2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th Intl. Conference on Data Science and Systems

Improved Parallel Algorithms for Sequential Minimal Optimization of Classification Problems

Wenjing Wei[1], Changrong Li[2], and Jun Guo[1]
[1]Computer Center
East China Normal University
3663 Zhong Shan Rd. N., Shanghai, China
Email: jguo@cc.ecnu.edu.cn
[2]CFETS Information Technology(Shanghai) Co., Ltd
1387 Zhang Dong Rd., Shanghai, China

***Abstract*—An approach presented in our previous work showed that sequential minimal optimization (SMO) algorithms could be executed in parallel. However, its convergence was not guaranteed in some cases. To solve the problem, we propose two novel algorithms in this paper. The first one is to add a condition to decide whether a single violating pair or multiple pairs should be chosen. Thus, the objective function decreases strictly in each iteration, and the algorithm is convergent. The second one aims at choosing better working set in each iteration to reduce the number of entire iterations. Due to the violating pairs chosen in each parallel way are violating at the updating moment, the convergence can be guaranteed. The results of our experiments show the two proposed algorithms can be executed successfully and their convergences are guaranteed totally.**

***Index Terms*—sequential minimal optimization (SMO), parallel, convergence, violating pair, working set**

I. Introduction

With many successful examples in recent years, deep learning shows that it has more excellent performances than the previous methods in machine learning including support vector machines (SVMs) [1, 2]. But as a strong classifier, SVM is still well applied on various occasions. For example, SVM can be used to classify the feature vectors which are extracted by convolutional neural network (CNN). Commonly, the features extracted by CNN are sent to a softmax layer. However, SVM, instead of the softmax layer, is used in many works [3–7] where this kind of CNN-SVM model shows higher accuracies. CNN can be trained by using multiple CPUs or GPUs, but usually SVM training is not executed in parallel. Thus, there exists a bottleneck of the training procedure.

Making SVM suitable for large-scale data is always a challenge.The training of SVM is actually to solve a convex quadratic programming (QP) problem. Therefore, it is time-consuming, especially for large data sets. Sequential minimal optimization (SMO) algorithm can speed up SVM training greatly [8]. However, how to select the working set in SMO depends on the violating pairs which are constantly changed in each iteration. Therefore, it is not easy to develop a parallelizing algorithm of SMO.

Some parallel algorithms for SVM training have been proposed, such as single program multiple data (SPMD) SVM [9], the cascade SVM [10], and parallel SMO (PSMO) [12]. In [9], the entire data set is divided into several subsets, which are assigned to the multiprocessors based on the SPMD model. Each subset is used to find the candidate working set, and then determine the global working set. The subset is updated at the same time to speed up the calculation. The idea of the cascade SVM [10] is to speed up the training by eliminating non-support vectors, which is similar to the chunking algorithm [11]. The different point is that the cascade SVM incorporates parallelism. The cascade method builds a layered architecture with SVM as a filter to extract support vectors. PSMO proposed in [12] is to execute SMO algorithm in parallel by choosing the two most maximal violating pairs. The two pairs are updated at the same time in each iteration before gradient information is updated. There exists a case that one pair will no longer be violating when the other pair is updated. Thus, the convergence of PSMO cannot be guaranteed.

In this paper, two novel algorithms for parallelizing the procedure of SMO are proposed. The first one called improved PSMO (it's denoted by IPSMO-1 here) focuses on keeping clear of no convergence. In IPSMO-1, working set selection (WSS) used in LIBSVM [13] is applied, and more than two violating pairs are updated in each iteration. Eventually, the changes of the objective function are considered to decide whether the variables corresponding to a single violating pair or multiple pairs should be updated. In this way, the objective function decreases strictly so that IPSMO-1's convergence can be guaranteed.

The second one is an improved version of IPSMO-1 (it's denoted by IPSMO-2 in this paper), where a strategy to choose multiple violating pairs reasonably is proposed. Let p denote the number of parallel ways. Firstly, p indices are picked up from set I_{up} (I_{up} is a set defined in SMO algorithm, see [9]) according to the gradient values. Each index is deemed as one element of a violating pair. Secondly, p index vectors are chosen as candidate indices. And each vector has a number of elements which is set in advance. Thus, the other element of the violating pair can be chosen from the corresponding vector. The convergence of this algorithm is guaranteed because each pair we choose is a violating pair at that moment. Finally, after updating p violating pairs, all gradient information is updated simultaneously in parallel.

In the experiments, we can see that the two parallel SMO can be executed effectively. When p is increased, IPSMO-2 has fewer iterations and converges more rapidly than IPSMO-1. The classifiers obtained by all these algorithms have similar accuracies on the test samples.

The rest of this paper is organized as follows. Section II gives a brief introduction to SMO algorithm. And some methods of working set selection are introduced in Section III. We propose two improved parallel algorithms in Section IV. The experimental results are shown in Section V and conclusion can be found in Section VI.

