LIBSVM: A Library for Support Vector Machines

Chih-Chung Chang and Chih-Jen Lin
Department of Computer Science
National Taiwan University, Taipei, Taiwan
Email: cjlin@csie.ntu.edu.tw

Initial version: 2001 Last updated: May 20, 2011

Abstract

LIBSVM is a library for Support Vector Machines (SVMs). We have been actively developing this package since the year 2000. The goal is to help users to easily apply SVM to their applications. LIBSVM has gained wide popularity in machine learning and many other areas. In this article, we present all implementation details of LIBSVM. Issues such as solving SVM optimization problems, theoretical convergence, multi-class classification, probability estimates, and parameter selection are discussed in detail.

Keywords: Classification, LIBSVM, optimization, regression, support vector machines, SVM

1 Introduction

Support Vector Machines (SVMs) are a popular machine learning method for classification, regression, and other learning tasks. Since the year 2000, we have been developing the package LIBSVM as a library for support vector machines. The Web address of the package is at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. LIBSVM is currently one of the most widely used SVM software. In this article,[1] we present all implementation details of LIBSVM. However, this article does not intend to teach the practical use of LIBSVM. For instructions of using LIBSVM, see the `README` file included in the package, the LIBSVM FAQ,[2] and the practical guide by Hsu et al. (2003). An earlier version of this article was published in Chang and Lin (2011).

LIBSVM supports the following learning tasks.

[1] This LIBSVM implementation document was created in 2001 and has been maintained at `http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf`.

[2] LIBSVM FAQ: `http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html`.

Table 1: Representative works in some domains that have successfully used LIBSVM.

Domain	Representative works
Computer vision	LIBPMK (Grauman and Darrell, 2005)
Natural language processing	Maltparser (Nivre et al., 2007)
Neuroimaging	PyMVPA (Hanke et al., 2009)
Bioinformatics	BDVal (Dorff et al., 2010)

1. SVC: support vector classification (two-class and multi-class).

2. SVR: support vector regression.

3. One-class SVM.

A typical use of LIBSVM involves two steps: first, training a data set to obtain a model and second, using the model to predict information of a testing data set. For SVC and SVR, LIBSVM can also output probability estimates. Many extensions of LIBSVM are available at libsvmtools.[3]

The LIBSVM package is structured as follows.

1. Main directory: core C/C++ programs and sample data. In particular, the file `svm.cpp` implements training and testing algorithms, where details are described in this article.

2. The `tool` sub-directory: this sub-directory includes tools for checking data format and for selecting SVM parameters.

3. Other sub-directories contain pre-built binary files and interfaces to other languages/software.

LIBSVM has been widely used in many areas. From 2000 to 2010, there were more than 250,000 downloads of the package. In this period, we answered more than 10,000 emails from users. Table 1 lists representative works in some domains that have successfully used LIBSVM.

This article is organized as follows. In Section 2, we describe SVM formulations supported 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). Section 3 then discusses performance measures, basic usage, and code organization. All SVM formulations

[3] LIBSVM Tools: `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools`.

supported in LIBSVM are quadratic minimization problems. We discuss the optimization algorithm in Section 4. Section 5 describes two implementation techniques to reduce the running time for minimizing SVM quadratic problems: shrinking and caching. LIBSVM provides some special settings for unbalanced data; details are in Section 6. Section 7 discusses our implementation for multi-class classification. Section 8 presents how to transform SVM decision values into probability values. Parameter selection is important for obtaining good SVM models. Section 9 presents a simple and useful parameter selection tool in LIBSVM. Finally, Section 10 concludes this work.

2 SVM Formulations

LIBSVM supports various SVM formulations for classification, regression, and distribution estimation. In this section, we present these formulations and give corresponding references. We also show performance measures used in LIBSVM.

2.1 C-Support Vector Classification

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

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

where $\phi(\boldsymbol{x}_i)$ maps $\boldsymbol{x}_i$ into a higher-dimensional space and $C > 0$ is the regularization parameter. Due to the possible high dimensionality of the vector variable $\boldsymbol{w}$, usually we solve the following dual problem.

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

where $\boldsymbol{e} = [1, \ldots, 1]^T$ is the vector of all ones, Q is an l by l positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ is the kernel function.

After problem (2) is solved, using the primal-dual relationship, the optimal $\boldsymbol{w}$ satisfies

$$\boldsymbol{w} = \sum_{i=1}^{l} y_i \alpha_i \phi(\boldsymbol{x}_i) \tag{3}$$

and the decision function is

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

We store $y_i\alpha_i \ \forall i$, b, label names,[4] support vectors, and other information such as kernel parameters in the model for prediction.

2.2 ν-Support Vector Classification

The ν-support vector classification (Schölkopf et al., 2000) introduces a new parameter $\nu \in (0, 1]$. It is proved that ν an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.

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

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}, \rho} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \nu\rho + \frac{1}{l}\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b) \geq \rho - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l, \quad \rho \geq 0. \end{aligned} \tag{4}$$

The dual problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq 1/l, \qquad i = 1, \ldots, l, \\ & \boldsymbol{e}^T\boldsymbol{\alpha} \geq \nu, \quad \boldsymbol{y}^T\boldsymbol{\alpha} = 0, \end{aligned} \tag{5}$$

where $Q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Chang and Lin (2001) show that problem (5) is feasible if and only if

$$\nu \leq \frac{2\min(\#y_i = +1, \#y_i = -1)}{l} \leq 1,$$

so the usable range of ν is smaller than $(0, 1]$.

[4] In LIBSVM, any integer can be a label name, so we map label names to ± 1 by assigning the first training instance to have $y_1 = +1$.

The decision function is

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

It is shown that $\boldsymbol{e}^T\boldsymbol{\alpha} \geq \nu$ can be replaced by $\boldsymbol{e}^T\boldsymbol{\alpha} = \nu$ (Crisp and Burges, 2000; Chang and Lin, 2001). In LIBSVM, we solve a scaled version of problem (5) because numerically α_i may be too small due to the constraint $\alpha_i \leq 1/l$.

$$\begin{aligned} \min_{\bar{\boldsymbol{\alpha}}} \quad & \frac{1}{2}\bar{\boldsymbol{\alpha}}^T Q \bar{\boldsymbol{\alpha}} \\ \text{subject to} \quad & 0 \leq \bar{\alpha}_i \leq 1, \qquad i = 1, \ldots, l, \\ & \boldsymbol{e}^T\bar{\boldsymbol{\alpha}} = \nu l, \quad \boldsymbol{y}^T\bar{\boldsymbol{\alpha}} = 0. \end{aligned} \tag{6}$$

If $\boldsymbol{\alpha}$ is optimal for the dual problem (5) and ρ is optimal for the primal problem (4), Chang and Lin (2001) show that $\boldsymbol{\alpha}/\rho$ is an optimal solution of C-SVM with $C = 1/(\rho l)$. Thus, in LIBSVM, we output $(\boldsymbol{\alpha}/\rho, b/\rho)$ in the model.[5]

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 $\boldsymbol{x}_i \in R^n, i = 1, \ldots, l$ without any class information, the primal problem of one-class SVM is

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

The dual problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq 1/(\nu l), i = 1, \ldots, l, \\ & \boldsymbol{e}^T\boldsymbol{\alpha} = 1, \end{aligned} \tag{7}$$

where $Q_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$. The decision function is

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

[5] More precisely, solving (6) obtains $\bar{\rho} = \rho l$. Because $\bar{\boldsymbol{\alpha}} = l\boldsymbol{\alpha}$, we have $\boldsymbol{\alpha}/\rho = \bar{\boldsymbol{\alpha}}/\bar{\rho}$. Hence, in LIBSVM, we calculate $\bar{\boldsymbol{\alpha}}/\bar{\rho}$.

Similar to the case of ν-SVC, in LIBSVM, we solve a scaled version of (7).

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le 1, \qquad i = 1, \ldots, l, \\ & \boldsymbol{e}^T \boldsymbol{\alpha} = \nu l. \end{aligned} \tag{8}$$

2.4 ϵ-Support Vector Regression (ϵ-SVR)

Consider a set of training points, $\{(\boldsymbol{x}_1, z_1), \ldots, (\boldsymbol{x}_l, z_l)\}$, where $\boldsymbol{x}_i \in R^n$ is a feature vector and $z_i \in R^1$ is the target output. Under given parameters $C > 0$ and $\epsilon > 0$, the standard form of support vector regression (Vapnik, 1998) is

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

The dual problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T Q (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \epsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} z_i (\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \boldsymbol{e}^T (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) = 0, \\ & 0 \le \alpha_i, \alpha_i^* \le C, i = 1, \ldots, l, \end{aligned} \tag{9}$$

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

After solving problem (9), the approximate function is

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

In LIBSVM, we output $\boldsymbol{\alpha}^* - \boldsymbol{\alpha}$ in the model.

