DDMSVM: Data-Driven Modular Support Vector Machines

Vibhatha Abeykoon[1], Minje Kim[1] and Geoffrey C. Fox[1]

Indiana University, Bloomington, Indiana 47408, USA
vlabeyko@iu.edu, minje@iu.edu, gcf@iu.edu
Intelligent Systems Engineering: https://www.engineering.indiana.edu/

Abstract. Support Vector Machines (SVM) is one of the unique machine learning algorithms which is a computationally intensive when it comes to classify millions of data points in a data set. There are different methods that have been proposed to solve the problem in a faster way, we propose DDMSVM, Data-Driven Modular Support Vector Machines providing a much faster sequential implementation by means of a model based training approach depending on randomly picking partitioned data and selectively picking partitions by using a simplified Sequential Minimal Optimization (SMO) approach on top of a data driven training model. Single Data Single Model (SDSM) and Multiple Data Multiple Model (MDMM) approaches proposed by us provides a much faster convergence with descent training accuracy with reference to the very low training time. We prove our concept by experimentally showing that randomized data partitioning and grouping can lead to a faster convergence with higher accuracy by implementing a novel data driven computational model on top of the proposed SDSM and MDMM models. We implement a novel data grouping approach called Correlation Modular Approach (CMA) enhancing the accuracy and improving the convergence rate significantly. In the experiments, our data driven computational model outperforms LibSVM and DC-SVM which can be considered as benchmark sequential implementations in Support Vector Machines implementations. With Webspam data set we get a speed up of 7.835 times with respect to DC-SVM and that ratio is 52.63 times with respect LibSVM performance along with an accuracy of 99.55%. Our objective is to show that a randomized data partitioning model with a less complex computational model can provide a much faster convergence with a significant accuracy allowing users to train Support Vector Machines, even for larger data sets which cannot be sequentially processed due to memory related problems.

Keywords: svm, smo, correlation, model, randomized data pooling, data-driven

1 Introduction

In the realm of machine learning, Support Vector Machines (SVM) [5] by Cortes and Vapnik plays a major role in classifying data in a much faster manner. The

focus of SVM is to identify the boundary between different classes to separate one class from the other class. The main concern with SVM is that it is very computational intensive because of the nature of the objective function. It is an exact quadratic problem which is a computational intensive problem. There are couple of sequential implementations like DC-SVM [1], LibSVM [2] and SMO [4] which can be considered as most prominent sequential implementations to solve the SVM problem. SVM becomes a computation expensive method depending on the number of data points in the data set. For a data set having few hundreds of Mega Bytes can cause memory issues when the algorithm has to compute a kernel matrix of size $n \times n$ where n is the number of data points in the data set. To overcome this problem there has been different researches done considering random samples via bootstrap techniques [12], described in SVM ensemble. But the performance improvement or the nature of execution on very large data sets has not been elaborated for bigger data sets. With bigger data sets the effect of random data partitioning causes accuracy degradation to a certain level. The approach of random data partitioning is a significant concept when it comes to training larger data sets in a sequential manner with the capability of improving up to a parallel executing framework. In LibSVM, DC-SVM and most of the SVM based implementations, the core algorithm used is the SMO algorithm which is computationally expensive.

In our proposed method we are trying to use a simplified version of the sequential minimal optimization algorithm to improve the performance of the algorithm by means of reducing the execution time. And also we have proposed a data-driven modular way of training random samples based on data partitioning before feeding the data set to the SVM algorithm. The modular architecture is supported by a data partitioning engine which partitions the data depending on two concepts. Random data partitioning and correlation based data partitioning. By experiments we have shown that our approaches provide a much faster convergence and higher accuracy for larger data sets with close to half a million data points by experimenting on standard LibSVM data sets with hundreds of Mega Bytes to Giga Byte level.

The section 2 describes about the research work done on SVM regarding different aspects of optimizing SVM, section 3 gives a brief idea about the mathematical approach taken in SVM, section 4 describes the proposed methodology to improve the performance of SVM using the data-driven modular based approach.

2 Related Work

In referring to vivid aspects of the SVM training process, the core algorithm has been mutated in ways such that it provides a much more improved performance. Here the quadratic problem solving mechanism has been improved using different mathematical tools. Support Vector Networks or SVN by Cortes and Vapnik [5] can be considered as the first proposed method on the SVM problem which employs a chunking strategy to solve the objective function. In SVM, the objective

function is not just an objective function that has to be maximized, but also it must successfully agree with the constraints imposed on it. Chunking algorithm proposes a way to find the boundary lines with respect to the data points by considering the Lagrange multipliers and it tries to optimize a set of Lagrange multipliers at a time. This mechanism is a computationally expensive approach. Sequential Minimal Optimization or SMO [4] is a research conducted on improving the performance of SVM algorithm implemented in chunking method by Cortes and Vapnik. In this algorithm, Platt explains how the existing chunking algorithm can be surpassed by means of considering the sequential minimal optimization concept which deals with two Lagrange multipliers at a time. This algorithm was used in most of the sequential implementations of SVM. LibSVM, a benchmark SVM implementation which can be considered as one of these implementations employing a type of Sequential Minimal Optimization [3]. For smaller data sets, these approaches provide a much faster convergence with higher accuracy. In case of processing bigger data sets with hundred thousands of data points to millions and billions of data points, the memory issues and computational expensiveness acts as a barrier when traditional sequential implementations are used.

In order to solve the problem with a sequential approach, DC-SVM [1] a divide and conquer model of SVM was developed by considering a data driven model along with pre clustering the data before training. DC-SVM employs the LibSVM as the core algorithm to do the classification. DC-SVM can be considered as a faster sequential version of SVM which has benchmarked its performance on top of a number of SVM implementations by considering larger data sets up to a half a million data points. The bottleneck in most of the SVM implementations comes with the memory boundaries and high number of Lagrange multipliers that has to be calculated to optimize the objective function. In order to get a faster convergence, Simplified Sequential Optimization algorithm has been proposed by Yang et.al [7]. In the original SMO algorithm it takes a longer time to calculate the Lagrange values in the optimizing the objective function, and there are heuristics to do this, but in a simpler way without over optimizing or optimizing Lagrange values to get maximum of the objective function. If this maximization can be limited to a certain level, the algorithm can optimize Lagrange values in a much faster way. This concept is discussed in [8].

3 Background of Support Vector Machines

In understanding a better model to train Support vector machines, the mathematics behind SVM must be clarified. Support Vector Machines or SVM is an exact quadratic problem which involves a larger amount of memory and a very high computation time in solving the problem in a sequential manner. The main concept in SVM is to classify a data set into the given classes by means of understanding which data points from the data set involves in contributing to decide that decision boundary which claims the boundaries for each class. In SVM, the data points which are known as support vectors are the data points

which involves in finding the decision boundaries and the rest of the data points are called non support vectors.

The main objective in this algorithm is to find out these support vectors. The Support Vector Machines algorithm understands which data points out of a data sets are contributing to decide decision boundaries in a classification problem.