II. Sequential Minimal Optimization

SVM learning can be formalized to solve a convex QP problem, which has no local minimum. Given a training data set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{l}$, where $\boldsymbol{x}_i \in R^n$ is a sample value of the input vector $\boldsymbol{x}$, $y_i \in \{-1, 1\}$ is the corresponding value of the model output $\boldsymbol{y}$, l is the number of training samples. Thus, the target of optimization problem is to find the Lagrange multipliers $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_l]^T$ that minimize

$$W(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{l}\alpha_i \qquad (1)$$

subject to the constraints

$$\begin{aligned} &\sum_{i=1}^{l}\alpha_i y_i = 0, \qquad (2)\\ &0 \le \alpha_i \le C, \quad i = 1, 2, \ldots, l \end{aligned}$$

where C is a penalty parameter, $K(\cdot,\cdot)$ is a kernel function which is assumed to satisfy Mercer's condition. Note that in our experiments in Section V, radial basis function (RBF) is used as a kernel, so Mercer's condition is always satisfied. Let $\boldsymbol{\alpha}^*$ be the optimal solution, we have the decision function as follows:

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

where b^* is an optimum value of the bias b.

The number of variables in Eq. (1) is equal to the number of training samples. Some algorithms [14, 15] were proposed to solve the above optimization problem. However, when there is a large set of training samples, these algorithms often become inefficient and even can not be used. Therefore, Platt proposed SMO algorithm [8], which is a special case of decomposition algorithms. After that Keerthi *et al.* gave improvements to Platt's algorithm [16].

SMO algorithm consists of two parts: the analytical method of solving QP problem with two variables and the heuristic method of selecting a violating pair. The working set of SMO only contains two multipliers so that SMO doesn't require QP numerical optimization step and extra matrix storage space. Without loss of generality, we assume that the selected two multipliers are α_1 and α_2, meanwhile the remaining variables are fixed. Thus, the subproblem is to minimize

$$\begin{aligned} D(\alpha_1, \alpha_2) =& \frac{1}{2}K_{11}\alpha_1^2 + \frac{1}{2}K_{22}\alpha_2^2 + y_1 y_2 K_{12}\alpha_1\alpha_2 \\ &- (\alpha_1 + \alpha_2) + y_1\alpha_1\sum_{i=3}^{l} y_i\alpha_i K_{i1} \\ &+ y_2\alpha_2\sum_{i=3}^{l} y_i\alpha_i K_{i2} \qquad (3) \end{aligned}$$

subject to the constraints

$$\alpha_1 y_1 + \alpha_2 y_2 = -\sum_{i=3}^{l}\alpha_i y_i = \rho, \qquad (4)$$

$$0 \le \alpha_i \le C, \quad i = 1, 2, \ldots, l \qquad (5)$$

where $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and ρ is a constant. For $l \times l$ matrix $\boldsymbol{K} = [K_{ij}]$ is symmetric, we have $K_{ij} = K_{ji}$. Note that the constant terms without α_1, α_2 are omitted in Eq. (3).

Because of equality constraints in Eq. (4), in fact there is only one free variable in above subproblem. Consider the optimization problem for variable α_2. Let α_1^{old} and α_2^{old} be the initial solutions, and let α_1^{new} and α_2^{new} be the optimal solutions. $\alpha_2^{\text{new,unclipped}}$ denotes the optimal solution of α_2 without being clipped by Eq. (5). According to Eqs. (3) and (4), we have

$$\alpha_2^{\text{new,unclipped}} = \alpha_2^{\text{old}} + \frac{y_2(E_1 - E_2)}{\eta}$$

where $\eta = K_{11} + K_{22} - 2K_{12}$ and $E_i = f^{\text{old}}(\boldsymbol{x}_i) - y_i$ which denotes the error between the predicted value and the given value. Considering the clipping, the optimal solution of α_2 is as follows:

$$\alpha_2^{\text{new}} = \begin{cases} H & , \ \alpha_2^{\text{new,unc}} > H \\ \alpha_2^{\text{new,unclipped}} & , \ L \le \alpha_2^{\text{new,unc}} \le H \\ L & , \ \alpha_2^{\text{new,unc}} < L \end{cases}$$

where $L = \max(0, \alpha_2^{\text{old}} - \alpha_1^{\text{old}})$, $H = \min(C, C + \alpha_2^{\text{old}} - \alpha_1^{\text{old}})$ when $y_1 \ne y_2$, and $L = \max(0, \alpha_1^{\text{old}} + \alpha_2^{\text{old}} - C)$, $H = \min(C, \alpha_1^{\text{old}} + \alpha_2^{\text{old}})$ when $y_1 = y_2$. After α_2^{new} is obtained, it's easy to calculate $\alpha_1^{\text{new}} = \alpha_1^{\text{old}} + y_1 y_2(\alpha_2^{\text{old}} - \alpha_2^{\text{new}})$.

III. Working Set Selection

In the following, we focus on how to choose a violating pair to be a working set. According to [8, 16], from the Karush-Kuhn-Tucker condition, we know the optimality condition can be expressed by a single inequality as follows:

$$\min_{i \in I_{\text{up}}^{\delta}(\boldsymbol{\alpha})} F_i(\boldsymbol{\alpha}) \ge \max_{i \in I_{\text{low}}^{\delta}(\boldsymbol{\alpha})} F_i(\boldsymbol{\alpha}) - \tau \qquad (6)$$

where threshold $\tau > 0$,

$$F_i(\boldsymbol{\alpha}) = y_i\frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i} = y_i(\sum_{j=1}^{l} y_i y_j \alpha_j K_{ij} - 1), \qquad (7)$$

and

$$I_{\text{up}}^{\delta}(\boldsymbol{\alpha}) = \{i : \alpha_i \le C - \delta, y_i = 1\} \cup \{i : \alpha_i \ge \delta, y_i = -1\}, \quad (8)$$

$$I_{\text{low}}^{\delta}(\boldsymbol{\alpha}) = \{i : \alpha_i \le C - \delta, y_i = -1\} \cup \{i : \alpha_i \ge \delta, y_i = 1\}. \quad (9)$$

Note that δ in Eqs. (8) and (9) is a relaxation factor, and $\delta \in (0, C/2)$.

