Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2021) 29: 2934 – 2949
© TÜBİTAK
doi:10.3906/elk-2008-155

Design development and performance analysis of distributed least square twin support vector machine for binary classification

Bakshi Rohit Prasad[*], **Sonali Agarwal**
Department of Information Technology, Indian Institute of Information Technology, Allahabad, India

Received: 30.08.2020 • **Accepted/Published Online:** 05.05.2021 • **Final Version:** 30.11.2021

Abstract: Machine learning (ML) on Big Data has gone beyond the capacity of traditional machines and technologies. ML for large scale datasets is the current focus of researchers. Most of the ML algorithms primarily suffer from memory constraints, complex computation, and scalability issues.The least square twin support vector machine (LSTSVM) technique is an extended version of support vector machine (SVM). It is much faster as compared to SVM and is widely used for classification tasks. However, when applied to large scale datasets having millions or billions of samples and/or large number of classes, it causes computational and storage bottlenecks. This paper proposes a novel scalable design for LSTSVM named distributed LSTSVM (DLSTSVM). This design exploits distributed computation on cluster of machines to provide a scalable solution to LSTSVM. Very large datasets are partitioned and distributed in the form of resilient distributed datasets on top of Spark cluster computing engine. LSTSVM is trained to generate two nonparallel hyper-planes. These hyper-planes are achieved by solving two systems of linear equations each of which involves data instances from either class. While designing DLSTSVM we employed distributed matrix operations using the MapReduce paradigm of computing to distribute the tasks over multiple machines in the cluster. Thus, memory constraints with extremely large datasets are averted. Experimental results show the reduction in time complexity as compared to existing scalable solutions to SVM and its variants. Moreover, detailed experiments depict the scalability of the proposed design with respect to large datasets.

Key words: Distributed machine learning, Big Data, cluster computing, least square twin support vector machine (LSTSVM), MapReduce, parallel processing

1. Introduction

Existing machine learning (ML) techniques may be classified into two broad categories; supervised ML techniques and unsupervised ML techniques [1]. In supervised ML, classification rules are built on a training set which contains data instance with class labels. Then after, new data instances are classified using the learnt classification rules. Support vector machine (SVM) is an extensively used supervised ML algorithm developed by Vapnik et al. [1]. SVM generates two parallel hyper-planes during training. These hyper-planes separate data from two classes. The optimal hyper-plane is generated by solving a complex quadratic programming problem (QPP). The overall training takes order of $O(n^3)$ time if there are n training data samples. A QPP size directly depends on the dataset size. Moreover, solving a QPP is computationally expensive job. Another faster variant of SVM has been proposed which solves two smaller QPPs unlike SVM. This technique is known as twin support vector machines (TWSVM) [2]. This causes significant reduction in time complexity involved

[*]Correspondence: rs151@iiita.ac.in

in SVM training and speeds up the training four times faster than that of SVM. Though, the QPPs are smaller, solving the same is still a complex task. A new faster version of TWSVM has been proposed developed by Kumar et al. [3] and is well known as LSTSVM which is widely used for the classification task. Unlike TWSVM, the LSTSVM is trained to generate two nonparallel hyper-planes. These hyper-planes are achieved by solving two systems of linear equations each of which involves data instances from either class. Thus, it is significantly faster than TWSVM. Figure 1 represents the categorization of two classes by using LSTSVM. As shown in the Figure 1 there are two classes; class 1 and class 2, which are divided by using two nonparallel planes in such a way that each plane is nearer to the data points of one class while farther from the other class.

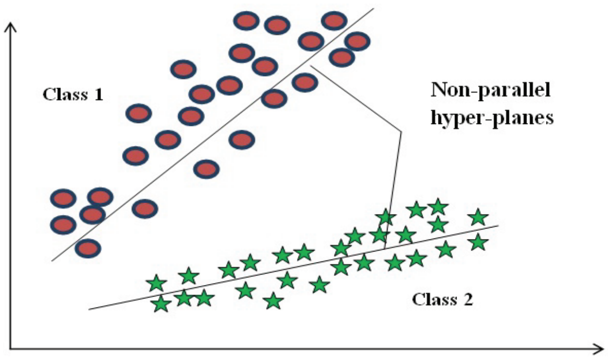

Figure 1. Hyper-planes generated by LSTSVM.

Although LSTSVM is much faster as compared to TWSVM, yet very large datasets cause computation and storage bottlenecks in case of LSTSVM. Due to this fact, capability of LSTSVM technique is limited to smaller dataset only. For very large datasets, training becomes too expensive and system may go out of memory as well.

1.1. Research contribution

To solve the underlying computation and storage challenges of LSTSVM on very large datasets, the proposed technique named DLSTSVM employs distributed computing over multiple machines in a parallel fashion on top of recent distributed parallel computing framework. Our research work makes following contribution:

- The way all computations are distributed over multiple machines makes DLSTSVM a scalable technique which is the prime objective of current research work.
- DLSTSVM not only handles scales well even with large number of input patterns, due to data parallel computing, it achieves significant speed-ups too.
- This work gives the designing of DLSTSVM based on MapReduce paradigm of computing along with detailed performance and scalability analysis.

The present research work is illustrated in six major sections. Section 1 gives a brief introduction to supervised ML techniques along with their underlying challenges in case of very large datasets. In section 2, state-of-the-art is presented around distributed frameworks for Big Data machine learning. Also, multiple variants of SVM, LSTSVM, and distributed SVM are discussed along with their inherent challenges. Section 3 gives a brief background of mathematical preliminaries of LSTSVM as well as basic fundamentals of Spark framework.

Section 4 describes the design of DLSTSVM algorithm and discusses its time and storage complexity. Detailed experimental results in section 5 compare the performance of the proposed algorithm with earlier techniques. Moreover, to assess the scalability of DLSTSVM, manifold experiments are performed on large-scale synthetically generated datasets. Finally, section 6 concludes the presented work with probable future prospects.

2. Literature survey

Related research works are explored in three major dimensions; first is related to distributed frameworks and libraries for handling Big Data, second is related to multiple single machine variants of SVM and LSTSVM, and third is related to scalable solutions to SVM and its variants along with their limitations.

2.1. Distributed frameworks and libraries for Big Data

To utilize the advantages of distributed computing to enable machine learning systems to deal with large-scale datasets multiple distributed ML frameworks and libraries are developed. Apache Mahout includes distributed scalable ML algorithms implemented for Hadoop and provide free scalable ML libraries. GraphLab implements Machine Learning-Data Mining in the cloud-based environment for graph related operations in parallel. Microsoft's DryadLINQ [5]using LINQ on top of Dryad, is well suited for data parallel applications. Spark [6], a distributed computing engine, well suited for iterative nature of ML and facilitates vivid ML tasks. Multiple recent survey [7], discusses and compares pros and cons of several computing engines such as Flink, Spark, H_2O, and Storm. Also, they compared ML libraries inside these engines such as Mahout and MLlib.[29]

2.2. SVM and LSTSVM variants

SVM and its various variants for binary class classifications are available. Several researchers extended the basic SVM and LSTSVM to multi-class scenarios too [9]. Multiple variants of SVMs have been proposed out of which some are exact solvers [11, 13], some fall under the categories of approximate solvers [14–17], divide and conquer SVM solvers [10] online SVM solvers [12]. Accurate solvers are time consuming whereas approximate solvers compromise with accuracy. Although, divide and conquer approach try to give faster solution but with large scale data instances they still exhibit high training time. In previous couple of years some more nonparallel variants of binary class LSTSVM have been proposed using Fuzzy Logic, KNN, and entropy based techniques [18–21] but with million scale data instances their computation time is extremely high.