The decision boundary has to be detected in such a way that the gap between the data points separated by boundaries takes a maximum value. Mathematical representation of this problem comes with an objective function that has to be optimized with respect to a constraint. The objective function is defined so that the maximum objective value is achieved in such a manner that it won't violate a certain constrains coming with KKT or Kursh Kuhn Tucker conditions. In identifying this fact, the mathematical trick used to formulate this problem is by assigning a Lagrange multiplier to each data point and maximizing the objective function meeting the constraint. The final output contains a set of Lagrange values greater than zero and the rest equal to zero.

The zero Lagrange multipliers are eliminated and non-zero Lagrange multipliers are used to solve the problem. In order to understand these support vectors, the standard implementation is to solve this exact quadratic problem using Lagrange multipliers and iteratively solving each multiplier and after an expected tolerance value is obtained in the training process, the algorithm exits to provide the final set of Lagrange multipliers which decide the weight vector, maximizing the objective function under the imposed constraints. In this passion, the training model can be obtained. There are many approaches taken by different researches on different aspects of SVM. A set of SVM implementations try to optimize the training procedure by dividing data sets in to small portions and training the SVM in a iterative passion on these partitions of data. In these approaches the core of the algorithm or the quadratic problem solving section is not being optimized that much, but the data partitioning and training procedure has been improved. In SVM there are two challenging things, first one is the memory issue, in calculating the Lagrange multipliers it involves a much bigger kernel matrix when the data set grows and it is directly proportional to the number of data points in the data set. The second challenge is the time consuming procedure of calculating the Lagrange multipliers. Sequential Minimal Optimization or SMO algorithm is one of the most prominent approaches used to optimize the objective function. In this paper, our main focus is towards a simplified version of SMO in order to increase the performance of the algorithm with reference to execution time.

3.1 Sequential Minimal Optimization Algorithm

In SMO algorithm the main focus is to make the support vector identification step much faster. Because through out the SVM algorithm the main problem that has to be dealt with is the maximization of the objective function which is a quadratic problem, **QP** subjecting to constraints. In SMO approach, instead of calculating all the Lagrange multipliers for the duality problem in equation

1, it calculates two Lagrange multipliers at a time and optimizes those two at each iteration. So per iteration only two Lagrange multipliers will be optimized.

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j d_i d_j k(x_i, x_j) \tag{1}$$

$$u_i = \sum_i^N d_i \alpha_i k(x_i, x) - b \tag{2}$$

$$\alpha_i = 0 \Longleftrightarrow d_i u_i \geq 1; 0 < \alpha_i < C \Longleftrightarrow d_i u_i = 1; \alpha_i = C \Longleftrightarrow d_i u_i \leq 1 \tag{3}$$

$$0 \leq \alpha_i \leq C \tag{4}$$

$$\sum_{i=1}^N y_i \alpha_i = 0 \tag{5}$$

In referring to equation 1, it is clear that the equation is about a quadratic problem which will take matrix format calculation a long time to simplify. According to Platt's idea the mechanism he provides is to use two Lagrange multipliers at a time and optimize the whole set of Lagrange multipliers in iterative passion. All the Lagrange multiplies or **LM** must be obeying KKT (Karush-Kuhn-Tucker) conditions explained in equation 3. In SMO, rather than focusing on the whole set of data points each having it's own **LM**, it focuses on two of **LM** at time and optimize them to the best and move to the next data point. Here it considers a particular data point and then focuses on finding a paring data point which can maximize the objective function 1 in with a maximum impact subjecting to the constraints, 4 and 5. In SMO two heuristics are being used to determine these **LM**.

Heuristic 1 Determining the first **LM** is the first objective of this heuristic. In the SVM algorithm the outer loop iterates through the whole data set and determines the data points which are violating KKT conditions explained in equation 3. In case of finding an example or data point violating KKT condition it is being selected as eligible for optimization. After one pass through the entire data set, outer loop iterates through the **LM** that are neither 0 nor C which are known as non-bound examples or data points. In this looping, the KKT violating data points are being optimized. The outer-loop goes through all the non-bound examples until they obey KKT conditions within a given tolerance value which is generally selected as 0.001. This is basically the first heuristic explained in a simple manner.

Heuristic 2 In selecting the 2nd **LM**, it must maximize the step taken in the optimization of objective function. So the second **LM** must be selected in a manner that it satisfies this requirement. Here we define a term $E_i = u_i - d_i$ and in this step, the **LM** must be selected in a way such that $|E_1 - E_2|$ gets the maximum value. In order to fulfill this requirement, in case of $E_1 > 0$, select E_2 with the minimum error and in case of $E_1 < 0$, select E_2 with maximum error. This is the basic thing that has to be done. In order to check this condition, there is a code level implementation done to avoid any unusual circumstances regarding this scenario. This step is a computational expensive step which searches for the highest optimizing pair. The unusual circumstance can happen due to having another input being identical to the current input vector. This causes the original algorithm to loop through non-bound examples again until it finds a possible data point which provides maximum optimization. This way the second heuristic is completed after finding a Lagrange multiplier providing the maximum optimization for the objective function.

In SMO, the main problem is it needs more space when forming the Kernel matrix which has the dimension of $N \times N$ and refers to the number of data points in the training data set. For a larger problem, in pragmatical wise it causes heap overflow problems in implementing this approach using a programming language. And also the approach used for finding unusual circumstances in SMO causes the algorithm to slow down in case of it finding a unusual data points which can be rare in certain data sets and it is a data set dependent feature. These are the two main problems regarding SMO algorithm.

3.2 Simplified Sequential Minimal Optimization Algorithm

In the proposed method, we use the Simplified Sequential Minimal Optimization (SSMO) method proposed by [7] and [8] to define the core SVM algorithm. In our DDMSVM framework, SSMO implementation is done using a sequential version of the algorithm in Java. This implementation is used as the core of the DDMSVM framework. In SMO algorithm, the α_i and α_j are selected such that it maximized the objective function as much as possible. By giving away a portion of the maximization, the algorithm can be run in a faster manner. When it comes to a much larger data set this factor can be very important as it has to optimize Lagrange multipliers from thousands to millions of data points. In Simplified Sequential Minimal Optimization (SSMO) algorithm, the algorithm iterates through the Lagrange multipliers and if they don't follow the KKT conditions within a given numerical tolerance, the paring Lagrange multiplier is being selected in a randomized way by considering the rest of the Lagrange values and then work on optimizing these two Lagrange multipliers. This choice can be higher advantage in running the algorithm in a much faster manner. The algorithm terminates when non of the Lagrange multipliers are changed. This is the unique nature in this approach discussed in [8]. This approach gives away a accuracy for a certain level. In this paper, we propose a data-driven approach to make the training process much faster to get a higher accuracy by grouping data in a correlation based modular approach.

