A Comparison of Methods for Multi-class Support Vector Machines

Chih-Wei Hsu and Chih-Jen Lin

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

Abstract

Support vector machines (SVM) were originally designed for binary classification. How to effectively extend it for multi-class classification is still an on-going research issue. Several methods have been proposed where typically we construct a multi-class classifier by combining several binary classifiers. Some authors also proposed methods that consider all classes at once. As it is computationally more expensive to solve multi-class problems, comparisons of these methods using large-scale problems have not been seriously conducted. Especially for methods solving multi-class SVM in one step, a much larger optimization problem is required so up to now experiments are limited to small data sets. In this paper we give decomposition implementations for two such "all-together" methods: [25], [27] and [7]. We then compare their performance with three methods based on binary classifications: "one-against-all," "one-against-one," and DAGSVM [23]. Our experiments indicate that the "one-against-one" and DAG methods are more suitable for practical use than the other methods. Results also show that for large problems methods by considering all data at once in general need fewer support vectors.

Keywords

Support vector machines, decomposition methods, multi-class classification.

I. Introduction

Support Vector Machines (SVM) [6] were originally designed for binary classification. How to effectively extend it for multi-class classification is still an on-going research issue. Currently there are two types of approaches for multi-class SVM. One is by constructing and combining several binary classifiers while the other is by directly considering all data in one optimization formulation. Up to now there are still no comparisons which cover most of these methods.

The formulation to solve multi-class SVM problems in one step has variables proportional to the number of classes. Therefore, for multi-class SVM methods, either several binary classifiers have to be constructed or a larger optimization problem is needed. Hence in general it is computationally more expensive to solve a multi-class problem than a binary problem with the same number of data. Up to now experiments are limited to small data sets. In this paper we will give a decomposition implementation for two such "all-together" methods: [25], [27] and [7]. We then compare their performance with three methods based on binary classification: "one-against-all," "one-against-one," and DAGSVM [23].

Note that it was pointed out in [11] that the primal forms proposed in [25], [27] are also equivalent to those in [3], [11]. Besides methods mentioned above, there are other implementations for multi-class SVM. For example, [14], [19]. However, due to the limit of space here we do not conduct experiments on them. An earlier comparison between one-against-one and one-against-all methods is in [5].

In Section II, we review one-against-all, one-against-one, and DAGSVM methods which are based on solving several binary classifications. In Section III, we give a brief introduction to the method in [25], [27] which considers all classes at once and show that the decomposition method proposed in [12] can be applied. Another method which also considers all variables together is by Crammer and Singer [7], which will be discussed in Section IV. Numerical experiments are in Section V where we show that "one-against-one" and DAG methods are more suitable for practical use than the other methods. Results also show that for large problems the method proposed in [25], [27] by considering all variables at once generally needs fewer support vectors. Finally we have some discussions and conclusions in Section VI.

II. One-against-all, One-against-one, and DAGSVM Methods

The earliest used implementation for SVM multi-class classification is probably the one-against-all method (for example, [2]). It constructs k SVM models where k is the number of classes. The ith SVM is trained with all of the examples in the ith class with positive labels, and all other examples with negative labels. Thus given l training data $(x_1, y_1), \ldots, (x_l, y_l)$, where $x_i \in R^n, i = 1, \ldots, l$ and

$y_i \in \{1, \dots, k\}$ is the class of x_i, the ith SVM solves the following problem:

$$\begin{aligned} \min_{w^i, b^i, \xi^i} \quad & \frac{1}{2}(w^i)^T w^i + C \sum_{j=1}^{l} \xi_j^i \\ & (w^i)^T \phi(x_j) + b^i \geq 1 - \xi_j^i, \text{ if } y_j = i, \\ & (w^i)^T \phi(x_j) + b^i \leq -1 + \xi_j^i, \text{ if } y_j \neq i, \\ & \xi_j^i \geq 0, \; j = 1, \dots, l, \end{aligned} \tag{1}$$

where the training data x_i are mapped to a higher dimensional space by the function ϕ and C is the penalty parameter.

Minimizing $\frac{1}{2}(w^i)^T w^i$ means that we would like to maximize $2/\|w^i\|$, the margin between two groups of data. When data are not linear separable, there is a penalty term $C \sum_{j=1}^{l} \xi_j^i$ which can reduce the number of training errors. The basic concept behind SVM is to search for a balance between the regularization term $\frac{1}{2}(w^i)^T w^i$ and the training errors.

After solving (1), there are k decision functions:

$$\begin{gathered} (w^1)^T \phi(x) + b^1, \\ \vdots \\ (w^k)^T \phi(x) + b^k. \end{gathered}$$

We say x is in the class which has the largest value of the decision function:

$$\text{class of } x \equiv \text{argmax}_{i=1,\dots,k}((w^i)^T \phi(x) + b^i). \tag{2}$$

Practically we solve the dual problem of (1) whose number of variables is the same as the number of data in (1). Hence k l-variable quadratic programming problems are solved.

Another major method is called the one-against-one method. It was introduced in [15], and the first use of this strategy on SVM was in [9], [16]. This method constructs $k(k-1)/2$ classifiers where each one is trained on data from two classes. For training data from the ith and the jth classes, we solve the following binary classification problem:

$$\begin{aligned} \min_{w^{ij}, b^{ij}, \xi^{ij}} \quad & \frac{1}{2}(w^{ij})^T w^{ij} + C \sum_{t} \xi_t^{ij} \\ & (w^{ij})^T \phi(x_t) + b^{ij} \geq 1 - \xi_t^{ij}, \text{ if } y_t = i, \\ & (w^{ij})^T \phi(x_t) + b^{ij} \leq -1 + \xi_t^{ij}, \text{ if } y_t = j, \\ & \xi_t^{ij} \geq 0. \end{aligned} \tag{3}$$

There are different methods for doing the future testing after all $k(k-1)/2$ classifiers are constructed. After some tests, we decide to use the following voting strategy suggested in [9]: if $\text{sign}((w^{ij})^T\phi(x)+b^{ij}))$ says x is in the ith class, then the vote for the ith class is added by one. Otherwise, the jth is increased by one. Then we predict x is in the class with the largest vote. The voting approach described above is also called the "Max Wins" strategy. In case that two classes have identical votes, thought it may not be a good strategy, now we simply select the one with the smaller index.

Practically we solve the dual of (3) whose number of variables is the same as the number of data in two classes. Hence if in average each class has l/k data points, we have to solve $k(k-1)/2$ quadratic programming problems where each of them has about $2l/k$ variables.

The third algorithm discussed here is the Directed Acyclic Graph Support Vector Machines (DAGSVM) proposed in [23]. Its training phase is the same as the one-against-one method by solving $k(k-1)/2$ binary SVMs. However, in the testing phase, it uses a rooted binary directed acyclic graph which has $k(k-1)/2$ internal nodes and k leaves. Each node is a binary SVM of ith and jth classes. Given a test sample x, starting at the root node, the binary decision function is evaluated. Then it moves to either left or right depending on the output value. Therefore, we go through a path before reaching a leaf node which indicates the predicted class.

An advantage of using a DAG is that [23] some analysis of generalization can be established. There are still no similar theoretical results for one-against-all and one-against-one methods yet. In addition, its testing time is less than the one-against-one method.

