DiP-SVM : Distribution Preserving Kernel Support Vector Machine for Big Data

Dinesh Singh, *Student Member, IEEE*, Debaditya Roy, *Student Member, IEEE* and C. Krishna Mohan, *Member, IEEE*

Abstract—In literature, the task of learning a support vector machine for large datasets has been performed by splitting the dataset into manageable sized "partitions" and training a sequential support vector machine on each of these partitions separately to obtain local support vectors. However, this process invariably leads to the loss in classification accuracy as global support vectors may not have been chosen as local support vectors in their respective partitions. We hypothesize that retaining the original distribution of the dataset in each of the partitions can help solve this issue. Hence, we present DiP-SVM, a distribution preserving kernel support vector machine where the first and second order statistics of the entire dataset are retained in each of the partitions. This helps in obtaining local decision boundaries which are in agreement with the global decision boundary, thereby reducing the chance of missing important global support vectors. We show that DiP-SVM achieves a minimal loss in classification accuracy among other distributed support vector machine techniques on several benchmark datasets. We further demonstrate that our approach reduces communication overhead between partitions leading to faster execution on large datasets and making it suitable for implementation in cloud environments.

Index Terms—Distributed SVM, Big Data, Distribution Preserving Partitioning, Distributed Computing, Clustering.

1 Introduction

SUPPORT vector machine is based on the statistical learning theory developed by Vapnik et al. [1]. The core of training a support vector machine (SVM) involves solving a quadratic programming problem which demands more computing power for large datasets. Quadratic programming is a type of mathematical optimization problem which optimizes a quadratic function of several variables subject to linear constraints on these variables. The quadratic programming problem solver separates support vectors from the rest of the training data. For moderately sized datasets, support vector machine (SVM) has long been used extensively for classification and regression problems in many areas like genomics, e-commerce, computer vision, cyber security, etc. due to its generalization capabilities. However, classification of high volume data in the form of text, images or videos into meaningful classes using support vector machine is quite challenging [2]. Various implementations of SVM are available such as LIBSVM [3], LS-SVM, SVMlight [4] and so on. LIBSVM is the most popular among them because it utilizes a highly optimized quadratic solver. However, training a kernel SVM is still difficult for large datasets where the sample size reaches more than one million instances. The bottleneck stems from the high computational cost and memory requirements of computing and storing the kernel matrix, which in general is not sparse. The time complexity for standard SVM training is $O(n^3)$ and the space complexity is $O(n^2)$, where n is the size of training dataset [5]. This calls for a distributed approach for applying support vector machine for large datasets.

Distributed environments like a high-performance computing [6] and cloud clusters [7] are widely used for solving data-intensive and time-consuming problems. However, sequential minimal optimization (SMO) [8], the most successful quadratic programming (QP) solver used extensively in SVM implementations cannot leverage the benefits of these distributed environments because there is high dependency among the parameters used for optimization. So far, there is no true parallel or distributed algorithm in literature that solves the constrained quadratic programming problem used to identify the support vectors in the training data. The existing state-of-the-art approaches for approximate distribution of SVM training suffer from significant loss of accuracy mainly because smaller partitions do not retain the distribution of the entire dataset, which results in local separating hyperplanes that may be completely contradictory to the global separating hyperplane. Further, few of the existing approached [9] do not rely on local support vectors and instead transfer all the data points to next level which leads to very high communication overhead. The local support vectors (LSVs) are the support vectors resulting from individual SVM training over the smaller partitions (i.e support vectors from local SVM models), while global support vectors (GSVs) are the support vectors of final SVM model obtained by training over assembled local support vectors.

We hypothesize that the twin issues of loss of accuracy and high communication overhead can be effectively addressed using a distribution preserving partitioning approach which retains the first and second order statistics (mean and variance) of the entire data in each of the partitions. To achieve this goal, we propose DiP-SVM, an efficiently distributed SVM approach for big data which is

• *D. Singh, D. Roy and C. Krishna Mohan are with the Visual Learning & Intelligence Group (VIGIL), Department of Computer Science and Engineering, Indian Institute of Technology, Hyderabad, TS, 502205 India e-mail: (cs14resch11003@iith.ac.in, cs13p1001@iith.ac.in, ckm@iith.ac.in).*

also well suited for training in the cloud environment due to its low communication overhead. Our approach emphasizes on the twin goals of minimal loss of classification accuracy and low communication overhead. At first, distribution preserving data partitioning approach is used to split the data in such a way that each subset is a sparse (i.e. distribution preserving) representation of the entire dataset. As these are reasonably sized partitions, sequential SVMs are trained on these partitions over multiple virtual machines (VMs) using enhanced version of the algorithms used in [9], [10], [11], [12]. We demonstrate that this approach reduces the loss of accuracy and produces only relevant local support vectors which are in agreement with the global separating hyperplane. Further, transferring only these relevant points to next level reduces the communication overhead on several large datasets like kddcup99, MNIST8M, etc.

The rest of the paper is organized as follows. Section 2 presents related work. The details of DiP-SVM are presented in section 3. Section 4 discusses the experimental setup and results. We conclude in section 5 with future directions.

2 Related Work

Many parallel and distributed implementations of SVM have been proposed in the literature [2], [13]. Broadly, we can group them into four categories, namely, parallel [14], distributed [15], [16], heterogeneous (MapReduce based) [10], [11], [17], [18], [19], [20], and GPU based [21]. In [22], Vazquez *et al.* propose a distributed support vector machine 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 learning 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. Similar approaches have been proposed by Lu *et al.* [16] for strongly connected networks (SCNs). Catak *et al.* [17] proposed a MapReduce-based implementation of the same methodology in the cloud environment in order to improve scalability and parallelism of training phase by splitting training dataset into smaller subsets as shown in Fig. 1.

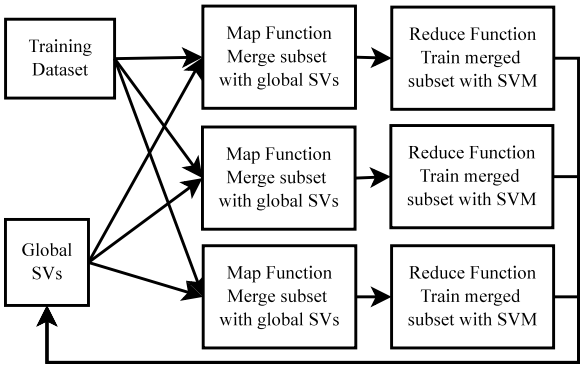


Fig. 1. Schematic of Cloud SVM architecture [17].

In [23], Graf *et al.* proposed cascade SVM, where the training samples are divided hierarchically into subsets and the training starts at the top nodes where each subset is then trained a sequential SVM (referred to as a subSVM). The support vectors of two smaller subSVMs are then passed down as input to next level subSVM and this process is repeated until we reach the final SVM at the bottom where the global SVM model is obtained as shown in Fig. 2. Sun *et al.* [10] implemented the same cascade SVM architecture with the help of MapReduce and Twister. A similar approach is taken by Vazquez *et al.*'s [22], where the size of training data at each node increases with each subsequent iteration causing high communication overhead. The communication overhead is the amount of data transfer over the communication network from one node to another during the training process.

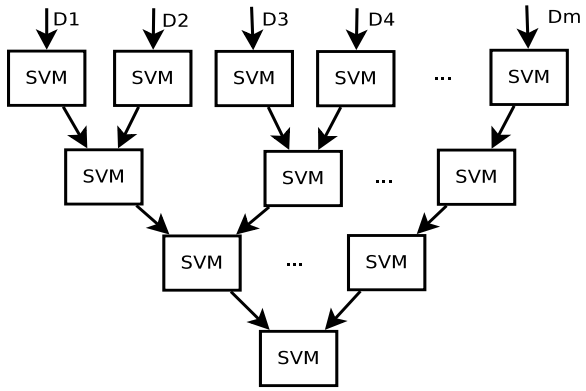


Fig. 2. Training flow of cascade SVM [23].

Hsieh *et al.* [9] instead propose a divide-and-conquer (DC-SVM) approach to solve the kernel SVM problem. DC-SVM is similar to cascade SVM with two major differences: 1) It uses K-means clustering to partition the dataset instead of sequential or random partition. 2) It passes all of the training vectors and solutions from one level to next level, instead of only SVs. However, the disadvantage of DC-SVM is that at the last level, it operates a single SVM on the whole training dataset which is essentially quite slower and non-scalable for larger datasets. A similar approach is also proposed by Alham *et al.* [11] [19] [20] for large scale image annotations using MapReduce. The entire training data is partitioned into smaller subsets and each of these subsets are allocated to separate Map tasks. Each Map task optimizes the partition in parallel. The outputs, Lagrangian multiplier $\boldsymbol{\alpha}$ arrays and bias b values, obtained from each Map task are then joined in the $Reduce$ task in order to produce the global Lagrangian multiplier $\boldsymbol{\alpha}$ array and the average bias b. Even though this model reduces training time, the blind partitioning of the sample dataset results in different classification performance with different initialization for the partitions.

