Accelerated Asynchronous Greedy Coordinate Descent Algorithm for SVMs

Bin Gu[*†], **Yingying Shan**[*], **Xiang Geng**[*], **Guansheng Zheng**[*]
[*]School of Computer & Software, Nanjing University of Information Science & Technology, P.R.China
[†]Department of Electrical & Computer Engineering, University of Pittsburgh, USA
big10@pitt.edu, yyshan@nuist.edu.cn, nuist_gengxiang@163.com, zgs@nuist.edu.cn

Abstract

Support vector machines (SVMs) play an important role in machine learning in the last two decades. Traditional SVM solvers (e.g. LIBSVM) are not scalable in the current big data era. Recently, a state of the art solver was proposed based on the asynchronous greedy coordinate descent (AsyGCD) algorithm. However, AsyGCD is still not scalable enough, and is limited to binary classification. To address these issues, in this paper we propose an asynchronous accelerated greedy coordinate descent algorithm (AsyAGCD) for SVMs. Compared with AsyGCD, our AsyAGCD has the following two-fold advantages: 1) Our AsyAGCD is an accelerated version of AsyGCD because active set strategy is used. Specifically, our AsyAGCD can converge much faster than AsyGCD for the second half of iterations. 2) Our AsyAGCD can handle more SVM formulations (including binary classification and regression SVMs) than AsyGCD. We provide the comparison of computational complexity of AsyGCD and our AsyAGCD. Experiment results on a variety of datasets and learning applications confirm that our AsyAGCD is much faster than the existing SVM solvers (including AsyGCD).

1 Introduction

Support vector machines (SVMs) [Vapnik and Vapnik, 1998] is an elegant and widely applied method with in-depth theoretical analysis in machine learning. However, the traditional SVM solvers (e.g. LIBSVM [Chang and Lin, 2011]) are not scalable in the current big data era because of the ordinary handle of large kernel matrix. To speed up the computation of SVMs, the techniques of kernel approximation and parallel computations have been exploited.

Kernel approximation approaches are trying to approximate the kernel matrix, then solving a linear SVM [Williams and Seeger, 2001; Rahimi and Recht, 2008; Gu *et al.*, 2018b]. However, the kernel approximation approaches cannot obtain the exact solution of SVMs, since the approximate kernel is used. In this paper, we focus to obtain the exact solution of SVMs.

Most parallel algorithms for training SVMs are based on the synchronous model [Chang and Lin, 2011; Zhao and Magoules, 2011; You *et al.*, 2015]. However, without waiting sync point the asynchronous parallel algorithm [Gu *et al.*, 2018a] is much more efficient. To the best of our knowledge, the only asynchronous parallel kernel learning method is the asynchronous parallel greedy coordinate descent (AsyGCD) algorithm for C-SVC [You *et al.*, 2016]. Specifically, at each iteration, workers of AsyGCD asynchronously conduct greedy coordinate descent updates on a block of variables. Comparing with the asynchronously stochastic coordinate descent algorithms [Liu *et al.*, 2015; Duchi *et al.*, 2015], the AsyGCD algorithm can achieve much faster convergence speed due to the greedy selection of updated coordinates.

As mentioned above, AsyGCD is a state of the art solver for solving C-SVC [You *et al.*, 2016]. Thus, it is natural to have the following two questions: 1) Can AsyGCD be further speeded up? 2) Can AsyGCD solve other SVMs, besides C-SVC? To address these issues, in this paper, we propose an asynchronous accelerated greedy coordinate descent algorithm (AsyAGCD) to further accelerate AsyGCD by the technique of active set and extend it to handle generalized SVMs. Active set (or called as shrinking) is a technique to improve the efficiency of dual coordinate descent methods [Hsieh *et al.*, 2008] which has been successfully used in many works [Chang and Lin, 2011; Chiang *et al.*, 2016; De Santis *et al.*, 2016; Birgin and Martínez, 2002] including LIBSVM. Experiment results on a variety of datasets and learning applications confirm that our AsyAGCD is much faster than the existing SVM solvers (including AsyGCD).

2 Generalized SVMs and AsyGCD Algorithm

In this section, we first give a generalized SVM formulation, and then give a brief review of the AsyGCD algorithm.

2.1 Generalized SVMs

In this paper, we consider a generalized SVM formulation as follows.

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & F(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \boldsymbol{p}^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & 0 \le \alpha_i \le C \quad i = 1, \cdots, \ell \end{aligned} \tag{1}$$

where Q is an $\ell \times \ell$ symmetric and positive semi-definite matrix, $\boldsymbol{p}$ and $\boldsymbol{\alpha}$ are the vectors with the size of ℓ. In the following, we show that C-SVC and ϵ-SVR can be formulated as the form of (1).

Given a training set $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^l$ with $\boldsymbol{x}_i \in \mathbb{R}^d$, and $y_i \in \{+1, -1\}$ for binary classification or $y_i \in \mathbb{R}$ for regression. To make the dual formulations of C-SVC and ϵ-SVR free of equality constraints, we append $\boldsymbol{x}$ with an additional dimension $\hat{\boldsymbol{x}}_i^T \leftarrow [\boldsymbol{x}_i^T, 1]$ as mentioned in [Hsieh *et al.*, 2008]. Thus, we have a new training sample set $\hat{S} = \{(\hat{\boldsymbol{x}}_i, y_i)\}_{i=1}^l$. For C-SVC, we let $Q_{ij} = y_i y_j K(\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{x}}_j)$, $K(\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{x}}_j) = \langle \phi(\hat{\boldsymbol{x}}_i), \phi(\hat{\boldsymbol{x}}_j) \rangle$ and $\boldsymbol{p} = -\mathbf{1}$. Thus, we can verify that C-SVC is a special case of (1).