4 Data-Driven Modular Support Vector Machines

In this paper, we propose a data-driven model to train SVM using a simplified sequential minimal optimization approach discussed in 3.2. The data-driven approach initially partitions the data into smaller data pools in a way that each data pool contains an equal amount of data. In pooling we kept the data points from 500-1000 data points for smaller data sets and we kept this in couple of thousands for much larger data sets. After main partitioning is done data chunks are randomly chosen for training. The unique nature in our approach is that the randomization used in optimizing the objective function.

In this paper, we propose three training methodologies to run SVM in a faster way. First method we propose is called Single Data Single Model (SDSM) 4.2, the second method we propose is Multiple Data Multiple Model (MDMM) 4.3 and the third method is known as Correlation Modular Approach (CMA) 4.4. Our methodology employs our own implementation of SSMO 3.2 and it is linked with three different models of training and they are described in sections 4.2, 4.3 and 4.4 followed by a data partition engine defined in two different ways described in sections 4.1 and 4.1.

4.1 DPE: Data Partition Engine

The system is fed with Libsvm formatted data sets in LibSVM archives [2]. In the initial stage the data is converted to ISESVM format (Intelligent Systems Engineering SVM data format) by separating feature points and respective labels into separate files using a csv format. The data partition engine partitions the data depending on the number of data chunks requested by the user as a parameter when running the framework. The partitioned data is saved locally into the disk for the further access in the training, cross-validating and testing process. The objective of the data partition engine is to provide faster convergence for the SSMO algorithm. The partitioned data chunks are ranked with a label. In the training procedure the data partitions are called upon the label name. The data partition model takes two different forms depending on the partitioning models. In this paper we propose two methods to partition data namely, Random Data Partitioning 4.1 and Correlation Based Data Partitioning 4.1.

RDP: Random Data Partition In this step data is partitioned without considering the relationship between data points and they are being shuffled and partitioned into m chunks depending on the user input. This data partitioning mechanism is employed by training approaches described in section 4.2 and 4.3.

CDP: Correlation Based Data Partition In CDP, the correlation among data points are calculated by randomly selecting a data point and with respect to that data point the correlation factor to each data point is calculated. The correlation factor can be positive or negative. Depending on these categories, data is first clustered into two different groups. Then the RDP operation is

applied on each cluster and a label is provided for each group. In selecting correlation groups for training and testing data sets the initially selected random sample will be governing the correlation value for each data point. In the initial round, the correlation values for all the data samples in training and testing data sets is being calculated and the mean value of the correlation values is being selected to group training and testing data sets into two sub groups.

The reasoning behind this partitioning was deduced by experiments. In binary classification or multi-class classification, each cluster or class will have a unique correlation factor among the data points belonging to each class or cluster. The idea was to find out the margin for this factor in a binary classification scenario. This can be extended for a multi-class problem by clustering the correlation values (clustering scalars) and finding out the centroids and obtaining the margin levels by the centroid values of the each cluster. This way the data can be grouped in a much faster way. In the training phase each group will be trained with a unique model. The training model will be explained in detail in section 4.4.

4.2 SDSM: Single Data Single Model

SDSM methodology employs a single partition of a data set out of m number of data partitions created in the first stage of the data processing as described in section 4.1. Then a random partition is selected from m models. That is why it is called as single data. This selected partition is submitted as the training data set for the SSMO algorithm and the training model is generated and it is written to the disk using ISE Model Format. We have defined a XML based format to record the model weight values, bias values and important statistics in the training process. This is the first phase of the SDMM approach. In the second stage, we pick k number of random data partitions from the rest of the partitions excluding the partition used for training in the first stage. Now these data sets are being tested with the trained model in the first stage by using these a single data partition and the accuracy is recorded. In this stage we calculate the average accuracy for k training models and we keep a base accuracy limit and checks whether it exceeds the base accuracy expected in the experiment, if this exceeds the threshold accuracy, we terminate the modular training approach. If the calculated average accuracy is lesser than the expected threshold accuracy (currently we keep the lower boundary in a range of 80% - 90% in the default configurations), another data partition is randomly chosen (excludes already trained data partitions) and training is done again. This process will go on, until a particular training model exceeds the threshold accuracy. Up to this stage we calculate the accuracy by considering k number of cross-validation data sets depending on a random computation strategy. This strategy gives away a portion of accuracy. For the tested data sets, our training model exits after the first level of training for the tested data sets. In this way we finalized the trained model weights. In the next stage we use the finalized model to do the predictions. Then we selects user defined number of data partitions from the pool of partitioned testing data set and the predictions are being recorded. The system records the

accuracy of each data partition and calculate the average accuracy for the overall data set. The algorithm for SDSM training is described in Algorithm 1.

Algorithm 1 SDSM Training Algorithm

INPUT:
Xtr: Training Feature Data Partition Id,
Ytr: Training Label Data Partition Id,
Xts: Testing Feature Data Partition Id,
Yts: Testing Label Data Partition Id
OUTPUT: Save Training Models, Save Training Statistics

```
1: procedure MAIN(Xtr,Ytr,Xts.Yts)
2:     while (avg_accuracy ≤ defined_tolerance) do
3:         procedure SDSM(Xtr_i, Ytr_i)  ▷ random data partition for each iteration
4:             α ← 0                                  ▷ Initializes Lagrange values
5:             w ← 0.1                                  ▷ Initializes weight vector
6:             b ← 0                                              ▷ Initializes bias
7:             procedure SMO(α, b, w, Xtr_i, Ytr_i)
8:                 Save Model ← model
9:                 return model
10:            procedure SDSM_TEST_ACCURACY(Xts_i, Yts_i, k, model)
11:                return avg_accuracy
```

In prediction, the saved model from the training process has to be provided when we take the prediction accuracy for the test data sets.

4.3 MDMM: Multiple Data Multiple Model

MDMM approach is functionally similar to the SDSM approach, but we have added multiple data partitions and we train multiple models in order to obtain a concrete definition on the training process. In this algorithm, we choose a m number of random partitions from the training data partition pool and after the training process we get m training models. Then we choose k number of random data partitions from the cross-validation partition pool and each data sample is being trained with each model and average accuracy is being obtained for each model using k data samples. Here we have m number of accuracy values for m models. If we get an average accuracy for m models above the threshold accuracy value that we have set for the training process, the algorithm terminates, if not it again searches for m models randomly (excluding the current samples) and the same process is being run until it reaches up to a the threshold accuracy level. After this stage, we normalize the accuracy distribution and get a weight vector for m number of training models. This weight vector becomes a biased coefficient for the prediction from each model. When the weight is a higher value for a particular model, the effect from that model on the final prediction is higher and if the weight is low, the prediction value is affected less by that model. For

the prediction stage, we selects samples from the testing data pool. MDMM algorithm is described in Algorithm **??** and the weight calculation algorithm is described in Algorithm 3.

