Pattern Recognition 109 (2020) 107592

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Solving large-scale support vector ordinal regression with asynchronous parallel coordinate descent algorithms

Bin Gu[a,b,1], Xiang Geng[a,1], Wanli Shi[a], Yingying Shan[a], Yufang Huang[c], Zhijie Wang[d], Guansheng Zheng[a,*]

[a] School of Computer & Software, Nanjing University of Information Science & Technology, Nanjing, PR China
[b] JD Finance America Corporation, USA
[c] eBay Inc., Shanghai, China
[d] Invision AI, Canada

ARTICLE INFO

Article history:
Received 8 December 2019
Revised 31 July 2020
Accepted 11 August 2020
Available online 14 August 2020

Keywords:
Asynchronous parallel
Coordinate descent
Support vector
Ordinal regression

ABSTRACT

Ordinal regression is one of the most influential tasks of supervised learning. Support vector ordinal regression (SVOR) is an appealing method to tackle ordinal regression problems. However, due to the complexity in the formulation of SVOR and the high cost of kernel computation, traditional SVOR solvers are inefficient for large-scale training. To address this problem, in this paper, we first highlight a special SVOR formulation whose thresholds are described implicitly, so that the dual formulation is concise to apply the state-of-the-art asynchronous parallel coordinate descent algorithm, such as AsyGCD. To further accelerate the training for SVOR, we propose two novel asynchronous parallel coordinate descent algorithms, called AsyACGD and AsyORGCD respectively. AsyACGD is an accelerated extension of AsyGCD using active set strategy. AsyORGCD is specifically designed for SVOR that it can keep the ordered thresholds when it is training so that it can obtain good performance with lower time. Experimental results on several large-scale ordinal regression datasets demonstrate the superiority of our proposed algorithms.

1. Introduction

In supervised learning, ordinal regression (OR) problems are very common and important. Because, lots of real world applications can be reduced as OR problems, such as medical research [1], credit rating [2], and information retrieval [3]. In OR problems, the training samples are marked by a set of ranks, which exhibits an ordering among the different categories. We use information retrieval as an example. Information retrieval needs to predict the relevance level of the references with respect to the given textual query, using a rank scale like: definitely, possibly, or not relevant. Thus the ordering information can be used to construct more accurate models and misclassification costs are not the same for different errors. OR is similar to standard regression [4–6] in maintaining the ordering information. However, the multiple thresholds between the ranks need to be learned which makes regression different to OR.

There have been a lot of OR algorithms proposed in the last two last decades, from support vector machine (SVM) [7–9] formulations [10,11] to Gaussian processes [12] or discriminant learning [13]. In this paper, we focus on the classical support vector ordinal regression (SVOR) method. Because of the good generalization performance, SVOR is an important method to tackle OR problems. More importantly, SVOR is much more straight-forward and interpretable.

There are several versions of SVOR. For example, the first SVOR algorithm [14] was proposed based on the idea of cumulative odds logit models. Nevertheless, the problem size is a quadratic function of the sample size due to loss function considers pairs of ranks to keep ordering information. To address this issue, several SVOR approaches were proposed to find $r-1$ parallel discrimination hyperplanes for the r ordered categories. Specifically, SVFMOR and SVSMOR[15] were proposed for OR problems based on the strategies of fixed margin and sum-of-margins. However, for these two SVOR algorithms, the thresholds inequalities (*i.e.*, $\boldsymbol{b}_1 \le \boldsymbol{b}_2 \le \cdots \le \boldsymbol{b}_{r-1}$) are not considered in their formulation. Thus, the thresholds may be unordered at the optimal solution, that makes these models violate the ordering distribution assumption.

* Corresponding author.
E-mail addresses: jsgubin@nuist.edu.cn (B. Gu), gengxiang@nuist.edu.cn (X. Geng), wanlishi@nuist.edu.cn (W. Shi), yyshan@nuist.edu.cn (Y. Shan), yufhuang@ebay.com (Y. Huang), zhijie@ualberta.ca (Z. Wang), zgs@nuist.edu.cn (G. Zheng).
[1] B. Gu and X. Geng make equal contribution to this work.

https://doi.org/10.1016/j.patcog.2020.107592
0031-3203/© 2020 Elsevier Ltd. All rights reserved.

To fix the problem of unordered thresholds in the fixed margin strategy of SVOR as mentioned above, two new SVOR formulations were proposed (*i.e.*, SVORIM and SVOREX) [11]. For SVOREX, they imposed the explicit ordinal inequalities constraints into the original SVOR formulation with the fixed margin strategy. Note that the dual problem of SVOREX have inequalities constraints due to the explicit ordinal inequalities constraints, while multiple equality constraints come from the multiple thresholds. For SVORIM, they allowed the instances in all the categories to contribute errors for each threshold. They proved that the ordinal inequalities of the thresholds can be satisfied automatically at the optimal solution of SVORIM. Without explicit ordinal inequalities constraints, SVORIM avoids the inequalities constraints in its dual problem. As reported in [16], SVOREX and SVORIM are the two of the best methods among 16 state-of-the-art threshold models based on the results on 41 benchmark datasets. The results also verify the significance of the ordered thresholds for OR.

However, in comparison with standard SVMs, the dual formulations of SVOR with ordered thresholds are more complicated because multiple inequalities or equality constraints are involved in the formulation as we describe above. Due to the complicated formulations, large-scale training for SVOR is still vacant as far as we know. An effective incremental method has been proposed for SVOR, which is very suitable in the incremental environment [17]. However, this method needs to compute an inverse matrix of the size of $(2r-2+m)^2$ in every iteration which results in a huge computational cost in the large-scale problems, where m is the size of margin support vectors strictly on the margins.

Support vector methods [18–21] usually suffer from the high computational cost and memory requirements, because the kernel matrix with the size of $O(l^2)$ need to be computed and stored where l is the training sample size [22–24]. There are several methods proposed to accelerate kernel SVM training on large-scale datasets. Kernel approximation approaches are trying to approximate the kernel matrix by a $l \times m$ approximation matrix, then solving a linear SVM [25,26]. However, as analyzed in [27,28], the m need to be $O(n)$ to obtain a good generalization ability. To address this issue, asynchronously parallel [29] coordinate descent algorithms (AsyCD) have been proposed to solve these problems, such as asynchronously stochastic coordinate descent algorithms (AsySCD) algorithm [30,31] and asynchronous parallel greedy coordinate descent (AsyGCD) algorithm [32]. Compared with AsySCD, AsyGCD algorithm can achieve a much faster convergence speed due to the greedy selection of updated coordinates. However, AsyGCD is still not scalable enough for SVOR.

To address the large-scale training of SVOR with ordered thresholds, in this paper, we first highlight a concise SVOR formulation based on the idea of SVORIM. Its dual formulation is a box constrained quadratic programming problem on which we can apply AsyGCD easily. We theoretically show that this formulation maintains the ordinal thresholds at the optimal solution. Then we propose two novel asynchronous parallel coordinate descent algorithms, called AsyACGD and AsyORGCD respectively. AsyACGD is an accelerated extension of AsyGCD using the active set strategy. Active set (or shrinking technique) is a technique to improve the efficiency of dual coordinate descent methods [33]. Specifically, it tries to identify the bounded elements and remove them from the active set, thus, the size of the optimization problem can be smaller. The most famous application of the active set technique is LIBSVM [34]. AsyORGCD is specifically designed for SVOR such that keeping the ordered thresholds during the training process. Thus, a early stopping rule can be used to obtain good performance with much more less time. Specifically, we let each thread update on their local gradients block until they reach optimal locally, AsyORGCD updates the gradients of optimal threads from the global ones every certain iterations. Experimental results on several large-scale ordinal regression datasets demonstrate that our algorithm is much faster than the existing state-of-the-art SVOR solvers and AsyGCD solver while retaining the similar generalization performance.

Table 1
Notations.

Symbol	Meaning
x	The lower-case letter denotes an element.
$\boldsymbol{x}$	The bold lower-case letter denotes a vector.
$\boldsymbol{x}_i$	The i-th element of vector $\boldsymbol{x}$.
$\boldsymbol{K}$	The bold upper-case letter denotes a matrix.
$\boldsymbol{K}_{cd}$	The c-th row and d-th column element of $\boldsymbol{K}$
$\mathcal{S}$	The flourish denotes an set.

Note that our AsyACGD algorithm have been published in the 27th International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence [35], this paper is a significant extension to [35]. In [35], we only compare the AsyAGCD and AsyGCD for classification and regression. While, in this paper, we explore how asynchronous parallel coordinate descent algorithms (including AsyAGCD) work on SVOR. Furthermore, based on the experimental observations, we propose a novel AsyORGCD to hold ordered thresholds during the training process. For saving space, we did not repeatedly show the results of classification and regression in this paper.

Contributions. The main contributions of this paper are summarized as follows.

1. To make the SVOR suitable for AsyCD algorithm, we highlight a concise SVOR formulation which can be seen as a special case of the general OR framework in [36]. We contribute to first highlight this useful formulation for AsyCD algorithms with some simple but meaningful theoretical analysis. As far as we know, it is the first time that solving SVOR with asynchronous parallel coordinate descent algorithms.
2. To further speeded up AsyGCD, in this paper, we propose an asynchronous accelerated greedy coordinate descent algorithm (AsyAGCD) to further accelerate AsyGCD by the technique of active set. It is the first time that exploring how AsyAGCD algorithm works on SVOR. Note that, in [32,35], AsyGCD and AsyAGCD is only used for classification and regression.
3. To keep the ordered thresholds when it is training, we propose an novel asynchronous OR greedy coordinate descent algorithm (AsyORGCD), so that it can obtain good performance with lower time. Our experiments show that our AsyAGCD and AsyORGCD are much more efficient SVOR than existing the existing state-of-the-art SVOR solvers and AsyGCD solver.

