A Parallel Decomposition Algorithm for Training Multiclass Kernel-based Vector Machines*

Lingfeng Niu [a,b] and Ya-xiang Yuan [b]

[a] Research Center on Fictitious Economy and Data Science
Graduate University of Chinese Academy Sciences

[b] State Key Laboratory of Scientific and Engineering Computing,
Institute of Computational Mathematics and Scientific/Engineering Computing,
AMSS, CAS, Beijing 100190, CHINA

Abstract

We present a decomposition method for training Crammer and Singer's multiclass kernel-based vector machine model. A new working set selection rule is proposed. Global convergence of the algorithm based on this selection rule is established. Projected gradient method is chosen to solve the resulting quadratic subproblem at each iteration. An efficient projection algorithm is designed by exploiting the structure of the constraints. Parallel strategies are given to utilize the storage and computational resources available on multiprocessor system. Numerical experiment on benchmark problems demonstrates that the good classification accuracy and remarkable time saving can be achieved.

1 Introduction

Let $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_l, y_l)\}$ be a set of training samples belonging to m different classes, where $\mathbf{x}_i \in \mathcal{X} \subseteq R^n$ and $y_i \in \mathcal{Y} = \{1, \cdots, m\}$ are the input data and corresponding label for the sample i, respectively. The goal of a classification problem is to construct a classifier which, given a new data point, will correctly predict the class to which the new point belongs. When $m = 2$, the problem is called binary classification; When $m > 2$, it is called multiclass classification. We consider the problem of multiclass classification in this work.

Support Vector Machine(SVM) [4, 11] is a powerful machine learning technique for classification. It is developed from statistical learning theory [45, 46] and gains lots of attention because of its excellent performance empirically [34, 36, 24]. However, SVM is inherently designed for binary classification. How to effectively extend it to multiclass case is still an ongoing research issue. Considerable efforts have been devoted to this area, which fall into two main categories: One is to build a multiclass predictor on top of a series of binary SVMs by some special scheme [5, 39, 19, 27, 37, 1, 12, 13, 25, 15]; The other is to construct a multiclass classifier by solving a single optimization problem, which is developed by borrowing the idea from the design of SVMs [46, 6, 47, 22, 28, 29].

*This work was partially supported by NSFC grants 10831006, 70621001,70921061,70531040 and CAS grant kjcx-yw-s7.

More concretely, for the first kind of methods, one-against-all is the earliest used implementation [5]. This strategy is to train m different binary classifiers, each one is trained to distinguish the samples in a single class from the samples in all remaining classes. When it is desired to classify a new sample, the m classifiers are run, and the classifier which outputs the largest value is chosen [38]. Another conceptual simple scheme to build the multiclassifier on top of binary classifiers is one-against-one, which was introduced in [39], and the first use on SVMs was in [19, 27]. In this approach, $\binom{m}{2}$ binary classifiers are trained; each classifier separates a pair of classes. When a new sample comes, all $m(m-1)/2$ classifiers are run. If the classifier between the ith and jth class predicts the sample is in the ith class, the vote for the ith class is added by one. Otherwise, the jth class is increased by one. At last, the sample is predicted as the class with the largest vote. The directed acyclic graph SVM(DAGSVM) [37] is the same as the one-against-one in the training phase - one SVM is trained for each pair of classes. In the testing phase, the ith class and jth class are compared, and whichever class achieves a lower score is removed from further consideration. By repeating this process $m-1$ times, $m-1$ classes are removed from consideration, and the final remaining class is predicted [23]. The idea of using error-correcting codes for multiclass classification was first introduced by Dietterich and Bakiri [17]. Then several error-correcting code approaches are developed to combine binary classifiers into a multiclass classification system [31, 1, 12, 13, 25, 15, 20]. Among them, [31, 1, 12, 13, 25, 15] can use SVMs as the underlying binary classifiers.

Hus and Lin [23] presented an empirical study comparing various methods of multiclass classification using SVMs as the binary learner. They concluded that one-against-one and DAGSVM are more accurate than one-against-all. However, Rifkin and Klautau discovered through extensive experiments that when the underlying binary SVMs are well-tuned, the performance of one-against-all is at least as good as one-against-one and several error-correcting code approaches in accuracy [38]. Therefore, they recommended one-against-all for practical use because of its simplicity. From these discussions we can see that how to make sure all the element binary SVMs are well-tuned is not an easy task. Even for the simplest one-against-all scheme, at least m sets of parameters need to be determined. Considering the complexity of tuning parameters, the method which can construct the multiclassifier by solving a single optimization problem is preferred.

The first SVM multiclass model, which solves a single optimization problem rather than combines the solutions to a collection of binary problems, was introduced simultaneously by Vapnik [46] and Weston and Watkins [47]. Later, it was pointed out in [22, 23] that the formulation proposed in [46] and [47] are essentially identical, whose primal forms are also equivalent to the models in [6, 22]. In these approaches, the penalty is payed based on the relative values output between different classes instead of considering the margin requirement for each class separately. As a consequence, the dual form of the optimization problem does not succeed in fully eliminating the primal variables and more equality constraints, which makes the model much more difficult to train than the standard SVMs [38]. Lee, Lin and Wahba presented another multiclassification model in [28, 29]. This model is constructed based on the asymptotic properties of SVM regularization for binary classification[30]. Because the asymptotic properties only hold as the number of data points goes to infinity and the regularization term is ignored, the performance of this method is not satisfiable in real-world classification tasks which use limited amounts of data. Crammer and Singer proposed their multiclass SVM model in [14], which can be considered as specific case of their formalism for multiclass classification using continuous output coding [12, 13, 15]. Different from the approach of Weston and Watkins, Crammer and

Singer's model only penalizes the class with the largest violation, which results in only one slack variable for each data point in the dual form, rather than $m-1$ slack variables per point in Weston and Watkins' formulation.

When Crammer and Singer gave the model, they also developed a decomposition algorithm to train the model at the same time, in which a single data point is chosen at each step and a fixed-point algorithm is designed to solve the reduced m-variable quadratic programming. Various experiments on a number of data sets from the UCI repository [2] are carried out by Crammer and Singer to compare their approach with the one-against-all scheme. Based on the experiment results, they claimed that the new model outperforms one-against-all in accuracy. In [23], Crammer and Singer's model was compared with one-against-all, one-against-one, DAGSVM and Weston and Watkins' model. The results also show that if the regularization parameter and the kernel are carefully chosen, Crammer and Singer's model can achieve state-of-the-art accuracy. However, the efficiency of Crammer and Singer's training algorithm is not comparable to the algorithm of standard SVMs, so the author recommended one-against-one and DAGSVM for the practical use.

In recent years, Crammer and Singer's model draws more and more attentions from the research community and lots of efforts are devoted to design efficient training algorithms for it and its extended models [43, 42, 9, 26]. A cutting plane algorithm [43] was proposed to train a general structured model in primal form with linear kernel[1], which takes Crammer and Singer's model as a special instance. The Bundle method considered in [42] extends the cutting plane algorithm to general regularized risk minimization problems. In this work, a modular framework is designed to support different regularizers and risk functions for different machine learning methods[2]. When the l_2 normalizer and vectorial loss function are used, the framework is reduced to Crammer and Singer's model. The development of all these algorithms require the explicit representation of samples in the primal feature space for the finding of cutting plane or calculation of subgradient of the objective function. Because the vector of primal variables could be in infinite dimension space for some kind of nonlinear kernels(for example, the RBF kernel), we can not extend these algorithms directly to the case of general nonlinear kernels.

In [9], the authors proposed the exponentiated gradient method for the training of structured log-linear and max-margin model, which covers Crammer and Singer's model as a special case. The related convex programming is solved in its dual form by the exponentiated gradient updating scheme. However, calculating the gradient of dual objective function still requires the representation of samples in primal form. In [26], an elegant coordinate descent algorithm is specially designed for Crammer and Singer's model[3]. Although the dual form of the model is considered, the inner-product of samples is decoupled to save the floating point operations when calculating the gradient, and thus the algorithm can not be generalized to solve the problem with nonlinear kernels either. Since general nonlinear kernels are of great importance for describing and analyzing considerable amount of multiclass classification problems [40], we attempt to design efficient training algorithm for Crammer and Singer's model with nonlinear kernels in this work.

[1]Efficient implementation is provided in the corresponding package SVM$^{\text{multiclass}}$, which is public available at `http://svmlight.joachims.org/svm_multiclass.html`.

[2]The framework is implemented in the ELEFANT package which can be downloaded at `http://elefant.developer.nicta.com.au`.

[3]The implementation of this algorithm is contained in the LIBLINEAR package, which is available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

The structure of the paper is as follow. In the next section, after briefly introducing the mathematical formulation of Crammer and Singer's model, we give the general decomposition algorithm for the model and propose our new working set selection rule under this framework. A discussion on global convergence of the algorithm is also given. Section 3 concentrates on applying the projected gradient(PG) method to solve the quadratic subproblem at each iteration efficiently. Firstly a special projection algorithm is designed by exploiting the structure of the constraints. Then we explain in more detail each of its main steps and how they differ from published PG algorithms. We address on the issue of implementation such as practical working set selection rule, kernel evaluations and parallelization strategies in section 4. Numerical experiments for the serial and parallel implementation on several public data sets are explained in section 5. A brief conclusion is given in section 6.

2 The New Decomposition Algorithm

From mathematical point of view, the classification problem can be interpreted as finding a function $H : \mathcal{X} \rightarrow \mathcal{Y}$ that maps an instance $\mathbf{x}$ into an element y of $\mathcal{Y}$. In [14], Crammer and Singer started from considering the classifier of the form

$$H_M(\mathbf{x}) = \arg \max_{1 \le r \le m} \{M_r \cdot \mathbf{x}\} \tag{2.1}$$

Where $M \in \Re^{m \times n}$ and M_r is the rth row of M. The value of inner-product of M_r with the sample $\mathbf{x}$ is called the similarity score for the r class. Therefore, according to the classifier, the predicted label is the class attaining the highest similarity score with $\mathbf{x}$. Crammer and Singer generalized the conception of margin from SVMs and proposed the following quadratic multiclass vector machine model

$$\min_{M,\xi} \quad \tfrac{1}{2}\beta\|M\|_2^2 + \textstyle\sum_{i=1}^m \xi_i \tag{2.2a}$$

$$s.t. \quad \max_{r=1}^m\{M_r \cdot \mathbf{x}_i + 1 - \delta_{y_i,r}\} - M_{y_i} \cdot \mathbf{x}_i = \xi_i, \quad \forall i = 1, \cdots, l. \tag{2.2b}$$

When there are only two classes(i.e. $m = 2$), the above formulation reduces to the primal form of binary SVM by setting $\mathbf{w} = M_1 - M_2$ and $C = \beta^{-1}$.