Algorithm 2 MDMM Training Algorithm

INPUT:
Xtr []: Training Feature Data Partition Ids,
Ytr []: Training Label Data Partition Ids,
Xts []: Testing Feature Data Partition Ids,
Yts []: Testing Label Data Partition Ids
OUTPUT: Save Training Models, Save Training Statistics

1: **procedure** MAIN(Xtr [],Ytr [],Xts [],Yts [])
2: **while** ($avg\_accuracy \leq defined\_tolerance$) **do**
3: **procedure** MDMM($Xtr_i[], Ytr_i[]$) ▷ random data partition for each iteration
4: $\alpha \leftarrow 0$ ▷ Initializes Lagrange values
5: $w \leftarrow 0.1$ ▷ Initializes weight vector
6: $b \leftarrow 0$ ▷ Initializes bias
7: **procedure** SMO_BULK(α, b, w, $Xtr_i[]$, $Ytr_i[]$)
8: Save Models $\leftarrow$ $model[]$, $weight\_vectors$
9: **return** $models[]$, $weight\_vectors$
10: **procedure** MDMM_TEST_ACCURACY($Xts_i[]$, $Yts_i[]$, k, $models[]$)
11: **return** $avg\_accuracy$ ▷ Return average accuracy for k partitions across m models.

After training process m models are being written to the disk and in the prediction stage we specify the location for saved models in the disk and the prediction is carried out by using data partitions from prediction pool of data sets.

4.4 CMA: Correlation Modular Approach

CMA approach is employed by the process explained in section 4.1. In this approach we implements an additional layer on top of SDSM or MDMM by means of grouping data into two groups. We separate training, validation and testing data sets in to three different pools and each pool is then divided into two sub groups which are with positive correlating data points and negative correlating data points. In this approach we train SDSM or MDMM separately depending on the data group and two types of models are created. By putting data with close correlation and same sign in the same group, we obtain positive correlated model and negative correlated model. For the prediction stage we use these models on pre-processed correlation wise grouped data and continue the approaches described in 4.2 or 4.3 to get the final prediction results for the test data. Algorithm 4, 5 provides the detail algorithm for CMA approach.

Algorithm 3 Weight Calculation Algorithm

INPUT: Accuracy Per Model
OUTPUT: Weight Vector

```
1: procedure WEIGHT CALCULATION(accuracy_per_model[])
2:     model_size ← len(accuracy_per_model)
3:     model_count ← 0
4:     total_accuracy ← 0
5:     while (model_count < model_size) do
6:         total_accuracy+ = accuracy_per_model[model_count]
7:         model_count ++
8:     weights[] ← 0
9:     weight_count ← 0
10:    while (weight_count < model_size) do
11:        weights[weight_count] = accuracy_per_model[weight_count]/total_accuracy
12:        weight_count++
13:    return weights[]
```

Algorithm 4 CMA-SDSM Training Algorithm

INPUT:
CMA_Xtr: Training Feature Data Partition Id,
CMA_Ytr: Training Label Data Partition Id,
CMA_Xts: Testing Feature Data Partition Id,
CMA_Yts: Testing Label Data Partition Id
OUTPUT: Save Training Models, Save Training Statistics

```
1: procedure MAIN(CMA_Xtr,CMA_Ytr,CMA_Xts,CMA_Yts)
2:     while (avg_accuracy ≤ defined_tolerance) do
3:         procedure CMA_SDSM(CMA_Xtr_i, CMA_Ytr_i)          ▷ random data
   partition for each iteration
4:             α ← 0                                  ▷ Initializes Lagrange values
5:             w ← 0.1                                  ▷ Initializes weight vector
6:             b ← 0                                                ▷ Initializes bias
7:             procedure CMA_SMO(α, b, w, CMA_Xtr_i, CMA_Ytr_i)
8:                 Save Model ← positive_model, negative_model
9:                 models ← positive_model, negative_model
10:                return models
11:            procedure SDSM_TEST_ACCURACY(CMA_Xts_i,CMA_Yts_i,k,models)
12:                avg_accuracy ← avg(avg_positive_accuracy, avg_negative_accuracy)
13:                return avg_accuracy
```

Algorithm 5 CMA-MDMM Training Algorithm

INPUT:
CMA_Xtr []: Training Feature Data Partition Ids,
CMA_Ytr []: Training Label Data Partition Ids,
CMA_Xts []: Testing Feature Data Partition Ids,
CMA_Yts []: Testing Label Data Partition Ids
OUTPUT: Save Training Models, Save Training Statistics

procedure MAIN(CMA_Xtr [],CMA_Ytr [],CMA_Xts [],CMA_Yts [])
  while ($avg\_accuracy \leq defined\_tolerance$) **do**
    procedure MDMM($CMA\_Xtr_i[], CMA\_Ytr_i[]$) ▷ random data partition for each iteration
      $\alpha \leftarrow 0$ ▷ Initializes Lagrange values
      $w \leftarrow 0.1$ ▷ Initializes weight vector
      $b \leftarrow 0$ ▷ Initializes bias
      procedure CMA_SMO_BULK(α, b, w, $CMA\_Xtr_i[]$, $CMA\_Ytr_i[]$)
        Save Model $\leftarrow$ $positive\_models[], negative\_models[]$, $weight\_vectors$
        $models \leftarrow positive\_models[], negative\_model[]s$
        return $models[]$, $weight\_vectors$
      procedure MDMM_TEST_ACCURACY($CMA\_Xts_i[]$,$CMA\_Yts_i[]$,k,$models[]$)
        $avg\_accuracy \leftarrow avg(avg\_positive\_accuracy, avg\_negative\_accuracy)$
        return $avg\_accuracy$

After SDSM or MDMM training with CMA approach the models are saved to the local disk along with the weight vectors (only for MDMM approach), in the prediction stage these models are loaded and tested with grouped test data and predictions are obtained.

5 Results

In testing our proposed model, we have used an Intel(R) Core(TM) i7-6700HQ CPU with 2.60 GHz node to test the single node experiments. Currently we have a single node single threaded implementation of this algorithm. We have implemented SSMO, SDSM, MDMM and CMA approaches in Java 1.8. In order to benchmark the results, we use standard LibSVM C++ implementation and DCSVM Matlab (only in Matlab) implementation. For the experiments we have selected 4 data sets. Ijcnn1 [10], Webspam [11], a9a [9] and Heart [2] data sets. When the data sets from the original source doesn't have a testing data set, we partitioned the main data set into 3:2 ratio for training and testing. The MSVM framework was tested with four different data sets under binary classification via linear kernel. In the current phase of the project, our implementation contains a tested version on the linear kernel. The Figure 1 gives the accuracy recorded by each framework with training configuration of C parameter as 1, tolerance of training as 0.001 and gamma value as 2. The execution time for each data set is recorded against each framework and it is depicted in Figure 2, 3, 4, 5.