Organization. The remainder of this paper is organized as follows. In Section 2, we introduce the concise SVOR formulation. In Section 3, we first give a brief review of AsyGCD, then we propose our AsyAGCD and AsyORGCD with some analyses. We present the experimental results in Section 4. The final section provides some concluding remarks.

Notations. Throughout this paper, we use notations in Table 1.

2. The concise SVOR formulation

In this section, we first introduce a concise primal problem of SVOR. Then, we give its dual formulation and prediction function.

2.1. Primal formulation

Consider an OR training set $\mathcal{S} := \{(\boldsymbol{x}_i, Y_i)|i = 1, 2, \cdots, n\}$ with $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$ and $Y_i \in \mathcal{Y} = \{1, 2, \cdots, r\}$, where $\mathcal{Y}$ represents the category of each sample. The number of samples in k-th rank is denoted as n^k. To learn a mapping function $h(\cdot) : \mathcal{X} \to \mathcal{Y}$, an OR

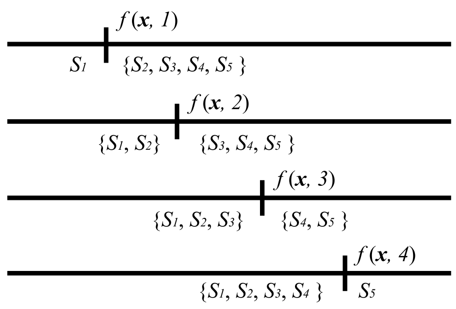


Fig. 1. $r-1$ parallel discrimination hyperplanes with r=5. $\mathcal{S}_k$ denotes the training set associated to k-th category.

problem can be reduced to $r-1$ binary classification problems [37]. Then, they considered $r-1$ parallel discrimination hyperplanes, $f(\boldsymbol{x}, j) = \boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{b}_j,\ j = 1, 2\cdots, r-1,$ with $\boldsymbol{b}_1 \leq \boldsymbol{b}_2 \leq \cdots \leq \boldsymbol{b}_{r-1}$, where $f(\boldsymbol{x}, j)$ corresponds to the j-th binary classification problem (please see Fig. 1). Thus, the mapping function $h(\cdot)$ can be obtained as $h(\boldsymbol{x}) := 1 + \sum_{j=1}^{r-1} \mathbb{k}[f(\boldsymbol{x}, j) > 0]$[2]

As mentioned above, the parallel-hyperplanes based mapping function highly depends the ordinal inequalities constraints $\boldsymbol{b}_1 \leq \boldsymbol{b}_2 \leq \cdots \leq \boldsymbol{b}_{r-1}$. However, the explicit ordinal inequalities constraints (SVOREX) could cause inequalities constraints in the dual problem. Without explicit ordinal inequalities constraints, SVORIM avoids the inequalities constraints in its dual problem, while multiple equality constraints still come from the multiple thresholds.

2.1.1. Thresholds embedding

In order to eliminate the equality constraints in SVORIM dual problem, we embed bias terms $\{\boldsymbol{b}_1, \cdots, \boldsymbol{b}_{r-1}\}$ into inner products to represent the bias terms implicitly. For the sample of the j-th binary classification problem, we add $r-1$ dimensions to the i-th instance w.r.t. k-th category as follows.

$$\hat{\boldsymbol{x}}_{ki}^j \leftarrow [\boldsymbol{x}_{ki}, \overbrace{\underbrace{0, \cdots, 0}_{j-1}, 1, \underbrace{0, \cdots, 0}_{r-j-1}}^{J_j}] \tag{1}$$

where j runs over $1, \cdots, r-1$, and J_j denotes the extended part of the new sample $\hat{\boldsymbol{x}}_{ki}^j$. Now we can define a new data set $\hat{\mathcal{S}}$,

$$\begin{aligned} \hat{\mathcal{S}}^j &= \{\{(\hat{\boldsymbol{x}}_{ki}^j, y_{ki}^j = -1)\}_{i=1}^{n^k}\}_{k=1}^{j} \cup \{\{(\hat{\boldsymbol{x}}_{ki}^j, y_{ki}^j = 1)\}_{i=1}^{n^k}\}_{k=j+1}^{r} \\ \hat{\mathcal{S}} &= \bigcup_{j=1}^{r-1} \hat{\mathcal{S}}^j. \end{aligned} \tag{2}$$

where the subset $\hat{\mathcal{S}}^j$ related to the j-th binary classification problem. Now the OR problem is reduce to single binary classification.

2.1.2. Variant kernel function

Due to the kernel method[38,39], we can learn a nonlinear model directly using kernel functions, instead of computing the inner product in the explicit RKHS. Suppose we are given a symmetric positive definite (SPD) kernel K which can be a common kernel function like the polynomial kernel, Laplacian kernel or Gaussian kernel etc.[40] and its feature map function $\phi(\cdot)$:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle\phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle \tag{3}$$

For the nonlinear setting, due to the effects of the kernel function, the implicitly included bias terms $\boldsymbol{b}_j$ (which can be directly embedded in linear SVOR) is also mapped into the kernel space. To guarantee the thresholds are properly ordered at the optimal solution and further reduce the computational complexity, we introduce a variant kernel function to uniform the linear and nonlinear problems. Similar to [36], based on kernel K a new kernel function $\widetilde{K}$ was proposed:

$$\widetilde{K}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}') = K(\boldsymbol{x}, \boldsymbol{x}') + J_j \cdot J_{j'} = \langle\phi(\hat{\boldsymbol{x}}), \phi(\hat{\boldsymbol{x}}')\rangle \tag{4}$$

The new map function $\hat{\phi}(\hat{\boldsymbol{x}}) = [\phi(\boldsymbol{x}), J_j]$. Theoretically, we have

Theorem 2.1. *(The proof can be found in Appendix A) If $K(\boldsymbol{x}, \boldsymbol{x}')$ is a SPD kernel, then the new kernel function $\widetilde{K}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}')$ as defined in (4) is a SPD kernel,.*

Remark 2.1. We denote $\boldsymbol{K}$ as the kenerl matrix of $K(\boldsymbol{x}, \boldsymbol{x}')$ on original input set S. The L^2 size kernel matrix of $\widetilde{K}$ on $\hat{\mathcal{S}}$ could be computed by the element of $\boldsymbol{K}$ which is the size of l^2 plus the element of matrix $\boldsymbol{J}$ correspondingly, where the element of c row d column is $\boldsymbol{J}_{cd} = J_c \cdot J_d$. $\boldsymbol{J}_{cd} = J_c \cdot J_d$ can be computed in $O(1)$ by determining if J_c and J_d are equal. Thus our framework only needs store and compute the kernel matrix with the size $O(l^2)$ although our problem size is $l * (r-1)$.

2.1.3. Primal SVOR formulation

Correspondingly, the parameter $\boldsymbol{w}$ in the hyperplane can be extended implicitly as follows.

$$\hat{\boldsymbol{w}} \leftarrow [\boldsymbol{w}, -\boldsymbol{b}] \tag{5}$$

Thus, the hyperplane for the j-th binary classification problem can be denoted by $f(\boldsymbol{x}, j) = \langle\hat{\boldsymbol{w}}, \phi(\hat{\boldsymbol{x}})\rangle$. Note that it is the same with the one in primal SVORIM. Based on the new data format, new kernel function and new hyperplane as discussed above, and following the rule of SVORIM, we can give our primal SVOR formulation as follows.

$$\begin{aligned} \min_{\hat{\boldsymbol{w}}, \xi, \xi^*} \quad & \frac{1}{2}\langle\hat{\boldsymbol{w}}, \hat{\boldsymbol{w}}\rangle + C\sum_{j=1}^{r-1}\left(\sum_{k=1}^{j}\sum_{i=1}^{n^k}\xi_{ki}^j + \sum_{k=j+1}^{r}\sum_{i=1}^{n^k}\xi_{ki}^{*j}\right) \\ s.t. \quad & \langle\hat{\boldsymbol{w}}, \hat{\phi}(\hat{\boldsymbol{x}}_i^k)\rangle \leq -1 + \xi_{ki}^j,\ \xi_{ki}^j \geq 0, for\ k = 1, \cdots, j\ \ i = 1, \cdots, n^k; \\ & \langle\hat{\boldsymbol{w}}, \hat{\phi}(\hat{\boldsymbol{x}}_i^k)\rangle \geq 1 - \xi_{ki}^{*j},\ \xi_{ki}^{*j} \geq 0, for\ k = j+1, \cdots, r\ \ i = 1, \cdots, n^k; \end{aligned} \tag{6}$$

where j runs over $1, \cdots, r-1$. More importantly, we have (Theorem 2.2, the proof can be found in Appendix B)

Theorem 2.2. *(The proof can be found in Appendix B) If samples and parameter are extended by (1) and (5) respectively, and the extended kernel (4) is used, we can have that the implicit thresholds solved by (6) have the partial order, i.e., $\boldsymbol{b}_1 \leq \boldsymbol{b}_2 \leq \cdots \leq \boldsymbol{b}_{r-1}$.*

2.2. Dual SVOR formulation and prediction function

For simplicity, we redfine the index of set $\hat{\mathcal{S}}$, we dnote the index of t-th sample related to j-th binary problem as $i = (j-1) * n + t$, the dual problem of the new SVOR (6) can be formulated as follows.

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \sum_{i=1}^{L}\boldsymbol{\alpha}_i \\ \text{s.t.} \quad & 0 \leq \boldsymbol{\alpha}_i \leq C,\ i = 1, \cdots, L \end{aligned} \tag{7}$$

where Q is an $L \times L$ matrix associated to $\hat{\mathcal{S}}$ with the element of c-th row and d-th column denoted by $Q_{cd} = y_c y_d \widetilde{K}(\hat{\boldsymbol{x}}_c, \hat{\boldsymbol{x}}_d)$, and $\boldsymbol{\alpha}$ is a L-vector. Once the optimal solution $\boldsymbol{\alpha}$ is obtained, given a new testing sample $\boldsymbol{x}'$ and its extended data form $\hat{\boldsymbol{x}}'^j \leftarrow [\boldsymbol{x}', J^j]$, the predictive value for j-th ordinal can be obtained by the following formula.

$$f(\hat{\boldsymbol{x}}'^j) = \langle\hat{\boldsymbol{w}}, \hat{\phi}(\hat{\boldsymbol{x}}'^j)\rangle = \sum_{i=1}^{L}\boldsymbol{\alpha}_i y_i \widetilde{K}(\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{x}}'^j) \tag{8}$$

[2] The boolean function $\mathbb{k}[\cdot]$ is 1 if the inner condition is true, and 0 otherwise.