Through taking the Lagrangian dual of problem (2.2) and replacing the inner-product with the general kernel function, Crammer and Singer gave the dual form of their model as following[4], which is the optimization problem we focus on in this work

$$\min_{\alpha} \quad \tfrac{1}{2}\textstyle\sum_{i,j=1}^l K(\mathbf{x}_i, \mathbf{x}_j)(\alpha_i^T \alpha_j) - \beta \sum_{i=1}^l \alpha_i^T e_{y_i} = f(\alpha) \tag{2.3a}$$

$$s.t. \quad \alpha_i^T \bar{1} = 0, \forall i = 1, \cdots, l; \tag{2.3b}$$

$$\alpha_i \le e_{y_i}, \forall i = 1, \cdots, l; \tag{2.3c}$$

where $\bar{1} \in \Re^l$ is a vector with all elements 1, $e_i \in \Re^l$ presents the ith unit vector. $K(\mathbf{x}_i, \mathbf{x}_j)$ is the (i, j)th element of kernel matrix K. Sometimes for simplicity, we write it as $K_{i,j}$. If the kernel function $K(\cdot, \cdot)$ satisfy Mercer Condition [32], K is positive semidefinite.

Suppose the optimal solution of problem (2.3) is α^*. Then the classifier is

$$H(\mathbf{x}) = \arg \max_{1 \le r \le m} \{\sum_{i=1}^l \alpha_{i,r} K(\mathbf{x}, \mathbf{x}_i)\} \tag{2.4}$$

[4]The detailed derivation can be found in [14]

If $\alpha_i^* \neq 0$, the corresponding sample i is called a Support Vector(SV). Furthermore, if $\alpha_{y_i,i}^* = 1$, it is called a Bounded Support Vector(BSV).

Readers may ask since model (2.3) is only a quadratic programming(QP) and there already exist several QP solvers, why we design a new algorithm for it. One of the main reasons is that the kernel matrix K is always dense and its size depends on the number of training samples. On one hand, prohibitively large memory is required to store K for large-scale problem. Take a problem with $60,000$ training samples as an example(this is exactly the scale of test problem "mnist", which will be used in our numerical experiment in Section 5), the required space to store K only in single precision will be $13.41GB$, which may be too large to be fitted into the memory. On the hand hand, the cost to recompute the matrix every time when it is needed is also extremely expensive. This difficulty prevents us from using off-the-shell quadratic solvers which require explicit storage of second order coefficient matrix in the objective function. In fact, this virtually happens for all the kernel based learning algorithm.

The decomposition method was first proposed to overcome the memory limitation for the standard binary SVM [34], which has become one of the most popular algorithms [36, 24, 10, 8, 49]. The basic idea is that since all the variables are very difficult to be updated altogether, only part of variables are chosen to be optimized at each iteration. The index set corresponding to these selected variables is called the working set. Different decomposition algorithms distinguish with each other on how to select the working sets, how to solve the subproblems and how to design the stopping criteria.

2.1 The Basic Decomposition Framework

To describe in detail the decomposition algorithm for Crammer and Singer's model (2.3) we need more notations. The gradient of the objective function (2.3a) is

$$F_{i,r}(\alpha) = \frac{\partial f(\alpha)}{\partial \alpha_{i,r}} = \sum_{j=1}^{m} K_{i,j}\alpha_{j,r} - \beta\delta_{y_i,r}, \ \forall\ i = 1, \cdots, l; r = 1, \cdots, m. \tag{2.5}$$

Here $\delta_{\cdot,\cdot}$ follows the general definition for delta function(i.e. $\delta_{i,j} = 1$ if $i = j$; otherwise $\delta_{i,j} = 0$) and $\delta_{y_i,r}$ is the rth element of vector e_{y_i}. At each decomposition iteration, the indices of the variables α_i, $i = 1, \cdots, l$, are split into the set $\mathcal{B}$ of *basic* variables(this is the working set mentioned above), and the set $\mathcal{N} = \{1, 2, \cdots, l\}\backslash\mathcal{B}$ of *nonbasic* variables. $n_{\mathcal{B}}$ is the maximum size of the working set, i.e. $\#\mathcal{B} \leq n_{\mathcal{B}}$ for all the iterations. (k) and $*$ are used to superscript the values at the k-th iteration and the values corresponding to the optimal solution, respectively. Then the quadratic subproblem at the k-th iteration can be written as:

$$\min_{\{\alpha_i\}_{i\in\mathcal{B}^{(k)}}} \quad \tfrac{1}{2}\textstyle\sum_{i,j\in\mathcal{B}^{(k)}} K_{i,j}\alpha_i^T\alpha_j + \sum_{i\in\mathcal{B}^{(k)}} \alpha_i^T(F_i^{(k)} - \sum_{j\in\mathcal{B}^{(k)}} K_{i,j}\alpha_j^{(k)}) \tag{2.6a}$$

$$s.t. \quad \alpha_i^T\bar{1} = 0, \forall i \in \mathcal{B}^{(k)}; \tag{2.6b}$$

$$\alpha_i \leq e_{y_i}, \forall i \in \mathcal{B}^{(k)}; \tag{2.6c}$$

The framework of the decomposition method for problem (2.3) is given in Algorithm 1. In this frame, the iterates start from a feasible point of (2.3). In each loop of the algorithm, a new working set is selected based on some given strategy; The quadratic subproblem is constructed and solved to update the values of variables in the working set, while fixing other variables to their current values; The stopping criteria is checked to determine whether we should terminate the iterative process. Among these steps, How to choose working set is one of the key components for the decomposition method. In fact, it does not only affect the efficiency of the algorithm, but also the convergence of the iterates. The stopping criteria are usually designed accompany with the working set selection rule. Usually, good stopping criteria can save unnecessary computing time and prevent the occurrence of the phenomenon of early termination. Therefore, we concentrate on present our new working set selection rule and corresponding stopping criteria in the next subsection.

Algorithm 1 Decomposition Method Framework

```
Step 0. Initialization. Input $1 \le n_{\mathcal{B}} \le l$, $\mathcal{B}^{(0)}$, $\alpha^{(0)} \in R^{l\times m}$ and $k = 0$.
Step 1. Quadratic Subproblem Solution. Obtain $\{\bar{\alpha}_i\}_{i\in\mathcal{B}^{(k)}}$ by solving (2.6). Set
    $$\alpha_i^{(k+1)} = \begin{cases} \bar{\alpha}_i & \text{if } i \in \mathcal{B}^{(k)}, \\ \alpha_i^{(k)} & \text{if } i \notin \mathcal{B}^{(k)}. \end{cases}$$
Step 2. Gradient Update. Compute
    $$F_{i,r}^{(k+1)} = F_{i,r}^{(k)} + \sum_{j\in\mathcal{B}^{(k)}} K_{i,j}(\alpha_{j,r}^{(k+1)} - \alpha_{j,r}^{(k)}),\ \forall\ i = 1,\cdots,l; r = 1,\cdots,m. \quad (2.8)$$
    Set $k := k + 1$.
Step 3. Working Set Selection. If stopping criteria is satisfied then output $\alpha^{(k)}$ and stop; Otherwise choose at most $n_B$ indices to form $\mathcal{B}^{(k)}$ and go to Step 1.
```

2.2 The Basic Working Set Selection Rule

When selecting working set, it is desirable for us to choose the set of variables which can make the largest progress towards the optimal objective function value. This aim can be obtained by taking $\arg\min_{\mathcal{B}:|\mathcal{B}|=n_{\mathcal{B}}} Qsub(\alpha^{(k)}, \mathcal{B})$ as working set, where $Qsub(\alpha^{(k)}, \mathcal{B})$ is the minimum of the following QP:

$$\begin{aligned} \min_d \quad & f(\alpha^{(k)} + d) - f^{(k)} && (2.9a)\\ s.t. \quad & d_i^T \bar{1} = 0 \qquad \forall i \in \mathcal{B}; && (2.9b)\\ & d_i \le e_{y_i} - \alpha_i^{(k)} \qquad \forall i \in \mathcal{B}; && (2.9c)\\ & d_i = 0, \qquad \forall i \notin \mathcal{B}; && (2.9d) \end{aligned}$$

However this choice requires solving $\binom{l}{n_{\mathcal{B}}}$ QPs (2.9), which is computationally very expensive. To reduce the expensive computational cost, we propose to remove the quadratic term in the objective function resulting in the following linear programming problem:

$$\begin{aligned} \min_d \quad & \textstyle\sum_{i\in\mathcal{B}} d_i^T F_i^{(k)} && (2.10a)\\ s.t. \quad & d_i^T \bar{1} = 0, \qquad \forall i \in \mathcal{B}; && (2.10b)\\ & d_i \le e_{y_i} - \alpha_i^{(k)}, \quad \forall i \in \mathcal{B}; && (2.10c)\\ & d_i = 0 \qquad \forall i \notin \mathcal{B}. && (2.10d) \end{aligned}$$

Denote the minimum of the above linear programming (LP) problem as $Lsub(\alpha^{(k)}, \mathcal{B})$. Then $\arg\min_{\mathcal{B}:|\mathcal{B}|=n_{\mathcal{B}}} Lsub(\alpha^{(k)}, \mathcal{B})$ is also a good candidate for working set $\mathcal{B}^{(k)}$. Grouping constraints

(2.10b)-(2.10c) by samples, problem (2.10) can be divided into $\#\mathcal{B}$ independent LPs. Namely, for each $i \in \mathcal{B}$, we solves

$$\begin{aligned} \min_{\bar{d}_i} \quad & d_i^T F_i^{(k)} && \text{(2.11a)} \\ s.t. \quad & d_i^T \bar{1} = 0; && \text{(2.11b)} \\ & d_i \leq e_{y_i} - \alpha_i^{(k)}. && \text{(2.11c)} \end{aligned}$$

Let $z_i = e_{y_i} - \alpha_i^{(k)} - d_i$, it can be rewritten as

$$\begin{aligned} \min_{\bar{z}_i} \quad & -z_i^T F_i(\alpha^{(k)}) + (e_{y_i} - \alpha_i^{(k)})^T F_i(\alpha^{(k)}) \\ s.t. \quad & z_i^T \bar{1} = 1; \\ & z_i \geq 0. \end{aligned}$$