2.5 ν-Support Vector Regression (ν-SVR)

Similar to ν-SVC, for regression, Schölkopf et al. (2000) use a parameter $\nu \in (0, 1]$ to control the number of support vectors. The parameter ϵ in ϵ-SVR becomes a

parameter here. With (C, ν) as parameters, ν-SVR solves

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

The dual problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T Q(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \boldsymbol{z}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) \\ \text{subject to} \quad & \boldsymbol{e}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) = 0,\ \boldsymbol{e}^T(\boldsymbol{\alpha} + \boldsymbol{\alpha}^*) \le C\nu, \\ & 0 \le \alpha_i, \alpha_i^* \le C/l, \qquad i = 1, \ldots, l. \end{aligned} \tag{10}$$

The approximate function is

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

Similar to ν-SVC, Chang and Lin (2002) show that the inequality $\boldsymbol{e}^T(\boldsymbol{\alpha}+\boldsymbol{\alpha}^*) \le C\nu$ can be replaced by an equality. Moreover, C/l may be too small because users often choose C to be a small constant like one. Thus, in LIBSVM, we treat the user-specified regularization parameter as C/l. That is, $\bar{C} = C/l$ is what users specified and LIBSVM solves the following problem.

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

Chang and Lin (2002) prove that ϵ-SVR with parameters $(\bar{C}, \epsilon)$ has the same solution as ν-SVR with parameters $(l\bar{C}, \nu)$.

3 Performance Measures, Basic Usage, and Code Organization

This section describes LIBSVM's evaluation measures, shows some simple examples of running LIBSVM, and presents the code structure.

3.1 Performance Measures

After solving optimization problems listed in previous sections, users can apply decision functions to predict labels (target values) of testing data. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\bar{l}}$ be the testing data and $f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_{\bar{l}})$ be decision values (target values for regression) predicted by LIBSVM. If the true labels (true target values) of testing data are known and denoted as $y_i, \ldots, y_{\bar{l}}$, we evaluate the prediction results by the following measures.

3.1.1 Classification

$$\begin{array}{rcl} & & \text{Accuracy} \\ & = & \dfrac{\text{\# correctly predicted data}}{\text{\# total testing data}} \times 100\% \end{array}$$

3.1.2 Regression

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

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

3.2 A Simple Example of Running LIBSVM

While detailed instructions of using LIBSVM are available in the `README` file of the package and the practical guide by Hsu et al. (2003), here we give a simple example.

LIBSVM includes a sample data set `heart_scale` of 270 instances. We split the data to a training set `heart_scale.tr` (170 instances) and a testing set `heart_scale.te`.

```
$ python tools/subset.py heart_scale 170 heart_scale.tr heart_scale.te
```

The command `svm-train` solves an SVM optimization problem to produce a model.[6]

```
$ ./svm-train heart_scale.tr
*
optimization finished, #iter = 87
nu = 0.471645
```

[6]The default solver is C-SVC using the RBF kernel (48) with $C = 1$ and $\gamma = 1/n$.

```
obj = -67.299458, rho = 0.203495
nSV = 88, nBSV = 72
Total nSV = 88
```

Next, the command `svm-predict` uses the obtained model to classify the testing set.

```
$ ./svm-predict heart_scale.te heart_scale.tr.model output
Accuracy = 83% (83/100) (classification)
```

The file `output` contains predicted class labels.

3.3 Code Organization

All LIBSVM's training and testing algorithms are implemented in the file `svm.cpp`. The two main sub-routines are `svm_train` and `svm_predict`. The training procedure is more sophisticated, so we give the code organization in Figure 1.

From Figure 1, for classification, `svm_train` decouples a multi-class problem to two-class problems (see Section 7) and calls `svm_train_one` several times. For regression and one-class SVM, it directly calls `svm_train_one`. The probability outputs for classification and regression are also handled in `svm_train`. Then, according to the SVM formulation, `svm_train_one` calls a corresponding sub-routine such as `solve_c_svc` for C-SVC and `solve_nu_svc` for ν-SVC. All `solve_*` sub-routines call the solver `Solve` after preparing suitable input values. The sub-routine `Solve` minimizes a general form of SVM optimization problems; see (11) and (22). Details of the sub-routine `Solve` are described in Sections 4-6.

4 Solving the Quadratic Problems

This section discusses algorithms used in LIBSVM to solve dual quadratic problems listed in Section 2. We split the discussion to two parts. The first part considers optimization problems with one linear constraint, while the second part checks those with two linear constraints.

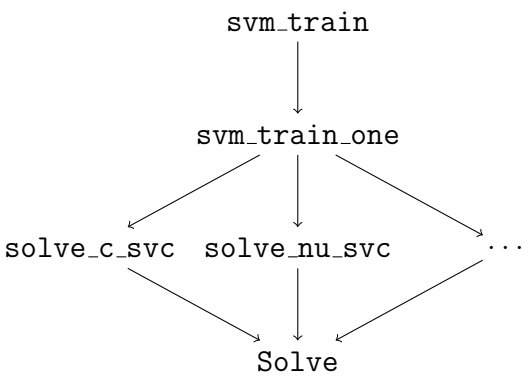


Figure 1: LIBSVM's code organization for training. All sub-routines are in `svm.cpp`.

4.1 Quadratic Problems with One Linear Constraint: C-SVC, ϵ-SVR, and One-class SVM

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

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) \\ \text{subject to} \quad & \boldsymbol{y}^T\boldsymbol{\alpha} = \Delta, \\ & 0 \le \alpha_t \le C, t = 1, \dots, l, \end{aligned} \tag{11}$$

where

$$f(\boldsymbol{\alpha}) \equiv \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \boldsymbol{p}^T\boldsymbol{\alpha}$$

and $y_t = \pm 1, t = 1, \dots, l$. The constraint $\boldsymbol{y}^T\boldsymbol{\alpha} = 0$ is called a *linear* constraint. It can be clearly seen that C-SVC and one-class SVM are already in the form of problem (11). For ϵ-SVR, we use the following reformulation of Eq. (9).

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

where

$$\boldsymbol{y} = [\underbrace{1, \dots, 1}_{l}, \underbrace{-1, \dots, -1}_{l}]^T.$$

We do not assume that Q is positive semi-definite (PSD) because sometimes non-PSD kernel matrices are used.

4.1.1 Decomposition Method for Dual Problems

The main difficulty for solving problem (11) is that Q is a dense matrix and may be too large to be stored. In LIBSVM, we consider a decomposition method to conquer this difficulty. Some earlier works on decomposition methods for SVM include, for example, Osuna et al. (1997a); Joachims (1998); Platt (1998); Keerthi et al. (2001); Hsu and Lin (2002b). Subsequent developments include, for example, Fan et al. (2005); Palagi and Sciandrone (2005); Glasmachers and Igel (2006). A decomposition method modifies only a subset of $\boldsymbol{\alpha}$ per iteration, so only some columns of Q are needed. This subset of variables, denoted as the working set B, leads to a smaller optimization sub-problem. An extreme case of the decomposition methods is the Sequential Minimal Optimization (SMO) (Platt, 1998), which restricts B to have only two elements. Then, at each iteration, we solve a simple two-variable problem without needing any optimization software. LIBSVM considers 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 problem (2), stop. Otherwise, find a *two-element* working set $B = \{i, j\}$ by WSS 1 (described in Section 4.1.2). Define $N \equiv \{1, \ldots, l\} \backslash B$. Let $\boldsymbol{\alpha}_B^k$ and $\boldsymbol{\alpha}_N^k$ be sub-vectors of $\boldsymbol{\alpha}^k$ corresponding to B and N, respectively.

3. If $a_{ij} \equiv K_{ii} + K_{jj} - 2K_{ij} > 0$,[7]

   Solve the following sub-problem with the variable $\boldsymbol{\alpha}_B = [\alpha_i \ \alpha_j]^T$.

$$\begin{aligned} \min_{\alpha_i, \alpha_j} \quad & \frac{1}{2} \begin{bmatrix} \alpha_i & \alpha_j \end{bmatrix} \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + (\boldsymbol{p}_B + Q_{BN} \boldsymbol{\alpha}_N^k)^T \begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} \\ \text{subject to} \quad & 0 \le \alpha_i, \alpha_j \le C, \\ & y_i \alpha_i + y_j \alpha_j = \Delta - \boldsymbol{y}_N^T \boldsymbol{\alpha}_N^k, \end{aligned} \tag{12}$$

   else

[7] We abbreviate $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to K_{ij}.

