2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)
Singapore, 7-9 April 2015

A Distributed Ensemble Scheme for Nonlinear Support Vector Machine

Wei-Chih Lai, Po-Han Huang, Yuh-Jye Lee, Alvin Chiang
National Taiwan University of Science and Technology
{m10115094, m10115095, yuh-jye, m10115088}@mail.ntust.edu.tw

***Abstract*—We propose an ensemble scheme with a parallel computational structure which we call Distributed Ensemble Support Vector Machine (DESVM) to overcome the difficulties of large scale nonlinear Support Vector Machines (SVMs) in practice. The dataset is split into many stratified partitions. Each partition might be still too large to be solved by using conventional SVM solvers. We apply the reduced kernel trick to generate a nonlinear SVM classifier for each partition that can be treated as an approximation model based on the partial dataset. Then, we use a linear SVM classifier to fuse the nonlinear SVM classifiers that are generated from all data partitions. In this linear SVM training model, we treat each nonlinear SVM classifier as an "*attribute*" or an "*expert*". In the ensemble phase, DESVM generates a fusion model which is a weighted combination of the nonlinear SVM classifiers. It can be explained as a weighted voting decision made by a group of experts. We test our proposed method on five benchmark datasets. The numerical results show that DESVM is competitive in accuracy and has a high speed-up. Thus, DESVM can be a powerful tool for binary classification problems with large scale not linearly separable datasets.**

Keywords. Distributed computing, ensemble learning, parallel algorithm, reduced kernel trick, support vector machine

I. Introduction

In the era of Big Data, many extremely large scale datasets that are generated from the real world applications are in front of us. When people use social media and live in sensor-equipped environments, tag data and log data will be recorded automatically with update times on the order of milliseconds. With the large number of data and the hidden opportunities behind these, people have to introduce new algorithms and structures to handle the large scale datasets. Support vector machine (SVM) is one of the most promising and popular learning algorithms for binary classification problems. Although many computational tricks have been proposed to dealing with the computational challenges [1]–[3] we are still in the need of new algorithms to handle the extremely large datasets generated from big data applications. The extremely large datasets here, we mean those datasets are too big to load in a single machine. Inspired by the MapReduce [4] computational framework, we propose the Distributed Ensemble Support Vector Machine for the extremely large dataset that consists of *Distributed* phase and *Ensemble* phase. In the *Distributed* phase, we split the dataset into small pieces so that the data can fit in each computation unit. The reduced support vector machine [1] will be applied here to generate a nonlinear SVM classifier for each small dataset. These nonlinear SVM classifiers will be utilized to define a nonlinear map in the *Ensemble* phase. Please note that we use *Distributed* term instead of *Divided* because in many real world applications the data might come from different sources or business unites. Our proposed framework can be fitted in more applications. In the *Ensemble* phase, we map the entire dataset into a new feature space via the nonlinear map defined by the nonlinear SVM classifiers generated in the *Distributed* phase. We treat each nonlinear SVM classifier as an "attribute" or "feature". Then we apply a linear SVM to the image of the entire dataset in this new feature space to fuse the result of nonlinear classifiers. The linear SVM classifier will give us a weighted sum of each nonlinear SVMs output and determine the threshold for positive instances. The Ensemble phase can be interpreted as a form of *meta learning* [5] or decision fusion. During the Ensemble phase the linear SVM that we call it the fusion model, is trained with all the training data (after their transformation by the locally trained nonlinear SVM models). Therefore the linear SVM, which is tasked with merging the decisions of models trained in the Distributed phase, is trained in such a way as to take the entire dataset into account. The size of the data matrix in the Ensemble phase is $\ell \times N$, where N is the number of nonlinear classifiers that is much smaller than the size of entire dataset ℓ. If we are unable to use the entire dataset in the Ensemble phase the *stratified sampling* scheme can be used here.

We present DESVM as a high-performance nonlinear classification algorithm and implement DESVM using the Message-Passing Interface (MPI) API [6]. MPI is a standard for communication among processes which is used to model a parallel program running on a distributed memory system. With MPI, DESVM can be run on a computing cluster consisting of computers with different operation systems. This flexibility makes it easy for DESVM to scale up for dealing with large scale problems. The experimental results show that DESVM achieves a speed-up of a factor of 10 to 20 over a competing algorithm.