2.3. Distributed and parallel machine learning algorithms

Parallel machine learning is an aspect to scale up existing sequential ML algorithms [28]. Advancements in the field of multi-core and cloud computing architectures have caused wide accessibility to distributed and parallel computing systems [8]. Faster variants of SVM (including LSTSVM) too become incapable of handling very large-scale datasets. Several solutions have been tried to scale SVM using various approaches like dividing the dataset, training SVM on each subset in parallel, and then combining multiple classifiers, thereby iterating this process [32–34]. Some used multiple repartitioning of data which involves sharing of training data among nodes, thereby causing significant training overhead. Some required frequent communication of subsolutions resulting in slow training time. Thus, drawback of these approaches is that reallocation process slows down the training process. Approach given in a research work [13] used cascading of support vectors in multiple stages of training till final set of support vectors is obtained which involves significant data transfer cost.

Development of distributed computation on commodity nodes and MapReduce kind of frameworks, the problems of aforementioned techniques are tried to resolve [35]. Some researchers employed high-speed costly graphics processing units (GPU) and applied MapReduce based technique. The major drawback of this approach is in tedious setting up specialized configurations which makes it difficult to use and develop applications using this approach [28]. Apart from state-of-the-art parallel SVM methods [22, 23] some recent parallel and distributed SVM techniques are also proposed in previous couple of years [24–26]. One of these approaches provides a distributed and parallel mechanism for alternating direction method of multipliers for non-convex penalized SVMs [24] at cost of accuracy. Another technique performs subspace partitioning on the datasets by using decision tree on projection of data showing maximum variance [25] and shows approximately 150 times speedup over sequential SVM. One group of researchers proposed a parallelization strategy in training SVM which finds a low rank approximation of matrices and uses random projection mechanism [26]. This solves the approximated optimization problem and appears as scalable approach still, its run-time is very high for million scale datasets.

3. Fundamental concepts of LSTSVM and Spark

This section specifies the basic formulation of LSTSVM and its incompetence to handle the large scale datasets with respect to storage and computation. Further, it briefs the basic description of Spark and its features.

3.1. Mathematical formulation of LSTSVM

As discussed earlier, LSTSVM finds two hyper-planes. These hyper-planes are not necessary to be parallel. The nonparallel hyperplanes are obtained by following two systems of linear equations given by 1 and 2.

$$min(w_1, b_1, \xi)\frac{1}{2}||X_1w_1 + e_1b_1||^2 + \frac{c_1}{2}\xi^T\xi\Big[(s.t.) - (X_2w_1 + e_2b_1) + \xi = e_2\Big]. \tag{1}$$

$$min(w_2, b_2, \eta)\frac{1}{2}||X_2w_2 + e_2b_2||^2 + \frac{c_2}{2}\eta^T\eta\Big[(s.t.)(X_1w_2 + e_1b_2) + \eta = e_1\Big]. \tag{2}$$

Here, X_1 and X_2 are data matrices corresponding to class -1 and class +1 respectively and have total l_1 and l_2 instances. Here, w_1, b_1 and w_2, b_2 are respective hyper-plane parameters know as weights and bias. Here, $e \in R^{1_1}$ and $e \in R^{1_2}$ are the vectors with entries of 1 only, c_1 and c_2 are nonnegative penalty parameters whereas $\xi \in R^{1_2}$ and $\eta \in R^{1_1}$ are known as slack variables for class -1 and class $+1$. Thus, optimization equations 1 and 2, for LSTSVM contain only equality constraints. Finally, upon solving these equations, hyper-plane parameters are attained as given in equation 3 and 4. Here, $G = [X_1e_1]$ and $H = [X_2e_2]$.

$$\begin{bmatrix} w_1 \\ b_1 \end{bmatrix} = -\Big(H^TH + \frac{1}{c_1}G^TG\Big)^{-1}H^Te_2. \tag{3}$$

$$\begin{bmatrix} w_2 \\ b_2 \end{bmatrix} = -\Big(G^TG + \frac{1}{c_2}H^TH\Big)^{-1}G^Te_1. \tag{4}$$

With the help above hyper-plane parameters, the obtained hyper-planes are given as per equation 5 and 6.

$$X^Tw_1 + b_1 = 0. \tag{5}$$

$$X^Tw_2 + b_2 = 0. \tag{6}$$

When a new data instance arrives, it is assigned a class based on the decision function give in equation 7.

$$f(x) = arg\min{}_{i=1,2}\frac{|w_i.x + b_i|}{||w_i||}. \tag{7}$$

With very large datasets, computing the equations 3 and 4 causes computation and storage bottlenecks, thereby training becomes too expensive and system may go out of memory as well. Thus, capability of LSTSVM technique is limited to smaller datasets only. The next subsection briefs the characteristics of Spark distributed processing engine used in our research work for distributed parallel computation over cluster of machines.

3.2. Apache Spark - a cluster computing framework

Apache Spark provides cluster computing which encompasses automatic locality-aware scheduling, load balancing, and fault tolerance. Spark facilitates data-parallel computations on cluster of commodity machines using MapReduce computing paradigm [4]. Spark is also well suited for iterative computing and interactive analytics. As shown in Figure 2, Spark engine core takes care of prime functions. These functions cover Spark computing environment set up, generation, transformation, and distribution of resilient distributed datasets RDDs. Spark driver internally distributes RDDs onto multiple machines in the cluster. It can be explicitly cached in main memory for reuse if required in an application [6].

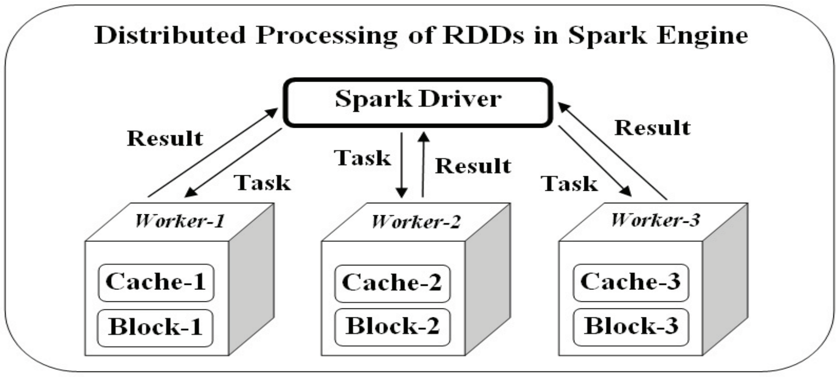

Figure 2. Spark distributed processing

4. Proposed methodology

Design of DLSTSVM includes three phases; first decomposes entire computation into different modules and identifies the parallel sections, second specifies the Map and Reduce steps for each module, and third discusses time and space complexities of DLSTSVM to show how the challenges of LSTSVM is resolved by DLSTSVM.

