Distributed and Robust Support Vector Machine*

Yangwei Liu[1], Hu Ding[2], Ziyun Huang[3], and Jinhui Xu[4]

1 **Department of Computer Science and Engineering, State University of New York at Buffalo, US**
2 **Department of Computer Science and Engineering, Michigan State University, US**
3 **Department of Computer Science and Engineering, State University of New York at Buffalo, US**
4 **Department of Computer Science and Engineering, State University of New York at Buffalo, US**

Abstract

In this paper, we consider the distributed version of Support Vector Machine (SVM) under the coordinator model, where all input data (*i.e.,* points in $\mathbb{R}^d$ space) of SVM are arbitrarily distributed among k nodes in some network with a coordinator which can communicate with all nodes. We investigate two variants of this problem, with and without outliers. For distributed SVM without outliers, we prove a lower bound on the communication complexity and give a distributed $(1-\epsilon)$-approximation algorithm to reach this lower bound, where ϵ is a user specified small constant. For distributed SVM with outliers, we present a $(1-\epsilon)$-approximation algorithm to explicitly remove the influence of outliers. Our algorithm is based on a deterministic distributed top t selection algorithm with communication complexity of $O(k \log(t))$ in the coordinator model. Experimental results on benchmark datasets confirm the theoretical guarantees of our algorithms.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Distributed Algorithm, Communication Complexity, Robust Algorithm, SVM

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.54

1 Introduction

Training a *Support Vector Machine (SVM)* [5] in a distributed setting is a commonly encountered problem in the big data era. This could be due to the fact that the data set is too large to be stored in a centralized site (*e.g.,* bioinformatics data [15]), or simply because the data set is collected in a distributed environment (*e.g.,* wireless sensor network data [22]). In many scenarios, large volume data transmission between different sites could be rather expensive and time consuming. In some applications, this may not even be allowed due to privacy concerns. Thus efficient distributed SVM algorithms are needed to minimize the communication cost and meanwhile preserve the quality of solution.

In recent years, a significant amount of effort has been devoted to this problem ([9, 3, 12, 14, 4, 16, 8, 18, 6]), and a number of distributed SVM algorithms with different strength have been developed. However most of them are still suffering from various limitations, such as high communication complexity, sub-optimal quality of solution, and slow convergence

* The research of this work was supported in part by NSF through grants CCF-1422324, IIS-1422591, and CNS-1547167. Ding was supported by a start-up fund from Michigan State University.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 54; pp. 54:1–54:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

rate. For instance, one type of extensively studied algorithms in recent years are the family of *incremental construction* algorithms [9, 3].

Such algorithms often have good performance in practice and some other nice features related to robustness and decentralization; but they generally do not have theoretical guarantee on the communication complexity, and some of them even have no quality guarantee on their solutions. Another type of popular algorithms are those which parallelize existing centralized algorithms ([12, 14, 4]). These algorithms typically focus on enhancing the ability of dealing with extremely large size data sets, but in general have no quality guarantee on communication complexity. There are also another family of algorithms called distributed stochastic gradient descent algorithms [20]; the main issue of such algorithms is that their running time (or number of iteration) is mostly sub-optimal, and they do not have a guarantee on communication cost. Very recently, there is an interesting result [2] which presents a similar lower bound on communication cost. However their lower bound applies only to those coreset-based algorithms, not the general algorithms, whereas the lower bound result in this paper is applicable to any distributed SVM algorithm.

From a geometric point of view, training an SVM can be interpreted as finding a hyperplane that separates two classes of points while maximizing the separating margin. It is also well known to be equivalent to computing the polytope distance of the two point sets. Recent research [10] shows that we can find an $(1-\epsilon)$-approximation of the polytope distance using Gilbert algorithm with a running time linearly depending on the input size. Roughly speaking, Gilbert algorithm is a gradient descent procedure that in each step greedily computes an optimal direction along which the primal solution should improve.

In this paper we present a distributed SVM algorithm that is theoretically guaranteed to have the lowest possible communication cost together with a guaranteed near-optimal solution, based on the classical Gilbert algorithm [11]. Comparing to previous distributed SVM algorithms, our algorithm has several advantages. (1) Our algorithm has a communication complexity which is theoretically guaranteed to reach the lower bound; (2) it does not make any assumption on the input data and its distribution; (3) its running time is only linearly dependent on the input size; and (4) it produces a $(1-\epsilon)$-approximation for the problem which is sparse (*i.e.,* the number of support vectors is small).

Since SVM is well known to be sensitive to outliers, we also consider the case of distributed SVM with outliers. We show that it is possible to explicitly avoid the influence of outliers in distributed settings by using a combinatorial tool called *Random Gradient Descent (RGD)* tree [7] to achieve a $(1-\epsilon)$-approximation on the quality of solution and meanwhile significantly reduce the communication cost. An underlying technique used for reducing the communication cost is an algorithm for the distributed selection problem (*i.e.,* finding the t-th smallest number from a set of numbers distributed in k sites). This problem has been extensively studied in the past ([17, 21, 13]). The best result (in terms of communication cost) is a randomized algorithm [13] which has a communication complexity of $O(k \log(t))$. The best deterministic algorithm has a communication complexity of $O(k \log^2(t))$. In this paper we give a deterministic algorithm with communication complexity $O(k \log(t))$ in the coordinator model.