For ϵ-SVR, we introduce another vector z with $z_i \in \mathbb{R}$ to denote the targets associated to $\boldsymbol{x}_i$. To present the ϵ-SVR in a compact form, we double the set $\{(\hat{\boldsymbol{x}}_i, z_i)\}_{i=1}^l$, and define an extended training set as $\hat{S} = \hat{S}^1 \cup \hat{S}^2$, where $\hat{S}^1 = \{(\hat{\boldsymbol{x}}_i, z_i, y_i = 1)\}_{i=1}^l$ and $\hat{S}^2 = \{(\hat{\boldsymbol{x}}_i, z_i, y_i = -1)\}_{i=1}^l$. Thus, the dual ϵ-SVR can be presented as follows.

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & F(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \left[\epsilon \mathbf{1}^T - \boldsymbol{z}; \epsilon \mathbf{1}^T + \boldsymbol{z}\right]^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & 0 \le \alpha_i \le C,\ i = 1, \cdots, 2l \end{aligned} \tag{2}$$

where ϵ defines the ϵ insensitive zone loss in the primal formulation of ϵ-SVR. Thus, it is easy to verify that the dual ϵ-SVR (2) is a special case of (1), if setting $\boldsymbol{p} = \left[\epsilon \mathbf{1}^T - \boldsymbol{z}; \epsilon \mathbf{1}^T + \boldsymbol{z}\right]$.

2.2 AsyGCD Algorithm

AsyGCD Algorithm was originally proposed by [You *et al.*, 2016] to solve the optimization problem (1) with $\boldsymbol{p} = -\mathbf{1}$. In this section, we modify the AsyGCD algorithm such that it can solve the generalized SVM formulation (1). Firstly, we present the i-th coordinated subproblem of (1) as follows.

$$\min_{\delta} F(\boldsymbol{\alpha} + \delta \boldsymbol{e}_i) \quad s.t. \quad 0 \le \alpha_i + \delta \le C \tag{3}$$

where $\boldsymbol{e}_i = [\underbrace{0, \cdots, 0}_{i-1}, 1, 0, \cdots, 0]^T$ is the i-th indicator vector. For each iteration of AsyGCD, we update the solution α with $\alpha_i = \alpha_i + \delta^*$, where δ^* is obtained by solving the corresponding i-th coordinated subproblem (3). Fortunately, δ^* has the closed form solution as follows.

$$\delta_i^* = P_{[0,C]}(\alpha_i - \nabla_i F(\boldsymbol{\alpha})/Q_{ii}) - \alpha_i \tag{4}$$

where $P_\Omega(\alpha)$ is the Euclidean projection of α onto Ω, as mentioned in [You *et al.*, 2016] the step size is set as $1/Q_{ii}$.

We summarize our AsyGCD algorithm in Algorithm 1. For each iteration of AsyGCD, the i-th coordinate is selected greedily according to $|\nabla_i^+ F(\boldsymbol{\alpha})|$, where $\nabla_i^+ F(\boldsymbol{\alpha})$ is defined as $\nabla_i^+ F(\boldsymbol{\alpha}) = \alpha_i - P_{[0,C]}(\alpha_i - \nabla_i F(\boldsymbol{\alpha}))$. To make AsyGCD run efficiently, we maintain the gradient $\nabla F(\boldsymbol{\alpha})$ as $\boldsymbol{G} = \nabla F(\boldsymbol{\alpha}) = Q\boldsymbol{\alpha} + \boldsymbol{p}$.

Note that, the asynchronously parallel algorithm would overwrite the variables in the shared memory due to without any lock. To ease the phenomenon of overwriting in AsyGCD, [You *et al.*, 2016] partitioned the dual variables into p groups (i.e., $S_1 \cup S_2 \cup \cdots \cup S_p = \{1, \cdots, l\}$, and $S_i \cap S_j = \emptyset, \forall i, j, i \neq j$), where p is the number of cores in a multi-core shared memory machine.

Input : The training set $\hat{S}$, an initial $\boldsymbol{\alpha} = \mathbf{0}$ and $G = -\mathbf{1}$.
Output: The vector $\boldsymbol{\alpha}$.

Each thread repeatedly performs the following updates in parallel:
for $t = 1, 2, \cdots$ **do**
  Pick $i = arg\max_i |\nabla_i^+ F(\boldsymbol{\alpha})|$;
  Compute δ_i^* according to (4);
  for $j = 1, 2, \cdots, l$ **do**
    $G_j = G_j + \delta_i^* Q_{j,i}$ **using atomic update**;
  end
  $\alpha_i = \alpha_i + \delta_i^*$.
end
return $\boldsymbol{\alpha}$.

Algorithm 1: AsyGCD Algorithm

3 Accelerated AsyGCD Algorithm

In this section, we first introduce the active set technique. Based on the active set technique, we propose our AsyAGCD. Finally, we compare the time complexity between AsyAGCD and AsyGCD.