Let τ be a small positive constant and solve

$$\begin{aligned} \min_{\alpha_i,\alpha_j} \quad & \frac{1}{2}\begin{bmatrix}\alpha_i & \alpha_j\end{bmatrix}\begin{bmatrix}Q_{ii} & Q_{ij}\\ Q_{ij} & Q_{jj}\end{bmatrix}\begin{bmatrix}\alpha_i\\ \alpha_j\end{bmatrix} + (\boldsymbol{p}_B + Q_{BN}\boldsymbol{\alpha}_N^k)^T\begin{bmatrix}\alpha_i\\ \alpha_j\end{bmatrix} \\ & + \frac{\tau - a_{ij}}{4}((\alpha_i - \alpha_i^k)^2 + (\alpha_j - \alpha_j^k)^2) \qquad (13) \\ \text{subject to} \quad & \text{constraints of problem (12).} \end{aligned}$$

4. Set $\boldsymbol{\alpha}_B^{k+1}$ to be the optimal solution of sub-problem (12) or (13), and $\boldsymbol{\alpha}_N^{k+1} \equiv \boldsymbol{\alpha}_N^k$. Set $k \leftarrow k+1$ and go to Step 2.

Note that B is updated at each iteration, but for simplicity, we use B instead of B^k. If Q is PSD, then $a_{ij} > 0$. Thus sub-problem (13) is used only to handle the situation where Q is non-PSD.

4.1.2 Stopping Criteria and Working Set Selection

The Karush-Kuhn-Tucker (KKT) optimality condition of problem (11) implies that a feasible $\boldsymbol{\alpha}$ is a stationary point of (11) if and only if there exists a number b and two nonnegative vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\xi}$ such that

$$\begin{aligned} \nabla f(\boldsymbol{\alpha}) + b\boldsymbol{y} = \boldsymbol{\lambda} - \boldsymbol{\xi}, \\ \lambda_i\alpha_i = 0, \xi_i(C - \alpha_i) = 0, \lambda_i \geq 0, \xi_i \geq 0, i = 1, \ldots, l, \qquad (14) \end{aligned}$$

where $\nabla f(\boldsymbol{\alpha}) \equiv Q\boldsymbol{\alpha} + \boldsymbol{p}$ is the gradient of $f(\boldsymbol{\alpha})$. Note that if Q is PSD, from the primal-dual relationship, $\boldsymbol{\xi}$, b, and $\boldsymbol{w}$ generated by Eq. (3) form an optimal solution of the primal problem. The condition (14) can be rewritten as

$$\nabla_i f(\boldsymbol{\alpha}) + by_i \begin{cases} \geq 0 & \text{if } \alpha_i < C, \\ \leq 0 & \text{if } \alpha_i > 0. \end{cases} \qquad (15)$$

Since $y_i = \pm 1$, condition (15) is equivalent to that there exists b such that

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

where

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

and

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

That is, a feasible $\boldsymbol{\alpha}$ is a stationary point of problem (11) if and only if

$$m(\boldsymbol{\alpha}) \le M(\boldsymbol{\alpha}). \tag{16}$$

From (16), a suitable stopping condition is

$$m(\boldsymbol{\alpha}^k) - M(\boldsymbol{\alpha}^k) \le \epsilon, \tag{17}$$

where ϵ is the tolerance.

For the selection of the working set B, we use the following procedure from Section II of Fan et al. (2005).

WSS 1

1. For all t, s, define

$$a_{ts} \equiv K_{tt} + K_{ss} - 2K_{ts}, \quad b_{ts} \equiv -y_t \nabla_t f(\boldsymbol{\alpha}^k) + y_s \nabla_s f(\boldsymbol{\alpha}^k) > 0, \tag{18}$$

and

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

Select

$$\begin{aligned} i &\in \arg\max_t \{-y_t \nabla_t f(\boldsymbol{\alpha}^k) \mid t \in I_{\text{up}}(\boldsymbol{\alpha}^k)\}, \\ j &\in \arg\min_t \left\{ -\frac{b_{it}^2}{\bar{a}_{it}} \mid t \in I_{\text{low}}(\boldsymbol{\alpha}^k), -y_t \nabla_t f(\boldsymbol{\alpha}^k) < -y_i \nabla_i f(\boldsymbol{\alpha}^k) \right\}. \end{aligned} \tag{19}$$

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

The procedure selects a pair $\{i, j\}$ approximately minimizing the function value; see the term $-b_{it}^2/\bar{a}_{it}$ in Eq. (19).

4.1.3 Solving the Two-variable Sub-problem

Details of solving the two-variable sub-problem in Eqs. (12) and (13) are deferred to Section 6, where a more general sub-problem is discussed.

4.1.4 Maintaining the Gradient

From the discussion in Sections 4.1.1 and 4.1.2, the main operations per iteration are on finding $Q_{BN}\boldsymbol{\alpha}_N^k + \boldsymbol{p}_B$ for constructing the sub-problem (12), and calculating

$\nabla f(\boldsymbol{\alpha}^k)$ for the working set selection and the stopping condition. These two operations can be considered together because

$$Q_{BN}\boldsymbol{\alpha}_N^k + \boldsymbol{p}_B = \nabla_B f(\boldsymbol{\alpha}^k) - Q_{BB}\boldsymbol{\alpha}_B^k \tag{20}$$

and

$$\nabla f(\boldsymbol{\alpha}^{k+1}) = \nabla f(\boldsymbol{\alpha}^k) + Q_{:,B}(\boldsymbol{\alpha}_B^{k+1} - \boldsymbol{\alpha}_B^k), \tag{21}$$

where $|B| \ll |N|$ and $Q_{:,B}$ is the sub-matrix of Q including columns in B. If at the kth iteration we already have $\nabla f(\boldsymbol{\alpha}^k)$, then Eq. (20) can be used to construct the sub-problem. After the sub-problem is solved, Eq. (21) is employed to have the next $\nabla f(\boldsymbol{\alpha}^{k+1})$. Therefore, LIBSVM maintains the gradient throughout the decomposition method.

4.1.5 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.

Note that b of C-SVC and ϵ-SVR plays the same role as $-\rho$ in one-class SVM, so we define $\rho = -b$ and discuss how to find ρ. If there exists α_i such that $0 < \alpha_i < C$, then from the KKT condition (16), $\rho = y_i \nabla_i f(\boldsymbol{\alpha})$. In LIBSVM, for numerical stability, we average all these values.

$$\rho = \frac{\sum_{i:0<\alpha_i<C} y_i \nabla_i f(\boldsymbol{\alpha})}{|\{i \mid 0 < \alpha_i < C\}|}.$$

For the situation that no α_i satisfying $0 < \alpha_i < C$, the KKT condition (16) becomes

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

We take ρ the midpoint of the preceding range.

4.1.6 Initial Values

Algorithm 1 requires an initial feasible $\boldsymbol{\alpha}$. For C-SVC and ϵ-SVR, because the zero vector is feasible, we select it as the initial $\boldsymbol{\alpha}$.

For one-class SVM, the scaled form (8) requires that

$$0 \leq \alpha_i \leq 1, \quad \text{and} \quad \sum_{i=1}^{l} \alpha_i = \nu l.$$

We let the first $\lfloor \nu l \rfloor$ elements have $\alpha_i = 1$ and the $(\lfloor \nu l \rfloor + 1)$st element have $\alpha_i = \nu l - \lfloor \nu l \rfloor$.

4.1.7 Convergence of the Decomposition Method

Fan et al. (2005, Section III) and Chen et al. (2006) discuss the convergence of Algorithm 1 in detail. For the rate of linear convergence, List and Simon (2009) prove a result without making the assumption used in Chen et al. (2006).

4.2 Quadratic Problems with Two Linear Constraints: ν-SVC and ν-SVR

From problems (6) and (10), both ν-SVC and ν-SVR can be written as the following general form.

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \boldsymbol{p}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T \boldsymbol{\alpha} = \Delta_1, \\ & \boldsymbol{e}^T \boldsymbol{\alpha} = \Delta_2, \\ & 0 \le \alpha_t \le C, t = 1, \ldots, l. \end{aligned} \tag{22}$$

The main difference between problems (11) and (22) is that (22) has two linear constraints $\boldsymbol{y}^T\boldsymbol{\alpha} = \Delta_1$ and $\boldsymbol{e}^T\boldsymbol{\alpha} = \Delta_2$. The optimization algorithm is very similar to that for (11), so we describe only differences.

