Projection-SVM: Distributed Kernel Support Vector Machine for Big Data using Subspace Partitioning

Dinesh Singh and C. Krishna Mohan
Visual Learning and Intelligence (VIGIL) Lab, Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad, Kandi, Sangareddy-502285, Telangana, India
Email: {cs14resch11003, ckm}@iith.ac.in

***Abstract*—The training of kernel support vector machine (SVM) is a computationally complex task for large datasets where the number of samples ranges in millions. This is because kernel matrix (in general not sparse) is both computation expensive and memory intensive. Existing methods hardly achieve a linear scale and suffer from high approximation loss. We propose *Projection-SVM*, a distributed implementation of kernel support vector machine for large datasets using subspace partitioning. In subspace partitioning, a decision tree is constructed on projection of data along the direction of maximum variance (i.e., dominant eigenvector) to obtain smaller partitions (i.e., subspaces) of the dataset. On each of these partitions, a kernel SVM is trained independently over a cluster thereby reducing the overall training time. Also, it results in reducing the prediction time significantly. We demonstrate the efficacy of the proposed approach on eight standard large datasets from various application domains, namely, mnist8m, kddcup99, webspam, etc. where *Projection-SVM* is on an average 150 times faster than sequential SVM while maintaining the classification accuracy. The experimental results also show the superiority of the *Projection-SVM* over the state-of-the-art approaches for distributed kernel SVMs, such as DCSVM, CASVM, and DTSVM.**

***Index Terms*—Machine Learning for Big Data, Distributed Computing, Kernel SVM**

I. Introduction

The support vector machine (SVM) [1] has been immensely successful in the classification of diverse inputs from the fields of computer vision, surveillance systems, cyber-security, genomics, e-commerce, etc. [2]–[6]. However, to make intelligent decisions from such data is becoming increasingly difficult due to easy availability of high volume data [7], most noticeably in the form of text, images, and videos. Since user preferences and trends keep on changing fluidly, analysis of such a large volume of data to support decision making is almost inevitable. Also, the SVM has been used for classification problems in many areas due to its generalization capabilities. SVM is based on statistical learning theory developed by Vapnik [1]. The core of SVM is in solving a quadratic programming (QP), which separates support vectors from the rest of the training data. Let $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ be the dataset with n data points. Where, $\mathbf{x}_i \in \mathbb{R}^d$ is the d dimensional data point with class label $y_i \in \{-1, +1\}$. Then the QP optimization problem for SVM in dual form can be written as

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{n} \alpha_i, \tag{1}$$

$$\textit{s.t.} \ \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C,$$

where, $\alpha_i \in \boldsymbol{\alpha}$ are the Lagrangian multipliers and C is a regularization parameter. Solving equation (1) gives $\boldsymbol{\alpha}$ and a bias value b. All vectors having non-zero $\alpha_i \in \boldsymbol{\alpha}$ are known as support vectors. Then, decision of a unknown test feature vector $\mathbf{x}$ is given by following decision function:

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right). \tag{2}$$

The equation (1) can be solved using existing implementations of SVM such as LIBSVM [8], LS-SVM [9], and SVMlight [10] but they are sequential and cannot be scale to large-scale problems because for a standard SVM, the time complexity of $O(n^3)$ and space complexity of $O(n^2)$ tend to increase rapidly with an increase in the size of the training data [11]. Thus making training of kernel SVM a difficult task for large-scale datasets.

In the modern era, the distributed environments like the high-performance cluster (HPC), big data analytics, and cloud clusters are widely used for solving the data-intensive and time-consuming problems. However, sequential minimal optimization (SMO) [12], the most successful quadratic programming (QP) solver, is a sequential method which cannot leverage the benefits of these distributed environments. The existing state-of-the-art approaches, namely, DCSVM [5], CASVM [13], and DTSVM [14] for approximated distribution of SVM training suffers from high loss of accuracy and increased communication overhead in a distributed system due to the exchange of a large amount of data over the communication network. The existing tree-based techniques like DTSVM [14] use single attribute for data partitioning and thus suffers from high loss of accuracy. The data partitioning based approaches such as DCSVM [5], CASVM [13] and DiP-SVM [6] show high loss in classification accuracy, high communication overhead, and hardly achieve a linear scaling. Thus, they are unable to achieve the desired level of scaling.

In this work, we propose *Projection-SVM* a distributed SVM for large datasets. The dominant eigenvector is used to split the data space into smaller subspaces. A tree-based structure is used to recursively split the data at each node in a similar fashion. This architecture can be viewed as a modified decision tree. In general, decision tree selects most relevant attribute which has the highest correlation to the class labels in comparison to other attributes. However, in this work, we use the projection of data points along the direction of maximum variance in the dataset which is derived from all input dimensions. After partitioning, a kernel SVMs is trained independently on each partition in a parallel or distributed system. The key properties of the proposed *Projection-SVM* are as follows:

1) Novel partitioning approach based on the projection of data on dominant eigenvector using all input dimensions not only reduces the training time but also makes the classification comfortable and reduces the loss of classification accuracy.
2) The proposed *Projection-SVM* does not need to combine the smaller local models into a single global model which further leads to reduced training time.
3) Due to (2) above, the amount of data transferred over the network is also reduced in comparison to existing approaches like DCSVM [5] and CASVM [13].
4) The prediction time is also reduced significantly due to use of a relevant smaller kernel SVM model which contains less number of support vectors in comparison to a single large kernel SVM model.

The rest of the paper is organized as follows. Section 2 discusses related work. The proposed distributed kernel SVM approach is discussed in section 3. Section 4 describes the experimental setup, evaluation method, and results. We conclude in section 5 with references at the end.

II. Related Work

To date, many parallel and distributed implementations of SVM are proposed [15]–[19]. Based on the underlying architecture, we can organize them into four groups, namely, parallel [20], distributed [5], [6], [13], [21], [22], MapReduce [3], [23]–[25], and GPU [26], [27]. Broadly, the distributed SVM approaches can be divided into two categories, namely, *ensemble-based approaches*, and *subspace-based approaches*.

Ensemble based approaches work on the concept of divide-and-conquer. A large-scale SVM training problem is divided into several smaller SVM problems. Each smaller SVM is trained independently and finally a single SVM model is produced by combining the results of all the smaller SVMs. Some of the well known state-of-the-art ensemble-based approaches are discussed below.

In [28], Graf *et al.* proposed a cascade SVM, where the training samples are divided into smaller subsets and local SVM models are trained for each subset using LIBSVM library. The support vectors of local SVMs are then passed as an input to next level SVM. Finally, a global SVM model is obtained by combining local SVM models. In [25], Z. Sun *et al.* implement cascade SVM with the help of MapReduce and Twister.