3.1 Active Set Technique

To present the active set technique smoothly, we first give the Karush-Kuhn-Tucker (KKT) conditions [Boyd and Vandenberghe, 2004] to the generalized SVM formulation (1), then give our active set technique to (1).

KKT Conditions: According to the KKT theory [Bertsekas, 1999], the KKT conditions of (1) are presented as follows.

$$\nabla_i F(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i + \boldsymbol{p}_i \begin{cases} \ge 0; & if\ \ \alpha_i = 0 \\ = 0; & if\ \ 0 < \alpha_i < C \\ \le 0; & if\ \ \alpha_i = C \end{cases} \tag{5}$$

where $\nabla F(\boldsymbol{\alpha}) = Q\boldsymbol{\alpha} + \boldsymbol{p}$ is the gradient to the function $F(\boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$.

According to the gradient $\nabla F(\boldsymbol{\alpha})$ used in (5), we can define the projected gradient $\nabla^P f(\boldsymbol{\alpha})$ as follows.

$$\nabla_i^P F(\boldsymbol{\alpha}) = \begin{cases} \nabla_i F(\boldsymbol{\alpha}) & if\ \ 0 < \alpha_i < C \\ \min(0, \nabla_i F(\boldsymbol{\alpha})) & if\ \ \alpha_i = 0 \\ \max(0, \nabla_i F(\boldsymbol{\alpha})) & if\ \ \alpha_i = C \end{cases} \tag{6}$$

If $\boldsymbol{\alpha}$ satisfies the KKT conditions, we have that

$$\nabla_i^P F(\boldsymbol{\alpha}) = 0, \quad \forall i = 1, \cdots, \ell \tag{7}$$

Thus, Eq. (7) is another form of the KKT conditions (5).

If $\boldsymbol{\alpha}$ does not satisfy the KKT conditions, it is easy to conclude that $\max_i \nabla_i^P F(\boldsymbol{\alpha}) > 0$ or $\min_i \nabla_i^P F(\boldsymbol{\alpha}) < 0$. These two values (i.e., $\max_i \nabla_i^P F(\boldsymbol{\alpha})$ and $\min_i \nabla_i^P F(\boldsymbol{\alpha})$, later we will define them as M and m respectively in (9)) measure how the solution $\boldsymbol{\alpha}$ violates the KKT conditions.

Active Set Technique: As mentioned before, AsyGCD algorithm is directly solving the optimization problem of C-SVC. In this section, we will use the active set technique to accelerate the AsyGCD algorithm by solving a smaller optimization problem.

Given an active set A (a subset of $\{1, \cdots, \ell\}$), correspondingly, we can define an inactive set $\bar{A}$ as $\bar{A} = \{1, \cdots, l\} - A$. If we have the prior knowledge that the variables $\alpha_{\bar{A}}$ are fixed, and only the variables α_A is active, the original optimization (1) can be reduced as a smaller optimization problem (8) as follows.

$$\begin{aligned} \min_{\boldsymbol{\alpha}_A} \quad & \frac{1}{2}\boldsymbol{\alpha}_A^T Q_{AA}\boldsymbol{\alpha}_A + (Q_{A\bar{A}}\boldsymbol{\alpha}_{\bar{A}} + \boldsymbol{p}_A)^T\boldsymbol{\alpha}_A \\ s.t. \quad & 0 \le \alpha_i \le C, \quad \forall i \in A \end{aligned} \tag{8}$$

where Q_{AA} and $Q_{A\bar{A}}$ are the sub-matrices of Q. Thus, we can save the computational time because of solving a smaller optimization problem (8). Now the only question is that how to set the active set A and the inactive set $\bar{A}$. To answer this question, we first give Theorem 1 which can be proved similarly according to the proof in [Hsieh *et al.*, 2008].

Theorem 1. *Assume $\{\boldsymbol{\alpha}^k\}_{k=1}^{\infty}$ is an infinite sequence of (1) generated by Algorithm 1. Specifically, $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \delta^* \boldsymbol{e}_{i_k}$, where the index i_k is selected by Algorithm 1 for the k-th iteration, and δ^* is the optimal solution of*

$$\min_{\delta} F(\boldsymbol{\alpha}^k + \delta \boldsymbol{e}_i) \quad s.t.\ 0 \le \alpha_i + \delta \le C, \quad i = 1, \cdots, \ell$$

Assume every limit point of $\{\boldsymbol{\alpha}^k\}_{k=1}^{\infty}$ is a stationary point, and let $\boldsymbol{\alpha}^*$ be the convergent point of $\{\boldsymbol{\alpha}^k\}_{k=1}^{\infty}$. We have that

1. *If $\boldsymbol{\alpha}_i^* = 0$ and $\nabla_i F(\boldsymbol{\alpha}^*) > 0$, then $\exists k_i$ such that $\forall k \ge k_i, \boldsymbol{\alpha}_i^k = 0$.*
2. *If $\boldsymbol{\alpha}_i^* = C$ and $\nabla_i F(\boldsymbol{\alpha}^*) < 0$, then $\exists k_i$ such that $\forall k \ge k_i, \boldsymbol{\alpha}_i^k = C$.*
3. *And $\lim_{k\to\infty} \nabla^P F(\boldsymbol{\alpha}^k) = \mathbf{0}$.*

Theorem 1 shows that the variables $\boldsymbol{\alpha}_i$ at upper and lower bounds would be fixed from a certain iteration. Thus, we can set the active set A and the inactive set $\bar{A}$ according to Theorem 1. Specifically, we define M and m as follows.

$$M \equiv \max_i \nabla_i^P f(\boldsymbol{\alpha}), \quad m \equiv \min_i \nabla_i^P f(\boldsymbol{\alpha}) \tag{9}$$

If one of the following two conditions is held, we can define the inactive set $\bar{A}$ as the collection of the elements satisfying the following conditions.