4.2.1 Stopping Criteria and Working Set Selection

Let $f(\boldsymbol{\alpha})$ be the objective function of problem (22). By the same derivation in Section 4.1.2, The KKT condition of problem (22) implies that there exist b and ρ such that

$$\nabla_i f(\boldsymbol{\alpha}) - \rho + b y_i \begin{cases} \ge 0 & \text{if } \alpha_i < C, \\ \le 0 & \text{if } \alpha_i > 0. \end{cases} \tag{23}$$

Define

$$r_1 \equiv \rho - b \text{ and } r_2 \equiv \rho + b. \tag{24}$$

If $y_i = 1$, (23) becomes

$$\nabla_i f(\boldsymbol{\alpha}) - r_1 \begin{cases} \ge 0 & \text{if } \alpha_i < C, \\ \le 0 & \text{if } \alpha_i > 0. \end{cases} \tag{25}$$

if $y_i = -1$, (23) becomes

$$\nabla_i f(\boldsymbol{\alpha}) - r_2 \begin{cases} \geq 0 & \text{if } \alpha_i < C, \\ \leq 0 & \text{if } \alpha_i > 0. \end{cases} \tag{26}$$

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

$$\max\left(m_p(\boldsymbol{\alpha}) - M_p(\boldsymbol{\alpha}), m_n(\boldsymbol{\alpha}) - M_n(\boldsymbol{\alpha})\right) < \epsilon, \tag{27}$$

where

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

The following working set selection is extended from WSS 1.

WSS 2 (Extension of WSS 1 for ν-SVM)

1. Find

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

2. Find

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

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

4.2.2 The Calculation of b and ρ

We have shown that the KKT condition of problem (22) implies Eqs. (25) and (26) according to $y_i = 1$ and -1, respectively. Now we consider the case of $y_i = 1$. If there exists α_i such that $0 < \alpha_i < C$, then we obtain $r_1 = \nabla_i f(\boldsymbol{\alpha})$. In LIBSVM, for numerical stability, we average these values.

$$r_1 = \frac{\sum_{i:0<\alpha_i<C, y_i=1} \nabla_i f(\boldsymbol{\alpha})}{|\{i \mid 0 < \alpha_i < C, y_i = 1\}|}.$$

If there is no α_i such that $0 < \alpha_i < C$, then r_1 satisfies

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

We take r_1 the midpoint of the previous range.

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

After r_1 and r_2 are obtained, from Eq. (24),

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

4.2.3 Initial Values

For ν-SVC, the scaled form (6) requires that

$$0 \le \alpha_i \le 1, \quad \sum_{i:y_i=1} \alpha_i = \frac{\nu l}{2}, \text{ and } \quad \sum_{i:y_i=-1} \alpha_i = \frac{\nu l}{2}.$$

We let the first $\nu l/2$ elements of α_i with $y_i = 1$ to have the value one.[8] The situation for $y_i = -1$ is similar. The same setting is applied to ν-SVR.

5 Shrinking and Caching

This section discusses two implementation tricks (shrinking and caching) for the decomposition method and investigates the computational complexity of Algorithm 1.

5.1 Shrinking

An optimal solution $\boldsymbol{\alpha}$ of the SVM dual problem may contain some bounded elements (i.e., $\alpha_i = 0$ or C). These elements may have already been bounded in the middle of the decomposition iterations. To save the training time, the shrinking technique tries to identify and remove some bounded elements, so a smaller optimization problem is solved (Joachims, 1998). The following theorem theoretically supports the shrinking technique by showing that at the final iterations of Algorithm 1 in Section 4.1.2, only a small set of variables is still changed.

Theorem 5.1 (Theorem IV in Fan et al., 2005) *Consider problem* (11) *and assume Q is positive semi-definite.*

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

$$I \equiv \{i \mid -y_i \nabla_i f(\bar{\boldsymbol{\alpha}}) > M(\bar{\boldsymbol{\alpha}}) \text{ or } -y_i \nabla_i f(\bar{\boldsymbol{\alpha}}) < m(\bar{\boldsymbol{\alpha}})\}.$$

   Further, for every $i \in I$, problem (11) *has a unique and bounded optimal solution at α_i.*

[8] Special care must be made as $\nu l/2$ may not be an integer. See also Section 4.1.6.

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

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

If we denote A as the set containing elements not shrunk at the kth iteration, then instead of solving problem (11), the decomposition method works on a smaller problem.

$$\begin{aligned} \min_{\boldsymbol{\alpha}_A} \quad & \frac{1}{2}\boldsymbol{\alpha}_A^T Q_{AA}\boldsymbol{\alpha}_A + (\boldsymbol{p}_A + Q_{AN}\boldsymbol{\alpha}_N^k)^T\boldsymbol{\alpha}_A \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \quad \forall i \in A, \\ & \boldsymbol{y}_A^T\boldsymbol{\alpha}_A = \Delta - \boldsymbol{y}_N^T\boldsymbol{\alpha}_N^k, \end{aligned} \tag{28}$$

where $N = \{1, \ldots, l\}\backslash A$ is the set of shrunk variables. Note that in LIBSVM, we always rearrange elements of $\boldsymbol{\alpha}, \boldsymbol{y}$, and $\boldsymbol{p}$ to maintain that $A = \{1, \ldots, |A|\}$. Details of the index rearrangement are in Section 5.4.

After solving problem (28), we may find that some elements are wrongly shrunk. When that happens, the original problem (11) is reoptimized from a starting point $\boldsymbol{\alpha} = \left[\begin{smallmatrix}\boldsymbol{\alpha}_A \\ \boldsymbol{\alpha}_N\end{smallmatrix}\right]$, where $\boldsymbol{\alpha}_A$ is optimal for problem (28) and $\boldsymbol{\alpha}_N$ corresponds to shrunk bounded variables.

In LIBSVM, we start the shrinking procedure in an early stage. The procedure is as follows.

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

$$m(\boldsymbol{\alpha}^k) > M(\boldsymbol{\alpha}^k) \tag{29}$$

because the condition (17) is not satisfied yet. Following Theorem 5.1, we conjecture that variables in the following set can be shrunk.

$$\begin{aligned} & \{t \mid -y_t\nabla_t f(\boldsymbol{\alpha}^k) > m(\boldsymbol{\alpha}^k), t \in I_{\text{low}}(\boldsymbol{\alpha}^k), \alpha_t^k \text{ is bounded}\} \cup \\ & \quad \{t \mid -y_t\nabla_t f(\boldsymbol{\alpha}^k) < M(\boldsymbol{\alpha}^k), t \in I_{\text{up}}(\boldsymbol{\alpha}^k), \alpha_t^k \text{ is bounded}\} \\ = & \{t \mid -y_t\nabla_t f(\boldsymbol{\alpha}^k) > m(\boldsymbol{\alpha}^k), \alpha_t^k = C, y_t = 1 \text{ or } \alpha_t^k = 0, y_t = -1\} \cup \\ & \quad \{t \mid -y_t\nabla_t f(\boldsymbol{\alpha}^k) < M(\boldsymbol{\alpha}^k), \alpha_t^k = 0, y_t = 1 \text{ or } \alpha_t^k = C, y_t = -1\}. \end{aligned} \tag{30}$$

Thus, the size of the set A is gradually reduced in every $\min(l, 1000)$ iterations. The problem (28), and the way of calculating $m(\boldsymbol{\alpha}^k)$ and $M(\boldsymbol{\alpha}^k)$ are adjusted accordingly.

2. The preceding shrinking strategy is sometimes too aggressive. Hence, when the decomposition method achieves the following condition for the first time.

$$m(\boldsymbol{\alpha}^k) \leq M(\boldsymbol{\alpha}^k) + 10\epsilon, \tag{31}$$

where ϵ is the specified stopping tolerance, we reconstruct the gradient (details in Section 5.3). Then, the shrinking procedure can be performed based on more accurate information.

3. Once the stopping condition

$$m(\boldsymbol{\alpha}^k) \leq M(\boldsymbol{\alpha}^k) + \epsilon \tag{32}$$

of the smaller problem (28) is reached, we must check if the stopping condition of the original problem (11) has been satisfied. If not, then we reactivate all variables by setting $A = \{1, \ldots, l\}$ and start the same shrinking procedure on the problem (28).

Note that in solving the shrunk problem (28), we only maintain its gradient $Q_{AA}\boldsymbol{\alpha}_A + Q_{AN}\boldsymbol{\alpha}_N + \boldsymbol{p}_A$ (see also Section 4.1.4). Hence, when we reactivate all variables to reoptimize the problem (11), we must reconstruct the whole gradient $\nabla f(\boldsymbol{\alpha})$. Details are discussed in Section 5.3.