We have implemented all three methods by modifying our SVM software LIBSVM [4].

III. A Method by Considering All Data at Once And A Decomposition Implementation

In [25], [27], an approach for multi-class problems by solving one single optimization problem was proposed. The idea is similar to the one-against-all approach. It constructs k two-class rules where the mth function $w_m^T\phi(x)+b$ separates training vectors of the class m from the other vectors. Hence there are k decision functions but all are obtained by solving one problem. The formulation is as follows:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\sum_{m=1}^{k} w_m^T w_m + C\sum_{i=1}^{l}\sum_{m\neq y_i} \xi_i^m \\ & w_{y_i}^T\phi(x_i) + b_{y_i} \geq w_m^T\phi(x_i) + b_m + 2 - \xi_i^m, \qquad (4) \\ & \xi_i^m \geq 0, i = 1,\dots,l,\ m \in \{1,\dots,k\}\backslash y_i. \end{aligned}$$

Then the decision function is

$$\text{argmax}_{m=1,\dots,k}(w_m^T\phi(x) + b_m),$$

which is the same as (2) of the one-against-all method. Like binary SVM, it is easier to solve the dual problem here. Following [27], the dual formulation of (4) is

$$\begin{aligned} \min_{\alpha} \quad & \sum_{i,j}(\frac{1}{2}c_j^{y_i}A_iA_j - \sum_m \alpha_i^m\alpha_j^{y_i} + \frac{1}{2}\sum_m \alpha_i^m\alpha_j^m)K_{i,j} - 2\sum_{i,m}\alpha_i^m \\ & \sum_{i=1}^{l}\alpha_i^m = \sum_{i=1}^{l} c_i^m A_i, m = 1,\dots,k, \qquad (5a) \\ & 0 \leq \alpha_i^m \leq C,\ \alpha_i^{y_i} = 0, \qquad (5b) \\ & A_i = \sum_{m=1}^{k}\alpha_i^m,\ c_j^{y_i} = \begin{cases} 1 & \text{if } y_i = y_j, \\ 0 & \text{if } y_i \neq y_j, \end{cases} \qquad (5c) \\ & i = 1,\dots,l, m = 1,\dots,k, \end{aligned}$$

where $K_{i,j} = \phi(x_i)^T\phi(x_j)$. Then

$$w_m = \sum_{i=1}^{l}(c_i^m A_i - \alpha_i^m)\phi(x_i), m = 1,\dots,k \qquad (6)$$

and the decision function is

$$\text{argmax}_{m=1,\dots,k}(\sum_{i=1}^{l}(c_i^m A_i - \alpha_i^m)K(x_i,x) + b_m).$$

Next we explain that (4) is equivalent to two formulations where one is from

[11]:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\sum_{o<m}^{k} \|w_m - w_o\|^2 + C\sum_{i=1}^{l}\sum_{m\neq y_i} \xi_i^m \\ & w_{y_i}^T\phi(x_i) + b_{y_i} \geq w_m^T\phi(x_i) + b_m + 2 - \xi_i^m, && \text{(7a)} \\ & \sum_{m=1}^{k} w_m = 0, && \text{(7b)} \\ & \xi_i^m \geq 0, i = 1, \ldots, l, \; m \in \{1, \ldots, k\}\backslash y_i, \end{aligned}$$

and the other is from [3]:

$$\min \frac{1}{2}\sum_{o<m}^{k} \|w_m - w_o\|^2 + \sum_{m=1}^{k} w_m^T w_m + C\sum_{i=1}^{l}\sum_{m\neq y_i} \xi_i^m \tag{8}$$

with the same constraints of (4).

For the optimal solution of (4), from (6) and (5c), we have

$$\sum_{m=1}^{k} w_m = \sum_{m=1}^{k}\sum_{i=1}^{l}(c_i^m A_i - \alpha_i^m)\phi(x_i) = \sum_{i=1}^{l}(A_i - \sum_{m=1}^{k}\alpha_i^m)\phi(x_i) = 0.$$

Hence adding (7b) to (4) does not affect the optimal solution set. Then (7b) implies

$$\sum_{o<m} \|w_o - w_m\|^2 = k\sum_{m=1}^{k} w_m^T w_m$$

so (4) and (7) have the same optimal solution set. Similar arguments can prove the relation between (4) and (8) as well. The equivalence of these formulations was first discussed in [11].

Note that (5) has kl variables where l of them are always zero. Hence we can also say it has $(k-1)l$ variables. Unfortunately (5) was considered as an impractical formula due to this huge amount of variables. Hence in [27] only small problems are tested. In the rest of this section we discuss a possible implementation for solving larger problems.

Remember that when solving binary SVM, a main difficulty is on the density of the kernel matrix as in general $K_{i,j}$ is not zero. Thus currently the decomposition method is the major method to solve binary support vector machines [21], [13], [22], [24]. It is an iterative process where in each iteration the index set of variables are separated to two sets B and N, where B is the working set. Then in that iteration variables corresponding to N are fixed while a sub-problem on variables corresponding to B is minimized. The size of B and the selection of its contents are both important issues on designing a decomposition method. For example, the Sequential Minimal Optimization (SMO) by Platt [22] considers only two variables in each iteration. Then in each iteration of the decomposition method the sub-problem on two variables can be analytically solved. Hence no optimization software is needed.

However, such working set selections in the binary case may not work here for the dual problem (5). Instead of only one linear constraint in the dual of (1), now in (5a) we have k linear constraints. Thus we can think (5a) as a system of k linear equations with kl variables. Note that the size of the working set is the number of variables we would like to change in one iteration. If it is less than k, then (5a) may be an over-determined linear system with more equations than variables so the solution of the decomposition algorithm cannot be moved. Therefore, in general we have to select more than k variables in the working set of each iteration. However, this is still not an easy task as bounded constraints (5b) have to be considered as well and we would like to have a systematic way which ensures that the selection leads to the decrease of the objective function. Unfortunately so far we have not found out effective ways of doing it.

Therefore, instead of working on (4), we consider the following problem by adding $\sum_{m=1}^{k} b_m^2$ to the objective function:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\sum_{m=1}^{k} \begin{bmatrix} w_m^T & b_m \end{bmatrix} \begin{bmatrix} w_m \\ b_m \end{bmatrix} + C\sum_{i=1}^{l}\sum_{m\neq y_i} \xi_i^m \\ & \begin{bmatrix} w_{y_i}^T & b_{y_i} \end{bmatrix} \begin{bmatrix} \phi(x_i) \\ 1 \end{bmatrix} \geq \begin{bmatrix} w_m^T & b_m \end{bmatrix} \begin{bmatrix} \phi(x_i) \\ 1 \end{bmatrix} + 2 - \xi_i^m, \\ & \xi_i^m \geq 0, i = 1, \ldots, l, m \in \{1, \ldots, k\} \backslash y_i. \end{aligned} \tag{9}$$

Then in the dual problem k linear constraints are removed:

$$\begin{aligned} \min \quad & \sum_{i,j} \left(\frac{1}{2} c_j^{y_i} A_i A_j - \sum_m \alpha_i^m \alpha_j^{y_i} + \frac{1}{2}\sum_m \alpha_i^m \alpha_j^m\right)(K_{i,j} + 1) - 2\sum_{i,m} \alpha_i^m \\ & 0 \leq \alpha_i^m \leq C, \ \alpha_i^{y_i} = 0, \\ & A_i = \sum_{m=1}^{k} \alpha_i^m, c_j^{y_i} = \begin{cases} 1 & \text{if } y_i = y_j, \\ 0 & \text{if } y_i \neq y_j, \end{cases} \\ & i = 1, \ldots, l, m = 1, \ldots, k. \end{aligned} \tag{10}$$

The decision function becomes

$$f(x) = \text{argmax}_{m=1,\ldots,k}(\sum_{i=1}^{l}(c_i^m A_i - \alpha_i^m)(K(x_i,x)+1)).$$

The idea of using bounded formulations for binary classification was first proposed in [10], [18]. For two-class problems, a detailed numerical study showing that the bounded formulation can achieve similar accuracy as the standard SVM is in [12], where the software BSVM was proposed. Without linear constraints, we hope that the problem of selecting the working set in the decomposition method becomes easier. Of course it is not clear yet if the accuracy will be affected as now k b^2 terms are added to the objective function. We will see the results in the experiment section.