II. Reduced Support Vector Machine

We consider the *supervised learning* classification problems. We are given a training dataset,

$$S = \{(\mathbf{x}_i, y_i) | i = 1, 2, \ldots, \ell\} \subset \mathbb{R}^d \times \mathcal{Y}$$

Each instance $\mathbf{x}_i$ is a point in the d-dimensional real space $\mathbb{R}^d$ and comes with a *class label* $y_i \in \mathcal{Y}$. We would like to

978-1-4799-8055-0/15/$31.00 ©2015 IEEE

construct a decision function $h : \mathbb{R}^d \to \mathcal{Y}$ in an inductive way based on the given dataset S. The image of $\mathbf{x}_i$, $h(\mathbf{x}_i)$ should be equal to or very close to $y_i \in \mathcal{Y}$. Thus, we can predict the class label of a new instance $\mathbf{x}$ that does not appear in the given training dataset S with $h(\mathbf{x})$, the image of $\mathbf{x}$. For the classification problems, the label set $\mathcal{Y}$ is a finite set, for example $\mathcal{Y} = \{1, 2, \ldots, c\}$. In particular, $\mathcal{Y} = \{-1, +1\}$ will be a binary classification problem.

The binary classification algorithm we have chosen to use for the generation of our local experts is the Smooth Support Vector Machine [7]. SSVM solves an unconstrained minimization problem whose formulation is given as follows:

$$\min_{(\mathbf{w},b)\in\mathbb{R}^{d+1}} \frac{1}{2}(\|\mathbf{w}\|_2^2 + b^2) + \frac{C}{2}\sum_{i=1}^{\ell} p(\{1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b\}, \beta)^2 \tag{1}$$

with smooth p-function:

$$p(x, \beta) = x + \frac{1}{\beta}\log(1 + e^{-\beta x}), \tag{2}$$

The objective function in problem (1) is strongly convex and infinitely differentiable.

In many cases, a dataset, as represented in vector form, cannot be linearly separated by a hyperplane. However, it is likely that the dataset becomes linearly separable after being projected to a higher dimensional space by a nonlinear map. A nice property of SVM methodology is that we do not even need to know the nonlinear map explicitly; still, we can apply a linear algorithm to the classification problem in the high dimensional space. The property comes from the dual form of SVM which can express the formulation in terms of inner product of data points. By taking the advantage of dual form, the "kernel trick" is used for the nonlinear extension of SVM. From the dual SVM formulation (3), all we need to know is simply the inner product between training data vectors. Let us map the training data points from the input space $\mathbb{R}^d$ to a higher-dimensional feature space $\mathcal{F}$ by a nonlinear map Φ. The training data x in $\mathcal{F}$ becomes $\Phi(x)$. Based on the above observation, if we know the inner product $\Phi(x_i)^\top \Phi(x_j)$ for all $i, j = 1, 2, \ldots, \ell$, then we can perform the linear SVM algorithm in the feature space $\mathcal{F}$. The separating hyperplane will be linear in the feature space $\mathcal{F}$ but is a nonlinear surface in the input space $\mathbb{R}^d$.

$$\begin{aligned} \min_{\alpha\in\mathbb{R}^\ell} \quad & \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j \\ \text{s.t.} \quad & \sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ & 0 \le \alpha_i \le C \text{ for } i = 1, 2, \ldots, \ell, \end{aligned} \tag{3}$$

Note that we do not need to know the nonlinear map Φ explicitly. It can be achieved by employing a kernel function. Let $K(\mathbf{x}^\top, \mathbf{z}) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be an inner product kernel function satisfying *Mercer's condition* [8]–[11]. We can construct a nonlinear map Φ such that $K(\mathbf{x}_i^\top, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j), i, j = 1, 2, \ldots, \ell$. Hence, the linear SVM formulation can be used on $\Phi(\mathbf{x})$ in the feature space $\mathcal{F}$ by replacing the $\mathbf{x}_i^\top \mathbf{x}_j$ in the objective function of (3) with a nonlinear kernel function $K(\mathbf{x}_i^\top \mathbf{x}_j)$. For the nonlinear case, the formulation of SSVM (1) can be extended to the nonlinear version by utilizing the kernel trick as follows:

$$\min_{(\mathbf{u},b)\in\mathbb{R}^{d+1}} \frac{1}{2}(\|\mathbf{u}\|_2^2 + b^2) + \frac{C}{2}\sum_{i=1}^{\ell} p([1 - y_i\{\sum_{j=1}^{\ell} u_j K(\mathbf{x}_i^\top, \mathbf{x}_j) + b\}], \beta)^2 \tag{4}$$

where $K(\mathbf{x}_i^\top, \mathbf{x}_j)$ is a kernel function. The nonlinear SSVM decision function $h(\mathbf{x})$ can be expressed as follows:

$$h(\mathbf{x}) = \sum_{u_j \neq 0} u_j K(\mathbf{x}_i^\top, \mathbf{x}_j) + b. \tag{5}$$