A pair of indices (i, j) is said to be a (τ, δ)-violating pair at $\boldsymbol{\alpha}$ if the following condition hold:

$$i \in I^{\delta}_{\text{up}}(\boldsymbol{\alpha}), j \in I^{\delta}_{\text{low}}(\boldsymbol{\alpha}),\ F_i(\boldsymbol{\alpha}) < F_j(\boldsymbol{\alpha}) - \tau.$$

For simplification, a (τ, δ)-violating pair is called a violating pair in the remaining part of this paper.

In Platt's SMO, in each iteration only one violating pair (i, j) is chosen to be a working set (α_i, α_j). This algorithm is seen as the process of continually selecting violating pairs and updating working sets. Therefore, it's necessary to study WSS. There are some existing approaches of WSS [13, 16–18]. Among them, the method of maximal violating pair (MVP) and the way used in LIBSVM are two popular WSS algorithms.

MVP is to select a pair (i, j) that violates the optimality condition most, which meets the following conditions:

$$i \in \arg\max_t \{-F_t(\boldsymbol{\alpha}) | t \in I^{\delta}_{\text{up}}(\boldsymbol{\alpha})\},$$
$$j \in \arg\min_t \{-F_t(\boldsymbol{\alpha}) | t \in I^{\delta}_{\text{low}}(\boldsymbol{\alpha})\}.$$

Actually, it is equivalent to the method of steepest gradient descent.

Chang and Lin [13] proposed a WSS algorithm which had been applied in LIBSVM. In this method, second order information of the objective function is used and it generally leads to faster convergent speed. Assume B is a working set, and $\boldsymbol{d}^T = [\boldsymbol{d}^T_B, \boldsymbol{0}^T_N]$ is the update vector of Lagrange multipliers, we have

$$\begin{aligned}
&W(\boldsymbol{\alpha}^k + \boldsymbol{d}) - W(\boldsymbol{\alpha}^k) \\
=&\nabla W(\boldsymbol{\alpha}^k)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \nabla^2 W(\boldsymbol{\alpha}^k)\boldsymbol{d} \\
=&\nabla W(\boldsymbol{\alpha}^k)^T_B \boldsymbol{d}_B + \frac{1}{2}\boldsymbol{d}^T_B \nabla^2 W(\boldsymbol{\alpha}^k)_{BB}\boldsymbol{d}_B \\
=&\left[\, \nabla W(\alpha^k_i)\, \nabla W(\alpha^k_j) \,\right] \begin{bmatrix} d_i \\ d_j \end{bmatrix} + \frac{1}{2}\left[\, d_i\ d_j \,\right] \begin{bmatrix} Q_{ii}\, Q_{ij} \\ Q_{ji}\, Q_{jj} \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} \\
=&(-F_i(\boldsymbol{\alpha}^k) + F_j(\boldsymbol{\alpha}^k))d'_j + \frac{1}{2}(K_{ii} + K_{jj} - 2K_{ij})d'^2_j
\end{aligned} \tag{10}$$

where k denotes the k-th iteration, $Q_{ij} = y_i y_j K_{ij}$, $d'_i = y_i d_i$ and $d'_j = y_j d_j$. From Eq. (4), we know $d'_i + d'_j = 0$. Eq. (10) is a quadratic function which takes a minimum at $d'_j = -b_{ij}/a_{ij} < 0$. And its minimum is $-b^2_{ij}/2a_{ij}$ where $a_{ij} = K_{ii} + K_{jj} - 2K_{ij} > 0$ and $b_{ij} = -F_i(\boldsymbol{\alpha}^k) + F_j(\boldsymbol{\alpha}^k) > 0$.

In summary, a violating pair selected in LIBSVM is as follows:

$$i \in \arg\max_t \{-F_t(\boldsymbol{\alpha}) \,|\, t \in I^{\delta}_{\text{up}}(\boldsymbol{\alpha})\}, \tag{11}$$

$$j \in \arg\min_t \{-\frac{b^2_{it}}{a_{it}} \,|\, t \in I^{\delta}_{\text{low}}(\boldsymbol{\alpha}), -F_i(\boldsymbol{\alpha}) > -F_t(\boldsymbol{\alpha})\}. \tag{12}$$

IV. Two Novel Parallel Algorithms

In this section, two novel improved parallel algorithms are proposed. In many cases PSMO does work under 2-way parallel, though, its convergence isn't guaranteed so that its application can not be expanded greatly. On the other hand, because of the large amount of data and the high dimensional features in the big data era, the training time will be extended enormously. Thus, increasing the number of parallel ways is a reasonable choice of SMO. Our proposed algorithms can improve PSMO to ensure the convergence and deploy to more parallel ways.

A. IPSMO-1

The first improved algorithm called IPSMO-1 is a greedy algorithm, where in each iteration a single way or multiple ways that cause more declining value of $W(\boldsymbol{\alpha})$ is selected to update the corresponding items. IPSMO-1 is described in detail as follows.