For (10), the same decomposition method as in [12] can be applied. We rewrite (10) as the following general form

$$\begin{aligned}\min_{\alpha} \quad f(\alpha) = \quad & \frac{1}{2}\alpha^T Q \alpha - 2e^T \alpha \\ & 0 \le \alpha_i^m \le C, \alpha_i^{y_i} = 0, \\ & i = 1,\ldots,l, m = 1,\ldots,k,\end{aligned} \tag{11}$$

where e is a kl by 1 vector and Q is a kl by kl matrix. Then in each iteration, the sub-problem is as follows:

$$\begin{aligned}\min_{\alpha_B} \quad & \frac{1}{2}\alpha_B^T Q_{BB}\alpha_B - (2e_B - Q_{BN}\alpha_N^k)^T \alpha_B \\ & 0 \le (\alpha_B)_i \le C, i = 1,\ldots,q,\end{aligned} \tag{12}$$

where $\left[\begin{smallmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN}\end{smallmatrix}\right]$ is a permutation of the matrix Q and q is the size of the working set. Note that as $\alpha_i^{y_i} = 0, \forall i$ are fixed variables, we do not select them into the working set.

Assume that α is the solution of the current iteration and q is the size of the working set. We use the following working set selection of BSVM:

Let r be the number of free variables at α and calculate

$$v_i^m \equiv \begin{cases} \min(\nabla f(\alpha)_i^m, 0) & \text{if } \alpha_i^m = 0, \\ -|\nabla f(\alpha)_i^m| & \text{if } 0 < \alpha_i^m < C, \\ -\max(\nabla f(\alpha)_i^m, 0) & \text{if } \alpha_i^m = C. \end{cases}$$

Select indices of the largest $\min(q/2, r)$ elements in v, where α_i^m is free (i.e. $0 < \alpha_i^m < C$) into B

Select the $(q - \min(q/2, r))$ smallest elements in v into B.

The main idea of this working set selection is from Zoutendijk's feasible-direction method [28]:

$$\begin{aligned} \min_{d} \quad & \nabla f(\alpha)^T d \\ & -1 \le d \le 1, \\ & d_i^m \ge 0, \text{ if } \alpha_i^m = 0, \; d_i^m \le 0, \text{ if } \alpha_i^m = C. \end{aligned} \tag{13}$$

If d is an optimal solution of (13), then d_i^m must be $+1$ or -1. The vector v defined above actually has $v_i^m = \nabla f(\alpha)_i^m d_i^m$. Thus selecting the smallest elements in v is like selecting components with the best steepest descent direction. This has been used in some early implementations of the decomposition methods (e.g. [13]). However, it was pointed out in [12] that such a selection of the working set may lead to very slow convergence on some difficult problems. With some theoretical and experimental analyses, in [12] the authors proposed to include some of the largest elements of v whose corresponding α_i^m are free (i.e. $0 < \alpha_i^m < C$). Therefore, if q is the size of the working set, we pick $q/2$ indices from the smallest elements of v and the other $q/2$ from the largest of v whose corresponding α_i^m are free.

The major difficulty for the implementation lies in calculating elements of Q which are mainly accessed as Q_{BB} and Q_{BN} in (12). As Q in (5) has a very complicated form, it is essential to avoid some possible computational overheads.

More importantly, the form in (10) is not symmetric to indices i and j. That is,

$$\begin{aligned} & (\frac{1}{2} c_j^{y_i} A_i A_j - \sum_m \alpha_i^m \alpha_j^{y_i} + \frac{1}{2} \sum_m \alpha_i^m \alpha_j^m) K_{i,j} \\ \neq \; & (\frac{1}{2} c_i^{y_j} A_j A_i - \sum_m \alpha_j^m \alpha_i^{y_j} + \frac{1}{2} \sum_m \alpha_j^m \alpha_i^m) K_{j,i}. \end{aligned}$$

This increases the difficulty of writing down the explicit form of Q. As any quadratic formulation can be written as a symmetric form, we must reformulate (10). This will be explained in (15).

Note that as

$$
\begin{aligned}
&\frac{1}{2}\alpha^T Q\alpha \\
= \;& \sum_{i,j}\Big(\frac{1}{2}c_j^{y_i}A_iA_j - \sum_m \alpha_i^m\alpha_j^{y_i} + \frac{1}{2}\sum_m \alpha_i^m\alpha_j^m\Big)(K_{i,j}+1) && \text{(14a)}\\
= \;& \frac{1}{2}\sum_{m=1}^{k}\Big(\sum_{i=1}^{l}(c_i^m A_i - \alpha_i^m)\left[\begin{smallmatrix}\phi(x_i)\\ 1\end{smallmatrix}\right]\Big)^T\Big(\sum_{j=1}^{l}(c_j^m A_j - \alpha_j^m)\left[\begin{smallmatrix}\phi(x_j)\\ 1\end{smallmatrix}\right]\Big), && \text{(14b)}
\end{aligned}
$$

for any nonzero vector $v \in R^{kl} \times R^1$,

$$
\begin{aligned}
&v^T Q v\\
= \;& \sum_{m=1}^{k}\Big(\sum_{i=1}^{l}(c_i^m\sum_{o=1}^{k}v_i^o - v_i^m)\left[\begin{smallmatrix}\phi(x_i)\\ 1\end{smallmatrix}\right]\Big)^T\Big(\sum_{j=1}^{l}(c_j^m\sum_{o=1}^{k}v_j^o - v_j^m)\left[\begin{smallmatrix}\phi(x_j)\\ 1\end{smallmatrix}\right]\Big) \geq 0.
\end{aligned}
$$

Hence Q is positive semi-definite and (10) is a convex optimization problem. Here (14a) to (14b) is by direct calculation. It is easy to see how the first and third terms of α in (14a) are obtained from (14b). For the second term, we have

$$
\sum_{m,i} c_i^m A_i \sum_j \alpha_j^m = \sum_{i,j} A_i\alpha_j^{y_i} = \sum_{i,j}\sum_m \alpha_i^m\alpha_j^{y_i}.
$$

Next we discuss the explicit form of Q. In particular, we will show what a column of Q is. We consider the vector α as the following form $[\alpha_1^1,\ldots,\alpha_l^1,\ldots,\alpha_1^k,\ldots,\alpha_l^k]^T$. In addition, for $\alpha_1^m,\ldots,\alpha_l^m$, we assume that if $y_i < y_j$, α_i^m is in a position before α_j^m.