Thus, the prediction function can be obtained by $h(\boldsymbol{x}) := 1 + \sum_{j=1}^{r-1} \mathbb{1}[f(\hat{\boldsymbol{x}}^j) > 0]$.

3. Asynchronous parallel coordinate descent methods for SVOR

In this section, for solving the dual SVOR problem (7), we first give a brief review of AsyGCD, then we propose our AsyAGCD and AsyORGCD. Finally, we give some analysis for AsyAGCD and AsyORGCD.

3.1. AsyGCD algorithm

For each parallel worker, greedy coordinate descent (GCD) algorithm selects the i-th coordinate greedily according to $|\nabla_i^+ f(\boldsymbol{\alpha})|$, where the projected gradient $\nabla_i^+ f(\boldsymbol{\alpha})$ is defined as follows.

$$\nabla_i^+ f(\boldsymbol{\alpha}) = \boldsymbol{\alpha}_i - P_{[0,C]}(\boldsymbol{\alpha}_i - \nabla_i f(\boldsymbol{\alpha})), \tag{9}$$

where $P_\Omega(\alpha)$ is the Euclidean projection of $\boldsymbol{\alpha}$ onto Ω, as mentioned in [32] the step size is set as $1/Q_{ii}$. the change for variable $\boldsymbol{\alpha}_i$ can be computed by:

$$\delta_i^* = P_{[0,C]}(\boldsymbol{\alpha}_i - \nabla_i f(\boldsymbol{\alpha})/Q_{ii}) - \boldsymbol{\alpha}_i, \tag{10}$$

The success of AsyGCD in solving kernel SVM is mainly due to the maintenance of the gradient $G_i := \nabla_i f(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i - 1$ by $G_i = G_i + \delta^* \boldsymbol{q}_i$, which can reduce $O(L)$ time for computing $(Q\boldsymbol{\alpha})_i$ from scratch to $O(1)$. To reduce the write-write conflicts in the implementation of asynchronous parallel computation, the variables $\boldsymbol{\alpha}$ is partitioned into several subsets, i.e.,

$$\hat{\mathcal{S}}_1 \cup \hat{\mathcal{S}}_2 \cup \cdots \cup \hat{\mathcal{S}}_n = \{1, 2, \cdots, L\}, \quad \textit{where } \hat{\mathcal{S}}_i \cap \hat{\mathcal{S}}_j = \emptyset, \ \forall i \neq j,$$

where n is normally the number of threads in a multi-core shared memory machines. Each worker conducts GCD iterations in one set $\hat{\mathcal{S}}_t$. We summarize our AsyGCD in Algorithm 1.

Algorithm 1: AsyGCD Algorithm.

```
Input  : The training set $\hat{\mathcal{S}}$.
Output: The vector $\alpha$.
Initialize $\alpha = \mathbf{0}$ and $G = -\mathbf{1}$;
Each thread $t$ repeatedly performs the following updates in
parallel:for $k = 1, 2, \cdots$ do
    Pick $i = \arg\max_i |\nabla_i^+ f(\alpha)|$ using $G$;
    compute $\delta_i^*$ by Eq. (10);
    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 $\alpha$.
```

3.2. AsyAGCD Algorithm

In this subsection, we propose an accelerate extension of AsyGCD via active set technique. To present the active set technique smoothly, we first give the Karush-Kuhn-Tucker (KKT) conditions to the new SVOR formulation (7), then give our active set technique to (7).

3.2.1. KKT Conditions

According to the KKT theory [41], the KKT conditions of (7) are presented as follows.

$$\nabla_i^P f(\boldsymbol{\alpha}) = \begin{cases} \nabla_i f(\boldsymbol{\alpha}) & if \ \ 0 < \boldsymbol{\alpha}_i < C \\ \min(0, \nabla_i f(\boldsymbol{\alpha})) & if \ \ \boldsymbol{\alpha}_i = 0 \\ \max(0, \nabla_i f(\boldsymbol{\alpha})) & if \ \ \boldsymbol{\alpha}_i = C \end{cases} \tag{11}$$

where $\nabla_i f(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i - \boldsymbol{e}_i$. If $\boldsymbol{\alpha}$ satisfies the KKT conditions, we have that

$$\nabla_i^P f(\boldsymbol{\alpha}) = 0, \quad \forall i = 1, \cdots, L \tag{12}$$

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., $M \equiv \max_i \nabla_i^P f(\boldsymbol{\alpha})$ and $m \equiv \min_i \nabla_i^P f(\boldsymbol{\alpha})$, later we will used in (14)) measure how the solution $\boldsymbol{\alpha}$ violates the KKT conditions.

3.2.2. Active set technique

We use the active set technique to accelerate the AsyGCD algorithm by solving a smaller optimization problem. Specifically, given an active set A (a subset of $\{1, \cdots, L\}$), correspondingly, we can define an inactive set as $\bar{A} = \{1, \cdots, l\} - A$. If we have the prior knowledge that the variables $\boldsymbol{\alpha}_{\bar{A}}$ are fixed, and only the variables $\boldsymbol{\alpha}_A$ is active, the original optimization (7) can be reduced as a smaller optimization problem (13) as follows.

$$\begin{aligned} \min_{\boldsymbol{\alpha}_A} \ & \frac{1}{2}\boldsymbol{\alpha}_A^T Q_{AA}\boldsymbol{\alpha}_A + (Q_{A\bar{A}}\boldsymbol{\alpha}_{\bar{A}} - \boldsymbol{e}_A)^T \boldsymbol{\alpha}_A \\ s.t. \ & 0 \le \boldsymbol{\alpha}_i \le C, \quad \forall i \in A \end{aligned} \tag{13}$$

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 (13). 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 3.1.

Theorem 3.1. *(See Appendix for the proof) Assume* $\{\boldsymbol{\alpha}^k\}_{k=1}^\infty$ *is an infinite sequence of (7) 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. 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 3.1 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 3.1. Specifically, 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. $\boldsymbol{\alpha}_i = 0$ and $\nabla_i f(\boldsymbol{\alpha}) > \bar{M}$.
2. $\boldsymbol{\alpha}_i = C$ and $\nabla_i f(\boldsymbol{\alpha}) < \bar{m}$.

where

$$\bar{M} = \begin{cases} M & if \ M > 0 \\ \infty & otherwise \end{cases}, \quad \bar{m} = \begin{cases} m & if \ m < 0 \\ -\infty & otherwise \end{cases} \tag{14}$$

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 (13) due to the following relation:

$$G_A = (Q\boldsymbol{\alpha})_A - \boldsymbol{e}_A = (Q_{AA}\boldsymbol{\alpha}_A) + (Q_{A\bar{A}}\boldsymbol{\alpha}_{\bar{A}}) - \boldsymbol{e}_A \tag{15}$$

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 $\bar{G} \in \mathbb{R}^L$ throughout iterations [34].

$$\bar{G}_i = C \sum_{j:\boldsymbol{\alpha}_j = C} Q_{ij}, \ \ i = 1, \cdots, L \tag{16}$$

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

$$G_i = \sum_{j=1}^{L} Q_{ij}\boldsymbol{\alpha}_j - e_i = \bar{G}_i - e_i + \sum_{\substack{j \in A \\ 0 < \boldsymbol{\alpha}_j < C}} Q_{ij}\boldsymbol{\alpha}_j \tag{17}$$

Eq. (17) can be used to reduce the computational cost of G_i because normally there exists a large proportion of samples are bounded (i.e., $\boldsymbol{\alpha}_j = 0$ or $\boldsymbol{\alpha}_j = C$). Thus, we can just focus on the computation on the elements in the active set A.

3.2.3. AsyAGCD

With the active set technique, we develop the AsyAGCD algorithm as presented in Algorithm 2. Algorithm 2 includes two parts, i.e., the inner loop and the outer loop.

Algorithm 2: AsyAGCD algorithm.

```
Input  : The training set $\hat{\mathcal{S}}$, a tolerance $\varepsilon$, MaxShrinkIter.
Output: The vector $\boldsymbol{\alpha}$.
1  Initialize $\boldsymbol{\alpha} = \mathbf{0}$, $G = -\mathbf{1}$,
   $A = \{1, \cdots, L\}, \bar{M}_A = \infty, \bar{m}_A = -\infty, M_A = -\infty, m_A = \infty, \bar{G} = \mathbf{0}, k = 0, IterInner = 0, IterOuter = 0$.
2  Divide the training set into $n$ groups.while $M_{\hat{\mathcal{S}}} - m_{\hat{\mathcal{S}}} \geq \varepsilon$ do
3      Each thread repeatedly performs the following updates in parallel:while $M_A - m_A \geq \varepsilon$ do
4          if $k \% MaxShrinkIter \neq 0$ then
5              Pick $i = arg\max_{i \in A} |\nabla_i^+ f(\boldsymbol{\alpha})|$ using $G_A$;
6              compute $\delta_i^*$ by Eq. (10);
7              for $j = 1, 2, \cdots, |A|$ do
8                  Update $G_A^j = G_A^j + \delta^* Q_A^{j,i}$ using atomic update;
9              end
10             Update $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_i + \delta^*$;
11             Update $\bar{G}$ by Eq. (16) ;
12             $k = k + 1$;
13         else
14             Calculate $M_A, m_A, \bar{M}_A, \bar{m}_A$;
15             Shrink by Algorithm 3;
16             Update the active set $A$ for each group;
17             $IterInner = IterInner + 1$;
18         end
19     end
20     Reconstruct gradients G using $\bar{G}$ by Eq. (17);
21     Calculate $M_{\hat{\mathcal{S}}}$ and $m_{\hat{\mathcal{S}}}$;
22     Update $A = \{1, \cdots, L\}$;
23     $IterOuter = IterOuter + 1$ and $k = 0$.
24 end
25 return the revised $\boldsymbol{\alpha}$.
```

Inner loop: The inner loop (see lines 5–21 of Algorithm 2) is mainly to optimize the subproblem (13) 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, 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).