In this paper We will concentrate on one-class SVM first since it captures the main difficulty of the problem, and then extend the results to two-class SVM (in the full version of this paper). We also performed experiments on benchmark datasets to evaluate the performance of the algorithms (in the full version of the paper).

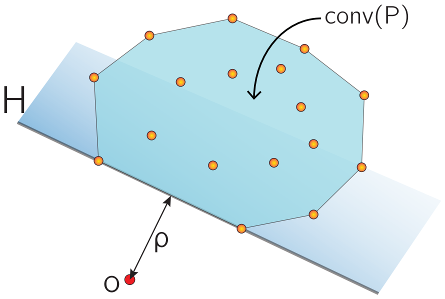

Figure 1 One-class SVM is equivalent to the polytope distance problem. ρ is the polytope distance, H is the separating hyperplane, with distance to o exactly ρ.

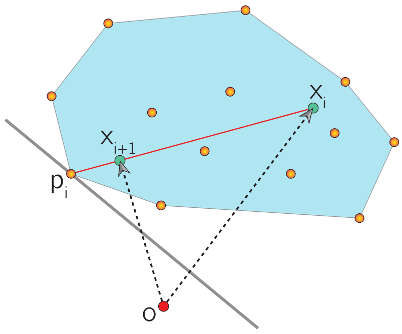

Figure 2 One step of the Gilbert Algorithm, updating x_i to x_{i+1}.

2 Preliminaries

2.1 Equivalence between SVM and Polytope Distance

In this section, we give several definitions which will be used throughout the paper.

▶ **Definition 1.** (One-class SVM): Given a point set $P \subseteq \mathbb{R}^d$, find a hyperplane H separating the origin o and P such that the separating margin (*i.e.*, the distance between o and H) is maximized.

It is well known that this problem is equivalent to the following problem of computing the polytope distance:

▶ **Definition 2.** (Polytope distance): Given a point set $P \subseteq \mathbb{R}^d$, compute the shortest distance between the origin o and a point p in the polytope $conv(P)$ (*i.e.*, the convex hull of P).

▶ **Lemma 3** ([10])**.** *Given a point x which realizes the polytope distance of $P \subseteq \mathbb{R}^d$, the hyperplane H passing x and orthogonal to $\overline{ox}$ is the maximum separating hyperplane of P. In other words, the polytope distance problem is equivalent to the one-class SVM problem.*

Figure 1 provides an intuitive explanation of the equivalence between one-class SVM and polytope distance. Notice that the polytope distance problem can be formulated as a standard convex quadratic optimization problem, thus can be solved optimally using standard techniques in $O(n^3)$ time. However this approach is mostly impractical because of the high time complexity. Actually in many applications, a near-optimal approximate solution is sufficient. Thus it is desirable to design efficient approximation algorithms. A $(1-\epsilon)$-*approximation* of the polytope distance can be defined as follows.

▶ **Definition 4.** ($(1-\epsilon)$-approximation of polytope distance): $x \in conv(P)$ achieves a $(1-\epsilon)$-approximation of the polytope distance problem, if

$$||x|| - p|_x \leq \epsilon ||x||, \forall p \in P$$

where $p|_x := \frac{\langle p,x \rangle}{||x||}$ is the signed length of the projection of p onto vector $\overline{ox}$.

In the rest of this paper, we also call the point x, rather its distance to o, as the approximation. This is only for ease of discussion, and does not affect the solution at all (since we can get the actual distance by simply computing the distance between x and o).

2.2 Gilbert Algorithm

Gilbert algorithm can be used to find a $(1-\epsilon)$-approximation of the polytope distance problem. Roughly speaking, Gilbert algorithm starts with an initial solution x_1 being the closest point of P to the origin. In step i, the algorithm finds the point $p_i \in P$ that has the smallest projection distance to vector $\overline{ox_i}$, and picks the point on the line segment $[p_i, x_i]$ that is closest to the origin as x_{i+1}. Figure 2 illustrates one step of Gilbert algorithm.

Algorithm 1 Gilbert Algorithm

```
1: INPUT : A $d$ dimensional point set $P$, the origin $o$.
2: OUTPUT : A $(1-\epsilon)$-approximation of the polytope distance of $P$ to $o$.
3: Initialize $i$ to be 1. Find the point $p_1$ that is closest to $o$, let $x_1 = p_1$
4: In step $i$, let $p_{i+1}$ be the point that has the smallest $p|_{x_i}$, let $x_{i+1}$ be the point that is
   closest to $o$ on line segment $[x_i, p_{i+1}]$.
5: Return $x_{i+1}$ when it is a $(1-\epsilon)$-approximation. Otherwise goto Line 4 with $i = i+1$.
```

Despite its simplicity, it has been shown [10] that Gilbert algorithm can actually find such a $(1-\epsilon)$-approximation in $O(\frac{1}{\epsilon})$ steps. Formally, we have the following theorem.