Algorithm 1: *Given a data set $\{\boldsymbol{x_i}, y_i\}^l_{i=1}$, a parameter g in RBF, a penalty parameter C, a relaxation factor δ and a threshold τ.*

Step 1: Set p. Let $k = 0$ and $\boldsymbol{\alpha}^k = \boldsymbol{0}$.

Step 2: If the optimality condition (6) is satisfied, then stop.

Step 3: Choose the first p indices denoted by $\{i^n_{up}\}^p_{n=1}$ from $I^{\delta}_{up}(\boldsymbol{\alpha})$ by sorting $-F_i(\boldsymbol{\alpha})$ in descending order.

Step 4: For each i^n_{up} , choose at most q violating indices denoted by $\{j^{n,m}_{low}\}^q_{m=1}$ from $I^{\delta}_{low}(\boldsymbol{\alpha})$ by sorting $-b^2_{i^n_{up},j_{low}}/a_{i^n_{up},j_{low}}$ in ascending order.

Step 5: For $n = 1, \ldots, p$, $m = 1, \ldots, q$, choose and mark a pair $(i^n_{up}, j^{n,m}_{low})$ if i^n_{up} and $j^{n,m}_{low}$ are both not marked before. If only $j^{n,m}_{low}$ marked, continue m loop. If i^n_{up} is marked or a pair is chosen in m loop, break m loop.

Step 6: For each violating pair (i, j), calculate α^{new}_i and α^{new}_j.

Step 7: Calculate the values of changes on $W(\boldsymbol{\alpha})$ caused by a single pair or multiple pairs, respectively. Select the mode that makes $W(\boldsymbol{\alpha})$ decrease more.

Step 8: Let $k = k + 1$ and update $\boldsymbol{\alpha}^k$, $\{F_i(\boldsymbol{\alpha}^k)\}^l_{i=1}$ and $W(\boldsymbol{\alpha}^k)$. Then unmark all the pairs.

Step 9: Go to Step 2.

In the above algorithm, except some initial parameter settings as in SMO, the number of parallel ways denoted by p should be set. In *Step 4*, j_{low} denotes an element in $I^{\delta}_{\text{low}}(\boldsymbol{\alpha})$. That means $j^{n,m}_{\text{low}}$ corresponding to different i^n_{up} maybe are the same index. Therefore, in *Step 5*, in some cases no violating pair is picked up in some ways, which will be leisure in current iteration. In *Step 7*, after multiple violating pairs are updated, the change of $W(\boldsymbol{\alpha})$ is calculated. Compare to the change caused by a single violating pair, we choose the multiple violating pairs if the effect of updating multiple pairs is better. Otherwise, we choose a single violating pair, which means the same as the serial SMO algorithm. Actually, in our experiments, we find that updating multiple pairs is better in most cases, and updating a single pair is usually easier to appear in the ending period of the training.

As p increases, the effectiveness of IPSMO-1 isn't promoted. From the experimental results shown in Section V, we can see that the best training time usually occurs when p is 4 or 8. The main cause is that some selected pairs in one iteration maybe are not violating after other pairs are updated. Despite the whole selected multiple pairs are helpful for convergence, but the speed do not meet expectations. That means IPSMO-1 may not be well applied to the training of big data.

B. IPSMO-2

To overcome the difficulties, we propose an improved version called IPSMO-2. It aims at selecting multiple pairs in each iteration where each pair is definitely violating. It simulates a serial selection strategy of SMO to select each violating pair, so that each pair contributes to speeding up convergence. IPSMO-2 executes the same steps as the first four steps of IPSMO-1. There are two different points. The one is that it is unnecessary to calculate $W(\boldsymbol{\alpha}^k)$. The other one is the method of WSS. In the following algorithm, the proposed WSS of IPSMO-2 is described.

Algorithm 2: *Given* $\{(i_{up}^n, j_{low}^{n,m})\}_{n,m=1}^{p,q}$ *which is from Step 4 in IPSMO-1.*

```
Step 1: Let $s = \{1, \ldots, p\}$.
Step 2: Get a violating pair $(i, j)$ from the $n$-th way which corresponds to the smallest value in $\{-b^2_{i_{up}^n, j_{low}^{n,1}} / a_{i_{up}^n, j_{low}^{n,1}}\}$, $n \in s$.
Step 3: Exclude $n$ from $s$.
Step 4: Calculate $\Delta\alpha_i$ and $\Delta\alpha_j$. And calculate $\alpha_i^{new}$ and $\alpha_j^{new}$.
Step 5: If $s$ is empty or no violating pair can be picked up, stop; if not, continue.
Step 6: For each $n \in s$, update $F_{i_{up}^n}(\boldsymbol{\alpha})$ and $\{F_{j_{low}^{n,m}}(\boldsymbol{\alpha})\}_{m=1}^q$ in parallel. And sort $\{j_{low}^{n,m}\}_{m=1}^q$ by $-b^2_{i_{up}^n, j_{low}^{n,m}} / a_{i_{up}^n, j_{low}^{n,m}}$ in ascending order.
Step 7: Go to Step 2.
```

In *Step 6* of the above algorithm, $F_r(\boldsymbol{\alpha})$ is assumed to be an item which should be updated. According to Eq. (7), we can derive a simpler update formula:

$$F_r(\boldsymbol{\alpha}) += (\Delta\alpha_i Q_{ir} + \Delta\alpha_j Q_{jr}) y_r. \tag{13}$$

Compare Eq. (13) with Eq. (7), we can see Eq. (13) is simpler and its computation is faster.