Algorithm 3: Shrinking Algorithm.

```
Input  : The active set $A$ and its gradients $G_A$, the current $\boldsymbol{\alpha}_A$.
Output: The active set $A$.
1  Initialize $M_A = -\infty, m_A = \infty, \bar{M}_A = \infty, \bar{m}_A = -\infty$.
2  for $i \in A$ do
3      $PG = 0$;
4      if $(\boldsymbol{\alpha}_i < C$ and $G_i < 0)$ or $(\boldsymbol{\alpha}_i > 0$ and $G_i > 0)$ then
5          $PG = G_i$;
6      end
7      $M_A = \max(M_A, PG)$;
8      $m_A = \min(m_A, PG)$;
9  end
10 Calculate $\bar{M}_A, \bar{m}_A$;
11 for $i \in A$ do
12     if $(\boldsymbol{\alpha}_i = 0$ and $G_i > \bar{M}_A)$ or $(\boldsymbol{\alpha}_i = C$ and $G_i < \bar{m}_A)$ then
13         $A = A \backslash \{i\}$;
14     end
15 end
16 return the active set $A$.
```

Outer loop: The outer loop is mainly used for checking the termination condition and reconstructing the active set. Specifically, we use the global variables $M_{\hat{\mathcal{S}}}$ and $m_{\hat{\mathcal{S}}}$ to check the quality of the solution (see lines 3 and 22–24 of Algorithm 2). Given a tolerance ε, if the condition $M_{\hat{\mathcal{S}}} - m_{\hat{\mathcal{S}}} \leq \varepsilon$ is satisfied, the ε-approximate solution of (7) is obtained, and the AsyAGCD algorithm terminate. Otherwise, we need to reconstruct the active set and then run the inner loop.

3.3. AsyORGCD

As we will show in Section 4, AsyGCD and AsyAGCD can achieve fast convergence rate for OR formulation (7). However, a bad phenomenon when using these two algorithms is that the thresholds are disordered until they approach the optimal as shown in Fig. 2. As mentioned above, the ordered thresholds are critical to a good generalization performance. Actually, the ordered thresholds cause the fluctuation on evaluation metrics at training process as shown in Figs. 4 and 5. If an algorithm can hold the ordered thresholds when the objective value going down, we can use early stopping rule to achieve a good generalization performance with much lower time.

AsyORGCD is an interesting by-product when we test the AsyGCD and AsyAGCD for OR model. Asynchronous algorithms enjoy the speed up of fully using all computing resource, however, these algorithms suffer from the conflicts cause by asynchronism. A major embodiment of conflicts is that the gradients hold by AsyGCD and AsyAGCD may be fake, which meaning they are not even old, they may not exist when algorithms run sequentially. The conflicts are the main reason for the disordered thresholds because we can hold the ordered thresholds in the middle stage of training instead of the end stage when we using a single thread for AsyGCD and AsyAGCD. To enjoy the property of ordered thresholds, we develop the AsyORGCD with fewer conflicts as we summarize in Algorithm 4.

Specifically, each thread of AsyORGCD also does a GCD step similar to AsyGCD and AsyAGCD (see lines 13–26). However, to reduce the conflicts each thread holds a local gradients vector G^t (the length of G^t is same as the size of the coordinate subset $\hat{\mathcal{S}}_t$) and the GCD step updates according to the local gradients instead of the global ones (see line 14). While to maintain the global information the GCD step updates both the local and global gradients (see lines 19–24). When one thread reach optimal on the local gradients (i.e.

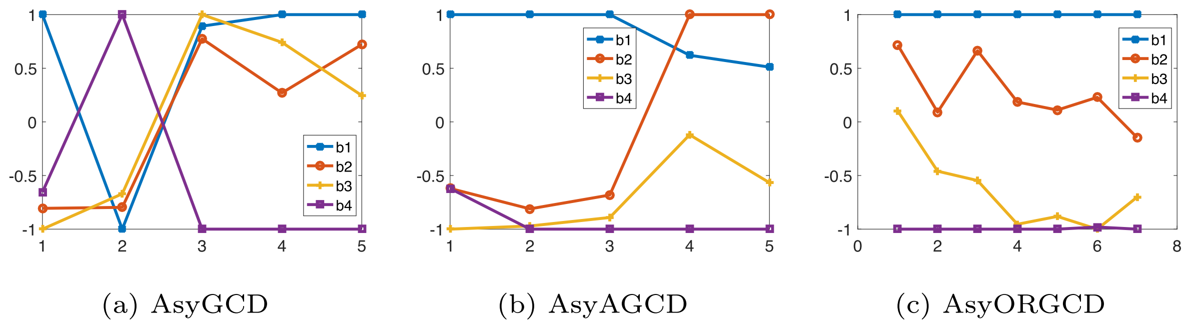


Fig. 2. The thresholds path of different AsyCD algorithms for SVOR on MR data set, the values are mapped to [-1,1] for a clear display.

Algorithm 4: AsyORGCD Algorithm.

```
Input  : The training set $\hat{S}$.
Output: The vector $\alpha$.
1  Redefine the training set $\hat{S}$ as Eqs. (1) and (2);
2  Initialize $\alpha = 0$, $G = -1$, and $G^t = -1$, $flag^t$ =NotOptimal for $t = 1 \cdots n$;
3  for $OuterIter = 1, 2, \cdots$ do
4      if $flag^t$==Optimal for all $t = 1 \cdots n$ then
5          End the outer loop.
6      end
7      for $t = 1, 2, \cdots n$ do
8          if $flag^t$==Optimal then
9              Update $G^t$ according to $G$;
10             $flag^t$=NotOptimal;
11         end
12     end
13     Each thread $t$ repeatedly performs the following updates in parallel:for $k = 1, 2, \cdots$ MaxInnerIter do
14         Pick $i = \arg\max_i |\triangledown_i^+ f(\alpha)|$ using $G^t$;
15         compute $\delta_i^*$ by Eq. (10);
16         if $|\triangledown_i^+ f(\alpha)| < \epsilon$ then
17             $flag^t$=Optimal;
18         end
19         for $j = 1, 2, \cdots, L$ do
20             $G_j = G_j + \delta_i^* Q_{j,i}$ using atomic update;
21         end
22         for $j = 1, 2, \cdots, Size(\hat{S}_t)$ do
23             $G_j^t = G_j^t + \delta_i^* Q_{j,i}$;
24         end
25         $\alpha_i = \alpha_i + \delta_i^*$.
26     end
27 end
28 return $\alpha$.
```

the max local projected gradient $|\nabla_i^+ f(\alpha)| < \epsilon$), the local gradients update from global ones, then repeat this process until global convergence (see lines 7–12). If all threads reach the optimal AsyORGCD stops (see lines 4–6), but we prefer to use some early stop rule to save time.

3.4. Time complexity

Table 2 provides the computational complexities of each iteration for AsyGCD, AsyAGCD, and AsyORGCD, and the total computational complexities of AsyGCD, AsyAGCD, and AsyORGCD (see the last row of Table 2). We denote L as the number of the extended training set. During the whole GCD procedure,

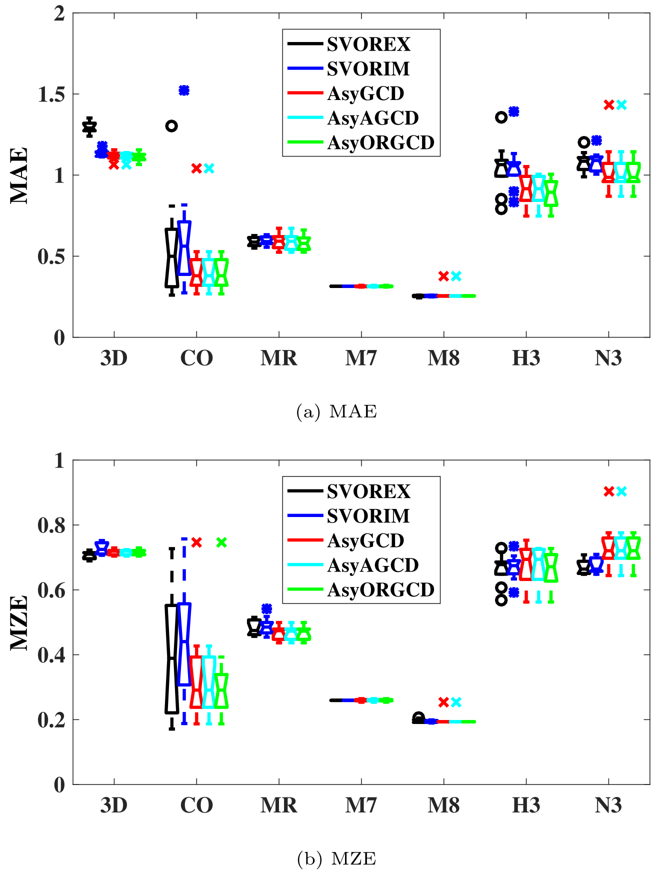
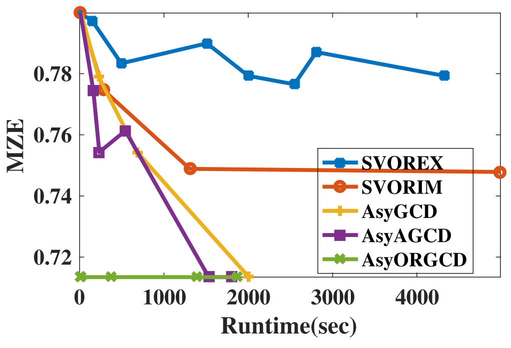
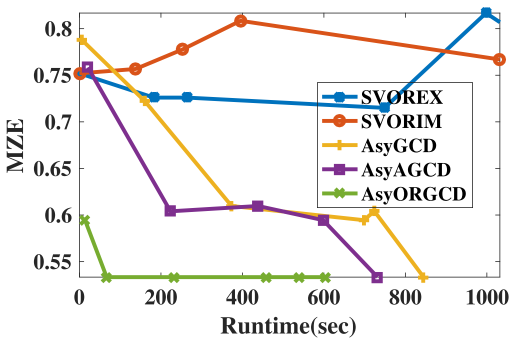


Fig. 3. The MAEs and MZEs boxplot of different solvers for SVOR on different data sets.