In [5], Hsieh *et al.* use k-means clustering for partitioning the dataset. In [26], [27], Herrero-Lopez *et al.* accelerate SVM training by integrating GPUs into MapReduce clusters. It distributes the matrix multiplications during the sequential update of the Lagrangian multipliers. However, it does not allow the desired level of acceleration due to the sequential nature of the SVM. In [29], Vazquez *et al.* propose a distributed SVM in which local support vectors (LSVs) are calculated on each subset. The set of global support vectors (GSVs) is the union of all the LSVs. Then the GSVs are merged with each training subset, and the process is repeated until convergence (i.e., no change in the empirical risk). However, the size of the subsets increases with the number of iterations which contributes to increased training time. Also, at each time, LSVs are collected from each node to form the GSVs and then these GSVs are broadcasted to all the nodes which further increases the communication overhead. This also results in high redundancy among LSVs across all the nodes. A similar approach has been proposed by Lu *et al.* [22] for strongly connected networks (SCNs). Catak *et al.* [23] proposed MapReduce-based implementation of the same methodology in the cloud environment to improve scalability and parallelism of training phase by splitting training dataset into smaller subsets.

On the other hand, the **subspace based approaches** also divide the large-scale SVM training problem into several smaller problems. These approaches splits the data space into several disjoint subspaces and kernel SVMs are trained on these subspaces, independently. However, these approaches do not involve combine step as each test point is mapped to a subspace and the prediction is made by the corresponding SVM model. Some of the well known state-of-the-art subspace based approaches are discussed below.

In [14], Chang *et al.* use a binary decision tree (C4.5) in order to split the training dataset into smaller subspaces. Where at a given node E, a certain feature f_E of the training samples at node E is compared with a certain value v_E so that all samples with $f_E < v_E$ are assigned to the left-hand child node, and the remaining samples are assigned to the right-hand child node. Finally, at each leaf nodes, kernel SVM models are trained independently. This approach accelerates the training and testing speed but suffers from high loss of accuracy because the splitting of the data space is merely based on one attribute while ignoring the spread of the data. In [30], Singh *et al.* use a similar approach for intrusion detection where decision tree (DT) is used to partition the entire dataset into smaller subsets. Then SVM models are trained on smaller subsets, which results in increased classification accuracy. Also, it reduces the training and testing time significantly. This method uses the domain-specific knowledge in the decision tree for partitioning the dataset. However, our proposed approach is generalized as it do not exploit any domain specific knowledge and thus applicable to all domains.

The existing partitioning based methods for training the SVM in a distributed system are not able to meet the desired level of the acceleration in training speed and resulting in high loss of accuracy. Also, the time taken in partitioning the dataset is considerably high. For example, kernel k-means clustering algorithm used for data partition in [5], [13] takes

$O(n^2d)$ time. As for large datasets where n is significantly high these methods also lead to high computational cost. Also, the methods using divide-and-conquer approach resulted into a single model which results in a large number of support vectors. The conquer step increases the time for training of model, and a large number of support vectors increase the prediction time. The cascade SVM [28] exchanges a significant amount of data during training which increases the communication overhead and also training time.

III. Details of Proposed Work

The proposed approach works in two steps: training and testing. In the first step, a decision tree is constructed using training data. The master node partitions the entire dataset into smaller subsets. For partitioning the dataset, it computes the dominant eigenvector of the entire dataset using an iterative procedure. The entire dataset is projected on the dominant eigenvector. The spread of the projection is partitioned into B bins, where B is the maximum number of branches at any node in the decision tree. A child node is created for each non-empty bin and the data of the bins is assigned to their respective child nodes. Similarly, at each child node, a subtree is constructed, recursively. The decision tree partitions the data at each node along the direction of maximum variance in the data as described in next section.

A. Subspace Partitioning using Decision Tree and Dominant Eigenvector

In this work, we partition the entire data space into smaller subspaces. Let $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, 2, ..., n\}$ be the entire dataset, where $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional data point with class label $y_i \in \{-1, +1\}$. The direction of the maximum variance is given by the dominant eigenvector of the dataset D. In theory, we can use any eigendecomposition method like singular value decomposition (SVD) or eigenvalue decomposition for this purpose. However, we have used iterative *power method* for computation of dominant eigenvector to achieve better computational and spatial efficiency. The computational complexity of SVD is $O(nd^2 + d^3)$, which is appropriate for computing all d eigenvectors. However, here our objective is to compute only dominant eigenvector so the suitable method for finding dominant eigenvector efficiently is *power method* with time complexity $O(nd^2)$. The power method begins with an initial vector $\mathbf{v}_0$ which has a non-zero component in the direction of the dominant eigenvector. Then dominant eigenvector $\mathbf{w}$ is given by following recurrence relation after t iterations:

$$\mathbf{w} = \mathbf{v}_t = \frac{\mathbf{\Sigma}\mathbf{v}_{t-1}}{||\mathbf{\Sigma}\mathbf{v}_{t-1}||}, \tag{3}$$

where $\mathbf{\Sigma}$ is the covariance matrix of the dataset $\mathbf{D}$ and is computed as

$$\mathbf{\Sigma} = cov(\mathbf{D}) = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T, \tag{4}$$

where $\boldsymbol{\mu}$, ($\boldsymbol{\mu} = \frac{1}{n}\sum_i^n \mathbf{x_i}$) is the means of the dataset $\mathbf{D}$.

Equation (3) is solved iteratively, multiplying $\mathbf{v}_{t-1}$ by $\mathbf{\Sigma}$ and then normalized. Initially, $\mathbf{v}_0$ is set to $\mathbf{e}$ (i.e. the vector with all its values set to one) which guarantees non-zero component in direction of dominant eigenvector.

Once the dominant vector $\mathbf{w}$ is computed, then the projection of a data point $\mathbf{x}_i$ on the $\mathbf{w}$ is given by

$$\hat{x}_i = \mathbf{w}^T\mathbf{x}_i. \tag{5}$$

Fig. 1 shows an example of the projection of all the points in $\mathbf{D}$ on the dominant eigenvector. The entire spread of the projection is partitioned into B bins. According to these bins, the dataset $\mathbf{D}$ is partitioned into B subsets. The following equation assigns the bin b for a data point $\mathbf{x}_i$:

$$b = \begin{cases} 1, if\ r \leq 0, \\ \lceil r \rceil \text{, otherwise,} \\ B, if\ r > B. \end{cases} \tag{6}$$

$$r = \frac{\hat{x}_i - \hat{x}_{min}}{\hat{x}_{max} - \hat{x}_{min}} \times B, \tag{7}$$

where, $\hat{x}_{min}$ and $\hat{x}_{max}$ are the minimum and maximum values of the spread of projection of data points on the dominant eigenvector, respectively.

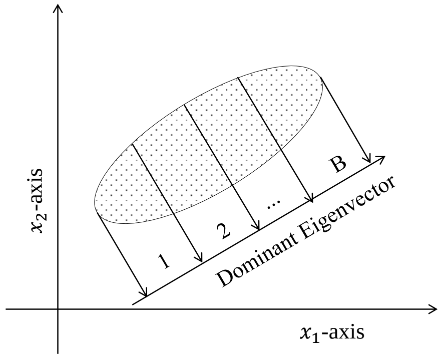


Fig. 1. Proposed approach for data partitioning using decision tree along the direction of maximum variability.

Finally, the entire dataset $\mathbf{D}$ is divided into B smaller datasets $\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_B$. Each dataset $\mathbf{D}_b$ contains n_b data points. If all the points at a node belong to the same class, then that node is declared as a leaf node labeled with the corresponding class. Otherwise, data space at a node is partitioned recursively until it reaches maximum level.