The SDSM training model training model outperforms LibSVM and DC-SVM under the same initial configurations with respect to accuracy in the Heart data set. SDSM provides a faster convergence with a descent accuracy in all the data sets. This model provides an accuracy above 80% at all times and it provides a ceiling value of 90% accuracy for the current experiments that we have completed. The SDSM model shows that even though the initial training sample is a portion of the original data set, it can still provide a descent accuracy with much faster convergence rate. By sacrificing a small portion like 8%-10%, we can get an average speed up of 2.5 for a9a data set with respect to libsvm and above 600 speed up for webspam data set with respect to libsvm. And also with respect to DC-SVM full for webspam data set a speed up of 10 and a speed up of 10 for the data set a9a. SDSM model proposed by us works well with larger data sets and the results are significant when the data set is larger.

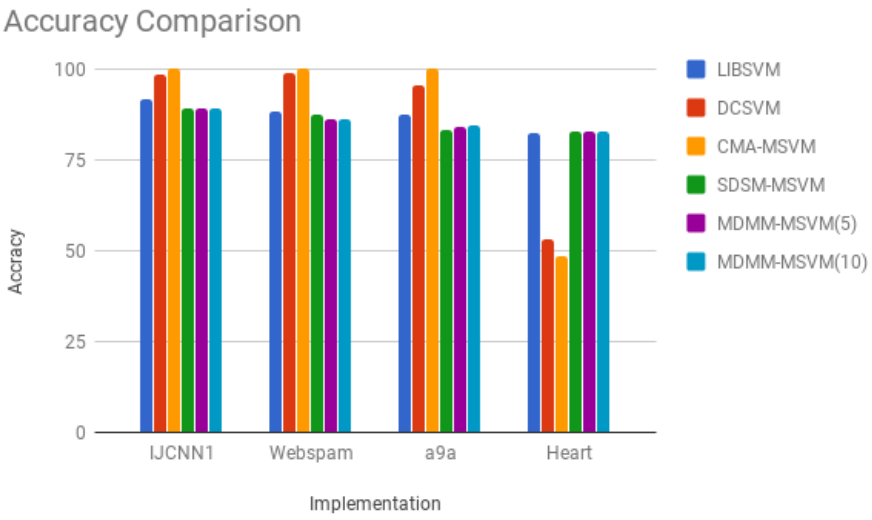

Fig. 1. Overall Accuracy Comparison

MDMM approach was designed to increase the accuracy of the training. But with the current experiments even with 10 models we can increase the accuracy by 1% and that is not much, and it costs a lot of computing time in the serial computation model. The objective of MDMM approach comes strongly with a parallel approach in which we can increase the number of models and increase the accuracy without giving away much convergence time.

CMA approach consumes both SDSM and MDMM approaches, but CMA can be highly effective with both approaches. In our current implementation with MDMM being sequential, the computation overhead is still there, but the

accuracy is very high for both cases. In SDSM approach the CMA speed up is lesser than SDSM standalone approach, but it has the ability to provide a very high accuracy. The speed up obtained using CMA approach with SDSM is 5 to 40 times for overall experiments with respect LibSVM and the speed up values are 16 to 30 with DC-SVM Full. DC-SVM early bears considerably same accuracy but much faster convergence and CMA scores the speed ups between 0.67 to 5.