In (10), the quadratic terms compose of three groups. First we have

$$
\begin{aligned}
&c_j^{y_i}A_iA_j\\
= \;& \begin{cases}(\sum_{m=1}^{k}\alpha_i^m)(\sum_{m=1}^{k}\alpha_j^m) & \text{if } y_i = y_j,\\ 0 & \text{if } y_i \neq y_j.\end{cases}
\end{aligned}
$$

Clearly this part forms a symmetric matrix as follows

$$
\left[\begin{array}{cccccc}
\bar{K}_{(1),(1)} & & & \bar{K}_{(1),(1)} & & \\
 & \ddots & \cdots & & \ddots & \\
 & & \bar{K}_{(k),(k)} & & & \bar{K}_{(k),(k)}\\
 & \vdots & & & & \\
\bar{K}_{(1),(1)} & & & & & \\
 & \ddots & & & & \\
 & & \bar{K}_{(k),(k)} & & &
\end{array}\right],
$$

where $\bar{K}_{(m),(m)}$ includes all elements $\bar{K}_{i,j} \equiv K_{i,j}+1$ with $y_i = y_j = m$. This implies that for the $((s-1)l+j)$th column, row indices corresponding to $\alpha^r_{(y_j)}, r = 1, \ldots, k$ will have $\bar{K}_{(y_j),j}$ contributed by this part. We use $\alpha^r_{(y_j)}$ to represent $\{\alpha^r_i \mid y_i = y_j\}$ and $\bar{K}_{(y_j),j}$ for $\{\bar{K}_i \mid y_i = y_j\}$.

For the third term, $\sum_{i,j,m} \alpha^m_i \alpha^m_j \phi(x_i)^T \phi(x_j)$, clearly it forms the following symmetric matrix

$$\begin{bmatrix} \bar{K} & & \\ & \ddots & \\ & & \bar{K} \end{bmatrix}.$$

Similarly we say that at the $((s-1)l+j)$th column, row indices corresponding to $\alpha^s_1, \ldots, \alpha^s_l$ will include $\bar{K}_{1,j}, \ldots, \bar{K}_{l,j}$.

The most complicated one is the second part $-2\sum_{i,j,m} \alpha^m_i \alpha^{y_i}_j \bar{K}_{i,j}$ as it is not a symmetric form. To make it symmetric we use the property that any quadratic term $xy = \frac{1}{2}xy + \frac{1}{2}yx$:

$$\begin{aligned} & 2\sum_{i,j,m} \alpha^m_i \alpha^{y_i}_j \bar{K}_{i,j} \\ = & \sum_{i,j,m} (\alpha^m_i \alpha^{y_i}_j \bar{K}_{i,j} + \alpha^{y_i}_j \alpha^m_i \bar{K}_{i,j}) \\ = & \sum_{i,j,m} \alpha^m_i \alpha^{y_i}_j \bar{K}_{i,j} + \sum_{i,j,m} \alpha^{y_j}_i \alpha^m_j \bar{K}_{i,j}. \end{aligned} \tag{15}$$

Hence for the $((s-1)l+j)$th column, elements corresponding to $\alpha^r_{(s)}, r = 1, \ldots, k$ and $\alpha^{y_j}_i, i = 1, \ldots, l$ will have $-\bar{K}_{(s),j}$ and $-\bar{K}_{i,j}, i = 1, \ldots, l$, respectively.

In summary, the $((s-1)l+j)$th column of Q can be obtained as follows:

1. Obtain the column vector $\bar{K}_{i,j}, i = 1, \ldots, l$. Initialize the $((s-1)l+j)$th column as the a kl by one zero vector.

2. For elements corresponding to $\alpha^s_1, \ldots, \alpha^s_l$, add $\bar{K}_{1,j}, \ldots, \bar{K}_{l,j}$ (from the third part)

3. For elements corresponding to $\alpha^{y_j}_1, \ldots, \alpha^{y_j}_l$, minus $\bar{K}_{1,j}, \ldots, \bar{K}_{l,j}$ (from the second part)

4. For elements corresponding to each of $\alpha^1_{(y_j)}, \ldots, \alpha^k_{(y_j)}$, add $\bar{K}_{(y_j),j}$ (from the first part)

5. For elements corresponding to each of $\alpha^1_{(s)}, \ldots, \alpha^k_{(s)}$, minus $\bar{K}_{(s),j}$ (from the second part)

Thus Q in (12) is a dense but not a fully dense matrix. Its number of nonzero elements is about $O(l^2k)$. For practical implementations, we compute and cache

$K_{i,j}$ instead of elements in Q. The reduction of the cached matrix from kl to l further improve the training time of this algorithm.

There are different implementations of the above procedure. As the situation may vary for different computational environment, here we do not get into further details.

We have implemented this method as an extension of BSVM and LIBSVM which will be used for the experiments in Section V. In addition, the software is available at `http://www.csie.ntu.edu.tw/~cjlin/bsvm`

IV. Method by Crammer and Singer

In [7], Crammer and Singer proposed an approach for multi-class problems by solving a single optimization problem. We will also give a decomposition implementation here. Basically [7] solves the following primal problem:

$$\begin{aligned} \min_{w_m,\xi_i} \quad & \frac{1}{2}\sum_{m=1}^{k} w_m^T w_m + C\sum_{i=1}^{l}\xi_i \\ & w_{y_i}^T\phi(x_i) - w_m^T\phi(x_i) \geq e_i^m - \xi_i, i = 1,\ldots,l, \end{aligned} \tag{16}$$

where $e_i^m \equiv 1 - \delta_{y_i,m}$ and

$$\delta_{y_i,m} \equiv \begin{cases} 1 & \text{if } y_i = m, \\ 0 & \text{if } y_i \neq m. \end{cases}$$

Then the decision function is

$$\text{argmax}_{m=1,\ldots,k} w_m^T\phi(x).$$

The main difference from (4) is that (16) uses only l slack variables $\xi_i, i = 1,\ldots,l$. That is, instead of using ξ_i^m as the gap between each two decision planes, here the maximum of k such numbers is considered:

$$\xi_i = (\max_m(w_m^T\phi(x_i) + e_i^m) - w_{y_i}^T\phi(x_i))_+,$$

where $(\cdot)_+ \equiv \max(\cdot, 0)$. In addition, (16) does not contain coefficients $b_i, i = 1,\ldots,l$. Note that here we do not have to explicitly write down constraints $\xi_i \geq 0$ as when $y_i = m$, $e_i^m = 0$ so (16) becomes

$$0 \geq 0 - \xi_i,$$

which is exactly $\xi_i \geq 0$.