1. $\alpha_i = 0$ and $\nabla_i F(\boldsymbol{\alpha}) > \bar{M}$.
2. $\alpha_i = C$ and $\nabla_i F(\boldsymbol{\alpha}) < \bar{m}$.

where

$$\bar{M} = \begin{cases} M & if\ M > 0 \\ \infty & otherwise \end{cases}, \quad \bar{m} = \begin{cases} m & if\ m < 0 \\ -\infty & otherwise \end{cases} \tag{10}$$

Notice that $\bar{M}$ and $\bar{m}$ are the auxiliary variables to M and m, such that $\bar{M} > 0$ and $\bar{m} < 0$. Correspondingly, the active set A is defined as $A = \{1, \cdots, l\} - \bar{A}$.

During the optimization process, the gradients for $i \in A$ can be obtained directly during solving problem (8) due to the following relation:

$$G_A = (Q\boldsymbol{\alpha})_A + \boldsymbol{p}_A = (Q_{AA}\boldsymbol{\alpha}_A) + (Q_{A\bar{A}}\boldsymbol{\alpha}_{\bar{A}}) + \boldsymbol{p}_A \tag{11}$$

However, the gradients for $i \in \bar{A}$ need to be recalculated and this step may be very time-consuming if we shrink many elements. In order to reduce the computational cost of this reconstruction, we maintain a vector $\bar{G} \in \mathbb{R}^{\ell}$ throughout iterations [Chang and Lin, 2011].

$$\bar{G}_i = C \sum_{j:\alpha_j = C} Q_{ij},\ i = 1, \cdots, \ell \tag{12}$$

We use the fact that $\alpha_j = 0$ or $\alpha_j = C$ if $j \in \bar{A}$. Thus, for $i \in \bar{A}$, we have

$$G_i = \sum_{j=1}^{\ell} Q_{ij}\alpha_j + p_i = \bar{G}_i + p_i + \sum_{\substack{j \in A \\ 0 < \alpha_j < C}} Q_{ij}\alpha_j \tag{13}$$

Eq. (13) can be used to reduce the computational cost of G_i because normally there exists a large proportion of samples are bounded (i.e., $\alpha_j = 0$ or $\alpha_j = C$). Thus, we can just focus on the computation on the elements in the active set A.

3.2 AsyAGCD

In this section, we introduce the accelerated asynchronous greedy coordinate descent algorithm (AsyAGCD) based on the active set technique. The AsyAGCD algorithm is presented in Algorithm 2. Algorithm 2 includes two parts, i.e., the inner loop and the outer loop.

Inner loop: The inner loop (see lines 5-21 of Algorithm 2) is mainly to optimize the subproblem (8) with the active set technique. In order to simplify the complexity of reconstructing gradients (see line 22 of Algorithm 2), we maintain the vector $\bar{G}$ (see line 13 of Algorithm 2) for each update. Simultaneously, while arriving the shrinking condition (i.e., $k\%MaxShrinkIter = 0$, where MaxShrinkIter is predefined by user as the maximum number of iterations for doing the shrinking), the inner loop (see lines 16-18 of Algorithm 2) do the shrinking (i.e., Algorithm 3), to reduce the size of active set A. Algorithm 3 describes the procedures of shrinking. All these computations can be run in parallel. Line 5 of Algorithm 2 is used to check whether the solution to the subproblem is an approximate optimal solution.

In addition, it should be noted that the training samples are partitioned into p independent group (see line 2 of Algorithm 2) similar to AsyGCD, where p is the number of cores in a multi-core shared memory machine, and each group is associated to a core. Thus, we update the active set A for each group to ensure that, the active set A is partitioned into the corresponding groups (see line 18 of Algorithm 2).

Outer loop: The outer loop is mainly used for checking the termination condition and reconstructing the active set. Specifically, we use the global variables $M_{\hat{S}}$ and $m_{\hat{S}}$ to check the quality of the solution (see lines 3 and 22-24 of Algorithm 2). Given a tolerance ε, if the condition $M - m \le \varepsilon$ is satisfied, the ε-approximate solution of (1) is obtained, and the AsyAGCD algorithm terminate. If the condition $M - m \le \varepsilon$ is not satisfied, we need to reconstruct the active set and then run the inner loop.

3.3 Comparison with AsyGCD

In this section, we compare our AsyAGCD with AsyGCD with respect to the computational complexity. Table 1 provides the computational complexities of each iteration for

Input : The training set $\hat{S}$, a tolerance ε, MaxShrinkIter, an initial $\boldsymbol{\alpha} = \mathbf{0}$ and $G = \boldsymbol{p}$.
Output: The vector $\boldsymbol{\alpha}$.