Obviously, the solution is $z_i = e_p$, where $p \in \arg\max_r F_{i,r}^{(k)}$. Therefore,

$$\bar{d}_i = e_{y_i} - \alpha_i^{(k)} - e_p,$$

is the solution of LP (2.10). The corresponding optimal objective function value is

$$\bar{d}_i^T F_i^{(k)} = \sum\nolimits_{\{r, F_{i,r}^{(k)} < \max_s F_{i,s}^{(k)}\}} (F_{i,r}^{(k)} - \max_s F_{i,s}^{(k)})(\delta_{y_i,r} - \alpha_{i,r}^{(k)}).$$

Define

$$\ell_i(\alpha) = \sum\nolimits_{\{r, F_{i,r} < \max_s F_{i,s}\}} (F_{i,r}(\alpha) - \max_s F_{i,s}(\alpha))(\delta_{y_i,r} - \alpha_{i,r}), \forall i = 1, \cdots, l. \quad \text{(2.12)}$$

Then,

$$Lsub(\alpha^{(k)}, \mathcal{B}) = \sum\nolimits_{i \in \mathcal{B}} \ell_i^{(k)}$$

Hence, $\arg\min_{\mathcal{B}:|\mathcal{B}|=n_{\mathcal{B}}} Lsub(\alpha^{(k)}, \mathcal{B})$ can be chosen by taking the indices corresponding to the $n_{\mathcal{B}}$ smallest $\ell_i^{(k)}$ values defined by (2.12). And the iterations will be naturally terminated when $\ell^{(k)} = 0$. We end this subsection by summarizing this working set selection rule in Algorithm 2.

Algorithm 2 Basic Working Set Selection Rule

```
Compute $\ell^{(k)}$ by (2.12).
if $\ell^{(k)} \neq 0$, then
    Sort $\ell^{(k)}$ in increasing order; Choose the first $n_{\mathcal{B}}$ indices as $\mathcal{B}^{(k)}$.
else
    Output $\alpha^{(k)}$ as the optimal solution and stop.
end if
```

2.3 Global Convergence Proof

In this subsection, we prove the global convergence of Algorithm 1 where the working set is selected by Algorithm 2.

Lemma 2.1. *For a feasible point α of problem (2.3), the vector $\ell(\alpha)$ vanishes if and only if α is a KKT point of problem (2.3).*

Proof. First we prove that $\ell(\alpha)$ vanishes at any KKT point. Suppose α is a KKT point of (2.3). From the Kuhn-Tucker theory[41], there exist Lagrangian multipliers u_i and v_i such that

$$F_{i,r}(\alpha) - v_i + u_{i,r} = 0, \qquad \forall i, r; \tag{2.13a}$$

$$u_{i,r}(\alpha_{i,r} - \delta_{y_i,r}) = 0, \quad \forall i, r; \tag{2.13b}$$

$$u_{i,r} \geq 0, \quad \forall i, r; \tag{2.13c}$$

hold. Because α is a feasible point of problem (2.3), for any $i = 1, \cdots, l$, there must exist an index $\hat{r}$ such that

$$\alpha_{i,\hat{r}} - \delta_{y_i,\hat{r}} < 0.$$

Otherwise, equality constraints (2.3b) will be violated. Furthermore, from (2.13), we can get

$$u_{i,\hat{r}} = 0 \text{ and } v_i = F_{i,\hat{r}}(\alpha) = \max_s F_{i,s}(\alpha).$$

For any i and r, if

$$F_{i,r}(\alpha) < v_i,$$

we have that

$$u_{i,r} > 0 \text{ and } \alpha_{i,r} - \delta_{y_i,r} = 0.$$

Thus, by the definition of $\ell(\alpha)$ in (2.12), it follows that

$$\ell_i(\alpha) = 0, \quad \forall i = 1, \cdots, l.$$

Now we prove a feasible point α satisfying $\ell(\alpha) = 0$ is also a KKT point of (2.3). From the definition of $\ell(\alpha)$ in (2.12), we know if

$$F_{i,r}(\alpha) < \max_s F_{i,s}(\alpha),$$

there must be

$$\delta_{y_i,r} - \alpha_{i,r} = 0.$$

So we can let

$$v_i = \max_s F_{i,s}(\alpha) \text{ and } u_{i,r} = v_i - F_{i,r}(\alpha), \quad \forall i = 1, \cdots, l; r = 1, \cdots, m.$$

It can be easily verified that α, u and v satisfy (2.13). This implies that α is a KKT point of (2.3). □

Lemma 2.2. *Suppose α is a feasible point of problem (2.3) and kernel function $K(\cdot, \cdot)$ satisfies Mercer condition, then for any index $p \in \{1, \cdots, l\}$*

$$Qsub(\alpha, p) \leq \frac{\ell_p(\alpha)}{2} \min\{1, \frac{-\ell_p(\alpha)}{4M}\},$$

where $M = \max_{i=1}^{l}\{K_{i,i} + 1\}$.

Proof. Consider the following linear programming subproblem $LP(\alpha, p)$:

$$\begin{aligned} \min_{d_p} \quad & d_p^T F_p(\alpha) \\ s.t. \quad & d_p^T \bar{1} = 0;\ d_p \le e_{y_p} - \alpha_p. \end{aligned}$$

Let $s \in \arg\max_r F_{i,r}(\alpha)$. Following the discussion for problem (2.10) in the last subsection, we know the optimal objective function value of $LP(\alpha, p)$ is $\ell_p(\alpha)$ and the optimal solution is

$$\bar{d}_p = e_{y_p} - \alpha_p - e_s,$$

Since $\|e_s\|_1 = 1$, $e_{y_p} - \alpha_p \ge 0$ and $\bar{1}^T(e_{y_p} - \alpha_p) = 1$, we have

$$\|\bar{d}_p\|_1 = \|e_{y_p} - \alpha_p - e_s\|_1 \le \|e_{y_p} - \alpha_p\|_1 + \|e_s\|_1 = \bar{1}^T(e_{y_p} - \alpha_p) + 1 = 2.$$

Because $Qsub(\alpha, p)$ is the objective function value for the following QP by definition,

$$\begin{aligned} \min_d \quad & f(\alpha + d) - f(\alpha) \\ s.t. \quad & d_i^T \bar{1} = 0,\ d_i \le e_{y_i} - \alpha_i, \quad i = p; \\ & d_i = 0, \qquad\qquad\qquad i \ne p; \end{aligned}$$

which can be simplified as

$$\begin{aligned} \min_{d_p} \quad & d_p^T F_p(\alpha) + \tfrac{1}{2} K_{p,p} d_p^T d_p \\ s.t. \quad & d_p \bar{1} = 0;\ d_p \le e_{y_p} - \alpha_p. \end{aligned}$$

and the line segment $\{t\bar{d}_p \mid t \in [0,1]\}$ lies in its feasible region, we have

$$Qsub(\alpha^{(k)}, p) \le \min_{t\in[0,1]}\{\bar{d}_p^T F_p(\alpha)t + \tfrac{1}{2}K_{p,p}\bar{d}_p^T \bar{d}_p t^2\} = \min_{t\in[0,1]}\{\ell_p(\alpha)t + \tfrac{1}{2}K_{p,p}\|\bar{d}_p\|_2^2 t^2\}$$

Since kernel function $K(\cdot,\cdot)$ satisfies the mercer condition, $K_{p,p} \ge 0$

$$\begin{aligned} \min_{t\in[0,1]}\{\ell_p(\alpha)t + \tfrac{1}{2}K_{p,p}\|\bar{d}_p\|_2^2 t^2\} &\le \min_{t\in[0,1]}\{\ell_p(\alpha)t + \tfrac{1}{2}(K_{p,p}+1)\|\bar{d}_p\|_2^2 t^2\} \\ &\le \min_{t\in[0,1]}\{\ell_p(\alpha)t + \tfrac{1}{2}(K_{p,p}+1)\|\bar{d}_p\|_1^2 t^2\} \\ &\le \min_{t\in[0,1]}\{\ell_p(\alpha)t + 2Mt^2\} \\ &\le \tfrac{1}{2}\ell_p(\alpha)t^* \end{aligned}$$

where,

$$t^* = \min\{1, \frac{-\ell_p(\alpha^{(k)})}{4M}\}.$$

Hence, we have

$$Qsub(p, \alpha^{(k)}) \le \frac{1}{2}\ell_p(\alpha^{(k)}) \min\{1, \frac{-\ell_p(\alpha^{(k)})}{4M}\}.$$

□

Theorem 2.3. *Suppose the kernel function $K(\cdot,\cdot)$ satisfies Mercer condition, $n_{\mathcal{B}} \ge 1$ and $\{\alpha^{(k)}\}$ is the sequences generated by Algorithm 1 and working set selection rule Algorithm 2. If $\{\alpha^{(k)}\}$ has only finite many elements, the last one is a KKT point of problem (2.3). If it has infinite many elements, any accumulation point of the sequence $\{\alpha^{(k)}\}$ is a KKT point of (2.3).*

Proof. If the sequence $\{\alpha^{(k)}\}$ has only finite many elements, the last iteration point must satisfy $\ell(\alpha^{(k)}) = 0$. By Lemma 2.1, it is a KKT point.

