# RD-SVM: A RESILIENT DISTRIBUTED SUPPORT VECTOR MACHINE

*Zhixiong Yang and Waheed U. Bajwa*

Dept. of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, USA
Emails: {zhixiong.yang, waheed.bajwa}@rutgers.edu

## ABSTRACT

Support vector machines (SVMs) are one of the most widely used supervised learning algorithms for classification problems. Recent years have witnessed an increasing interest in distributed variants of SVMs, in which the (labeled) training data is distributed across different nodes. While a number of algorithms have been developed in this regard, they all make the simplified assumption that every node in the network operates as intended. In many applications, however, it is common for some of the nodes to undergo failures due to faulty equipment, cyber attacks, etc., and inject faulty data into the network. This kind of failure, termed Byzantine failure, is impossible to protect against using existing distributed SVM algorithms. This paper revisits the problem of distributed SVM under the possibility of Byzantine failures in the network. In this regard, it proposes a novel algorithm for distributed SVM that remains resilient to Byzantine failures as long as the number of faulty nodes in the network is not too large. Numerical results on real-world data confirm the superiority of the proposed algorithm over existing approaches.

***Index Terms*—** Byzantine fault tolerance, distributed classification, distributed consensus, support vector machine

## 1. INTRODUCTION

Classification is one of the most common tasks in machine learning. While a number of techniques exist in the literature for learning (linear and nonlinear) classifiers from (labeled) training data, *support vector machines* (SVMs) have emerged as one of the most popular approaches to (supervised) classifier training. This popularity of SVMs, originally proposed as maximum-margin classifiers for linearly separable classes [1], can be partly attributed to their ease of implementation, their ability to handle non-separable classes [2], and their generalizations to nonlinear classification problems [3]. The last two decades in particular have seen significant advances in the theory and practice of SVMs under the assumption of training data available at a single location; see, e.g., [4–8].

In recent years, proliferation of sensors, smart devices, and cloud storage have brought to the fore another important aspect of classification problems in many application areas, namely, classifier learning from training data distributed across multiple locations. While a number of techniques have been proposed in the literature for distributed classifier training [9, 10], distributed variants of SVMs are once again emerging as one of the most popular means of addressing this problem; see, e.g., [11–14]. Notwithstanding the algorithmic and analytical differences among these and related works, existing literature on distributed SVMs makes the simplified assumption that all locations (henceforth referred to as nodes) involved in distributed classifier training operate within the learning framework as originally intended. Unfortunately, this somewhat idealized assumption is seldom true in real-world settings. Rather, nodes in large-scale distributed systems routinely undergo *undetected failures* due to malfunctioning equipment, cyber attacks, etc., with the end result being undetected injection of faulty data into the learning framework. Such failures in which nodes can arbitrarily deviate from their intended behavior, referred to as *Byzantine failures* [15, 16], can render even the most sophisticated distributed SVM algorithms ineffective.

Our goal in this paper is to revisit the problem of distributed SVM under the realistic assumption that some nodes will undergo Byzantine failures during classifier training. Our main contribution in this regard is development of a Byzantine fault-tolerant algorithm, termed *resilient distributed support vector machine* (RD-SVM), for learning of a maximum-margin classifier from training data distributed across a network of nodes. The distinguishing feature of RD-SVM is that it learns a common separating hyperplane at every functioning node in the network, even when some of the nodes in the network transmit (arbitrarily) faulty data to their neighboring nodes. In order to demonstrate the effectiveness of RD-SVM and its superiority over existing work, we also report outcomes of binary and multiclass classification experiments performed using RD-SVM on CIFAR-10 [17] and MNIST [18] datasets, respectively.

To the best of our knowledge, RD-SVM is the first distributed SVM algorithm capable of tolerating Byzantine failures in networks. It accomplishes this by leveraging some of the ideas in [13], which puts forth a distributed SVM algorithm based on distributed consensus [19], and in [20–22], which present fault-tolerant distributed strategies for *averaging* consensus [23]. We conclude by noting that our focus in this initial work is on linear SVM and resilience to faulty nodes, while extensions to nonlinear (kernel) SVM and explicit identification of faulty nodes will be investigated in future work.