Initialize $A = \{1, \cdots, l\}, \bar{M}_A = \infty, \bar{m}_A = -\infty, M_A = -\infty, m_A = \infty, \bar{G} = \mathbf{0}, k = 0, IterInner = 0, IterOuter = 0$.
Divide the training set into p groups.
while $M_{\hat{S}} - m_{\hat{S}} \geq \varepsilon$ **do**
  Each thread repeatedly performs the following updates in parallel:
  while $M_A - m_A \geq \varepsilon$ **do**
    if $k\%MaxShrinkIter \neq 0$ **then**
      Pick $i = arg\max_{i \in A} |\nabla_i^+ f(\boldsymbol{\alpha})|$ using G_A;
      compute δ_i^* by Eq.(4);
      for $j = 1, 2, \cdots, |A|$ **do**
        Update $G_A^j = G_A^j + \delta^* Q_A^{j,i}$ **using atomic update**;
      end
      Update $\alpha_i = \alpha_i + \delta^*$;
      Update $\bar{G}$ by Eq. (12) ;
      $k = k + 1$;
    else
      Calculate $M_A, m_A, \bar{M}_A, \bar{m}_A$;
      Shrink by Algorithm 3;
      Update the active set A for each group;
      $IterInner = IterInner + 1$;
    end
  end
  Reconstruct gradients G using $\bar{G}$ by Eq. (13);
  Calculate $M_{\hat{S}}$ and $m_{\hat{S}}$;
  Update $A = \{1, \cdots, \ell\}$;
  $IterOuter = IterOuter + 1$ and $k = 0$.
end
return the revised $\boldsymbol{\alpha}$.

Algorithm 2: AsyAGCD algorithm

Input : The active set A and its gradients G_A, the current $\boldsymbol{\alpha}_A$.
Output: The active set A.

Initialize $M_A = -\infty, m_A = \infty, \bar{M}_A = \infty, \bar{m}_A = -\infty$.
for $i \in A$ **do**
  $PG = 0$;
  if *($\alpha_i < C$ and $G_i < 0$) or ($\alpha_i > 0$ and $G_i > 0$)* **then**
    $PG = G_i$;
  end
  $M_A = \max(M_A, PG)$;
  $m_A = \min(m_A, PG)$;
end
Calculate $\bar{M}_A, \bar{m}_A$;
for $i \in A$ **do**
  if *($\alpha_i = 0$ and $G_i > \bar{M}_A$) or ($\alpha_i = C$ and $G_i < \bar{m}_A$)* **then**
    $A = A \backslash \{i\}$;
  end
end
return the active set A.

Algorithm 3: Shrinking Algorithm

AsyGCD and AsyAGCD, and the total computational complexities of AsyGCD and AsyAGCD (see the last row of Table 1).

For the AsyGCD algorithm (i.e., Algorithm 2), You et al. [You *et al.*, 2016] pointed out the computational complexity of greedy coordinate selection is $O(\ell)$. The computational complexity of updating the gradients is also $O(\ell)$. Assume the size of total iterations of AsyGCD (or AsyAGCD) is $IterTotal$, the total computational complexity of AsyGCD is $IterTotal \times O(\ell)$, please see the last row of Table 1.

For the AsyAGCD algorithm (i.e., Algorithm 2), we first consider the computational complexity for the inner loop. Specifically, the computational complexities of greedy coordinate selection, updating G_A, calculating M_A and m_A and shrinking are $O(|A|)$. The computational complexity of updating the values of $\bar{G}$ is $O(\ell)$. For the outer loop of Algorithm 2, The computational complexity of reconstructing gradients is $O(\ell \times (\ell - |A|))$. The computational complexity of calculating $M_{\hat{S}}$ and $m_{\hat{S}}$ is $O(\ell)$. Thus, the total computational complexity of AsyAGCD can be described as $(IterTotal + IterInner) \times O(|A|) + \#GbarCount \times O(\ell) + IterOuter \times O(\ell \times (\ell - |A|))$, where $\#GbarCount$ denotes the times of updating $\bar{G}$. Note that the above $|A|$ represents the average size of the active set A. To have a deep insight to the computational complexity of AsyAGCD, we give the following two remarks.

Remark 1. *Generally, we have the relationship $(IterTotal + IterInner) \gg \#GbarCount > IterOuter$. Thus, the computational complexity of our AsyAGCD is dominated by the average size of the active sets and the iteration number $IterTotal + IterInner$.*

Remark 2. *Generally, the active set size may become smaller and smaller, as the number of iterations increases. Then the average active set size $|A|$ of Table 1 is smaller accordingly and shrinking technique can shorten the training time.*

According to the above Remarks 1 and 2, if the number of iterations is large, our AsyAGCD would have a speedup compared to AsyGCD, because of using the active set technique. This is because the subproblem (8) would become very lightweight (i.e., $|A| \ll \ell$ in Table 1) and the cache hit ratio improves simultaneously (the cache strategy is widely used in [Chang and Lin, 2011; You *et al.*, 2015]). In addition, we can update $\bar{G}$ quickly because the size of A would become small enough. Thus, our AsyAGCD can converge much faster than AsyGCD for the second half of iterations.