For ν-SVC and ν-SVR, because the stopping condition (27) is different from (17), variables being shrunk are different from those in (30). For $y_t = 1$, we shrink elements in the following set

$$\{t \mid -y_t \nabla_t f(\boldsymbol{\alpha}^k) > m_p(\boldsymbol{\alpha}^k), \alpha_t = C, y_t = 1\} \cup$$
$$\{t \mid -y_t \nabla_t f(\boldsymbol{\alpha}^k) < M_p(\boldsymbol{\alpha}^k), \alpha_t = 0, y_t = 1\}.$$

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

$$\{t \mid -y_t \nabla_t f(\boldsymbol{\alpha}^k) > m_n(\boldsymbol{\alpha}^k), \alpha_t = 0, y_t = -1\} \cup$$
$$\{t \mid -y_t \nabla_t f(\boldsymbol{\alpha}^k) < M_n(\boldsymbol{\alpha}^k), \alpha_t = C, y_t = -1\}.$$

5.2 Caching

Caching is an effective technique for reducing the computational time of the decomposition method. Because Q may be too large to be stored in the computer memory, Q_{ij} elements are calculated as needed. We can use available memory (called kernel cache) to store some recently used Q_{ij} (Joachims, 1998). Then, some kernel elements may

not need to be recalculated. Theorem 5.1 also supports the use of caching because in final iterations, only certain columns of the matrix Q are still needed. If the cache already contains these columns, we can save kernel evaluations in final iterations.

In LIBSVM, we consider a simple least-recent-use caching strategy. We use a circular list of structures, where each structure is defined as follows.

```
struct head_t
{
        head_t *prev, *next;    // a circular list
        Qfloat *data;
        int len;                // data[0,len) is cached in this entry
};
```

A structure stores the first `len` elements of a kernel column. Using pointers `prev` and `next`, it is easy to insert or delete a column. The circular list is maintained so that structures are ordered from the least-recent-used one to the most-recent-used one.

Because of shrinking, columns cached in the computer memory may be in different length. Assume the ith column is needed and $Q_{1:t,i}$ have been cached. If $t \le |A|$, we calculate $Q_{t+1:|A|,i}$ and store $Q_{1:|A|,i}$ in the cache. If $t > |A|$, the desired $Q_{1:|A|,i}$ are already in the cache. In this situation, we do not change the cached contents of the ith column.

5.3 Reconstructing the Gradient

If condition (31) or (32) is satisfied, LIBSVM reconstructs the gradient. Because $\nabla_i f(\boldsymbol{\alpha}), i = 1, \ldots, |A|$ have been maintained in solving the smaller problem (28), what we need is to calculate $\nabla_i f(\boldsymbol{\alpha}), i = |A|+1, \ldots, l$. To decrease the cost of this reconstruction, throughout iterations we maintain a vector $\bar{\boldsymbol{G}} \in R^l$.

$$\bar{G}_i = C \sum_{j:\alpha_j = C} Q_{ij}, i = 1, \ldots, l. \tag{33}$$

Then, for $i \notin A$,

$$\nabla_i f(\boldsymbol{\alpha}) = \sum_{j=1}^{l} Q_{ij}\alpha_j + p_i = \bar{G}_i + p_i + \sum_{\substack{j:j\in A \\ 0<\alpha_j<C}} Q_{ij}\alpha_j. \tag{34}$$

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

The calculation of $\nabla f(\boldsymbol{\alpha})$ via Eq. (34) involves 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 the differences next.

1. i first: for $|A| + 1 \le i \le l$, calculate $Q_{i,1:|A|}$. Although from Eq. (34), only $\{Q_{ij} \mid 0 < \alpha_j < C, j \in A\}$ are needed, our implementation obtains all $Q_{i,1:|A|}$ (i.e., $\{Q_{ij} \mid j \in A\}$). Hence, this case needs at most

$$(l - |A|) \cdot |A| \tag{35}$$

   kernel evaluations. Note that LIBSVM uses a column-based caching implementation. Due to the symmetry of Q, $Q_{i,1:|A|}$ is part of Q's ith column and may have been cached. Thus, Eq. (35) is only an upper bound.

2. j first: let

$$F \equiv \{j \mid 1 \le j \le |A| \text{ and } 0 < \alpha_j < C\}.$$

   For each $j \in F$, calculate $Q_{1:l,j}$. Though only $Q_{|A|+1:l,j}$ is needed in calculating $\nabla_i f(\boldsymbol{\alpha}), i = |A| + 1, \ldots, l$, we must get the whole column because of our cache implementation.[9] Thus, this strategy needs no more than

$$l \cdot |F| \tag{36}$$

   kernel evaluations. This is an upper bound because certain kernel columns (e.g., $Q_{1:|A|,j}, j \in A$) may be already in the cache and do not need to be recalculated.

We may choose a method by comparing (35) and (36). 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 because they have been used recently. In contrast, $Q_{i,1:|A|}, i \notin A$ needed by method 1 may be less likely in the cache because columns not in A are not used to solve problem (28). In such a situation, method 1 may require almost $(l - |A|) \cdot |A|$ kernel valuations, while method 2 needs much fewer evaluations than $l \cdot |F|$.

Because method 2 takes an advantage of the cache implementation, we slightly lower the estimate in Eq. (36) and use the following rule to decide the method of calculating Eq. (34):

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

[9] We always store the first $|A|$ elements of a column.

This rule may not give the optimal choice because we do not take the cache contents into account. However, we argue that in the worst scenario, the selected method by the preceding rule is only slightly slower than the other method. This result can be proved by making the following assumptions.

- A LIBSVM training procedure involves *only* two gradient reconstructions. The first is performed when the 10ϵ tolerance is achieved; see Eq. (31). 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 x" as the whole LIBSVM training time (where method x is used for reconstructing gradients), and "reconstruction time of method x" as the time of one single gradient reconstruction via method x. We then consider two situations.

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

   We have

   $$\begin{aligned} & \text{Total time of method 1} \\ \le\ & (\text{Total time of method 2}) + 2 \cdot (\text{Reconstruction time of method 1}) \\ \le\ & 2 \cdot (\text{Total time of method 2}). \end{aligned} \tag{37}$$

   We explain the second inequality in detail. Method 2 for gradient reconstruction requires $l \cdot |F|$ kernel elements; however, the number of kernel evaluations may be smaller because some elements have been cached. Therefore,

   $$l \cdot |F| \le \text{Total time of method 2}. \tag{38}$$

   Because method 1 is chosen and Eq. (35) is an upper bound,

   $$2 \cdot (\text{Reconstruction time of method 1}) \le 2 \cdot (l - |A|) \cdot |A| < l \cdot |F|. \tag{39}$$

   Combining inequalities (38) and (39) leads to (37).

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

   We consider the worst situation where Q's first $|A|$ columns are not in the cache. As $|A|+1, \ldots, l$ are indices of shrunk variables, most likely the remaining $l-|A|$

Table 2: A comparison between two gradient reconstruction methods. The decomposition method reconstructs the gradient twice after satisfying conditions (31) and (32). 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 two rows give the total training time (gradient reconstructions and other operations) in seconds. We use the RBF kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$.

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

$l = 16,100$			Cache = 1,000 MB		Cache = 10 MB	
Reconstruction	$\lvert F\rvert$	$\lvert A\rvert$	Method 1	Method 2	Method 1	Method 2
First	10,597	12,476	0	21,470,526	45,213,024	170,574,272
Second	10,630	12,476	0	0	45,213,024	171,118,048
Training time ⇒			102s	108s	341s	422s
			No shrinking: 111s		No shrinking: 381s	

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

$l = 49,900$			Cache = 1,000 MB		Cache = 10 MB	
Reconstruction	$\lvert F\rvert$	$\lvert A\rvert$	Method 1	Method 2	Method 1	Method 2
First	1,767	43,678	274,297,840	5,403,072	275,695,536	88,332,330
Second	2,308	6,023	263,843,538	28,274,195	264,813,241	115,346,805
Training time ⇒			189s	46s	203s	116s
			No shrinking: 42s		No shrinking: 87s	

columns of Q are not in the cache either and $(l - |A|) \cdot |A|$ kernel evaluations are needed for method 1. Because $l \cdot |F| \le 2 \cdot (l - |A|) \cdot |A|$,

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

Therefore,

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

Table 2 compares the number of kernel evaluations in reconstructing the gradient. We consider problems a7a and ijcnn1.[10] Clearly, the proposed rule selects the better method for both problems. We implement this technique after version 2.88 of LIBSVM.