To reduce the communication involved in the methods discussed above, You *et al.* [12] propose a communication avoiding SVM (CA-SVM) for shared memory architecture by combining several approaches like the cascade SVM, DC-SVM etc. Basically, a divide and conquer filter selects the LSVs and their corresponding Lagrange multipliers $\boldsymbol{\alpha}$

values from one level in order to initialize next level Lagrange multipliers $\boldsymbol{\alpha}$ in cascade SVM which results in faster convergence. The partitions are obtained as the clusters resulting from a balanced k-means clustering algorithm. This approach performs better when the dataset contains well-separated clusters and each cluster contains points from the corresponding classes. However, its performance degrades in the case of overlapping clusters and also if a cluster contains examples belonging to only one class.

Yu *et al.* [24] make an attempt to train the SVM on a large dataset which is suitable only for well separated low dimensional data. This method uses a hierarchical clustering based tree called CF-tree where data at the root of the tree is first used to train an initial model which is refined successively. This refinement is carried out at each subsequent lower level by considering the clusters near to the decision boundaries in the upper level to train an SVM from the beginning. This is in stark contrast to our proposed approach where the Lagrangian multipliers of the SVM in a lower level are initialized from the upper level Lagrangian multipliers that is empirically shown to help in the faster convergence of the SVM.

In our proposed DiP-SVM, with distribution preserving partitioning approach, we aim to address the performance degradation issue in overlapping clusters. To reduce communication overhead, we only pass LSVs as in [23]. We show that if the LSVs are in strong agreement with the GSVs, this methodology works without any significant loss of accuracy.

3 Details of Proposed Approach

The proposed DiP-SVM approach operates in two distinct phases, namely, *distribution preserving partitioning (DPP)* phase and *distributed learning* phase. In the DPP phase, the entire dataset is divided in such a way that the first and second-order statistics of the dataset are preserved in every partition. In the distributed learning phase, each partition is trained in parallel within a distributed environment and the global decision boundary is obtained by hierarchically combining local decision boundaries. The detailed discussion of these two phases is given below:

3.1 Distribution Preserving Partitioning (DPP) Phase

This phase revolves around balanced partitioning of data points while preserving the statistical properties of the entire dataset. A dataset $\mathbf{D}$ consisting of N data points can be expressed as

$$\mathbf{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}, \tag{1}$$

with mean $\boldsymbol{\mu}_{\mathbf{D}}$ and variance $\boldsymbol{\Sigma}_{\mathbf{D}}$. In this phase, we divide $\mathbf{D}$ into P balanced partitions $\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_P$, each of size $N_p = \lceil \frac{N}{P} \rceil$ such that the p^{th} partition can be expressed as

$$\mathbf{D}_p = \{(\mathbf{x}_{\pi(n)}, y_{\pi(n)})\}_{n=1}^{N_p}, \tag{2}$$

with mean $\boldsymbol{\mu}_{\mathbf{D}_p}$ and variance $\boldsymbol{\Sigma}_{\mathbf{D}_p}$. The mapping function $\pi(n)$ gives the corresponding index in $\mathbf{D}$ of a point at index n in $\mathbf{D}_p$. These partitions are created in such a manner that the first and second order statistics (mean and variance) of each partition is approximately close to the given dataset $\mathbf{D}$ which can be expressed as the following objective functions:

$$\min \sum_{p=1}^{P} ||\boldsymbol{\mu}_{\mathbf{D}_p} - \boldsymbol{\mu}_{\mathbf{D}}|| \quad \text{and} \tag{3}$$

$$\min \sum_{p=1}^{P} ||\boldsymbol{\Sigma}_{\mathbf{D}_p} - \boldsymbol{\Sigma}_{\mathbf{D}}||. \tag{4}$$

Also, we want each partition to retain the ratio of the number of points in each class as in the complete dataset $\mathbf{D}$. So, we use the partitioning approach separately on each of the constituent classes $\mathbf{D}_c$ instead of $\mathbf{D}$. To formalize this idea more clearly, we represent $\mathbf{D}$ as a collection of $\mathcal{C}$ classes:

$$\mathbf{D} = \{\mathbf{D}_c\}_{c=1}^{\mathcal{C}}, \tag{5}$$

where the c^{th} class contains N_c data points. The research work proposed in [25] attempts to achive similar objective, however, this method is highly dependant on the initial sets chosen and thus may lead to partitions whose distribution may not be balanced. In order to solve the objective functions stated in Equation 3 and Equation 4, we employ K-means clustering on each of the classes separately whose objective is defined as

$$\underset{\mathbf{D_c}}{\arg\min} \sum_{k=1}^{K} \sum_{\mathbf{x} \in D_{ck}} \|\mathbf{x} - \boldsymbol{\mu}_{ck}\|^2, \tag{6}$$

where K is the number of clusters, $\mathbf{D}_{ck}$ is the k^{th} cluster of the c^{th} class and $\boldsymbol{\mu}_{ck}$ is the mean of points in $\mathbf{D}_{ck}$. From each cluster $\mathbf{D}_{ck}$ having N_{ck} points, P balanced partitions $\{D_{ck}^p\}_{p=1}^{P}$ are created, each containing $N_{ck}^p = \left\lceil \frac{N_{ck}}{P} \right\rceil$ points selected according to uniform distribution. This is carried out for all the $\mathcal{C}$ classes using Algorithm 1. The flowchart for this process is shown in Fig. 3.

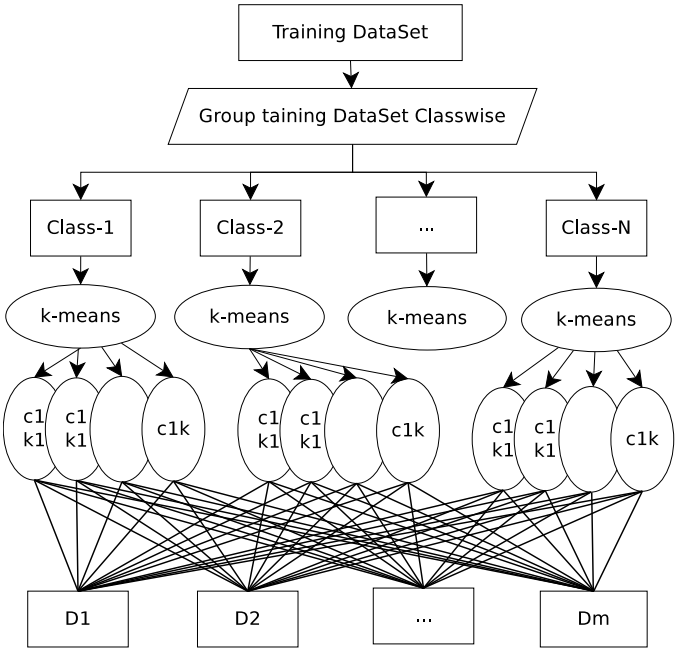


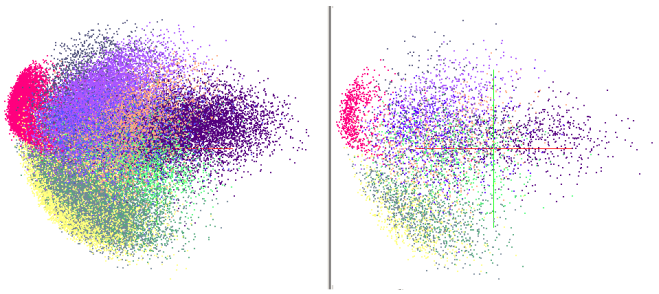
Fig. 3. The flowchart of proposed distribution preserving partitioning (DPP).

Algorithm 1 Distribution Preserving Partitioning (DPP)

Input:
$\mathbf{D}$: $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N},\ \mathbf{x}_n \in \mathbb{R}^d,\ y_n \in \{c = 1, 2, ..., \mathcal{C}\}$
N : #instance (vectors) in $\mathbf{D}$
d : #dimensions
$\mathcal{C}$: #classes in the dataset
P : #partitions
K : #clusters
$\mathcal{U}(N_{ck}^p, N_{ck})$: generate N_{ck}^p random indexes in range $[1 - N_{ck}]$

Output:
$\{\mathbf{D}_p\}_{p=1}^{P}$: partitions

$\{\mathbf{D}_c\}_{c=1}^{\mathcal{C}} \leftarrow groupClasswise(\mathbf{D}, K)$; //where $\mathbf{D}_c$ is the set of points belongs to c_{th} class.
for $c = 1 \rightarrow \mathcal{C}$ **do**
  $\{\mathbf{D}_{ck}\}_{k=1}^{K} \leftarrow kmeans(\mathbf{D}_c, K)$; //where $\mathbf{D}_{ck}$ is the k_{th} cluster of the c_{th} class.
  for $k = 1 \rightarrow K$ **do**
    for $p = 1 \rightarrow P$ **do**
      $N_{ck}^p \leftarrow \left\lceil \frac{N_{ck}}{P} \right\rceil$;
      $\mathbf{D}_{ck}^p \leftarrow \mathbf{D}_{ck}[\mathcal{U}(N_{ck}^p, N_{ck})]$
      $\mathbf{D}_p \leftarrow \mathbf{D}_p \cup \mathbf{D}_{ck}^p$;
      $\mathbf{D}_{ck} \leftarrow \mathbf{D}_{ck} - \mathbf{D}_{ck}^p$;
      $N_{ck} \leftarrow N_{ck} - N_{ck}^p$;
    end for
  end for
end for
return $\{\mathbf{D}_p\}_{p=1}^{P}$;



(A) (B)

Fig. 4. (A) MNIST [26] dataset containing 60,000 samples. (B) A sample partition generated using Algorithm 1. Colors represent various classes. Best viewed in color.

