IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Random Partition based Adaptive Distributed Kernelized SVM for Big Data

AMRIT PAL[1], (Member, IEEE), ABISHI CHOWDHURY[1], SATAKSHI[2], HUSNU S. NARMAN[3],(Senior Member, IEEE), ARKABANDHU CHOWDHURY[4], and MANISH KUMAR.[5], (Senior Member, IEEE)

[1]School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, India
[2]Sam Higginbottom University of Agriculture, Technology and Sciences, Prayagraj, India.
[3]Department of Computer Sciences and Electrical Engineering, Marshall University, Huntington, West Virginia, USA.
[4]Amazon, USA
[5]Department of Information Technology, Indian Institute of Information Technology Allahabad, Prayagraj, India.

Corresponding author: Amrit Pal (e-mail: er.amritkabir@gmail.com).

ABSTRACT In this paper, we present a distributed classification technique for big data by efficiently using distributed storage architecture and data processing units of a cluster. While handling such large data, the existing approaches consider specific data partitioning techniques which demand complete data be processed before partitioning. This leads to an excessive overhead of high computation and data communication. The proposed method does not require any pre-structured data partitioning technique and is also adaptive to big data mining tools. We hypothesize that an effective aggregation of the information generated from data partitions by subprocesses of the complete learning process can lead to accurate prediction results while reducing the overall time complexity. We build three SVM based classifiers, namely one phase voting SVM (1PVSVM), two phase voting SVM (2PVSVM), and similarity based SVM (SIMSVM). Each of these classifiers utilizes the support vectors as the local information to construct the synthesized learner for efficiently reducing the training time and ensuring minimal communication between processing units. In this context, an extensive empirical analysis demonstrates the effectiveness of our classifiers when compared to other existing approaches on several benchmark datasets. However, among existing methods and three of our proposed (1PVSVM, 2PVSIM, and SIMSVM) methods, SIMSVM is the most efficient. Considering MNIST dataset, SIMSVM achieves an average speedup ratio of 0.78 and minimum scalability of 73% when the data size is scaled up to 10 times. It also retains high accuracy (99%) similar to centralized approaches.

INDEX TERMS Distributed learning, Large datasets, SVM, Classification, Distributed processing, Distributed storage

I. INTRODUCTION

Support vector machine (SVM) works on the principle of maximizing the margin between two classes using a segregating hyperplane [1]. Maximization of margin is an optimization problem which is solved applying the quadratic programming (QP) problems subject to linear constraints. The support vectors of one class act as its representatives and are used as boundary points for margin maximization from support vectors of other classes. SVM is extensively used to solve several real world problems [2][3][4][5] , such as face and object detection, fault prediction, handwriting recognition, image segmentation, protein fold and remote homology detection, e-learning, intrusion detection, speech recognition, and so on [6] through classification and regression analysis for low to average volume datasets. However, it is still challenging for SVM to deal with big data or large datasets [7], like, consider the associated challenges when SVM handles a large dataset:

Challenge 1. Large dataset:

Training time of standard SVM is $O(N^3)$ and space complexity is $O(N^2)$, where, N is the number of training data points. Therefore, it is an obvious challenge when SVM is trained for large datasets [8]. A distributed approach can help in overcoming these challenges by distributing the training task over parallel processing units as shown in Fig. 1.

Challenge 2. Distributed modeling:

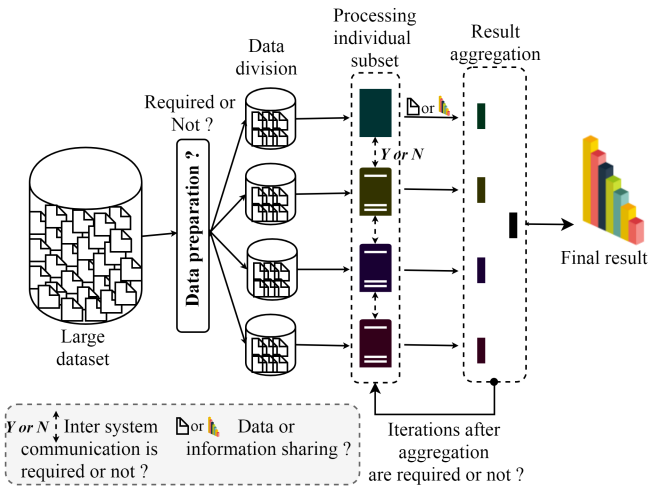


FIGURE 1: An overview of distributed learning and challenges associated while handling large datasets.

Distributed solutions like MapReduce [9], Spark [10], Flink [11], etc. are widely adopted to provide efficient solutions for big data analytics [12]. The overall upshot of these solutions is that multiple processing units are working in parallel over a distributed storage of data. The inherent challenge in using such solutions is the modeling of complete data analysis problem into subproblems with minimum dependencies among them. High independence among subproblems results in high scalability and speedup of the overall process. The optimization, done for margin maximization in SVM, is not suitable for such models because of high dependencies between their parameters [1].

Challenge 3. Data preparation:

Though the above mentioned distributed solutions are available to process large data up to a satisfactory level, the data first requires an efficient pre-processing before it is partitioned as shown in Fig.1. Performing pre-processing over complete data or independent subsets of the large volume data is again a challenge [13] in terms of computational cost and memory requirements, for example, performing a pre-clustering of the data before applying the prediction algorithm.

Challenge 4. Data transmission:

Efficient modeling of the complete analysis process into subproblems depends on the independence among the subproblems. The challenge is to reduce the transmission [14][15] of data to the subproblems, and between the subproblems.

The solution is to model the complete learning process over a large dataset into a distributed pipeline scenario in which results from independently and distributively running learners are used to generate the final learning results. The advantage achieved by this type of solution is a reduction in time complexity because of the distribution of data processing over a cluster. Here, cluster is a set of nodes where each node contains a processing unit and a distributed storage space.

Revisiting all the above mentioned challenges, in this paper, we propose three different solutions for distributed learning using SVM: i) one phase voting SVM (1PVSVM), ii) two phase voting SVM (2PVSVM), and iii) similarity based SVM (SIMSVM) for performing classification over large datasets. Each of the proposed schemes considers that data is distributed over a distributed space in form of blocks or chunks. We design our approaches to accommodate the cluster computing scenario where some blocks are locally available to the processing units. The complete proposed model consists of 4 phases; each phase maintains total independence among the parallel executing subproblems by ensuring no data or information exchange among them. An effective intermediate result aggregation scheme is also proposed to aggregate the results of subproblems. This aggregation further guarantees that there are no anomalies in the final aggregated result using a deduplication method. Further, our approaches do not require any specific data partitioning scheme and is adaptive to commonly used distributed storage solution like Hadoop Distributed File System (HDFS).

Contributions: Some of the key features of the proposed distributed and scalable approaches are as follows:

- The approaches allow random data partitioning, i.e., these approaches do not require specific partitioning technique to be applied before distributing data over a distributed storage.
- It can handle a large amount of data by efficiently dividing the learning process over the disjoint subsets of data.
- Three distributed solutions, namely, 1PVSVM, 2PVSVM, and SIMSVM are proposed to train the classifier using linear and non-linear kernels. The techniques have the same accuracy as the native SVM. Moreover, the techniques show a good amount of speedup and are highly scalable with low training time.
- The methods are instinctively adaptive to clustered or cloud solutions for large data analysis.
- No exchange of data is required between two distributed components resulting in low communication overhead.

Roadmap: The immediate section II demonstrates the need of support vector machine to handle large datasets. The overall description of the proposed approaches is described in section III where section IV explains how to handle multiclass classification problem through the proposed approaches. The ensuring section V comprises the experimental results and analysis on several benchmark datasets to validate the efficacy of the proposed approaches. Section VI indicates some key observations from this research work with some important research implications. Finally, section VII concludes the paper with the final remarks.

II. SVM FOR LARGE DATASETS

SVM is one of the most efficient ways of classification based on statistical learning theory and thereby is applied to several applications [16] of binary classification and regression analysis. However, there are certain challenges of applying

SVM over large datasets. Targeting these challenges, ample research has been conducted in recent years [17], [18], [19], [20], [21].

To apply SVM over large datasets for big data analytics, the available approaches can be broadly classified into three prominent categories. First category emphasizes on efficiently optimizing the margin maximization problem. These approaches do not achieve much speedup and scalability required for large datasets. [22] suggested decreasing the running time of quadratic programming solver which is used in SVM for optimization by shrinking and caching mechanism. The running time of this quadratic programming can also be decreased by solving the quadratic optimization problem matrix in a distributed manner by dividing it into smaller problems [23], and then combining the results of these problems. However, these approaches lack learning from large datasets due to their less adaptability to distributed modeling as the submodules generated from the complete learning process are not independent of each other. In contrast, our proposed approaches assure significant independence among the submodules of the learning process by efficiently employing the local learners on a subset of data. Moreover, reducing the number of data points for margin maximization can also decrease the overall training time. Based on this observation, [24] proposed a sampling-based technique, in which, some active points (near to support vectors) are selected instead of using the complete dataset for margin maximization. Selection of active points in case of non-linearly separable multiclass data is difficult, on the other hand, our proposed approaches do not require any predefined sampling technique.

