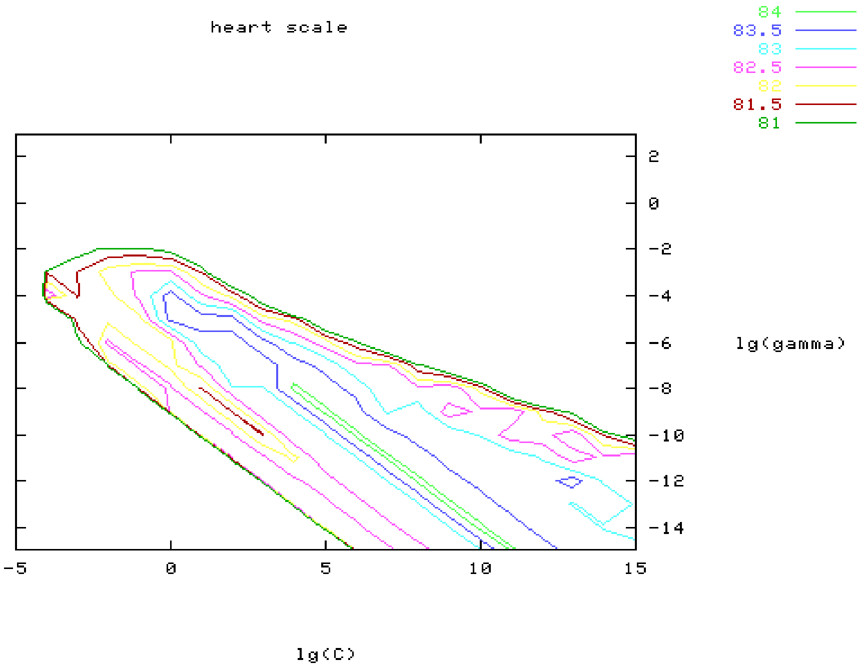


Figure 1: Contour plot of heart_scale included in the LIBSVM package

Currently, LIBSVM provides a very simple tool so that jobs are dispatched to a cluster of computers which share the same file system.

Note that now under the same (C, γ), the one-against-one method is used for training multi-class data. Hence, in the final model, all $k(k-1)/2$ decision functions share the same (C, γ).

LIBSVM also outputs the contour plot of cross validation accuracy. An example is in Figure 1.

8 Probability Estimates

Support vector classification (regression) predicts only class label (approximate target value) but not probability information. In the following we briefly describe how we extend SVM for probability estimates. More details are in Wu et al. (2004) for classification and in Lin and Weng (2004) for regression.

Given k classes of data, for any $\mathbf{x}$, the goal is to estimate

$$p_i = p(y = i \mid \mathbf{x}), i = 1, \ldots, k.$$

Following the setting of the one-against-one (i.e., pairwise) approach for multi-class classification, we first estimated pairwise class probabilities

$$r_{ij} \approx p(y = i \mid y = i \text{ or } j, \mathbf{x})$$

using an improved implementation (Lin et al., 2003) of (Platt, 2000):

$$r_{ij} \approx \frac{1}{1 + e^{A\hat{f}+B}}, \tag{8.1}$$

where A and B are estimated by minimizing the negative log-likelihood function using known training data and their decision values $\hat{f}$. Labels and decision values are required to be independent so here we conduct *five-fold cross-validation* to obtain decision values.

Then the second approach in Wu et al. (2004) is used to obtain p_i from all these r_{ij}'s. It solves the following optimization problem:

$$\min_{\mathbf{p}} \frac{1}{2} \sum_{i=1}^{k} \sum_{j:j \neq i} (r_{ji}p_i - r_{ij}p_j)^2 \qquad \text{subject to } \sum_{i=1}^{k} p_i = 1, p_i \geq 0, \forall i. \tag{8.2}$$

The objective function comes from the equality

$$p(y = j \mid y = i \text{ or } j, \mathbf{x}) \cdot p(y = i \mid \mathbf{x}) = p(y = i \mid y = i \text{ or } j, \mathbf{x}) \cdot p(y = j \mid \mathbf{x})$$

and can be reformulated as

$$\min_{\mathbf{p}} \frac{1}{2} \mathbf{p}^T Q \mathbf{p}, \tag{8.3}$$

where

$$Q_{ij} = \begin{cases} \sum_{s:s \neq i} r_{si}^2 & \text{if } i = j, \\ -r_{ji}r_{ij} & \text{if } i \neq j. \end{cases} \tag{8.4}$$