Fig. 4, gives an example of entire dataset $\mathbf{D}$ and one of its partition $\mathbf{D}_p$ obtained by using Algorithm 1 for the MNIST dataset. It can be observed that partition $\mathbf{D}_p$ is a sparse representations of the entire training data set $\mathbf{D}$. Also, the statistical properties of the partitions $\{\mathbf{D}_p\}_{p=1}^{P}$ are approximately close to the statistical properties of entire dataset $\mathbf{D}$. This can be formally stated as below:

From the maximum likelihood estimation (MLE), it is intuitive that mean and variance of the entire dataset $\mathbf{D}$ containing N samples (N is very large) will be approximately close to mean and variance of its partition $\mathbf{D}_p$ for sufficiently large size $\mathbf{N}_p$, i.e. $\boldsymbol{\mu}_{\mathbf{D}_p} \approxeq \boldsymbol{\mu}_{\mathbf{D}}$ and $\boldsymbol{\Sigma}_{\mathbf{D}_p} \approxeq \boldsymbol{\Sigma}_{\mathbf{D}}$. In order to justify this statement, we empirically show that for K number of clusters (K is large), mean and variance of partitions obtained using Algorithm 1 are approximately close to the entire dataset.

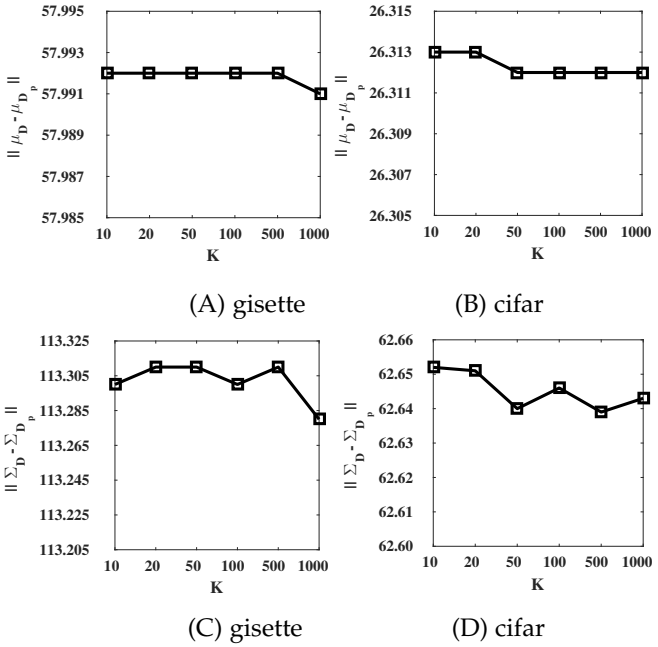


Fig. 5. Comparison of deviation in the means (A & B) and variances (C & D) of partitions from the mean and variance of the full dataset on gisette and cifar dataset for different values of K.

Fig. 5 presents the effect of various values of K on different datasets. It can be observed that the high values of K reduces the deviation in the means and variances of partitions from the the mean and variance of full dataset. Empirically, it was found that over all datasets, setting a large value of $K(\approx 1000)$ caused the least distortion in mean and variance between the resulting partitions and the original dataset leading to better distribution preservation. One such result is shown in Table 1 where a comparison is made between the mean and variance of the partitions formed using the proposed approach and random partitioning approach [10], [11] on the kddcup99 dataset ($K = 1000$ and $P = 100$). It can be seen that the partitions formed using DPP are up to $10^3\times$ closer to the mean and variance of the entire dataset than the random partitions (all distances are scaled by the product of norms in order to show the relative differences).

TABLE 1
Distortion in the Distributions of Partitions for Random Partitioning vs. proposed Distribution Preserving Partitioning on kddcup99 Dataset

	Min	Mean±Deviation	Max
Random $\|\|\boldsymbol{\mu}_{\mathbf{D}_p^*} - \boldsymbol{\mu}_{\mathbf{D}}\|\|$	1.76e-06	1.21e-05±9.15e-06	4.84e-05
Random $\|\|\boldsymbol{\Sigma}_{\mathbf{D}_p^*} - \boldsymbol{\Sigma}_{\mathbf{D}}\|\|$	1.14e-05	4.63e-05±2.78e-05	1.83e-04
Proposed $\|\|\boldsymbol{\mu}_{\mathbf{D}_p} - \boldsymbol{\mu}_{\mathbf{D}}\|\|$	7.53e-08	1.10e-07±1.99e-08	1.60e-07
Proposed $\|\|\boldsymbol{\Sigma}_{\mathbf{D}_p} - \boldsymbol{\Sigma}_{\mathbf{D}}\|\|$	1.02e-06	1.32e-06±1.90e-07	1.78e-06

Fig. 6 & 7 show the comparison of classification performance for individual partitions and the average classification of all the partitions formed by using random partitioning and distribution preserving partitioning (DPP) on the datasets *ijcnn1* and *kddcup99*. The results clearly

show the superiority of DPP over random partitioning. On the *ijcnn1* dataset, all the partitions (partition number denoted on x-axis of both Fig. 6 and Fig. 7) generated using random partitioning produce lower classification accuracy than the partitions generated using DPP. This trend can also be observed on the *kddcup99* dataset as the classification performance of each partition produced using DPP is almost consistent while the partitions generated using random partitioning shows high deviation (see Fig. 7). These results indicate that DPP causes each partition to contain either global support vectors or points which are very close to the global support vectors.

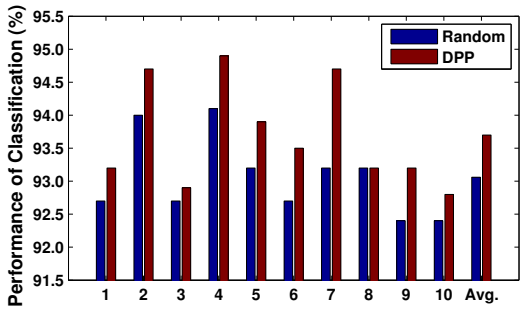


Fig. 6. A comparison of classification performance (%) for the partitions generated using random partitioning and proposed distribution preserving partitioning (DPP) for ijcnn1 dataset.

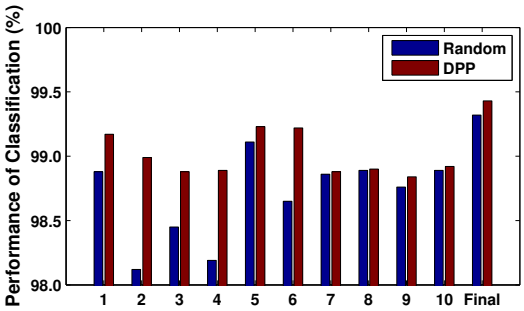


Fig. 7. A comparison of classification performance (%) for the partitions generated using random partitioning and proposed distribution preserving partitioning (DPP) kddcup99 dataset.

3.2 Distributed Learning Phase

After data partitioning, we use a modified cascade SVM for distributed learning of support vectors. This learning approach differs from the usual cascade SVM [23] in:

1) The proposed framework uses distribution preserving partitioning (DPP) approach given in Algorithm 1, to partition the dataset instead of sequential/random partition as used in cascade SVM or clusters as partitions used in DC-SVM [9] and CA-SVM [12]. Due to this, the local support vectors (LSVs) in each of the partitions are shown to be in agreement to the global support vectors (GSVs) as shown in Fig. 9, which helps in maintaining classification accuracy.
2) The proposed approach passes only relevant training vectors $\mathbf{R}_p^{(l)}$ for partition p from level l to $l+1$ instead of only LSVs as in cascade SVM or all the training vectors as in DC-SVM. Relevant training vectors are those vectors whose distance from the local hyperplane is less than threshold T_γ. This results in a significant reduction in communication overhead than DC-SVM.
3) As the Lagrangian multipliers $\boldsymbol{\alpha}^{(l+1)}$ in the next stage are initialized from previous level Lagrangian multipliers $\boldsymbol{\alpha}^{(l)}$ similar to DC-SVM, the method achieves early convergence. Also, the number of Lagrangian multipliers $\boldsymbol{\alpha}^{(l)}$ passed from level l to level $l+1$ is significantly lower than DC-SVM.