The second class of approaches adeptly utilize the kernel function to reduce the overall training time as the kernels are used for distance calculation and are the essential part of margin maximization process. Here, kernel executions are performed in parallel or by using high performance systems like GPU. [25] suggested breaking the kernel function used for distance calculation into smaller computing blocks which can be solved in parallel by executing concurrent threads for each block. Choosing an appropriate kernel function can also reduce the overall training time; this concept is used by [26] in their proposed kernel selection method which selects a best suited kernel function for a given dataset and then SVM is applied using that kernel method. Generation of computing blocks and kernel selection require a pre-scan of the data. Further, these approaches are not suitable for a distributed storage of data. In the proposed solutions, the kernel function is integrated with each submodule of the learning process which performs the learning task over a subset of the data. This process automatically distributes the computation overhead of the kernel function over different submodules.

To further optimize the distance margin maximization process, framed inputs are provided to the SVM. This results in the third type of techniques. In this type of approaches, initially, clustering of the complete dataset is performed and then SVM is applied on the clustered data. The training time can be diminished by performing the division of data such that a *pre-clustered* data is present in each partition and is used as an input to the classifier for classification. In a distributed manner, different clustering algorithms can be applied such as k-means which divides the data into k clusters [27] before feeding the data into SVM. CSVM [28] uses k-means algorithm to classify the big data into k independent clusters. Then learning is performed using weighted local SVMs trained in the cluster. To reduce the communication cost, communication avoiding support vector machine (CASVM) [29] uses a balanced k-means clustering to reduce the data communication. This technique performs better when the data is non-overlapping and performance degrades for overlapping datasets. Another approach, DTSVM [30], applies decision tree algorithms to classify the large data into k disjoint regions (tree leaves). A local SVM is used to classify the regions having heterogeneous points (in context of class labels). So, for any test point, first, decision tree is used to select the appropriate region and then, SVM is used for final class prediction. Using decision tree algorithm, [31] suggested a classification method that identifies low entropy data regions. Later, Fisher's linear discriminant is applied to reduce the overall data by only selecting data from decision tree boundaries. This reduced data is further used by SVM for generation of the separating planes. [32] [33] proposed a distributed SVM approach which is a combination of k local SVMs. The complete data is partitioned into k-clusters and local SVMs are applied over each cluster of the data. A difference in[33] approach is that it follows a balanced clustering method for load balancing among the subproblems. One vital issue with this third category is that these methods require pre-clustering of the complete data that means complete data must be present at one place which in turn increases the complexity of the overall process. *However, our proposed methods do not have such centralized data requirement and thus, can save such data preparation time.*

III. DETAILED DESCRIPTION OF PROPOSED APPROACHES

Problem Statement: Large datasets are often stored in small-sized partitions over a distributed storage. Removal of the overhead of any special partitioning, which preserves the centrality of data over a distributed storage, is a challenge for large data mining. This work addresses the challenges in extracting the relevant information from randomly partitioned dataset subsets and performing the aggregation of this information without any duplication to generate the complete information for classification while maintaining the accuracy and minimizing the training time.

Foundation: The basis of the proposed approaches is that SVM aims to find the data points belonging to different classes that are the nearest to one another. These data points on the gutter determine the support vectors. The support vectors determine the maximum margin between the classes and define the decision boundary in SVM. It can be observed

that data points on the gutter that do not define the decision boundary and those data points that are far from the decision boundary do not contribute to the separating hyperplane. For large datasets, this entails that data partitioning based on distribution preserving sampling does not help as only data points near the decision boundaries are important. Trivial partitioning of a dataset can suffice. Moreover, only support vectors for each partitioned dataset subset are required for finding the final support vectors that define the decision boundary for the original dataset. The extraction of support vectors from each partitioned dataset subset is independent of each other. Hence, no communication is required between different processing units running SVM. This provides the notion to design our classification model.

Design: The complete dataset is partitioned into a number of subsets using data partitioning. Each subset of data is considered located on different machines, and it is expedient to be processed locally. Apparently, higher independence among local processing modules results in higher scalability in the overall learning process. On the whole, the goal of our intended approach is to execute *independent learners* over *slave nodes* in a *cluster*, and develop a *synthesized learner* using a *distributed pool* as a storage area.

- ***Independent learner:*** It processes an available partition D_i from the data and produces intermediate leaning results. The learning process is independent of both the other partitions and other learners.
- ***Synthesized learner:*** A collection of potential independent learners together form a synthesized learner, which integrates the results from independent learners to yield the final learning results.
- ***Cluster:*** It is combination of processing nodes which have access to a distributed file system and are connected through a network switch.
- ***Distributed pool:*** It is a shared distributed space available to all the nodes in the cluster.
- ***Slave nodes:*** Independent processing units of the cluster or simply nodes of the cluster having processing and storage capability. Virtually distributed pool is a combined form of slave nodes' storage space (like in HDFS).

Our proposed prediction model is an ensemble of independent learners, which is pertinent enough for a distributed processing and storage environment. The entire learning process is instrumented using parallel independent learners which are applied to each block of data in order to extract intermediate results. The synthesized learning is performed over a cluster where each independent learner works as a slave process that utilizes the processing capability of a node in the cluster. This approach is adaptable to several big data processing frameworks like Hadoop MapReduce, Spark, etc. The overall approach is divided into four phases:

- Data partitioning
- Distributed intermediate information generation from different data partitions
- Information aggregation and deduplication process

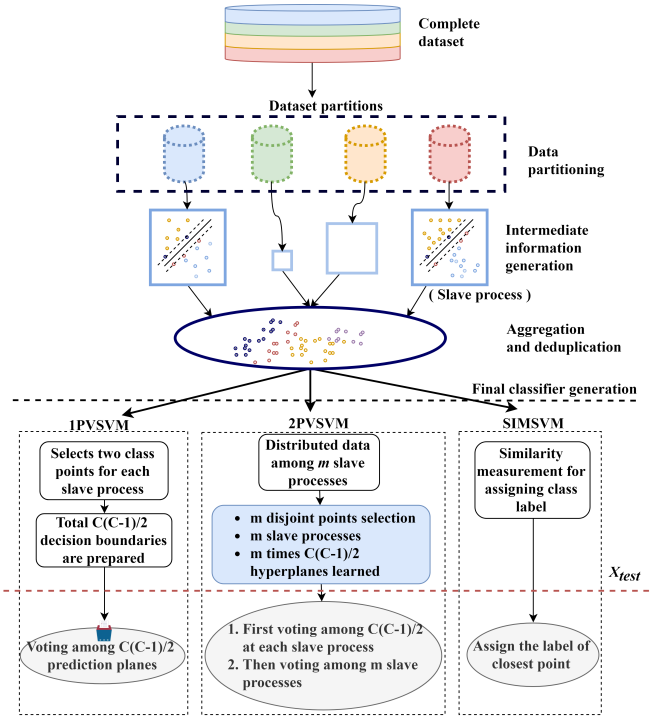


FIGURE 2: Four-phased proposed model.

- Final classifier generation
  - -- One phase voting SVM (1PVSVM)
  - -- Two phase voting SVM (2PVSVM)
  - -- Similarity based SVM (SIMSVM)

The four phases as shown in Fig. 2, the *first phase* is data partitioning in which complete data is divided into m subsets. The *second phase* is executed separately over each partition or a collection of partitions of data. The partitioned data is used to generate the intermediate results in form of support vectors which are compressed boundary points for class labeling. In the *third phase*, the aggregation of the results generated in second phase is performed using a deduplication process which removes the duplicate results. Finally, the fourth phase is executed to develop the three types of classifier either by applying any voting scheme (one phase, or two phase) or by applying similarity based approach. Table 1 depicts some commonly used notations in the paper.

A. FIRST PHASE: DATASET PARTITION

One of the core challenges with large datasets is that, it is hard to process the complete data in a single iteration because of limited storage and processing power of a system. Therefore, data partitioning is an effective way to deal with such massive datasets. In our approach, the complete dataset D is divided into m small blocks/data chunks as $\{D_1, D_2, ..., D_m\}$ and stored over a distributed storage. The analysis over this distributed data mimics a non-uniform sampling process. The entire dataset D is given by:

$$D = \{(x_i, y_i)\}^N \tag{1}$$

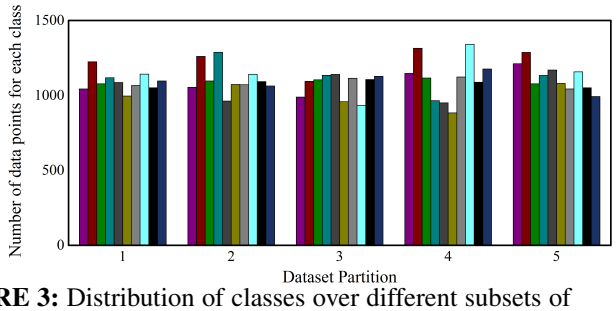


FIGURE 3: Distribution of classes over different subsets of MNIST dataset.