B. Training in Distributed Environment

The above partitioning method partitions the entire data space into subspaces. In the decision tree, for a leaf node following two scenarios can occur while partitioning i) all points in the subspace belong to the same class, or ii) subspace contains data points from both classes. As discussed earlier, node in case (i) is a leaf node with label same as data points. However, for case (ii), a kernel SVM model is trained using data points in that subspace. The smaller kernel SVM models on subspace data are independent and thus enable the proposed approach to be trained in a distributed system. Let $\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_P$ represent the data in P subspaces which have data points from both the classes. Then the master node sends the data of these subspaces to P compute nodes. Each node with identifier p trains a SVM model on its data using equation 1 as follows:

$$\min_{\boldsymbol{\alpha}_p} \frac{1}{2} \sum_{i=1}^{n_p} \sum_{j=1}^{n_p} \alpha_{p,i} \alpha_{p,j} y_{p,i} y_{p,j} K(\mathbf{x}_{p,i}, \mathbf{x}_{p,j}) - \sum_{i=1}^{n_p} \alpha_{p,i}, \quad (8)$$

where, $\boldsymbol{\alpha}_p$ is the set of Lagrangian multipliers for the data of p^{th} subspace. The final SVM model $\mathbf{SM}_p$ constitutes

$$\mathbf{SM}_p = \begin{cases} \mathbf{SV}_p = \mathbf{D}_p(\boldsymbol{\alpha}_p > 0) \\ \boldsymbol{\alpha}_p^* = \boldsymbol{\alpha}_p(\boldsymbol{\alpha}_p > 0), \end{cases} \quad (9)$$

where $\mathbf{SV}_p$ are the support vectors and $\boldsymbol{\alpha}_p^*$ are the corresponding non-zero Lagrangian multipliers. Once training is completed at a compute node, then it sends the trained SVM model $\mathbf{SM_p}$ back to the master node. The master node creates a leaf node in the decision tree at the respective branch which contains the returned SVM model $\mathbf{SM_p}$. Algorithm 1 gives the pseudo-code of complete procedures of data partitioning and distributed SVM training for the proposed distributed SVM approach. After successful training, the final tree model looks like a sample tree shown in the Fig. 2. A non-leaf node contains 1) dominant eigenvector $\mathbf{w}$, and 2) $\hat{x}_{min}$ and $\hat{x}_{max}$ are the minimum and maximum values of the spread of projection of data points on the dominant eigenvector. A leaf node contains either a class label or an SVM model.

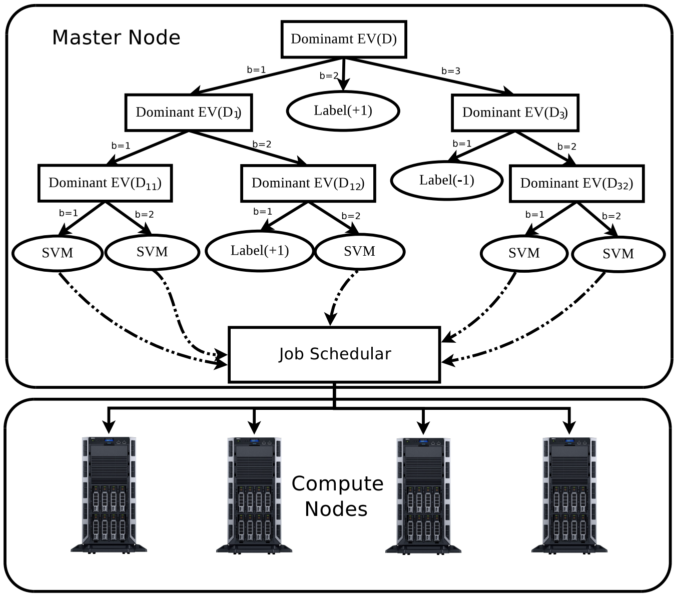


Fig. 2. Block diagram of the Projection-SVM training over the cluster. Master node contains a sample tree model. The job scheduler evenly distribute the task of training SVMs to compute nodes

C. Prediction using proposed Distributed SVM

In order to test an unknown data point on proposed distributed SVM, we traverse the decision tree from root to leaf; if leaf node has a class label, then this is the predicted label. If a leaf node has an SVM model, then the classification label is predicted using that SVM model.

Algorithm 1 Training of Proposed Distributed SVM

```
Input:
$\mathbf{D}$: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n,\ \mathbf{x}_i \in \mathbb{R}^d,\ y_i \in \{-1, +1\}$
$n$ : #data points in $\mathbf{D}$. $d$ : #dimensions
$B$ : #branches (max) at each internal node
$h$ : Maximum height of the tree
$\epsilon$ : tolerance for evaluating dominant eigenvector
$P$ : #partitions and also #node processors
Output:
$tree$: final tree model for prediction
$train\_SVM(\mathbf{D}, h)$
if $\forall y_i \in \mathbf{D}, y_i = 1$ then
    return $Leaf(1)$;
else if $\forall y_i \in \mathbf{D}, y_i = -1$ then
    return $Leaf(-1)$;
else if $h = 0 \;||\; n < min\_size$ then
    return $Leaf(svm\_train(\mathbf{D}))$;
else
    $\boldsymbol{\mu} = \frac{1}{n}\sum_i^n \mathbf{x_i}; \boldsymbol{\Sigma} = \frac{1}{n-1}\sum_{i=1}^n (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$;
    $t \leftarrow 0; \mathbf{v}_0 \leftarrow \mathbf{e}$;
    while $||\mathbf{v}_t - \mathbf{v}_{t-1}|| > \epsilon$ do
        $\mathbf{v}_t \leftarrow \boldsymbol{\Sigma}\mathbf{v}_{t-1}; t = t + 1$;
    end while
    $\mathbf{w} \leftarrow \mathbf{v}_t$;
    $\mathbf{D}_b \leftarrow \phi, b = 1, 2, ..., B$
    $\hat{x}_i \leftarrow \mathbf{w}^T\mathbf{x}_i, i = 1, 2, ..., n$;
    $\mathbf{x}_{min} \leftarrow min(\hat{x}_i); \mathbf{x}_{max} \leftarrow max(\hat{x}_i)$
    $tree \leftarrow Node(\mathbf{w}, \hat{x}_{min}, \hat{x}_{max})$
    for $i = 1, 2, ..., n$ do
        $b \leftarrow min\Big(max\Big(\Big\lceil \frac{\hat{x}_i - \hat{x}_{min}}{\hat{x}_{max} - \hat{x}_{min}} \times B \Big\rceil, 1\Big), B\Big)$;
        $\mathbf{D}_b \leftarrow \mathbf{D}_b \cup (\mathbf{x}_i, y_i)$;
    end for
    for $b = 1, 2, ..., B$ {In parallel} do
        $tree.child_b \leftarrow train\_SVM(\mathbf{D}_b, h - 1)$;
    end for
    return $tree$;
end if
```

Let $\mathbf{x} \in \mathbb{R}^d$ be a test data point, and $tree$ be the trained model. When traversing the $tree$, at any node, there are three possibilities:

1) *Internal Node:* If the current node is an internal node, then based on the parameters ($\mathbf{w}, \hat{x}_{min}, \hat{x}_{max}$), it computes the bin index using Equations (5),(6) & (7) and selects the corresponding branch. According to the selected branch, it visits a child node, and this procedure is continued until it reaches a leaf node.
2) *Leaf Node with Class Label:* If the current node is a leaf node with a class label, then it assigns the class label of the leaf node as the predicted class of the test point $\mathbf{x}$ and the procedure is terminated.