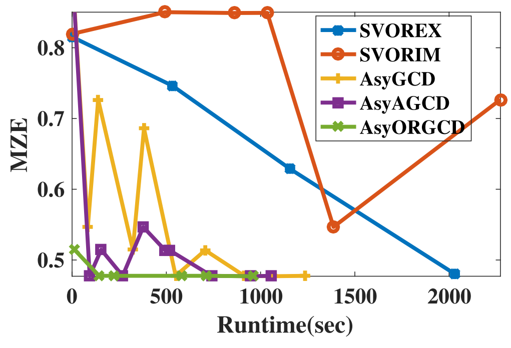
each update only costs $O(L)$ using the trick of maintaining G (i.e., the gradient G is available in memory). Considering the size of the total iterations *IterTotal*, the whole time complexity of AsyGCD and AsyORGCD can be presented as $O(L^* IterTotal)$. For the AsyAGCD algorithm, the computational complexity seriously depends on the average size $|A|$ of the active set. The total computational complexity of AsyAGCD can be described as $(IterTotal + IterInner) \times O(|A|) + GbarCount \times O(L) + IterOuter \times O(L \times (L - |A|))$, where *GbarCount* denotes the times of updating $\bar{G}$. Since $(IterTotal + IterInner) \gg GbarCount > IterOuter$, the computational complexity of our AsyAGCD is dominated by the average size of the active sets and the iteration number $IterTotal + IterInner$. If the active set size becomes enough small as the iteration continues, AsyAGCD will enjoy a good speedup compared with AsyGCD and AsyORGCD.



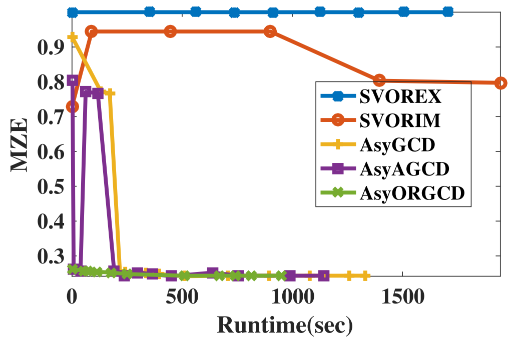
(a) 3D



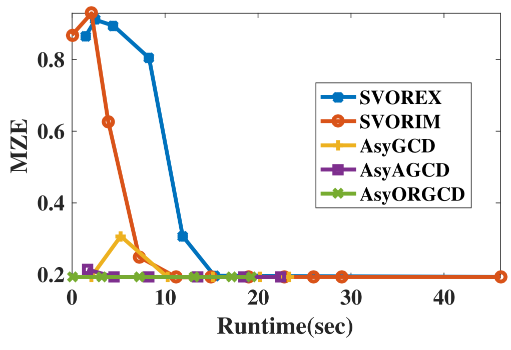
(b) CO



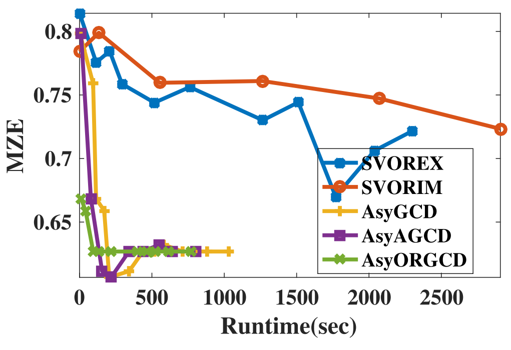
(c) MR



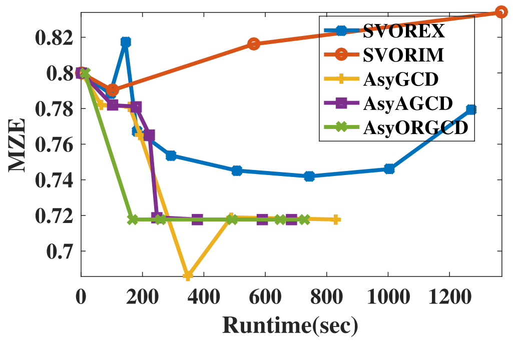
(d) M7



(e) M8

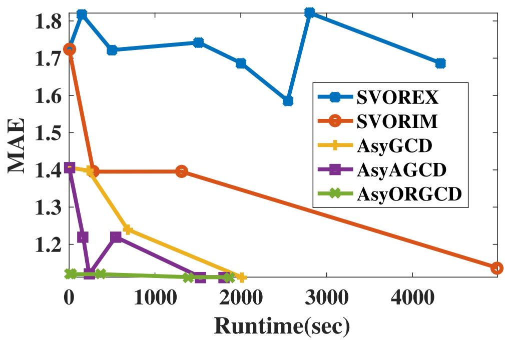


(f) H3

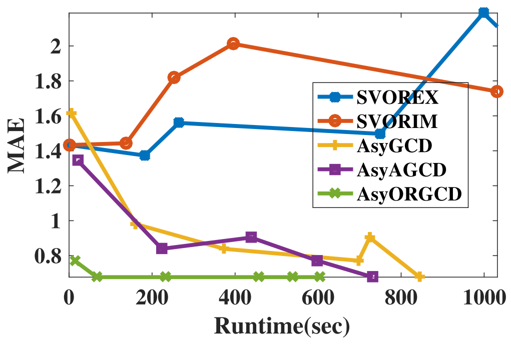


(g) N3

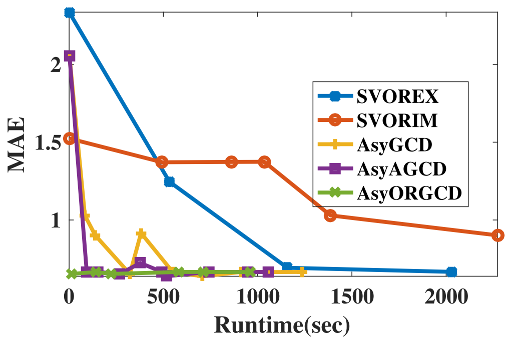
Fig. 4. Running time v.s. MZEs of different solvers for SVOR on different data sets.



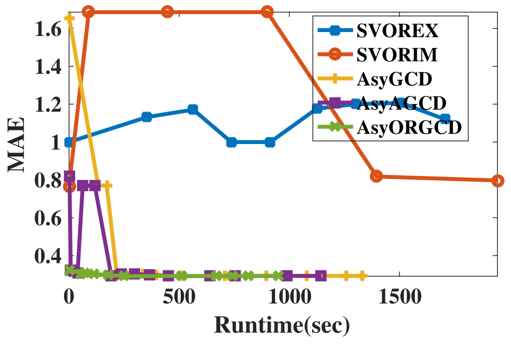
(a) 3D



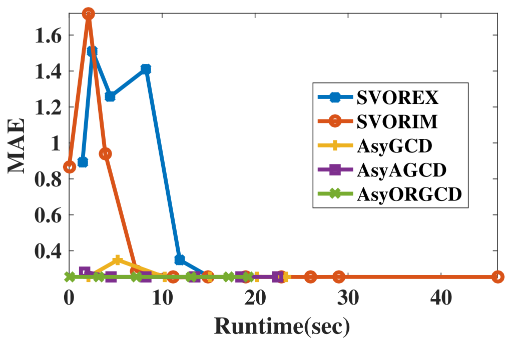
(b) CO



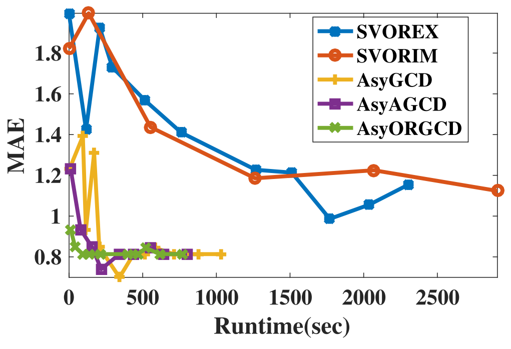
(c) MR



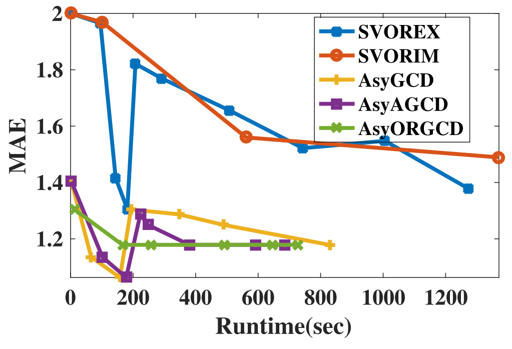
(d) M7



(e) M8



(f) H3



(g) N3

Fig. 5. Running time v.s. MAEs of different solvers for SVOR on different data sets.

Table 2
The comparison of computational complexities of AsyGCD, AsyAGCD, and AsyORGCD.