▶ **Theorem 5** ([10]). *Gilbert algorithm succeeds after at most $O(\frac{1}{\epsilon})$ steps.*

Since each step of the algorithm involves only computing the projection of all points in P to a specific direction, which can be done in linear time, Gilbert algorithm has a total running time linearly depending on $n = |P|$ which is the number of input points.

Besides the fast running time, Gilbert Algorithm (and its variants) has also many other properties that make it a good SVM solver.

1. The algorithm works for arbitrary kernels. Since the algorithm only requires the computation of projection distance, which can be obtained by scalar product, we only need one kernel evaluation at each time when a projection distance is computed. This also implies that our proposed algorithms can be trivially kernelized.
2. The result is sparse, or in other words, the number of support vectors is small. In the kernelized version, the solution x_i is a linear combination of input points, and we can easily observe that x_i involves at most one more point in each step, so the total number of support vectors is $O(\frac{1}{\epsilon})$.

3 Communication Complexity of Distributed SVM

In a distributed setting, the point set P is arbitrarily distributed among k nodes. Based on different modes of communication, there are different models to be considered. In this paper we mainly study the widely used coordinator model which contains an extra coordinator node, and all other nodes are only allowed to communicate with the coordinator. The algorithm itself can be easily modified for a more general communication model, *e.g.*, network of general topology. Such generalization will at most induce an additional $O(D)$ (where D is the diameter of the network) [1] increase on the communication complexity.

We are interested in determining the minimum amount of communication that is needed to find a $(1-\epsilon)$-approximation of the polytope distance problem. We define the communication cost as the number of points needed to be transfered between nodes, where transferring one d-dimensional point (together with an optional $O(d)$ constants) takes $O(1)$ communication. This way of defining the communication cost simplifies the proof of Theorem 6. Below is our lower bound result.

▶ **Theorem 6.** *A $(1-\epsilon)$-approximation of the distributed polytope distance problem requires $\Omega(kd)$ communication for any $\epsilon < \frac{\sqrt{17}-4}{16d}$.*

We prove Theorem 6 by giving a reduction from the following *k-OR* problem.

▶ **Definition 7** (k-OR)**.** Given k players with each holding an n-bit binary vector, find the bitwise OR of all k vectors.

An example of the k-OR problem can be like the following: Player 1 holds vector $(1,0,0,1,0)$, Player 2 holds $(0,0,0,1,1)$, and Player 3 holds $(1,1,0,0,1)$. Then the output should be the vector $(1,1,0,1,1)$, where each bit is just the OR of the same bit of all players' vectors. For the communication complexity of this problem we have the following result.

▶ **Lemma 8** ([19])**.** *k-OR problem requires $\Omega(nk)$ communication in the coordinator model.*

Proof of Theorem 6. We prove Theorem 6 by reducing the k-OR problem to the problem of finding an ϵ-approximation of distributed polytope distance.

Given an instance of the k-OR problem with k players each holding an n-bit binary vector, construct an instance of distributed polytope distance in $d=n$ dimensional space with k nodes as follows:

For player i, for $j=1,\cdots,d$, if the j'th bit of his vector is 0, add point e_j to node i's point set; otherwise add point λe_j to its point set, where $e_j = (\underbrace{0,\cdots,0}_{\text{j-1}},1,\underbrace{0,\cdots,0}_{\text{d-j}})$ is the d-dimensional point whose only non-zero entry is the j'th coordinate with value 1, and λ is a constant smaller than 1 to be determined later.

In this construction, each node holds exactly d points, and there are kd points in total. Since for the j'th point there are only two possible positions to place it, *i.e.*, e_j or λe_j, there will be points from different nodes sharing the same position. This does not affect the proof, since we can add a small enough perturbation to points sharing the same location.

▶ **Definition 9** (Configuration)**.** A configuration C of an instance of the polytope distance problem constructed as above is a size d point set, where the i'th point is λe_i if there is at least one λe_i in the point sets of all nodes; otherwise the i'th point is set to be e_i. The **order** of a configuration C is the number of λe it contains.

It is easy to see that a configuration encodes the solution of the corresponding k-OR problem. So if we can find out the configuration based on an ϵ-approximation solution of the distributed polytope distance problem, we can solve the k-OR problem, thus proving Theorem 6. The following lemma ensures that we can indeed achieve that.

▶ **Lemma 10.** *If $\epsilon < \frac{\sqrt{17}-4}{16d}$, it is possible to determine the configuration purely based on an ϵ-approximation of the distributed polytope distance problem with λ satisfying $\frac{1}{2}-\sqrt{\frac{1}{4}-\frac{\epsilon d}{1-\epsilon}} < \lambda < \min\{\sqrt{1-d(1-(1-\epsilon)^2)}, (\frac{1}{2}+\sqrt{\frac{1}{4}-\frac{\epsilon d}{1-\epsilon}})\}$.*

Notice that $\epsilon < \frac{\sqrt{17}-4}{16d}$ guarantees that $\frac{1}{4}-\frac{\epsilon d}{1-\epsilon} > 0$, $1-d(1-(1-\epsilon)^2) > 0$, as well as $\frac{1}{2}-\sqrt{\frac{1}{4}-\frac{\epsilon d}{1-\epsilon}} < \sqrt{1-d(1-(1-\epsilon)^2)}$. Thus such a λ always exists. Denote the polytope distance of a configuration C as $\rho(C)$, the order of C as *order*(C). We call the set of configurations with the same order d' as an ***order-d' layer*** of configurations. We prove Lemma 10 in two steps:

- **Claim 1**: *order*(C_i) = *order*$(C_j) \implies \rho(C_i)=\rho(C_j)$; *order*$(C_i) >$ *order*$(C_j) \implies \rho(C_i) < (1-\epsilon)\rho(C_j)$.