## 2. PROBLEM FORMULATION

We consider a connected network of $M$ nodes, expressed as an undirected, static, connected graph $G(J, E)$ (see Fig. 1). Here, the set of vertices $J := \{1, \ldots, M\}$ represents nodes in the network, while the set of edges $E$ represents bidirectional communication links between nodes, with $(j, i) \in E$ if and only if node $j$ and node $i$ are connected to each other. It is assumed that each node $j$ has access to a local training set $S_j := \{(x_{j,n}, y_{j,n})\}_{n=1}^{N_j}$, where $x_{j,n} \in \mathbb{R}^p$ denotes a training data vector and $y_{j,n} \in \{-1, 1\}$ denotes the corresponding class label.[1] Our main objective then is to develop a distributed algorithm that enables each node to learn a maximum-margin linear classifier $g_j(x) : x \mapsto \{-1, 1\}$ such that $\forall j, g_j(x) \approx g(x)$, where

This work is supported in part by the ARO (grant W911NF-14-1-0295), NSF (grant CCF-1453073), and ARL (Robotics CTA subaward).

[1] Extension of this binary classification setup to a multiclass setting is straightforward and will be discussed in Sec. 5.

$g(x)$ denotes the centralized maximum-margin linear classifier that could have been learned using the entire training data $\{S_j\}_{j=1}^M$.

The assumption that makes distributed classifier training challenging is that the local training sets cannot be gathered at a single location due to bandwidth limitations, privacy concerns, storage constraints, etc. Rather, the local $g_j(\cdot)$'s need to be iteratively learned using the following message passing protocol. Suppose $v_j[k]$ denotes some (private) "summary statistic" at node $j$ at the start of iteration $k$. Then each node $j$ broadcasts this summary statistic to the nodes in its neighborhood, defined as $\mathcal{N}_j := \{i \in J : (j,i) \in E\}$. Next, all nodes update their respective summary statistics as follows: $v_j[k+1] = f_j(\{v_i^j[k]\}_{i\in\mathcal{N}_j}), k \in \mathbb{Z}_{\geq 0}$, where $v_i^j[k]$ denotes data received by node $j$ from node $i$ in iteration $k$, and the update functions $f_j(\cdot)$, which need not be the same for all nodes, are assigned to the nodes at the start of the algorithm. Finally, each node $j$ uses its updated summary statistic to modify the local classifier $g_j(\cdot)$.

While distributed classifier training using such message passing protocols has been previously studied [11–14], such works require all nodes in the network to operate as intended. In contrast, the main assumption in this paper is that some of the nodes in the network can undergo Byzantine failures, formally defined as follows.

**Definition 1.** *A node $j \in J$ is said to be Byzantine if during any iteration $k \in \mathbb{Z}_{\geq 0}$, it either sends one of more of its neighbors an incorrect version of its summary statistic, i.e., $\exists i \in \mathcal{N}_j : v_j^i[k] \neq v_j[k]$, or it updates its summary statistic using $f_j'(\cdot) \neq f_j(\cdot)$.*

In this paper, we assume there are at most $F$ Byzantine nodes in the neighborhood of any functioning (non-Byzantine) node. Our goal then is development of: $(i)$ a Byzantine fault-tolerant algorithm for distributed classifier training, and $(ii)$ conditions on $F$ that allow the developed algorithm to be resilient to Byzantine failures.

## 3. BACKGROUND

Our focus here is on a (linear) support vector machine (SVM), which is a popular approach to classifier training. In words, an SVM uses (labeled) training data to learn a hyperplane that separates data into two classes. In a centralized setting, the training of an SVM can be described as follows. A single location is given access to a labeled training set $S := \{(x_n, y_n)\}_{n=1}^N$ of $N$ samples. The goal then is to make use of $S$ and learn a maximum-margin decision hyperplane $g(x)$, defined in terms of a normal vector $w$ and an intercept $b$ as $g(x) = w^T x + b$. In order to learn the corresponding pair $(w, b)$, a centralized SVM solves the following optimization problem:

$$\min_{w,b,\xi_n} \quad \frac{1}{2}\|w\|_2^2 + C\sum_{n=1}^N \xi_n \quad \text{such that}$$
$$\forall n, y_n(w^T x_n + b) \geq 1 - \xi_n, \text{and } \forall n, \xi_n \geq 0. \qquad (1)$$