The dual problem of (16) is

$$\begin{aligned}\min_{\alpha} \quad f(\alpha) = \quad & \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} K_{i,j}\bar{\alpha}_i^T\bar{\alpha}_j + \sum_{i=1}^{l}\bar{\alpha}_i^T\bar{e}_i \\ & \sum_{m=1}^{k}\alpha_i^m = 0, i = 1, \ldots, l, & \text{(17a)} \\ & \alpha_i^m \leq 0, \text{ if } y_i \neq m, & \text{(17b)} \\ & \alpha_i^m \leq C, \text{ if } y_i = m, \\ & i = 1, \ldots, l, m = 1, \ldots, k,\end{aligned}$$

where $K_{i,j} \equiv \phi(x_i)^T\phi(x_j)$,

$$\bar{\alpha}_i \equiv [\alpha_i^1, \ldots, \alpha_i^k]^T, \text{ and } \bar{e}_i \equiv [e_i^1, \ldots, e_i^k]^T.$$

Then

$$w_m = \sum_{i=1}^{l}\alpha_i^m\phi(x_i).$$

If we write $\alpha \equiv [\alpha_1^1, \ldots, \alpha_1^k, \ldots, \alpha_l^1, \ldots, \alpha_l^k]^T$ and $e \equiv [e_1^1, \ldots, e_1^k, \ldots, e_l^1, \ldots, e_l^k]^T$, then the dual objective function can be written as

$$\frac{1}{2}\alpha^T(K \otimes I)\alpha + e^T\alpha,$$

where I is an k by k identity matrix and $\otimes$ is the Kronecker product. Since K is positive semi-definite, $K \otimes I$, the Hessian of the dual objective function is also positive semi-definite. This is another way to explain that (17) is a convex optimization problem.

The decision function is

$$\text{argmax}_{m=1,\ldots,k}\sum_{i=1}^{l}\alpha_i^m K(x_i, x).$$

The main difference between linear constraints (5a) and (17a) is that (5a) has k equations while (17a) has l. It is interesting that they come from the KKT condition on different primal variables. (5a) is due to the unconstrained variables $b_1, \ldots, b_k$ in (4) while (17a) is from the unconstrained variables $\xi_1, \ldots, \xi_l$. In addition, (17a) is much simpler than (5a) as each of its equations involves exactly k variables. In a sense we can say that (17a) contains l independent equations. Because of this advantage, unlike (5a) where we have to remove linear constraints and use (9), here it is easier to directly conduct working set selections for the decomposition method.

In [7] the authors proposed to choose k variables associated with the same x_i in the working set. That is, $\alpha_i^1, \ldots, \alpha_i^k$ are elements of the working set where the selection of the index i will be discussed in (20). Then the sub-problem is

$$\begin{aligned} \min \quad & \frac{1}{2} A \bar{\alpha}_i^T \bar{\alpha}_i + B^T \bar{\alpha}_i \\ & \sum_{m=1}^{k} \alpha_i^m = 0, \\ & \alpha_i^m \leq C_{y_i}^m, m = 1, \ldots, k, \end{aligned} \tag{18}$$

where

$$A = K_{i,i} \text{ and } B = \bar{e}_i + \sum_{j \neq i} K_{j,i} \bar{\alpha}_j,$$

In addition, $\bar{C}_{y_i}^m, m = 1, \ldots, k$ is a k by 1 vector with all elements zero except that the (y_i)th component is C.

The main reason of this setting is that (18) is a very simple problem. In [7], an $O(k \log k)$ algorithm was proposed for (18) while in [8], a simple iterative approach was used. Here we use the first method. In addition, it is easier to systematically calculate A and B so many complicated derivations in the previous section are avoided.

Note that the gradient of the dual objective function is

$$\begin{bmatrix} \sum_{j=1}^{l} K_{1,j} \alpha_j^1 + e_1^1 \\ \vdots \\ \sum_{j=1}^{l} K_{1,j} \alpha_j^k + e_1^k \\ \sum_{j=1}^{l} K_{2,j} \alpha_j^1 + e_2^1 \\ \vdots \\ \sum_{j=1}^{l} K_{2,j} \alpha_j^k + e_2^k \\ \vdots \end{bmatrix}.$$

Then B, a k by 1 vector, can be calculated as follows by the information of the gradient:

$$B_m = \frac{\partial f(\alpha)}{\partial \alpha_i^m} - K_{i,i} \alpha_i^m = \frac{\partial f(\alpha)}{\partial \alpha_i^m} - A \alpha_i^m, m = 1, \ldots, k.$$

Therefore, during iterations it is essential to always keep the gradient updated. This is done after new $\alpha_i^1, \ldots, \alpha_i^k$ are obtained by (18) and $O(kl)$ operations are needed.

Next we discuss the selection of the working set and the stopping condition of the decomposition method. The KKT condition requires that there are $b_1, \ldots, b_l$ and $\lambda_1^1 \geq 0, \ldots, \lambda_l^k \geq 0$ such that for all $i = 1, \ldots, l, m = 1, \ldots, k$,

$$\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m - b_i = -\lambda_i^m \text{ and } \lambda_i^m(\bar{C}_{y_i}^m - \alpha_i^m) = 0.$$

They are equivalent to that for all $i = 1, \ldots, l, m = 1, \ldots, k$,

$$\begin{aligned} \sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m - b_i \quad &= 0 \quad \text{if } \alpha_i^m < \bar{C}_{y_i}^m, \\ &\leq 0 \quad \text{if } \alpha_i^m = \bar{C}_{y_i}^m. \end{aligned}$$

We can rewrite this as

$$\max_{\alpha_i^m \leq \bar{C}_{y_i}^m} (\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m) \leq b_i \leq \min_{\alpha_i^m < \bar{C}_{y_i}^m} (\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m). \tag{19}$$

Then during iterations, we select the next working set $\{\alpha_i^1, \ldots, \alpha_i^k\}$ with i from

$$\text{argmax}_i(\max_{\alpha_i^m \leq \bar{C}_{y_i}^m} (\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m) - \min_{\alpha_i^m < \bar{C}_{y_i}^m} (\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m)). \tag{20}$$

In other words, among the l k-component groups of variables, we select the on with the largest violation of the KKT condition. For binary SVM, choosing indices which most violate the KKT condition has been a common strategy (e.g. [4]) though here instead we choose a whole group at once. Then for variables $\{\alpha_i^1, \ldots, \alpha_i^k\}$ which are selected, they do not satisfy the KKT condition of the sub-problem (18) so solving (18) will guarantee the strict decrease on the objective function of the dual problem.

Following (19) the stopping criterion can be

$$\max_i(\max_{\alpha_i^m \leq \bar{C}_{y_i}^m} (\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m) - \min_{\alpha_i^m < \bar{C}_{y_i}^m} (\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m)) < \epsilon, \tag{21}$$

where ϵ is the stopping tolerance.

The convergence of the above decomposition method has been proved in [17]. In addition, [17] shows that the limit of

$$\max_i \Big(\max_{\alpha_i^m \leq \bar{C}_{y_i}^m} \big(\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m\big) - \min_{\alpha_i^m < \bar{C}_{y_i}^m} \big(\sum_{j=1}^{l} K_{i,j}\alpha_j^m + e_i^m\big)\Big)$$

goes to zero as the number of iterations goes to infinity. Hence in a finite number of iterations, the decomposition method stops as (21) is satisfied. The implementation is now part of BSVM 2.0 which is also publicly available to users.