4.1. Problem decomposition

Consider equations 3 and 4 which yields two hyper-planes each corresponding to data instances from a single class. Following subsections identifies parallel computable parts involved in these series of matrix calculations.

(i) Identification of parallel computations: Equation 3 involves series of matrix operations. Entire computation is divided into two parts specified by equation 8 and 9. Each part is computed in parallel on the data residing on each machine. The serial and parallel portion of computation is clearly depicted in the flow diagram of Figure 3. Thus, the hyper-plane parameter u can be given as $u = R_{c\times c} \times S_{c\times 1}$ such that:

$$R = -((H^T H) + (1/c)(G^T G))^{-1}. \quad (8)$$

$$S = H^T e_2. \quad (9)$$

(ii) Data representation and partitioning: Matrix R requires computation of $G^T G$ and $H^T H$. For very large matrix G, computation of $G^T G$ needs to be distributed in such a way that may avert memory bottleneck as well as speedsup the processing. Therefore, matrix G is distributed as RDD over cluster where each element of RDD represents a data instance from matrix G in mutually exclusive fashion. Figure 4 depicts this partitioning of matrix $G_{r\times c}$ having r rows (representing data instances) and c columns (representing dimensions of each data instance). Here, g_i specifies i^{th} data instance of matrix G. Then, $G^T = [g_1^T, g_2^T, g_3^T ... g_r^T]_{c\times r}$, where each g_i^T is a column vector achieved by taking the transpose of corresponding row vector g_i.

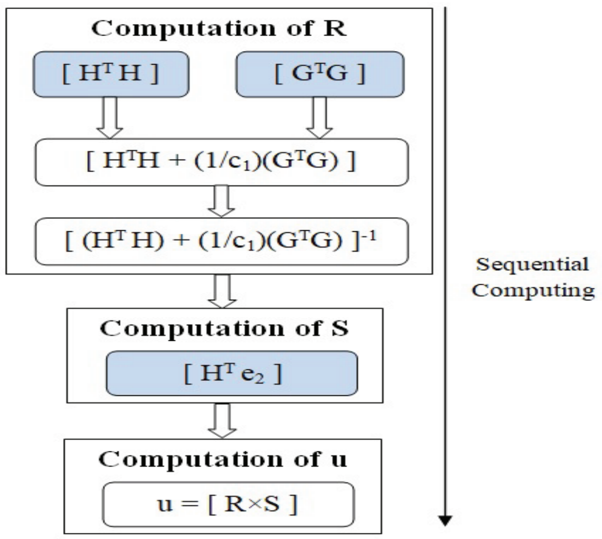

Figure 3. Computation flow of parallel (shaded block) and serial sections

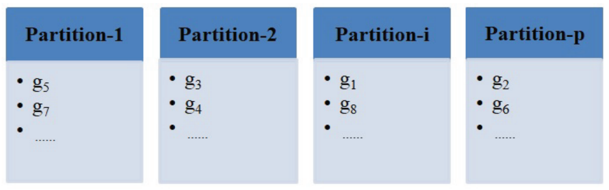

Figure 4. Data partitioning over cluster of machines

4.2. Modeling parallel computations as MAP-REDUCE steps

(i) For computing matrix R: MAP-REDUCE steps for $G^T G$ are modeled in following subsections:

MAP step: Figure 5 shows the MAP step which multiplies j^{th} column of G^T to j^{th} row of G, i.e. $g_i^T \times g_i$. Here, g_i is a row vector of dimension $1 \times c$ and g_i^T is a column vector of dimension $c \times 1$. In each partition-i, this MAP step is executed in parallel for each data instance g_j. Thus, $t_j = g_j^T \times g_j$ is a matrix of dimension $c \times c$, generated for very instance of matrix G.

REDUCE step: If each machine in cluster has m data instances then it generates m intermediate matrices. Since, matrix G has r such data instances, therefore total r intermediate matrices throughout the cluster. REDUCE step is designed to sum up all the temporary matrices t_j to produce the matrix $G^T G$ as $T = \Sigma_{j=1}^{r} t_j$

i.e. all t_j are reduced on sum operation for corresponding elements at $index(i,j)$. A simple matrix addition is applied on two intermediate matrices at a time in each partition. Another intermediate matrix is added to the sum matrix produced in the previous step. Cascading the sum, matrix T is computed as shown in Figure 6. Similarly, H^TH is also computed in parallel fashion, added to $(1/c_1)G^TG$ and inversed to yield matrix R.

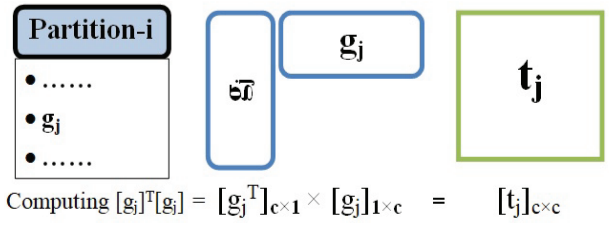
Partition-i
• ……
• g_j
• ……
g_j
g_j
t_j
Computing $[g_j]^T[g_j] = [g_j^T]_{c\times 1} \times [g_j]_{1\times c} = [t_j]_{c\times c}$

Figure 5. MAP step applied to each data instance

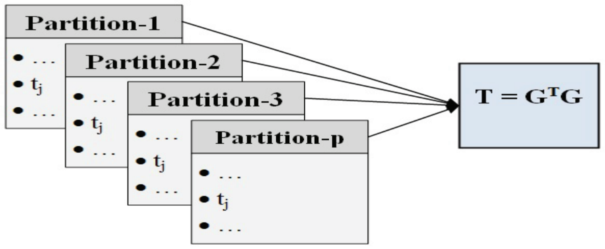

Figure 6. REDUCE step for summing all intermediate matrices

(ii) For computing $S = H^Te_2$***:*** For matrix H which is distributed over cluster, it has been evaluated that vector obtained by multiplying H^T to e_2 and the vector obtained by transposing the row-vector achieved after column-wise addition of elements of matrix H are same. Thus, MAP and REDUCE steps are designed as:

MAP step: For the approach proposed in the work, this step does not make any changes to underlying matrix elements. It simply forwards the RDD of matrix H for reduction by the following REDUCER.

REDUCE step: In reduce step, the column-vectors of the RDD corresponding to matrix H are reduced by sum operation defined on the $index(j)$. Here, j indicates a particular column and $1 <= j <= c$. This reduction is applied in parallel on each machine in the cluster in order to compute H^Te_2 in distributed parallel fashion. Internally, two row-vectors of H are picked up at a time in each partition and their corresponding column elements are added, resulting in new sum-vector. Next row-vector is added to the sum-vector produced in the previous step. Thus, all row-vectors on multiple machines are summed distributively to give vector S.

Algorithm 1 specifies the required sequence of steps involved in DLSTSVM.