On the contrary, if the number of iterations is too small, the active set size would be relatively large during the whole procedure. Thus, the active set technique may not be so useful. This is because that, the time saved by the active set

	Descriptions	AsyGCD (Algorithm 1)		AsyAGCD (Algorithm 2)	
		Steps	Computational complexity	Steps	Computational complexity
Inner loop	Pick i	3	$O(\ell)$	7	$O(\|A\|)$
	Compute δ_i^*	4	$O(1)$	8	$O(1)$
	Update G or G_A	5-7	$O(\ell)$	9-11	$O(\|A\|)$
	Update α_i	8	$O(1)$	12	$O(1)$
	Update $\bar{G}$	–	–	13	$O(\ell)$
	Calculate M_A, etc	–	–	16	$O(\|A\|)$
	Shrinking	–	–	17	$O(\|A\|)$
	Update A for each block	–	–	18	$O(\|A\|)$
Outer loop	Reconstruct gradients	–	–	22	$O(\ell \times (\ell - \|A\|)$
	Calculate $M_{\hat{S}}$ and $m_{\hat{S}}$	–	–	23	$O(\ell)$
Total computational complexity		$IterTotal \times O(\ell)$		$(IterTotal + IterInner) \times O(\|A\|)$ $+\#GbarCount \times O(\ell)$ $+IterOuter \times O(\ell \times (l - \|A\|))$	

Table 1: The comparison of computational complexities of AsyGCD and our AsyAGCD

technique cannot make up the time introduced by the active set technique (i.e., $IterInner \times O(|A|) + \#GbarCount \times O(\ell) + IterOuter \times O(\ell \times (\ell - |A|))$ as analyzed in Table 1). In this case, the AsyGCD algorithm without the active set technique may be faster than our AsyAGCD.

4 Experiments

In this section, we first give the experimental setup, then show our experimental results and discussions.

4.1 Experimental Setup

Design of experiments: In the experiments, we mainly compare the efficiency of our AsyAGCD with the existing state-of-art solver of SVMs. The compared algorithms used in our experiments include:

1. ***AsyAGCD***[1]: Our AsyAGCD algorithm which can solve C-SVC and ϵ-SVR. As mentioned before, AsyAGCD is defaultly with shrinking (i.e., the active set technique).
2. ***AsyGCD***[2]: The AsyGCD algorithm which was proposed by You et al. [You *et al.*, 2016]. Note that AsyGCD can only solve C-SVC.
3. ***AsyAGCD without shrinking***: A new version of AsyGCD without shrinking which can solve ϵ-SVR.
4. ***LIBSVM(OMP) with shrinking***[3]: The parallel version of LIBSVM with shrinking, where a column of kernel matrix is computed parallelly.

For C-SVC, the AsyAGCD, AsyGCD and LIBSVM(OMP) with shrinking methods are compared. Specifically, we compare the accuracies and objective values over the running time among these methods. Note that, because of the different formulation of C-SVC solved by LIBSVM, we only compare the objective values over the running time between AsyAGCD and AsyGCD. For ϵ-SVR, the AsyAGCD, AsyAGCD without shrinking and LIBSVM(OMP) with shrinking methods are compared. We compare the mean absolute errors (MAE) and the mean squared errors (MSE) over the running time among these methods.

Implementation: We implement our proposed AsyAGCD algorithm by C++ based on AsyGCD, where the shared memory parallel computation is handled via OpenMP [Chandra, 2001].We perform experiments on a 36-core two-socket Intel Xeon E5-2696 machine with 48 GB RAM, where each socket has 18 cores. All the experiments are running with the Gaussian kernel $K(x_1, x_2) = \exp(-\kappa\|x_1 - x_2\|^2)$, the parameter κ, C and ϵ are the default values at LIBSVM. $MaxShrinkIter$ in AsyAGCD is set as $10,000$. All the experiments are running at least 10 times with 20 cores, and using 24GB memory space for kernel caching.

Datasets: Table 2 summarizes the eight datasets used in experiments which divided into two parts according to the tasks, i.e., the binary classification (*BC*) and regression (*R*). They are from https://archive.ics.uci.edu/ml/datasets.html and https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. All the inputs of these datasets are mapped into the interval $[0, 1]$.

Task	Datasets	Training set	Testing set	Attributes
BC	Covtype	464,810	116,202	54
	Ijcnn1	49,990	91,701	22
	Webspam	280,000	70,000	254
	Skin-noskin	200,000	45,057	3
R	3D-spatial-network	391,387	43,487	3
	Cadata	18,576	2,064	8
	SliceLoc	48,150	5,350	386
	YearPredictionMSD	463,715	51,630	90

Table 2: Datasets used in the experiments

4.2 Experimental Results and Discussion

C-SVC: Fig. 1 shows the accuracies(objective values) v.s. the running time of different algorithms. From the top graphs, it is easy to find that both AsyAGCD and AsyGCD converge faster than the LIBSVM(OMP) in the most cases. It should be noted that, the advantages of AsyAGCD are not obvious on

[1]The code is available in https://sites.google.com/site/jsgubin/our-software-codes/AsyAGCD_code.zip.