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

5.4 Index Rearrangement

In solving the smaller problem (28), we need only indices in A (e.g., α_i, y_i, and $\boldsymbol{x}_i$, where $i \in A$). Thus, a naive implementation does not access array contents in a continuous manner. Alternatively, we can maintain $A = \{1, \ldots, |A|\}$ by rearranging array contents. This approach allows a continuous access of array contents, but requires costs for the rearrangement. We decide to rearrange elements in arrays because throughout the discussion in Sections 5.2-5.3, we assume that a cached ith kernel column contains elements from the first to the tth (i.e., $Q_{1:t,i}$), where $t \leq l$. If we do not rearrange indices so that $A = \{1, \ldots, |A|\}$, then the whole column $Q_{1:l,i}$ must be cached because l may be an element in A.

We rearrange indices by sequentially swapping pairs of indices. If t_1 is going to be shrunk, we find an index t_2 that should stay and then swap them. Swapping two elements in a vector $\boldsymbol{\alpha}$ or $\boldsymbol{y}$ is easy, but swapping kernel elements in the cache is more expensive. That is, we must swap $(Q_{t_1,i}, Q_{t_2,i})$ for *every* cached kernel column i. To make the number of swapping operations small, we use the following implementation. Starting from the first and the last indices, we identify the smallest t_1 that should leave the largest t_2 that should stay. Then, (t_1, t_2) are swapped and we continue the same procedure to identify the next pair.

5.5 A Summary of the Shrinking Procedure

We summarize the shrinking procedure in Algorithm 2.

Algorithm 2 (Extending Algorithm 1 to include the shrinking procedure)
Initialization

1. Let $\boldsymbol{\alpha}^1$ be an initial feasible solution.
2. Calculate the initial $\nabla f(\boldsymbol{\alpha}^1)$ and $\bar{\boldsymbol{G}}$ in Eq. (33).
3. Initialize a counter so shrinking is conducted every $\min(l, 1000)$ iterations
4. Let $A = \{1, \ldots, l\}$

For $k = 1, 2, \ldots$

1. Decrease the shrinking counter
2. If the counter is zero, then shrinking is conducted.
   (a) If condition (31) is satisfied for the first time, reconstruct the gradient

(b) Shrink A by removing elements in the set (30). The implementation described in Section 5.4 ensures that $A = \{1, \ldots, |A|\}$.

(c) Reset the shrinking counter

3. If $\boldsymbol{\alpha}_A^k$ satisfies the stopping condition (32)

   (a) Reconstruct the gradient

   (b) If $\boldsymbol{\alpha}^k$ satisfies the stopping condition (32)

   Return $\boldsymbol{\alpha}^k$

   Else

   Reset $A = \{1, \ldots, l\}$ and set the counter to one[11]

4. Find a two-element working set $B = \{i, j\}$ by WSS 1
5. Obtain $Q_{1:|A|,i}$ and $Q_{1:|A|,j}$ from cache or by calculation
6. Solve sub-problem (12) or (13) by procedures in Section 6. Update $\boldsymbol{\alpha}^k$ to $\boldsymbol{\alpha}^{k+1}$
7. Update the gradient by Eq. (21) and update the vector $\bar{\boldsymbol{G}}$

5.6 Is Shrinking Always Better?

We found that if the number of iterations is large, then shrinking can shorten the training time. However, if we loosely solve the optimization problem (e.g., by using a large stopping tolerance ϵ), the code without using shrinking may be much faster. In this situation, because of the small number of iterations, the time spent on all decomposition iterations can be even less than one single gradient reconstruction.

Table 2 compares the total training time with/without shrinking. For a7a, we use the default $\epsilon = 0.001$. Under the parameters $C = 1$ and $\gamma = 4$, the number of iterations is more than 30,000. Then shrinking is useful. However, for ijcnn1, we deliberately use a loose tolerance $\epsilon = 0.5$, so the number of iterations is only around 4,000. Because our shrinking strategy is quite aggressive, before the first gradient reconstruction, only $Q_{A,A}$ is in the cache. Then, we need many kernel evaluations for reconstructing the gradient, so the implementation with shrinking is slower.

If enough iterations have been run, most elements in A correspond to free α_i ($0 < \alpha_i < C$); i.e., $A \approx F$. In contrast, if the number of iterations is small (e.g., ijcnn1 in Table 2), many bounded elements have not been shrunk and $|F| \ll |A|$. Therefore, we can check the relation between $|F|$ and $|A|$ to conjecture if shrinking

[11] That is, shrinking is performed at the next iteration.

is 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.

5.7 Computational Complexity

While Section 4.1.7 has discussed the asymptotic convergence and the local convergence rate of the decomposition method, in this section, we investigate the computational complexity.

From Section 4, two places consume most operations at each iteration: finding the working set B by WSS 1 and calculating $Q_{:,B}(\boldsymbol{\alpha}_B^{k+1} - \boldsymbol{\alpha}_B^k)$ in Eq. (21).[12] Each place requires $O(l)$ operations. However, if $Q_{:,B}$ is not available in the cache and assume each kernel evaluation costs $O(n)$, the cost becomes $O(ln)$ for calculating a column of kernel elements. Therefore, the complexity of Algorithm 1 is

1. #Iterations $\times O(l)$ if most columns of Q are cached throughout iterations.
2. #Iterations $\times O(nl)$ if columns of Q are not cached and each kernel evaluation costs $O(n)$.

Several works have studied the number of iterations of decomposition methods; see, for example, List and Simon (2007). However, algorithms studied in these works are slightly different from LIBSVM, so there is no theoretical result yet on LIBSVM's number of iterations. Empirically, it is known that the number of iterations may be higher than linear to the number of training data. Thus, LIBSVM may take considerable training time for huge data sets. Many techniques, for example, Fine and Scheinberg (2001); Lee and Mangasarian (2001); Keerthi et al. (2006); Segata and Blanzieri (2010), have been developed to obtain an approximate model, but these are beyond the scope of our discussion. In LIBSVM, we provide a simple sub-sampling tool, so users can quickly train a small subset.

6 Unbalanced Data and Solving the Two-variable Sub-problem

For some classification problems, numbers of data in different classes are unbalanced. Some researchers (e.g., Osuna et al., 1997b, Section 2.5; Vapnik, 1998, Chapter 10.9)

[12] Note that because $|B| = 2$, once the sub-problem has been constructed, solving it takes only a constant number of operations (see details in Section 6).

have proposed using different penalty parameters in the SVM formulation. For example, the C-SVM problem becomes

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C^+\sum_{y_i=1}\xi_i + C^-\sum_{y_i=-1}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i)+b) \geq 1-\xi_i, \\ & \xi_i \geq 0, i=1,\ldots,l, \end{aligned} \tag{40}$$

where C^+ and C^- are regularization parameters for positive and negative classes, respectively. LIBSVM supports this setting, so users can choose weights for classes. The dual problem of problem (40) is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C^+, \text{ if } y_i = 1, \\ & 0 \leq \alpha_i \leq C^-, \text{ if } y_i = -1, \\ & \boldsymbol{y}^T\boldsymbol{\alpha} = 0. \end{aligned}$$

A more general setting is to assign each instance $\boldsymbol{x}_i$ a regularization parameter C_i. If C is replaced by $C_i, i = 1,\ldots,l$ in problem (11), most results discussed in earlier sections can be extended without problems.[13] The major change of Algorithm 1 is on solving the sub-problem (12), which now becomes

$$\begin{aligned} \min_{\alpha_i,\alpha_j} \quad & \frac{1}{2}\begin{bmatrix}\alpha_i & \alpha_j\end{bmatrix}\begin{bmatrix}Q_{ii} & Q_{ij}\\ Q_{ji} & Q_{jj}\end{bmatrix}\begin{bmatrix}\alpha_i\\ \alpha_j\end{bmatrix} + (Q_{i,N}\boldsymbol{\alpha}_N + p_i)\alpha_i + (Q_{j,N}\boldsymbol{\alpha}_N + p_j)\alpha_j \\ \text{subject to} \quad & y_i\alpha_i + y_j\alpha_j = \Delta - \boldsymbol{y}_N^T\boldsymbol{\alpha}_N^k, \\ & 0 \leq \alpha_i \leq C_i, 0 \leq \alpha_j \leq C_j. \end{aligned} \tag{41}$$

Let $\alpha_i = \alpha_i^k + d_i$ and $\alpha_j = \alpha_j^k + d_j$. The sub-problem (41) can be written as

$$\begin{aligned} \min_{d_i,d_j} \quad & \frac{1}{2}\begin{bmatrix}d_i & d_j\end{bmatrix}\begin{bmatrix}Q_{ii} & Q_{ij}\\ Q_{ij} & Q_{jj}\end{bmatrix}\begin{bmatrix}d_i\\ d_j\end{bmatrix} + \begin{bmatrix}\nabla_i f(\boldsymbol{\alpha}^k) & \nabla_j f(\boldsymbol{\alpha}^k)\end{bmatrix}\begin{bmatrix}d_i\\ d_j\end{bmatrix} \\ \text{subject to} \quad & y_i d_i + y_j d_j = 0, \\ & -\alpha_i^k \leq d_i \leq C_i - \alpha_i^k, -\alpha_j^k \leq d_j \leq C_j - \alpha_j^k. \end{aligned}$$

Define a_{ij} and b_{ij} as in Eq. (18), and $\hat{d}_i \equiv y_i d_i, \hat{d}_j \equiv y_j d_j$. Using $\hat{d}_i = -\hat{d}_j$, the objective function can be written as

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

[13] This feature of using $C_i, \forall i$ is not included in LIBSVM, but is available as an extension at libsvmtools.