This problem is convex, so the optimality conditions that there is a scalar b such that

$$\begin{bmatrix} Q & \mathbf{e} \\ \mathbf{e}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}. \tag{8.5}$$

Here $\mathbf{e}$ is the $k \times 1$ vector of all ones, $\mathbf{0}$ is the $k \times 1$ vector of all zeros, and b is the Lagrangian multiplier of the equality constraint $\sum_{i=1}^{k} p_i = 1$. Instead of directly solving the linear system (8.5), we derive a simple iterative method in the following.

As

$$-\mathbf{p}^T Q\mathbf{p} = -\mathbf{p}^T Q(-bQ^{-1}\mathbf{e}) = b\mathbf{p}^T\mathbf{e} = b,$$

the solution $\mathbf{p}$ satisfies

$$Q_{tt}p_t + \sum_{j:j\neq t} Q_{tj}p_j - \mathbf{p}^T Q\mathbf{p} = 0, \text{ for any } t. \tag{8.6}$$

Using (8.6), we consider the following algorithm:

Algorithm 2

1. Start with some initial $p_i \geq 0, \forall i$ and $\sum_{i=1}^{k} p_i = 1$.

2. Repeat $(t = 1, \ldots, k, 1, \ldots)$

$$p_t \leftarrow \frac{1}{Q_{tt}}[-\sum_{j:j\neq t} Q_{tj}p_j + \mathbf{p}^T Q\mathbf{p}] \tag{8.7}$$

$$\text{normalize } \mathbf{p} \tag{8.8}$$

   until (8.5) is satisfied.

This procedure guarantees to find a global optimum of (8.2). Using some tricks, we do not need to recalculate $\mathbf{p}^T Q\mathbf{p}$ in each iteration. Detailed implementation notes are in Appendix C of Wu et al. (2004). We consider a relative stopping condition for Algorithm 2:

$$\|Q\mathbf{p} - \mathbf{p}^T Q\mathbf{p}\mathbf{e}\|_1 = \max_t |(Q\mathbf{p})_t - \mathbf{p}^T Q\mathbf{p}| < 0.005/k.$$

When k is large, $\mathbf{p}$ will be closer to zero, so we decrease the tolerance by a factor of k.

Next, we discuss SVR probability inference. For a given set of training data $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in R^n, y_i \in R, i = 1, \ldots, l\}$, we suppose that the data are collected from the model:

$$y_i = f(\mathbf{x}_i) + \delta_i, \tag{8.9}$$

where $f(\mathbf{x})$ is the underlying function and δ_i are independent and identically distributed random noises. Given a test data $\mathbf{x}$, the distribution of y given $\mathbf{x}$ and $\mathcal{D}$,

$P(y \mid \mathbf{x}, \mathcal{D})$, allows one to draw probabilistic inferences about y; for example, one can construct a predictive interval $\mathcal{I} = \mathcal{I}(\mathbf{x})$ such that $y \in \mathcal{I}$ with a pre-specified probability. Denoting $\hat{f}$ as the estimated function based on $\mathcal{D}$ using SVR, then $\zeta = \zeta(\mathbf{x}) \equiv y - \hat{f}(\mathbf{x})$ is the out-of-sample residual (or prediction error), and $y \in \mathcal{I}$ is equivalent to $\zeta \in \mathcal{I} - \hat{f}(\mathbf{x})$. We propose to model the distribution of ζ based on a set of out-of-sample residuals $\{\zeta_i\}_{i=1}^{l}$ using training data $\mathcal{D}$. The ζ_i's are generated by first conducting a k-fold cross validation to get $\hat{f}_j$, $j = 1, \ldots, k$, and then setting $\zeta_i \equiv y_i - \hat{f}_j(\mathbf{x}_i)$ for $(\mathbf{x}_i, y_i)$ in the jth fold. It is conceptually clear that the distribution of ζ_i's may resemble that of the prediction error ζ.