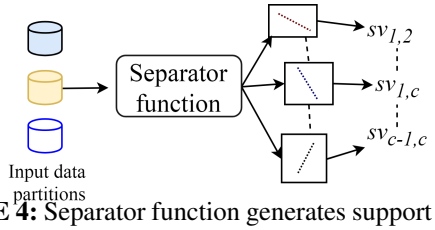


FIGURE 4: Separator function generates support vectors for each combination of the classes in phase 1.

Where, x_i is the i^{th} feature vector and y_i is its class label, the division of D in m non-overlapping subset is expressed as:

$$D_j = \{(x_i, y_i)\}^{N_j}; \quad j = 1 \; to \; m \tag{2}$$

Where, N_j is the sample size for j^{th} partition. However, disjoint sets yield a smaller number of data partitions. So, any random partition of the data can contain a random number of instances of different classes. Thus, the distribution of data is adaptive to real time distributed data processing techniques [10][34]. For example, applying this data partitioning technique over MNIST [35] dataset, when divided among five blocks, shows an imbalanced distribution of classes over each subset as shown in Fig. 3. Each of these blocks is processed in a distributed manner using proposed approaches to generate the final results.

B. SECOND PHASE: DISTRIBUTED INTERMEDIATE INFORMATION GENERATION

Each independent learner performs this phase and takes one single block or multiple blocks as input and considers it as a single data partition. Suppose, there are k number of data points in a partition $D_i \in \{D_1, D_2, ...D_m\}$. In this phase, a *separator function* (independent learner) attempts to identify an optimal hyperplane that separates two classes and collects all the points existing on the surface of the hyperplane (support vectors) as shown in Fig. 4. The classification function for a set of data points using *separator function* is given by equation 3.

$$\begin{aligned} W^t x_j + b >= 1 \quad & for \; y_j = +1 \\ W^t x_j + b <= 1 \quad & for \; y_j = -1 \end{aligned} \tag{3}$$

Where, x_j is a vector which represents a data point in D_i input space, $y_j \in \{+1, -1\}$ is the target class label for j^{th} data point, b is a scalar which is a bias term, and W is the coefficient weight vector corresponding to the hyperplane separating two classes.

TABLE 1: Common notations with descriptions

Notation	Description
W	Weight vector
D	Large dataset(Initial)
D_i	i^{th} partition of dataset D
m	Number of data partitions
C	Number of classes in the dataset
sv_i	Complete support vectors generated from D_i data partition
$sv_i(j,k)$	List of support vectors of class j and k generated from data partition D_i
$SV(j,k)$	Collection of all support vectors of j and k classes
SV	$\bigcup_{i=1,j=1,k=i+1}^{m,C,C} SV_{i,j,k}$
y	Class of of any single point {1,-1}.
A	Matrix of dataset D_i without label.
x_i	i^{th} Feature vector
X_{test}	A test point
Y	Diagonal matrix of y
ζ	Soft margin parameter
α	Matrix of Lagrangian multiplier.
$K(x_i, x_j)$	Kernel function.

The exploration of all the hyperplanes along with the corresponding support vectors from each partition is done using constrained optimization originally suggested by Vapnik [1]. Suppose, there exists a hyperplane $W^t x + b = 0$ then, the following equation will always remain true:

$$Y_i(W^t x_i + b) >= 1, \quad \text{for all } i \tag{4}$$

For each point which exists on the hyperplane of any of the classes, the following equality holds:

$$Y_i(W^t x_i + b) = 1 \tag{5}$$

For each *independent learner* executed over the set of data partition (D_i), same margin is considered. The margin maximization is the maximization of distance $\left(\frac{2}{||W||}\right)$ between two classes which is the same as minimizing $\frac{1}{2}||W||$. The margin minimization function for each partition is given in equation 6 in case of hard margin and 7 in case of soft margin.

$$\begin{aligned} & \min_{W,b} \frac{1}{2}||W||^2 \\ \text{subject to} \quad & Y_i(W^t x_i + b) - 1 \geq 0, \quad i = 1, 2, ..., k \end{aligned} \tag{6}$$

$$\begin{aligned} & \min_{W,b,\zeta} \left(\frac{||W^2||}{2} + \sum_{i=1}^{k} \zeta_i \right) \\ \text{subject to} \quad & y_i(W^t x_i + b) \geq 1 - \zeta_i, \quad i = 1, 2, ..., k \end{aligned} \tag{7}$$

Where, $\zeta \geq 0$ and a regularization parameter r is added to regularize ζ. This leads to the soft-margin formulation as follows:

$$\begin{aligned} & \min_{W,b,\zeta} \left(\frac{||W^2||}{2} + r \sum_{i=1}^{k} \zeta_i \right) \\ \text{subject to} \quad & y_i(W^t x_i + b) \geq 1 - \zeta_i, \quad \zeta \geq 0 \end{aligned} \tag{8}$$

This optimization problem is solved using Lagrangian dual form. This same problem can be written in matrix form as:

$$\min_{W,b} \frac{W^t W}{2} \quad \text{subject to} \quad P(AW - eb) \geq e \tag{9}$$

Where, e is a $k \times 1$ vector of ones. Applying Lagrangian formulation, we get the dual optimization problem as:

$$\max L_Y(\alpha) = \alpha^t e - \frac{1}{2}\alpha^t Y A A^t Y \alpha \quad \text{subject to} \quad e^t Y \alpha = 0 \;\; \alpha >= 0 \tag{10}$$

This optimization problem can be solved using the standard quadratic programming method and the result for matrix α is obtained. sv_i denotes the set of support vectors which fall on the hyperplane (or the data points for which $0 < \alpha_i < r$, where, r is the error-penalty multiplier). Algorithm 1 describes the process of SV generation from different data partitions. Each SV entry consists of a pair of support vectors and the corresponding class label.

Algorithm 1 Intermediate information generation from dataset partition

1: $D = \{D_1, D_2..., D_m\}$ and j, k are two classes
2: $SV = [\,]$
3: **for** $i \leftarrow 0$ *to* $m - 1$ **do**
4: $\quad sv_{j,k} = [\,]$
5: $\quad sv_{j,k}$=svm(D_i)
6: $\quad SV(j,k)$.append($sv_{j,k}$)
7: **end for**
8: **add** $SV(j,k)$ to storage pool

This algorithm is applied to all the D_i in parallel and the generated sv_i are appended to SV. Equation 11 shows a simple structure of results generated by this phase as $sv_i(j,k)$, which is a set of support vectors for class j and k while considering the hyperplane between these two classes.

$$sv_i = \begin{bmatrix} & & \\ sv_{2,1}\{\} & & \\ \vdots & & \\ sv_{k,1}\{\} & \ddots & \\ \vdots & \vdots & \\ sv_{C,1}\{\} & . & sv_{C-1,C}\{\} \end{bmatrix} \tag{11}$$

Thus, all the generated sv_i are added to the distributed storage pool and further, the aggregation step is performed to generate the final accumulated results.

C. THIRD PHASE: INFORMATION AGGREGATION AND DEDUPLICATION

Each *separator function* procreates a compressed set of points $sv_i(j,k)$ while being executed on a slave node. This phase takes these generated points $\{sv_1, sv_2, ...sv_m\}$ as input and performs the tasks of deduplication and aggregation as shown in Fig. 5.

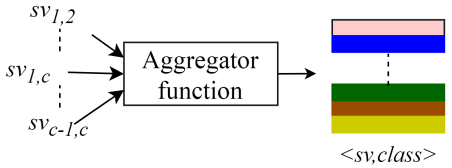


FIGURE 5: Aggregator function takes extracted sv as input and performs deduplication and generates aggregated $< sv, class >$ pairs.

1) Duplicate data

The assemblage of compressed points can have duplicate entries for a point because one support vector of a class can be engaged as a boundary point for multiple classes. For example, a point p which is associated with class c_i and contributes as a boundary point for hyperplanes $\{h_{ij}, h_{ik}, and\, h_{il}\}$ those classify the i^{th} class from $\{j,k,l\}$ classes. So, after the completion of second phase, the distributed pool will contain three entries for the point p with its class c_i. This condition results in an exponential growth in the number of support vectors in the distributed pool. In order to handle this type of overflow situation, deduplication is an obligatory step.

2) Aggregator function

The detection and elimination of duplicate points are done using a $< key, value >$ (like in MapReduce) pair which is basically a combination of support vector and its class label. Results from all the slaves are aggregated based on a $< sv\{\}, class >$ pair (using equation 12). This process continues until there is more than one $sv\{\}$ with the same class pair.

$$SV(j,k) = \bigcup_{i}^{m} sv_i(j,k) \tag{12}$$

Thus, the aggregator function forms a final set of points after deduplication process which can be used for class level predictions.