In a large scale SVM, the full kernel matrix will be very large and dense, so it may not be appropriate to use the full kernel matrix when dealing with (1). To avoid facing such a large and dense full kernel matrix, we apply the reduced kernel technique [12]. The key idea of the reduced kernel technique is to randomly select a small portion of data and to generate a thin rectangular kernel matrix, then to use this much smaller rectangular kernel matrix to replace the full kernel matrix. The formulation of reduced SSVM is expressed as follows:

$$\min_{(\tilde{\mathbf{u}},b)} \frac{1}{2}(\|\tilde{\mathbf{u}}\|_2^2 + b^2) + \frac{C}{2}\sum_{i=1}^{\tilde{\ell}} p([1 - y_i\{\sum_{j=1}^{\tilde{\ell}} \tilde{u}_j K(\mathbf{x}_i^\top, \mathbf{x}_j) + b\}], \beta)^2 \tag{6}$$

where $\tilde{u} \in \mathbb{R}^{\tilde{\ell}}$ with $\tilde{\ell} \ll \ell$, and its decision function is in the form

$$h(\mathbf{x}) = \sum_{j=1}^{\tilde{\ell}} \tilde{u}_j K(\mathbf{x}_i^\top, \mathbf{x}_j) + b. \tag{7}$$

III. Ensemble Learning

In this section, we will review ensemble learning and compare DESVM with traditional ensemble methods such as Adaboost [13] and stacking [14]. The structure of DESVM has similarities with Adaboost and stacking. But since we are actually inspired by MapReduce and stacking, we wish to point out the differences between our method and these two traditional ensemble methods.

Ensemble learning is a method that combines multiple classifiers to obtain better predictive performance rather than using a single classifier [15]. Ensemble learning is a reliable method which uses multiple weak learners to generate a single strong learner [16]. In this theory, even if each weak learner behaves a poor performance, ensemble learning can still generate a useful model from the ensemble of weak learners.

A. Adaboost

Adaboost is an ensemble learning algorithm that adaptively adjusts for the errors returned from the weak learners. For a training dataset, Adaboost use a series of weighted weak learners to generate a classification model. In the training procedure, Adaboost will update the weighting of weak learners according to training errors in each iteration. After many iterations, Adaboost gives the ensemble of all weak learner models as the final classification model.

There are two major differences between our method and Adaboost. First, Adaboost uses training errors from weak learners to adjust ensemble model. Our method considers each model in Distribution phase as a local expert for providing estimation as a new training attribute. Second, each weak learner in Adaboost uses all training data and adjusts in each iteration. In our method, each model in Distribution phase only uses data subset to generate model. In Ensemble phase, each local expert uses sampled training data to make new feature for generating fusion model.

B. Stacking

Stacked generalization, or stacking, generally involves training a learning algorithm to combine the predictions of several other learning algorithms. First, all of the other algorithms are trained using the same training data, then a fusion algorithm is trained to make a final prediction using all the predictions of the other algorithms as inputs.

Our proposed method is very similar to stacking although we use the estimate value rather than the decision function as the the final model inputs. Also, in DESVM we only use SVM models. Because the estimation of SVM is the distance between data and hyperplane, we can reasonably see the estimation as new feature.

IV. Distributed Ensemble Support Vector Machine

In order to apply nonlinear method upon super large scale problem, we provide an algorithm named Distributed Ensemble Support Vector Machine (DESVM). We partition the whole problem into smaller ones instead of rewriting the SVM formula into a parallel algorithm. Each smaller set will be trained by individual nonlinear SVM. After training nonlinear SVMs, we introduce an ensemble structure which treat every nonlinear SVM as a local expert. Those local experts will combine their estimations to solve the whole problem. Fig. 1 shows the whole structure of our algorithm. In this paper, we focus on binary classification to simplify our problem.