For the rest of the proof, we assume that Algorithm 1 generates an infinite sequence $\{\alpha^{(k)}\}$. Suppose $\bar{\alpha}$ is an accumulation point of the sequence. By relabeling $\{\alpha^{(k)}\}$ if necessary, we can assume that $\{\alpha^{(k)}\}$ converges to $\bar{\alpha}$. Because the feasible region of problem (2.3) is bounded in $R^{l\times m}$ and the iteration points generated by algorithm 2 are always feasible, $\bar{\alpha}$ is still in feasible region and $f(\bar{\alpha})$ has a finite value. Let p is the index such that

$$\ell_p^{(k)} = \min_{1\le i\le l} \ell_i^{(k)} = -\|\ell^{(k)}\|_\infty.$$

The last equality is based on the definition of ℓ_∞ norm and the fact that all the elements of ℓ is negative. Because Algorithm 2 chooses the samples corresponding to the smallest $n_{\mathcal{B}}$ elements in $\ell^{(k)}$ and $n_{\mathcal{B}} \ge 1$, index p must be contained in the working set $\mathcal{B}^{(k)}$. Hence

$$f^{(k+1)} - f^{(k)} \le Qsub(\alpha^{(k)}, p). \tag{2.14}$$

From lemma 2.2,

$$f^{(k+1)} - f^{(k)} \le \frac{1}{2}\ell_p^{(k)} \min\{1, -\frac{\ell_p^{(k)}}{4M}\} = -\frac{1}{2}\|\ell^{(k)}\|_\infty \min\{1, \frac{\|\ell^{(k)}\|_\infty}{4M}\}$$

Sum it from 0 to s we get

$$\frac{1}{2}\sum_{k=0}^{s}\|\ell^{(k)}\|_\infty \min\{1, \frac{\|\ell^{(k)}\|_\infty}{4M}\} \le f^{(0)} - f^{(s+1)}.$$

Let $s \to \infty$, we have

$$\frac{1}{2}\sum_{k=0}^{\infty}\|\ell^{(k)}\|_\infty \min\{1, \frac{\|\ell^{(k)}\|_\infty}{4M}\} \le f^{(0)} - f(\bar{\alpha}) < +\infty.$$

Consequently,

$$\|\ell(\bar{\alpha})\|_\infty = \lim_{k\to+\infty} \|\ell(\alpha^{(k)})\|_\infty = 0.$$

This shows that $\bar{\alpha}$ is a KKT point of (2.3). □

3 Projected Gradient for Quadratic Subproblems

Another important part that affects the efficiency of decomposition method is the solving of QP subproblems (2.6). In order to make the algorithm more efficient, instead of using the off-the-shell QP solvers [7, 44, 33, 21], we consider developing a special projected gradient(PG) method, whose projection step is specially designed for constraints (2.6b)-(2.6c). Without loss of generality, we assume $\mathcal{B}^{(k)} = \{1, 2, \cdots, n_{\mathcal{B}}\}$ and use w instead of α to denote the variables in this section. Then subproblem (2.6) can be rewritten as:

$$\min_w \quad \tfrac{1}{2}\textstyle\sum_{i,j=1}^{n_{\mathcal{B}}} K_{i,j} w_i^T w_j + \sum_{i=1}^{n_{\mathcal{B}}} w_i^T b_i = \hat{f}(w) \tag{3.1a}$$

$$s.t. \quad w_i \le e_{y_i}, \forall i = 1, \cdots, n_{\mathcal{B}}; \tag{3.1b}$$

$$w_i^T \bar{1} = 0, \forall i = 1, \cdots, n_{\mathcal{B}}; \tag{3.1c}$$

where $b_i = F_i^{(k)} - \sum_{j=1}^{n_{\mathcal{B}}} K_{i,j}\alpha_j^{(k)}, \forall i = 1, \cdots, n_{\mathcal{B}}$.

Theoretically, the projected gradient method can be applied to any problems whose feasible region are convex[41]. However, because of the high cost for computing the projection to a general convex set, special techniques have been studied to projected gradient algorithms to special problems, such as the bound constrained problem or bound constrained problem with singly linear constraint [16, 35].

For our subproblem (3.1), the feasible region

$$\Omega = \{w | w_i \le e_{y_i}, w_i^T \bar{1} = 0, \forall i = 1, \cdots, n_{\mathcal{B}}\}$$

is a convex set, which is composed of $n_{\mathcal{B}}$ linear constraints together with upper and lower bounds. There is no ready-made projection routine can be used. So firstly we need to derive an efficient method for computing the projection. Let $P_\Omega(\cdot)$ be the projection function on Ω and $\hat{w} \in R^{m \times n_{\mathcal{B}}}$, $P_\Omega(\hat{w})$ is the solution of the following QP

$$\begin{aligned} \min_{\{w_i\}_{i\in\mathcal{B}}} \quad & \tfrac{1}{2}||w - \hat{w}||_2^2 = \tfrac{1}{2}\textstyle\sum_{i\in\mathcal{B}} ||w_i - \hat{w}_i||_2^2 && (3.2a) \\ s.t. \quad & w_i^T \bar{1} = 0, \forall i = 1, \cdots, n_{\mathcal{B}}, && (3.2b) \\ & w_i \le e_{y_i}, \forall i = 1, \cdots, n_{\mathcal{B}}. && (3.2c) \end{aligned}$$

Separate (3.2) into $n_{\mathcal{B}}$ individual subproblems, we have

$$\begin{aligned} \min_{w_i} \quad & \tfrac{1}{2}||w_i - \hat{w}_i||_2^2 = \tfrac{1}{2}\textstyle\sum_{r=1}^{m} (w_{i,r} - \hat{w}_{i,r})^2 && (3.3a) \\ s.t. \quad & w_i^T \bar{1} = 0; && (3.3b) \\ & w_i \le e_{y_i}. && (3.3c) \end{aligned}$$

Suppose we know the Lagrangian multiplier λ_i for the equality constraint (3.3b), the above problem can be translated as

$$\begin{aligned} \min_{w_i} \quad & \frac{1}{2}||w_i - \hat{w}_i||_2^2 - \lambda_i w_i^T \bar{1} \\ s.t. \quad & w_i \le e_{y_i}. \end{aligned}$$

Dropping the constant term in the objective function, we get

$$\begin{aligned} \min_{w_i} \quad & \textstyle\sum_{r=1}^{m} \{\tfrac{1}{2} w_{i,r}^2 - (\hat{w}_{i,r} + \lambda_i) w_{i,r}\} \\ s.t. \quad & w_i \le e_{y_i}. \end{aligned}$$

This problem can be further divided into m one dimensional subproblems,

$$\begin{aligned} \min_{w_{i,r}} \quad & \tfrac{1}{2} w_{i,r}^2 - (\hat{w}_{i,r} + \lambda_i) w_{i,r} \\ s.t. \quad & w_{i,r} \le \delta_{y_i,r}. \end{aligned}$$

The minimizer is

$$w_{i,r}^*(\lambda_i) = \min\{\hat{w}_{i,r} + \lambda_i, \delta_{y_i,r}\}$$

Substitute it back into the linear constraint in (3.3), we get the following piecewise linear equation for λ_i,

$$\bar{1}^T w_i^*(\lambda_i) = \textstyle\sum_{r=1}^{m} (\min\{\hat{w}_{i,r} + \lambda_i, \delta_{y_i,r}\}) = 0, \qquad (3.4)$$

which can be solved in linear time [16, 35]. Denote the optimal solution as λ_i^*, we have

$$P_\Omega(\hat{w})^T = (w_i^*(\lambda_i^*))_{i=1,\cdots,n_\mathcal{B}}. \tag{3.5}$$

Other two things need to be mentioned are our special terminating condition and the choice of initial steplength. The gradient of objective function (3.1) is

$$\hat{F}_{i,r}(w) = \frac{\partial \hat{f}(w)}{\partial w_{i,r}} = \sum\nolimits_{j\in\mathcal{B}} K_{i,j} w_{j,r} + F_{i,r}^{(k)} - \sum\nolimits_{j\in\mathcal{B}} K_{i,j}\alpha_{j,r}^{(k)}, \forall i = 1,\cdots,n_\mathcal{B}, r = 1,\cdots m. \tag{3.6}$$

Define

$$\hat{\ell}_i(w) \equiv \sum\nolimits_{\{r,\hat{F}_{i,r}(w)<\max_r \hat{F}_{i,r}(w)\}} (\hat{F}_{i,r}(w) - \max_r \hat{F}_{i,r}(w))(\delta_{y_i,r} - w_{i,r}),\ \forall\ i = 1,\cdots,n_\mathcal{B}.$$

Follow the discussion for (2.3) in Section 2, we can easily get the conclusion that w is an optimal solution of (3.1) if and only if $\hat{\ell}(w) = 0$. So the iteration is terminated when $\|\hat{\ell}(w)\|$ is less than a predefined tolerance $\epsilon > 0$. $\hat{\ell}(w)$ is also used to construct the initial steplength. For example, the first step size can be taken as $\|\hat{\ell}(w^{(0)})\|_\infty^{-1}$ or $\|\hat{\ell}(w^{(0)})\|_1^{-1}$. Especially, for all $i \in \mathcal{B}^{(k)}, r = 1,\cdots,m$, we have

$$\hat{F}_{i,r}(\{\alpha_i^{(k)}\}_{i\in\mathcal{B}^{(k)}}) = F_{i,r}^{(k)} \text{ and } \hat{F}_{i,r}(\{\alpha_i^{(k+1)}\}_{i\in\mathcal{B}^{(k)}}) = F_{i,r}^{(k+1)}.$$

So if $\{\alpha_i^{(k)}\}_{i\in\mathcal{B}^{(k)}}$ is inherited as initial point for the inner QP solver, we can get the initial step length from $\ell^{(k)} = \hat{\ell}(\{\alpha_i^{(k)}\}_{i\in\mathcal{B}^{(k)}})$ directly without any extra computation.

The overall algorithm is described in Algorithm 3, which follows the framework of PG method for general problems with convex feasible region in [3]. Specifically, after getting the projection point at each iteration, we use the steplength computation method and the adaptive nonmonotone strategy proposed in [16] to accelerate the convergence of the PG method.