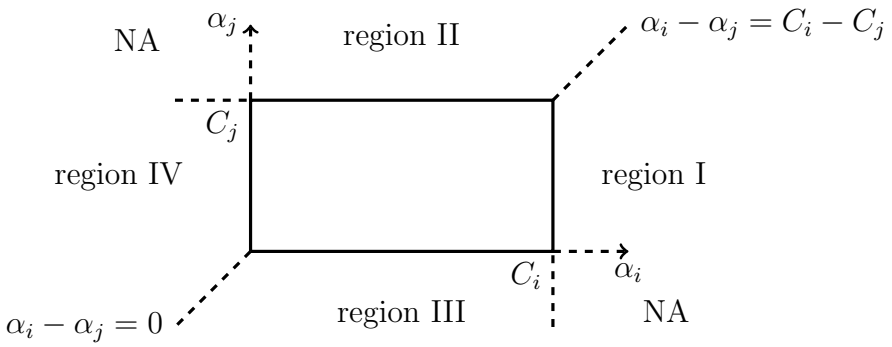
Minimizing the previous quadratic function leads to

$$\begin{aligned}\alpha_i^{\text{new}} &= \alpha_i^k + y_i b_{ij}/\bar{a}_{ij},\\ \alpha_j^{\text{new}} &= \alpha_j^k - y_j b_{ij}/\bar{a}_{ij}.\end{aligned} \tag{42}$$

These two values may need to be modified because of bound constraints. We first consider the case of $y_i \neq y_j$ and re-write Eq. (42) as

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

In the following figure, a box is generated according to bound constraints. An infeasible $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ must be in one of the four regions outside the following box.



Note that $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ does not appear in the "NA" regions because (α_i^k, α_j^k) is in the box and

$$\alpha_i^{\text{new}} - \alpha_j^{\text{new}} = \alpha_i^k - \alpha_j^k.$$

If $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ is in region I, we set

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

Of course, we must identify the region that $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ resides. For region I, 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. We have the following pseudo code to identify which region $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ is in and modify $(\alpha_i^{\text{new}}, \alpha_j^{\text{new}})$ to satisfy bound constraints.

```
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;
                }
        }
}
```

If $y_i = y_j$, the derivation is the same.

7 Multi-class classification

LIBSVM implements the "one-against-one" approach (Knerr et al., 1990) for multi-class classification. Some early works of applying this strategy to SVM include, for example, Kressel (1998). If k is the number of classes, then $k(k-1)/2$ classifiers are constructed and each one trains data from two classes. For training data from the

ith and the jth classes, we solve the following two-class classification problem.

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

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

In case that two classes have identical votes, though it may not be a good strategy, now we simply choose the class appearing first in the array of storing class names.

Many other methods are available for multi-class SVM classification. Hsu and Lin (2002a) give a detailed comparison and conclude that "one-against-one" is a competitive approach.

8 Probability Estimates

SVM predicts only class label (target value for regression) without probability information. This section discusses the LIBSVM implementation for extending SVM to give 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 $\boldsymbol{x}$, the goal is to estimate

$$p_i = P(y = i \mid \boldsymbol{x}), \quad i = 1, \ldots, k.$$

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

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

using an improved implementation (Lin et al., 2007) of Platt (2000). If $\hat{f}$ is the decision value at $\boldsymbol{x}$, then we assume

$$r_{ij} \approx \frac{1}{1 + e^{A\hat{f}+B}}, \tag{43}$$

where A and B are estimated by minimizing the negative log likelihood of training data (using their labels and decision values). It has been observed that decision values

from training may overfit the model (43), so we conduct *five-fold cross-validation* to obtain decision values before minimizing the negative log likelihood.

After collecting all r_{ij} values, Wu et al. (2004) propose several approaches to obtain $p_i, \forall i$. In LIBSVM, we consider their second approach and solve the following optimization problem.

$$\begin{aligned} \min_{\boldsymbol{p}} \quad & \frac{1}{2}\sum_{i=1}^{k}\sum_{j:j\neq i}(r_{ji}p_i - r_{ij}p_j)^2 \\ \text{subject to} \quad & p_i \geq 0, \forall i, \quad \sum_{i=1}^{k} p_i = 1. \end{aligned} \tag{44}$$

The objective function in problem (44) comes from the equality

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

and can be reformulated as

$$\min_{\boldsymbol{p}} \quad \frac{1}{2}\boldsymbol{p}^T Q \boldsymbol{p},$$

where

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

Wu et al. (2004) prove that the non-negativity constraints $p_i \geq 0, \forall i$ in problem (44) are redundant. After removing these constraints, the optimality condition implies that there exists a scalar b (the Lagrange multiplier of the equality constraint $\sum_{i=1}^{k} p_i = 1$) such that

$$\begin{bmatrix} Q & \boldsymbol{e} \\ \boldsymbol{e}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix}, \tag{45}$$

where $\boldsymbol{e}$ is the $k \times 1$ vector of all ones and $\boldsymbol{0}$ is the $k \times 1$ vector of all zeros.

Instead of solving the linear system (45) by a direct method such as Gaussian elimination, Wu et al. (2004) derive a simple iterative method. Because

$$-\boldsymbol{p}^T Q \boldsymbol{p} = -\boldsymbol{p}^T Q(-b\boldsymbol{e}) = b\boldsymbol{p}^T \boldsymbol{e} = b,$$

the optimal solution $\boldsymbol{p}$ satisfies

$$(Q\boldsymbol{p})_t - \boldsymbol{p}^T Q \boldsymbol{p} = Q_{tt}p_t + \sum_{j:j\neq t} Q_{tj}p_j - \boldsymbol{p}^T Q \boldsymbol{p} = 0, \quad \forall t. \tag{46}$$

Using Eq. (46), we consider Algorithm 3.

Algorithm 3

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

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

$$p_t \leftarrow \frac{1}{Q_{tt}}[-\sum_{j:j\neq t} Q_{tj}p_j + \boldsymbol{p}^T Q\boldsymbol{p}] \tag{47}$$
$$\text{normalize } \boldsymbol{p}$$

   until Eq. (45) is satisfied.

Eq. (47) can be simplified to

$$p_t \leftarrow p_t + \frac{1}{Q_{tt}}[-(Q\boldsymbol{p})_t + \boldsymbol{p}^T Q\boldsymbol{p}].$$

Algorithm 3 guarantees to converge globally to the unique optimum of problem (44). Using some tricks, we do not need to recalculate $\boldsymbol{p}^T Q\boldsymbol{p}$ at each iteration. More implementation details are in Appendix C of Wu et al. (2004). We consider a relative stopping condition for Algorithm 3.

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

When k (the number of classes) is large, some elements of $\boldsymbol{p}$ may be very close to zero. Thus, we use a more strict stopping condition by decreasing the tolerance by a factor of k.

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

$$y_i = f(\boldsymbol{x}_i) + \delta_i,$$

where $f(\boldsymbol{x})$ is the underlying function and δ_i's are independent and identically distributed random noises. Given a test data $\boldsymbol{x}$, the distribution of y given $\boldsymbol{x}$ and $\mathcal{D}$, $P(y \mid \boldsymbol{x}, \mathcal{D})$, allows us to draw probabilistic inferences about y; for example, we can estimate the probability that y is in an interval such as $[f(\boldsymbol{x})-\Delta, f(\boldsymbol{x})+\Delta]$. Denoting $\hat{f}$ as the estimated function based on $\mathcal{D}$ using SVR, then $\zeta = \zeta(\boldsymbol{x}) \equiv y - \hat{f}(\boldsymbol{x})$ is the out-of-sample residual (or prediction error). We propose modeling the distribution of ζ based on cross-validation residuals $\{\zeta_i\}_{i=1}^{l}$. The ζ_i's are generated by first conducting a five-fold cross-validation to get $\hat{f}_j$, $j = 1, \ldots, 5$, and then setting $\zeta_i \equiv y_i - \hat{f}_j(\boldsymbol{x}_i)$ for $(\boldsymbol{x}_i, y_i)$ 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 data set. 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 $\hat{f}(\boldsymbol{x})$ by Gaussian and Laplace with mean $\hat{f}(\boldsymbol{x})$.