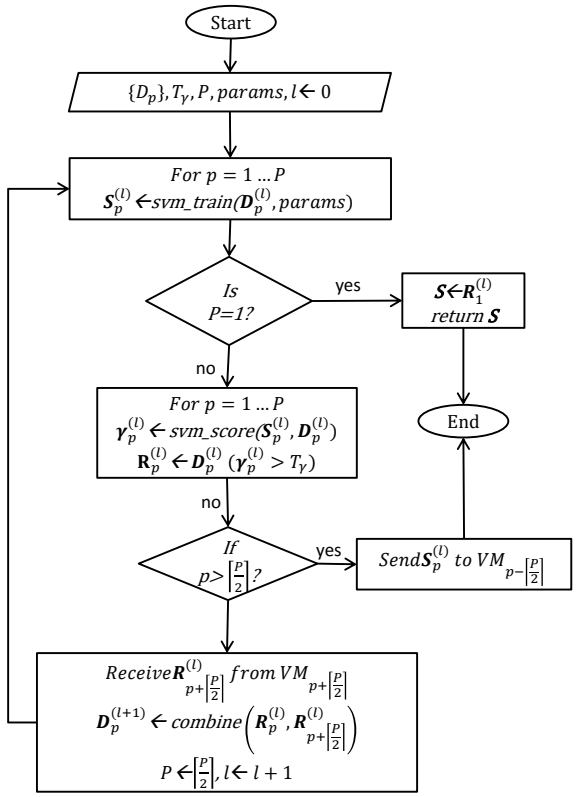


Fig. 8. Flow chart of DiP-SVM training phase.

For each partition $\mathbf{D}_p$ obtained using Algorithm 1, a sequential SVM model is trained independently at level l (=0 initially) which results in local support vectors $\mathbf{S}_p^{(l)}$. The sequential SVM can be used for each of these partitions separately as the sequential minimal optimization (SMO) rapidly converges for small datasets. The sequential minimal optimization (SMO) is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support vector machines as shown in Equation 7. At level l for p^{th} partition, SMO solves the following dual objective function as defined in [1]:

$$\max_{\boldsymbol{\alpha}_p^{(l)}, b_p^{(l)}} \sum_{i=1}^{N_p} \alpha_i - \frac{1}{2} \sum_{i=1}^{N_p} \sum_{j=1}^{N_p} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i^T, \mathbf{x}), \tag{7}$$

subject to conditions:

1) $\sum_{i=1}^{N_p} \alpha_i y_i = 0$, and

Algorithm 2 DiP-SVM Learning

Input:
$\mathbf{D} : \{(x_n, y_n)\}_{n=1}^{N},\ x_n \in \mathbb{R}^d,\ y_n \in \{-1, +1\}$
N : #instance (vectors) in $\mathbf{D}$
d : #dimensions
P : #partitions
K : #clusters
T_γ :Threshold
$params : \{KernelFunction, KernelParameters\}$

Output:
$\mathbf{S}$: Global support vectors
$\boldsymbol{\alpha}$: Global Lagrangian Multipliers

$\{\mathbf{D}_p\}_{p=1}^{P} \leftarrow DPP(\mathbf{D},\ P,\ K)$;
Level $l \leftarrow 0$;
$\boldsymbol{\alpha}_p^{(l)} \leftarrow 0$;
while $true$ **do**
  for $p = 1 \rightarrow P$ {Parallelly over cluster} **do**
    $\{\boldsymbol{\alpha}_p^{(l)}, b_p^{(l)}\} \leftarrow svm\_train(\mathbf{D}_p^{(l)}, \boldsymbol{\alpha}_p^{(l)},\ params)$;
    $\mathbf{S}_p^{(l)} \leftarrow \mathbf{D}_p^{(l)}\{\boldsymbol{\alpha}_p^{(l)} > 0\}$;
    if $P = 1$ **then**
      $\mathbf{S} \leftarrow \mathbf{S}_1^{(l)}$;
      return $\mathbf{S}$;
    else
      $\boldsymbol{\gamma}_p^{(l)} \leftarrow svm\_score(\mathbf{S}_p^{(l)}, \boldsymbol{\alpha}_p^{(l)}, b_p^{(l)},\ \mathbf{D}_p^{(l)})$;
      $\mathbf{R}_p^{(l)} \leftarrow \mathbf{D}_p^{(l)}(\boldsymbol{\gamma}_p^{(l)} \geq T_\gamma)$;
      if $p > \lceil P/2 \rceil$ **then**
        $Send\ \{\mathbf{R}_p^{(l)}, \boldsymbol{\alpha}_p^{(l)}\}\ to\ VM_{p-\lceil P/2 \rceil}$;
      else
        $Receive$ $\{\mathbf{R}_{p+\lceil P/2 \rceil}^{(l)}, \boldsymbol{\alpha}_{p+\lceil P/2 \rceil}^{(l)}\}\ from\ VM_{p+\lceil P/2 \rceil}$;
        $\mathbf{D}_p^{(l+1)} \leftarrow combine(\mathbf{R}_p^{(l)},\ \mathbf{R}_{p+\lceil P/2 \rceil}^{(l)})$;
        $\boldsymbol{\alpha}_p^{(l+1)} \leftarrow combine(\boldsymbol{\alpha}_p^{(l)},\ \boldsymbol{\alpha}_{p+\lceil P/2 \rceil}^{(l)})$;
      end if
    end if
  end for
  $P \leftarrow \lceil P/2 \rceil$
  $l \leftarrow l + 1$;
end while

2) $0 \leq \alpha_i \leq C$,

where, $\alpha_i \in \alpha_p^{(l)}$, $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N_p} \in \mathbf{D}_p^{(l)}$, and $b_p^{(l)}$ is the bias. At p^{th} virtual machine (VM), the local support vectors $\mathbf{S}_p^{(l)}$ are those points having $\alpha_i > 0$ i.e.

$$\mathbf{S}_p^{(l)} = \mathbf{D}_p^{(l)}\{\boldsymbol{\alpha}_p^{(l)} > 0\}, \tag{8}$$
$$\boldsymbol{\alpha}_{\boldsymbol{p}}^{(\boldsymbol{l})} = \boldsymbol{\alpha}_p^{(l)}\{\boldsymbol{\alpha}_p^{(l)} > 0\}. \tag{9}$$

The distance $\gamma_{\mathbf{x}_i}$ of each point $(\mathbf{x}_i, y_i) \in \mathbf{D}_p^{(l)}$ from the hyperplane is calculated as

$$\gamma_{\mathbf{x}_i} = \sum_{(\mathbf{x},y)\in\mathbf{S}_p^{(l)}, \alpha\in\boldsymbol{\alpha}_p^{(l)}} \left(\alpha y K(\mathbf{x}^T, \mathbf{x}_i) + b_p^{(l)}\right). \tag{10}$$

All those points having $\gamma_{\mathbf{x}_i} > T_\gamma$ are considered as relevant vectors $\mathbf{R}_p^{(l)}$.

$$\mathbf{R}_p^{(l)} = \mathbf{D}_p^{(l)}\{\boldsymbol{\gamma}_p^{(l)} \geq T_\gamma\}, \tag{11}$$
$$\boldsymbol{\alpha}_p^{(l)} = \boldsymbol{\alpha}_p^{(l)}\{\boldsymbol{\gamma}_p^{(l)} \geq T_\gamma\}. \tag{12}$$

Here, T_γ is the threshold on distance from the local hyperplane. For $T_\gamma = 1$, only vectors which are on or within the margin are selected. If $T_\gamma < 1$, the number of points transferred over the network reduces but it may increase the loss of accuracy as some key points may be missed. For $T_\gamma > 1$, more points than the points which are on or within the margin are passed over the network which may reduce the loss of accuracy incurred.

These relevant vectors $\mathbf{R}_p^{(l)}$ at level l are then collected and used as training data to learn an SVM model in the next level. The training data for partition p at level $l+1$ ($\mathbf{D}_p^{(l+1)}$) and the Lagrangian multipliers $\boldsymbol{\alpha}_p^{(l+1)}$ are calculated as follows:

$$\mathbf{D}_p^{(l+1)} = \{\mathbf{R}_p^{(l)},\ \mathbf{R}_{p+\lceil P/2 \rceil}^{(l)}\}, \tag{13}$$
$$\boldsymbol{\alpha}_p^{(l+1)} = \{\boldsymbol{\alpha}_p^{(l)}, \boldsymbol{\alpha}_{p+\lceil P/2 \rceil}^{(l)}\}. \tag{14}$$

This process is repeated $L-1$ times. Finally, at level $l = L$, it produces the final model which constitutes the global support vectors $\mathbf{S}$, global Lagrangian multipliers $\boldsymbol{\alpha}$, and bias b. This is explained in Algorithm 2. Fig 8 gives the flowchart of the DiP-SVM training process. We make adjustments to T_γ to achieve a trade-off between communication overhead and loss of accuracy.