D. FOURTH PHASE: FINAL CLASSIFIER GENERATION

The final phase of our proposed method is performed in three different ways, namely i) one phase voting SVM (1PVSVM), ii) two phase voting SVM (2PVSVM), and iii) similarity based SVM (SIMSVM). 1PVSVM, and 2PVSVM include generation of final cumulative hyperplanes by using a reduced set of points after deduplication process. The primary differences between these two approaches are input selection process, and counting process. The SIMSVM is a similarity based approach which uses a similarity measure to decide the class label for a test point.

1) One phase voting SVM (1PVSVM)

In this scheme, a set of reduced points ($SV(j,k)$) containing all points belonging to any two classes is used by each slave process and cumulative hyperplanes are generated for final prediction.

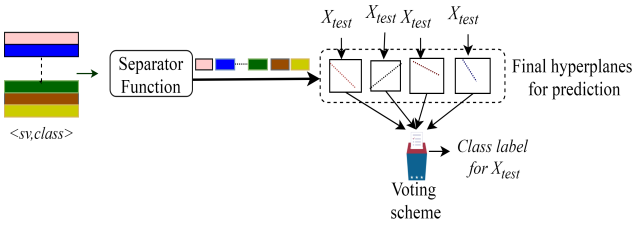


FIGURE 6: Incremental SVM hyperplane generation using separator function from aggregated results and one phase voting scheme.

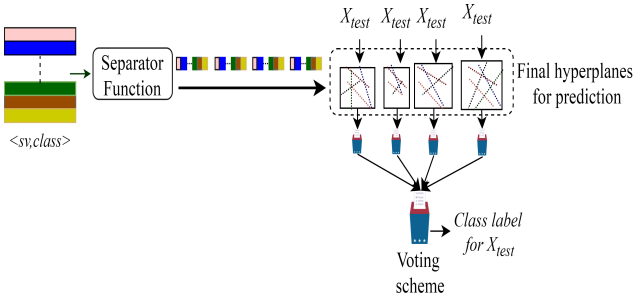


FIGURE 7: Incremental SVM hyperplane generation using separator function from aggregated results and two phase voting scheme.

a: Cumulative hyperplane generation

In this technique, each slave process applies a *separator function* to generate the cumulative hyperplanes for different combinations of the classes as shown in Fig. 6. The set $SV(j,k)$ is used as an input to separator function which generates the final separating plane for class j and k. This process of cumulative hyperplanes generation is similar to phase 2 (subsection III-B) till equation 11. For different pairs of classes, weight vector W and bias term b, the coefficients of hyperplanes, are retained instead of sv_i. Thus, the generated hyperplanes are used for classification of a test point X_{test} using the following classification function:

$$\begin{aligned} f(X_{test}) &= sign(W_t X_{test} - b) \\ f(X_{test}) &= sign[(A^t Y \alpha)^t - b] \\ or,\ sign(\alpha^t Y A X_{test} &- b) \end{aligned} \tag{13}$$

In case of a multiclass scenario, using W and bias term b, classification of a test point according to equation 13 is done using a maximum voting strategy.

Property: For C number of classes, only $\frac{C(C-1)}{2}$ number of slave computations of *separator function* are required which is adaptive to parallel processing.

Now, to classify the test point X_{test}, a majority vote among $\frac{C(C-1)}{2}$ classification results is taken as shown in Fig. 6.

2) Two phase voting SVM (2PVSVM)

Unlike 1PVSVM scheme, 2PVSVM considers disjoint subsets (= no of slave nodes) of the reduced points, which may contain points belonging to different classes, are used for cumulative hyperplane generation process. For each separator function, the weight vector W and bias term b are retained for final predictions.

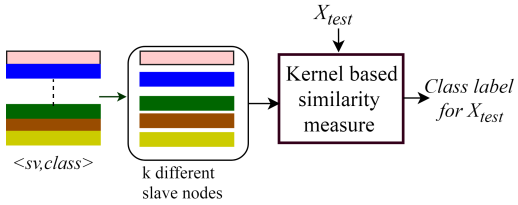


FIGURE 8: Labeling of X_{test} using similarity measure.

Any X_{test} point is tested by using a two phase voting scheme as follows:

1) First, the voting is performed within the $\frac{C(C-1)}{2}$ hyperplanes (C = no of classes) generated by a single separator function.
2) Second, the voting is performed within the separator functions for final classification.

Fig. 7 shows the overall process of 2PVSVM which clearly states the differences from 1PVSVM in context of input section process and the counting process for class level generation.

3) Similarity based SVM (SIMSVM)

1PVSVM and 2PVSVM carry out several executions of the complete SVM as both these approaches require to construct the decisive separating planes from the reduced points which increases the high computation requirement in this phase. To prevail this computational overhead, we propose another approach, similarity based SVM (SIMSVM), which exploits reduced points and their class labels from the distributed pool $(< x_1, y1 >, < x2, y2 >, ... < x_z, y_z >)$ as input and predict the class labels for the test points as shown in Fig. 8. Algorithm 2 demonstrates the complete pseudocode for predicting class label for a test point X_{test}.

Algorithm 2 Label prediction for X_{test} by SIMSVM

X_{test} : A test point
$TotalSV = \sum_{i=1}^{\frac{C(C-1)}{2}} len(SV_i)$
define $K(X_i, X_j)$ **as** $\exp(-\frac{||X_i - X_j||^2}{2\rho^2}$
for $u \leftarrow 0$ to $len(test)$ **do**
  MinDist=INTMAX
  Minindex=0
  x_i=X_{test}
  for $v \leftarrow 0$ to $len(TotalSV)$ **do**
    x_j=TotalSV[v]
    dist=$\sqrt{K(x_i,x_i) + K(x_j,x_j) - 2K(x_i,x_j)}$
    if $MinDist > dist$ **then**
      MinDist=dist
      Minindex=v
    end if
  end for
end for
return Class of vector on v^{th} $Minindex$

The classification of a test point X_{test} is done based on a similarity measure between the existing set of reduced points in the distributed pool each of which is associated with a class label. SIMSVM scheme estimates the distance between two points as a similarity measure and test point X_{test} is classified into the class of that reduced point which has the minimum distance from it. Suppose, for a test point X_{test}, the class label y_{test} need to be predicted. Now, y_{test} is labeled as $y_{test} = y_m$ if X_m is the closest point to X_{test} and y_m is the label for X_m point.

a: Distance functions

To handle non-linearly separable data, transformation of the data is required in a higher dimension. It is evident that the use of kernel function for measuring the distances in higher dimension provides better separating hyperplanes as kernel requires only the inner product between two vectors in the higher dimension. This property of the kernel function is known as kernel trick [36]. Inner product in the transformed linearized space is same as taking the inner product in the original space. This solution contributes a non-linear variant for linear similarity measure. For example, a transformation from $\mathbb{R}^2$ to $\mathbb{R}^3$ for a vector z can be done using a linearization function ϕ as follows:

$$\phi(z) = \begin{pmatrix} \phi_1(z) \\ \phi_2(z) \\ \phi_3(z) \end{pmatrix} = \begin{pmatrix} z_1^2 \\ \sqrt{2} z_1 z_2 \\ z_2^2 \end{pmatrix} \tag{14}$$

This is a simple example of a polynomial kernel in the form $k(x, y) =< x, y >^3$. This transformation is equivalent to the inner product of vectors in the two dimensional space. Suppose, given two vectors x_i and x_j, kernel function $K(x_i, x_j)$ is defined as the inner product of $\phi(x_i)$ and $\phi(x_j)$, where, ϕ is the basis function that maps the vectors x_i and x_j from R (original space) to H (higher dimensional space). The distance between two vectors can be calculated as:

$$d(x_i, x_j) = ||\phi(x_i) - \phi(x_j)||^2 \tag{15}$$

$$= \sqrt{K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j)} \tag{16}$$

We use Gaussian kernel as shown in the equation 17 to find the distance between two vectors using kernel trick which also uses the dot product of the vectors in the original space and is equivalent to explicitly determine the coordinates of data vector from R space to the H space.

$$k(x_i, x_j) = exp^{-\frac{||x_i - x_j||^2}{\sigma^2}} \tag{17}$$

SIMSVM uses this kernel to estimate the distance between a test point and a labeled point.

IV. HANDLING MULTICLASS CLASSIFICATION PROBLEM

SVMs are able to generate binary classifiers, however, large datasets often contain more than two classes. There can be several ways to handle multiclass classification using SVM. In case of our approaches, we perform multiclass classification by dividing it into multiple binary classification problems.

A. ONE-AGAINST-ONE APPROACH

To accomplish multiclass classification, instead of trying to discriminate one class from all the others, this approach distinguishes one class from another one. As a result, it trains the model iteratively for different pairs of classes and generates the required hyperplanes and support vectors.