Algorithm 3 Projected Gradient Method for Quadratic Subproblems

```
**Step 0. Initialization.** Input $\epsilon$, $w^{(0)} \in \Omega$; Compute $\hat{f}(w^{(0)})$, $\hat{F}(w^{(0)})$ and $\hat{\ell}(w^{(0)})$; Set $0 < \rho_{min} < \rho_{max}$, integer $h \geq 0$ and $U > 0$; Let $\hat{f}_{best} = \hat{f}_c = \hat{f}(w^{(0)})$, $f_{ref} = \infty$, $\rho^{(0)} = \|\hat{\ell}(w^{(0)})\|_\infty^{-1}, s^{(-1)} = y^{(-1)} = 0, u = 0$, and $q = 0$.
**Step 1. Termination Test.** If $\|\hat{\ell}(w^{(q)})\|_\infty \leq \epsilon$, return $w^{(q)}$ and stop.
**Step 2. Projection Computation.** $\hat{w} = w^{(q)} - \rho^{(q)}\hat{F}(w^{(q)})$. Obtain $\lambda_i$ by solving (3.4) for $i = 1, \cdots, n_{\mathcal{B}}$. Compute
    $$P_\Omega(\hat{w}) = (\min\{\hat{w}_{i,r} + \lambda_i, \delta_{y_i,r}\})_{i=1,\cdots,n_{\mathcal{B}},r=1,\cdots,m}.$$
**Step 3. New Iteration Point Selection.** If $\hat{f}(P_\Omega(\hat{w})) \leq \hat{f}_{ref}$, set $w^{(q+1)} = P_\Omega(\hat{w})$;
Otherwise, set $d^{(q)} = P_\Omega(\hat{w}) - w^{(q)}$, $\theta^{(q)} = \arg\min_{\theta\in[0,1]} \hat{f}(w^{(q)} + \theta d^{(q)})$, $w^{(q+1)} = w^{(q)} + \theta^{(q)} d^{(q)}$.
**Step 4. Steplength Calculation.** $s^{(q)} = w^{(q+1)} - w^{(q)}$, $y^{(q)} = \hat{F}(w^{(q+1)}) - \hat{F}(w^{(q)})$.
Compute $h^{(q)} = \min\{h, q, \arg\min_i\{{s^{(q-i)}}^T y^{(q-i)} \leq 0\} - 1\}$.
    $$\rho^{(q+1)} = \begin{cases} \rho_{max} & \text{if } h^{(q)} = -1; \\ \min\{\rho_{max}, \max\{\rho_{min}, \frac{\sum_{i=0}^{h^{(q)}} {s^{(q-i)}}^T s^{(q-i)}}{\sum_{i=0}^{h^{(q)}} {s^{(q-i)}}^T y^{(q-i)}}\}\} & \text{otherwise.} \end{cases}$$
**Step 5. Parameter Update.** If $\hat{f}^{(q)} < \hat{f}_{best}$, set $\hat{f}_{best} = \hat{f}^{(q)}, \hat{f}_c = \hat{f}^{(q)}, u = 0$;
Otherwise, $\hat{f}_c := \max\{\hat{f}_c, \hat{f}^{(q)}\}, u := u + 1$.
If $u = U$, set $\hat{f}_{ref} = \hat{f}_c, \hat{f}_c = \hat{f}^{(q)}$ and $u = 0$.
Set $q := q + 1$ and go to Step 1.
```

4 Efficient Implementation

4.1 Practical Working Set Selection Rule

When applying the working set selection rule in Algorithm 2 directly to the decomposition framework in Algorithm 1, zigzagging phenomenon(some variables enters and leaves the working set many times) occurs, which results in slow convergence. We use the ad-hoc technique of keeping part of indices from previous working sets[24, 48] to avoid zigzagging in practice. To be more precise, at most n_N new indices are allowed to enter into the working set at each iteration, where $1 \le n_N \le n_{\mathcal{B}}$. Other indices are taken from the working set of the last iteration. Now our problem becomes which indices we should inherit to make the algorithm more efficient?

For problem (2.3), if a proper kernel type is chosen, most of the samples are not SVs. Usually the less SVs there are, the faster the decomposition algorithm converges. Based on this observation, we should try to keep the number of SVs in the training result as small as possible. Furthermore, for the problem with noise in training data, most of SVs are BSVs. And the optimal values for these BSVs can always be settled down much earlier than the iteration terminates. So our first set of candidates for filling working set are the indices whose corresponding variables satisfy $0 < \alpha_{i,y_i}^{(k)} < 1$. And the last set of choice are indices of samples which are recognized as BSVs currently.

Another principle for us to inherit indices is that we should always believe in our computation results. That is to say, if a variable has been consecutively optimized many times(i.e. its corresponding index stays in the working set many times), we should have more confidence in the current value of this variable to be optimal. So the indices staying in working set with small number of consecutive iterations are selected into the working set again with high priority. The whole revised working set selection rule is given in Algorithm 4.

Algorithm 4 Practical Working Set Selection Rule

```
Compute $\ell(\alpha^{(k)})$ by (2.12).
if $\|\ell(\alpha^{(k)})\|_\infty > \beta\epsilon$ then
  Set $\hat{\mathcal{B}} = \mathcal{B}^{(k)}$ and $\mathcal{B}^{(k+1)} = \emptyset$.
  Sort $\{1, \cdots, l\}$ by $\{\ell_i(\alpha^{(k)})\}_{i=1}^l$ in increasing order and add the first $n_N$ indices to $\mathcal{B}^{(k+1)}$.
  Set $\hat{\mathcal{B}} := \hat{\mathcal{B}} \backslash \mathcal{B}^{(k+1)}$ and divide $\hat{\mathcal{B}}$ into three sets as $\mathcal{C}_0 = \{i | i \in \hat{\mathcal{B}}, 0 < \alpha_{i,y_i}^{(k)} < 1\}$, $\mathcal{C}_1 = \{i | i \in \hat{\mathcal{B}}, \alpha_{i,y_i}^{(k)} = 0\}$ and $\mathcal{C}_2 = \{i | i \in \hat{\mathcal{B}}, \alpha_{i,y_i}^{(k)} = 1\}$.
  for i=0,1,2 do
    while $\mathcal{C}_i \ne \emptyset$ and $\#\mathcal{B}^{(k+1)} < n_{\mathcal{B}}$ do
      Select the index $j$ with lowest number of consecutive iterations from $\mathcal{C}_i$.
      $\mathcal{B}^{(k+1)} := \mathcal{B}^{(k+1)} \cup \{j\}$ and $\mathcal{C}_i := \mathcal{C}_i \backslash \{j\}$.
    end while
  end for
  Return $\mathcal{B}^{(k+1)}$ as the new working set.
else
  Return $\alpha^{(k)}$ and stop.
end if
```

Because the inequality (2.14) still holds when $n_N \ge 1$. The global convergence proof in Section 2 also apply to Algorithm 4 by setting $\epsilon = 0$. To be more clarified, we state this conclusion in the following corollary.

Corollary 4.1. *Suppose the kernel function $K(\cdot,\cdot)$ satisfies Mercer condition, $n_{\mathcal{B}} \geq n_N \geq 1$ and $\{\alpha^{(k)}\}$ is the sequences generated by Algorithm 1 and the working set selection rule in Algorithm 4 with $\epsilon = 0$. If $\{\alpha^{(k)}\}$ has only finite many elements, the last one is a KKT point of problem (2.3). If it has infinite many elements, any accumulation point of the sequence $\{\alpha^{(k)}\}$ is a KKT point of (2.3).*

4.2 Kernel Evaluation

Kernel evaluation is one of the most time consuming tasks for training. In our decomposition framework described in Algorithm 1, kernel elements are used in both **Subproblem Solution** and **Gradient Update** steps. To construct the subproblem, $K_{\mathcal{B}^{(k)},\mathcal{B}^{(k)}}$ is required in (2.6), which contains $n_{\mathcal{B}}^2$ kernel elements. To update the gradient, $n_{\mathcal{B}}$ columns of kernel matrix are required in (2.5). For large scale problem($n_{\mathcal{B}} \ll l$), most of the kernel evaluations stem from the step of gradient update. Therefore, we focus on how to compute kernel efficiently when updating gradient and ignore the kernel evaluations in subproblem construction step. Three techniques are employed in all, which will be introduced in the next. One thing need to be stressed is that none of them is totally new. These basic ideas have been used in several SVM packages [24, 10, 8, 49]. Therefore, we only introduce the key parts for adapting these techniques to our special algorithm briefly instead of describing all the details.

For linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$, gradient updating formula (2.8) can be restated as

$$F_{i,r}(\alpha^{(k+1)}) = F_{i,r}(\alpha^{(k)}) + \mathbf{x}_i^T \textstyle\sum_{j \in B^{(k)}} (\alpha_{j,r}^{(k+1)} - \alpha_{j,r}^{(k)})\mathbf{x}_j,$$

which allows us to update gradient without calculating the kernels.

For nonlinear kernel, such as Gaussian or polynomial kernel, the kernel evaluation is inevitable anyway. We then try to reduce the kernel evaluation times as many as possible by the following two schemes. Firstly, a least-recent-used cache is maintained to avoid the recomputation of kernel elements ever being calculated. When the cache buffer is full, the elements being least recently used are removed from the buffer and the new calculated ones are added in. Secondly, even kernel elements $K_{\cdot,j}$ of index j are required in $(k+1)$th iteration and they are not cached, we still do not need to calculate this kernel column if the related dual variables are not changed in this step, i.e., $\alpha_{j,\cdot}^{(k+1)} = \alpha_{j,\cdot}^{(k)}$. This is because in gradient updating formulation (2.8), the individual item $K_{\cdot,j}(\alpha_{j,\cdot}^{(k+1)} - \alpha_{j,\cdot}^{(k)})$ of j in the sum is zero no matter what kernel elements $K_{\cdot,j}$ are.

4.3 Algorithm Parallelization

Two major steps in the decomposition Algorithm 1 are solving subproblem by the projected gradient method in Algorithm 3 and updating the gradient by (2.8). Both of these two steps are time consuming and, fortunately, suitable for parallelization. We illustrate how these steps can be parallelized in this subsection.

One of the major steps in Algorithm 3 is the projection computing. As derived in Section 3, the main task for projection is to solve $n_{\mathcal{B}}$ independent piecewise linear equations (3.4), which are naturally parallel with no communication required. This is the first thing reminding us to utilize the power of multiprocessors for accelerating the training process. Besides this, calculating and updating of the gradient(Step 4 of Algorithm 3) can be parallelized as well, whose brief idea is described as follows. According to (3.6), the gradient of the subproblem can be updated by

$$\hat{F}_{i,r}^{(q+1)} = \hat{F}_{i,r}^{(q)} + \sum\nolimits_{j\in\mathcal{B}} K_{i,j}(w_{j,r}^{(q+1)} - w_{j,r}^{(q)}), \forall i = 1,\cdots,n_{\mathcal{B}}, r = 1,\cdots,m. \tag{4.1}$$

To parallelize the gradient updating calculation, we first distribute the indices of working set $\mathcal{B}$ evenly to the processors. Suppose there are N_P processors under consideration. Denote the index set of processor p as I_p, which satisfies

$$\cup_{p=1}^{N_P} I_p = \{1,\cdots,n_{\mathcal{B}}\} \text{ and } I_p \cap I_{p'} = \emptyset, \forall p \neq p'.$$

Then each processor p calculates a slice of the whole incremental item in the righthand side of the equation (4.1), i.e.

$$v_p \equiv \left\{\sum\nolimits_{j\in\mathcal{B}} K_{i,j}(w_{j,r}^{(q+1)} - w_{j,r}^{(q)})\right\}_{i\in I_p}.$$

Then the final incremental item is obtained by assembling the slices from each of the processors. Note that all the kernel elements $K_{\mathcal{B},\mathcal{B}}$ are already calculated and stored in memory before solving the subproblem, and thus there is no kernel evaluation required for the gradient updating in this case. We summarize the complete description of the parallel PG method in Algorithm 5 and omit the details which are the same as the serial version in Algorithm 3 to highlight the parallelization strategy.