- **Claim 2**: a solution x can't be an ϵ-approximation for more than one configuration in the same layer.

Proof of Claim 1. Consider a configuration $C = \{p_1, p_2, \cdots, p_d\}$ with order d'. This means that there are $d - d'$ points with their only non-zero coordinate as 1, and d' points with their non-zero coordinate as λ. Suppose that the point $x = \alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_d p_d$ on $conv(C)$ is the closest point to the origin. We observe that for $i \in [1, d]$, we have $0 < \alpha_i < 1$. This can be proved by contradiction as follows. First of all, since x is on $conv(C)$, we naturally have $0 \le \alpha_i \le 1$ and $\sum_i \alpha_i = 1$. Suppose that there exists an $\alpha_l = 0$. This means that the l'th coordinate of x is 0, and thus x is on the simplex spanned by $C - \{p_l\}$. Notice that in this case, we have $\overline{op_l}$ perpendicular to the simplex spanned by $C - \{p_l\}$. Thus, $\angle oxp_l < \frac{\pi}{2}$ and $\angle op_l x < \frac{\pi}{2}$, which means that the projection of o onto $\overline{xp_l}$ is within the line segment $\overline{xp_l}$, resulting in a point closer to o than x, contradicting the fact that x is the closest point to o. Then $\alpha < 1$ follows immediately.

The above observation guarantees that the closest point to o is always within $conv(C)$, instead of on the boundary. Now let us compute $\rho(C)$.

We partition the points of C into two subsets based on their non-zero coordinates. C_1 contains points whose non-zero coordinates are 1, and C_λ contains the rest. By the symmetry of the dimensions, we can safely assume that C_1 contains the first $d - d'$ points, and C_λ contains the latter d' points without loss of generality. Now let $a = \frac{1}{(d-d')\lambda + d'\frac{1}{\lambda}}$, and consider the point $x = (\underbrace{\lambda a \cdot 1, \cdots, \lambda a \cdot 1}_{d-d'}, \underbrace{\frac{a}{\lambda} \cdot \lambda, \cdots, \frac{a}{\lambda} \cdot \lambda}_{d'})$. It is clear that (1) x is within the boundary of $conv(C)$, since $(d - d')\lambda a + d'\frac{a}{\lambda} = 1$; and (2) the projection distance of any point in C onto $\overline{ox}$ is $\frac{\lambda a}{\|x\|}$. Thus $\overline{ox}$ is perpendicular to the subspace spanned by C. Combining the above two facts, we know that x is the closest point to o on $conv(C)$, and $\rho(C) = \|\overline{ox}\| = \sqrt{\frac{1}{(d-d') + \frac{d'}{\lambda^2}}}$. Since $\rho(C)$ depends only on the order of the configuration, we have proved the first half of Claim 1.

Since each layer of configuration has the same polytope distance, we let $\rho_{d'}$ denote the polytope distance for order-d' layer. Now for the second half of Claim 1, let us consider two consecutive layers with order d' and $d' + 1$. Then we have

$$\begin{aligned}
\frac{\rho_{d'+1}}{\rho_{d'}} &= \sqrt{\frac{d + (\frac{1}{\lambda^2} - 1)d'}{d + (\frac{1}{\lambda^2} - 1)(d' + 1)}} \\
&\le \sqrt{\frac{d - (1 - \lambda^2)}{d}} && (1) \\
&< \sqrt{\frac{d - (1 - \sqrt{1 - d(1 - (1 - \epsilon)^2)}^2)}{d}} \\
&= 1 - \epsilon && (2)
\end{aligned}$$

in which inequality (1) holds because of $\frac{1}{\lambda^2} - 1 > 0$. Using (2), we immediately have $\frac{\rho_{d'+t}}{\rho_{d'}} < (1 - \epsilon)^t < 1 - \epsilon$. Thus the second part of Claim 1 is proved. ◀

Proof of Claim 2. In order to prove Claim 2, we first make another observation of the ϵ-approximation x of configuration $C = \{p_1, \cdots, p_d\}$ of order d'. Since x is an ϵ-approximation, by definition it has to be on $conv(C)$. So x takes the form of $x = \sum \alpha_i p_i$, and $\sum \alpha_i = 1$. Also by definition, we have $p_i|_x > (1 - \epsilon)\|x\|$. Together with the definition of $p_i|_x$, we have

$$\alpha_i \ge \begin{cases} (1 - \epsilon)\|x\|^2 & \text{, if } p_i = e_i \\ \frac{1-\epsilon}{\lambda^2}\|x\|^2 & \text{, otherwise.} \end{cases}$$