Algorithm 3 Multiclass distributed SVM calculation

1: $D = \{D_1, D_2..., D_m\}$
2: $SV = [\,]$
3: **for** $i \leftarrow 0$ to $m - 1$ **do**
4: **for** $j \leftarrow 0$ to $C - 2$ **do**
5: **for** $k \leftarrow j + 1$ to $C - 1$ **do**
6: $sv = svm(D_i(j, k))$
7: $sv_i[j][k].append(sv)$
8: $SV[j][k].append(sv_i[j][k])$
9: **end for**
10: **end for**
11: **end for**
12: **for** $i \leftarrow 0$ to $C - 2$ **do**
13: **for** $j \leftarrow i + 1$ to $C - 1$ **do**
14: $W, b = SVM(SV[j][k])$
15: $weight.append(W)$
16: $bias.append(b)$
17: **end for**
18: **end for**

Algorithm 3 demonstrates the complete one-against-one classification process. The loops in the 4^{th} and 5^{th} lines lead to $C(C-1)/2$ times calculation of support vectors for each dataset subset D_i and input data points of two classes at a time as $D_i(j, k)$. The sv_i is a matrix whose each individual element can be written as $sv_i(j, k)$ that denotes the support vectors of class j with the class k in i_{th} subset of the dataset. Similarly, we can find the m number of matrices (where, m is the total number of partitions of data). The dimension of each matrix is $C \times C$ which have elements only in the lower triangle and these matrices are combined as $\bigcup_{i=1}^{i=m} sv_i(j, k)$. This approach is used by each separator function while dealing with multiclass data.

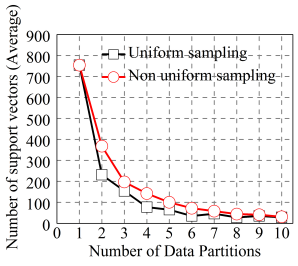
V. EVALUATION

We have conducted a rigorous experimental analysis in order to validate the efficacy of our proposed classifiers. The essential objectives of this analysis are to answer the following explicit queries while handling a large dataset in a distributed environment as follows:

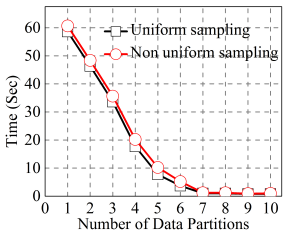
- How much intermediate information in form of support vectors is generated from different random data partitions?

TABLE 2: Dataset used for analysis

S.No.	Dataset	Domain	# Features	# Instances	# Classes
1	Two Moon [37]	Classification	2	10000	2
2	IRIS [38]	Classification	4	150	3
3	MNIST [35]	Digit Classification	784	60000	10
4	Adult [39]	Economics	14	48842	2
5	Synthetic Data	Classification	2	2000	10
6	MNIST-8m [40]	Classification	784	8000000	10
7	CIFAR [41]	Classification	3072	50000	10
8	Fashion-MNIST [42]	Classification	784	60000	10



(a) Analysis of average number of support vectors generated by the independent learners as the number of data partitions increases.



(b) Time required for generation of intermediate information as the number of partitions increases.

FIGURE 9: Analysis of information generation by the slave processes while training in phase 2.

- How much training time is required to train the proposed model using independent learners/separator function? What is the difference between the time required for training of the classifier by the proposed approach as compared to the existing approaches?
- What is the classification accuracy of the proposed approaches with respect to the existing approaches for different datasets?
- How the proposed approaches perform when used in an incremental manner?
- How much speedup and scalability the proposed approaches can achieve?

a: Comparables

We have compared the performance of our approaches, 1PVSVM, 2PVSVM, and SIMSVM against other existing approaches, LIBSVM [22], CASVM [29], and BCSVM [33] on different benchmark datasets shown in Table 2. LIBSVM is a state-of-the-art approach and CASVM is selected for its low communication requirements. BCSVM is the latest approach in this domain and uses pre clustered partitions of the data. To compare the proposed approach with pre-clustered partition based approach BCSVM is selected.

A. EXPERIMENTAL SETUP

There are several tools used in the literature for implementing distributed SVM like Apache Hadoop, spark, etc. To evaluate our proposed classifiers, we have used the MapReduce programming model for distributed programming and HDFS as a distributed pool for storing data and intermediate results. The proposed approaches are implemented on a 10 node cluster, where, each node has 2 CPU cores with $6GB$ of RAM and a frequency of 2.6 GHz. Each cluster node is capable of running two tasks in parallel. Our proposed approaches have been analyzed using different real-world datasets as shown in Table 2. The datasets are from different domains and are linearly separable/non-linearly separable.

B. EXPERIMENTAL RESULTS AND ANALYSIS

We have measured the various schemes based on a few important aspects: the training time and accuracy. To inspect the speedup of the proposed approaches, we calculate the change in training time when new resources are added and to analyze the scalability of the proposed approaches, we calculate the speedup when resources, as well as data size, is increasing. To observe the incremental adaptation, we estimate the accuracy with the addition of each new data chunk.

1) Data Reduction Analysis

The analysis, shown in Fig. 9, has been done creating several partitions of data for analyzing the amount of information generated by the separator function in phase 2 for Two Moon dataset. The complete dataset has been divided into 10 partitions and stored over a distributed storage from 1 to 10 in two ways using uniform sampling and non-uniform sampling of the data. Each subprocess (mapper) accesses these data blocks and uses Algorithm 1 to process it. After processing, it writes all the results back to the distributed storage in a clustered environment. In the proposed clustered environment, instead of processing the complete dataset, each slave node can work on its locally available partition of data. These slave nodes independently process these partitions, and the analysis shows the amount of information generated by slave processes from these partitions as the number of partitions increases.

The first analysis, shown in Fig. 9a, depicts that as the number of partitions increases the amount of data available for a local subprocess decreases which will also decrease the average number of generated support vectors by each

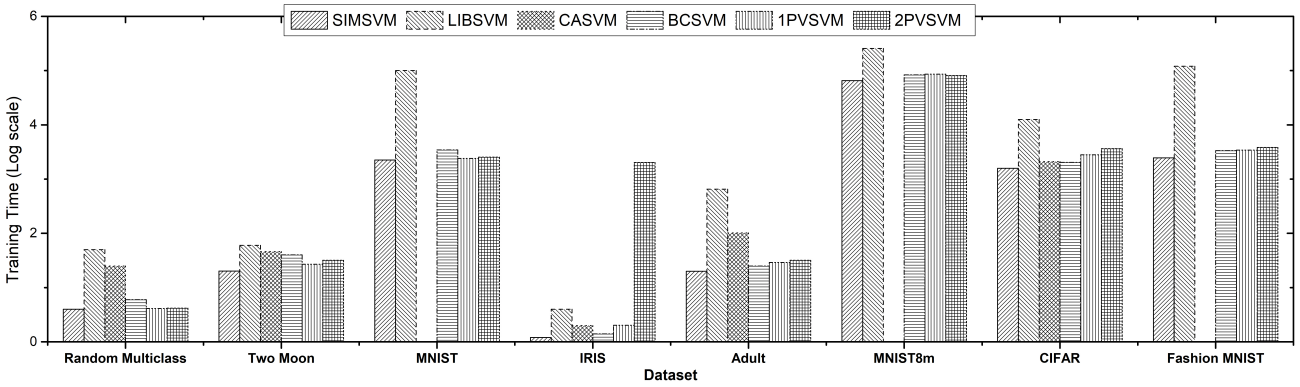


FIGURE 10: Training time (log scale) analysis for different datasets

TABLE 3: Training time (sec.) analysis of different approaches over different datasets

S.No.	Dataset	SIMSVM	LIBSVM	CASVM	BCSVM	1PVSVM	2PVSVM
1	Random Multiclass	4	50	25	6	4.11	4.18
2	Two Moon	20.2	60	46	40	27	32
3	MNIST	2255.4	100000	NA	3456	2400	2542
4	IRIS	1.2	4	2	1.4	2.02	2.015
5	Adult	20	650	102	25	29	32
6	MNIST8m	65265	256728	NA	83467	85634	81054
7	CIFAR 10	1585	12565	2101	2037	2802	3625
8	Fashion-MNIST	2458	120012	NA	3345	3425	3847

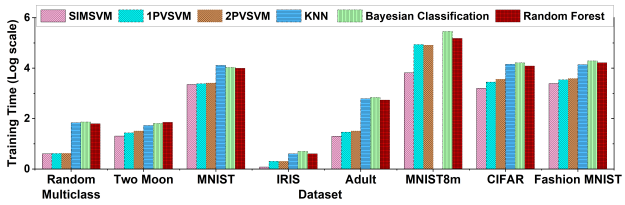


FIGURE 11: A comparative analysis of training time with respect to well known machine learning approaches

slave process. If we increase the number of partitions after 5, there is not much significant decrease in the average number of support vectors generated by each slave. This is because, after this point, these many support vectors are necessary for classification. Results indicate that in case of non-uniform sampling, the number of support vectors are slightly high, but due to its simplicity and adaptability, it is used for data partitioning for further analysis. The proposed approaches fetch the full advantage of the distributed storage and processing power. Fig. 9b illustrates that the amount of time required to process the complete data decreases while increasing the number of partitions. As we increase the number of partitions, we also add a processing unit to the existing data processing cluster, which processes this new partition and further results in decreasing processing time. Although, the decrement of time remains valid up to a certain number of partitions, like, in the considered case, it is up to 5^{th} partition. After this, there is no significant reduction in time because this amount of time is necessarily required to process even a very small amount of data. However, this time amount also depends on the power of each processing unit in the cluster.