3) *Leaf Node with SVM Model:* If the current node is a leaf node with a trained SVM model $\mathbf{SM}$, then it predicts the class of the test point $\mathbf{x}$ using equation 2.

Algorithm 2 gives the pseudo-code of complete procedure of prediction using proposed distributed SVM.

Algorithm 2 Prediction using Proposed Distributed SVM

```
Input:
$\mathbf{x} \in \mathbb{R}^d$ :unlabeled data point
$d$ : #dimensions
$B$ : #branches (max) at each internal node
$tree$: trained tree model

Output:
$y$: predicted label for $\mathbf{x}$

$predict\_SVM(tree, \mathbf{x})$
if $isLabel(tree) = true$ then
    $y \leftarrow tree.ClassLabel$;
else if $isSVM(tree) = true$ then
    $svm\_model \leftarrow tree.svm\_model$;
    $y \leftarrow f(x)$;{using equation 2}
else
    $\mathbf{w} \leftarrow tree.\mathbf{w}$;
    $\mathbf{x}_{min} \leftarrow tree.\mathbf{x}_{min}$;
    $\mathbf{x}_{max} \leftarrow tree.\mathbf{x}_{max}$;
    $\hat{x} \leftarrow \mathbf{w}^T\mathbf{x}$
    $b \leftarrow min\Big(max\Big(\Big\lceil \frac{\hat{x}-\hat{x}_{min}}{\hat{x}_{max}-\hat{x}_{min}} \times B\Big\rceil, 1\Big), B\Big)$;
    $y \leftarrow predict\_SVM(tree.child_b, \mathbf{x})$;
end if
return $y$;
```

D. Time Complexity Analysis

The time taken in partitioning at a node includes the time taken in computing the dominant eigenvector, projecting data points on the dominant eigenvector, and partitioning data to each branch, i.e.

$$T_{partition} = O(nd^2 + nd + n) \approx O(nd^2). \tag{10}$$

For sequential decision tree construction, the total time is

$$T_{tree} = \sum_h \sum_{b=1}^{B^h} \frac{n}{B^h} d^2 = O(nd^2 h). \tag{11}$$

However, for parallel construction of decision tree, the total time is

$$T_{tree} = \sum_h \frac{n}{B^h} d^2 = nd^2 \sum_h \frac{1}{B^h} \approx O(nd^2), \tag{12}$$

since, $1 \leq \sum_h \frac{1}{B^h} \leq 2$.

The *best case* for the proposed approach occurs when the data on all children nodes after partitioning belong to one class only as shown in Fig. 3-(B). Thus in best case the training time includes only partitioning time as no SVM is trained. The *average case* is when the decision tree makes the balanced partitions, and each class contains data points from both the classes. Then for height h and maximum number of branches B, the scaling factor (SF) is

$$SF = \frac{n^3}{\left(\frac{n}{B^h}\right)^3} = B^{3h}. \tag{13}$$

The *worst case* corresponds to highly imbalanced partitioning here the scaling factor depends on n_{max}, the size of biggest partition among all partitions at height h, i.e.

$$SF = \left(\frac{n}{n_{max}}\right)^3. \tag{14}$$

IV. Experiments and Results

The proposed distributed SVM is implemented in C++ using *libsvm* [8], *armadillo* [31], *openmp* and *openmpi* which are the de facto standards for scientific computing over the high-performance cluster (HPC). The experiments are conducted over a cluster having:

- master node (with configuration as Intel Xeon(R) 2.70GHz×48 processor and 128GB memory), and
- 100 compute nodes (with configuration as Intel Xeon(R) 2.70GHz×8 processor and 12GB memory each).

In the proposed approach, the master node is responsible for data partitioning and maintaining the tree hierarchy. The number of partitions (P) are determined by $P = B^h$ where B is the number of branches and h is the height of the tree. The values of B and h depend on the size of each dataset. The details of the datasets are listed in Table I. Also, the details of the hyper-parameters (C,γ) for SVM along with values of B and h are given in Table III. Once the data is partitioned into P sub-problems, they are distributed across the m compute nodes.

A. Sketches of Correctness

1) Case-1: Each Class as a Single Gaussian Distribution: If both classes are well separable, and projection of points on the dominant eigenvector is producing two non-overlapping spreads for both classes as shown in Fig. 3(A), then it splits the data into disjoint partitions containing data points from either class. In this case, no SVM is trained as the data on each child node belongs to the same class as shown in Fig. 3(B) for $B = 2$. The overall classification is similar to Fisher's linear discriminant analysis (LDA). However, if the projection of the two separable classes is slightly overlapping as shown in Fig. 3(C), then for each overlapping bins, it trains SVM as shown in Fig. 3(D). In such cases, the overlapping bins contain data points of different classes which are relatively close to each other. Now, the SVM model is trained with data points which are more likely to be the support vectors resulting in a faster training of SVM. If the projection of the points in both the classes on the dominant eigenvector produces two completely overlapping spreads as shown in Fig. 3(E) & Fig. 3(G), then each bin will contain points from both the classes. If classes are separable, then at some height, a decision tree can discriminate the points of the two classes. For example, the tree shown in Fig. 3(F) does it at $h = 2$ for the case shown in Fig. 3(E). However, if classes are non-separable as shown in Fig. 3(G), then it needs to train an SVM model for each bin as shown in Fig. 3(H).

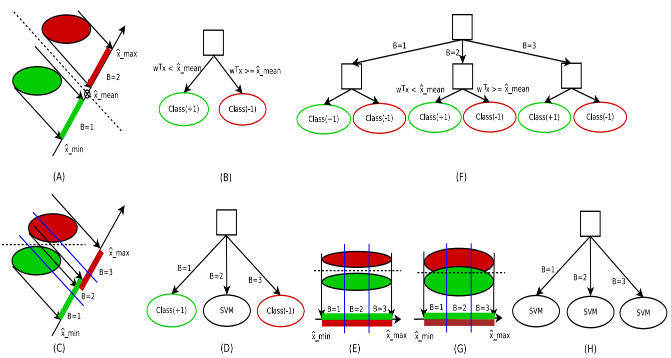


Fig. 3. Illustration of the working of proposed distributed SVM for well separable classes. (A) Two separable classes and their non-overlapping projections on the dominant eigenvector for the combined data of both the classes. (B) The corresponding tree for (A) contains two leaf nodes with each class label. (C) Two separable classes and their overlapping projections on the dominant eigenvector for the combined data of both the classes. (D) The corresponding tree for (C) contains two leaf nodes with each class label and a leaf node with SVM model for the overlapping region (B=2). (E) Two separable classes and their completely overlapping projections on the dominant eigenvector for the combined data of both the classes. (F) The corresponding tree for (E) can discriminate the two classes at height h=2. (G) Two overlapping classes and their completely overlapping projections on the dominant eigenvector for the combined data of both the classes. (H) The corresponding tree for (G) can discriminate the two classes using SVM at height h=2.