Algorithm 1 DLSTSVM

```
1:  procedure SEGREGATE THE DATA OF CLASS-A AND CLASS-B AND TRANSFORM IT INTO RDD(as $A, B$)
2:      Define following two matrices: $G = [Ae_1]$ and $H = [Be_2]$ and two vectors as $e_1, e_2$
3:      Calculate matrix $R_1 = (H^TH + 1/c_1(G^TG))^{-1}$ and $R_2 = (G^TG + 1/c_2(H^TH))^{-1}$ as follows:
4:      a: Calculate $H^TH$ distributively as follows:
5:          MAP Phase: $t_1$ = H.map(rowTrans_row)
6:          REDUCE Phase: $t_1$.reduce(sumMatrix)
7:      b: Calculate $G^TG$ distributively:
8:          MAP Phase: $t_2$ = G.map(rowTrans_row)
9:          REDUCE Phase: $t_2$.reduce(sumMatrix)
10:     c: Compute $(H^TH + 1/c_1(G^TG))^{-1}$
11:     d: Compute $(G^TG + 1/c_1(H^TH))^{-1}$
12:     Calculate $S_1 = H^Te_2$ distributively as follows:
13:         MAP Phase: $d_1$ = H.map(feedToReduce)
14:         REDUCE Phase: $S_1 = d_1$.reduce(sumRowVector)
15:     Calculate $S_2 = G^Te_1$ distributively as follows:
16:         MAP Phase: $d_2$ = G.map(feedToReduce)
17:         REDUCE Phase: $S_2 = d_2$.reduce(sumRowVector)
18:     Find hyper-plane parameters as follows:
19:         $[W_1, b_1] = u = R_1 \times S_1$
20:         $[W_2, b_2] = u = R_2 \times S_2$
```

4.3. Analysis of time and space complexity

Consider matrix G be of order $r \times c$. Hence, the resultant matrix $T = G^T \times G$ will be of order $c \times c$. Since, LSTSVM applies usual matrix multiplication hence, takes $O(rc^2)$ time and requires $O(2rc + c^2) = O(rc + c^2)$ storage, $O(2rc)$ for G^T and G and $O(c^2)$ for matrix T. Very large values of r cause memory bottlenecks. $O(rc^2)$ computations are time-expensive too. Furthermore, we discuss the time and space complexity of DLSTSVM:

(i) Time and space complexity for R: Since, $G^T \times G$ is being computed in distributed parallel way, let there are n machines in the cluster. Computation of each intermediate matrix t_j requires c^2 products and c^2 storage. Thus, for a single machine $O(mc^2)$ time and $O(mc)$ space required which can be handled by a single machine. Here, $m = (r/n)$ specifies number of data instances lying on each machine. It can be stated that in DLSTSVM, $O(rc^2)$ computation is now distributed among n machines and each machine performs only $O(mc^2)$ computations. Also, storage is distributed, reducing to $O(c^2)$ only on a each machine.

Since, matrix G^TG and H^TH are of dimension $c \times c$, adding $(1/c_1(G^TG))$ and H^TH takes $O(c^2)$ addition operations and $O(c^2)$ storage. Resulting addition matrix is of order $c \times c$, hence, taking inverse will take $O(c^3)$ computation and $O(c^2)$ storage only. Thus, matrix R specified is computed successfully in $O(mc^2 + c^3 + c^2) = O(mc^2 + c^3)$ operations and $O(c^2 + c^2 + c^2) = O(c^2)$ storage on each machine.

(ii) Time and space complexity for S: As discussed in the REDUCE step of sub-section 4.2 to compute vector S, sum of corresponding elements from two subsequent vectors performs c additions. Since, each machine keeps m data instances, total $O((m-1) \times c) = O(mc)$ sum operations are executed on each machine, all machines running in parallel in cluster. Furthermore, each machine stores only two row-vectors to be added thereby requiring $O(c)$ space.

In summary, computation of R requires $O(mc^2 + c^3)$ time and $O(c^2)$ space whereas computation of S requires $O(mc)$ time and $O(c)$ space. Thus, matrix $u = R \times S$, is computed without any bottleneck.

5. Results and discussion

5.1. Experimental setup and configurations

System's configuration used for our experiment is listed in Table 1 specifying processor, memory, and operating system. Since, experimental setup requires distributed cluster hence, Table 1 also specifies the cluster size in terms of nodes, memory, and processor as well as the version of Spark and SBT (simple build tool) used here.

Table 1. System and cluster configuration.

System configuration	Parameter value
Processor	Intel(R) Core-i3 3.30 GHz
Total number of cores per computer system	4
RAM per computer system	6 GB
Operating system	Ubuntu 12.04 Linux

Cluster configuration	Parameter value
SBT version	0.11.3
Spark version	1.5.0
Number of nodes in the cluster	8
Total number of cores in cluster	24 (3 cores from each node /SPARK_WORKER_CORES = 3)
Total executor memory in cluster	32 GB (4 GB from each node /SPARK_WORKER_MEMORY= 4g)

5.2. Comparison with existing solutions

Effective libraries have been developed for SVM such as LIBSVM [31] and SVMLight. However, SVM and their optimized implementations in these libraries still do not scale efficiently even for 1 million data samples. Computation cost is too high and memory requirements are also high. Though some approximate solvers [32–34] are proposed, but they compromise with accuracy. For the sake of comparison, datasets that have been used are listed in Table 2. In addition, Table 2 shows a comparison of results of other existing techniques with DLSTSVM that clearly depicts the time efficiency of DLSTSVM over other state-of-the-art techniques.

Table 2. Performance comparison with earlier techniques.

Dataset	Ijcnn1	Census	Webspam	Kddcup99	Cifar
#Instances	141691	199523	350000	5209460	60000
#Attributes	22	409	254	125	1024
Parameter setting	C=32 , $\gamma = 2$	C=512, $\gamma = 2^{-9}$	C=8, $\gamma = 32$	C=256, $\gamma = 2^{-1}$	C=8, $\gamma = 2^{-22}$
Techniques	Time(s)/Acc(%)	Time(s)/Acc(%)	Time(s)/Acc(%)	Time(s)/Acc(%)	Time(s)/Acc(%)
DC-SVM-early [10]	12/98.35	261/94.9	670/99.13	470/92.61	1977/87.02
DC-SVM [10]	41/98.69	1051/94.2	10485/99.28	2739/92.59	16314/89.5
LIBSVM [11]	115/98.69	2920/94.2	29472/99.28	6580/92.51	42688/89.5
LaSVM [12]	251/98.57	3514/93.2	20342/99.25	6700/92.13	57204/88.19
CascadeSVM [13]	17.1/98.08	849/93.0	3515/98.1	1155/91.2	6148/86.8
LLSVM [14]	38/98.23	1212/92.8	2853/97.74	3015/91.5	9745/86.5
FastFood [15]	87/95.95	851/91.6	5563/96.47	2191/91.6	3357/80.3
SpSVM [16]	20/94.92	3121/90.4	6235/95.3	5124/90.5	21335/85.6
LTPU [17]	248/96.64	1695/92.0	4005/96.12	5100/92.1	17418/85.3
	c1=c2=10^{-1}	c1=c2=10^{-3}	c1=c2=10^{-3}	c1=c2=10^{-1}	c1=c2=10^{-2}
LSTSVM [3]	743.65/98.62	31716.24/95.2	23041.36/99.15	*/NA	*/NA
DLSTSVM	10/98.62	238/95.2	121/99.15	190/94.4	5245/91.3