3.3 Empirical Evaluation

To show that the LSVs obtained from DiP-SVM are very close to the global support vectors in comparison to other recent methods for partitioning in [9], [12], we consider two cases that can arise: 1) data that can be well clustered and 2) data that has considerable overlap. In the first case, we use a synthetic dataset, which is a mixture of four 2D Gaussian distributions and is well separable into two clusters. In Fig. 9, we can see the two clusters obtained by the methods in [9], [12] contain data from both the classes. The local support vectors obtained from both the partitions are dissimilar causing the local decision hyperplanes to completely contradict each other. The final global decision hyperplane also shows high deviation from the decision hyperplane produced using a sequential SVM. On the other hand, the proposed DiP-SVM method produces local and global decision hyperplanes which show high correspondence to each other as well as to the decision hyperplane of sequential SVM (cosine similarity ≈ 1). This shows the suitability of the DiP-SVM over the existing clustering-based methods in [9], [12] for well-separated clusters.

The second case is more realistic as the features obtained from most of the datasets create overlapping clusters for distributed processing. This is demonstrated in Fig. 10, where clustering based partitioning schemes are not able to produce isolated clusters. In this experiment, we use a synthetic dataset, which is a mixture of four 2D Gaussian distributions and has significant overlap between the two classes. In such a case, clustering-based methods in [9], [12] generate partitions which produce LSVs that are in strong disagreement (cosine similarity ≈ 0) to each other as well

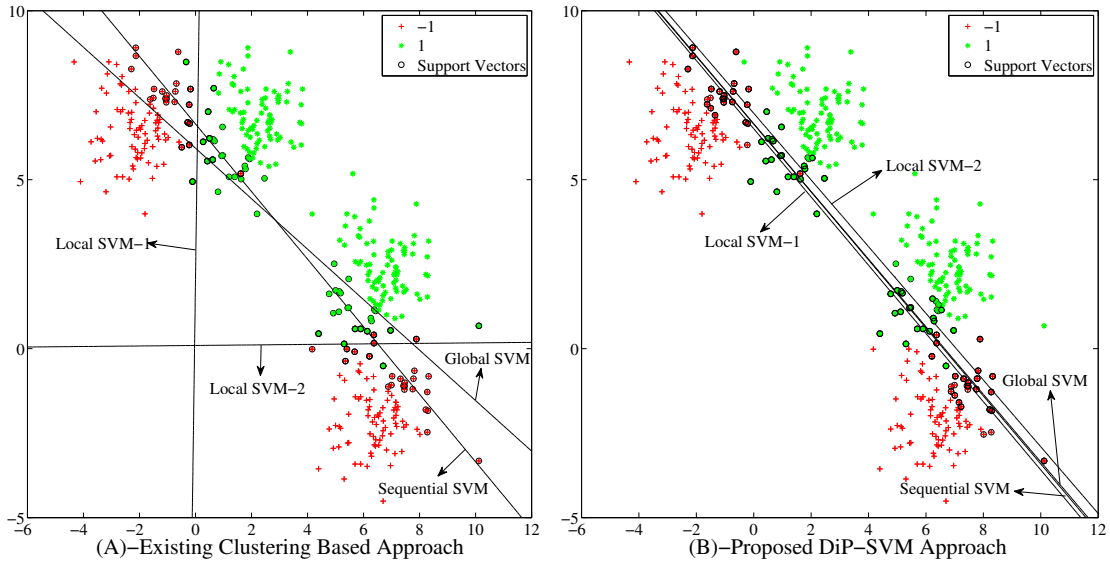


Fig. 9. Comparison of DiP-SVM with the existing clustering based methods in [9] [12] for local and global solutions on well separable clusters having points from both the classes. (Best viewed in colors)

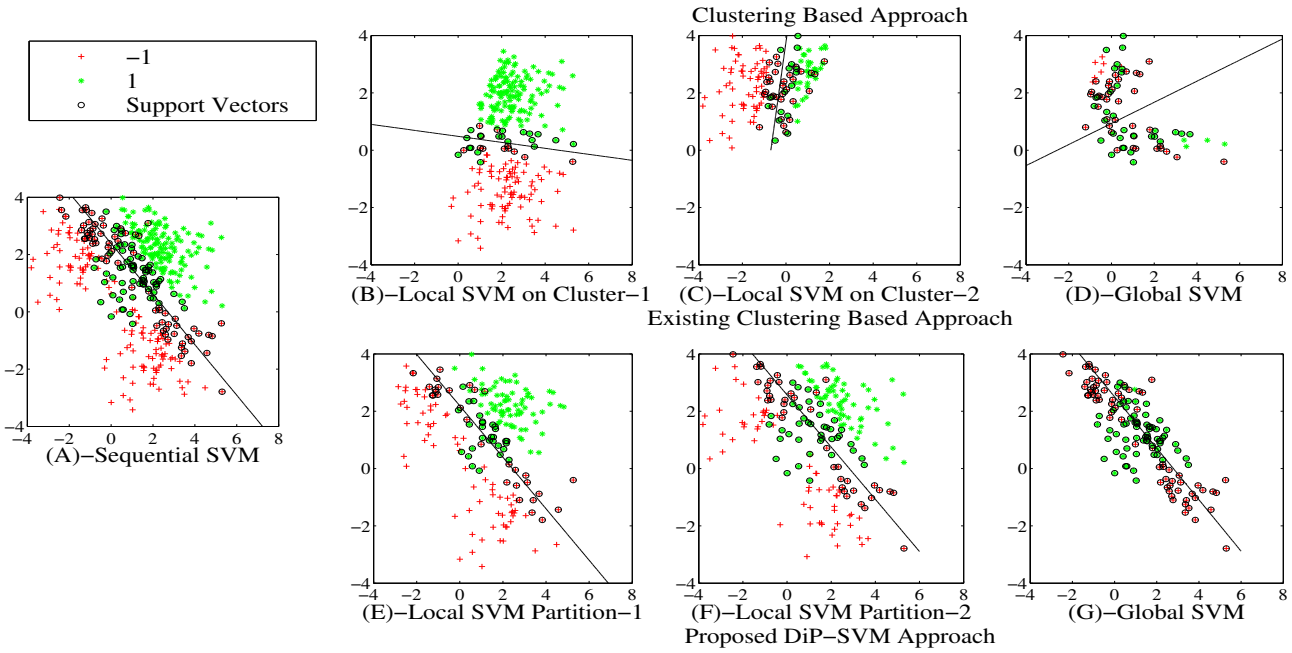


Fig. 10. Comparison of DiP-SVM with the existing clustering based methods in [9] [12] for local and global solutions using linear kernel. (Best viewed in colors)

as to sequential SVM as shown in Figures. Even if the partition contains points from both the classes (green star and red plus), most of the support vectors obtained from the two clusters do not contribute to global support vectors. Further, the points which would have contributed to the GSVs according to the sequential SVM implementation as shown in Fig. 10-(D) are already eliminated at the local level. Fig. 10-(E),(F)&(G) also demonstrate the suitability of the DiP-SVM over the the existing clustering-based methods in [9], [12] on the same dataset using linear kernel. The local and global decision hyperplanes produced by DiP-SVM show high cosine similarity to each other as well as to the decision hyperplane produced by sequential SVM (cosine similarity $\approxeq 1$).

The case for DiP-SVM grows stronger in a case of overlapping clusters as it not only produces decision boundaries at the local level which are in strong agreement among themselves but also preserves the LSVs which contribute to the global decision boundary. Fig. 11 demonstrates the suitability of the DiP-SVM over the existing clustering-based

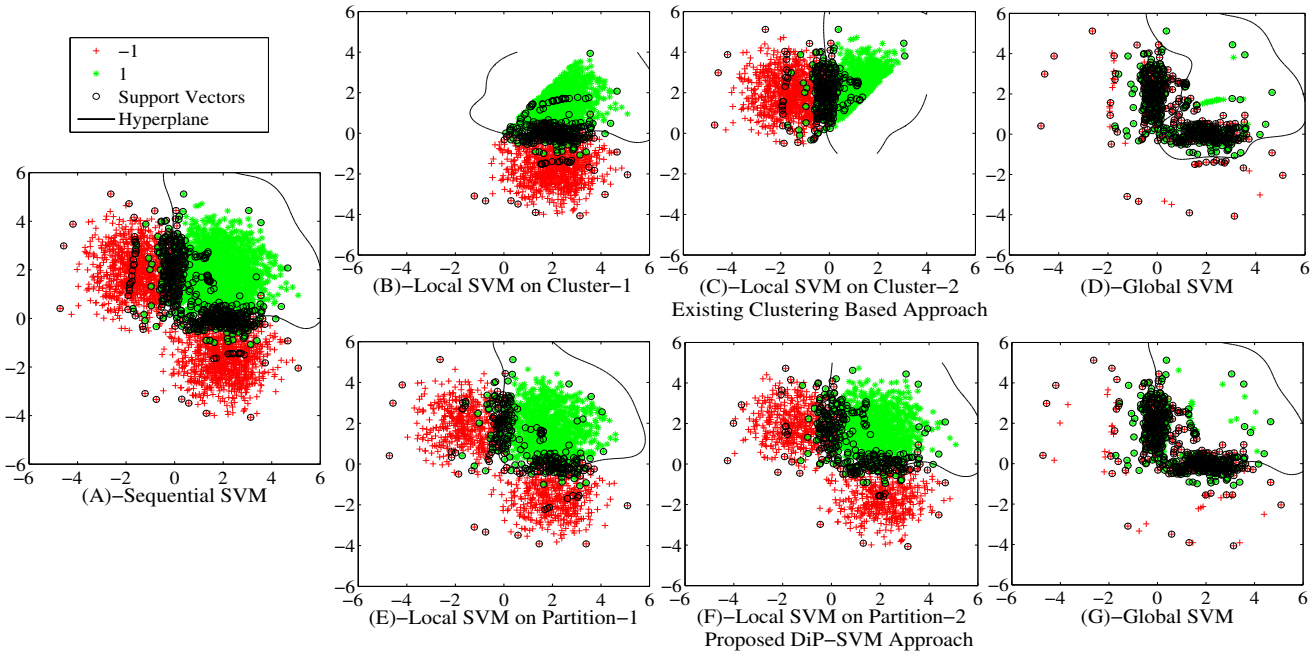


Fig. 11. Comparison of DiP-SVM with the existing clustering based methods in [9] [12] for local and global solutions using non-linear kernel. (Best viewed in colors)