Here, $\{\xi_n\}$ are termed slack variables, while $C$ is a positive scalar constant; see [24] for a derivation of (1). This process of obtaining optimal $w$ and $b$ using (1) is referred to as "training" in the literature. After training is finished, any new sample $x$ can be classified as $y = 1$ if $g(x) > 0$ and $y = -1$ if $g(x) < 0$.

While traditional literature on SVM training focuses on a centralized setup, recent years have seen an interest in training an SVM for the case when training data is partitioned across $M$ nodes, i.e., $S = \bigcup_{j\in J} S_j$. Under the assumption of no Byzantine nodes in the network, this can be accomplished using the *distributed support vector machine* (DSVM) approach described in [13]. The training problem in this case can be defined as

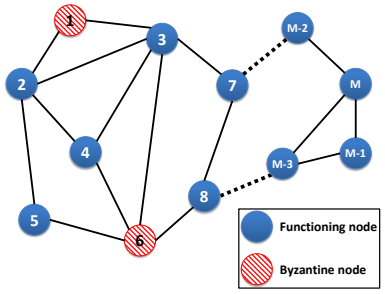


**Fig. 1**. An example of a network with Byzantine nodes. (Undetected) Byzantine nodes can make distributed learning algorithms go awry through communication of faulty data to their neighbors.

$$\min_{w_j,b_j,\xi_{j,n}} \quad \frac{1}{2}\sum_{j=1}^M \|w_j\|_2^2 + MC\sum_{j=1}^M\sum_{n=1}^{N_j}\xi_{j,n} \qquad (2)$$
$$\text{s.t.} \quad y_{j,n}(w_j^T x_{j,n} + b_j) \geq 1 - \xi_{j,n}, \quad \forall j \in J, n = 1, ..., N_j,$$
$$\xi_{j,n} \geq 0, \quad \forall j \in J, n = 1, ..., N_j,$$
$$w_j = w_i, b_j = b_i, \quad \forall j \in J, i \in \mathcal{N}_j.$$

In [13], the optimal value of $v_j$ is obtained by solving the dual of a modified version of (2).[2] Specifically, let $\lambda$ denote the Lagrangian corresponding to the constraint $y_{j,n}(w_j^T x_{j,n} + b_j) \geq 1 - \xi_{j,n}$ and $\alpha$ denote the Lagrangian corresponding to the constraint $v_j = v_i$. Then [13] solves (2) in a distributed fashion using the *alternating direction method of multipliers* [25]. This involves in each iteration $k$, node $j$ storing a vector $v_j[k] = [w_j^T, b_j]^T$ as its summary statistic and then broadcasting $v_j[k]$ to all nodes $i \in \mathcal{N}_j$. Next, all nodes in the network update their local $\lambda_j$'s, $v_j$'s, and $\alpha_j$'s as follows:

$$\begin{aligned}\lambda_j[k+1] = \underset{\mathbf{0}_j \preceq \lambda_j \preceq MC\mathbf{1}_j}{\arg\max} & -\frac{1}{2}\lambda_j^T Y_j X_j U_j^{-1} X_j^T Y_j \lambda_j \\ & + (1_j + Y_j X_j U^{-1} f_j[k])^T \lambda_j, \\ v_j[k+1] = U_j^{-1}(X_j^T Y_j \lambda_j[k+1] - f_j[k]), & \text{ and} \\ \alpha_j[k+1] = \alpha_j[k] + \frac{\eta}{2}\sum_{i\in\mathcal{N}_j}(v_j[k+1] - v_i[k+1]), &\end{aligned} \qquad (3)$$