Here, Table 2 specifies the best performances achieved out of multiple executions of various techniques over a grid of points corresponding to a range of values for balancing parameters $C = [2^{-10}, 2^{-9}, ..., 2^{10}]$ and kernel parameter $\gamma = [2^{-10}, 2^{-9}, ..., 2^{10}]$ for RBF kernel. Moreover, the stopping criterion is set to the default value 10^{-3} in case of LIBSVM and DC-SVM. Rank, $\#clusters$, and $\#features$ (corresponding to LLSVM, LTPU, and FastFood respectively) are taken as 3000. However, for DLSTSVM, value of c1=c2 has been set at the same value in the range $[10^{-5}, 10^{-4}, ..., 10^{5}]$, on which LSTSVM gives the best performance. It is evident from the results that at same parameter setting, the predictive performance of LSTSVM and DLSTSVM is same. Another set of experiments are performed to compare the run-time and predictive performance of DLSTSVM with state-of-art recent variants of LSTSVM. Results are calculated on bench-marking large scale datasets like CovType and Mnist [30] as well as David's NDC data generator [27]. Table 3 lists the properties of these datasets, parameter settings used in experiments and performance comparison of various techniques. It is clearly observed that DLSTSVM outperforms all the non-parallel state-of-art LSTSVM and its variant techniques in terms of run-time performance and at the same time DLSTSVM exhibits comparable predictive performance too with state-of-the-art techniques.

Moreover, further experiments are conducted to compare DLSTSVM with some existing state-of-the-art distributed parallel techniques too. Benchmarking datasets like adults, gisette, covType, and Mnist are used in our experiments. Table 4 specifies the properties of these datasets, parameter settings used in experiments and performance comparison of various techniques. Results clearly show better or comparable predictive performance of DLSTSVM with most of the other techniques. At the same time, run-time performance is much better than parallel SVM, ADMM parallel, xSVM, and parallel distributed penalized SVM techniques. Though, projection-SVM shows better run-time here for large scale feature vector size however, since, DLSTSVM is scalable in nature, there is scope of adding more number of executors to reduce this computation time further.

Table 3. Performance comparison with state-of-art LSTSVM variants.

Dataset	NDC-100k	NDC-500k	NDC-1m	CovType	Mnist
#Instances	100,000	500,000	1,000,000	464,810	2,000,000
#Attributes	32	32	32	54	784
Parameter setting	C=1 , $\gamma = 1$	C=1, $\gamma = 1$	C=1, $\gamma = 1$	C=32, $\gamma = 32$	C=8, $\gamma = 2^{-3}$
Techniques	Time (s)/Acc.(%)	Time (s)/Acc.(%)	Time (s)/Acc.(%)	Time (s)/Acc.(%)	Time (s)/Acc.(%)
FLSTSVM [18]	21.67/87.48	89.5/86.24	209.28/ 86.10	242.34/94.74	9242.83/95.83
EFLSTSVM [19]	15.53/87.21	91.27/86.62	190.63/87.34	980.19/95.26	14855.62/96.18
ULSTPMSVM [20]	228.56/88.72	1144.95/87.13	2289.84/88.48	2082.46/95.42	*/NA
KNN-based LSTSVM [21]	1919.5/87.69	119968.46/87.37	*/ NA	*/NA	*/NA
	c1=c2=10^{-1}	c1=c2=10^{-3}	c1=c2=10^{-3}	c1=c2=10^{-1}	c1=c2=10^{-2}
LSTSVM [3]]	421.76/88.14	2102.34/87.42	4263.78/87.58	15017.35/96.15	#/NA
DLSTSVM	6.31/88.14	30.22/87.42	63.47/87.58	91.26/96.15	4827.68/97.58

*Represents experiment stopped as computation time was very high, #Represents memory out of run.

Table 4. Performance comparison with state-of-art distributed and parallel SVM techniques.

Dataset	Adult	Gisette	CovType	Mnist
#Instances	32,561	6000	464,810	2,000,000
#Attributes	123	5000	54	784
Parameter setting	C=32 , $\gamma = 2^{-7}$	C=1, $\gamma = 2^{-4}$	C=32, $\gamma = 32$	C=1, $\gamma = 2^{-5}$
Techniques	Time (s)/Acc.(%)	Time (s)/Acc.(%)	Time (s)/Acc.(%)	Time (s)/Acc.(%)
Parallel SVM [22]	488.32/84.42	8241.47/97.56	5281.74/95.89	*/NA
ADMM Parallel [23]	59.87/85.67	1251.52/97.62	934.54/96.03	1855.12/97.21
Parallel and distributed penalized SVM [24]	33.51/84.31	745.20/96.43	567.44/96.24	1101.59/97.04
Projection-SVM [25]	58.86/84.83	1007.94/97.20	877.15/97.12	2986.21/97.59
xSVM [26]	39.11/85.14	712.23/97.41	543.56/96.03	4079.74/97.70
	c1=c2=10^{-1}	c1=c2=10^{-3}	c1=c2=10^{-3}	c1=c2=10^{-2}
DLSTSVM	2.87/85.74	4107.36/97.91	91.26/96.85	3827.68/97.58

5.3. Scalability analysis of DLSTSVM

This section conducts experiments to test the scalability of DLSTSVM and observes its performance with respect to #cores, #instances, and #features on synthetically generated datasets. Multiple datasets are generated with a different number of data instances; 1, 3, 5, 7, 9, and 11 million, a different number of attributes; 100, 300, 500, 700, and 900. Run-times of DLSTSVM are estimated with varying number of cores, instances, and features in the following subsections.

(i) Number of cores versus computation time: Several graphs depicted in Figures 7 and 8 are drawn between cores (at x-axis) and computation time (at y-axis). Graphs are plotted for 1, 3, 5, 7, 9, and 11 million data instances respectively. Multiple curves in each Figure 7a–8f shows the run-time for 100, 300, 500, 700, and 900 attributes. Run-time decreases with addition of executor-cores in the cluster. This decrement in

time however decreases gradually because addition of more machines puts communication overheads, meta-information management cost, and task scheduling delays too. Also, more attributes results in more computation time.

(ii) Number of data instances versus computation time: Figures 9 and 10 analyze the computation time with increasing number of data instances where each graph corresponds to a fixed number of cores. It establishes the scalability of DLSTSVM with respect to large-scale data instances. Machines in cluster do not hang even for 11 million instances whereas LSTSVM is unable to handle even 1 million instances. Multiple curves in each Figures 9a–9d and 10a–10c shows the run-time for 100, 300, 500, 700, and 900 attributes. It is evident from Figures 9a to 10c that computation time of DLSTSVM behaves linearly with respect to increase in data instances. However, for a higher number of attributes ($>= 300$), the slope of the curve gets steeper.