2) Training Time Analysis

Our proposed classifiers have been trained using different datasets. The complete training time consists of the time required to execute the second, third, and fourth phase of the proposed model. Fig. 10 (shows a log scale training time due to a wide range of training time for different datasets) and Table 3 (shows the actual training time in second) present the comparison of training time taken by the proposed approaches with the existing approaches creating four partitions of each dataset. The results from the plots and the table clearly mark the significant improvements over the existing approaches. 1PVSVM performs better as compared to 2PVSVM except for MNIST8m dataset due to its efficient sampling of only two classes at once. Whereas, SIMSVM exhibits the best performance as compared to LIBSVM, CASVM and BCSVM because it efficiently distributes the learning process among several slave processes. SIMSVM gains over BCSVM, as clustering of the data consumes a significant amount of time and the difference can be observed in the Fig. 10. SIMSVM also shows improvements over 1PVSVM and 2PVSVM as it does not require the separator function to be executed over reduced data points. An analysis has also been conducted for comparing the proposed approaches with the existing machine leaning approaches [43]. We have compared the proposed approaches with random forest, Bayesian classification, and KNN algorithms. The Log scaled training time analysis is presented in Figure 11. The proposed approaches show significant gain in context of training time and accuracy over the existing machine learning approaches due to its distributed properties. The proposed 1PVSVM, 2PVSVM, and SIMSVM can scale with the increasing number of partitions and the time required for training decreases as number of partitions increases.

TABLE 4: Accuracy analysis of the different approaches using different datasets

S.No.	Dataset	SIMSVM	LIBSVM	CASVM	BCSVM	1PVSVM	2PVSVM
1	Random Multiclass	97.2	97.2	96	96	97	96.8
2	Two Moon	99.8	98.2	83	98.5	98	98
3	MNIST	99.9	99.9	NA	98.5	99	98.7
4	IRIS	99.2	99.3	96	99	98.8	98.4
5	Adult	85	85.06	83	84	85	84.5
6	MNIST8m	99. 9	99.9	NA	98	99	99
7	CIFAR	89	89	64	88	87	86.8
8	Fashion-MNIST	92.4	92.4	NA	90	92.2	92

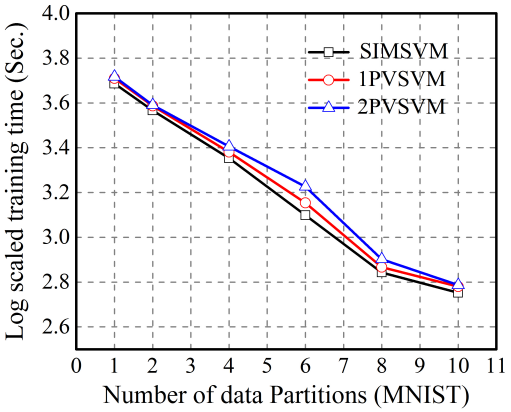


FIGURE 12: Log scale time to train the classifiers for MNIST dataset while increasing the number of partitions.

Fig. 12 plots this analysis for MNIST dataset; as the number of partitions increases less number of input points are available for intermediate information generation in phase 2, hence training time decreases. The number of partitions corresponds to parallel running of slave processes in each phase of the proposed approaches. These parallel running processes are modeled using mapper functions of the MapReduce programming model. The mapper functions execute phase 2 and generate the intermediate results. The rate of reduction in training time persists till 8 number of partitions as there is no significant decrement in the number of reduced points beyond 8 partitions.

3) Accuracy analysis

Fig. 13 and Table 4 show the accuracy analysis of the proposed approaches with respect to LIBSVM [22], CASVM [29], and BCSVM [33] on different datasets. For this analysis, we have considered four partitions of each dataset and each data partition is again divided into 80% for training and 20% for testing. The training dataset partition is first distributed and used as an input to phase 2 of the proposed model. The test data partition is used for testing in phase 4 and the accuracy is measured. Fig.13 and Table 4 evidently state that there is no significance loss in the accuracy while using the proposed SIMSVM distributed approach as compared to LIBSVM. The proposed approach shows either similar or improvement over the BCSVM approach. This consistency in the accuracy is achieved as the information about each partition of the data is effectively compressed in phase 2 of the proposed model. 1PVSVM shows a bit higher accuracy as compared to 2PVSVM as in case of 2PVSVM it losses a small amount of accuracy due to the random sampling from the reduced points in phase 4. This selection results in less accurate planes between two classes as all points of these classes are not considered at once for generating the final hyperplanes. In contrast, SIMSVM shows better performance than 1PVSVM as it uses a direct similarity measure (along with kernel trick) between a test and a reduced point. The proposed approaches are also compared with the other machine learning techniques and a comparative analysis is presented in Fig.14. All the three proposed approaches maintain similar accuracy with respect to other machine learning techniques. Each proposed approach is also tested for its implementation as the incremental model of classification to serve the purpose of continuous data stream mining. In this analysis, the data partitions are incrementally added to the existing pool of information which is used for testing of the test data. Fig. 15 depicts this basic model used for testing, where, information represents the support vectors extracted after completing phase 2 of the proposed model. Each time a new data partition generates the new data points (D_i) which are combined with the existing reduced points ($info_{i-1}$). Thus, a new reduced information set is produced and considered as the information($info_i$) for the next iteration. Phase 3 and 4 then use this $info_i$ for further processing and testing, which results in a better prediction of the class label for the test data. Fig. 16a and 16b plot the classification accuracy of the proposed approaches using MNIST and Two moon dataset by dividing each dataset into 10 disjoint subsets and then incrementally adding these subsets for incremental training. As new data points are added, SIMSVM takes the advantage (as shown in both the figures) of kernelized similarity calculation and exempts itself from calculating any hyperplane, which in turn reduces its chances of misclassification due to missing data points. It is further observed in Fig. 16a that 2PVSVM in case of MNIST dataset shows accuracy equivalent to 1PVSVM with the addition of $4th$ data chunk and the $8th$ data chunks onwards as at this point the sampling in phase 4 for 2PVSVM resulting in better hyperplanes for classification.

The rate of enhancement in the accuracy depends on the properties of the considered data. The results reveal that proposed approaches can also be used in real time scenarios where data is incrementally added to the existing pool, for e.g. prediction from a continuous stream of data.

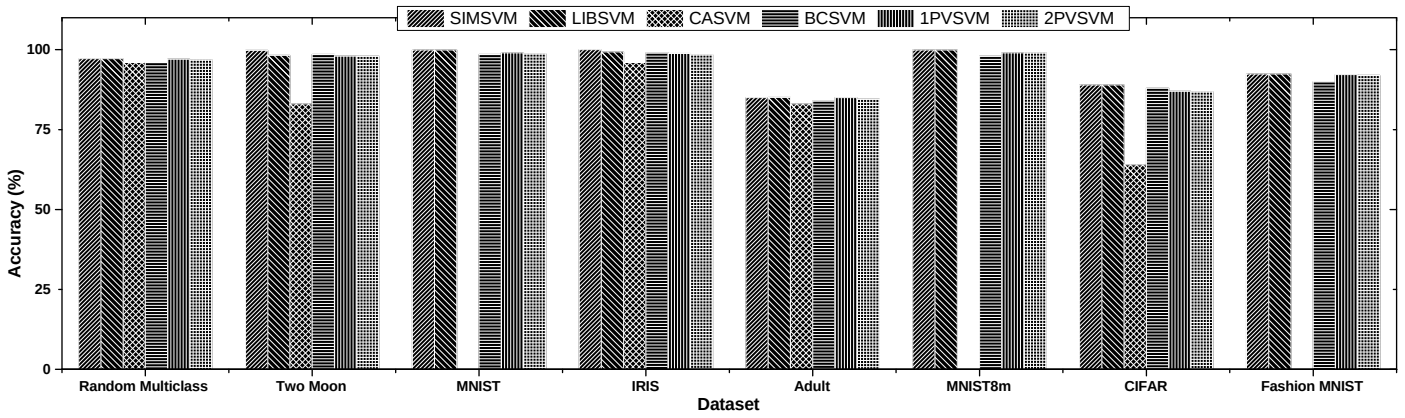


FIGURE 13: A comparison of accuracy analysis of proposed approaches with respect to LIBSVM and CASVM on different datasets.