where $\preceq$ represents element-wise inequality and $\mathbf{0}_j$ (resp., $\mathbf{1}_j$) is a vector with all entries equal to 0 (resp., 1). Here, $U_j$ and $f_j$ are defined as $U_j := (1 + 2\eta|\mathcal{N}_j|)I_{p+1} - \prod_{p+1}$ and $f_j[k] := 2\alpha_j[k] - \eta\sum_{i\in\mathcal{N}_j}(v_j[k] + v_i[k])$, where $\eta > 0$ and $\prod_{p+1}$ is a $(p+1) \times (p+1)$ matrix with zeros everywhere except $[\prod_{p+1}]_{(p+1)(p+1)} = 1$. Denoting the optimal value of $(w_j, b_j)$ in (2) as $(w^*, b^*)$, [13, Prop. 1] shows that $v_j[k] \to [w^{*T}, b^*]^T$. Finally, note that $(w^*, b^*)$ also correspond to the solution of the following equivalent problem:

$$\min_{w_j,b_j} \frac{1}{2}\sum_{j=1}^M \|w_j\|_2^2 + MC\sum_{j=1}^M\sum_{n=1}^{N_j}\ell(y_{j,n}, x_{j,n}, w_j, b_j) \qquad (4)$$
$$\text{such that} \quad w_j = w_i, b_j = b_i, \quad \forall j \in J, i \in \mathcal{N}_j,$$

where the hinge-loss function $\ell(\cdot)$ is given by $\ell(y_{j,n}, x_{j,n}, w_j, b_j) = \max\{0, 1 - y_{j,n}(w_j^T x_{j,n} + b_j)\}$.

[2] Connected nature of the network and the last constraint in (2) result in all optimal $v_j$'s taking the same value [13, Lemma 1].

## 4. RD-SVM: A RESILIENT DISTRIBUTED SVM

While the DSVM algorithm performs well in ideal settings, its performance can be arbitrarily bad when there are Byzantine nodes in the network. In order to illustrate this point, suppose there are only 2 nodes in the network, termed 'A' and 'B'. Node A has local training set $x_{1,1} = (1, 0), y_{1,1} = 1$ and $x_{1,2} = (0, 1), y_{1,2} = -1$, while node B has a set $x_{2,1} = (0, -1), y_{2,1} = 1$ and $x_{2,2} = (-1, 0), y_{2,2} = -1$. When both nodes operate normally and update their values according to (3), they learn the hyperplane described by the equation $x_2 = x_1$. However, if node B is Byzantine and holds its hyperplane $L_2$ at $x_1 = -1.5$, node A's hyperplane ($L_1$) will iteratively converge to $x_1 = -1.5$ also ($L_1'$), which is far from the optimal one (Fig. 2). Here, we propose an algorithm termed *resilient distributed support vector machine* (RD-SVM) to address this issue.

The main idea behind RD-SVM is to ignore some of the values that a node receives from its neighbors in each iteration. Indeed, the only way that a Byzantine node can push another node $j$ in the wrong direction is for it to send node $j$ a summary statistic that makes node $j$ incorrectly update its own summary statistic. However, when the majority of neighbors of node $j$ are non-Byzantine, the "faulty" received statistic can be considered an outlier and eliminated after identification. Below, we describe our approach to such outlier identification and elimination.

Recall that hinge loss is an important metric to evaluate the performance of linear classifiers. When comparing two classifiers, a lower hinge loss stands for a better performance. Notice further from (4) that distributed SVM training effectively reduces to minimization of the local hinge loss at each node. RD-SVM therefore uses local hinge loss to determine how different the neighborhood classifiers are from the local classifier at each node. It then declares the neighboring classifiers with the largest differences from the local classifier as *potential* outliers. To this end, we partition the set $S_j$ at each node into two sets $A_j := \{(x, y) : (x, y) \in S_j, y = 1\}$ and $B_j := \{(x, y) : (x, y) \in S_j, y = -1\}$. In words, $A_j$ contains samples with label 1 and $B_j$ contains samples with label $-1$. Next, for each neighboring classifier and the local classifier, we calculate their hinge losses on sets $A_j$ and $B_j$ as follows:

$$\begin{aligned}
&\forall i \in \mathcal{N}_j, \ell_{Ai,j} := \sum_{(x,y)\in A_j} \max(0, 1 - y(w_i^T x + b_i)),\\
&\ell_{Aj,j} := \sum_{(x,y)\in A_j} \max(0, 1 - y(w_j^T x + b_j)),\\
&\ell_{Bi,j} := \sum_{(x,y)\in B_j} \max(0, 1 - y(w_i^T x + b_i)), \quad \text{and}\\
&\forall i \in \mathcal{N}_j, \ell_{Bj,j} := \sum_{(x,y)\in B_j} \max(0, 1 - y(w_j^T x + b_j)).
\end{aligned} \tag{5}$$

Finally, we also define two sets $E_{Aj} := \{i : i \in \mathcal{N}_j, \ell_{Ai,j} > \ell_{Aj,j}\}$ and $E_{Bj} := E_{Bj} = \{i : i \in \mathcal{N}_j, \ell_{Bi,j} > \ell_{Bj,j}\}$. Here, $E_{Aj}$ (resp., $E_{Bj}$) contains all nodes that result in classifiers with larger hinge losses on set $A_j$ (resp., $B_j$) than the local classifier.

Identification of potential outliers in RD-SVM is a function of the maximum number of Byzantine nodes in any neighborhood in the network. Suppose there are at most $F$ Byzantine nodes in the neighborhood of any node $j$. Then if there are no more than $F$ elements in $E_{Aj}$, we declare all nodes in $E_{Aj}$ as outliers. On the other hand, if there are more than $F$ elements in $E_{Aj}$, we declare $F$ nodes with the largest $\ell_{Ai,j}$'s as outliers. Similarly, if there are no more than $F$ elements in $E_{Bj}$, we declare all nodes in $E_{Bj}$ as outliers. But if there are more than $F$ elements in $E_{Bj}$, we only declare $F$ nodes with the largest $\ell_{Bi,j}$'s as outliers. Next, if node $i$ is declared as an outlier by node $j$, then node $j$ eliminates node $i$ from its neighborhood $\mathcal{N}_j$ during the current iteration. Let $\mathcal{N}_j'$ denote the new neighborhood set of node $j$ after elimination of the outliers. Then the update step (3) becomes

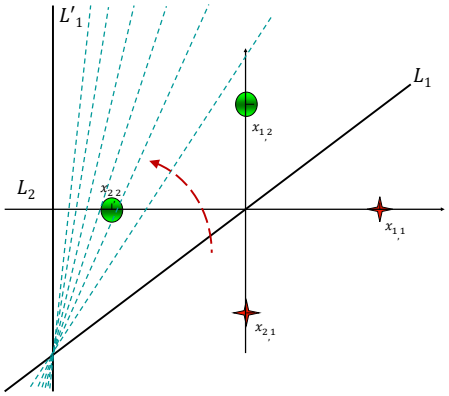


**Fig. 2**. $L_1$ and $L_2$ denote the initial separating hyperplanes of nodes $A$ and $B$, respectively, while $L_1'$ denotes the final hyperplane at node $A$. When node $B$ holds $L_2$ at $x_1 = -1.5$, DSVM makes $L_1$ converge to $L_2$ (dashed lines show iterative convergence of $L_1$ to $L_1'$).

$$\begin{aligned}
\lambda_j[k+1] = &\underset{\mathbf{0}_j \preceq \lambda_j \preceq MC\mathbf{1}_j}{\arg\max} -\frac{1}{2}\lambda_j^T Y_j X_j U_j^{-1} X_j^T Y_j \lambda_j\\
&+ (1_j + Y_j X_j U^{-1} f_j[k])^T \lambda_j,\\
v_j[k+1] = &\ U_j^{-1}(X_j^T Y_j \lambda_j[k+1] - f_j[k]), \text{ and}\\
\alpha_j[k+1] = &\ \alpha_j[k] + \frac{\eta}{2}\sum_{i\in\mathcal{N}_j'}(v_j[k+1] - v_i[k+1]).
\end{aligned} \tag{6}$$

A detailed description of RD-SVM is provided in Algorithm 1.