Then we have for $p_i = e_i$,

$$\begin{aligned}
\alpha_i &= 1 - \sum_{j \neq i} \alpha_j \\
&\leq 1 - (\sum_{j \neq i, p_j = e_j} (1-\epsilon)\|x\|^2 + \sum_{j \neq i, p_j = \lambda e_j} \frac{(1-\epsilon)}{\lambda^2}\|x\|^2) \\
&= 1 - ((d - d' - 1)(1-\epsilon)\|x\|^2 + d'\frac{1-\epsilon}{\lambda^2}\|x\|^2) \\
&= (\frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}))\|x\|^2. \qquad (3)
\end{aligned}$$

Now, suppose that x is an ϵ-approximation for two configurations C_1 and C_2 in the same layer. Since all configurations in the same layer have the same order, this indicates that they have the same number of points whose non-zero coordinate is 1 and the same number of points whose non-zero coordinate is λ. This means that C_1 and C_2 differs by at least two points, with different non-zero coordinate. Denote the index of such pair of points as i and j. W.O.L.G., assume that in C_1 the i'th point $p_i^{(1)} = e_i$, and the j'th point $p_j^{(1)} = \lambda e_j$. Then in C_2, the i'th point $p_i^{(2)} = \lambda e_i$, and the j'th point $p_j^{(2)} = e_j$. Since x is an ϵ-approximation of C_1, it has to take the form of $x = \sum \alpha_i p_i^{(1)}$,, and $p_i^{(1)} = e_i$, following (3) we have

$$\alpha_i \leq (\frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}))\|x\|^2.$$

Since x is also an ϵ-approximation of C_2, it has to satisfy

$$\frac{\lambda \alpha_i}{\|x\|} = p_i^{(2)}|_x \geq (1-\epsilon)\|x\|.$$

Together we have

$$\frac{1-\epsilon}{\lambda}\|x\|^2 \leq \alpha_i \leq (\frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}))\|x\|^2,$$

which means that

$$\frac{1-\epsilon}{\lambda} \leq \frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}).$$

However, since $\frac{1}{2} - \sqrt{\frac{1}{4} - \frac{\epsilon d}{1-\epsilon}} < \lambda < \frac{1}{2} + \sqrt{\frac{1}{4} - \frac{\epsilon d}{1-\epsilon}}$, and $\|x\|^2 \geq \rho_{d'}^2 = \frac{1}{(d-d') + \frac{d'}{\lambda^2}}$, we always have

$$\begin{aligned}
\frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}) &\leq (d-d') + \frac{d'}{\lambda^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}) \\
&= d - (d-1)(1-\epsilon) + \epsilon(\frac{1}{\lambda^2} - 1)d' \\
&\leq d - (d-1)(1-\epsilon) + \epsilon(\frac{1}{\lambda^2} - 1)d \\
&= 1 - \epsilon + \frac{\epsilon d}{\lambda^2} \\
&< \frac{1-\epsilon}{\lambda},
\end{aligned}$$

which is a contradiction. Therefore x cannot be an ϵ-approximation for two configurations of the same layer. ◀

Now we are ready to prove Lemma 10. At the coordinator, pre-compute all possible $\rho_{d'}$, with $d' = (0, 1, \cdots, d)$. Compare the polytope distance of the ϵ-approximation x to every segment $[\rho, \frac{1}{1-\epsilon}\rho]$. If $\|\overline{ox}\|$ falls within the segment corresponding to $\rho_{d'}$, then Claim 1 guarantees that the configuration that we want is in the layer with order d'; Then for all configurations C in that layer, check if x is an ϵ-approximation for C by testing whether for all $p \in C$, $p|_x \geq (1-\epsilon)\|x\|$. Since x is an ϵ-approximation, so there is at least one configuration C that will pass the test; and Claim 2 guarantees that there is only one such C. Return C as the desired configuration. This completes the proof for Lemma 10

Lemma 10 means that if we solve the approximate version of the distributed polytope distance problem, we can solve the k-OR problem. Hence the communication complexity of the approximate distributed SVM is $\Omega(kd)$, proving Theorem 6. ◀

Now we are ready to give the distributed version of Gilbert Algorithm (*i.e.*, Algorithm 2) in the coordinator model.

Algorithm 2 Distributed Gilbert Algorithm

```
1: INPUT : A $d$ dimensional point set $P$ arbitrarily distributed among $k$ nodes.
2: OUTPUT(by the coordinator) : A $(1-\epsilon)$-approximation of the polytope distance of $P$
   to $o$.
3: Initialize $i = 1$; All nodes send to the coordinator its closest point to $o$; The coordinator
   picks the global closest point to $o$ as $p_1$;
4: In step $i$, the coordinator sends $x_i$ to all nodes; upon receiving $x_i$, each node picks one
   of its points that has the smallest $p|_{x_i}$ and send back to the coordinator; the coordinator
   picks the point that has the smallest $p|_{x_i}$ based on the points it received; Denote this
   point as $p_{i+1}$ and find $x_{i+1}$ as in non-distributed version.
5: Return $x_{i+1}$ when it is a $(1-\epsilon)$-approximation. Otherwise go to Line 4 with $i = i + 1$.
```

In each step of Algorithm 2, the coordinator sends the current solution x_i to all k nodes. Upon receiving x_i, each node computes the smallest projection distance onto vector $\overline{ox_i}$ based on its own share of points, and return it to the coordinator. The coordinator picks the point that has the smallest projection distance onto vector x_i among all returned points, and uses it as p_{i+1}. Each step of the algorithm involves one round of communication between the coordinator and all k nodes. Thus the communication of each step is $O(k)$. By Theorem 5, we have the following theorem.