methods in [9], [12] for overlapping clusters using Gaussian kernel. In order to show the effectiveness of the selected local SVs at any level we use the relevant SV index defined as

$$Relevant\ SV\ Index = \frac{|S^l \cap S^{l+1}|}{|S^l|}.$$

This index measures the number of SVs at level l which are also considered as SVs at level $l+1$.

In this experiment, we use a synthetic dataset, which is a mixture of four 2D Gaussian distributions. We compare the relevant SV index of the local SVs produced by various clustering based approaches with the GSVs produced at the last step as shown in Fig 11. It can be seen in Fig 11(B),(C) and (D) that a large number of LSVs produced by the existing clustering based SVMs [9], [12] are not included in GSVs. This makes the relevant SV index between final GSVs and the LSVs quite low (< 0.5). On the other hand, the LSVs produced by DiP-SVM as shown in Fig 11(E),(F) and (G) are in close agreement. A high relevant SV index of $\approxeq 1$ confirms this proposition. Further, the GSVs produced in Fig 11(G) is closer to the sequential SVM based SVs (relevant SV index of $\approxeq 1$) than the GSVs produced in Fig 11(D).

4 Experimental Evaluation

In literature, most of the existing distributed SVM implementations are based on OpenMPI [6] or Hadoop. In our approach, we use OpenMPI architecture which is a default standard for multi-core systems programming and makes it suitable for comparison with other distributed approaches like Hadoop. OpenMPI program can easily be integrated into the cloud environment using tools like StarCluster [27]. StarCluster is an open source cluster computing toolkit for Amazons EC2 [7] released under the LGPL license. StarCluster has been designed to automate and simplify the process of building, configuring, and managing clusters of virtual machines on Amazons EC2 cloud. StarCluster allows anyone to easily create a cluster computing environment in the cloud, suited for distributed and parallel computing applications and systems. The local SVMs as well as the global SVMs were trained using LIBSVM [3].

4.1 Results and Discussion

In order to evaluate the proposed approach, we conducted experiments on several real-world datasets from the domains like computer vision, cyber security, economics, text classification, etc. Table. 2 gives the statistics for each of these datasets.

TABLE 2
Details of the Datasets Used

Dataset	Application Domain	#Dim.	#Train	#Test
gisette [12]	Digit Classification	5000	6000	1000
adult [12]	Economics	123	32561	16281
ijcnn1 [12]	Text Classification	22	49990	91000
cifar [9]	Visual Recognition	3072	50000	10000
webspam [9]	Spam Detection	254	280000	70000
covtype [9]	Forest Classification	54	464810	116202
kddcup99 [9]	Intrusion Detection	123	4898431	311029
mnist8m [9]	Digit Classification	784	8000000	100000
url	URL Classification	2396130	3131961	100000

Fig. 12 shows the comparison of classification performance of the proposed DPP-based partitioning with existing random partitioning approach. Fig. 13 presents the comparison of training times taken by the proposed DPP-based distributed SVM with random partitioning based distributed SVM. The results in these plots clearly indicate that the proposed distributed SVM is able to maintain its

TABLE 3
Comparison of Classification Performance (in %) and Average Runtime (in Sec.)

Method	Parameters				LIBSVM		DC-SVM		CA-SVM		Proposed		Change	
Dataset	C	γ	B	h	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Scale
gisette	1	2e-4	2	1	97.70	125	97.60	299	96.00	81	97.90	29	+0.20	4×
adult	32	2^{-7}	2	2	85.08	761	84.79	78	83.00	121	84.01	58	−1.07	13×
ijcnn1	32	2	2	5	98.69	20	98.53	318	90.16	121	98.61	3	−0.08	7×
cifar	8	2^{-22}	2	4	89.50	13892	80.15	22330	63.94	2143	89.49	2378	−0.01	6×
webspam	8	32	2	10	99.28	15056	99.28	10485	99.11	3093	99.15	1942	−0.13	8×
covtype	32	32	2	10	96.01	31785	95.95	17456	75.04	34025	93.07	3919	−2.94	8×
kddcup99	256	0.5	2	10	99.57	37684	99.49	23346	NA	NA	99.53	3170	−0.04	12×
mnist8m	1	2^{-21}	2	10	99.91	$\approx 10^6$	99.91*	NA	NA	NA	99.99	99874	+0.08	11×
url	100	3e-7	2	10	96.75	486559	NA	NA	NA	NA	96.88	53374	+0.13	9×

Acc.-Accuracy (%), Change-with respect to LIBSVM, NA - Not Available, *taken from [9]

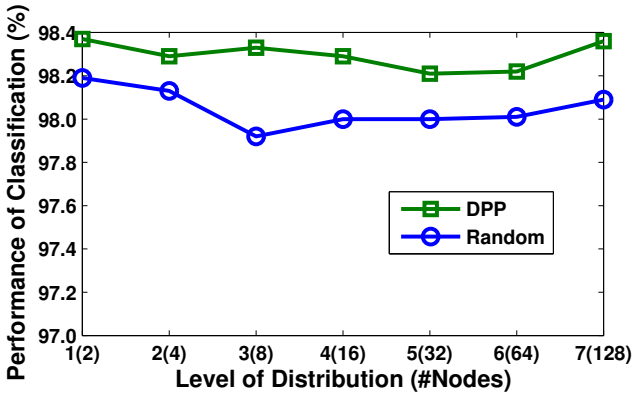


Fig. 12. Level wise comparison of classification performance (%) for random partitioning and distribution preserving partitioning (DPP) for ijcnn1 dataset.

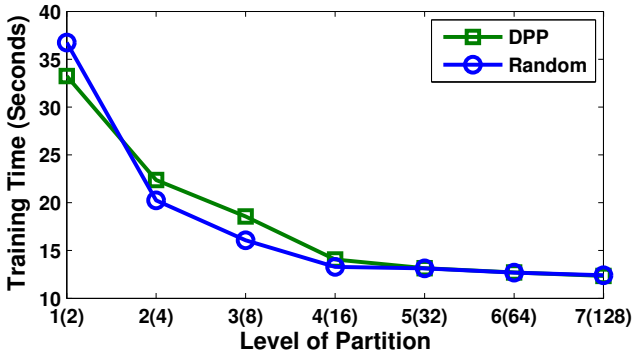


Fig. 13. Level wise comparison of training time for random partitioning and distribution preserving partitioning (DPP) for ijcnn1 dataset.

performance at the various level of distribution while the performance of existing random partitioning degrades at higher levels of distribution.

Table 3 gives the performance comparison of DiP-SVM with Sequential LIBSVM, DC-SVM [9], CA-SVM [12] on the datasets as listed in Table. 2. All results are calculated in a cluster of 11 virtual machine where one virtual machine acts as a master node and remaining 10 virtual machines act as worker nodes. The SVM parameters C and γ for each dataset are selected using grid evaluation. It can be observed from the Fig. 14 that the DiP-SVM achieves better performance consistently irrespective of the distribution of the data. The performance is evaluated in terms of loss of accuracy with respect to sequential SVM. Each dataset shows the only small change in the accuracy which is comparable to sequential SVM (less than 0.5%), and better than existing methods DC-SVM [9] and CA-SVM [12] as shown in Fig. 15. However, many times, it also shows small improvements too. This reduction in loss of accuracy is because of the proposed distribution preserving partitioning approach. Fig. 16 presents the training time performance which shows that the time take in training on various datasets by proposed DiP-SVM is comparatively less than the sequential SVM and existing distributed implementations of SVM. The results show that the training of DiP-SVM is approximate $9\times$ faster than the training of sequential SVM for each dataset.