Note that q can be set to control the size of candidate indices $\{j_{\text{low}}^{n,m}\}$. Usually, q is far less than the number of training samples. Therefore, the computational consumption is acceptable.

After executing the above algorithm, we can get at most p violating pairs, and the corresponding Lagrange multipliers have been updated. At the end of each iteration of IPSMO-2, all elements of $\{F_i(\boldsymbol{\alpha})\}_{i=1}^l$ will be updated in parallel by using Eq. (13).

Even though the WSS of IPSMO-2 is a serial procedure, the updating of $F_r(\boldsymbol{\alpha})$ is executed in parallel. And for each pair in the working set is violating, the number of iterations can be reduced greatly. As a result, the total training time decreases obviously along with increasing the number of ways.

C. Convergence Analysis

The convergence of SMO algorithm has been clearly proven [19–21]. In summary, as long as the pairs that we choose to update are violating, the global convergence of the algorithm can be guaranteed rigorously.

In IPSMO-1, when updating the single pair leads to a more decrease of objective function than updating the multiple pairs, we will update the single pair. As we know, the single pair is violating definitely so the algorithm is convergent rigorously.

In IPSMO-2, we design the WSS to make sure that the selected multiple pairs are violating definitely. Thus, IPSMO-2 is also convergent.

Although, it takes some additional time to calculate $W(\boldsymbol{\alpha}^k)$ in IPSMO-1 and to update each pair serially in one iteration in IPSMO-2, the main strategy of the algorithm is that we can make up for the whole training time by reducing the number of iterations. Finally, the two algorithms can converge to a status where Eq. (6) is satisfied. Therefore, the accuracies obtained by the proposed parallel algorithms are always at the same level as the serial SMO algorithm.

V. Experiments

In this section, we show some experimental results on 14 data sets, which are from the web site of LIBSVM [22]. As shown in TABLE I, the sizes of the data sets are from 270 to 15000. We compare the two proposed algorithms with PSMO and serial SMO. In all algorithms, Eqs. (11) and (12) are used to choose violating pairs. And also we analyze the results of the proposed algorithms under different conditions. IPSMO-1 and IPSMO-2 are implemented by modifying the source code of LIBSVM. To make it fair, any optimization tools (such as cache and shrinking) are not used in our experiments. RBF is adopted in the training, and the best parameters (C and g) are determined by grid search and 10-fold cross validation. Using the best parameters, the training time and the number of iterations on each training set are obtained. The training time is calculated by averaging the results of five times. And the number of iterations is changeless for each data set. Note that all the models trained by these algorithms on each data set have the same level accuracy.

TABLE I
The data sets used in our experiments

Data set	Size	Attrs.	Data set	Size	Attrs.
a1a	1605	123	heart	270	13
a4a	4781	123	letter.scale	15000	16
australian	690	14	mushrooms	8124	112
breast-cancer	683	10	splice	1000	60
diabetes	768	8	fourclass	862	2
w1a	2477	300	geman.numer	1000	24
w4a	7366	300	svmguide1	3089	4

Firstly, we show the results of serial SMO, PSMO and IPSMO-1 in TABLE II. For the sake of fairness, let $q = 1$,

TABLE II
THE RESULTS OF SMO, PSMO AND IPSMO-1 WHEN $q = 1$.

Data Set	Evaluation	SMO	PSMO (p=2)	PSMO (p=4)	PSMO (p=8)	PSMO (p=16)	IPSMO-1 (p=2)	IPSMO-1 (p=4)	IPSMO-1 (p=8)	IPSMO-1 (p=16)
a1a	iterations	3432	2165	**N/A**	**N/A**	**N/A**	2084	1488	1149	1114
	time(s)	3.215	2.075	**N/A**	**N/A**	**N/A**	2.058	1.541	1.253	1.296
a4a	iterations	2612	1388	866	667	**N/A**	1425	903	637	624
	time(s)	7.109	3.898	2.539	2.032	**N/A**	4.089	2.697	2.008	2.085
australian	iterations	58585	35160	28481	27081	**N/A**	38541	32091	26253	22204
	time(s)	17.818	10.465	8.667	8.394	**N/A**	11.611	9.915	8.393	7.504
breast-cancer	iterations	1628	1566	1560	1558	1557	1566	1560	1558	1557
	time(s)	0.467	0.459	0.463	0.483	0.510	0.461	0.467	0.479	0.506
diabetes	iterations	1054	683	430	471	**N/A**	644	418	370	471
	time(s)	0.300	0.203	0.133	0.151	**N/A**	0.197	0.131	0.122	0.163
fourclass	iterations	945	602	477	442	302	562	477	429	308
	time(s)	0.267	0.175	0.149	0.147	0.100	0.165	0.143	0.133	0.102
german_numer	iterations	358885	252415	**N/A**	**N/A**	**N/A**	233967	164328	146727	162753
	time(s)	181.152	130.542	**N/A**	**N/A**	**N/A**	122.496	87.575	79.331	91.691
heart	iterations	11087	8084	11171	**N/A**	**N/A**	7647	6806	6826	6878
	time(s)	1.283	0.965	1.351	**N/A**	**N/A**	0.915	0.848	0.907	0.998
letter.scale	iterations	220651	131522	**N/A**	**N/A**	**N/A**	131909	91059	68711	53130
	time(s)	122.141	76.882	**N/A**	**N/A**	**N/A**	82.397	74.390	81.276	86.345
mushrooms	iterations	4861	2688	**N/A**	**N/A**	**N/A**	3020	2553	2461	2635
	time(s)	23.180	12.937	**N/A**	**N/A**	**N/A**	14.925	12.888	12.683	14.024
splice	iterations	2664	1438	926	640	540	1438	926	640	540
	time(s)	2.542	1.307	0.869	0.626	0.561	1.364	0.970	0.641	0.571
svmguide1	iterations	971437	756193	**N/A**	**N/A**	**N/A**	746127	672908	655300	712649
	time(s)	916.579	752.363	**N/A**	**N/A**	**N/A**	729.997	673.437	677.352	764.177
w1a	iterations	4583	2900	1818	1542	1527	2444	1855	1108	1248
	time(s)	5.437	3.753	2.535	2.314	2.351	3.275	2.703	1.745	2.050
w4a	iterations	3715	2279	1559	1254	1111	2247	1580	1285	1099
	time(s)	13.534	8.595	6.264	5.263	5.236	8.553	6.345	5.350	5.339