Algorithm 5 Parallel Projected Gradient Method for Quadratic Subproblems

```
Step 0. Initialization.
[PARALLELIZATION PART] Distribute indices $\{1,\cdots,n_{\mathcal{B}}\}$ to $N_P$ processors evenly.
Step 1. Termination Test.
Step 2. Projection Computation. Let $\hat{w} = w^{(q)} - \rho^{(q)}\hat{F}(w^{(q)})$.
[PARALLELIZATION PART] For $p = 1,\cdots,N_P$, solve (3.4) for $\lambda_i^{(q)}$ and compute

    $\breve{w}_i = (\min\{\hat{w}_{i,j} + \lambda_i^{(q)}, \delta_{y_i,j}\})_{j=1,\cdots,m} \ \forall i \in I_p.$

Assemble to get $P_\Omega(\hat{w}) = \breve{w} = (\breve{w}_i)_{i=1,\cdots,N_B}$
Step 3. New Iteration Point Selection.
[PARALLELIZATION PART] For $p = 1,\cdots,N_P$, compute $d_i^{(q)} = \breve{w}_i - w_i^{(q)}, \forall i \in I_p$.
If $\hat{f}(P_\Omega(\hat{w})) \leq \hat{f}_{ref}$, set $w^{(q+1)} = P_\Omega(\hat{w})$ and $\theta^{(q)} = 1$; Otherwise, calculate $\theta^{(q)}$, set $w^{(q+1)} = w^{(q)} + \theta^{(q)}d^{(q)}$.
Step 4. Steplength Calculation.
[PARALLELIZATION PART] For $p = 1,\cdots,N_P$, compute $v_p = \left\{\sum_{j=1}^{n_{\mathcal{B}}} K_{i,j}d_j^{(q)}\right\}_{i\in I_p}$ and send it to all the other processors. Update gradient by

    $\hat{F}_i^{(q+1)} = \hat{F}_i^{(q)} + \theta^{(q)}\sum_{j=1}^{n_{\mathcal{B}}} K_{i,j}d_j^{(q)}, \forall i = 1,\cdots,n_{\mathcal{B}}.$

Step 5. Parameter Update.
```

In Algorithm 1, updating the gradient by (2.8) is another time consuming job at each iteration. Therefore, the incremental item on the right side of equation (2.8) is what we consider to parallelize. We first distributed the working set $\mathcal{B}^{(k)}$ evenly to N_P processors. Each processor p then calculates the incremental item by summing over the subset I_p,

$$z_{i,p} \equiv \sum_{j \in I_p} K_{i,j}(\alpha_{j,r}^{(k+1)} - \alpha_{j,r}^{(k)}), \forall i = 1, \cdots, l.$$

The final incremental item is calculated by summing up the resulting z_p from all the processors. The major benefit from this parallel strategy is that the kernel evaluation can be distributed to multiple processors. Each processor calculates the required kernel elements and stores in its own cache. As we will see in the experiment section, this parallelization results in remarkable speedup.

With the parallel projected gradient algorithm and parallel gradient updating strategy, we summarize the detailed parallel implementation for the decomposition framework (Algorithm 1) in Algorithm 6.

```
Algorithm 6 Parallel algorithm for kernel-based multiclass training
  Step 0. Initialization. Input $1 \le n_{\mathcal{B}} \le l$, $N_P \ge 1$, $\mathcal{B}^{(0)}$ and $\alpha^{(0)} \in R^{l \times m}$; Set $k = 0$ and
  $\mathcal{N}^{(k)} = \{1, \cdots, m\} \backslash \mathcal{B}^{(k)}$. Initialize cache $W_p = \emptyset$ for $p = 1, \cdots, N_P$.
  Step 1. Quadratic Subproblem Solution.
  [PARALLELIZATION PART] For $p = 1, \cdots, N_P$, evaluate
      $(K_{i,j})_{i \in I_p, j=1,\cdots,n_{\mathcal{B}}}$ and $\{b_i = F_i^{(k)} - \sum_{j \in \mathcal{B}^{(k)}} K_{i,j}\alpha_j^{(k)}\}_{i \in I_p}$.
  Obtain $\{\bar{\alpha}\}_{i \in \mathcal{B}^{(k)}}$ by (3.1) in parallel by Algorithm 5. Set
      $\alpha_i^{(k+1)} = \begin{cases} \bar{\alpha}_i & \text{if } i \in \mathcal{B}^{(k)}, \\ \alpha_i^{(k)} & \text{if } i \notin \mathcal{B}^{(k)}. \end{cases}$
  Step 2. Gradient Update. $\mathcal{B}_n = \{i \in \mathcal{B}^{(k)} | \alpha_i^{(k+1)} \neq \alpha_i^{(k)}$and $K_{i,\cdot} \notin \cup_{p=1}^{N_P} W_p, \}$, $\mathcal{B}_c = \{i \in$
  $\mathcal{B}^{(k)} | \alpha_i^{(k+1)} \neq \alpha_i^{(k)}$and $K_{i,\cdot} \in \cup_{p=1}^{N_P} W_p, \}$. Distribute $\mathcal{B}_n$ and $\mathcal{B}_c$ evenly to $N_P$ processors with
  $\mathcal{B}_{n,p}$ and $\mathcal{B}_{c,p}$ as the subset belonging to processor $p$, respectively.
  [PARALLELIZATION PART] For $p = 1, \cdots, N_P$, evaluate kernel $K_{i,\cdot}$ for $i \in \mathcal{B}_{n,p}$ and store them
  to $W_p$ by the least-recently-used principle; calculate
      $z_{i,p} = \sum_{j \in (\mathcal{B}_{n,p} \cup \mathcal{B}_{c,p})} K(x_i, x_j)(\alpha_j^{(k+1)} - \alpha_j^{(k)}), \forall i = 1, \cdots, l.$
  Assemble $z^{(k)} = \sum_{p=1}^{N_P} z_p$ and update $F^{(k+1)} = F^{(k)} + z^{(k)}$. Set $k := k + 1$.
  Step 3. Working Set Selection. Test termination and choose $\mathcal{B}^{(k)}$ by Algorithm 4.
```

5 Numerical Experiments

We test the effectiveness and efficiency of the proposed algorithm in this section. All the experiments described are carried out on LSSC-II in the State Key Laboratory of Scientific and Engineering Computing, Chinese Academy of Sciences. This environment has 256 computational nodes. Each node is equipped with two 2GHz Xeon processors and 1GB memory. The serial code is run on one computational node. The parallel version is tested with up to 16 nodes.

Table 1: Statistics of small data set used in the experiments

Problem	#training	#test	#classes	#features	β	γ
vowel	528	462	11	10	2^{-1}	2^{3}
glass	214	N.A.	6	9	2^{-4}	2^{1}
satimage	4,435	2,000	6	36	2^{-2}	2^{2}
isolet	6,238	1,559	26	617	2^{-8}	2^{-6}
letter	20,000	N.A.	26	16	2^{-3}	2^{2}
shuttle	43,500	14,500	7	9	2^{-3}	2^{-9}
mnist	60,000	10,000	10	784	2^{-2}	2^{-5}
protein	67,557	N.A.	3	126	1	2^{-2}
connect-4	78,823	19,705	3	100	2^{-1}	2^{-4}
vechicle(combined)	17,766	6,621	3	357	2^{-3}	2^{-3}

Nine different data sets, which cover a wide range of applications, such as speech and hand written letters recognition, image classification, bioinformation and etc., are used in our experiment. Data statistics are list in Table 1. Among them, "protein" and "vehicle(combined)" are downloaded from `http://www.csie.nut.edu.tw/~cjlin/libsvmtools/datasets/`. Others are from UCI repository [2]. Many thanks to both authors for collecting these data and making them public available.

As mentioned in Section 1, Crammer and Singer's multiclass SVM model with linear kernel has been implemented in several well known packages. We focus on training the model with nonlinear kernels in this work. Therefore, we test the performance our algorithm with nonlinear kernels in this section. The most generally used RBF kernel $K(x_i, x_j) = e^{-\gamma\|x_i - x_j\|^2}$ are chosen to train models in the following experiments.

The regularization parameter β and kernel parameter γ are chosen by the tuning method proposed in [23]. For each data set, if its size is less than $10,000$, the whole training data set is used for tuning, otherwise we uniformly sample $5,000$ examples as the tuning set. Then, the resulting tuning data set is divided into two parts with 70% for training and 30% for validation. In the tuning process, the search range for the parameters β and γ are $\{2^{-12}, 2^{-11}, 2^{-10}, \cdots, 2^{0}, 2^{1}, 2^{2}\}$ and $\{2^{-10}, 2^{-9}, 2^{-8}, \cdots, 2^{2}, 2^{3}, 2^{4}\}$, respectively. This setting results in 225 pairs of parameter are explored, which covers a reasonable large tuning space for β and γ. The chosen values whose corresponding model achieves the best performance in the selected range for each data set are listed in the last two columns of Table 1. We observed that for the chosen parameter pair, when increasing or decreasing the parameters further, the performance drops with clear trend. Therefore, it is believed that the parameters we used in the following are well tuned. Note that because of the random seed used in partition, different splits of the tuning data set could end up with slightly different values of β and γ. For the readers who want to duplicate the experiments exactly, our code and data sets partition are available upon the request to authors.

5.1 Comparison with Crammer and Singer's Package

As we mentioned in Section 1, Crammer and Singer proposed a decomposition method together with model problem (2.3), whose working set size is fixed to one. Define

$$\psi_i(\alpha) = \max_{r=1,\cdots,l} F_{i,r}(\alpha) - \min_{r:\alpha_{i,r}<\delta_{y_i,r}} F_{i,r}(\alpha), \forall i = 1, \cdots, l.$$

Their algorithm chooses the index corresponding the largest element of $\psi(\alpha)$ as working set at each iteration and terminates when $\|\psi_i(\alpha)\|_\infty$ is less than a predefined tolerance. More details of this algorithm can be found in [14]. The implementation is available at `http://www.cis.upenn.edu/~crammer/code/MCSVM/MCSVM_1_0.tar.gz.`.