Fig. 17 shows the performance of the DiP-SVM (Levels $L = 10$ with binary splitting) on kddcup99 dataset having ≈ 5 million records. It can be observed from the figure that a large portion of the irrelevant data ($\approx 96\%$) is eliminated at first layer itself as shown in Fig. 17-(A)&(B). Thus from the second layer onwards the amount of data remaining is quite less for calculating the global decision boundary which leads to low communication overhead. After the first level, very less data is eliminated as most of the data points in hand turn out to be support vectors in the subsequent levels as shown in Fig. 17-(C). Fig. 17-(D) shows that the data in all the nodes is evenly divided at each layer. Fig. 17-(E)&(F) show the amount of data transferred over the network at each level during training phase on respective datasets. This shows that the amount of data transferred decreases as the levels increase. Fig. 17-(G) shows the cumulative data transferred. The total data transferred is $\approx 8\%$ of the entire dataset. Fig. 17-(H) shows the time taken to train SVM model at each level for respective datasets. This figure shows that as the SVs are combined at each successive level, the computation time increases on an average. However, the top level shows a reduction in the training time for this two possible reasons are 1) the use of Lagrangian multipliers from the previous level due to which it converges into fewer iterations only, or 2) the size of training data is less in comparison to the previous level. Fig. 17-(I) shows the cumulative time taken to train SVM model at each level. The entire training takes ≈ 30 minutes for training on the kddcup99 dataset.

Further, the partitions generated using the proposed distribution preserving partitioning (DPP) approach are

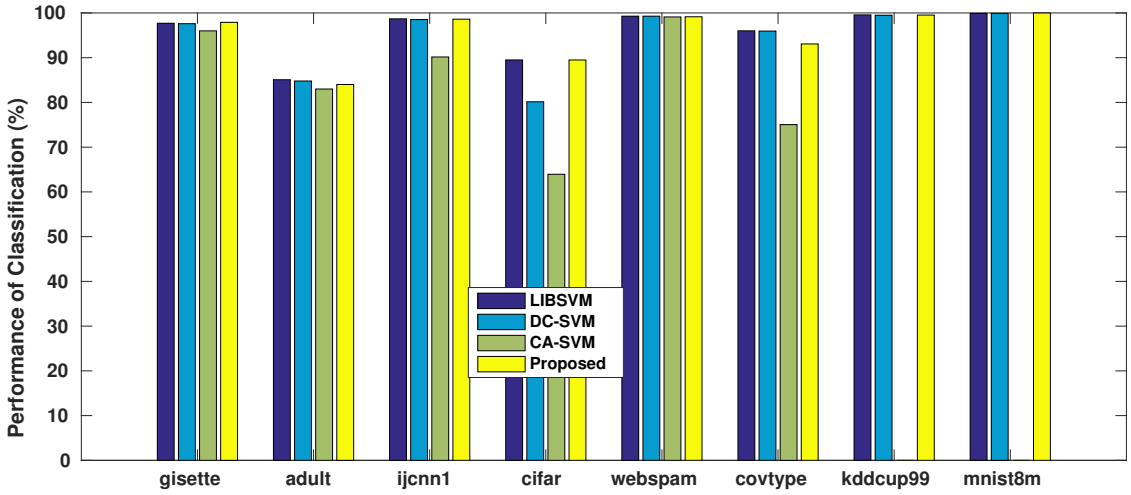


Fig. 14. A comparison of the classification performance (%) of LIBSVM, DC-SVM, CP-SVM and proposed distributed SVM on publicly available datasets.

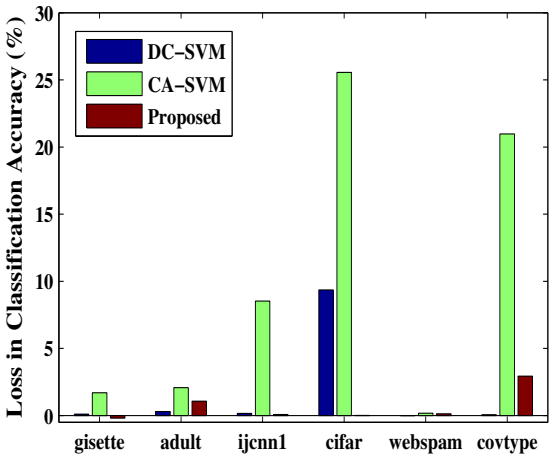


Fig. 15. A comparison of the loss in classification accuracy (%) of DC-SVM, CP-SVM and proposed distributed SVM with respect to LIBSVM on publicly available datasets.

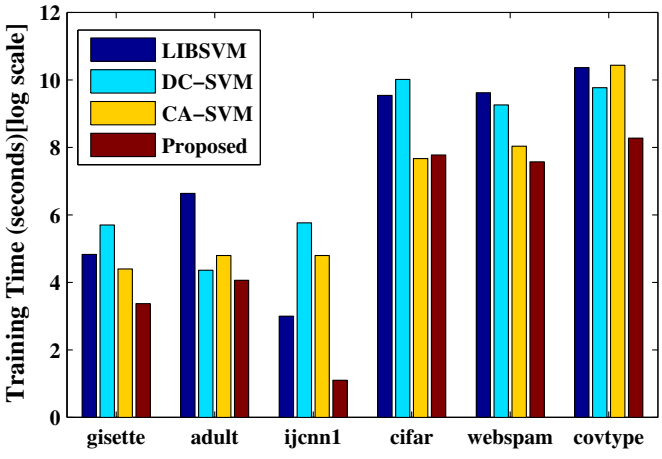


Fig. 16. A comparison of the training time (seconds) of LIBSVM, DC-SVM, CP-SVM and proposed distributed SVM on publicly available datasets.

suitable for mini-batch training of DiP-SVM algorithm. In order to train SVM on a large dataset, mini-batches are generated using DPP. The initial SVM model is trained using first mini-batch. Then, the distance from the decision boundary is calculated for each point in the second mini-batch using Equation (10) and only points closer to the decision boundary are selected and merged with the support vectors obtained in the previous step. For these new points, the Lagrangian multipliers are set to zero whereas for the existing support vectors, the old values are retained. Fig. 18 presents the results of classification performance for the mini-batch training of DiP-SVM, in which labels 1-10 on the x-axis correspond to 10 mini-batches, label 11 corresponds to distributed training of DiP-SVM and label 12 corresponds to the sequential SVM. It can be observed from the figure that using DPP, even the first mini-batch produces results which are quite close to the final classification results. Also, the accuracy increases as new points are introduced in incremental training. The use of previous Lagrangian multipliers and advanced elimination of irrelevant points before including them into the dataset for next SVM training also result in improved training time.

All the experiments performed with DiP-SVM confirm its suitability in comparison to existing approaches for distributed SVM training both in terms of scaling to large datasets and the performance of classification.

5 Conclusion

While distributed SVMs have generally proven to be much faster than sequential SVMs on large datasets, loss of classification performance and high communication overhead are still challenging issues. Through DiP-SVM, we aimed at solving both these issues by introducing a distribution preserving kernel SVM approach for a distributed environment. It was empirically shown that preserving the first