[2]The code is available in https://github.com/cjhsieh/asyn_kernel_svm

[3]The code is available in http://www.csie.ntu.edu.tw/~cjlin/libsvm/

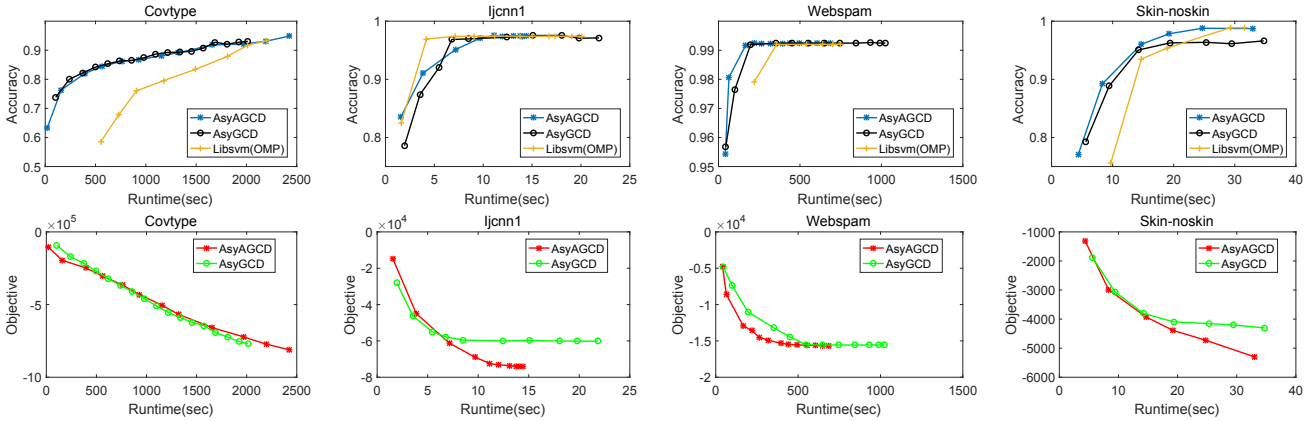

Figure 1: Running time v.s. accuracies (objective values) of different solvers for training C-SVC. The top graphs give accuracies results and the bottom graphs give the objective values results.

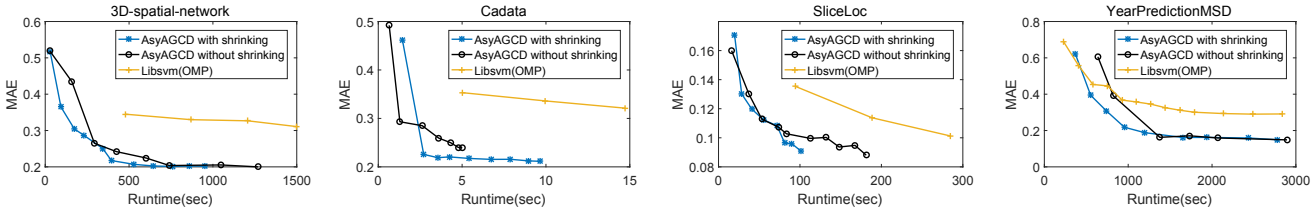

Figure 2: Running time v.s. MAEs of different solvers for training ϵ-SVR.

the Ijcnn1 dataset. This phenomenon shows our AsyAGCD is more applicable to handle large datasets. For the Covtype dataset, the performance of AsyAGCD and AsyGCD is similar. This is because only a small number of variables are shrunk in each shrinking procedure. Thus, the running time saved from the active set technique cannot make up the time consuming introduced by the shrinking technology, as discussed in Section 3.3.

From the bottom graphs in Fig. 1, it is easy to find that our AsyAGCD is faster than AsyGCD in terms of the objective values. This follows the comparison of the computational complexities of AsyGCD and AsyAGCD as discussed in Section 3.3. Note that, in ijcnn1 and skinnoskin dataset, it looks like that the convergence points of AsyGCD and AsyAGCD are different, which contradicts to the theoretical result that, AsyGCD and AsyAGCD should converge to a same value of the objective function. Actually, if running AsyGCD and AsyAGCD for enough time, they can converge to a same objective values. However, the running time shown in figures is limited, which leads that the convergence points of AsyGCD and AsyAGCD look different. This phenomenon also confirms that our AsyAGCD is faster than AsyGCD.

Remark 3. *In fact, we observe that most of computational time is wasted on selecting the active set and calculating the kernel vector. During the second half of iterations, selecting the active set is fairly rapid because the size of the active set becomes relatively small. Thus, our AsyAGCD can converge much faster than AsyGCD for the second half of iterations.*

ϵ-SVR: Fig.2 shows the MAE values v.s. the running time of different algorithms. It is easy to find that both AsyAGCD with and without shrinking converge faster than the LIBSVM(OMP). This is because AsyAGCD is an asynchronous parallel algorithm, and LIBSVM(OMP) is a synchronous parallel algorithm. Normally, asynchronous parallel computation is much faster than synchronous parallel computation. In addition, [Steinwart *et al.*, 2011] pointed out that solving an SVM without offset can be significantly faster.

5 Conclusion

In this paper, we propose an asynchronous accelerated greedy coordinate descent algorithm (AsyAGCD) for SVMs. Compared with the existing state-of-art solver AsyGCD, our AsyAGCD has the following two-fold advantages: 1) AsyAGCD is an accelerated version of AsyGCD and converge much faster than AsyGCD for the second half of iterations. 2) Our AsyAGCD can handle more SVM formulations than AsyGCD. We provide the comparison of computational complexity of AsyGCD and our AsyAGCD. Experiment results confirm that our AsyAGCD is much faster than the existing SVM solvers (including AsyGCD). In future, we plan to extend our AsyAGCD to other learning problems [Gu *et al.*, 2015a; 2015b; 2017; Gu and Ling, 2015; Gu *et al.*, 2018c; Huo *et al.*, 2018].