We test the effectiveness of our working set selection rules from two aspects. We first change ψ with ℓ in Crammer and Singer's package, which is just the basic working set selection rule described in Algorithm 2 with $n_{\mathcal{B}} = 1$. All the other parts of the code are not changed. Their original code and our modification are referred as "ψ-based" and "ℓ-based" in the following respectively to represent that different working set selection rules are used. Because the performance differences stem only from working set selection schemes, the effect of different working set selection strategies is well investigated in this situation.

We then developed a new package for solving problem (2.3) in C language. The serial version is based on Algorithm 1 with the practical working set selection rule in Algorithm 4. The inner QP solver follows Algorithm 3. We call this implementation as DASMC(Decomposition Algorithm Solver for Multiclassification). The parallel version(PDASMC)(based on Algorithm 6 and 5) uses standard MPI communication routines(Message Passing Interface Forum) [18] for message passing among the processes, which can be easily portable on many multiprocessor systems. The source code is available upon request authors.

We compare "ψ-based", "ℓ-based" and DASMC on all the seven small data sets used in [14]. Two different cache sizes(400MB and 40MB) are used in the experiments. For other parameters, "ψ-based" and "ℓ-based" follow the default parameter setting in Crammer and Singer's package. For DASMC, $\rho_{min} = 10^{-10}$, $\rho_{max} = 10^{10}$, $L = 2$, $q = 2$, $\epsilon = 10^{-3}$, $n_N = 0.5n_{\mathcal{B}}$ and $\epsilon = 0.001$. We run the same experiment three times to reduce the variance of actual computing and report the average training time on each data set. The computation results are reported in Table 2. Column "sec.", "it." and "#kernel" record the training time in seconds, iteration numbers and kernel element evaluation times respectively. From the result we can see that the same classification accuracy is achieved on all the test problems. Moreover, "ℓ-based" achieves less training time than "ψ-based" when the cache size is limited to 40MB. This stems from the fact that the selecting working set by the value of ℓ usually leads to less kernel evaluation times. So when the cache area is not large enough to contain all the kernel elements ever been calculated, training time is saved. This phenomenon shows that our ℓ-based working set selection rule can indeed accelerate Crammer and Singer's algorithm when the cache size, compared with the scale of the problem, is relatively small.

For DASMC, we use working set size $n_{\mathcal{B}} = 40$ in order to examine the difference between DASMC with Crammer and Singer's implementation, whose working set size in fixed to one. Results are also listed in Table 2. It can be seen that, for tested data sets "mnist", "isolet" and "letter", least training time is achieved by DASMC on both cache sizes. However, for the small scale problems "vowel", "satimage" and "glass", DASMC can not obtain less training time with larger working set size. This is because for small problems, the PG inner solver consumes most of the training time. For Crammer and Singer's implementation, the subproblem corresponding to only one training sample. So it can be solved analytically with less computation effort. Therefore, Crammer and Singer's implementation benefits from the simplicity of the inner solver on small scale problems. However, for large scale problem, the overall number of iterations needed for Crammer and Singer's algorithm will becomes extremely large, which slows down the training process. So the training time is greatly saved by large working set size in this case. Hence, we conduct more experiments on larger data sets to further analyze the behavior of our new

Table 2: Comparison of the training time and accuracy

		cache=400MB			cache=40MB	
Data set	Accuracy	sec.	#kernel	it.	sec.	#kernel
		ψ-based				
satimage	92.35	22.41	11,890,235	15,615	54.23	42,642,525
shuttle	99.7	3.27	1,420,000	11,479	3.75	1,420,000
mnist	96.26	89.83	14,170,000	17,892	297.66	65,700,000
isolet	96.6	430.2	24,927,048	26,723	1644.94	138,988,878
letter	95.12	300.93	18,170,000	38,596	366.06	139,765,000
vowel	52.16	0.64	231,264	3,965	0.63	231,264
glass	68.28	0.18	20,887	7,230	0.18	20,887
		ℓ-based				
satimage	92.35	**21.02**	11,810,405	15,117	41.06	33,346,765
shuttle	99.7	**2.06**	1,380,000	6,774	**2.09**	1,380,000
mnist	96.26	92.42	14,095,000	18,662	240.96	51,695,000
isolet	96.6	456.94	22,575,322	30,476	944.76	70,364,630
letter	95.12	242.41	17,950,000	31,165	284.41	94,865,000
vowel	52.16	**0.47**	230,736	2,781	**0.48**	230,736
glass	68.28	**0.08**	20,099	4,037	**0.08**	20,099
		DASMC				
satimage	92.35	29.84	12,018,850	650	**30.95**	15,957,130
shuttle	99.7	2.48	1,790,000	74	2.49	1,790,000
mnist	96.26	**45.55**	14,405,000	898	**64.99**	28,435,000
isolet	96.6	**225.07**	23,498,546	1,451	**315.52**	43,641,048
letter	95.12	**89.36**	18,895,000	1,233	**101.41**	50,995,000
vowel	52.16	0.66	240,240	68	0.68	240,240
glass	68.28	0.66	22127.2	25.6	0.66	22127.2

implementation.

Careful readers may notice that, for the test problems "glass" and "vowel", the classification accuracies are lower than the results reported in [23]. However, this does not mean that model (2.3) performs bad on these two problems. In fact, the classification accuracy is determined by lots of factors besides the model itself. In our experiment, in order to follow the test settings of Crammer and Singer described in [14], 5-fold cross validation on training data set is utilized for "glass". For data set "vowel", significant difference of classification performance is observed between using cross-validation on the training set and testing directly with the existing test set [38]. In this experiment, we choose the later case to follow the experiment described in [14]. In [23], the authors conduct a 10-fold cross-validation on the whole of each data set. The more folds are split for the data set, the better classification performance will be obtained due to more samples being used for training in each round. We believe that if the same experimental settings are used as in [23], the same high classification accuracies would be achieved by our implementation.

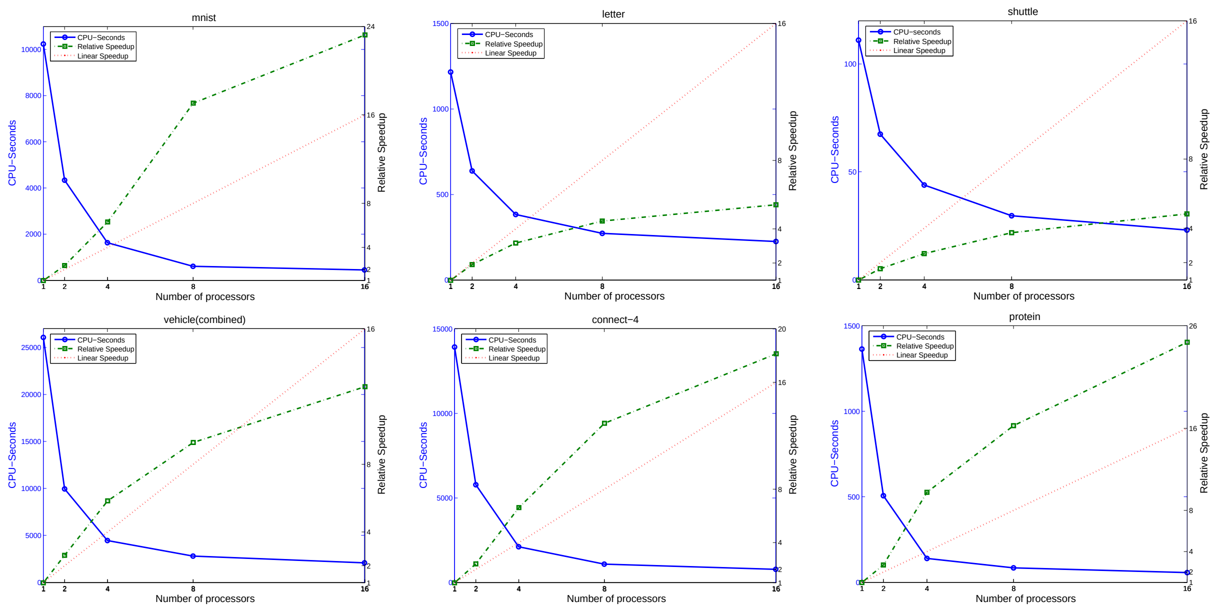

Figure 1: Parallel speedup on mnist(top,left), letter(top,middle), shuttle(top,right), vehicle(combined)(bottom,left), connect(bottom,middle) and protein(bottom,right)

5.2 Relative Speedup of the Parallel Implementation

The multiclass classification with nonlinear kernels for very large scale data set is the widely admitted challenge task. In fact, we tried to train a nonlinear classifier with the implementation of Crammer and Singer on the whole “mnist” data set. It turns out that training time required is prohibitively long for practice. For the same data set, our decomposition based method achieves the model within an hour by using 4 nodes, and is even faster with more nodes being employed. Therefore, in this section, we evaluate the relative speedup of our parallel implementation PDASMC on the six largest problems in Table 1.

Working set size $n_{\mathcal{B}}$ is set to 400 in the following if not notified. The relative speedup is defined as: $sp_r = T_s/T_p$, where T_s is the training time spent on a single processor and T_p is the training time with p processors. We vary the number of processors in the range $\{1, 2, 4, 8, 16\}$ and run the code three times on each data set for the same number of processors to reduce the variance of the computing time in parallel environment. The average training time is used to compute the ratio. Results are showed in Figure 1. Superlinear speedup is observed on test problems “mnist”, “connect-4” and “protein”. This phenomenon benefits from the distributed caching strategy employed in the parallel implementation. However when increasing the number of processors further, the communication cost becomes heavier and the relative speedup drops consequently. For problems “letter” and “shuttle”, because the kernel evaluation is not so time consuming(for example, for “letter” problem, the computing time for kernel evaluation only accounts for 6.35% of the total training time in the case of single processor), the advantage of joint cache is not so obvious. Therefore, only sublinear speedup can be attained. For this kind of problems, the speedup of PDASMC may be improved by either further optimizing the job distribution among processors or distributing the tasks which are not currently parallelized, which will be one of our future works.