2) Case-2: Each Class as a Mixture of Gaussian Distributions: Let's consider a complex case where each class is composed of several distributions spread non-uniformly in the space as shown in Fig. 4(A). In the first step, the proposed SVM splits the complex set of data points into smaller subsets which are relatively less complex as shown in Fig. 4(C). Here, the classification is relatively easier, faster, and may lead to good performance because it focuses on the local data points only. However, it is also possible that local boundaries are less regularized in comparison to global decision boundary. Fig. 4(B)&(F)-(G) show the local decision boundaries corresponding to each subset. The experiments on randomly generated data points show $2\% - 10\%$ improvement in the classification performance in comparison to LIBSVM.

First, we conducted experiments on the synthetic data to show that the proposed method works well for the data having complex distributions. For this, we generated a mixture of K-Gaussian distributions, where $K = 10, 20, 30, 40, 50, 60$. The labels to each Gaussian distribution are assigned $+1$ or -1 randomly. One such data which is a mixture of 50 Gaussian distributions is shown in Fig. 5. In the data, both the classes are spread over the entire space with significant overlap of positive and negative examples. Fig. 6 showed the comparisons of the classification performance for sequential SVM and proposed method on the synthetic datasets. The proposed approach performs similar to sequential SVM for the low values of K, but for the high values of K, it achieved much better performance than sequential SVM. Because finding a single separating hyperplane using SVM for such a complex data distribution is very hard. However, proposed distributed SVM converts a complex large problem into multiple simple smaller problems and then solves them independently.

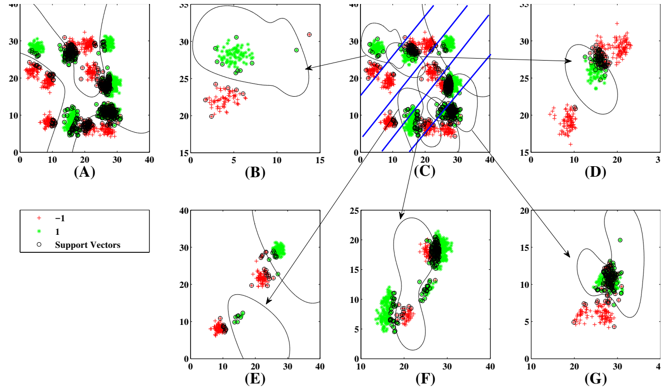


Fig. 4. A comparison of the sequential SVM and the proposed distributed SVM on a sample 2D-data which is a mixture of the 20-Gaussian distributions. (A) Decision boundary for a sequential SVM using RBF kernel on all data points. (C) Shows the partitioning lines and the decision boundary of all subsets. (B)&(D)-(G) each show the decision boundary of a respective subset.

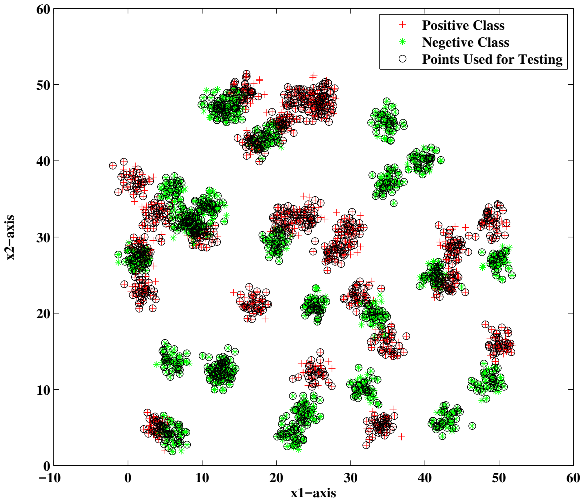


Fig. 5. A sample data used to validate the proposed SVM. Data is a mixture of 50-Gaussian distributions. Class labels are assigned randomly. The data used is the mixture of K-Gaussian distributions. $K = 10, 20, 30, 40, 50, 60$, Number of data points $n = 2400$.

B. Parameter Selection

The selection of parameters for training a SVM is based on grid search for parameters $\gamma = [2^{-10}, 2^{-10}, ..., 2^{10}]$ and $C = [2^{-10}, 2^{-10}, ..., 2^{10}]$. The parameters for each local SVM can be tuned independently. However, for the proposed distributed SVM approach, the classification accuracy and training time not only depend on the hyper-parameters of the SVM but also depend on the number of branches at each

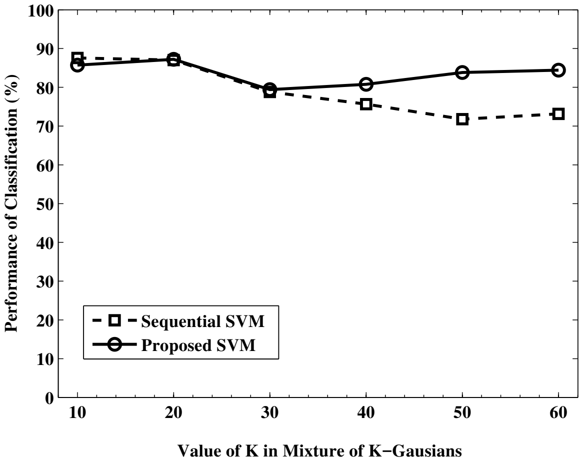


Fig. 6. A comparison of the classification performance (%) for sequential SVM and proposed distributed SVM. The data used is the mixture of K-Gaussian distributions. $K = 10, 20, 30, 40, 50, 60$, Number of data points $n = 2400$.

non-leaf node (i.e. B) and the maximum height of the tree (i.e. h). If for some dataset, the projection for n data points on the dominant eigenvector is uniformly distributed, then the relation of the height h and branches B can be given by $h = log_B n$ and the number of resulted splits will be $\leq B^h$. Thus, a large number of branches result in a low height which further reduces the testing time. However, the classification accuracy depends highly on the distribution of the data. Fig. 7 & Fig. 8 give the experimental results on the *ijcnn1* dataset. The results show that the performance of classification for five distinct values of the $B = 2, 4, 6, 8, 10$ with maximum height $h = 0, 1, 2, 3, 4, 5$. Our experiments demonstrate that a large number of branches leads to the high acceleration in the training time in comparison to a small number of branches. However, a small number of branches preserve the classification accuracy in comparison to a large number of branches.

C. Comparison with state-of-the-art methods

In order to evaluate the performance of the proposed distributed SVM, experiments are conducted on various large scale high dimensional datasets from different application domains. The datasets used for benchmarking are publicly available at [32], [33]. The details of these datasets are listed in Table I.

A comparison of DC-SVM and proposed distributed SVM approach at various size levels of the adult dataset in Fig. 9 show that the proposed approach is approximately 10 times faster than the DC-SVM method while producing the comparable classification accuracy. The scalability of the proposed distributed SVM approach increases with an increase in the size of the dataset rapidly.