Acknowledgments

Dr. Heng Huang made scientific contributions to this paper. Unfortunately the conference chair doesn't allow us to change the author list. We acknowledge Dr. Heng Huang's contributions. This work was partially supported by the Nat-

ural Science Foundation of Jiangsu Province of China (No. BK20161534), Six talent peaks project (No. XYDXX-042) and the 333 Project (No. BRA2017455) in Jiangsu Province and the National Natural Science Foundation of China (No 61573191). BG and HH were partially supported by U.S. NSF-IIS 1302675, NSF-IIS 1344152, NSF-DBI 1356628, NSF-IIS 1619308, NSF-IIS 1633753, NIH R01 AG049371.

References

[Bertsekas, 1999] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[Birgin and Martínez, 2002] Ernesto G Birgin and José Mario Martínez. Large-scale active-set box-constrained optimization method with spectral projected gradients. *Computational Optimization and Applications*, 23(1):101–125, 2002.

[Boyd and Vandenberghe, 2004] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[Chandra, 2001] Rohit Chandra. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[Chiang *et al.*, 2016] Wei-Lin Chiang, Mu-Chu Lee, and Chih-Jen Lin. Parallel dual coordinate descent method for large-scale linear classification in multi-core environments. In *KDD*, pages 1485–1494, 2016.

[De Santis *et al.*, 2016] Marianna De Santis, Stefano Lucidi, and Francesco Rinaldi. A fast active set block coordinate descent algorithm for ℓ_1-regularized least squares. *SIAM Journal on Optimization*, 26(1):781–809, 2016.

[Duchi *et al.*, 2015] John C Duchi, Sorathan Chaturapruek, and Christopher Ré. Asynchronous stochastic convex optimization. *arXiv preprint arXiv:1508.00882*, 2015.

[Gu and Ling, 2015] Bin Gu and Charles Ling. A new generalized error path algorithm for model selection. In *International Conference on Machine Learning*, pages 2549–2558, 2015.

[Gu *et al.*, 2015a] Bin Gu, Victor S Sheng, Keng Yeow Tay, Walter Romano, and Shuo Li. Incremental support vector learning for ordinal regression. *IEEE Transactions on Neural networks and learning systems*, 26(7):1403–1416, 2015.

[Gu *et al.*, 2015b] Bin Gu, Victor S Sheng, Zhijie Wang, Derek Ho, Said Osman, and Shuo Li. Incremental learning for ν-support vector regression. *Neural Networks*, 67:140–150, 2015.

[Gu *et al.*, 2017] Bin Gu, Victor S Sheng, Keng Yeow Tay, Walter Romano, and Shuo Li. Cross validation through two-dimensional solution surface for cost-sensitive svm. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1103–1121, 2017.

[Gu *et al.*, 2018a] Bin Gu, Zhouyuan Huo, and Heng Huang. Asynchronous doubly stochastic group regularized learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1791–1800, 2018.

[Gu *et al.*, 2018b] Bin Gu, Xin Miao, Zhouyuan Huo, and Heng Huang. Asynchronous doubly stochastic sparse kernel learning. In *AAAI*, 2018.

[Gu *et al.*, 2018c] Bin Gu, De Wang, Zhouyuan Huo, and Heng Huang. Inexact proximal gradient methods for non-convex and non-smooth optimization. In *AAAI*, 2018.

[Hsieh *et al.*, 2008] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM, 2008.

[Huo *et al.*, 2018] Zhouyuan Huo, Bin Gu, Ji Liu, and Heng Huang. Accelerated method for stochastic composition optimization with nonsmooth regularization. In *AAAI*, 2018.

[Liu *et al.*, 2015] Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *The Journal of Machine Learning Research*, 16(1):285–322, 2015.

[Rahimi and Recht, 2008] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[Steinwart *et al.*, 2011] Ingo Steinwart, Don Hush, and Clint Scovel. Training svms without offset. *Journal of Machine Learning Research*, 12(Jan):141–202, 2011.

[Vapnik and Vapnik, 1998] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[Williams and Seeger, 2001] Christopher KI Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682–688, 2001.

[You *et al.*, 2015] Yang You, Haohuan Fu, Shuaiwen Leon Song, Amanda Randles, Darren Kerbyson, Andres Marquez, Guangwen Yang, and Adolfy Hoisie. Scaling support vector machines on modern hpc platforms. *Journal of Parallel and Distributed Computing*, 76:16–31, 2015.

[You *et al.*, 2016] Yang You, Xiangru Lian, Ji Liu, Hsiang-Fu Yu, Inderjit S Dhillon, James Demmel, and Cho-Jui Hsieh. Asynchronous parallel greedy coordinate descent. In *Advances In Neural Information Processing Systems*, pages 4682–4690, 2016.

[Zhao and Magoules, 2011] Hai-xiang Zhao and Frédéric Magoules. Parallel support vector machines on multi-core and multiprocessor systems. In *11th International Conference on Artificial Intelligence and Applications (AIA 2011)*. IASTED, 2011.