V. Numerical Experiments

A. Data and Implementation

In this section we present experimental results on several problems from the Statlog collection [20] and the UCI Repository of machine learning databases [1]. From UCI Repository we choose the following datasets: iris, wine, glass, and vowel. Those problems had already been tested in [27]. From Statlog collection we choose all multi-class datasets: vehicle, segment, dna, satimage, letter, and shuttle. Note that except problem dna we scale all training data to be in [-1, 1]. Then test data are adjusted using the same linear transformation. For the problem dna, we do not scale its binary attributes. We give problem statistics in Table VI.1. For some of these problems test sets are available. Note that for problems glass and satimage, there is one missing class. That is, in the original application there is one more class but in the data set no examples are with this class. In the last column we also give the best test rate listed in statlog homepage. Note that these best rates were obtained by four different learning methods.

The most important criterion for evaluating the performance of these methods is their accuracy rate. However, it is unfair to use only one parameter set and then compare these five methods. Practically for any method people find the best parameters by performing the model selection. This is conducted on the training data where the test data are assumed unknown. Then the best parameter set is used for constructing the model for future testing. Note that details of how we conduct the model selection will be discussed later in this section. To reduce the search space of parameter sets, here we train all datasets only with the RBF kernel $K(x_i, x_j) \equiv e^{-\gamma\|x_i - x_j\|^2}$. In addition, for methods solving several binary SVMs (one-against-one, one-against-all, and DAG), for each model we consider that C and γ of all binary problems are the same. Note that this issue does

TABLE V.1

PROBLEM STATISTICS

Problem	#training data	#testing data	#class	#attributes	statlog rate
iris	150	0	3	4	
wine	178	0	3	13	
glass	214	0	6	13	
vowel	528	0	11	10	
vehicle	846	0	4	18	
segment	2310	0	7	19	
dna	2000	1186	3	180	95.9
satimage	4435	2000	6	36	90.6
letter	15000	5000	26	16	93.6
shuttle	43500	14500	7	9	99.99

not arise for two all-together methods as each model corresponds to only one optimization problem.

We use similar stopping criteria for all methods. For each problem we stop the optimization algorithm if the KKT violation is less than 10^{-3}. To be more precise, each dual problem of the one-against-one and one-against-all approaches has the following general form:

$$\begin{aligned} \min_{\alpha} \quad & f(\alpha) \\ & y^T\alpha = 0, \\ & 0 \leq \alpha_i \leq C, \end{aligned}$$

where $y_i = \pm 1$. Using a similar derivation of the stopping criterion (21) of the method by Crammer and Singer, we have

$$\begin{aligned} & \max(\max_{\alpha_i<C, y_i=1} -\nabla f(\alpha)_i, \max_{\alpha_i>0, y_i=-1} \nabla f(\alpha)_i) \\ \leq & \min(\min_{\alpha_i<C, y_i=-1} \nabla f(\alpha)_i, \min_{\alpha_i>0, y_i=1} -\nabla f(\alpha)_i) + 10^{-3}. \end{aligned} \tag{22}$$

For (10) of the all-together approach, (22) becomes even simpler as there are no vector y. Unfortunately though these stopping criteria are nearly the same, they are not fully comparable due to different size of dual problems in these approaches. We will elaborate more on this issue in Section VI. Note that for problems letter and shuttle, a relaxed tolerance 0.1 is used for the method by Crammer and Singer. as otherwise it takes too much training time. More information on the stopping criteria of the decomposition method can be found in [17].

The computational experiments for this section were done on a Pentium III-500 with 384MB RAM using the gcc compiler. For each optimization problem (either binary SVMs or the all-together approaches), we allocate 256MB memory as the cache for storing recently used kernel elements. Each element of Q_{ij} stored in the cache is in double precision. For the same size of the cache, if the single precision is used, the number of elements which can be stored in the cache is doubled. We have both implementations but here only the double-precision one is used.

While implementing these approaches using the decomposition method, we can use a shrinking technique for reducing the training time. To be more precise, if most variables are finally at bounds, the shrinking technique reduces the size of the working problem by considering only free variables. We have implemented the shrinking technique on all five methods. For the three methods based on binary classifiers, details are in [4, Section 4]. For two all-together methods, the implementation is more sophisticated. This is a disadvantage of all-together methods. Though they consider only one optimization problem, this problem is more complicated for practical implementations.

B. Results and Discussions

For each problem, we estimate the generalized accuracy using different kernel parameters γ and cost parameters C: $\gamma = [2^4, 2^3, 2^2, \ldots, 2^{-10}]$ and $C = [2^{12}, 2^{11}, 2^{10}, \ldots, 2^{-2}]$. Therefore, for each problem we try $15 \times 15 = 225$ combinations. We use two criteria to estimate the generalized accuracy. For datasets dna, satimage, letter, and shuttle where both training and testing sets are available, for each pair of (C, γ), the validation performance is measured by training 70% of the training set and testing the other 30% of the training set. Then we train the whole training set using the pair of (C, γ) that achieves the best validation rate and predict the test set. The resulting accuracy is presented in the "rate" column of Table VI.2. Note that if several (C, γ) have the same accuracy in the validation stage, we apply all of them to the test data and report the highest rate. For the other six smaller datasets where test data may not be available, we simply conduct a 10-fold cross-validation on the whole training data and report the best cross-validation rate.

Table VI.2 presents the result of comparing five methods. We present the optimal parameters (C, γ) and the corresponding accuracy rates. Note that the "C&S" column means the method by Crammer and Singer. It can be seen that optimal parameters (C, γ) are in various ranges for different problems so it is

TABLE V.2

A comparison using the RBF kernel (best rates bold-faced)

	One-against-one		DAG		One-against-all		[25], [27]		C&S	
Problem	(C,γ)	rate	(C,γ)	rate	(C,γ)	rate	(C,γ)	rate	(C,γ)	rate
iris	$(2^{12},2^{-9})$	**97.333**	$(2^{12},2^{-8})$	96.667	$(2^{9},2^{-3})$	96.667	$(2^{12},2^{-8})$	**97.333**	$(2^{10},2^{-7})$	**97.333**
wine	$(2^{7},2^{-10})$	**99.438**	$(2^{6},2^{-9})$	98.876	$(2^{7},2^{-6})$	98.876	$(2^{0},2^{-2})$	98.876	$(2^{1},2^{-3})$	98.876
glass	$(2^{11},2^{-2})$	71.495	$(2^{12},2^{-3})$	**73.832**	$(2^{11},2^{-2})$	71.963	$(2^{9},2^{-4})$	71.028	$(2^{4},2^{1})$	71.963
vowel	$(2^{4},2^{0})$	**99.053**	$(2^{2},2^{2})$	98.674	$(2^{4},2^{1})$	98.485	$(2^{3},2^{0})$	98.485	$(2^{1},2^{3})$	98.674
vehicle	$(2^{9},2^{-3})$	86.643	$(2^{11},2^{-5})$	86.052	$(2^{11},2^{-4})$	**87.470**	$(2^{10},2^{-4})$	86.998	$(2^{9},2^{-4})$	86.761
segment	$(2^{6},2^{0})$	97.403	$(2^{11},2^{-3})$	97.359	$(2^{7},2^{0})$	97.532	$(2^{5},2^{0})$	**97.576**	$(2^{0},2^{3})$	97.316
dna	$(2^{3},2^{-6})$	95.447	$(2^{3},2^{-6})$	95.447	$(2^{2},2^{-6})$	95.784	$(2^{4},2^{-6})$	95.616	$(2^{1},2^{-6})$	**95.869**
satimage	$(2^{4},2^{0})$	91.3	$(2^{4},2^{0})$	91.25	$(2^{2},2^{1})$	91.7	$(2^{3},2^{0})$	91.25	$(2^{2},2^{2})$	**92.35**
letter	$(2^{4},2^{2})$	**97.98**	$(2^{4},2^{2})$	**97.98**	$(2^{2},2^{2})$	97.88	$(2^{1},2^{2})$	97.76	$(2^{3},2^{2})$	97.68
shuttle	$(2^{11},2^{3})$	99.924	$(2^{11},2^{3})$	99.924	$(2^{9},2^{4})$	99.910	$(2^{9},2^{4})$	99.910	$(2^{12},2^{4})$	**99.938**