▶ **Theorem 11.** *The communication complexity of Algorithm 2 is $O(\frac{k}{\epsilon})$ in the coordinator model.*

In the construction of the proof of Theorem 6 we can take $\epsilon = \Theta(\frac{1}{d})$, resulting in a communication cost of $\Theta(kd)$ for Algorithm 2. Suppose that there exists an algorithm with asymptotically smaller communication cost than Algorithm 2, it will also solve k-OR problem using $o(kd)$ communication, contradicting Lemma 8. So there doesn't exist an algorithm that computes a $(1-\epsilon)$ approximation with asymptotically smaller communication than Algorithm 2.

4 Robust Distributed SVM

In this section we present an algorithm for explicitly avoiding the influence of outliers in the distributed SVM problem. We first consider the following problem.

▶ **Definition 12** (Distributed One-class SVM with Outliers)**.** Given P as a point set of size n in d dimensional space that is arbitrarily distributed among k nodes, and γ as the fraction of outliers in P, find the subset $P^{'} \subseteq P$ of size $(1-\gamma)n$ so that the margin separating the origin and $P^{'}$ is maximized.

Notice that in this problem setting, there are two factors that need to be considered when evaluating the quality of the approximation result: the width of the margin, and the number of outliers accurately pruned out. Thus we need a new definition of approximation.

▶ **Definition 13.** For two constants $\epsilon, \delta > 0$, a margin M is an (ϵ, δ)-approximation, if the width of M is larger than or equal to $(1-\epsilon)\rho$, and the number of outliers identified by M is no more than $(1+\delta)\gamma|P|$.

In this problem, we aim to prune out exactly γn points (as outliers) so that the rest of the points can be separated from the origin by the largest possible margin. In this scenario, Gilbert algorithm may perform arbitrarily bad. This is because in each step, Gilbert algorithm finds the point that has the minimum projection distance and there is a possibility that this point happens to be an outlier. As a gradient descent procedure, Gilbert algorithm does not have the ability to recover from the negative impact of picking an outlier. To avoid this problem, a key observation is that Gilbert algorithm does not need to always identify the point with the minimum projection distance; it is actually sufficient to find one point (to maintain a fast convergence rate) as long as its projection distance is one of the w smallest for some w to be determined later. Based on this observation, Ding and Xu [7] has developed a new framework, called *Random Gradient Descent (RGD)* tree, to explicitly deal with outliers using Gilbert algorithm.

4.1 RGD Tree: Explicitly Avoiding the Influence of Outliers in SVM

Roughly speaking, RGD tree is a modified version of Gilbert algorithm. In each step, it randomly samples w points from the $t > 1$ points which have the smallest projection distances, and considers each of the w points as if it is the point with smallest projection distance. This results in a computation tree with a branching factor of w. The value w is chosen to have the property that there is a high probability that the w chosen points contain at least one point that is not an outlier. Together with the fast convergence rate of Gilbert algorithm, we can have a node in the RGD tree whose path to the root contains only points that are not outliers with high probability. Then this path is the desired computation path of Gilbert algorithm in an outlier-free environment.

The following theorem from [7] guarantees the performance of the RGD tree:

▶ **Theorem 14.** *With high probability (larger than $1-\mu$), there exists at least one node in the resulting RGD tree which yields an (ϵ, δ)-approximation, with running time linear in n and d.*

4.2 Extending RGD Tree to Distributed Settings

Given the fact that RGD tree is a variant of Gilbert algorithm, it can be naturally extended to distributed settings. One simple solution is to let the coordinator send the current solution (x_i) to each distributed node for them to compute and return its own t points with the smallest projection distance to $\overline{ox_i}$. However such a naive approach will incur large communication cost, because now each step will need $O(kt)$ communication, instead of $O(k)$ communication as in the no-outlier case. Actually the problem of finding the t-th smallest number in a distributed setting has been extensively studied ([17, 21, 13]). [13] gives a randomized algorithm that has communication complexity of $O(k \log(t))$, and a deterministic algorithm with communication complexity $O(k \log^2(t))$. In this paper we give a deterministic algorithm with communication complexity $O(k \log(t))$ in the coordinator model. Formally, consider the following problem.

▶ **Definition 15.** (Distributed t-selection): Given n distinct numbers distributed among k nodes, find the t-th smallest number.

In this problem, we only consider distinct numbers, since we can use any tie-breaker to distinguish duplicated numbers. Then we have the following result.

▶ **Lemma 16.** *Distributed t-selection can be solved deterministically with $O(k \log(t))$ transfers of numbers in the coordinator model.*

To prove Lemma 16, we give an algorithm (Algorithm 3) with the claimed communication complexity.