A. Training

Distribution phase is the first step of training. In this phase, a super large scale dataset will be split into many small subset pieces. Each subset piece will be trained by an RSVM independently to generate its own model. The next step is Ensemble phase. We use the term $estimate(A, F)$ as our feature generate function which we will define later to combine the Distribution phase's results.

The whole structure is to make nonlinear SVM possible to train a super large scale problem. In the meantime, every small nonlinear SVM is independent from each other so that it can build nonlinear SVM model on many different machines. We use MPI technique to implement out algorithm. Moreover, we can also have speed advantage because DESVM generates nonlinear models by smaller kernels rather than a single gigantic full kernel.

Algorithm 1 Distributed Ensemble SVM

Input: Large scale dataset A, split number N, sample size ℓ.
Output: RSVM models $\{F_j\}_{j=1}^{N}$, the $sample$ instances form origin dataset, the linear SSVM model $\{F_{fusion}\}$ with new features generated by $\{F_j\}_{j=1}^{N}$

1) Split A into N subsets $\{A_j\}_{j=1}^{N}$ and choose sample instances, $sample$, with size ℓ
2) Learn the RSVM model F_j from each subset A_j
3) Generate a new representation $B \in \mathbb{R}^{\ell \times N}$
   The jth column of B is represented as follow:
   $B_j = estimate(sample, F_j)$
4) Learn the linear SSVM model with B:
   $F_{fusion} \leftarrow \text{SSVM}_{train}^{linear}(B)$
5) Return $\{F_j\}_{j=1}^{N}$, $sample$, and F_{fusion}

1) Distributed phase: Build local expert for data subset: Since we have limited computing resources in the real world, we are confronted with the task of building a $O(\ell^2)$ kernel for a ℓ instances data when we are trying to train a nonlinear SVM model. If ℓ grows large, it will not be possible to use a single machine to train the whole data.

In response, we partition the whole problem into N subsets. Making our complexity become $O(N(\ell/N)^2)$. Letting our machine able to afford to solve the problem. Furthermore, we use RSVM as our local expert to have our complexity $O(N(r\ell/N)^2)$ which r is the ratio in RSVM to have greater advantage on it. We also let the data split in similar distribution as in the original whole dataset to remain the balance of subsets.

Each trained RSVM model has a decision function of the form (7) reproduced here:

$$h(\mathbf{x}) = \sum_{j=1}^{\tilde{\ell}} \tilde{u}_j K(\mathbf{x}_i^\top, \mathbf{x}_j) + b.$$

These estimates will be used as the input features in the ensemble phase.

2) Ensemble phase – Learning from local experts: In this phase, we construct a fusion table to combine the features that were generated by the models trained in the Distributed phase. We use the estimations, produced by the RSVM models for each data instance, as the inputs of our final linear SVM model. In order to increase performance, we choose to use the estimations which are randomly selected portion of the original dataset rather than the entire dataset. We give each instance an estimation from each local expert. Recall the decision function

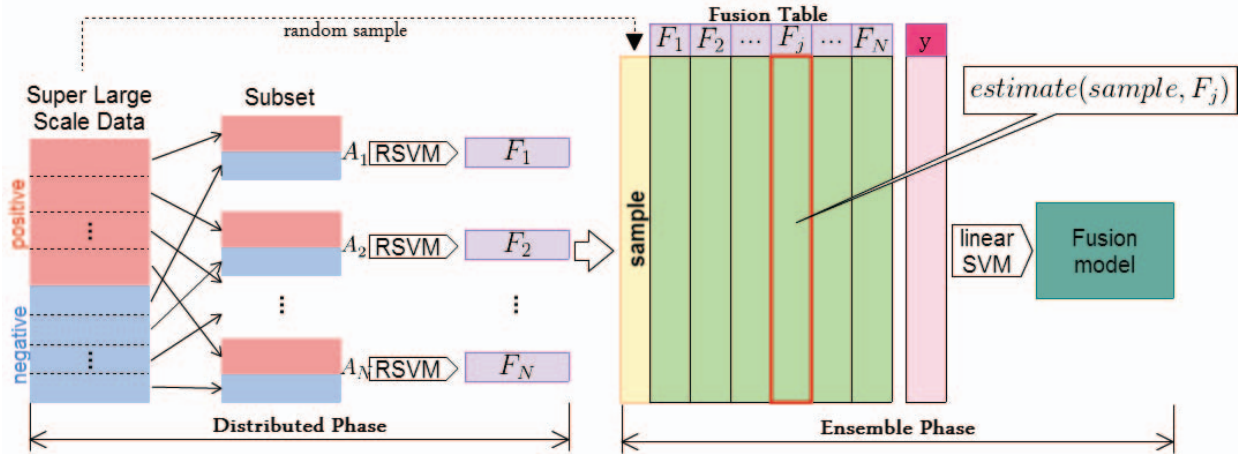

Fig. 1: Distributed Ensemble Support Vector Machine structure

of the RSVM models (7). Here we rewrite it to show its role in the DESVM procedure:

$$estimate(A, F) = w_F \times K(A, ReduceSet_F) + b_F \quad (8)$$

A contains the instances we want to have estimation values from the expert F. w_F, b_F represents the weights and bias of the RSVM model and K is the kernel function. In this case, the input instances will build a kernel with the reduced set we used to generate our RSVM model. The reduced set can also be regarded as the local expert's representation of its training set.

Since we partition the data into N pieces, we will have N local experts that can generate N features for one data instance. In other words, DESVM recasts the problem into a new N dimension space. Instances in the new feature space are linear separable as shown in Fig. 2. Using a linear SVM to create the fusion model is the last step. This fusion model will classify the input instances on the new feature space.

(a) ijcnn1 using PCA (b) ijcnn1 in Ensemble phase

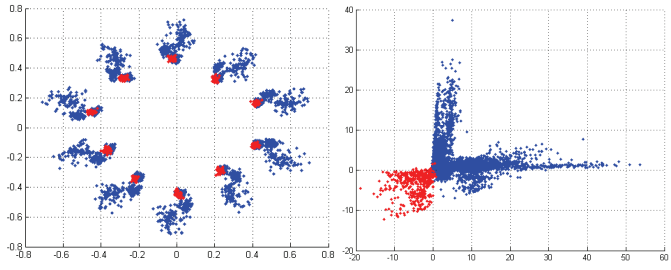

Fig. 2: PCA vs. Ensemble phase

B. Predicting

Making predictions on new data, we go through the same steps as training. If there are too many test data, we partition the dataset into smaller subsets. Then, we use the local experts to produce new features for our test instances. Finally, we can get the predicted label from the fusion model in the new N-dimension space.

Algorithm 2 Distributed Ensemble SVM predict

Input: Testing instance $\mathbf{x}$, $\{F_j\}_{j=1}^{N}$, and F_{fusion}
Output: Predicted label

1) Split $\mathbf{x}$ into N subsets: $\{\mathbf{x}_i\}_{i=1}^{N}$
2) Generate the new representation $B \in \mathbb{R}^{L \times N}$ with L equals to the size of $\mathbf{x}$: $B_{ij} = estimate(\mathbf{x}_i, F_j)$
3) Predicted label $\leftarrow$ sign$(F_{fusion}(B))$

C. Implementation details

We implement DESVM by using MPI on C++. The C++ language is used for our implementations of RSVM as local expert and SSVM as the ensemble model. This implementation has high performance and is able to handle large scale data in sparse format. MPI is used to transport meta information and serialized representations of the local experts.

1) Training: In the Distributed phase, DESVM trains local experts by assigning training tasks for many individual idle computing units until all training tasks are completed. The number of training tasks is equal to the split number. If a training task is finished, the local expert will be send to a host computing unit and stored. Since we implement DESVM by MPI and all training tasks are independent, the training procedure of DESVM can be processed in parallel. Moreover, we use MPI to find idle processors and minimize the task waiting time. For a cluster comprising computing units having unequal computing ability, this is an useful characteristic.

In the Ensemble phase, the host computing unit loads all local experts and sends them to all other idle computing units. After sending complete, each local expert will give a estimation value for each sampled training data. As in

the Distributed phase, DESVM assigns estimation tasks to different computing units so that we can generate estimation values in parallel. Those estimation values will be combined into a fusion table used as the training data for the ensemble model. Creating the fusion table is usually the most time-consuming step of the entire DESVM procedure.

2) Testing: The testing procedure is just like the Ensemble phase. But we use testing data as input instead of sampled training data. We implement loading testing data in batch to prevent a lack of memory.