TABLE V.3

Training time, testing time, and number of support vectors (time in seconds; best training and test time bold-faced; least number of SVs italicized)

	One-against-one		DAG		One-against-all		[25], [27]		C&S	
Problem	training testing	#SVs	training testing	#SVs	training testing	#SVs	training testing	#SVs	training testing	#SVs
iris	**0.04**	16.9	**0.04**	*15.6*	0.10	16.0	0.15	16.2	16.84	27.8
wine	**0.12**	56.3	0.13	56.5	0.20	*29.2*	0.28	54.5	0.39	41.6
glass	**2.42**	*112.5*	2.85	114.2	10.00	129.0	7.94	124.1	7.60	143.3
vowel	**2.63**	345.3	3.98	365.1	9.28	392.6	14.05	*279.4*	20.54	391.0
vehicle	**19.73**	302.4	35.18	293.1	142.50	343.0	88.61	*264.2*	1141.76	264.9
segment	**17.10**	442.4	23.25	*266.8*	68.85	446.3	66.43	358.2	192.47	970.3
dna	**10.60**	967	10.74	967	23.47	1152	13.5	951	16.27	*945*
	6.91		**6.30**		8.43		6.91		6.39	
satimage	**24.85**	1611	25.1	1611	136.42	2170	48.21	*1426*	89.58	2670
	13.23		12.67		19.22		**11.89**		23.61	
letter	**298.08**	8931	298.62	8931	1831.80	10129	8786.20	7627	1227.12*	*6374*
	126.10		**92.8**		146.43		142.75		110.39	
shuttle	170.45	301	**168.87**	301	202.96	330	237.80	202	2205.78*	*198*
	6.99		5.09		5.99		4.64		**4.26**	

*: stopping tolerance $\epsilon = 0.1$ is used.

essential to test so many parameter sets. We also observe that their accuracy is very similar. That is, no one is statistically better than the others. Comparing to earlier results listed in Statlog (see the last column of Table VI.1), the accuracy obtained by SVM is competitive or even better. For example, among the four problems dna to shuttle, the one-against-one approach obtains better accuracy on satimage and letter. For the other two problems, the accuracy is also close to that in Table VI.1.

We also report the training time, testing time, and the number of unique support vectors in Table VI.3. Note that they are results when solving the optimal model. For small problems there are no testing time as we conduct cross validation. Here we say "unique" support vectors because a training data may correspond to different nonzero dual variables. For example, for the one-against-one and one-against-all approaches, one training data may be a support vector in different binary classifiers. For the all-together methods, there are kl variables so one data may associate with different nonzero dual variables. Here we report only the number of training data which corresponds to at least one nonzero dual variable. We will explain later that this is the main factor which affects the testing time. Note that the number of support vectors of the first six problems are not integers. This is because they are the average of the 10-fold cross-validation.

TABLE V.4

A COMPARISON USING THE LINEAR KERNEL (BEST RATES BOLD-FACED)

	One-against-one		DAG		One-against-all		[25], [27]		C&S	
Problem	C	rate	C	rate	C	rate	C	rate	C	rate
iris	2^{4}	**97.333**	2^{8}	**97.333**	2^{12}	96.000	2^{5}	**97.333**	2^{0}	87.333
wine	2^{-2}	**99.438**	2^{-2}	98.315	2^{2}	98.876	2^{-1}	98.876	2^{-1}	99.438
glass	2^{8}	**66.355**	2^{4}	63.551	2^{5}	58.879	2^{9}	65.421	2^{6}	62.617
vowel	2^{5}	**82.954**	2^{6}	81.439	2^{11}	50.000	2^{8}	67.424	2^{6}	63.068
vehicle	2^{5}	80.615	2^{5}	**80.851**	2^{12}	78.132	2^{10}	80.142	2^{4}	79.669
segment	2^{12}	**96.017**	2^{11}	95.844	2^{12}	93.160	2^{8}	95.454	2^{-2}	92.165

For the training time, one-against-one and DAG methods are the best. In fact the two methods have the same training procedure. Though we have to train as many as $k(k-1)/2$ classifiers, as each problem is smaller (only data from two classes), the total training time is still less. Note that in Table VI.3 the training time of one-against-one and DAG methods may be quite different for the same problem (e.g. vehicle). This is due to the difference on the optimal parameter sets.

Although we improve the method from [25], [27] with efforts in Section III, its training speed remains slow. The convergence speed of the method by Crammer and Singer is also not good. Especially for some problems (iris, vehicle, and shuttle) its training time is huge. For these problems we note that the optimal parameter of C is quite large. The experience in [12] shows that if the working set selection is not good, the convergence of the decomposition method may be slow when using a large C. Thus the difficulty might be on the working set selection. As in each iteration only one of the l equalities is involved, there may not be enough interactions among these l groups of variables.

Regarding the testing time, though the decision function is more complicated than the binary case, our experimental results indicate that in general the testing time is still dominated by the kernel evaluations. Note that to save testing time, we always calculate and store all $K(x_i, x)$ first, where x_i is any "unique" support vector and x is the test data. Then this $K(x_i, x)$ may be used in several places of the decision function. We observe that if k is small (≤ 10), kernel evaluations take more than 90% of the testing time. Therefore, we can say that in general the testing time is proportional to the number of "unique" support vectors.

We also observe that between the one-against-one and DAG methods, DAG is really a little faster on the testing time.

We then discuss the number of support vectors. We can see that for larger problems, the method from [25], [27] returns fewer support vectors than all three binary-based approaches. Note that we generally observe that for the same parameter set, it needs fewer support vectors. This is consistent with the results in [27]. Here we do not really have such a comparison as the optimal parameters vary for all approaches. Especially for small problems their optimal parameters are very different. However, for large problems their optimal parameters are similar so in Table VI.2 the "#SVs" column of the method from [25], [27] really shows smaller numbers. Therefore, if the testing time is very important, this method can be an option. On the other hand, we cannot draw any conclusions about the method by Crammer and Singer. Sometimes it needs very few support vectors but sometimes the number is huge.

We would like to note that for the problem dna, several parameters get the best result during the validation stage. Then when applying them to the test data, some of them have the same accuracy again. In Table VI.2 we present only the result which has the smallest number of support vectors.

Overall, except the training time, other factors are very similar for these approaches. Thus we suggest that one-against-one and DAG approaches are more suitable for practical use.