Descriptions	AsyGCD	AsyAGCD	AsyORGCD
Pick i	$O(L)$	$O(\|A\|)$	$O(L)$
Compute δ_i^*	$O(1)$	$O(1)$	$O(1)$
Update G or G_A	$O(L)$	$O(\|A\|)$	$O(L)$
Update G^t	–	–	$O(L)$
Update $\boldsymbol{\alpha}_i$	$O(1)$	$O(1)$	$O(1)$
Update $\bar{G}$	–	$O(L)$	–
Calculate M_A, etc	–	$O(\|A\|)$	–
Shrinking	–	$O(\|A\|)$	–
Update A for each block	–	$O(\|A\|)$	–
Reconstruct gradients	–	$O(L \times (L - \|A\|)$	–
Calculate $M_{\bar{S}}$ and $m_{\bar{S}}$	–	$O(L)$	–
Total	$IterTotal \times O(L)$	$(IterTotal + IterInner) \times O(\|A\|)$ $+GbarCount \times O(L)$ $+IterOuter \times O(L \times (l - \|A\|))$	$IterTotal \times O(L)$

4. Experiments and analysis

In this section, we first give the experimental setup and then present our experimental results and discussion.

4.1. Experimental setup

Design of experiments: In order to verify the convergence and effectiveness of AsyAGCD and AsyORGCD for SVOR formulation (7), we mainly compare these algorithms with the existing state of the art SVOR solvers with ordered thresholds and AsyGCD. Specifically, the compared algorithms used in our experiments include[3]:

1. ***SVORIM*** (Support Vector Ordinal Regression with Implicit Constraints): This is a SVORIM solver proposed in [11] based on sequential minimal optimization (SMO) [42] methods.
2. ***SVOREX*** (Support Vector Ordinal Regression with Explicit Constraints): This is a SVOREX solver proposed in [11] based on SMO methods.
3. ***AsyGCD***: AsyGCD algorithm for solving SVOR formulation (7).
4. ***AsyAGCD***: Our proposed AsyAGCD algorithm for SVOR learning.
5. ***AsyORGCD***: Our proposed AsyORGCD algorithm for SVOR learning.

For verifying accuracy, we compare the mean zero-one errors (MZEs) and the mean absolute errors (MAEs) among these methods via 5-fold cross-validation. We compare the MAEs, the MZEs and objective values over the running time among these methods to test the acceleration of our AsyAGCD and AsyORGCD. Note that, because of the different formulations of SVOR solved by SVORIM and SVOREX, we only compare the objective values over the running time between AsyGCD, AsyAGCD, and AsyORGCD. MAEs and MZEs are used widely in OR problems [21,43]. MZE is simply the fraction of incorrect predictions on individual samples. MAE is the average deviation of the prediction from the true target, i.e.$\frac{1}{l}\sum_1^l |h(\boldsymbol{x_i}) - y_i|$. OR problem aims to seek the correct classification of the ordinal labels which can be measured by MZEs, even if not the predicted results should as close as possible to the true labels due to the ordinal relation of each classification which can be measured by MAEs.

Implementation: We implement AsyGCD, AsyAGCD, and AsyORGCD algorithm for SVOR learning by C++ based on AsyGCD which was developed on LIBSVM [34], where the shared memory parallel computations are handled via OpenMP. We perform experiments on 36-cores two-socket Intel Xeon E5-2696 machine with 48GB RAM, where each socket has 18 cores.

Table 3
Datasets used in the experiments.

Type	Datasets	#Training/Test	#Sample.	#Attr.
D	Ethylene-(CO)	75,000/15,000	4,208,262	16
	(3D)-network	75,000/15,000	434,874	3
R	MQ2007 (M7)	50,000/10,000	69,623	46
	MQ2008 (M8)	10,000/2,000	15,211	46
	MSLR (MR)	75,000/15,000	723,412	136
	HP2003 (H3)	75,000/15,000	14,7606	64
	NP2003 (N3)	75,000/15,000	14,7606	64

We use the proposed kernel function (4) for all the experiments, where $K(\boldsymbol{x}, \boldsymbol{x}')$ is choosed as Gaussian kernel, $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\sigma||\boldsymbol{x} - \boldsymbol{x}'||^2)$. We use 5-fold cross-validation to determine the optimal values (MAEs and MZEs) of model parameters (the regularization factor C and the Gaussian kernel parameter ω) that involved in the problem formulation (7), and the test error was obtained using the optimal model parameters similar to [44]. The search was done in the region $\{(log_{10}C, log_{10}\kappa)| -3 \leq log_{10}C \leq 3, -3 \leq log_{10}\kappa \leq 3\}$ and the step size is set as one. In addition, for the sake of uniform comparison, we speed up all contrast support vector algorithm through compute kernel function parallel. All the experiments are running at least 10 times and all the results are the average values. The default thread number in the experiment is 20, some of the datasets use fewer threads but no less than 10 according to guides in [32].

Datasets: Table 3 summarizes the 7 datasets used in our experiments. They are divided into two parts according to the source of datasets, i.e., the discretized regression dataset (D)[4] and the real ordinal regression dataset (R). The size of the training and test is shown in the second column of Table 3. Due to SVORIM and SVOREX cannot handle too large scale, the max size of training we use is 75,000.[5]

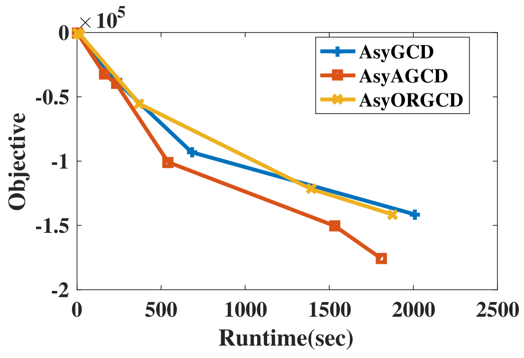
4.2. Experimental results and discussion

The boxplot Fig. 3 shows the accuracies (MAEs and MZEs) of SVORIM, SVOREX, AsyGCD, AsyAGCD, and AsyORGCD. The results were tested at the optimal solution for SVORIM and SVOREX.

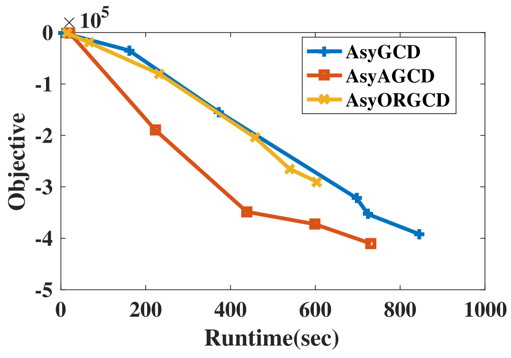
[3] The original SVORIM and SVOREX can be found in http://www.gatsby.ucl.ac.uk/~chuwei/svor.htm. We modified them for outputting the results during training and computing the kernel matrix in parallel.

[4] The targets of these datasets were discretized into 5 equal-frequency bins

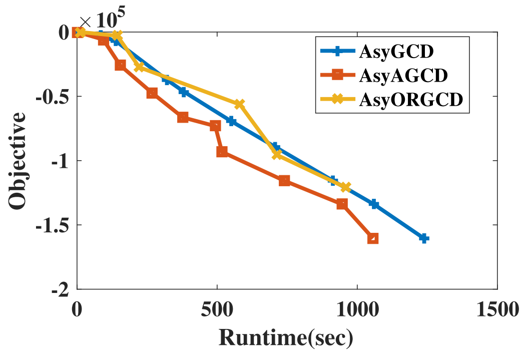
[5] MQ2007 and MQ2008 can be found in [45], HP2003 and NP2003 can be found in [46] MSLR can be found in [3]. Others can be found at http://archive.ics.uci.edu/ml/datasets.html.



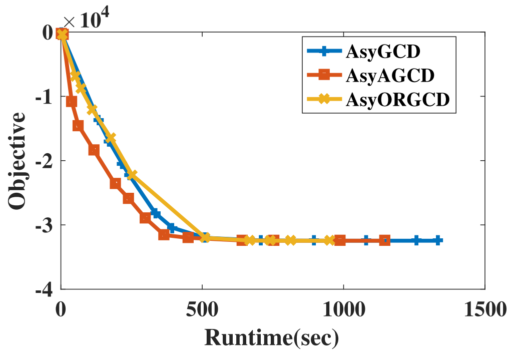
(a) 3D



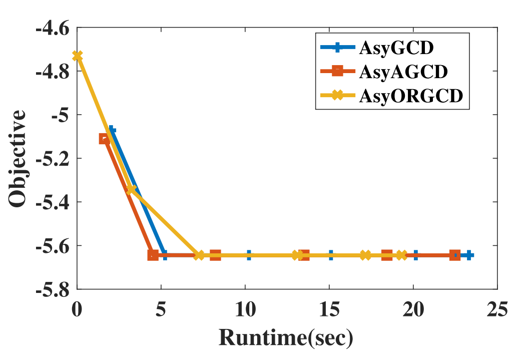
(b) CO



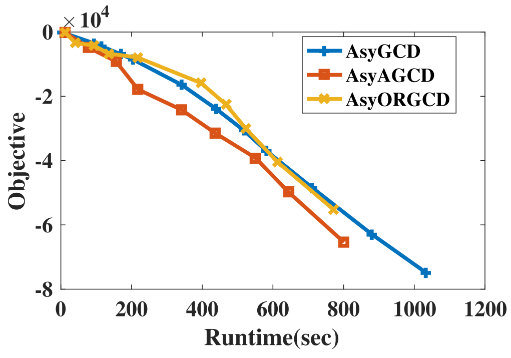
(c) MR



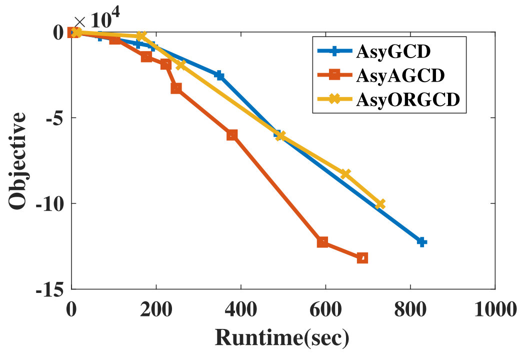
(d) M7



(e) M8



(f) H3



(g) N3

Fig. 6. Running time v.s. objective values of different solvers for SVOR on different data sets.