Before analyzing the communication complexity of Algorithm 3, we first show its *Correctness.* The algorithm acts like the classical linear-time selection algorithm. The coordinator computes two "weighted" medians of medians ($m_{\hat{l}_h}$ and $m_{\hat{l}_{h+1}}$) for all distributed nodes, and tell them to discard certain portion of its numbers based on the location of the t-th smallest number. In Line **11**, case 1 means that the t-th smallest number is smaller than the first "weighted" median $m_{\hat{l}_h}$, so it is safe to discard all numbers no smaller than $m_{\hat{l}_h}$; case 2 means that the t-th smallest number is between $m_{\hat{l}_h}$ and $m_{\hat{l}_{h+1}}$, so it is safe to discard all numbers no larger than $m_{\hat{l}_h}$ and all numbers no smaller than $m_{\hat{l}_{h+1}}$; Similarly, we can show for Case 3. We also update the value of t if numbers smaller than the t-th smallest number are discarded. The algorithm either finds the t-th smallest number during the loop of Lines **6** to **12**, or finds it by the coordinator in a non-distributed fashion (when there are fewer numbers left). Thus we have the correctness of Algorithm 3.

For communication complexity, we have the following lemma (proof of Lemma 16 is left in the appendix due to space limit).

▶ **Lemma 17.** *Each iteration of Lines **6** to **12** will discard at least a fraction of $\frac{1}{4}$ of the current numbers holden by all nodes.*

Now we are ready to present the distributed version of the RGD tree algorithm (*i.e.,* Algorithm 4), and the analysis of its communication complexity.

The RGD tree has $O(w^h)$ nodes (which is constant w.r.t. n and d). To generate one node, we need $O(k \log(t))$ communication. Notice that each time we draw the sample S_v, there are $O(w)$ extra communication. Thus on average each node in the sample (*i.e.,* its associated point belongs to the sample) is only charged $O(1)$ extra communication, which does not change asymptotically the $O(k \log(t))$ communication incurred by applying Algorithm 3. This leads to the following theorem.

▶ **Theorem 18.** *The communication complexity of Algorithm 4 is $O(w^h k \log(t))$.*

4.3 Extension to Two-Class SVM

We will briefly explain the reasons why an extension to two-class SVM is straight forward. More details are left to the full version of the paper. Finding the polytope distance between two point sets is equivalent to finding the polytope distance between the Minkowski difference of the two point sets and the origin. Taking the Minkowski difference will increase the size of the problem from $O(n)$ to $O(n^2)$, however using the tricks in [7] and [10] we can achieve the same running time and communication performance as the one-class case.

Algorithm 3 Deterministic distributed t-selection algorithm

```
INPUT : $n$ distinct numbers arbitrarily distributed among $k$ nodes, a natural number $t$.

OUTPUT(by the coordinator) : the $t$-th smallest number of the $n$ numbers.
(Pre-process:) For each node, if it holds more than $t$ numbers, do a local sorting and keep the smallest $t$ numbers and discard the rest.
(Pre-process:) The coordinator sends a distinct number $l \in [1, k]$ to each node as their label.
repeat
    For each node, send to the coordinator a message containing a triple $(m_l, n_l, l)$, where $m_l$ is the median of the numbers stored in the node, $n_l$ is the $\#$ of numbers in the node, and $l$ is its label.
    Upon receiving messages from all nodes, the coordinator first sorts the messages in ascending order of their $m_l$. Suppose in the new order we have $m_{\hat{l}_1} < m_{\hat{l}_2} < \cdots < m_{\hat{l}_k}$.

    Let $m_{\hat{l}_0} = -\infty$. The coordinator computes a value $h$ such that $\sum_{l:m_l \le m_{\hat{l}_h}} n_l < \frac{1}{2}\sum_l n_l$ and $\sum_{l:m_l \le m_{\hat{l}_{h+1}}} n_l \ge \frac{1}{2}\sum_l n_l$
    The coordinator sends a pair $(m_{\hat{l}_h}, m_{\hat{l}_{h+1}})$ to all nodes.
    Upon receiving $(m_{\hat{l}_h}, m_{\hat{l}_{h+1}})$ from the coordinator, each node sends to the coordinator a triple $(a_l, b_l, l)$, where $a_l$ is the $\#$ of its numbers smaller than $m_{\hat{l}_h}$, $b_l$ is the $\#$ of its numbers smaller than $m_{\hat{l}_{h+1}}$, $l$ is its label.
    After receiving all $(a, b, l)$ messages from all nodes, the coordinator checks which of the following cases will happen (notice that since $m_{\hat{l}_h} < m_{\hat{l}_{h+1}}$ we always have $\sum a < \sum b$):
        1. $t - 1 < \sum a$;
        2. $\sum a < t - 1 < \sum b$;
        3. $t - 1 > \sum b$;
        4. $t - 1 = \sum a$;
        5. $t - 1 = \sum b$.
    For case 4, output $m_{\hat{l}_h}$ and halt; for case 5, output $m_{\hat{l}_{h+1}}$ and halt; otherwise, send $i$ to all nodes for cases $i$. For case 2, update $t$ to be $t - \sum a$; for case 3, update $t$ to be $t - \sum b$.
    For each node, if it receives a "1" from the coordinator, discard all numbers larger than $m_{\hat{l}_h}$; if it receives a "2", discard all numbers smaller than $m_{\hat{l}_h}$ and numbers larger than $m_{\hat{l}_{h+1}}$; otherwise discard all numbers smaller than $m_{\hat{l}_{h+1}}$
until $\sum n_l = O(k)$
All nodes send their numbers to the coordinator.
Now the numbers are no longer distributed, and the coordinator simply outputs the $t$-th smallest number.
```