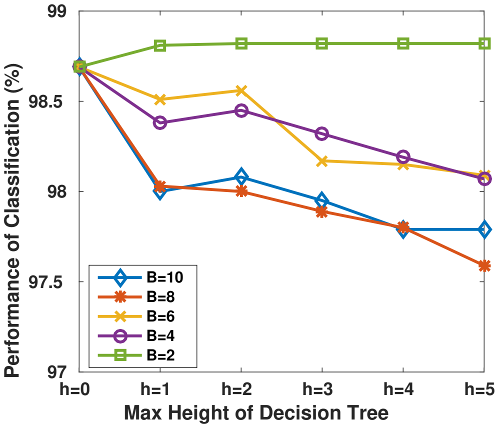


Fig. 7. Comparison of classification performance for various combinations of height h of decision tree and number of branches B for *ijcnn1* dataset.

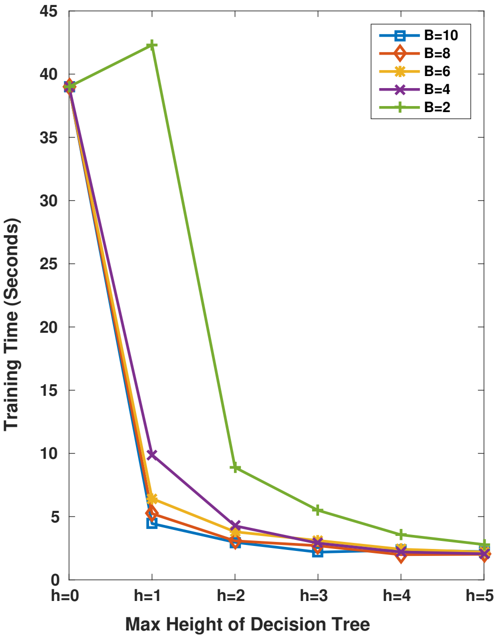


Fig. 8. Comparison of training time for various combinations of height h of decision tree and number of branches B for *ijcnn1* dataset.

TABLE I
DETAILS OF THE DATASETS USED

Dataset	Application Domain	#Dim.	#Train	#Test
gisette [13]	Digit Classification	5,000	6,000	1,000
adult [13]	Economics	123	32,561	16,281
ijcnn1 [13]	Text Classification	22	49,990	91,000
cifar [5]	Visual Recognition	3,072	50,000	10,000
webspam [5]	Spam Detection	254	280,000	70,000
covtype [5]	Forest Classification	54	464,810	116,202
kddcup99 [5]	Intrusion Detection	123	4,898,431	311,029
mnist8m [5]	Digit Classification	784	8,000,000	100,000

The loss of classification accuracy with respect to sequential SVM as well as the existing distributed SVMs is used for comparison of performance. Table II shows the performance

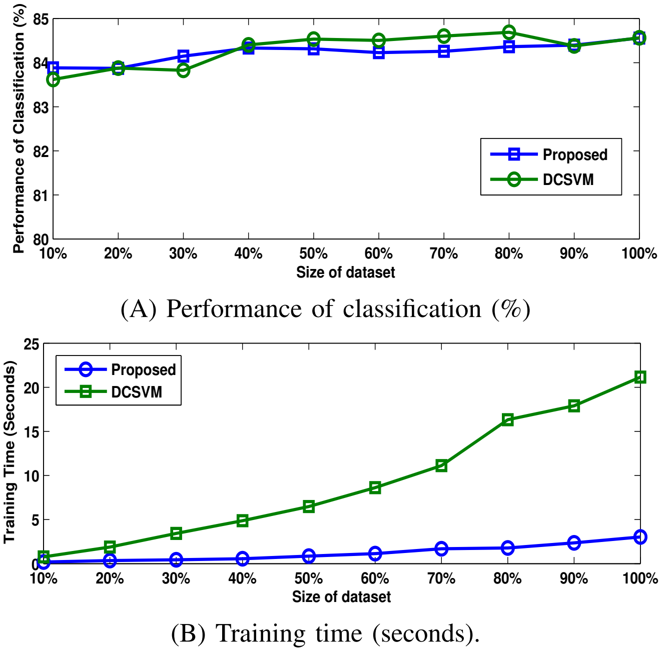
TABLE II
PERFORMANCE OF CLASSIFICATION (%) OF PROPOSED DISTRIBUTED SVM AND COMPARISON WITH LIBSVM, DC-SVM, CA-SVM AND DT-SVM.

Method	**LIBSVM**		**DC-SVM**		**CA-SVM**		**DT-SVM**		**Proposed**		**Change**	
Dataset	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Scale
gisette	97.70	125	97.60	299	96	81	97.60	355	97.50	**43**	−0.20	3×
adult	85.08	761	84.79	78	83	121	84.79	45	84.46	**9**	−0.62	85×
ijcnn1	98.69	20	98.53	318	90.16	121	94.33	27	98.82	**1**	+**0.13**	20×
cifar	89.50	13892	80.15	22330	63.94	2143	75.82	540	87.11	**193**	−2.39	72×
webspam	99.28	15056	99.28	10485	99.11	3093	NA	NA	98.81	**28**	−0.47	538×
covtype	96.01	31785	95.95	17456	75.04	34025	NA	NA	95.77	**119**	−0.24	267×
kddcup99	99.57	37684	99.49	23346	NA	NA	NA	NA	99.02	**40**	−0.55	942×
mnist8m	99.91	≈13 d	99.91*	NA	NA	NA	NA	NA	99.77	**6786**	−0.14	166×

NA - *Particular method is unable to calculate, *taken from [5]

TABLE III
VARIOUS EVALUATION METRICS FOR EFFECTIVENESS & EFFICIENCY OF THE PROPOSED DISTRIBUTED SVM.

Dataset	C	γ	B	h	Precision (%)	Recall (%)	F-measure (%)	Kappa Index (-1,1)	Partition Time (Seconds)	Train Time (Seconds)	Test Time (Seconds)
gisette	1	2e-4	2	1	97.03	98.00	97.51	0.9500	0.941	42.179	6.865
adult	32	2^{-7}	2	2	87.00	93.65	90.20	0.5292	0.019	3.981	7.000
ijcnn1	32	2	2	5	94.69	92.80	93.74	0.9309	0.073	0.609	0.356
cifar	8	2^{-22}	2	4	88.16	90.70	89.41	0.7295	10.299	183.190	111.271
webspam	8	32	2	10	99.45	98.59	99.01	0.9751	5.490	22.083	12.231
covtype	32	32	2	10	96.23	95.53	95.88	0.9153	3.707	115.202	4.375
kddcup99	256	0.5	2	10	97.13	98.43	97.78	0.9715	23.952	15.638	1.520
mnist8m	1	2^{-21}	2	10	99.74	99.80	99.77	0.9954	1184.090	5601.990	123.561



(A) Performance of classification (%)

(B) Training time (seconds).