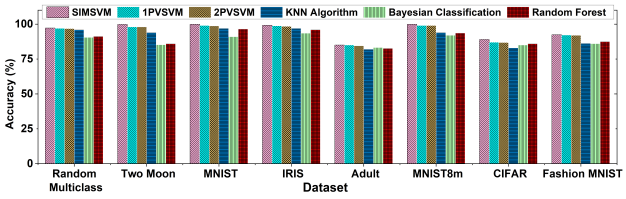


FIGURE 14: A comparative analysis of accuracy with respect to well known machine learning approaches

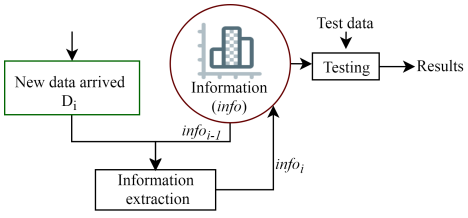
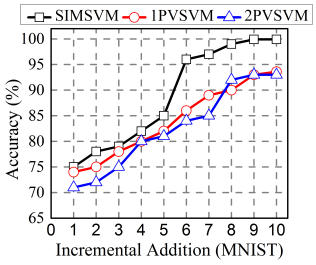
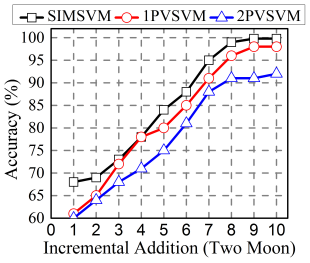


FIGURE 15: Model used for incremental testing.



(a) Accuracy analysis with MNIST datasets.



(b) Accuracy analysis with Two Moon datasets.

FIGURE 16: Incremental analysis of accuracy for MNIST and Two Moon dataset while incrementally adding data chunks for classification.

4) Speedup and Scalability Analysis

The speedup analysis is done to analyze the parallel processing capabilities of the proposed approaches by dividing the integral problem and adding resources to solve each division of the problem. In order to perform this analysis, a continuous partition up to 10 of the complete data is carried out while adding processing units in the same proportion and analyze the training time [44] of the three proposed approaches to observe the speedup. For example, if the number of partitions is four, then four parallel processing modules (mappers) are used, one for each partition (generates SV in phase 2). The objective of this analysis is to observe the speedup in the amount of time required to process the data as the number of processing units increases. We have calculated a speedup ratio measure which is the ratio between training time ($TT(1)$) required for training over the complete dataset using single processing module and the training time ($TT(p)$) required for training over p partitions of the dataset using p processing modules.

$$SpRatio = \frac{TT(1)}{TT(p)} \tag{18}$$

Let, $TT(p)$ is the training time for p number of partitions. The expected value of the $SpRatio$ is p for an ideal speedup implementation. For example, if proposed approach takes $TT(1)$ amount of time to train the dataset while considering a single partition, then, for 2 partitions, we can expect that the amount of time required to train is half of the time required for 1 partition, i.e. $TT(2) = TT(1)/2$. Hence, $SpRatio(2) = 2$ Figure 17 illustrates the analysis of the expected speedup with the observed speedup for 1PVSVM,

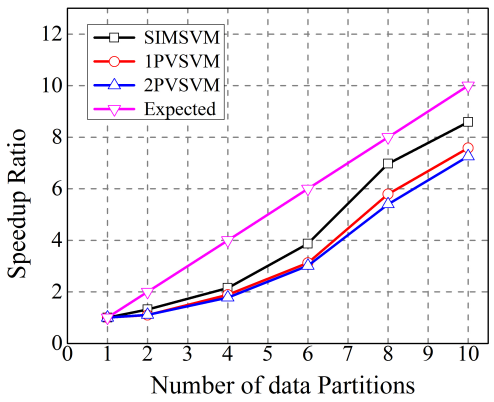


FIGURE 17: Speedup analysis for MNIST dataset with increasing number of partitions.

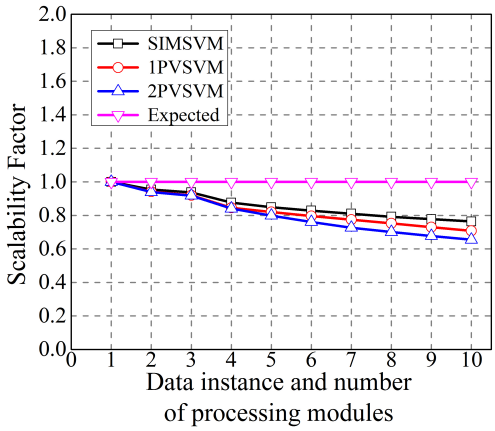


FIGURE 18: Scalability factor analysis for MNIST dataset by iteratively adding data instances as well as processing modules.

2PVSVM, and SIMSVM considering MNIST dataset. Till the number of partitions equals to four, the speedup of the approaches remains similar, however, after this point, SIMSVM takes the leverage of distributed processing along with the reduction in the number of data points available to each subprocess for kernelized distance calculation and achieves a better speedup than other two approaches.

Further, we have investigated the scalability of our approaches by increasing the problem size as well as the resources. The steps for conducting the same as follows:

- MNIST data is partitioned into 10 disjoint subsets.
- Training is performed on the first subset using a single processing module $TS(1)$.
- Iteratively new data is added along with processing modules and training time is observed as $TS(p)$.

The scalability is calculated using a scalability factor(SF) which is the ratio between $TS(1)$ and $TS(p)$ as shown in equation 19.

$$SF = \frac{TS(1)}{TS(p)} \tag{19}$$

Figure 18 shows the scalability analysis for MNIST dataset, where, the expected value of SF is 1. The proposed approaches indicate at least 60% scalability can be achieved even when the size of the problem is scaled to 10 times. As the scale of the problem increases, the minimum scalability that SIMSVM shows is 73% due to its high independence among the processing modules as compared to 1PVSVM and 2PVSVM.

VI. DISCUSSION

Our intended approaches, 1PVSVM, 2PVSVM, and SIMSVM, are the distributed classification techniques that utilize the power of distributed computing to predict the class labels for massive datasets. These approaches are quite appropriate and well adaptive to the recently used data processing models like MapReduce, Spark, etc. In this section, we have highlighted some significant findings from our approach.

- It has been empirically observed that adding more resources, the training time can be minimized during the initial additions. After a certain point, depending on the dataset distribution, the rate of decrement in training time gets slow down as at this point each data partition contains the reduced set of data points that are marked as essential points for classification purpose and passed to the next phase.
- Although the proposed approaches attain a good amount of accuracy over different datasets similar to other centralized and distributed approaches in the literature, one important research question that might arise is the cost associated with the addition of processing units which requires a trade-off between the training time and overall resource cost.
- The proposed approaches efficiently retain the accuracy when the distributed data contains imbalanced class distribution. This property makes the approaches more adaptive to real time scenario and removes the overhead of preprocessing of data for its balancing.
- We have attempted to present the scalability analysis of the proposed approaches considering addition of a new processing unit for each new data partition. However, deciding the number of partitions or processing units is crucial depending on the availability of the resources.
- The three proposed approaches are tested for their incremental versions. Here, again an interesting research question might arise that when to update the classifiers or when to add incremental information to the existing information pool.

VII. CONCLUSION

This work addressed the challenges of classification using support vector machine in a distributed environment for big

data analytics. Through the proposed approaches, we aimed to solve the issue of efficiently handling trivial partitioning of large data without loss of accuracy. Independent operations have been applied over each subset of data ensuring no communication among the processing units. The proposed techniques illustrate that retaining only support vectors from trivially partitioned dataset subset is sufficient to develop the global classification criteria for a new testing point. The empirical results show the support to this argument and reduce the overall training time while utilizing the power of distributed processing. All the three intended approaches achieve high speedup similar to the expected speedup and follow a positive slope as the new processing units are added. Further, the approaches maintain the scalability of the overall process closed to the expected as new resources and data are added. The complete proposed model overcomes the problem of explosion of reduced points in the distributed storage by applying the deduplication process. The proposed approaches are also tested to check their suitability for an incremental analysis with no loss in the accuracy when new data is added.

As a future research direction, we would extend our work to handle real time data like a continuous stream of IoT data by efficiently distributing the processing and storage load of the overall learning process. Further, a parallel feature reduction technique may be developed and incorporated into the distributed classification process.

REFERENCES

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

[2] J. Liu and E. Zio, "A scalable fuzzy support vector machine for fault detection in transportation systems," *Expert Systems with Applications*, vol. 102, pp. 36–43, 2018.

[3] W. Fu, K. Shao, J. Tan, and K. Wang, "Fault diagnosis for rolling bearings based on composite multiscale fine-sorted dispersion entropy and svm with hybrid mutation sca-hho algorithm optimization," *IEEE Access*, vol. 8, pp. 13 086–13 104, 2020.

[4] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, and C. Stachniss, "Uav-based crop and weed classification for smart farming," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3024–3031.