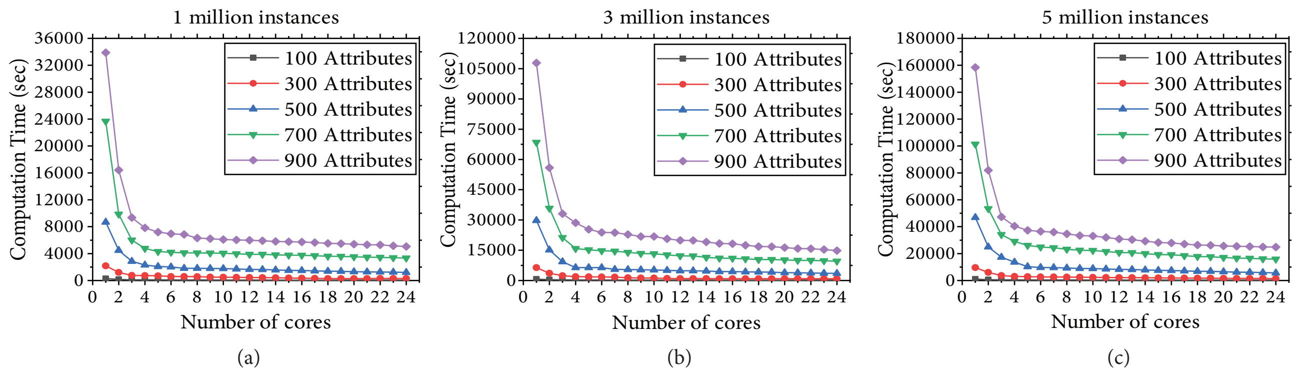

Figure 7. Run-time performance for different cores.

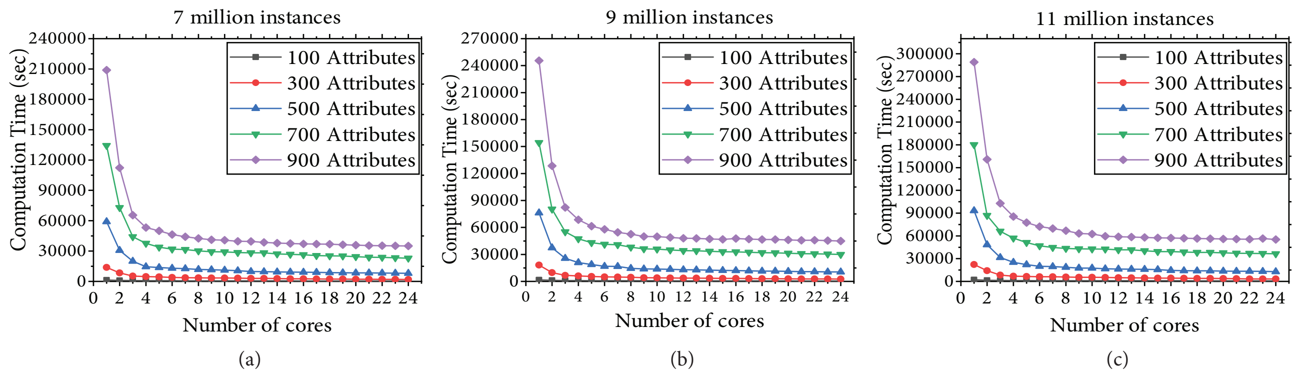

Figure 8. Run-time performance for different cores.

(iii) Number of attributes versus computation time: Each graph in Figures 11 and 12 is plotted for execution time against varying number of attributes and for a fixed value of data instances. Multiple curves in each graph are drawn corresponding to different number of cores from among 24 cores available. All plots show similar pattern of performance. It is also evident that greater the number of attributes, the higher is the computation time. It is also observed that computation time inflates as we go beyond 300 attributes which supports the computational analysis for DLSTSVM given in subsection 4.2.

(iv) 3S based performance evaluation of DLSTSVM: Performance of distributed parallel algorithms can be assessed along 3S evaluators; Sizeup, Speedup, and Scaleup. These parameters specify three different aspects

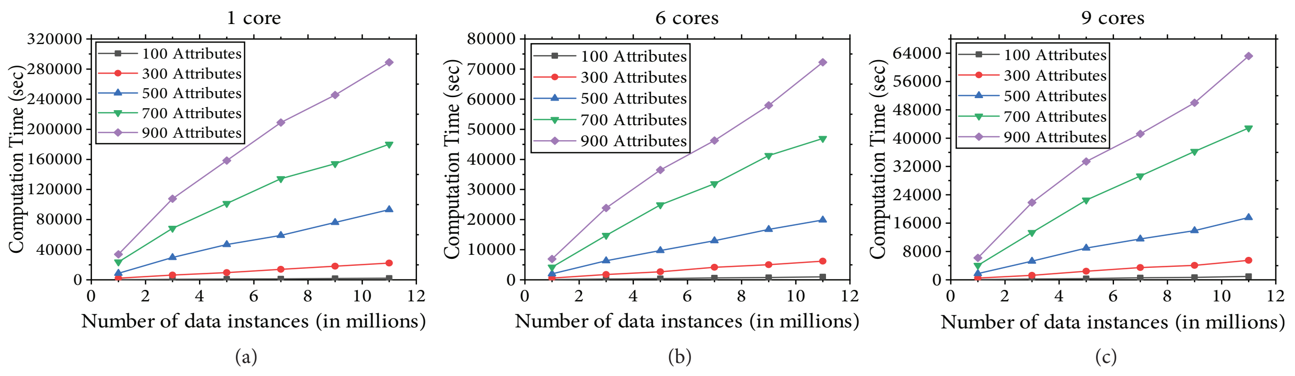

Figure 9. Run-time performance for different data instances.

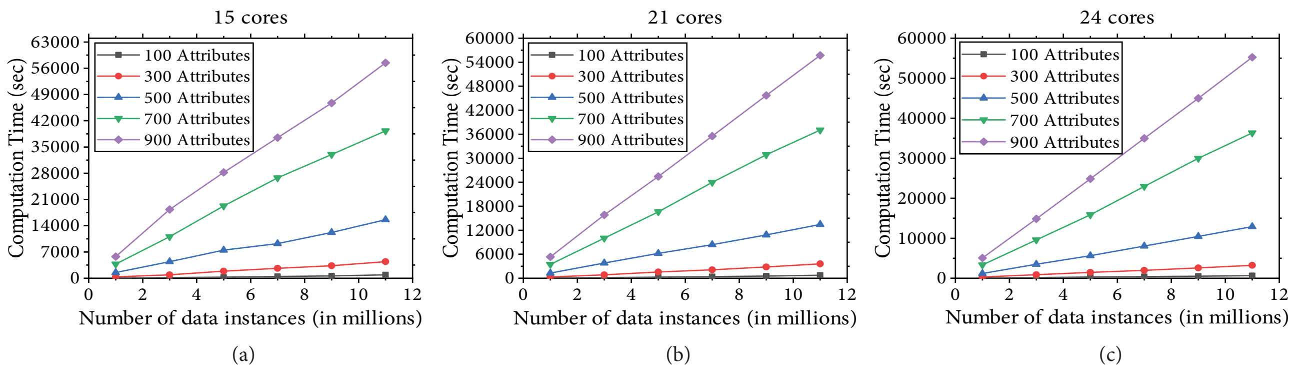

Figure 10. Run-time performance for different data instances.