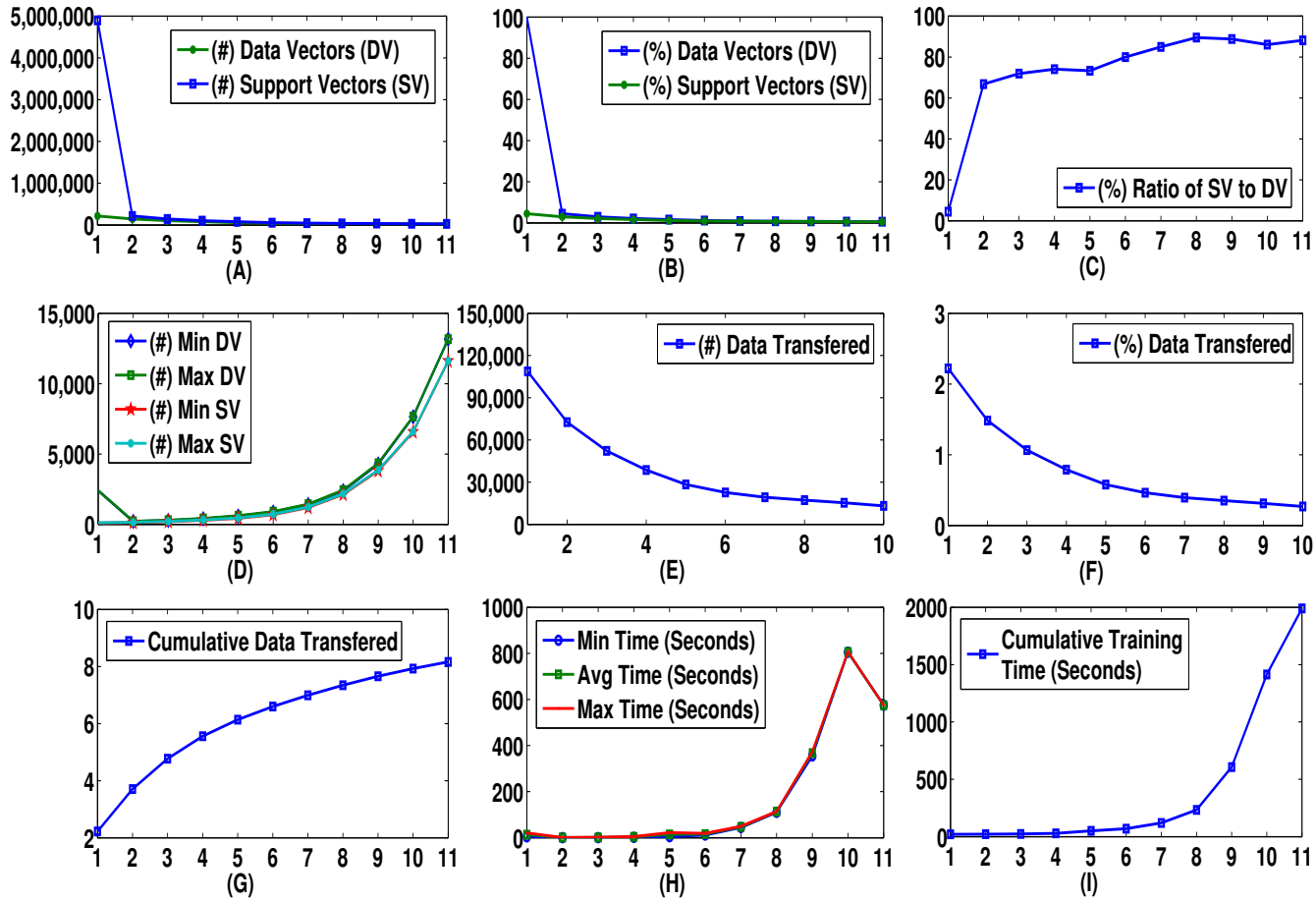


Fig. 17. Computational and communication efficiency of the proposed approach on the dataset kddcup99 consisting of $\approx 5M$ records.

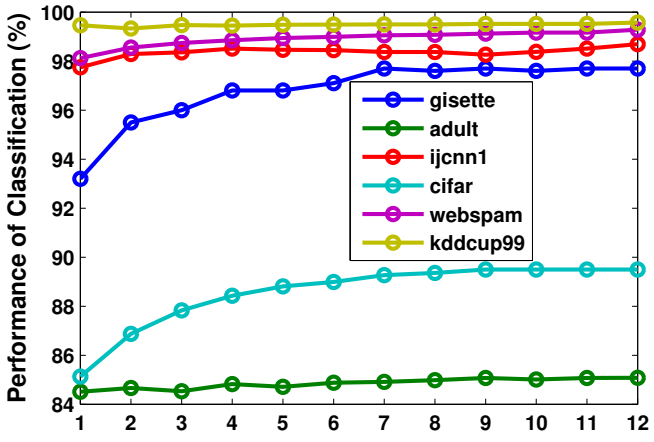


Fig. 18. Performance of classification for mini-batch training of DiP-SVM.

and second order statistics of the entire dataset in each of the partitions helped in obtaining local support vectors which were shown to be in agreement with the global decision boundary. This also helped the proposed approach to achieve comparable classification performance to a sequential SVM while having the computational gains of a distributed approach on benchmark datasets. A comparison with state-of-the-art distributed approaches revealed that owing to the better distribution of data, DiP-SVM performs at-par or better on all the datasets tested. We also showed that the communication overhead between partitions was greatly reduced making it suitable for high latency cloud environments.

REFERENCES

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

[2] X. Ke, R. Jin, X. Xie, and J. Cao, "A Distributed SVM Method based on the Iterative MapReduce," in *Proc. of the IEEE Int. Conf. on Semantic Computing (ICSC)*, Anaheim, CA, USA, 7–9 Feb 2015, pp. 7–10.

[3] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. on Intelligent Systems and Technology*, vol. 2, pp. 1–27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[4] T. Joachims, "Making large-Scale SVM Learning Practical," in *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 169–184.

[5] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core VectorMachines: Fast SVMTraining on Very Large Data Sets," *J. of Mach. Learning Res. (JMLR)*, vol. 6, no. 1, pp. 363–392, 2005.

[6] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca, and A. Lumsdaine, "Open MPI: A high-performance, heterogeneous MPI," in *Proc. of the IEEE Int. Conf. on Cluster Computing*, Barcelona, Spain, 25–28 Sep 2006, pp. 1–9.

[7] "Amazon Elastic Compute Cloud (Amazon EC2)." [Online]. Available: http://aws.amazon.com/ec2

[8] J. C. Platt, "Fast training Support Vector Machines using parallel sequential minimal optimization," in *Proc. of the Int. Conf. on Intelligent System and Knowledge Engineering*, 2008, pp. 997–1001.

[9] C.-J. Hsieh, S. Si, and I. Dhillon, "A Divide-and-Conquer Solver for Kernel Support Vector Machines," in *Proc. of the Int. Conf. on Machine Learning (ICML)*, vol. 32, no. 1, Beijing, China, 21–26 Jun 2014, pp. 566–574.

[10] Z. Sun and G. Fox, "Study on Parallel SVM Based on MapReduce," in *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, 2012, pp. 16–19.

[11] N. K. Alham, M. Li, S. Hammoud, Y. Liu, and M. Ponraj, "A distributed SVM for image annotation," in *Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD)*, Yantai, Shandong, China, 10-12 Aug 2010, pp. 2983–2987.

[12] Y. You, J. Demmel, K. Czechowski, L. Song, and R. Vuduc, "CA-SVM: Communication-Avoiding Support Vector Machines on Distributed Systems," in *Proc. of the IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, Hyderabad, India, 25-29 May 2015, pp. 847–859.

[13] X. Wang and S. Matwin, "A Distributed Instance-weighted SVM Algorithm on Large-scale Imbalanced Datasets," in *Proc. of the IEEE Int. Conf. on Big Data (ICBD)*, 2014, pp. 45–51.

[14] L. Zanni, T. Serafini, and G. Zanghirati, "Parallel software for training large scale support vector machines on multiprocessor systems," *J. of Mach. Learning Res. (JMLR)*, vol. 7, pp. 1467–1492, 2006.

[15] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent," *J. of Mach. Learning Res. (JMLR)*, vol. 10, pp. 1737–1754, 2009.

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

[17] F. Ö. Çatak and M. E. Balaban, "CloudSVM: Training an SVM Classifier in Cloud Computing Systems," in *Pervasive Computing and the Networked World - Joint International Conference, ICPCA/SWS*, Istanbul, Turkey, 28–30 Nov 2012, pp. 57–68.

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

[19] N. K. Alham, M. Li, Y. Liu, S. Hammoud, and M. Ponraj, "A distributed SVM for scalable image annotation," in *Proc. of the Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD)*, Shanghai, China, 26-28 Jul 2011, pp. 2655–2658.

[20] 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.

[21] S. Herrero-lopez, J. R. Williams, and A. Sanchez, "Parallel multiclass classification using SVMs on GPUs," in *Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units - GPGPU '10*, Pittsburgh, PA, USA, 14 Mar 2010, pp. 2–11.

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

[23] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik, "Parallel Support Vector Machines : The Cascade SVM," *In Advances in Neural Information Processing Systems*, pp. 521–528, 2005.

[24] H. Yu, J. Yang, J. Han, and X. Li, "Making svms scalable to large data sets using hierarchical cluster indexing," *Data Min. Knowl. Discov.*, vol. 11, no. 3, pp. 295–321, 2005.

[25] X. Zeng and T. R. Martinez, "Distribution-balanced stratified cross-validation for accuracy estimation," *J. Exp. Theor. Artif. Intell.*, vol. 12, no. 1, pp. 1–12, 2000.

[26] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, 2012.

[27] "StarCluster." [Online]. Available: http://star.mit.edu/cluster/index.html

Dinesh Singh is currently pursuing Ph.D. degree in Computer Science and Engineering from Indian Institute of Technology Hyderabad, India. He received the M. Tech degree in Computer Engineering from the National Institute of Technology Surat, India, in 2013. He received B. Tech degree from R. D. Engineering College, Ghaziabad, India, in 2010. He joined the Department of Computer Science and Engineering, Parul Institute of Engineering and Technology, Vadodara, India as an Assistant Professor from 2013 to 2014. His research interests include machine learning, big data analytics, visual computing, and cloud computing.

Debaditya Roy is currently pursuing his Ph.D. in Computer Science and Engineering from Indian Institute of Technology Hyderabad, India. He graduated with a Master of Technology in Computer Science and Engineering from the National Institute of Technology Rourkela, India, in 2013. He received his Bachelor of Technology in Computer Science and Engineering from West Bengal University of Technology, Kolkata, India in 2011. His research interests include deep learning, feature selection, and visual intelligence.

Dr. C. Krishna Mohan received Ph.D. Degree in Computer Science and Engineering from Indian Institute of Technology Madras, India in 2007. He received the Master of Technology in System Analysis and Computer Applications from National Institute of Technology Surathkal, India in 2000. He received the Master of Computer Applications degree from S. J. College of Engineering, Mysore, India in 1991 and the Bachelor of Science Education (B.Sc.Ed) degree from Regional Institute of Education, Mysore, India in 1988. He is currently an Associate Professor in the Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, India. His research interests include video content analysis, pattern recognition, and neural networks.