Algorithm 4 Distributed RGD tree

```
INPUT : A $d$ dimensional point set $P$ arbitrarily distributed among $k$ nodes, with a fraction $\gamma$ of it being outliers; three parameters $0 < \mu, \delta < 1$, $h = 2(\frac{1}{\epsilon}(\frac{D}{\rho}+1))^2 \ln(\frac{D}{\rho}+1)$.
OUTPUT(by the coordinator) : An RGD tree with each node associated with a candidate for an approximation solution to the One-class SVM with outliers problem.
The coordinator randomly select a point from $P$ as $x$. Initialize the tree at root $x$.
Recursively grow the tree in the following manner:
For a node $v$ associated with point $x_v$, if its height is $h$, it becomes a leaf; Otherwise, do the following:
    1. Let $t = (1+\delta)\gamma|P|$. The coordinator finds the point $p_t$ whose projection distances to $\overline{ox_v}$ are the $t$'th smallest using Algorithm 3;
    2. Take a random sample $S_v$ of size $w = (1+\frac{1}{\delta})\ln\frac{\mu}{h}$ in the following manner: the coordinator randomly take a label of the nodes, and ask the node with this label for a random point of its holdings whose projection distance is smaller than the projection distance of $p_t$. Repeat until the coordinator has a sample $S_v$ of size $w$. For each point $s \in S_v$, create a child of $v$ in the RGD tree and associate it with point $x_v^s$ which is the point on line segment $[sx_v]$ closest to $o$.
```

References

1 Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed clustering on graphs. *CoRR*, 2013.

2 Aurélien Bellet, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan, and Fei Sha. Distributed frank-wolfe algorithm: A unified framework for communication-efficient sparse learning. *CoRR*, 2014.

3 G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. *NIPS*, 2000.

4 E. Y. Chang, K. Zhu, H. Wang, H. Bai, H. Li, Z. Qiu, and H. Cui. Psvm: Parallelizing support vector machines on distributed computers. *21st Neural Info. Processing Systems Conf*, 2007.

5 C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 1995.

6 Hal Daumé, Jeff M. Phillips, Avishek Saha, and Suresh Venkatasubramanian. Efficient protocols for distributed classification and optimization. In *Proceedings of the 23rd International Conference on Algorithmic Learning Theory*, 2012.

7 H. Ding and J. Xu. Random gradient descent tree: A combinatorial approach for svm with outliers. *Association for the Advanced of Artificial Intelligence Conf.*, 2015.

8 P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-based distributed support vector machines. *Journal of Machine Learning Research*, 2010.

9 G. Fung and O. L. Mangasarian. Incremental support vector machine classification. *Proc. of the SIAM Intl. Conf. on Data Mining*, 2002.

10 G. Gartner and M. Jaggi. Coresets for polytope distance. *SOCG*, 2009.

11 E. G. Gilbert. An iterative procedure for computing the minimum of a quadratic form on a convex set. *SIAM Journal on Control*, 1966.

12 H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. *Advances in Neural Information processing Systems*, 2005.

13 F. Kuhn, T. Locher, and R. Wattenhofer. Tight bounds for distributed selection. *Symposium on Parallelism in Algorithms and Architectures*, 2007.

14 Y. Lu, V. Roychowdhury, and L. Vandenberghe. Distributed parallel support vector machines in strongly connected networks. *IEEE Tran. on Neural Networks*, 2008.

15 N. M. Luscombe, D. Greenbaum, and M. Gerstein. Review: What is bioinformatics? an introduction and overview. *Yearbook of Medical Informatics*, 2001.

16 A. Navia-Vazquez, D. Gutierrez-Gonzalez, E. Parrado-Hernandez, and J. J. Navarro-Abellan. Distributed support vector machines. *IEEE Tran. on Neural Networks*, 2006.

17 A. Negro, N. Santoro, and J. Urrutia. Efficient distributed selection with bounded messages. *IEEE Trans. Parallel and Distributed Systems*, 1997.

18 D. Pechyony, L. Shen, and R. Jones. Solving large scale linear svm with distributed block minimization. *NIPS 2011 Workshop on Big Learning: Algorithms, Systems, and Tools for Learning at Scale*, 2011.

19 J. Phillips, E. Verbin, and Q. Zhang. Lower bounds for number in hand multiparty communication complexity, made easy. *SODA*, 2012.

20 Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*. NIPS, 2011.

21 N. Santoro, J. B. Sidney, and S. J. Sidney. A distributed selection algorithm and its expected communication complexity. *Theoretical Computer Science*, 1992.

22 J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 2008.