We now comment on the workings of RD-SVM. Since Algorithm 1 eliminates at most $2F$ nodes during each iteration, where $F$ is the maximum number of Byzantine nodes in any network neighborhood, there are two conditions that need to be met in the network. The first one concerns the size of the neighborhood set, $|\mathcal{N}_j|$. Each node has to have at least $2F + 1$ neighbors to guarantee that it can communicate with the rest of the network. Take node 5 in Fig. 1 as an example. If we set $F = 1$, and $\ell_{A2,5} > \ell_{A5,5}$ and $\ell_{B6,5} > \ell_{B5,5}$, then node 5 eliminates both nodes 1 and 2 from its neighborhood, which means node 5 will be disconnected from the rest of the network. The second condition concerns the *redundancy* in the network. Specifically, there is a possibility after the elimination step that the network becomes divided into two or more disjoint subsets. Once again, we revert to Fig. 1 for an example. If node 3 eliminates node 7 and node 6 eliminates node 8 during some iteration $k$, then nodes $1, 2, \ldots, 6$ will end up communicating only with each other, which might make it impossible to achieve consensus among the functioning nodes. This notion of redundancy (or robustness) can be formulated using the following definition from [22].

**Definition 2.** *Let $S_m$ denotes a subset of $J$ and $\gamma_{S_m}^r$ denotes the set $\gamma_{S_m}^r = \{j \in S_m : |\mathcal{N}_j \setminus S_m| \geq r\}$. A nonempty, nontrivial digraph $G = (J, E)$ of $M \geq 2$ nodes is $(r, s)$-robust for nonnegative integers $r \in \mathbb{Z}_{\geq 0}, 1 \leq s \leq M$ if for every pair of nonempty, disjoint subsets $S_1, S_2$ of $J$, at least one of the following holds: (i) $|\gamma_{S_1}^r| = |S_1|$; (ii) $|\gamma_{S_2}^r| = |S_2|$; and (iii) $|\gamma_{S_1}^r| + |\gamma_{S_2}^r| \geq s$.*

**Algorithm 1** *Resilient Distributed Support Vector Machine*

**Input** $\forall j \in J$, data $S_j$, initial values $\lambda_j[0]$, $v_j[0]$, and $\alpha_j[0] := \mathbf{0}$

$\forall j \in J$, define the sets $A_j$ and $B_j$
**repeat**
  $\forall j \in J$, compute $\lambda_j[k+1]$ using (6)
  $\forall j \in J$, compute $v_j[k+1]$ using (6)
  $\forall j \in J$, broadcast $v_j[k+1]$ to all $i \in \mathcal{N}_j$
  **for** all $j \in J$ **do**
    Compute $\ell_{Ai,j}$, $\ell_{Aj,j}$, $\ell_{Bi,j}$ and $\ell_{Bj,j}$ using (5)
    Define the sets $E_{Aj}$ and $E_{Bj}$
    **if** $|E_{Aj}| \leq F$
      $\mathcal{N}_j' \leftarrow \mathcal{N}_j \setminus E_{Aj}$
    **else**
      $\ell_A^* \leftarrow$ the $F$-th largest value among all $\ell_{Ai,j}$'s
      Define $E'_{Aj} \leftarrow \{i : \ell_{Ai,j} \geq \ell_A^*\}$
      $\mathcal{N}_j' \leftarrow \mathcal{N}_j \setminus E'_{Aj}$
    **end if**
    **if** $|E_{Bj}| \leq F$
      $\mathcal{N}_j' \leftarrow \mathcal{N}_j' \setminus E_{Bj}$
    **else**
      $\ell_B^* \leftarrow$ the $F$-th largest value among all $\ell_{Bi,j}$'s
      Define $E'_{Bj} \leftarrow \{i : \ell_{Bi,j} \geq \ell_B^*\}$
      $\mathcal{N}_j' \leftarrow \mathcal{N}_j' \setminus E'_{Bj}$
    **end if**
    Compute $\alpha_j[k+1]$ using (6)
  **end for**
**until** $v_i = v_j \quad \forall i, j \in J$

Here, we conjecture that RD-SVM can achieve consensus on functioning nodes in the presence of at most $F$ Byzantine nodes in each neighborhood as long as the network is at least $(F+1, F+1)$-robust. In words, this $(F+1, F+1)$-robustness condition means that regardless of the division of network into sets $S_1$ and $S_2$, there will always be at least 1 node left in $S_1$ that can communicate with $S_2$ even after elimination of $F$ outliers by each node in every iteration.