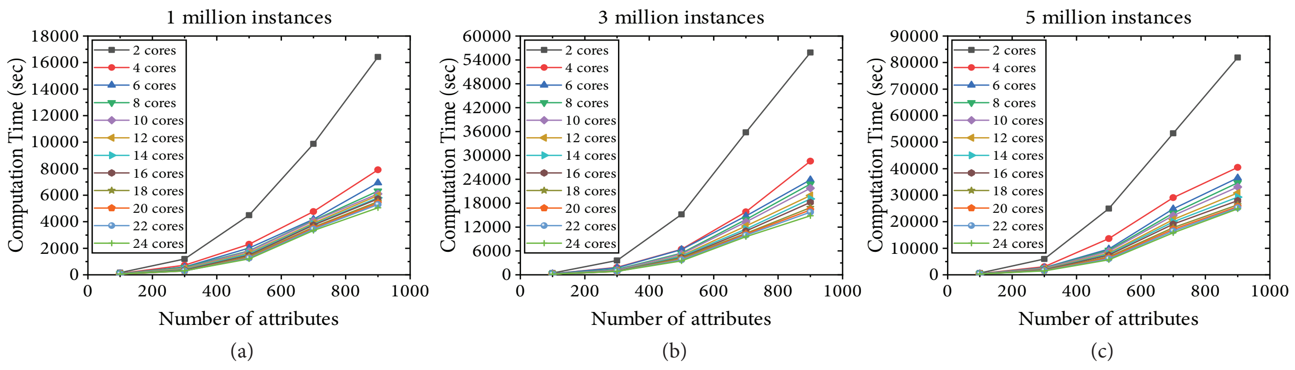

Figure 11. Run-time performance for different attributes.

of a distributed parallel algorithm. Description and significance of each parameter along with the measurements for DLSTSVM is illustrated in following subsections.

Sizeup:- Performance evaluation parameter Sizeup is a measure of computation time of the algorithm with respect to increment in input size keeping number of computation unit fixed. For an ideal situation graph, depicting Sizeup of an algorithm should be linear. However, practically Sizeup may be sublinear. Figure 13a shows Sizeup of DLSTSVM for different computation units (executor-cores) where multiple curves show the Sizeup for 100, 300, 500, 700, and 900 attributes respectively. Evidently, Sizeup of DLSTSVM is almost linear.

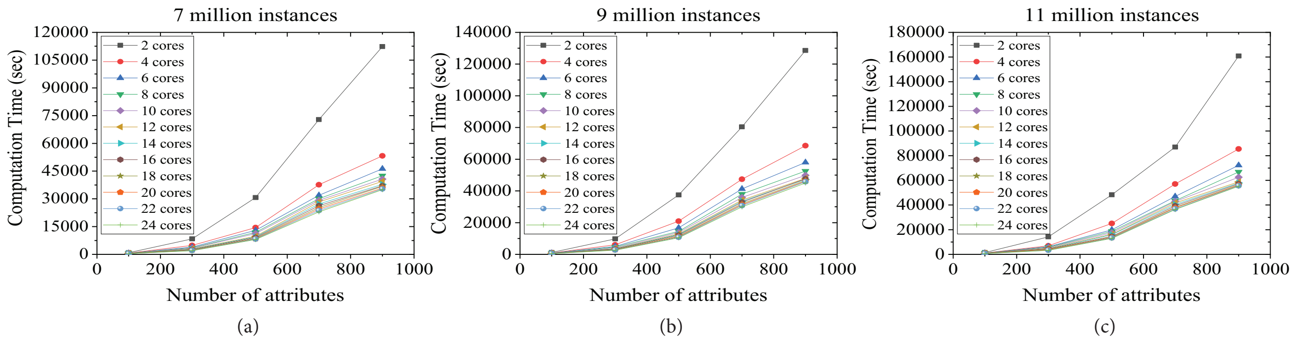

Figure 12. Run-time performance for different attributes.

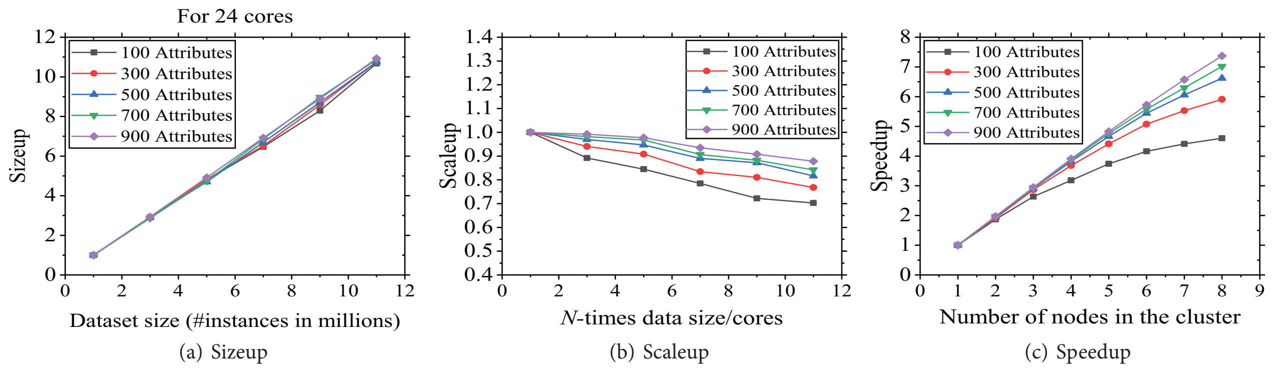

Figure 13. 3S based performance evaluation of DLSTSVM.

Scaleup:- This parameter captures the behavior of algorithm in case when data size and computation units are scaled in same ratio. Let a system with 1 computation unit takes time t on data size d. If k times larger computing system takes t time on kd size data, then the system scales perfectly. However, perfect scalability is practically typical to achieve. Figure 13b specifies Scaleup of DLSTSVM for different settings of attributes. To measure Scaleup, number of executor cores in the cluster is set to 1, 3, 5, 7, 9, and 11 corresponding to 1 million, 3 million, 5 million, 7 million, 9 million, and 11 million data instances respectively. It is observed that Scaleup degrades with rise of executor cores (or #machines) due to inter-machine communication cost, task scheduling delays, uneven machine performance and skewed data distribution. In case of lesser attributes such as 100, all afore-mentioned overheads are dominant over computation cost, hence, degrade in Scaleup is maximum for it. However, in case of more attributes (300 and above) and with 9 to 11 million data instances, Scaleup varies from 88% to 70%, whereas with 1 to 9 million data instances between 100% to 88% which is quite good.

Speedup:- Speedup describes the behavior of a distributed parallel algorithm with increase in number of machines in the cluster keeping the data size fixed. Presented work evaluates speedup over 1 to 8 nodes/machines (i.e. number of cores from 1 to 24) in the cluster. Figure 13c depicts speedup where different curves corresponds to number of attributes 100, 300, 500, 700, and 900 respectively. Here, number of instances is fixed at 11 million. DLSTSVM exhibits prominent speedups. For number of attributes >= 300 speed up is extremely good. Speedup decreases with adding more number of nodes in the cluster which is expected behavior of any scalable distributed algorithm. For 100 attributes, the decrement in speedup is more because of dominance of communication and scheduling overheads over computation costs. All discussions regarding the performance of DLSTSVM for the 3S evaluators are summarized in Table 5.