Observation 1. Generally speaking, the special SVOR formulation solved by AsyGCD-like algorithms retaining the similar generalization performance with SVORIM and SVOREX methods no matter on regression data sets or real-world data sets. Thus the experimental results confirm that all three asynchronous algorithms have good convergence and this SVOR formulation can learn the ordinal information as good as the-state-of-art ones.

Fig. 4 shows the MZEs values v.s. the running time of different algorithms. Note that SVOREX and SVORIM run much slower than AsyGCD-like algorithms, most of the cases take all day, we only show the early stage of training for SVOREX and SVORIM in Fig. 4 and Fig. 5, and otherwise the figure will be ambiguous to show our methods.

Observation 2. The MZEs and MAEs of AsyGCD-like algorithms actually much better than SVOREX and SVORIM, that reflects the notable speedup of asynchronous parallel.

Observation 3. The MZEs and MAEs of AsyAGCD algorithms are better than AsyGCD in most cases, that reflects the speedup of active set technique.

Observation 4. The AsyGCD and AsyAGCD algorithms have a little fluctuation on both MAEs and MZEs which arise from the disordered thresholds at training process. While the MAEs and MZEs of AsyORGCD are stable in most cases, which also confirm that AsyORGCD can hold ordered thresholds at training process.

Observation 5. From the bottom graphs in Fig. 6, it is easy to find that our AsyAGCD is faster than AsyGCD and AsyORGCD in terms of the objective values. This follows the comparison of the computational complexities of AsyGCD, AsyAGCD, and AsyORGCD as shown in Table 2. Note that, in ijcnn1 and skinnoskin dataset, it looks like that the convergence points of AsyGCD, AsyAGCD, and AsyORGCD are different, which contradicts to the theoretical result that, AsyGCD, AsyAGCD, and AsyORGCD 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, AsyAGCD, and AsyORGCD look different. This phenomenon also confirms that our AsyAGCD is faster than AsyGCD.

5. Conclusion

In this paper, we use asynchronous parallel coordinate methods to solve a concise SVOR formulation. We propose two novel asynchronous parallel coordinate descent algorithms, called AsyACGD and AsyORGCD respectively. AsyACGD is an accelerated extension of AsyGCD using active set strategy. AsyORGCD is specifically designed for SVOR that it can keep the ordered thresholds when it is training so that using early stopping rule can obtain good performance with less time. Experimental results on several large-scale ordinal regression datasets demonstrate the superiority of our asynchronous parallel coordinate methods for solving SVOR problems. Besides, our proposed AsyACGD and AsyORGCD are better than existing state-of-the-art synchronous parallel coordinate algorithm AsyGCD. In the feature, we plan to give some theoretical analysis for AsyORGCD to answer how it converges.

Declaration of Competing Interest

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

Acknowledgments

B. Gu was partially supported by the Natural Science Foundation (No. BK20161534), Six talent peaks project (No. XYDXX-042) and the 333 Project (No. BRA2017455) in Jiangsu Province and the Project Funded by the Priority Academic Program Development (PAPD) of Jiangsu Higher Education Institutions.

Appendix A. Proof of the Theorem 2.1

When a SPD kenerl is used for formula (4), we denote $\boldsymbol{K}$ as the kenerl matrix of $K(\boldsymbol{x}, \boldsymbol{x}')$, then $\boldsymbol{K}$ is a symmetric positive definite matrix according to the definition of SPD kernel with $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x}', \boldsymbol{x})$. Next, we prove that $\widetilde{K}$ has the symmetric property and positive definite property as follows.

1. **Symmetric Property:**$\widetilde{K}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}') = K(\boldsymbol{x}, \boldsymbol{x}') + J_j \cdot J_{j'} = K(\boldsymbol{x}', \boldsymbol{x}) + J_{j'} \cdot J_j = \widetilde{K}(\hat{\boldsymbol{x}}', \hat{\boldsymbol{x}})$.
2. **Positive Definite Property:** We denote a matrix $\boldsymbol{J}$, the element of c row d column is $\boldsymbol{J}_{cd} = J_c \cdot J_d$, while $J_c \cdot J_d$ can only be 0 or 1 and all diagonal elements $\boldsymbol{J}_{cc} = 1$, it easy to see that all order principal minor determinants of $\boldsymbol{J}$ is greater than 0, so $\boldsymbol{J}$ is positive definite. Because both $\boldsymbol{K}$ and $\boldsymbol{J}$ are positive definite, $\widetilde{K}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}') = \boldsymbol{K} + \boldsymbol{J}$ is positive definite.

Thus, $\widetilde{K}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}')$ is a SPD kenerl.

Appendix B. Proof of the Theorem 2.2

First, to eliminate the slack variables, we define, for $1 \le k \le r$, where b is implicit:

$$I_k^{low}(b) \overset{\text{def}}{=} \{i \in 1, \cdots, n^k : \langle \hat{\boldsymbol{w}}, \hat{\phi}(\hat{\boldsymbol{x}}_i^k) \rangle \ge -1\},$$
$$I_k^{up}(b) \overset{\text{def}}{=} \{i \in 1, \cdots, n^k : \langle \hat{\boldsymbol{w}}, \hat{\phi}(\hat{\boldsymbol{x}}_i^k) \rangle \le 1\}$$

Then we define the function as follow:

$$e_j(b) = \sum_{k=1}^{j} \sum_{I_k^{low}} (\langle \hat{\boldsymbol{w}}, \hat{\phi}(\hat{\boldsymbol{x}}_i^k) \rangle + 1) + \sum_{k=j+1}^{r} \sum_{I_k^{up}} (\langle -\hat{\boldsymbol{w}}, \hat{\phi}(\hat{\boldsymbol{x}}_i^k) \rangle + 1)$$

As described above, when the proposed kernel (4) is used, we have

$$e_j(b) = \sum_{k=1}^{j} \sum_{I_k^{low}} (\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i^k) \rangle - b + 1) + \sum_{k=j+1}^{r} \sum_{I_k^{up}} (\langle -\boldsymbol{w}, \phi(\boldsymbol{x}_i^k) \rangle + b + 1)$$

So, the derivative of $e_j(b)$ with respect to b is

$$g_j(b) = -\sum_{k=1}^{j} \left| I_k^{low}(b) \right| + \sum_{k=j+1}^{r} \left| I_k^{up}(b) \right|$$

It is easy to see that $\boldsymbol{b}_j$ is optimal iff it minimizes the function $e_j(b)$. Take any one j with $1 \le j < r$ for consideration, and suppose $b_j^* > b_{j+1}^*$. Since b_{j+1}^* is strictly to the left of the bias b_j^* that minimizes $e_j(b)$, we have $g_j(b_{j+1}^*) < 0$. Since b_{j+1}^* is a minimizer of $e_{j+1}(b)$, we also have $g_{j+1}(b_{j+1}^*) \ge 0$. Thus we have $g_{j+1}(b_{j+1}^*) - g_j(b_{j+1}^*) > 0$, but by the formulation of $g_j(b)$, we get

$$g_{j+1}(b_{j+1}^*) - g_j(b_{j+1}^*) = -\left| I_{j+1}^{low}(b_{j+1}^*) \right| - \left| I_{j+1}^{up}(b_{j+1}^*) \right| \le 0$$

which is impossible, so $b_j^* \le b_{j+1}^*$, and similarly we can get $b_1^* \le b_2^* \le \cdots \le b_{r-1}^*$. This completes the proof.

Appendix C. Proof of the Theorem 3.1

For the sake of simplicity, we consider the primal-dual problem of C-SVC since our SVOR formulation can be seen as a variant C-SVC via change the index of data set. We have $\sum_{i=1}^{l} y_i \boldsymbol{\alpha}_i^* \phi(\boldsymbol{x}_i) = \boldsymbol{w}^*$, where $\boldsymbol{w}^*$ is the primal optimal solution and $\phi(\boldsymbol{x})$ is a mapping function to reproducing kernel Hilbert space (RKHS). Here $\boldsymbol{w}^*$ is unique due to the strong convexity of primal problem and $\{\boldsymbol{\alpha}^k\}$ is in a compact set, and we have

$$\lim_{k\to\infty}\sum_{i=1}^{l} y_i\alpha_i^k\phi(\boldsymbol{x}_i) = \lim_{k\to\infty}\boldsymbol{w}^k = \boldsymbol{w}^* \tag{C.1}$$

$$\begin{aligned}\lim_{k\to\infty}\nabla_i f(\boldsymbol{\alpha}^k) &= \lim_{k\to\infty}\sum_{j=1}^{l} y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)\alpha_j^k - 1\\ &= \lim_{k\to\infty} y_i\langle\boldsymbol{w}^k,\phi(\boldsymbol{x}_i)\rangle - 1\\ &= y_i\langle\boldsymbol{w}^*,\phi(\boldsymbol{x}_i)\rangle - 1\end{aligned} \tag{C.2}$$

We define $d_i^* \equiv \nabla_i f(\boldsymbol{\alpha}^*) = y_i\langle\boldsymbol{w}^*,\phi(\boldsymbol{x}_i)\rangle - 1$. The subsequent proofs are similar to Lemma 1 proof for linear SVM in [33]. To make the paper self-contained, we present the proofs as follows.