## 5. NUMERICAL RESULTS

We first test the performance of RD-SVM on the CIFAR-10 dataset [17], which includes 60000 different images of 10 subjects. Here, we do binary classification to distinguish between cats and ships. We generate an Erdős–Rényi graph of 25 nodes with parameter 0.51, with the final graph having an average degree of 11.76. The training set at each node corresponds to 2000 images (vectorized into 3072-dimensional vector) each of cats and ships, resulting in a total of 50000 training and 10000 test samples per class. We perform three different experiments. First, we let all the nodes be non-faulty and record the performance of DSVM on the test data. Next, we randomly pick 4 nodes to be Byzantine and make each Byzantine node broadcast a random vector to its neighborhood in each iteration. In this setting, we first record the performance of DSVM on the test data and then record the performance of RD-SVM on the test data.

Fig. 3 shows the results of our experiments for DVSM without Byzantine nodes and RD-SVM with Byzantine nodes. Since RD-SVM strives for consensus only among the non-Byzantine nodes, our results are plotted only for non-faulty nodes. DSVM with Byzantine nodes (not plotted) results in an error rate of 50%, which shows its fragility in the presence of Byzantine nodes. On the other hand, RD-SVM tolerates Byzantine failures with only a slight decrease in performance compared to DSVM without Byzantine nodes.

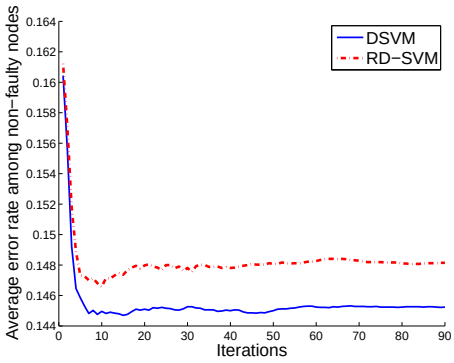


**Fig. 3**. Numerical results on CIFAR-10 dataset for DSVM without Byzantine nodes and RD-SVM with Byzantine nodes. The blue curve shows the error rate as a function of the number of iterations for DSVM, while the red curve shows the error rate (on non-faulty nodes) as a function of the number of iterations for RD-SVM in the presence of 4 Byzantine nodes.

Next, we carry out multiclass classification on MNIST dataset [18], which contains images of handwritten digits from '0' to '9'. The 4 digits picked in our experiments are '0','1','6', and '7'. The network setup in here is the same one described earlier for CIFAR-10 data. We use 200 training and 32 test samples per digit for each node, resulting in a total of 20000 training and 3200 test samples. We make use of the binary tree method in [26] for multiclass classification using binary SVM. Once again, we run experiments using DSVM without Byzantine nodes as well as using RD-SVM and DSVM with 4 random Byznatine nodes. The results of these experiments are reported in Table 1.

| Algorithm | # of Byzantine nodes | Error Rate | Consensus |
|---|---|---|---|
| DSVM | 0 | 2.24% | Yes |
| DSVM | 4 | 75% | No |
| RD-SVM | 4 | 2.25% | Yes |

**Table 1**. Numerical results on MNIST dataset for DSVM without Byzantine nodes, and DSVM and RD-SVM with Byzantine nodes.

It can once again be seen from Table 1 that the DSVM algorithm breaks down in the presence of Byzantine nodes. In fact, the algorithm cannot even achieve consensus in this case due to the random behavior of the Byzantine nodes. On the other hand, RD-SVM is once again resilient to Byzantine nodes and its performance (on non-faulty nodes) is almost as good as in the ideal setting.