To have more complete analyses, we test these methods by using the linear kernel $K(x_i, x_j) = x_i^T x_j$. Results are in Table VI.4. Due to the limit of computational time, we report only small problems. Now the only parameter is C so we test 15 different C's and report the best rate. Comparing to Table VI.2, the difference on the best rates is apparent. The one-against-all method returns the worst accuracy for some problems. Overall one-against-one and DAG still perform well. The comparison on linear and nonlinear kernels also reveals the necessity of using nonlinear kernels in some situations. The observation that overall the RBF kernel produces better accuracy is important as otherwise we do not even need to study the decomposition methods which is specially designed for the nonlinear case. There are already effective methods to solve very large problems with the linear kernel.

Finally we would like to draw some remarks about the implementation of these methods. The training time of the one-against-all method can be further improved as now for each parameter set, k binary problems are treated independently. That is, kernel elements used when solving one binary problem are not stored and passed to other binary problems though they have the same kernel matrix. Hence the same kernel element may be calculated several times. However, we expect that even with such improvements it still cannot compete with one-against-one and DAG on the training time. For all other approaches, caches have been implemented so that all problems involved in one model can share them. On the other hand, for all approaches, now different models (i.e. different parameter sets) are fully independent. There are no caches for passing kernel elements from one model to another.

VI. Discussions and Conclusions

We note that a difference between all-together methods is that the method by Crammer and Singer does not include bias terms $b_1, \ldots, b_m$. We are wondering whether this may affect the training time. Here we give a brief discussion on this issue. If $b_1, \ldots, b_k$ are added, (16) becomes

$$\begin{aligned} \min \quad & \frac{1}{2}\sum_{m=1}^{k} w_m^T w_m + C\sum_{i=1}^{l} \xi_i \\ & (w_{y_i}^T \phi(x_i) + b_{y_i}) - (w_m^T \phi(x_i) + b_m) \geq e_i^m - \xi_i, i = 1, \ldots, l, \end{aligned} \tag{23}$$

Then from the KKT condition the dual has some additional equalities:

$$\sum_{i=1}^{l} \alpha_i^m = 0, m = 1, \ldots, k.$$

Then again the same difficulty on the working set selection for (5) happens again so we may have to add $\frac{1}{2}\sum_{m=1}^{k} b_m^2$ to the objective function of (23). The dual problem hence is

$$\begin{aligned} \min \quad f(\alpha) = \; & \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}(K_{i,j}+1)\bar{\alpha}_i^T\bar{\alpha}_j + \sum_{i=1}^{l}\bar{\alpha}_i^T\bar{e}_i \\ & \sum_{m=1}^{k} \alpha_i^m = 0, \\ & \alpha_i^m \leq 0, \text{ if } y_i \neq m, \\ & \alpha_i^m \leq C, \text{ if } y_i = m, \\ & i = 1, \ldots, l, m = 1, \ldots, k, \end{aligned} \tag{24}$$

which can be solved by the same decomposition method described in Section IV. Then

$$\begin{bmatrix} w_m \\ b_m \end{bmatrix} = \sum_{i=1}^{l} \alpha_i^m \begin{bmatrix} \phi(x_i) \\ 1 \end{bmatrix}$$

so the decision function is

$$\text{argmax}_m w_m^T \phi(x) + b \equiv \text{argmax}_m \sum_{i=1}^{l} \alpha_i^m (K(x_i, x) + 1).$$

We modify the code for (24) and by using the optimal parameters listed in the last column of Table VI.3, a comparison on the number of iterations between solving (17) and (24) is in Table VI.5. We provide only results of the four large problems. It can be clearly seen that after adding the bias term, the performance is not better. It is not clear yet why the number of iterations is nearly doubled but this is not surprising as in [12] we have demonstrated that for binary SVM, with or without b^2 in the objective function, the performance of the same decomposition method can be quite different. Overall we realize that the working set selection (20) may not be very good to have fast convergence for (16).

TABLE VI.5

Number of iterations: a comparison on solving (17) and (24)

Problem	Eq. (17)	Eq. (24)
dna	8653	15475
satimage	13109	27751
letter	16341	26424
shuttle	163394	286734

The second issue which will be discussed here is about the stopping criteria. Though we use stopping criteria from the same derivation, they are affected by the problem size. That is, the smaller the dual problem is, fewer variables are involved in the calculation of $\nabla f(\alpha)_i$ of (22). Therefore, in some sense approaches like one-against-one which use smaller dual problems take advantages (or say they stop earlier). A possible remedy is to divide the left-hand-side of (22) by the size of the dual problem. More investigation are needed for this issue. However, even with such differences, our conclusion that all-together methods take more training time should remain as from Table VI.3 we can see for both approaches on some problems their training time is much longer. For example, the method by Crammer and Singer solve a kl-variable problem and for problems letter and shuttle, we relax the stopping tolerance to 0.1. This is like that we divide the left-hand-side of (21) by 100 which is grater then k, the number of classes. However, its training time still cannot compete with that of the one-against-all approach which solves dual problems with l variables.

In conclusion, we have discussed decomposition implementations for two all-together methods and compared them with three methods based on several binary classifiers: one-against-one, one-against-all and DAG. Experiments on large problems show that one-against-one method and DAG may be more suitable for practical use. A future work is to test data with a very large number of classes. Especially people have suspected that there may have more differences among these methods if the data set has few points in many classes [26].

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant NSC 89-2213-E-002-106. Part of our implementations benefited from the work of Chih-Chung Chang. We also thank Jason Weston, Andre Elisseeff, Sathiya Keerthi, Yann Guermeur, and Yoram Singer for many helpful comments.

References

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

[2] L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwriting digit recognition. In *International Conference on Pattern Recognition*, pages 77–87. IEEE Computer Society Press, 1994.

[3] E. J. Bredensteiner and K. P. Bennett. Multicategory classification by support vector machines. *Computational Optimizations and Applications*, pages 53–79, 1999.

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

[5] K. K. Chin. Support vector machines applied to speech pattern classification. Master's thesis, University of Cambridge, 1998.

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

[7] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. In *Computational Learing Theory*, pages 35–46, 2000.

[8] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. Technical report, School of Computer Science and Engineering, Hebrew University, 2001.

[9] 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`.

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

[11] Y. Guermeur. Combining discriminant models with new multiclass svms. Neuro COLT Technical Report NC-TR-00-086, LORIA Campus Scientifique, 2000.

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

[13] 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.

[14] J. Kindermann, E. Leopold, and G. Paass. Multi-class classification with error correcting codes. In E. Leopold and M. Kirsten, editors, *Treffen der GI-Fachgruppe 1.1.3, Maschinelles Lernen*, 2000. GMD Report 114.

[15] 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.

[16] 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.

[17] C.-J. Lin. Stopping criteria of decomposition methods for support vector machines: a theoretical justification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2001.

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

[19] E. Mayoraz and E. Alpaydin. Support vector machines for multi-class classification. In *IWANN (2)*, pages 833–842, 1999.

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

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

[22] 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.

[23] J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems*, volume 12, pages 547–553. MIT Press, 2000.

[24] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola. Support vector machine reference manual. Technical Report CSD-TR-98-03, Royal Holloway, University of London, Egham, UK, 1998.

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

[26] J. Weston, 2001. Private communication.

[27] J. Weston and C. Watkins. Multi-class support vector machines. In M. Verleysen, editor, *Proceedings of ESANN99*, Brussels, 1999. D. Facto Press.

[28] G. Zoutendijk. *Methods of feasible directions.* Elsevier, 1960.