According to (C.2), there exists an index k_i and for all $k \geq k_i$,

$$\begin{aligned}&\nabla_i f(\boldsymbol{\alpha}^k) > 0 \; if \; d_i^* > 0,\\ &\nabla_i f(\boldsymbol{\alpha}^k) < 0 \; if \; d_i^* < 0.\end{aligned} \tag{C.3}$$

During updating $\boldsymbol{\alpha}^k$ to $\boldsymbol{\alpha}^{k+1}$ by changing the i-th element, we derive the relations (C.4) by the KKT conditions.

$$\begin{aligned}&\nabla_i f(\boldsymbol{\alpha}^{k+1}) > 0 \;\Rightarrow \alpha_i^{k+1} = 0,\\ &\nabla_i f(\boldsymbol{\alpha}^{k+1}) < 0 \;\Rightarrow \alpha_i^{k+1} = C.\end{aligned} \tag{C.4}$$

Hence either α_i is upper or lower bounded forever or $0 < \alpha_i < C$ and the i-th element is never selected for update. The second situation means that there is a limit point $\bar{\boldsymbol{\alpha}}$ with $0 < \bar{\alpha}_i < C$ and $\nabla_i f(\bar{\boldsymbol{\alpha}}) \neq 0$, which contradicts to the assumption that every limit point is optimal. Thus for all $k > k_i$, we have that

$$\begin{aligned}&\alpha_i^k = 0, \quad if \; d_i^* > 0,\\ &\alpha_i^k = C, \quad if \; d_i^* < 0.\end{aligned} \tag{C.5}$$

which are the first and the second conclusions of Theorem 3.1.

This result and (C.3) imply for $k > k_i$,

$$\nabla_i^P f(\boldsymbol{\alpha}^k) = 0, \quad if \; d_i^* > 0 \; or \; d_i^* < 0. \tag{C.6}$$

For any index i with $d_i^* = 0$, (C.2) means that $\lim_{k\to\infty}\nabla_i^P f(\boldsymbol{\alpha}^k) = 0$. Thus, we have that $\lim_{k\to\infty}\nabla^P f(\boldsymbol{\alpha}^k) = \mathbf{0}$ for $k > k_i$.

References

[1] M. Pérez-Ortiz, M. Cruz-Ramírez, M.D. Ayllón-Terán, N. Heaton, R. Ciria, C. Hervás-Martínez, An organ allocation system for liver transplantation based on ordinal regression, Appl. Soft Comput. 14 (2014) 88–98.
[2] K.-j. Kim, H. Ahn, A corporate credit rating model using multi-class support vector machines with an ordinal pairwise partitioning approach, Comput. Oper. Res. 39 (8) (2012) 1800–1811.
[3] T. Qin, T. Liu, Introducing LETOR 4.0 datasets, CoRR abs/1306.2597 (2013).
[4] W.L. Chao, J.Z. Liu, J.J. Ding, Facial age estimation based on label-sensitive learning and age-oriented regression, Pattern Recognit. 46 (3) (2013) 628–641.
[5] K.W. Lau, Q.H. Wu, Local prediction of non-liner time series using support vector regression, Pattern Recognit. 41 (5) (2008) 1539–1547.
[6] Z. Zhai, B. Gu, X. Li, H. Huang, Safe sample screening for robust support vector machine, arXiv:1912.11217 (2019).
[7] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[8] Chen, Dandan, Tian, Yingjie, Liu, Xiaohui, Structural nonparallel support vector machine for pattern recognition, Pattern Recognit. (2016).
[9] X. Geng, B. Gu, X. Li, W. Shi, G. Zheng, H. Huang, Scalable semi-supervised svm via triply stochastic gradients, arXiv:1907.11584 (2019).
[10] R. Herbrich, T. Graepel, K. Obermayer, Large margin rank boundaries for ordinal regression (2000).
[11] W. Chu, S.S. Keerthi, Support vector ordinal regression, Neural Comput. 19 (3) (2007) 792–815.
[12] W. Chu, Z. Ghahramani, Gaussian processes for ordinal regression, J. Mach. Learn. Res. 6 (Jul) (2005) 1019–1041.
[13] B.-Y. Sun, J. Li, D.D. Wu, X.-M. Zhang, W.-B. Li, Kernel discriminant learning for ordinal regression, IEEE Trans. Knowl. Data Eng. 22 (6) (2010) 906–910.
[14] R. Herbrich, T. Graepel, K. Obermayer, Support vector learning for ordinal regression (1999).
[15] A. Shashua, A. Levin, Ranking with large margin principle: two approaches, in: Advances in Neural Information Processing Systems, 2003, pp. 961–968.
[16] P.A. Gutiérrez, M. Pérez-Ortiz, J. Sanchez-Monedero, F. Fernández-Navarro, C. Hervas-Martinez, Ordinal regression methods: survey and experimental study, IEEE Trans. Knowl. Data Eng. 28 (1) (2016) 127–146.
[17] B. Gu, V.S. Sheng, K.Y. Tay, W. Romano, S. Li, Incremental support vector learning for ordinal regression., IEEE Trans. Neural Netw. Learn.Syst. 26 (7) (2017) 1403–1416.
[18] J.A. Suykens, J. Vandewalle, B. De Moor, Optimal control by least squares support vector machines, Neural Netw. 14 (1) (2001) 23–35.
[19] X. Peng, Tsvr: an efficient twin support vector machine for regression, Neural Netw. 23 (3) (2010) 365–372.
[20] X. Peng, Tpmsvm: a novel twin parametric-margin support vector machine for pattern recognition, Pattern Recognit. 44 (10–11) (2011) 2678–2692.
[21] B. Gu, A regularization path algorithm for support vector ordinal regression, Neural Netw. 98 (2018) 114–121.
[22] V. Vapnik, Statistical Learning Theory. 1998, 3, Wiley, New York, 1998.
[23] W. Shi, B. Gu, X. Li, X. Geng, H. Huang, Quadruply stochastic gradients for large scale nonlinear semi-supervised auc optimization, arXiv:1907.12416 (2019).
[24] X. Li, B. Gu, S. Ao, H. Wang, C.X. Ling, Triply stochastic gradients on multiple kernel learning., UAI, 2017.
[25] C.K.I. Williams, M. Seeger, Using the nystroŋm method to speed up kernel machines, in: International Conference on Neural Information Processing Systems, 2000, pp. 661–667.
[26] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: Advances in Neural Iinformation Processing Systems, 2008, pp. 1177–1184.
[27] P. Drineas, M.W. Mahoney, On the nyström method for approximating a gram matrix for improved kernel-based learning, . Mach. Learn. Res. 6 (Dec) (2005) 2153–2175.
[28] D. Lopez-Paz, S. Sra, A. Smola, Z. Ghahramani, B. Schŋlkopf, Randomized nonlinear component analysis, Comput. Sci. 4 (2014) 1359–1367.
[29] B. Gu, M. Xin, Z. Huo, H. Huang, Asynchronous doubly stochastic sparse kernel learning, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[30] J. Liu, S.J. Wright, C. Ré, V. Bittorf, S. Sridhar, An asynchronous parallel stochastic coordinate descent algorithm, J. Mach. Learn. Res. 16 (1) (2015) 285–322.
[31] J.C. Duchi, S. Chaturapruek, C. RȨ, Asynchronous stochastic convex optimization, Mathematics 18 (1) (2015) 51–52.
[32] Y. You, X. Lian, J. Liu, H.-F. Yu, I.S. Dhillon, J. Demmel, C.-J. Hsieh, Asynchronous parallel greedy coordinate descent, in: Advances In Neural Information Processing Systems, 2016, pp. 4682–4690.
[33] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S.S. Keerthi, S. Sundararajan, A dual coordinate descent method for large-scale linear svm, in: Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 408–415.
[34] C.-C. Chang, C.-J. Lin, Libsvm: a library for support vector machines, ACM Trans. Intell. Syst.Technol. (TIST) 2 (3) (2011) 27.
[35] B. Gu, Y. Shan, X. Geng, G. Zheng, Accelerated asynchronous greedy coordinate descent algorithm for svms., in: IJCAI, 2018, pp. 2170–2176.
[36] H.-T. Lin, L. Li, Reduction from cost-sensitive ordinal ranking to weighted binary classification, Neural Comput. 24 (5) (2012) 1329–1367.
[37] L. Li, H.-T. Lin, Ordinal regression by extended binary classification, in: Advances in Neural Information Processing Systems, 2007, pp. 865–872.
[38] M. Filippone, F. Camastra, F. Masulli, S. Rovetta, A survey of kernel and spectral methods for clustering, Pattern Recognit. 41 (1) (2008) 176–190.
[39] W. Jie, H. Lu, K.N. Plataniotis, J. Lu, Gaussian kernel optimization for pattern classification, Pattern Recognit. 42 (7) (2009) 1237–1247.
[40] S.-i. Amari, S. Wu, Improving support vector machine classifiers by modifying kernel functions, Neural Netw. 12 (6) (1999) 783–789.
[41] D.P. Bertsekas, Nonlinear Programming, Athena Scientific Belmont, 1999.
[42] C.P. John, Sequential minimal optimization: a fast algorithm for training support vector machines, MSRTR 3 (1) (1998) 88–95.
[43] C. Li, M. de Rijke, Incremental sparse bayesian ordinal regression, Neural Netw. 106 (2018) 294–302.
[44] B. Gu, X. Geng, X. Li, G. Zheng, Efficient inexact proximal gradient algorithms for structured sparsity-inducing norm, Neural Netw. 118 (2019) 352–362.
[45] T. Qin, T.Y. Liu, J. Xu, H. Li, Letor: a benchmark collection for research on learning to rank for information retrieval, Inf. Retr. Boston 13 (4) (2010) 346–374.
[46] T. Qin, T.Y. Liu, Introducing letor 4.0 datasets, Comput. Sci. (2013).

Bin Gu received the B.S. and Ph.D. degrees in computer science from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2005 and 2011, respectively. He joined the School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, in 2010, as a Lecturer, where he iscurrently a Full Professor. His current research interests include machine learning, data mining, and medical image analysis.