Fig. 9. A level wise comparison of DC-SVM and proposed distributed SVM on adult dataset.

of the classification and training time on datasets listed in Table I. The details of various evaluation metrics used for evaluation of the proposed approach is given in Table III. The proposed distributed SVM approach reduces the loss in the classification accuracy, and the results are approximately equal to the results of the sequential SVM. On all the datasets considered for evaluation of the proposed approach achieves the least drop in the classification accuracy among existing approach (i.e. DCSVM [5], CASVM [13], and DTSVM [14]) as compared to sequential SVM as shown in Fig. 10. One possible reason for this reduction in the loss of classification

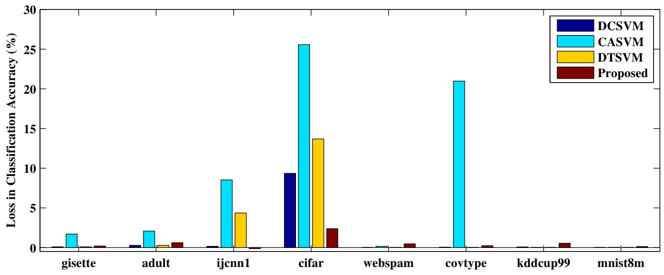


Fig. 10. A comparison of the loss in classification accuracy (%) of DCSVM, CASVM, DTSVM, and proposed distributed SVM with respect to LIBSVM on publicly available datasets.

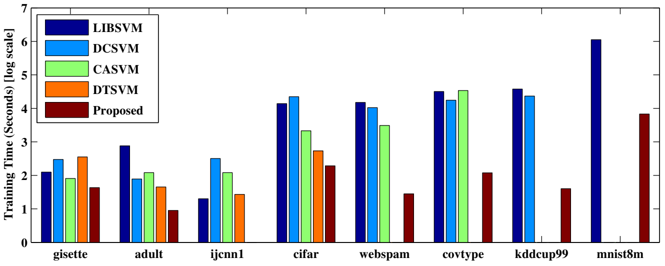


Fig. 11. A comparison of the training time (seconds) for LIBSVM, DCSVM, CASVM, DTSVM, and proposed distributed SVM on publicly available datasets.

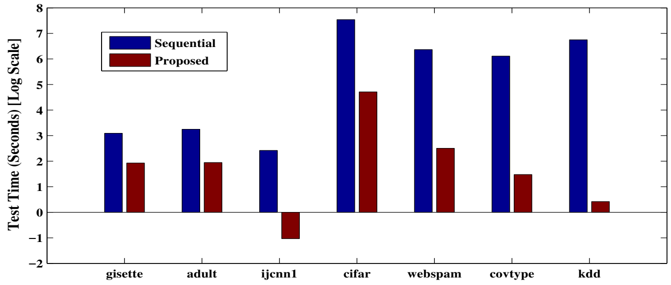


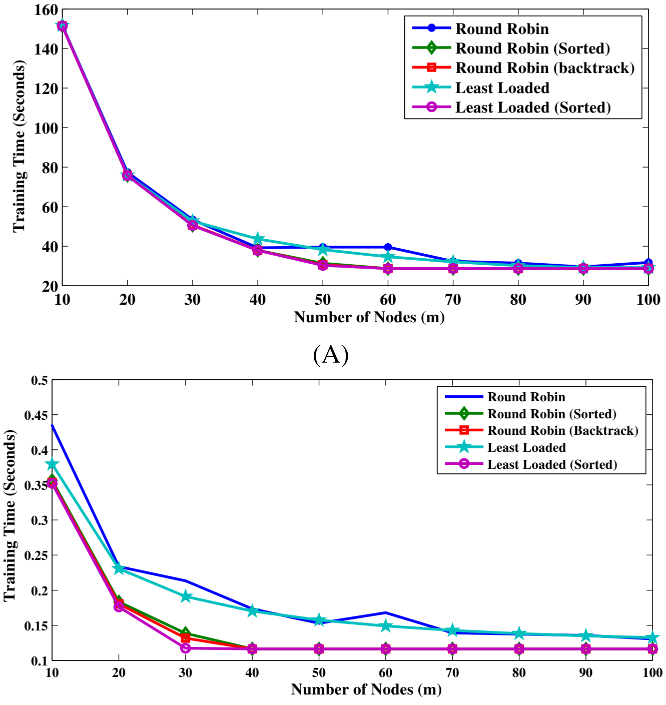
Fig. 12. Comparison of the test time of sequential SVM and the proposed approach

accuracy is that the proposed approach finds the decision boundary in the smaller subspaces only, which may help in better classification within the subspaces. The decision tree splits the large complex problem into smaller simple problems which are then solved with more precision.

The proposed SVM approach reduces the training time as well as testing time significantly as shown in Fig. 11 and Fig. 12, respectively. The reduction in training time can be attributed to the distributed training of smaller independent SVMs. The proposed approach uses decision tree for partitioning of the dataset which is computationally less expensive as compared to kernel-clustering approach for very large datasets. Use of the dominant eigenvector efficiently divides the entire space along the direction of maximum variance. This partitioning leads to the reduction in the variance of the data points in subspace. As there is no conquer step, so it further reduces the training time. This approach also reduces the communication overhead significantly as it does not send the data from one level to another after training as required in [5]. The proposed approach needs to communicate twice, once to send data from the master node to worker nodes and later to receive model parameters back from worker nodes to master node.

Finally, the tree model predicts the label at leaf nodes using leaf label or trained SVM. There are three types of nodes in tree model: 1) Internal node which decides to which subspace the test data point belongs. 2) Leaf node with a class label which directly predicts the classification label for the test data point without any computation. 3) Leaf node with SVM model which uses the trained SVM model to predict classification label. As these SVMs are trained on smaller sub-datasets which generally will contain less number of support vectors with respect to global SVM.

To balance the load, we used five load balancing algorithms, namely, 1) *round robin* - Here the jobs are assigned sequentially in each round, 2) *round robin (sorted)* - It assigns jobs based on the estimated run time in each round, 3) *round robin (backtrack)* - This is similar to round robin (sorted) but after every round, it reverses the direction of assignment, 4) *least loaded* - It picks a random job and assigns it to the least loaded node, and 5) *least loaded (sorted)* This is same as least loaded but assigns the longest job first. Fig. 13 (A) & (B) show the scaling behaviour of the proposed approach while using various load balancing strategies for $mnist8m$, and $kddcup99$ datasets, respectivly. It can be observed from the figures that the least loaded (sorted) performs better than the others on both the datasets. Also, the proposed approach scales well with an increase in the number of compute nodes. However, after a certain number of nodes, there is no significant change in scaling of the algorithm which is entirely depends on the size of data and the complexity of the classification problem.



(A)

(B)

Fig. 13. Scaling behaviour of the proposed approach on (A) $mnist8m$, and (B) $kddcup99$ datasets using various load balancing strategies. The proposed approach scales well with an increase in the number of compute nodes (m). The least loaded on sorted sequence of the jobs performs better than the others.

V. Conclusion

In this work, a distributed SVM for big data using decision tree and dominant eigenvector is proposed. This distributed SVM approach trains the model faster and requires less time in prediction for new data points. The use of dominant eigenvector and decision tree for partitioning of the dataset is also computationally less expensive with a complexity of $O(nd^2)$ in comparison to kernel k-means approach with a complexity of $O(n^2d)$ as proposed in [5] [13]. The proposed approach also achieves good classification performance with a small change in accuracy. The experimental results on eight standard datasets confirm that the proposed approach is on an average ≈ 150 times faster than sequential SVM and $\approx 10-50$ times faster than other existing distributed SVM approaches, namely, DC-SVM, CA-SVM, and DTSVM while sustaining the accuracy.