TABLE I: Computer Manufacture

CPU	Core Number	RAM(GB)
Intel E3-1230 V2 @ 3.30GHz	8	16
Intel i3-540 @ 3.07GHz	4	4
Intel i3-2100 @ 3.10GHz	4	8
AMD A8-3850 @ 2.90GHz	4	8

V. Experiments

In this section, we will show this method retains the classification ability of nonlinear SVM. Also, we are interested in the performance of each RSVM models and the difference between those RSVM models and fusion model. In all our experiments we use the radial basis function (RBF) kernel:

$$K(\mathbf{x}_i^\top, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2}$$

All experiments were running on the computers in Table. I.

A. DCSVM

We compare our method with another state-of-the-art large scale nonlinear SVM algorithm, Divide-and-Conquer Support Vector Machine [17]. DCSVM divides the big problem into smaller ones. After computing all the subproblem solutions, it concatenates them as an approximate solution for the whole problem. In the division step, it solves smaller problems instead of the big one to reduce computing time. In the conquer step, it uses appropriate solution to initialize the solver for the whole problem for speeding up training process. Moreover, the authors propose a multilevel stacking approach which further increases the speed up. It also provides early version for early prediction strategy. The early DCSVM simply use the sum of the subproblem solutions rather than solving the whole problem. With those advantage, DCSVM outperforms state-of-the-art methods in terms of training speed and is our main competitor.

B. Preprocessing

In the first step of preprocessing, we normalize each attribute column between $[0, 1]$. Second, we split the dataset into N subsets for training the local experts. We ensure that each subset retains the same proportion of positive and negative labels. Also, we sample instances from each sub-dataset for creating the fusion table used to train the final combiner classification model. In each sampled dataset, the number of positive instance is equal to the number of negative instance.

TABLE II: Dataset information

Dataset Name	Number of Training	Dimension	Number of Testing	Split Number
ijcnn1	49,990	22	91,701	20
usps01	266,079	676	75,383	8
webspam	280,000	254	70,000	40
kddcup99	4,898,431	122	311,029	200
mnist8m	8,000,000	784	100,000	200

C. Experiment Setting

While training RSVM model in distributed phase, we need to choose the value of cost C, gamma γ (for the RBF kernel) and the reduce ratio r (used in selection the reduced kernel). And for the ensemble phase, we need to decide the value of cost C for SSVM.

D. Experiment Results

We use different sizes of datasets to test the ability of DESVM. The data information are in Table II.

1) RSVM models vs. Fusion model: In the Ensemble phase, we use ensemble method to fuse local experts and generate the fusion model with better predictive ability. We apply our method on ijcnn1 to justify our theory. The result is shown in Fig. 3. As we expected, even each local expert has poor predictive ability. Our fusion model can still has amazing performance by ensemble learning.

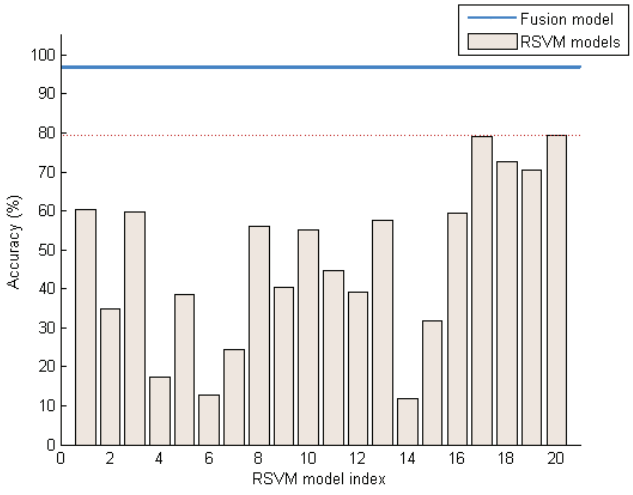

Fig. 3: RSVM models vs. Fusion model

2) Computation time: In this part, we only compare the training times with DCSVM and early DCSVM on five datasets. Those datasets are in different scale e.g. ijcnn1 has 10,000 of training instance number; upsp01 and webspam have 100,000; kddcup99 and mnist8m have 1,000,000. In Fig. 4 shows that our method is faster than the state-of-the-art method DCSVM by a factor of 10 to 20.