Figure 2 illustrates ζ_i's from a real data. Basically, a discretized distribution like histogram can be used to model the data; however, it is complex because all ζ_i's must be retained. On the contrary, distributions like Gaussian and Laplace, commonly used as noise models, require only location and scale parameters. In Figure 2 we plot the fitted curves using these two families and the histogram of ζ_i's. The figure shows that the distribution of ζ_i's seems symmetric about zero and that both Gaussian and Laplace reasonably capture the shape of ζ_i's. Thus, we propose to model ζ_i by zero-mean Gaussian and Laplace, or equivalently, model the conditional distribution of y given $\hat{f}(\mathbf{x})$ by Gaussian and Laplace with mean $\hat{f}(\mathbf{x})$.

(Lin and Weng, 2004) discussed a method to judge whether a Laplace and Gaussian distribution should be used. Moreover, they experimentally show that in all cases they have tried, Laplace is better. Thus, here we consider the zero-mean Laplace with a density function:

$$p(z) = \frac{1}{2\sigma} e^{-\frac{|z|}{\sigma}}. \tag{8.10}$$

Assuming that ζ_i are independent, we can estimate the scale parameter by maximizing the likelihood. For Laplace, the maximum likelihood estimate is

$$\sigma = \frac{\sum_{i=1}^{l} |\zeta_i|}{l}. \tag{8.11}$$

(Lin and Weng, 2004) pointed out that some "very extreme" ζ_i may cause inaccurate estimation of σ. Thus, they propose to estimate the scale parameter by discarding ζ_i's which exceed $\pm 5 \times$ (standard deviation of ζ_i). Thus, for any new data $\mathbf{x}$, we consider that

$$y = \hat{f}(\mathbf{x}) + z,$$

where z is a random variable following the Laplace distribution with parameter σ.

In theory, the distribution of ζ may depend on the input $\mathbf{x}$, but here we assume that it is free of $\mathbf{x}$. This is similar to the model (8.1) for classification. Such an assumption works well in practice and leads to a simple model.

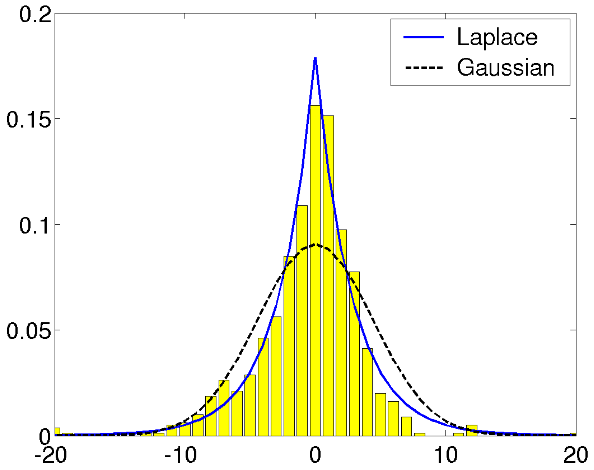


Figure 2: Histogram of ζ_i's from a data set and the modeling via Laplace and Gaussian distributions. The x-axis is ζ_i using five-fold CV and the y-axis is the normalized number of data in each bin of width 1.

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

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

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

P.-H. Chen, R.-E. Fan, and C.-J. Lin. A study on SMO-type decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 17:893–908, July 2006. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/generalSMO.pdf`.

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

D. J. Crisp and C. J. C. Burges. A geometric interpretation of ν-SVM classifiers. In S. Solla, T. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, Cambridge, MA, 2000. MIT Press.

R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf`.

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

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.

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.

S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In J. Fogelman, editor, *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag, 1990.

U. Kreßel. Pairwise classification and support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 255–268, Cambridge, MA, 1999. MIT Press.

C.-J. Lin and R. C. Weng. Simple probabilistic predictions for support vector regression. Technical report, Department of Computer Science, National Taiwan University, 2004. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/svrprob.pdf`.

H.-T. Lin, C.-J. Lin, and R. C. Weng. A note on Platt's probabilistic outputs for support vector machines. Technical report, Department of Computer Science, National Taiwan University, 2003. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/plattprob.ps`.

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, 1997a. IEEE.

E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. AI Memo 1602, Massachusetts Institute of Technology, 1997b.

J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, Cambridge, MA, 2000. MIT Press. URL `citeseer.nj.nec.com/platt99probabilistic.html`.

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.

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

B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

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

T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf`.