References

[1] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[2] D. Singh and C. K. Mohan, “Graph formulation of video activities for abnormal activity recognition,” *Pattern Recognition*, vol. 65, pp. 265–272, 2017.

[3] N. K. Alham, M. Li, Y. Liu, and S. Hammoud, “A MapReduce-based distributed SVM algorithm for automatic image annotation,” *Computers and Mathematics with Applications*, vol. 62, no. 7, pp. 2801–2811, 2011.

[4] D. Singh and C. K. Mohan, “Deep Spatio-Temporal Representation for Detection of Road Accidents Using Stacked Autoencoder,” *IEEE Trans. Intelligent Transportation Systems*, pp. 1–9, 2018.

[5] C.-J. Hsieh, S. Si, and I. Dhillon, “A Divide-and-Conquer Solver for Kernel Support Vector Machines,” in *Proc. of Int. Conf. Machine Learning (ICML)*, Beijing, 21-26 Jun 2014, pp. 566–574.

[6] D. Singh, D. Roy, and C. K. Mohan, “DiP-SVM : Distribution Preserving Kernel Support Vector Machine for Big Data,” *IEEE Trans. Big Data*, vol. 3, no. 1, pp. 79–90, 2017.

[7] Y. Yu, K. I. Diamantaras, T. McKelvey, and S. Y. Kung, “Class-Specific Subspace Kernel Representations and Adaptive Margin Slack Minimization for Large Scale Classification,” *IEEE Trans. Neural Networks and Learning Systems*, vol. 29, no. 2, pp. 440–456, 2018.

[8] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Trans. Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, 2011.

[9] J. A. K. Suykens and J. Vandewalle, “Least Squares Support Vector Machine Classifiers,” *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[10] T. Joachims, “Advances in Kernel Methods,” B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA, USA: MIT Press, 1999, ch. Making Large-scale Support Vector Machine Learning Practical, pp. 169–184.

[11] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, “Core Vector Machines: Fast SVM Training on Very Large Data Sets,” *J. Machine Learning Research*, vol. 33, no. 2, pp. 211–220, 2008.

[12] J. C. Platt, “Fast Training of Support Vector Machines Using Sequential Minimal Optimization,” in *Advances in Kernel Methods*, Cambridge, MA, 1999, pp. 185–208.

[13] Y. You, J. Demmel, K. Czechowski, L. Song, and R. Vuduc, “Design and Implementation of a Communication-Optimal Classifier for Distributed Kernel Support Vector Machines,” *IEEE Trans. Parallel and Distributed Systems*, vol. 28, no. 4, pp. 974–988, 2017.

[14] F. Chang, C.-Y. Guo, X.-R. Lin, C.-C. Liu, and C.-J. Lu, “Tree Decomposition for Large-Scale SVM Problems,” *J. Machine Learning Research*, vol. 11, pp. 2935–2972, 2010.

[15] Z. Lu, J. Sun, and K. Butts, “Multiscale Support Vector Learning With Projection Operator Wavelet Kernel for Nonlinear Dynamical System Identification,” *IEEE Trans. Neural Networks and Learning Systems*, vol. 28, no. 1, pp. 231–243, 2017.

[16] C. Kyrkou, C. S. Bouganis, T. Theocharides, and M. M. Polycarpou, “Embedded Hardware-Efficient Real-Time Classification With Cascade Support Vector Machines,” *IEEE Trans. Neural Networks and Learning Systems*, vol. 27, no. 1, pp. 99–112, 2016.

[17] D. Singh and C. K. Mohan, “Distributed Quadratic Programming Solver for Kernel SVM using Genetic Algorithm,” in *IEEE Congress on Evolutionary Computation (CEC)*, Vancouver, Canada, July 24–29 2016, pp. 152–159.

[18] X. Wang and S. Matwin, “A Distributed Instance-weighted SVM Algorithm on Large-scale Imbalanced Datasets,” in *Proc. of IEEE Int. Conf. Big Data*, Washington, 27-30 Oct 2014, pp. 45–51.

[19] M. Papadonikolakis and C. S. Bouganis, “Novel Cascade FPGA Accelerator for Support Vector Machines Classification,” *IEEE Trans. Neural Networks and Learning Systems*, vol. 23, no. 7, pp. 1040–1052, 2012.

[20] L. Zanni, T. Serafini, and G. Zanghirati, “Parallel software for training large scale support vector machines on multiprocessor systems,” *J. Machine Learning Research*, vol. 7, no. 7, pp. 1467–1492, 2006.

[21] A. Bordes, L. Bottou, and P. Gallinari, “SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent,” *J. of Machine Learning Research*, vol. 10, no. 7, pp. 1737–1754, 2009.

[22] Y. Lu, V. Roychowdhury, and L. Vandenberghe, “Distributed Parallel Support Vector Machine in Strongly Connected Networks,” *IEEE Trans. Neural Networks*, vol. 19, no. 7, pp. 1167–1178, 2008.

[23] F. O. Catak and M. E. Balaban, “CloudSVM : Training an SVM Classifier in Cloud Computing Systems,” in *Pervasive Computing and the Networked World*, Istanbul, Turkey, 28–30 Nov 2013, pp. 57–68.

[24] K. Xu, C. Wen, Q. Yuan, X. He, and J. Tie, “A MapReduce based Parallel SVM for Email Classification,” *J. Networks*, vol. 9, no. 6, pp. 1640–1647, 2014.

[25] Z. Sun and G. Fox, “Study on Parallel SVM Based on MapReduce,” in *Proc. of Int. Conf. Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, USA, 16–19 Jul 2012.

[26] S. Herrero-Lopez, “Accelerating SVMs by integrating GPUs into MapReduce clusters,” in *Proc. of IEEE Int. Conf. Systems, Man and Cybernetics*, Anchorage, AK, 9-12 Oct 2011, pp. 1298–1305.

[27] S. Herrero-lopez, J. R. Williams, and A. Sanchez, “Parallel multiclass classification using SVMs on GPUs,” in *Proc. of Workshop on General-Purpose Computation on Graphics Processing Units - GPGPU ’10*, Pittsburgh, PA, USA, 14 Mar 2010, p. 2.

[28] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, “Parallel Support Vector Machines: The Cascade SVM,” in *Proc. of Advances in Neural Information Processing Systems (NIPS)*, vol. 17, 2005, pp. 521–528.

[29] A. Navia-Vázquez, D. Gutiérrez-González, E. Parrado-Hernández, and J. J. Navarro-Abellán, “Distributed support vector machines,” *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 1091–1097, 2006.

[30] D. Singh, D. Patel, B. Borisaniya, and C. Modi, “Collaborative IDS Framework for Cloud,” *Int. J. Network Security (IJNS)*, vol. 18, no. 4, pp. 699–709, 2016.

[31] C. Sanderson and R. Curtin, “Armadillo: a template-based C++ library for linear algebra,” *J. Open Source Software*, vol. 1, pp. 1–26, 2016.

[32] “cifar,” https://www.cs.toronto.edu/ kriz/cifar.html.

[33] “DataSets,” available at https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/.