TABLE III
THE RESULTS OF IPSMO-1 WITH DIFFERENT p WHEN $q = p/2$.

Data Set	Evaluation	IPSMO-1 (p=2)	IPSMO-1 (p=4)	IPSMO-1 (p=8)	IPSMO-1 (p=16)	IPSMO-1 (p=32)	IPSMO-1 (p=64)	IPSMO-1 (p=128)
a1a	iterations	2084	1243	**1182**	1255	1447	1824	1870
	time(s)	2.100	**1.313**	1.362	1.667	2.326	3.550	4.299
a4a	iterations	1425	761	726	**707**	882	1164	1270
	time(s)	4.089	**2.313**	2.404	2.601	3.634	5.499	7.309
australian	iterations	38541	31022	**29173**	29301	40203	48769	46593
	time(s)	11.611	**9.715**	9.804	10.933	17.870	28.759	41.683
breast-cancer	iterations	1566	817	458	296	175	102	**69**
	time(s)	0.461	0.254	0.153	0.113	0.082	**0.068**	0.086
diabetes	iterations	644	**434**	482	517	671	647	659
	time(s)	0.197	**0.138**	0.167	0.205	0.313	0.371	0.513
fourclass	iterations	562	395	251	**201**	266	332	457
	time(s)	0.165	0.121	0.085	**0.077**	0.127	0.222	0.485
german_numer	iterations	233967	175440	**167256**	219757	222812	301920	332908
	time(s)	122.496	93.678	**92.244**	131.120	151.463	242.659	310.663
heart	iterations	7647	**6103**	7161	8230	9599	8934	9420
	time(s)	0.915	**0.780**	0.995	1.323	1.779	1.967	2.707
letter.scale	iterations	131909	74381	**56948**	58878	68924	85072	103219
	time(s)	82.397	70.128	83.857	99.095	**64.945**	107.079	167.597
mushrooms	iterations	3020	**2458**	2847	3434	4252	4536	4689
	time(s)	14.925	**12.518**	14.900	19.210	26.227	32.963	42.792
splice	iterations	1438	703	**399**	547	694	865	947
	time(s)	1.364	0.735	**0.421**	0.668	1.030	1.567	2.148
svmguide1	iterations	746127	**674766**	764384	853552	811341	835219	771001
	time(s)	729.997	**680.247**	819.393	996.149	1021.456	1126.583	1186.524
w1a	iterations	2444	1576	**1240**	1872	1930	2318	2735
	time(s)	3.275	2.373	**2.096**	3.349	3.866	5.248	7.177
w4a	iterations	2247	**1408**	1470	1742	2363	2663	3008
	time(s)	8.553	**5.863**	6.742	8.989	14.864	20.899	30.251

TABLE IV
THE RESULTS OF IPSMO-2 WITH DIFFERENT p WHEN $q = p/2$.

Data Set	Evaluation	IPSMO-2 (p=2)	IPSMO-2 (p=4)	IPSMO-2 (p=8)	IPSMO-2 (p=16)	IPSMO-2 (p=32)	IPSMO-2 (p=64)	IPSMO-2 (p=128)
a1a	iterations	2392	1212	929	639	575	234	**211**
	time(s)	2.341	1.265	1.044	0.789	0.832	**0.484**	0.590
a4a	iterations	1532	848	613	437	315	199	**128**
	time(s)	4.441	2.603	2.037	1.580	1.239	**0.907**	0.784
australian	iterations	37395	23178	14793	10423	6951	4864	**3369**
	time(s)	11.052	7.076	4.737	3.655	**2.941**	3.058	4.680
breast-cancer	iterations	1566	781	387	196	97	49	**25**
	time(s)	0.455	0.241	0.130	0.078	**0.055**	0.056	0.089
diabetes	iterations	565	392	285	280	**97**	119	114
	time(s)	0.168	0.122	0.098	0.105	**0.048**	0.077	0.113
fourclass	iterations	584	329	187	98	67	49	**32**
	time(s)	0.168	0.101	0.065	0.041	**0.039**	0.051	0.082
german_numer	iterations	236649	146163	107704	73458	63703	50163	**23784**
	time(s)	123.157	76.277	57.886	41.269	39.001	39.230	**32.708**
heart	iterations	7455	4900	4120	2501	1429	1194	**1081**
	time(s)	0.899	0.618	0.563	0.399	**0.314**	0.410	0.484
letter.scale	iterations	132208	69761	42024	29042	21548	14373	**8177**
	time(s)	82.297	66.003	63.605	50.266	**21.025**	22.315	28.074
mushrooms	iterations	2805	1961	1135	757	628	459	**302**
	time(s)	13.690	9.768	6.021	4.013	3.489	2.785	**2.199**
splice	iterations	1582	791	432	265	178	133	**103**
	time(s)	1.639	0.770	0.462	0.333	**0.293**	0.342	0.524
svmguide1	iterations	630094	528329	409835	343769	**174343**	175760	196397
	time(s)	606.193	515.603	408.822	351.319	**197.597**	226.529	285.438
w1a	iterations	2611	1611	940	604	470	365	**288**
	time(s)	3.490	2.375	1.541	1.044	0.880	0.728	**0.751**
w4a	iterations	2311	1286	785	592	452	371	**323**
	time(s)	8.757	5.252	3.532	3.012	2.525	**2.257**	2.279