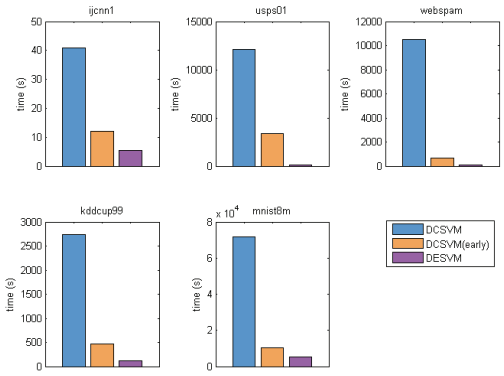

Fig. 4: Computation time comparison: DCSVM vs. DESVM

E. Accuracy

In this section, we would like to show that our proposed method stays competitive with DCSVM. We compare the predictive performance of DCSVM, early DCSVM and DESVM on five different datasets. Fig. 5 shows that accuracy of these methods have almost no difference.

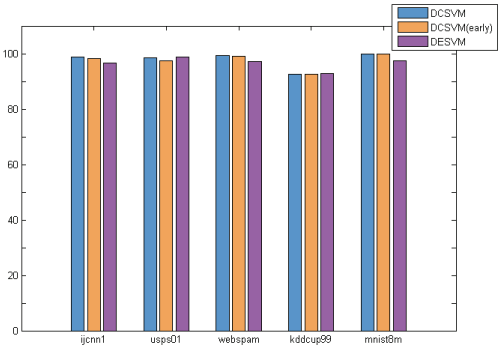

Fig. 5: Accuracy of DCSVM and DESVM

F. Training Time vs. number of cores

To demonstrate DESVM can speed up the training procedure of nonlinear SVM by distributed structure. We test DESVM on a large dataset, kddcup99, with different number of processes on different machines. Experiments show that we can have linear speed-up with the number of cores.

VI. Conclusion

In this paper, we have described a parallel algorithm which we call Distributed Ensemble Support Vector Machine (DESVM). It's purpose is to speed up the application of nonlinear SVM methods on super large scale data through the use of a parallel training and prediction structure. In the Distributed phase, it splits super large data into small pieces and trains nonlinear SVM models from each piece. In the Ensemble phase, it combines all models into a linear SVM as a ensemble model. Since each training data in Distributed phase is much smaller, the training time is radically decreased. Moreover, training for each model can be done in parallel.

We now have the ability to process super large scale datasets. Compared to normal batch learning algorithms, DESVM has the advantage of smaller computing and memory requirements.

The experimental results show that we have successfully extended Reduced Support Vector Machine from handling large data to having the ability to process super large scale data. For a dataset which is extremely large and needs to be trained by a nonlinear SVM, our method can do it quickly and in parallel both in training and prediction.

Acknowledgement

This work was supported by Ministry of Science and Technology, National Taiwan University and Intel Corporation under Grants MOST 103-2911-I-002-001, 102-2923-E-011-001-MY2, 103-2221-E-011-109-MY2 and NTU-ICRP-104R7501

References

[1] Y.-J. Lee and O. L. Mangasarian, "Rsvm: Reduced support vector machines." in *SDM*, vol. 1. SIAM, 2001, pp. 325–361.

[2] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *Signal Processing, IEEE Transactions on*, vol. 52, no. 8, pp. 2165–2176, 2004.

[3] H. Ouyang and A. G. Gray, "Fast stochastic frank-wolfe algorithms for nonlinear svms." in *SDM*. SIAM, 2010, pp. 245–256.

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

[5] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.

[6] M. P. Forum, "Mpi: A message-passing interface standard," Knoxville, TN, USA, Tech. Rep., 1994.

[7] Y.-J. Lee and O. L. Mangasarian, "Ssvm: A smooth support vector machine for classification," *Computational optimization and Applications*, vol. 20, no. 1, pp. 5–22, 2001.

[8] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[9] V. Cherkassky and F. M. Mulier, *Learning from data: concepts, theory, and methods*. John Wiley & Sons, 2007.

[10] R. Courant and D. Hilbert, *Methods of mathematical physics*. CUP Archive, 1966, vol. 1.

[11] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[12] Y.-J. Lee and S.-Y. Huang, "Reduced support vector machines: A statistical theory," *Neural Networks, IEEE Transactions on*, vol. 18, no. 1, pp. 1–13, 2007.

[13] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational learning theory*. Springer, 1995, pp. 23–37.

[14] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[15] R. Maclin and D. Opitz, "Popular ensemble methods: An empirical study," *arXiv preprint arXiv:1106.0257*, 2011.

[16] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine learning*, vol. 51, no. 2, pp. 181–207, 2003.

[17] C.-J. Hsieh, S. Si, and I. S. Dhillon, "A divide-and-conquer solver for kernel support vector machines," *arXiv preprint arXiv:1311.0914*, 2013.