Table 5. Summary of performance of DLSTSVM for 3S evaluators.

Performance evaluators	# Data instances	Attribute range		
		100	100-300	300
Sizeup	1–11 million	Almost Linear		
		Range of Scaleup achieved		
Scaleup	1–7 million	100%–79%		100%–89%
	7–11 million	84%–70%		93%–82%
		Speedup achieved		
Speedup	1–11 million	Average	Good	Very good

6. Conclusion and future work

In this research work a novel technique DLSTSVM is proposed which achieves scalability by using MapReduce to process distributed data partitions on cluster of machines in parallel fashion. It not only averts memory and computational bottlenecks of existing nonparallel version LSTSVM, but also achieves significant speedups too without losing predictive accuracy. Detailed experimental comparison with existing nonparallel versions of SVM and LSTSVM show that DLSTSVM exhibits speedups in running-time and achieve scalability with respect to large scale datasets. The number of attributes has more impact on run-time. DLSTSVM shows speedup 3x to 20x performance gains. Speedups performance of DLSTSVM is evaluated along well known 3S criterion, i.e. Sizeup, Scaleup, and Speedup, and found to be effective. Sizeup of DLSTSVM is almost linear. For attributes above 300 and data instances 7 to 11 million a Scaleup of 77% to 53% is observed, whereas for 1 to 7 million data instances, a Scaleup of 100% to 70% is achieved effectively. DLSTSVM exhibits prominent speedups and is extremely good for number of attributes $\geq$ 300.

References

[1] Cortes C, Vapnik V. Support-vector networks. Machine Learning 1995; 20 (3): 273-297.

[2] Khemchandani R, Chandra S. Twin support vector machines for pattern classification. IEEE transactions on pattern analysis and machine intelligence 2007; 29 (5): 905-910.

[3] Kumar MA, Gopal M. Least squares twin support vector machines for pattern classification. Expert systems with applications 2009; 36 (4): 7535-7543.

[4] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM 2008; 51 (1): 107-113.

[5] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: distributed data-parallel programs from sequential building blocks. InProceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007; 59-72.

[6] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, 10–10. Berkeley, CA, USA: USENIX Association, 2010.

[7] Inoubli W, Aridhi S, Mezni H, Maddouri M, Nguifo EM. An experimental survey on big data frameworks. Future Generation Computer Systems 2018; 86: 546-564.

[8] Bal H, Pal A. Parallel and distributed machine learning algorithms for scalable Big Data analytics. Future Generation Computer Systems 2020; 108: 1159-1161.

[9] Zhang M L, Zhou Z H. A review on multi-label learning algorithms. IEEE Transactions on Knowledge and Data Engineering 2013; 26 (8): 1819-1837.

[10] Hsieh CJ, Si S, Dhillon I. A divide-and-conquer solver for kernel support vector machines. In: International Conference on Machine Learning 2014, pp. 566-574.

[11] Chang CC, Lin C J. LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) 2011; 2 (3): 1-27.

[12] Bordes A, Ertekin S, Weston J, Bottou L. Fast kernel classifiers with online and active learning. Journal of Machine Learning Research 2005; 6: 1579-1619.

[13] Graf H, Cosatto E, Bottou L, Dourdanovic I, Vapnik V. Parallel support vector machines: the cascade svm. Advances in Neural Information Processing Systems 2004; 17: 521-528.

[14] Zhang K, Lan L, Wang Z, Moerchen F. Scaling up kernel svm on limited resources: a low-rank linearization approach. In: Artificial Intelligence and Statistics 2012: 1425-1434.

[15] Le Q, Sarlós T, Smola A. Fastfood-computing hilbert space expansions in loglinear time. In: International Conference on Machine Learning 2013 Feb 13, pp. 244-252.

[16] Selvaraj SK, Decoste D M, inventors; Altaba Inc, assignee. Building support vector machines with reduced classifier complexity. United States patent US 7,630,945. 2009 Dec 8.

[17] Moody J, Darken CJ. Fast learning in networks of locally-tuned processing units. Neural Computation 1989; 1 (2): 281-294.

[18] Sartakhti JS, Afrabandpey H, Ghadiri N. Fuzzy least squares twin support vector machines. Engineering Applications of Artificial Intelligence 2019; 85: 402-409.

[19] Chen S, Cao J, Chen F, Liu B. Entropy-based fuzzy least squares twin support vector machine for pattern classification. Neural Processing Letters 2020; 51 (1) :41-66.

[20] Richhariya B, Tanveer M. Universum least squares twin parametric-margin support vector machine. In: 2020 International Joint Conference on Neural Networks (IJCNN) 2020; 19: 1-8.

[21] Mir A, Nasiri J A. KNN-based least squares twin support vector machine for pattern classification. Applied Intelligence 2018; 48 (12): 4551-4564.

[22] Zhu K, Wang H, Bai H, Li J, Qiu Z, Cui H et al. Parallelizing support vector machines on distributed computers. In: Advances in Neural Information Processing Systems 2008, pp. 257-264.

[23] Boyd S, Parikh N, Chu E. Distributed optimization and statistical learning via the alternating direction method of multipliers. Now Publishers Inc; 2011.

[24] Guan L, Sun T, Qiao LB, Yang ZH, Li DS et al. An efficient parallel and distributed solution to nonconvex penalized linear SVMs. Frontiers of Information Technology & Electronic Engineering 2019; 5: 1-7.

[25] Singh D, Mohan CK. Projection-SVM: distributed Kernel Support Vector Machine for Big Data using Subspace Partitioning. In: 2018 IEEE International Conference on Big Data (Big Data) 2018 Dec 10, pp. 74-83.

[26] Shah R, Zhang S, Lin Y, Wu P. xSVM: Scalable Distributed Kernel Support Vector Machine Training. In: 2019 IEEE International Conference on Big Data (Big Data) 2019; 9: 155-164.

[27] Musicant D R. NDC: normally distributed clustered datasets. Computer Sciences Department, University of Wisconsin, Madison. 1998.

[28] Tavara S. Parallel computing of support vector machines: a survey. ACM Computing Surveys (CSUR) 20198; 51 (6):1-38.

[29] Landset S, Khoshgoftaar TM, Richter AN, Hasanin T. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. Journal of Big Data 2015; 2 (1): 24.

[30] University of California, Irvine, Datasets

[31] Chang CC, Lin CJ. LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) 2011; 2 (3):1-27.

[32] Zhang K, Lan L, Wang Z, Moerchen F. Scaling up kernel svm on limited resources: a low-rank linearization approach. In: Artificial Intelligence and Statistics 2012, pp. 1425-1434.

[33] Le VL, Lauer F, Bako L, Bloch G. Learning nonlinear hybrid systems: from sparse optimization to support vector regression. In: Proceedings of the 16th International Conference on Hybrid systems: computation and control 2013, pp. 33-42.

[34] Hsieh CJ, Si S, Dhillon I. A divide-and-conquer solver for kernel support vector machines. In: International Conference on Machine Learning 2014, pp. 566-574.

[35] Çatak FÖ, Balaban ME. A MapReduce-based distributed SVM algorithm for binary classification. Turkish Journal of Electrical Engineering & Computer Sciences 2016; 24 (3): 863-873.