TABLE V
THE RESULTS OF IPSMO-2 WITH DIFFERENT q WHEN $p = 128$.

Data Set	Evaluation	*q=1*	*q=2*	*q=4*	*q=8*	*q=16*	*q=32*	*q=64*	*q=128*
a1a	iterations	621	486	396	281	244	245	211	**171**
	time(s)	0.945	0.771	0.654	0.498	**0.473**	0.538	0.590	0.611
a4a	iterations	302	252	199	178	151	120	128	**103**
	time(s)	1.215	1.045	0.861	0.794	0.711	**0.639**	0.784	0.841
australian	iterations	9379	8879	7757	8227	5927	4192	3369	**2787**
	time(s)	4.818	4.686	4.194	4.581	3.646	**3.323**	4.680	7.585
breast-cancer	iterations	1468	746	382	191	96	48	25	**15**
	time(s)	0.826	0.440	0.236	0.136	0.090	**0.076**	0.089	0.129
diabetes	iterations	197	148	156	119	162	**108**	114	161
	time(s)	0.121	0.096	0.100	**0.083**	0.114	0.098	0.113	0.158
fourclass	iterations	127	89	65	58	53	46	**32**	38
	time(s)	0.080	0.062	0.053	**0.053**	0.056	0.065	0.082	0.140
german_numer	iterations	81519	77120	77513	63172	56714	39789	23784	**14009**
	time(s)	56.125	53.623	54.677	45.981	44.150	36.652	32.708	**26.106**
heart	iterations	2938	2590	2205	1925	1332	1093	**1081**	1084
	time(s)	0.716	0.650	0.571	0.530	**0.432**	0.459	0.484	0.508
letter.scale	iterations	24727	18424	14414	11835	10478	9533	8177	**7284**
	time(s)	28.545	23.785	20.588	**19.005**	19.433	22.051	28.074	41.094
mushrooms	iterations	626	558	563	496	442	414	302	**212**
	time(s)	3.622	3.244	3.322	2.951	2.688	2.678	2.199	**2.049**
splice	iterations	382	259	192	141	135	125	103	**87**
	time(s)	0.566	0.410	0.326	**0.272**	0.313	0.392	0.524	0.863
svmguide1	iterations	378187	348515	310316	221920	**170278**	175538	196397	193027
	time(s)	471.300	434.783	390.588	284.739	**228.128**	251.197	285.438	285.130
w1a	iterations	875	663	436	309	243	**149**	288	200
	time(s)	1.599	1.258	0.864	0.650	0.554	**0.410**	0.751	0.898
w4a	iterations	692	562	489	398	465	384	323	**292**
	time(s)	4.089	3.354	2.910	2.389	2.802	2.441	**2.279**	2.401

which means there is only one candidate index in I_{low} in each parallel way of PSMO and IPSMO-1. From TABLE II, we can see if PSMO can converge it is usually faster than IPSMO-1. The reason is that IPSMO-1 should calculate $W(\boldsymbol{\alpha})$ additionally in each iteration. However, as p increases, the non-convergence phenomenon of PSMO occurs frequently. That means PSMO has some potential risks in real applications. On the other hand, as p increases, the number of iterations of IPSMO-1 is always less than SMO. Thus, the total training time decreases obviously. Also, we can see the convergence of IPSMO-1 is guaranteed definitely.

Secondly, we record the experimental results of IPSMO-1 when p is from 2 to 128 and $q = p/2$, which are shown in TABLE III. In each line, the best results are highlighted by bold fonts. We find that the number of iterations is decreasing as p increases, and most of data sets work best when p is 4 or 8. When p continues to increase, the number of iterations tends to increase and the training time begins to rise. Thus, the effectiveness of IPSMO-1 becomes worse. It means that the parallel ways are not the more the better.

Thirdly, in TABLE IV, we show the results of IPSMO-2 when p is from 2 to 128 and $q = p/2$. Comparing TABLE IV with TABLE III, we can see the performance of IPSMO-2 isn't better than IPSMO-1 when $p = 2$. This is because we need to update $F_j(\boldsymbol{\alpha}), j \in \{j_{\text{low}}^{n,m}\}_{m=1}^{q}, n \in s$ in IPSMO-2, which takes a little more time. However, as p increases, no matter the number of iterations or the training time, it is reduced greatly. Most of data sets work best when p is 32, 64 or 128. That means IPSMO-2 can select better violating pairs to obtain better performance than IPSMO-1. For the data sets with larger size, the best results often occur in the case of much more parallel ways. That means IPSMO-2 can be applied to deal with large size data sets provided that there are enough CPUs or GPUs.