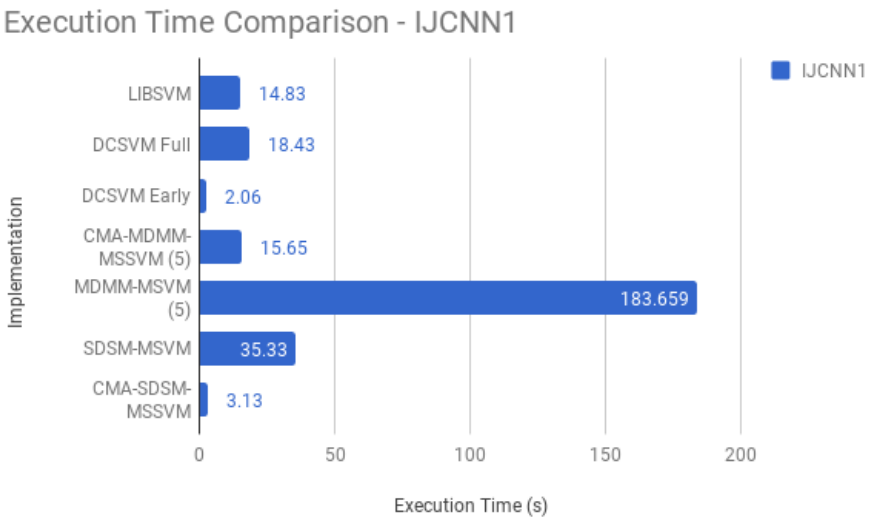

Fig. 2. Execution Time Comparison of IJCNN1 Dataset

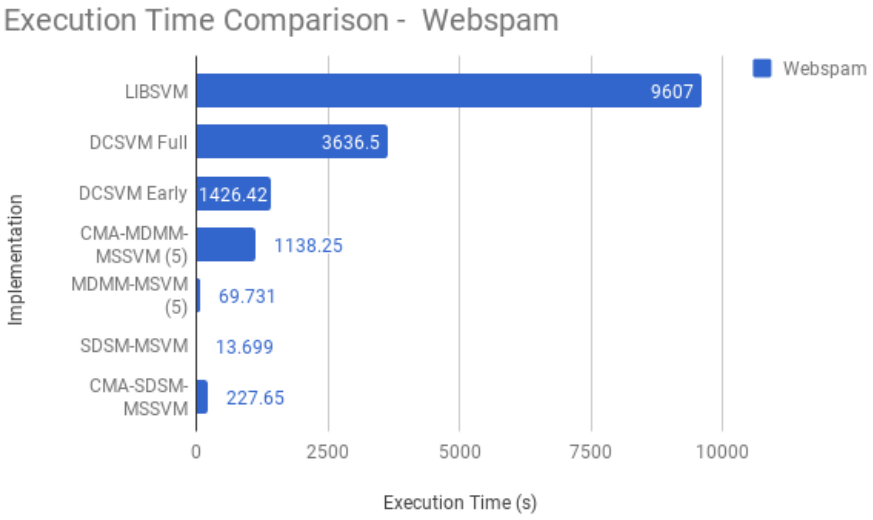

Fig. 3. Execution Time Comparison of Webspam Dataset

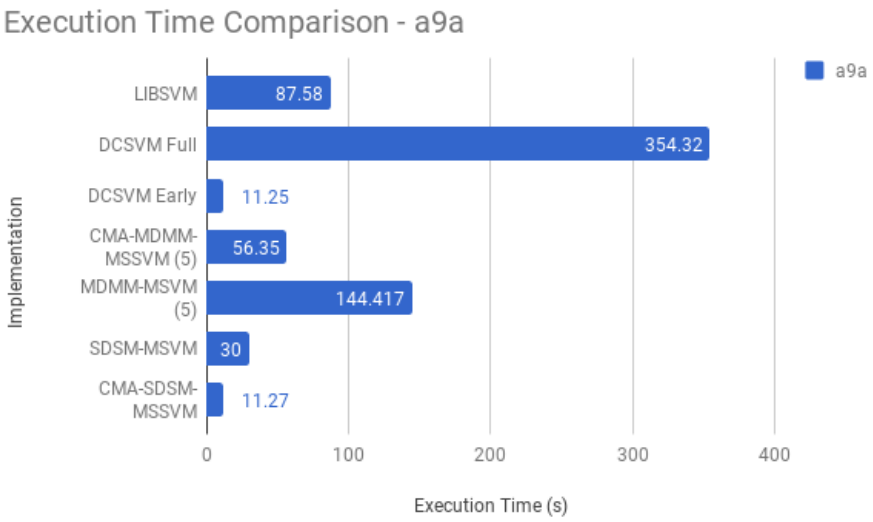

Fig. 4. Execution Time Comparison of a9a Dataset

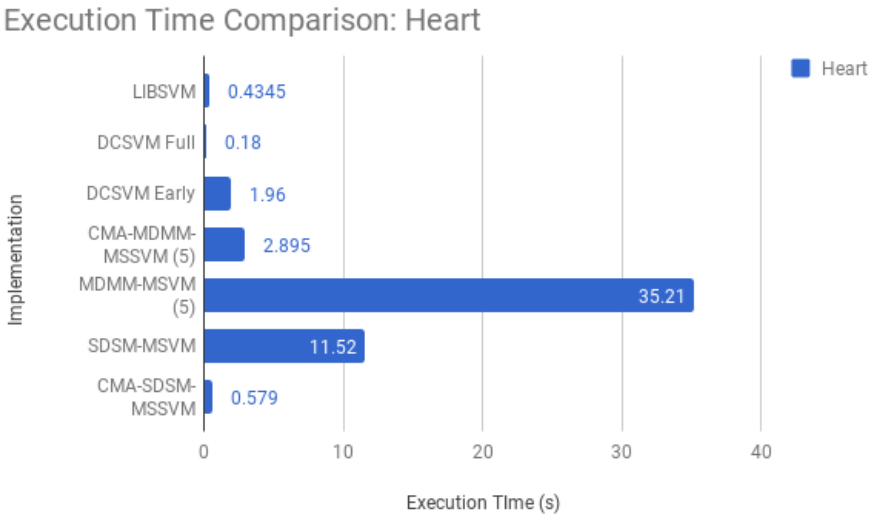

Fig. 5. Execution Time Comparison of Heart Dataset

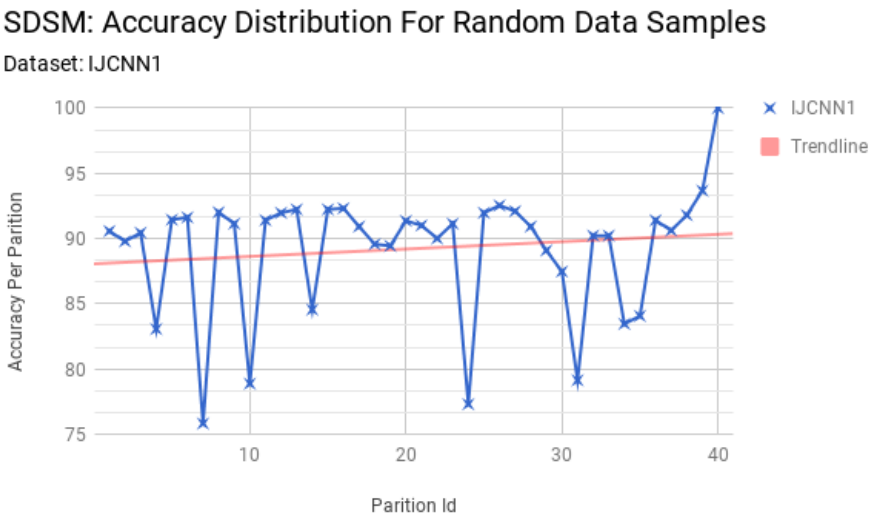

Fig. 6. IJCNN1: SDSM Accuracy Distribution Per Data Partition

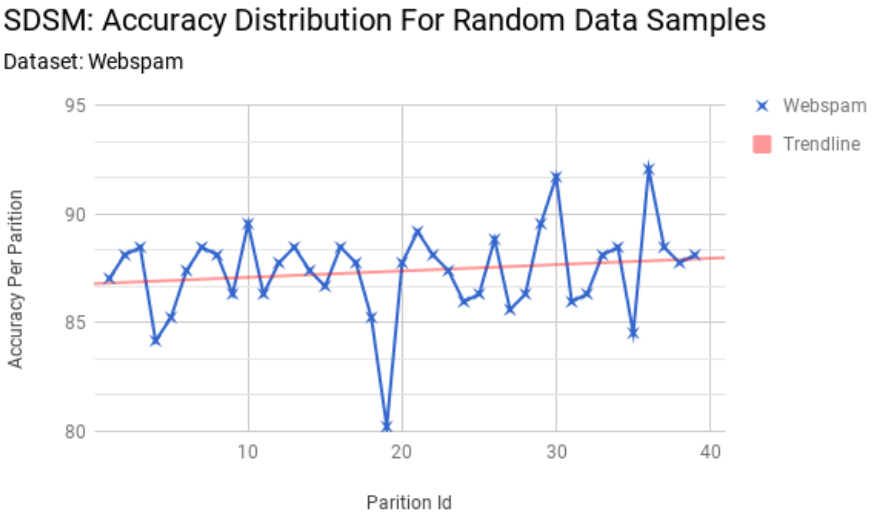

Fig. 7. Webspam: SDSM Accuracy Distribution Per Data Partition

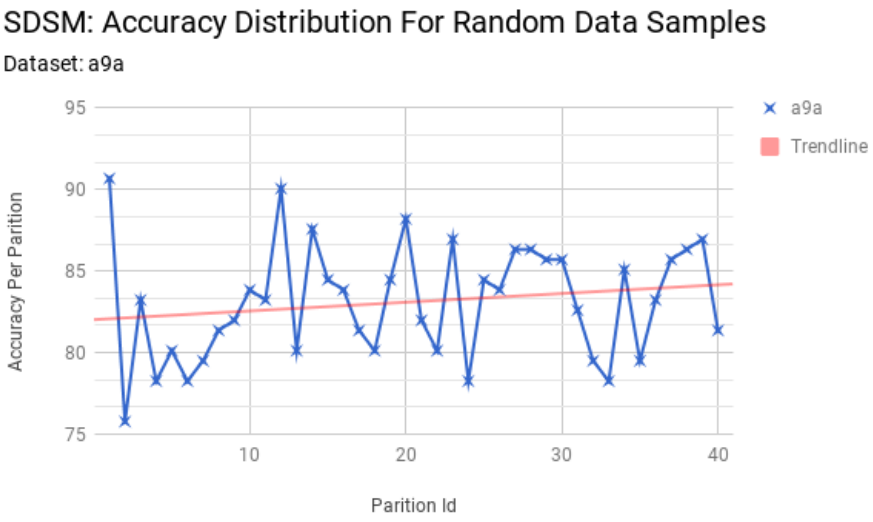

Fig. 8. a9a: SDSM Accuracy Distribution Per Data Partition

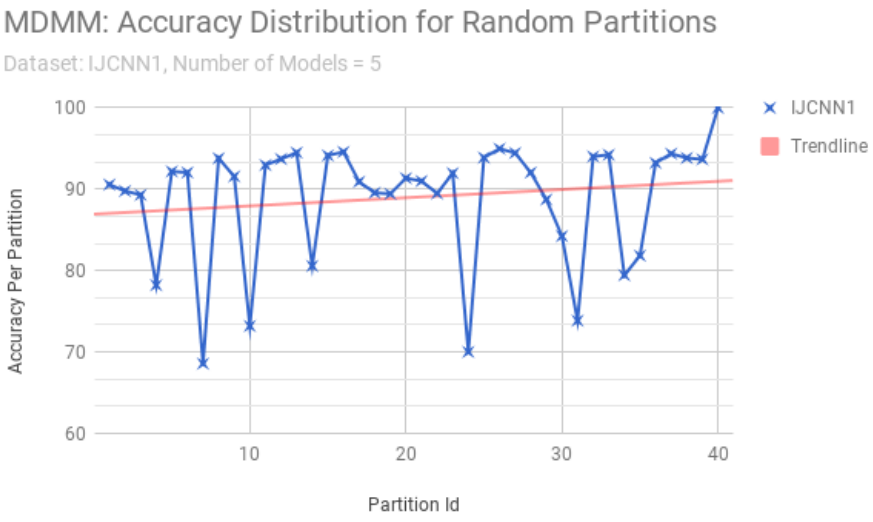

Fig. 9. Ijcnn1: 5 Model-MDMM Accuracy Distribution Per Data Partition

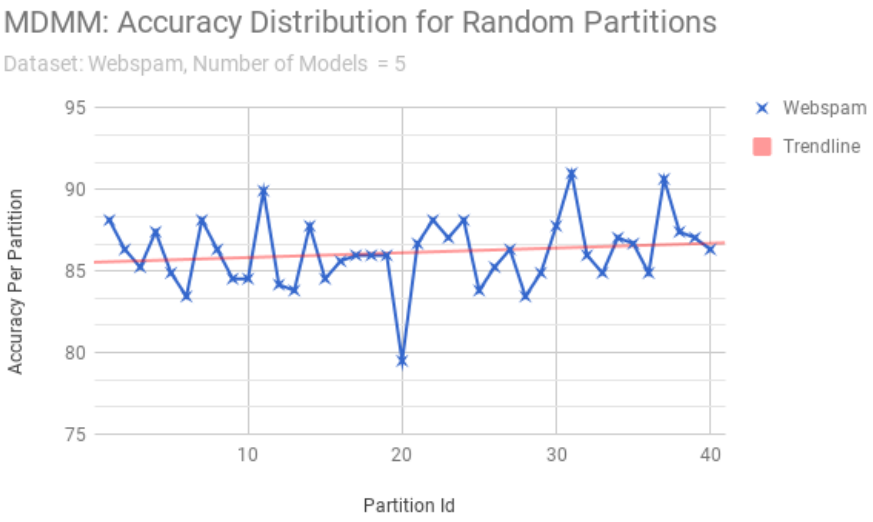

Fig. 10. Webspam: 5 Model-MDMM Accuracy Distribution Per Data Partition

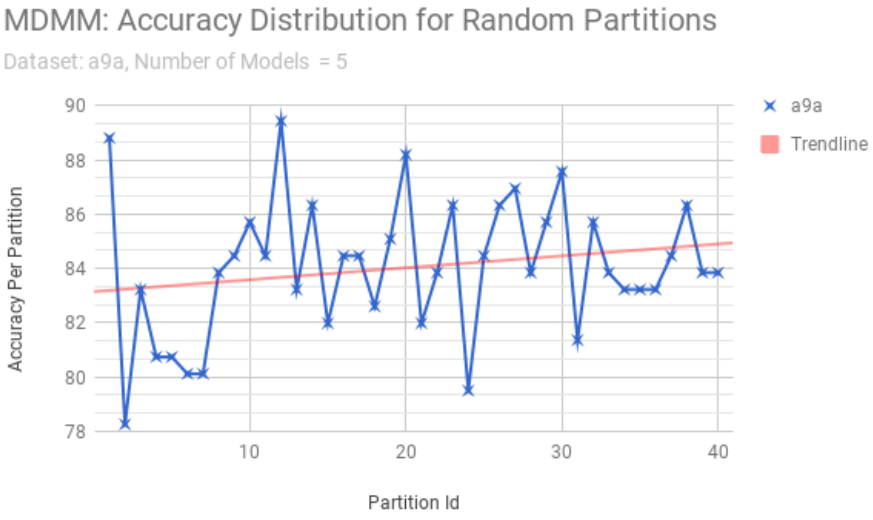

Fig. 11. a9a: 5 Model-MDMM Accuracy Distribution Per Data Partition

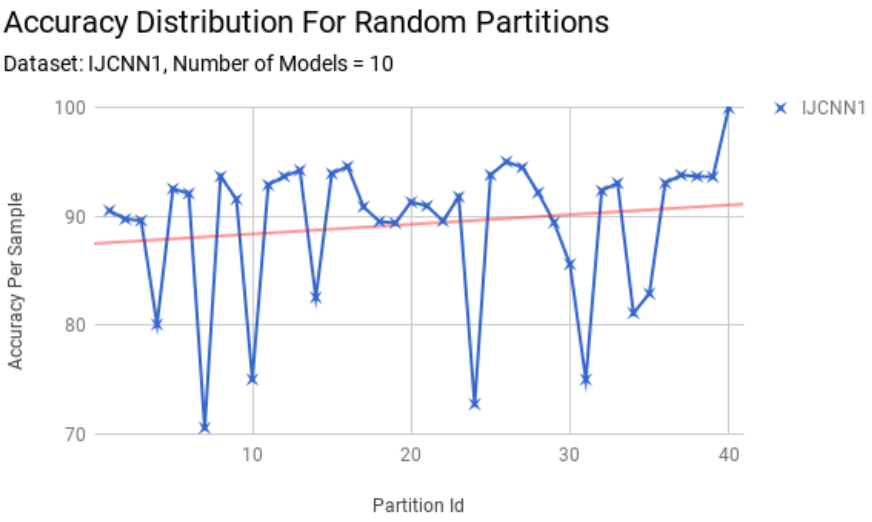

Fig. 12. Ijcnn1: 10 Model-MDMM Accuracy Distribution Per Data Partition

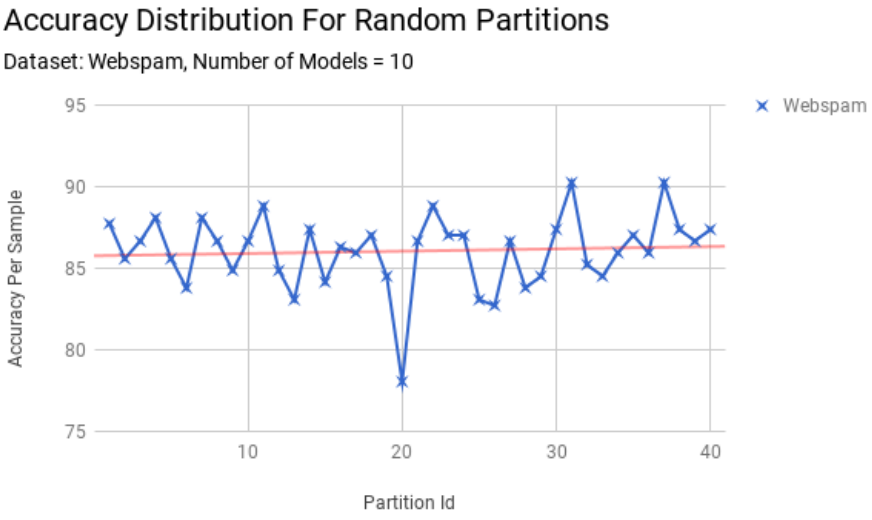

Fig. 13. Webspam: 10 Model-MDMM Accuracy Distribution Per Data Partition

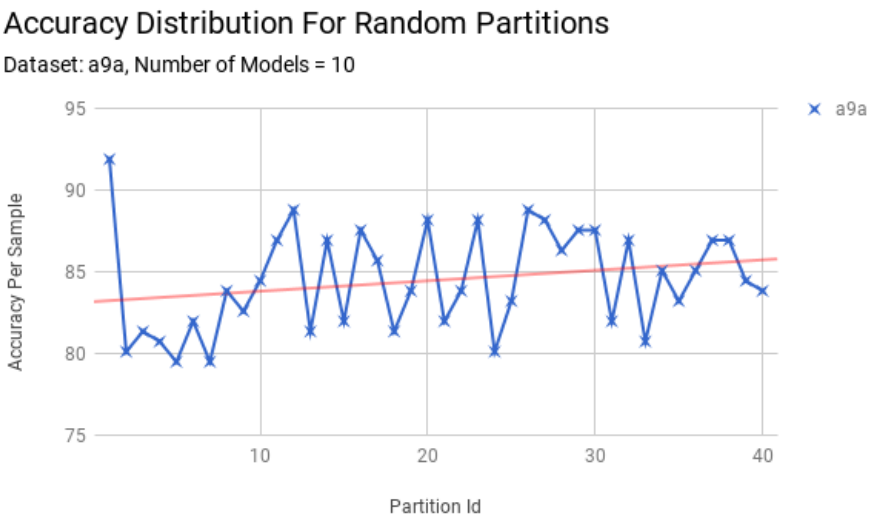

Fig. 14. a9a: 10 Model-MDMM Accuracy Distribution Per Data Partition

6 Conclusion

In this paper, we propose a novel method to train a support vector machines algorithm using a simplified version of sequential minimal optimization algorithm as the core of the training model and we provide a data partitioning method to overcome the training overhead in a larger data sets. The proposed SDSM and MDMM methods proves that a partitioned data set can provide a fair accuracy and much faster convergence after training the svm for a smaller portion of data from the original data set. By comparing the MSVM framework with the state of the art SVM implementations like DC-SVM and LibSVM we have proved that SDSM implementation