## 6. CONCLUSION

In this paper, we presented a distributed, supervised learning algorithm that relaxes the assumption that all nodes in the network operate as intended. We showed through numerical simulations that the proposed algorithm, termed resilient distributed support vector machine (RD-SVM), can successfully tolerate Byzantine failures of nodes in the network during distributed training of an SVM. This is in contrast to prior work in the literature on distributed SVM that tends to be fragile in the presence of Byzantine nodes. Our future work in this direction includes rigorous characterization of the conditions under which RD-SVM achieves consensus on non-faulty nodes in the network.

## 7. REFERENCES


[1] V. N. Vapnik, *Statistical learning theory*, vol. 1, Wiley-Interscience, 1998.

[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th Annu. Workshop Computational Learning Theory (COLT'92)*, 1992, pp. 144–152.

[4] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Machine Learning: ECML-98*, C. Nedellec and C. Rouveirol, Eds., vol. 1398 of *Lecture Notes in Computer Science*, pp. 137–142. Springer Berlin Heidelberg, 1998.

[5] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds., chapter 12. MIT Press, 1999.

[6] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[7] V. Vapnik and O. Chapelle, "Bounds on error expectation for support vector machines," *Neural Computation*, vol. 12, no. 9, pp. 2013–2036, 2000.

[8] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with Gaussian kernel," *Neural Computation*, vol. 15, no. 7, pp. 1667–1689, 2003.

[9] R. Bekkerman, M. Bilenko, and J. Langford, *Scaling up machine learning: Parallel and distributed approaches*, Cambridge University Press, 2011.

[10] Z. Shakeri, H. Raja, and W. U. Bajwa, "Dictionary learning based nonlinear classifier training from distributed data," in *Proc. 2nd IEEE Global Conf. Signal and Information Processing (GlobalSIP'14), Symposium on Network Theory*, Atlanta, GA, Dec. 2014, pp. 759–763.

[11] T. Do and F. Poulet, "Classifying one billion data with a new distributed SVM algorithm," in *Proc. Intl. Conf. Research, Innovation and Vision for the Future*, 2006, pp. 59–66.

[12] A. Navia-Vázquez and E. Parrado-Hernandez, "Distributed support vector machines," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 1091–1097, 2006.

[13] P. Forero, A. Cano, and G. Giannakis, "Consensus-based distributed support vector machines," *J. Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[14] C. Lee and D. Roth, "Distributed box-constrained quadratic optimization for dual linear SVM," in *Proc. 32nd Int. Conf. Machine Learning (ICML'15)*, 2015, pp. 987–996.

[15] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[16] R. Friedman, A. Mostefaoui, and M. Raynal, "Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems," *IEEE Trans. Dependable and Secure Computing*, vol. 2, no. 1, pp. 46–56, 2005.

[17] A. Krizhevsky, "Learning multiple layers of features from tiny images," Technical report, Department of Computer Science, University of Toronto, 2009.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] J. N. Tsitsiklis, *Problems in decentralized decision making and computation*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, Dec. 1984.

[20] S. Sundaram and C. Hadjicostis, "Distributed function calculation via linear iterative strategies in the presence of malicious agents," *IEEE Trans. Autom. Control*, vol. 56, no. 7, pp. 1495–1508, 2011.

[21] H. Zhang and S. Sundaram, "Robustness of information diffusion algorithms to locally bounded adversaries," in *Proc. American Control Conf. (ACC'12)*, 2012, pp. 5855–5861.

[22] H. J. LeBlanc, Haotian Zhang, X. Koutsoukos, and S. Sundaram, "Resilient asymptotic consensus in robust networks," *IEEE J. Select. Areas Commun.*, vol. 31, no. 4, pp. 766–781, Apr. 2013.

[23] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems Control Lett.*, vol. 53, no. 1, pp. 65–78, Sept. 2004.

[24] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[25] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, vol. 3 of *Foundations and Trends in Machine Learning*, Now Publishers Inc., Hanover, MA, Jan. 2011.

[26] G. Madzarov, D. Gjorgjevikj, and I. Chorbev, "A multi-class SVM classifier utilizing binary decision tree," *Informatica*, vol. 33, no. 2, 2009.