Finally, the results of IPSMO-2 when $p = 128$ and q is from 1 to 128 are shown in TABLE V. With different p, we can see the effectiveness of IPSMO-2 changes. Although it takes more time to update $F_j(\boldsymbol{\alpha})$, as q increases, the possibility of finding better violating pairs increases too. Thus, the best results always occur when q is bigger.

VI. Conclusion

In this paper, we proposed two novel parallel SMO algorithms. In IPSMO-1, the changes of the objective function are considered to decide how to update the violating pairs. In IPSMO-2, in order to search better violating pairs, the selected gradient information is updated in each parallel way. Thus, each selected pair is always violating. Both IPSMO-1 and IPSMO-2 can solve the convergence problem, which occurs in previous PSMO. In spite of the fact that the training in each iteration is more time-consuming, by reducing the number of iterations, the total training time is decreased obviously. Experimental results showed that IPSMO-2 has better performance than IPSMO-1. In our future work, using optimization tools such as cache and shrinking will be considered to further accelerate the training procedure. And we will also consider how to apply the proposed method to solve the problems with large data sets combined with CNN.

References

[1] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[2] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[3] M. Matsugu, K. Mori, and T. Suzuki, “Face recognition using SVM combined with CNN for face detection,” *Neural Information Processing*. Berlin Heidelberg: Springer-Verlag, pp. 356–361, 2004.

[4] K. Mori, M. Matsugu, T. Suzuki, “Face recognition using SVM fed with intermediate output of CNN for face detection,” *Iapr Conference on Machine Vision Applications*, pp. 410–413, 2005.

[5] X. X. Niu, C. Y. Suen, “A novel hybrid CNN-SVM classifier for recognizing handwritten digits,” *Pattern Recognition*, vol. 45, no. 4, pp. 13180-1325, 2012.

[6] D. X. Xue, R. Zhang, H. Feng, Y. L. Wang, “CNN-SVM for microvascular morphological type recognition with data augmentation,” *Journal of Medical and Biological Engineering*, vol. 36, no. 6, pp. 755–764, 2016.

[7] Z. Wang, Z. Wang, H. Zhang, X. Guo, “A novel fire detection approach based on CNN-SVM using tensorflow,” *Intelligent Computing Methodologies*, pp. 682–693, 2017.

[8] J. C. Platt, “Fast training of support vector machines using sequential minimal optimisation,” *Advances in Kernel Methods-Support Vector Learning*. Cambridage, MA: MIT Press, pp. 185–208, 1998.

[9] L. J. Cao, S. S. Keerthi, C. J. Ong, J. Q. Zhang, U. Periyathamby, X. J. Fu, H. P. Lee, “Parallel sequential minimal optimization for the training of support vector machines,” *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 1039–1049, 2006.

[10] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, V. Vapnik, “Parallel support vector machines: The cascade svm,” *Advances in Neural Information Processing Systems*, pp. 521–528, 2004.

[11] B. E. Boser, I. M. Guyon, V. N. Vapnik, “A training algorithm for optimal margin classifiers,” *The Workshop on Computational Learning Theory*, pp. 144–152, 1992.

[12] X. Wang, J. Guo, “An Algorithm for Parallelizing Sequential Minimal Optimization,” *Neural Information Processing*. Berlin Heidelberg: Springer-Verlag, pp. 657–664, 2013.

[13] C. C. Chang, C. J. Lin, “Libsvm: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[14] E. Osuna, R. Freund, F. Girosi, “An improved training algorithm for support vector machines,” *Neural Networks for Signal Processing*, pp. 276–285, 1997.

[15] T. Joachims, “Making large-scale support vector machine learning practical,” *Advances in Kernel Methods-Support Vector Learning*. Cambridage, MA: MIT Press, pp. 169–184, 1998.

[16] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, K. R. K. Murthy, “Improvements to platt's SMO algorithm for SVM classifier design,” *Neural Computation*, vol. 13, no. 3, pp. 637–649, 2001.

[17] R. E. Fan, P. H. Chen, C. J. Lin, “Working set selection using second order information for training support vector machines,” *Journal of Machine Learning Research*, vol. 6, no. 4, pp. 1889–1918, 2005.

[18] P. H. Chen, R. E. Fan, C. J. Lin, “A study on SMO-type decomposition methods for support vector machines,” *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 893–908, July 2006.

[19] C.-J. Lin, “On the convergence of the decomposition method for support vector machines,” *IEEE Trans. Neural Networks*, vol. 12, no. 6, pp. 1288–1298, Nov. 2001.

[20] C.-J. Lin, “Asymptotic convergence of an SMO algorithm without any assumption,” *IEEE Trans. Neural Network*, vol. 13, no. 1, pp. 248–250, Jan. 2002.

[21] S. S. Keerthi and E. G. Gilbert,“Convergence of a generalized SMO algorithm for SVM classifier design,” *Mach. Learn.*, vol. 46, pp. 351–360, 2002

[22] LIBSVM Data [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/