can be used as a faster training approach to obtain fair accuracy with much faster convergence. The CMA approach provides a faster convergence and a very high accuracy compared to DC-SVM and Libsvm. We kept Libsvm as the benchmark for accuracy and DC-SVM as the benchmark for covergence rate evaluation. DC-SVM can be considered as a faster sequential implementation for SVM to obtain higher accuracy and faster convergence rate. The MDMM approach discussed in this paper provides higher accuracy when the number of models involved in training increases, but in order to increase the performance the programme must be improved with a parallel implementation. With our experiments done larger and moderate data sets, we have proved that our model works very well with larger data sets with a significant improvement in convergence rate of the algorithm with a higher accuracy. Among the three methods proposed in this paper the CMA convergence rate and accuracy is the highest. In our current experiments, we have proved the model for binary classification and even with a linear kernel we can obtain a higher accuracy with MSVM while LibSVM and DC-SVM uses different kernels and different settings to get higher accuracy. With MSVM for binary classification less tweaking needs to be done in tuning. MSVM shows a significant improvement in training SVM with faster convergence and higher accuracy.

7 Future Work

In improving the MDMM approach we are planning to implement a MPI version to increase the performance. And also we are working on improving CMA approach for multi-class classification problems as well. In addition to that we plan to release MSVM framework for research purposes.

8 Acknowledgement

This research was supported by Digital Science Center, Indiana University Bloomington and I would also like to thank my colleagues and my wife for being a part of this research.

References

1. Cho-Jui Hsieh, Si Si, Inderjit S. Dhillon, A Divide-and-Conquer Solver for Kernel Support Vector Machines, Article:CoRR, eprint. 1311.0914,(2013). http://arxiv.org/abs/1311.0914.
2. Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. 2, 3, Article 27 (May 2011), 27 pages. DOI:http://dx.doi.org/10.1145/1961189.1961199
3. R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. Journal of Machine Learning Research, 6:1889–1918, 2005. URL http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf
4. John C. Platt. 1999. Fast training of support vector machines using sequential minimal optimization. In Advances in kernel methods, Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (Eds.). MIT Press, Cambridge, MA, USA 185-208.
5. Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. Mach. Learn. 20, 3 (September 1995), 273-297. DOI: https://doi.org/10.1023/A:1022627411411
6. Edward Y. Chang, Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, and Hang Cui. 2007. PSVM: parallelizing support vector machines on distributed computers. In Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS'07), J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (Eds.). Curran Associates Inc., USA, 257-264.
7. Yang, Jifei Ye, C.-Z Quan, Y Chen, Ni-Yun. (2004). Simplified SMO algorithm for support vector regression. 33. 533-537.
8. Machine Learning Course, The Simplified SMO Algorithm, North Eastern University, College of Computer and Information Science, http://cs229.stanford.edu/materials/smo.pdf
9. Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science
10. Danil Prokhorov. IJCNN 2001 neural network competition. Slide presentation in IJCNN'01, Ford Research Laboratory, 2001. http://www.geocities.com/ijcnn/nnc_ijcnn01.pdf
11. De Wang, Danesh Irani, and Calton Pu. "Evolutionary Study of Web Spam: Webb Spam Corpus 2011 versus Webb Spam Corpus 2006". In Proc. of 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012). Pittsburgh, Pennsylvania, United States, October 2012.
12. Kim HC, Pang S, Je HM, Kim D, Bang SY. Constructing support vector machine ensemble. Pattern recognition. 2003 Dec 1;36(12):2757-67.