[5] C. Luo, L. Yu, J. Yan, Z. Li, P. Ren, X. Bai, E. Yang, and Y. Liu, "Autonomous detection of damage to multiple steel surfaces from 360 panoramas using deep neural networks," *Computer-Aided Civil and Infrastructure Engineering*, vol. 36, no. 12, pp. 1585–1599, 2021.

[6] Y. Ma and G. Guo, *Support vector machines applications*, Y. Ma and G. Guo, Eds. Springer, 2014.

[7] M. I. Pramanik, R. Y. Lau, H. Demirkan, and M. A. K. Azad, "Smart health: Big data enabled health paradigm within smart cities," *Expert Systems with Applications*, vol. 87, pp. 370–383, 2017.

[8] L. Bottou and C.-J. Lin, "Support vector machine solvers," *Large scale kernel machines*, vol. 3, no. 1, pp. 301–320, 2007.

[9] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, no. 3-4, pp. 145–164, 2016.

[11] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[12] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. M. Nguifo, "An experimental survey on big data frameworks," *Future Generation Computer Systems*, vol. 86, pp. 546–564, 2018.

[13] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[14] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile networks and applications*, vol. 19, no. 2, pp. 171–209, 2014.

[15] M. J. Kaur, "A comprehensive survey on architecture for big data processing in mobile edge computing environments," in *Edge Computing*. Springer, 2019, pp. 33–49.

[16] R. C. B. Madeo, S. M. Peres, and C. A. de Moraes Lima, "Gesture phase segmentation using support vector machines," *Expert Systems with Applications*, vol. 56, pp. 100–115, 2016.

[17] X. Ke, H. Jin, X. Xie, and J. Cao, "A distributed svm method based on the iterative mapreduce," in *Semantic Computing (ICSC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 116–119.

[18] Y. Hou, Y. Wang, X. Ma, and L. Cheng, "Hdsvm: A high efficiency distributed svm framework over data stream," in *Ubiquitous Computing and Communications (ISPA/IUCC), 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on*. IEEE, 2017, pp. 352–359.

[19] L. Futou and L. Liang, "Decision making based on grey model and support vector machine," *Cluster Computing*, pp. 1–7, 2018.

[20] R. Kashef, "A boosted svm classifier trained by incremental learning and decremental unlearning approach," *Expert Systems with Applications*, vol. 167, p. 114154, 2021.

[21] A. Torres-Barrán, C. M. Alaíz, and J. R. Dorronsoro, "Faster svm training via conjugate smo," *Pattern Recognition*, vol. 111, p. 107644, 2021.

[22] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[23] T. D. Nguyen, V. Nguyen, T. Le, and D. Phung, "Distributed data augmented support vector machine on spark," in *Pattern Recognition (ICPR), 2016 23rd International Conference on*. IEEE, 2016, pp. 498–503.

[24] H. Yu, J. Yang, and J. Han, "Classifying large data sets using svms with hierarchical clusters," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 306–315.

[25] T. Sun, H. Wang, Y. Shen, and J. Wu, "Accelerating support vector machine learning with gpu-based mapreduce," in *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 876–881.

[26] H.-Y. Huang and C.-J. Lin, "Linear and kernel classification: When to use which?" in *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 2016, pp. 216–224.

[27] L.-D. Bui, M.-T. Tran-Nguyen, Y.-G. Kim, and T.-N. Do, "Parallel algorithm of local support vector regression for large datasets," in *International Conference on Future Data and Security Engineering*. Springer, 2017, pp. 139–153.

[28] Q. Gu and J. Han, "Clustered support vector machines," in *Artificial Intelligence and Statistics*, 2013, pp. 307–315.

[29] Y. You, J. Demmel, K. Czechowski, L. Song, and R. Vuduc, "Ca-svm: Communication-avoiding support vector machines on distributed systems," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 847–859.

[30] F. Chang and C.-C. Liu, "Decision tree as an accelerator for support vector machines," in *Advances in Character Recognition*. InTech, 2012.

[31] A. L. Chau, X. Li, and W. Yu, "Support vector machine classification for large datasets using decision tree and fisher linear discriminant," *Future Generation Computer Systems*, vol. 36, pp. 57–65, 2014.

[32] T.-N. Do and F. Poulet, "Parallel learning of local svm algorithms for classifying large datasets," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXI*. Springer, 2017, pp. 67–93.

[33] H. A. Fayed and A. F. Atiya, "Decision boundary clustering for efficient local svm," *Applied Soft Computing*, vol. 110, p. 107628, 2021.

[34] D. Borthakur, "The hadoop distributed file system: Architecture and design, 2007," *Apache Software Foundation*, vol. 133, 2011.

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

[36] A. Vedaldi and A. Zisserman, "Efficient additive kernels via explicit feature maps," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 3, pp. 480–492, 2012.

[37] G. Camps-Valls, T. V. B. Marsheva, and D. Zhou, "Semi-supervised graph-based hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, no. 10, pp. 3044–3054, 2007.
[38] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2. Montreal, Canada, 1995, pp. 1137–1145.
[39] D. Dheeru and E. Karra Taniskidou, "Adult dataset UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[40] G. Loosli, S. Canu, and L. Bottou, "Training invariant support vector machines using selective sampling," *Large scale kernel machines*, vol. 2, 2007.
[41] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Technical report, University of Toronto*, vol. 1, no. 4, p. 7, 2009.
[42] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[43] P. C. Sen, M. Hajra, and M. Ghosh, "Supervised classification algorithms in machine learning: A survey and review," in *Emerging technology in modelling and graphics*. Springer, 2020, pp. 99–111.
[44] H. Feng, D. Eyers, S. Mills, Y. Wu, and Z. Huang, "Principal component analysis based filtering for scalable, high precision k-nn search," *IEEE Transactions on Computers*, no. 2, pp. 252–267, 2018.

Amrit Pal has received his B.Tech(2011) from Kurukshetra University Kurukshetra, India and M.Tech (2014) from National Institute of Technical Teachers' Training and Research Bhopal, India. He had received his Ph.D (2020) from Indian Institute of Information Technology Allahabad. He worked as an Assistant professor at Centre for advanced studies AKTU Lucknow, India. Currently, he is working as an assistant professor at Vellore Institute of Technology Chennai Campus, Chennai India. His research areas includes big data analytics, cloud computing and internet of things.

Abishi Chowdhury has done her B. E. from University Institute of Technology, West Bengal, India and M. Tech. from National Institute of Technical Teachers' Training and Research, Bhopal, Madhya Pradesh, India. She has done her Ph. D. from Visvesvaraya National Institute of Technology, Nagpur, India. Currently, she is working as an assistant professor at Vellore Institute of Technology Chennai Campus, Chennai India. Her research interest includes cloud computing, cloud resource scheduling, machine learning, and internet of things.

Satakshi Satakshi received her M.Sc. (1998) and M.Phil(1999) in Mathematics from University of Roorkee. In 2004, she received a Ph.D. in Applied Mathematics from IIT, Roorkee and later joined Birla Institute of Technology and Sciences(BITS) Pilani, where she worked as a faculty of Mathematics for two years. In 2017, she joined the Sam Higginbottom University of Agriculture, Technology and Sciences, where she is working as a faculty in the Department of Mathematics and Statistics. Her research interests include Order Reduction of Linear Systems, Optimization, Evolutionary Algorithms, Machine Learning, Time Series etc.

Husnu S. Narman received the BS degree in Mathematics from Abant Izzet Baysal University, Turkey, in 2006, the MS degree in Computer Science from the University of Texas at San Antonio, San Antonio, Texas, in 2011, and the PhD degree in Computer Science from the University of Oklahoma, Norman, Oklahoma, in 2016. Currently, he is a faculty member at Department of Computer Sciences and Electrical Engineering, Marshall University, Huntington, West Virginia, USA. His research interests include queuing theory, network management, network topology, Internet of Things, LTE, and cloud computing.

Arkabandhu Chowdhury is currently working as an applied scientist at Amazon Alexa in Boston, USA. He received both his PhD and Master's degrees in Computer Science from Rice University, Houston, USA and Bachelor's degree in Electronics and Telecommunication Engineering from Jadavpur University, Kolkata, India. His field of research is machine learning. He is interested in computer vision, natural language processing, optimization, and statistical data mining on large-scale data with an emphasis on Bayesian inference and deep neural networks.

Manish Kumar received his PhD degree on Data Management in Wireless Sensor Networks from Indian Institute of Information Technology Allahabad, Prayagraj India in 2011. He received his M.Tech. degree in Computer Science from Birla Institute of Technology, Mesra (Ranchi) India. He is a professional member of IEEE and ACM. Currently he is working as an Associate Professor in Department of Information Technology at Indian Institute of Information Technology, Allahabad India. His research interest includes data mining and warehousing, data management in wireless sensor networks and Big Data Analytics. He has contributed in a number of books in the same areas and has many national and international publications in renowned journals and conferences.