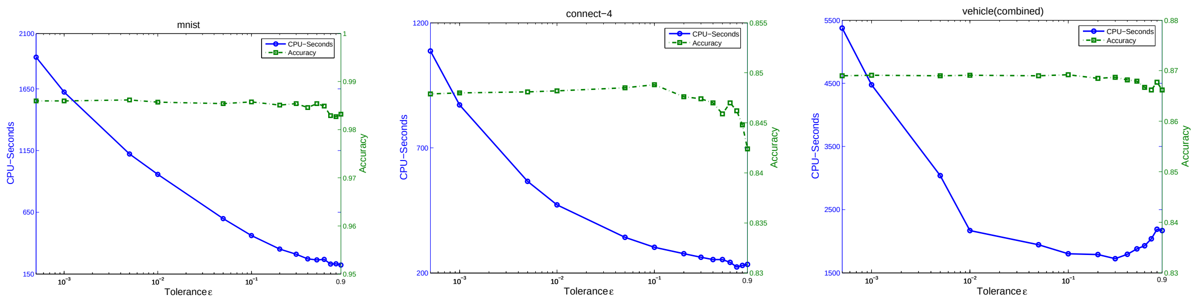

Figure 2: Training time and accuracy as a function of ϵ on mnist(left), connect-4(middle) and vehicle_combined(right)

5.3 Other Behaviors

We use three largest data sets “mnist”, “connect-4” and “vehicle(combined)” in Table 1 to further analyze the behaviors of PDASMC in this subsection. Since superlinear speedup can be obtained on these data sets with 4 processors, $N_P = 4$ is used in all the following experiments for the sake of training time saving.

As it is illustrated in Section 5.1, all the algorithms can achieve almost the same classification accuracy on the tested data sets by setting ϵ to 0.001. However, maybe a lower precision also suffices and reduces the training time at the same time. Therefore, we analyze the sensitivity of our algorithm to the choice of termination tolerance parameter ϵ. Termination tolerance ϵ is varied from 0.0005 to 0.9 to see the change of the classification accuracy and training time. We report the results in Figure 2. It can be seen that the training time drops quickly as the increasing of ϵ. However, the changes of the classification accuracy become smaller with the decreasing of ϵ. So we choose $\epsilon = 0.001$ as the default value to obtain stable classification accuracy within reasonable training time. More smaller tolerance is unnecessary according to the results.

Another thing needs to be inspected is the relationship between total training time and working set size. Figure 3 gives us some intuitive explanation. The training time drops sharply with the increasing of working set size at the beginning because of the reduction of iteration numbers. However, when the working set becomes too large, the time for solving the QP subproblem will dominate the training procedure and the training time starts to increase gradually. Perhaps we can exploit and benefit from more larger working set size with more robust inner QP solver. This topic is one of our future works as well.

At last, we test how PDASMC scales with the number of training samples. We increase the number of training samples of these three data sets from 5,000 to the whole data set size. The log training time required for each size is reported in Figure 4. It can be seen that PDASMC scales roughly in the order of $O(n^{1.8})$ for these three data sets. Similar results are also observed on the decomposition methods for the standard binary SVM [24, 49]. Also, we plot the number of support vectors on different scales of the data sets. This result demonstrates that the training time is roughly proportional to the number of SVs, which validates our analysis in Section 4.1.

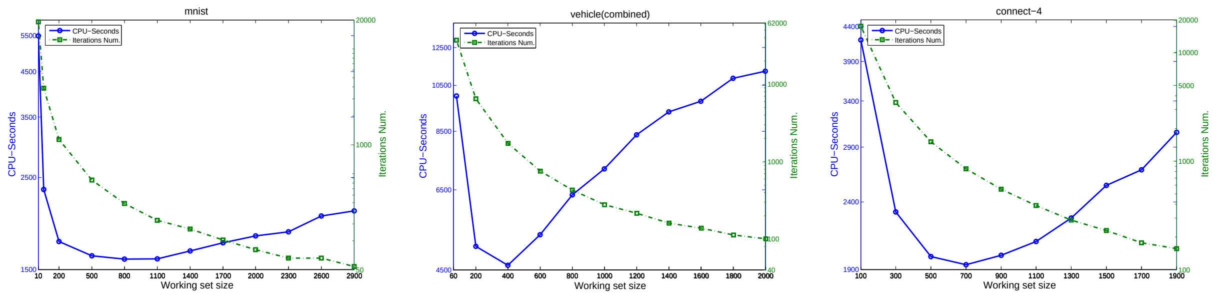

Figure 3: Working set size to training time and number of iteration: mnist(left), vehicle_combined(middle) and connect-4(right) data set

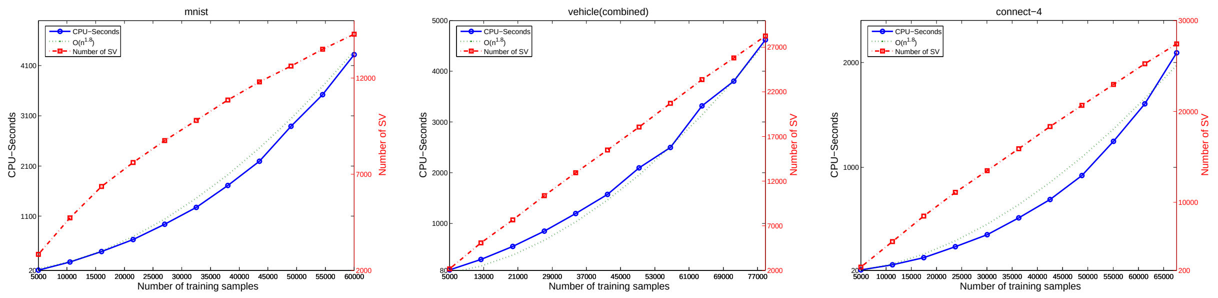

Figure 4: CPU-Seconds and Number of SVs as a function of the number of training samples: mnist(left), vehicle_combined(middle) and connect-4(right)

6 Conclusions

A decomposition algorithm based on the new working set selection rule for problem (2.3) together with the global convergence proof of the algorithm is presented. A special projection process is designed by exploiting the structure of the resulting QP subproblem at each iteration. Therefore PG method is chosen to get the subproblem solution. Techniques for acceleration and parallelization are also given. Both serial and parallel implementations are provided. Experiments on benchmark problems validate the effectiveness of the new algorithm.

One of the our ongoing work currently is further improving the efficiency of our implementation. We plan to include other types of QP solver for the subproblem solution, which can handle very bad scale problems better. Especially, if a proper parallel scheme of the interior point method(IPM) can be developed for our special QP subproblem (2.6), we believe that switching to IPM as the subproblem solver can improve the efficiency of the whole training algorithm, particularly when bad scale subproblems occurred and PG solver needs lots of iterations to converge. We also consider introducing more careful working set selection strategy to reduce iteration numbers. One possible way is introducing part of second order information to the working set selection process. Finally, as pointed by a referee, a careful shrinking strategy can be used to further reduce the number of kernel evaluations.

Acknowledgments

The authors would like to thank two anonymous referees whose comments on the earlier versions of the paper helped us greatly.

References

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[2] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[3] E. G. Birgin, J. M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT'92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

[5] 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 *Proceeding of International Conference of Pattern Recognition*, pages 77–87, 1994.

[6] E. J. Bredensteiner and K. P. Bennett. Multicategory classification by support vector machines. *Computational Optimization and Applications*, 12(1-3):53–79, 1999.

[7] R. Byrd, J. Nocedal, and R. Waltz. Knitro: An integrated package for nonlinear optimization. In *Large-Scale Nonlinear Optimization*, pages 35–59. Springer-Verlag, 2006.

[8] C.-C. Chang and C.-J. Lin. Libsvm, http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

[9] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *Journel of Machine Learning Research*, 9:1775–1822, 2008.

[10] R. Collobert, S. Bengio, and C. Williamson. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

[11] C. Cortes and V. Vapnik. Support vector networks. In *Machine Learning*, pages 273–297, 1995.

[12] K. Crammer and Y. Singer. Improved output coding for classication using continuous relaxation. In *Proceedings of the Thirteenth Annual Conference on Neural Information Processing Systems*, 2000.

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

[14] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[15] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 2(47):201–233, 2002.

[16] Y.-H. Dai and R. Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Mathematical Programming*, 106(3):403–421, 2006.

[17] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[18] M. P. Forum. MPI: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[19] J. Friedman. Another approach to polychotomous classification. Technical report, Deptament of Statistics, Stanford University, 1996.

[20] J. Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[21] E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29:58–81, 2001.

[22] Y. Guermeur. Combining discriminant models with new multiclass svms. Neuro COLT2 Technical Report Seriers NC-TR-00-086, LORIA Campus Scientifique, 2000.

[23] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

[24] T. Joachims. Making large-scale support vector machine learning practical. In A. S. B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines.* 1998.

[25] E. L. Jörg Kindermann 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.

[26] S. S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A sequential dual method for large scale multi-class linear svms. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 408–416, New York, NY, USA, 2008. ACM.

[27] U. Kreßl. Pairwise classification and support vector machines. In B. Schökopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 255–168. MIT Press, 1999.

[28] Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines. Technical Report 1043, Department of Statistics, University of Wisconsin, 2001.

[29] Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines. In *Proceedings of the 33rd Symposium on the Interface*, 2001.

[30] Y. Lin. Support vector machines and the bayes rule in classification. Technical Report 1014, Department of Statistics, University of Wisconsin, 1999.

[31] E. Mayoraz and E. Alpaydin. Support vector machines for multi-class classification. IDIAP Research Repor 98-06, Dallel Molle Insitutue for Perceptual Artifical Intelligence, Martigny, Valais, Switzerland, May 1998.

[32] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, 1909.

[33] B. A. Murtagh and M. Saunders. Minos 5.5. user's guide. SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA, USA, 1998.

[34] E. Osuna, R. Freund, and F. Girosi. Training support vector machines:an application to face detection, 1997.

[35] P. M. Pardalos and N. Kovoor. An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. *Math. Program.*, 46(3):321–328, 1990.

[36] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines, 1998.

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

[38] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, January 2004.

[39] L. P. S. Knerr 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*, New York, 1990. Springer-Verlag.

[40] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, New York, NY, USA, 2004.

[41] W. Sun and Y. Yuan. *Optimization Theory and Methods: Nonlinear Programming.* Springer, New York, USA, 2006.

[42] C. H. Teo, A. Smola, S. V. Vishwanathan, and Q. V. Le. A scalable modular convex solver for regularized risk minimization. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 727–736, New York, NY, USA, 2007. ACM.

[43] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[44] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 11&12:451–484, 1999.

[45] V. N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[46] V. N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, September 1998.

[47] J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, University of London, Department of Computer Science, Royal Holloway, 1998.

[48] G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Computing*, 29(4):535–551, 2003.

[49] L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7:1467–1492, 2006.