Lin and Weng (2004) discuss 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, in LIBSVM, we consider the zero-mean Laplace with a density function.

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

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

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

Lin and Weng (2004) point out that some "very extreme" ζ_i's may cause inaccurate estimation of σ. Thus, they propose estimating the scale parameter by discarding ζ_i's which exceed $\pm 5 \cdot$ (standard deviation of the Laplace distribution). For any new data $\boldsymbol{x}$, we consider that

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

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

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

9 Parameter Selection

To train SVM problems, users must specify some parameters. LIBSVM provides a simple tool to check a grid of parameters. For each parameter setting, LIBSVM obtains cross-validation (CV) accuracy. Finally, the parameters with the highest CV accuracy

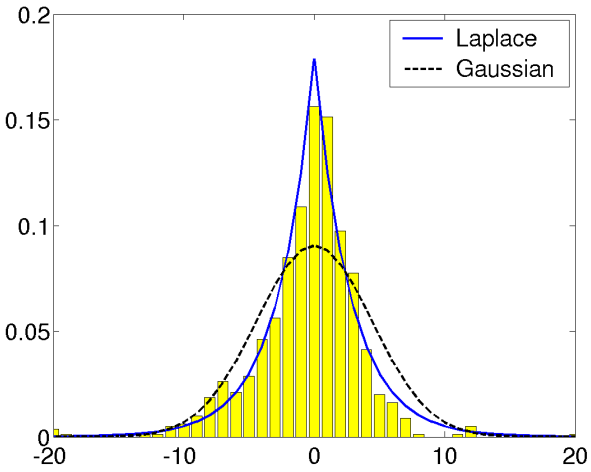


Figure 2: Histogram of ζ_i's and the models via Laplace and Gaussian distributions. The x-axis is ζ_i using five-fold cross-validation and the y-axis is the normalized number of data in each bin of width 1.

are returned. The parameter selection tool assumes that the RBF (Gaussian) kernel is used although extensions to other kernels and SVR can be easily made. The RBF kernel takes the form

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}, \tag{48}$$

so (C, γ) are parameters to be decided. Users can provide a possible interval of C (or γ) with the grid space. Then, all grid points of (C, γ) are tried to find the one giving the highest CV accuracy. Users then use the best parameters to train the whole training set and generate the final model.

We do not consider more advanced parameter selection methods because for only two parameters (C and γ), the number of grid points is not too large. Further, because SVM problems under different (C, γ) parameters are independent, LIBSVM provides a simple tool so that jobs can be run in a parallel (multi-core, shared memory, or distributed) environment.

For multi-class classification, under a given (C, γ), LIBSVM uses the one-against-one method to obtain the CV accuracy. Hence, the parameter selection tool suggests the same (C, γ) for all $k(k-1)/2$ decision functions. Chen et al. (2005, Section 8) discuss issues of using the same or different parameters for the $k(k-1)/2$ two-class problems.

LIBSVM outputs the contour plot of cross-validation accuracy. An example is in

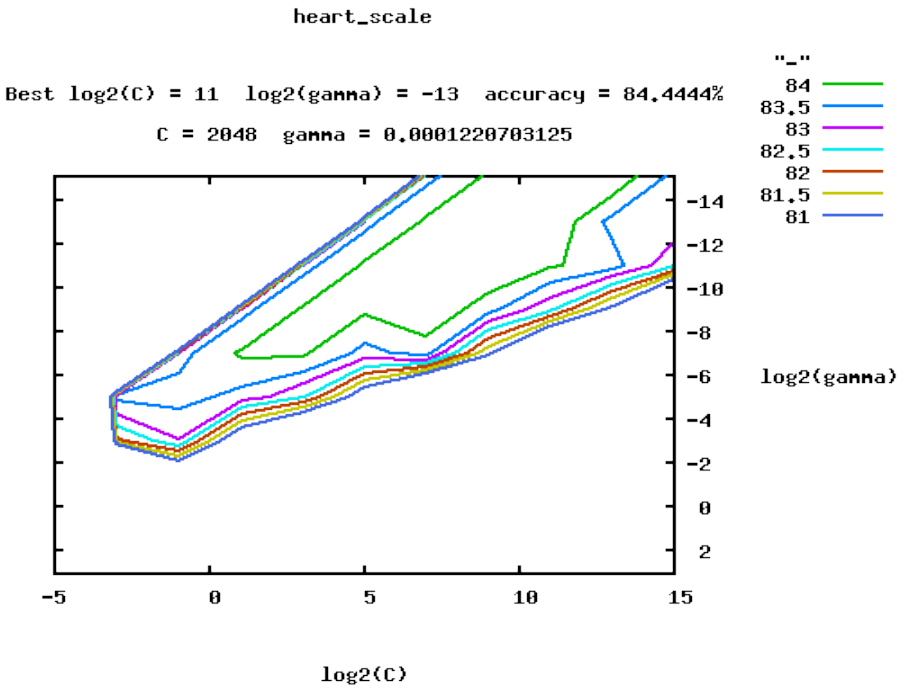


Figure 3: Contour plot of running the parameter selection tool in LIBSVM. The data set heart_scale (included in the package) is used. The x-axis is $\log_2 C$ and the y-axis is $\log_2 \gamma$.

Figure 3.

10 Conclusions

When we released the first version of LIBSVM in 2000, only two-class C-SVC was supported. Gradually, we added other SVM variants, and supported functions such as multi-class classification and probability estimates. Then, LIBSVM becomes a complete SVM package. We add a function only if it is needed by enough users. By keeping the system simple, we strive to ensure good system reliability.

In summary, this article gives implementation details of LIBSVM. We are still actively updating and maintaining this package. We hope the community will benefit more from our continuing development of LIBSVM.

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 their group members and users for many helpful comments. A list of acknowledgments is at `http://www.csie.ntu.edu.tw/~cjlin/libsvm/acknowledgements`.

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.

C.-C. Chang and C.-J. Lin. Training ν-support vector regression: Theory and algorithms. *Neural Computation*, 14(8):1959–1977, 2002.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

P.-H. Chen, C.-J. Lin, and B. Schölkopf. A tutorial on ν-support vector machines. *Applied Stochastic Models in Business and Industry*, 21:111–136, 2005. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/nusvmtoturial.pdf`.

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.

K. C. Dorff, N. Chambwe, M. Srdanovic, and F. Campagne. BDVal: reproducible large-scale predictive model development and validation in high-throughput datasets. *Bioinformatics*, 26(19):2472–2473, 2010.

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`.

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

T. Glasmachers and C. Igel. Maximum-gain working set selection for support vector machines. *Journal of Machine Learning Research*, 7:1437–1466, 2006.

K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Proceedings of IEEE International Conference on Computer Vision*, 2005.

M. Hanke, Y. O. Halchenko, P. B. Sederberg, S. J. Hanson, J. V. Haxby, and S. Pollmann. PyMVPA: A Python toolbox for multivariate pattern analysis of fMRI data. *Neuroinformatics*, 7(1):37–53, 2009. ISSN 1539-2791.

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, 2002a.

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

C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

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*, pages 169–184, Cambridge, MA, 1998. MIT Press.

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

S. S. Keerthi, O. Chapelle, and D. DeCoste. Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research*, 7:1493–1515, 2006.

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. H.-G. 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, 1998. MIT Press.

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

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. *Machine Learning*, 68:267–276, 2007. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/plattprob.pdf`.

N. List and H. U. Simon. General polynomial time decomposition algorithms. *Journal of Machine Learning Research*, 8:303–321, 2007.

N. List and H. U. Simon. SVM-optimization and steepest-descent line search. In *Proceedings of the 22nd Annual Conference on Computational Learning Theory*, 2009.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kubler, S. Marinov, and E. Marsi. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 130–136, 1997a.

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

L. Palagi and M. Sciandrone. On the convergence of a modified version of SVM^{light} algorithm. *Optimization Methods and Software*, 20(2–3):315–332, 2005.

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.

J. C. 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.

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.

N. Segata and E. Blanzieri. Fast and scalable local kernel machines. *Journal of Machine Learning Research*, 11:1883–1926, 2010.

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`.