A Distributed SVM Method based on the Iterative MapReduce

Xijiang Ke, Hai Jin*, Xia Xie, Jie Cao
Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
Email: hjin@hust.edu.cn

***Abstract*—Linear classification is useful in many applications, but training large-scale data remains an important research issue. Recent advances in linear classification have shown that distributed methods can be efficient in improving the training time. However, for most of the existing training methods,based on MPI or Hadoop, the communication between nodes is the bottleneck. To shorten the communication between nodes, we propose and analyze a method for distributed support vector machine and implement it on an iterative MapReduce framework. Through our distributed method, the local SVMs are generic and can make use of the state-of-the-art SVM solvers. Unlike previous attempts to parallelize SVMs the algorithm does not make assumptions on the density of the support vectors, i.e., the efficiency of the algorithm holds also for the "difficult" cases where the number of support vectors is very high. The performance of the our method is evaluated in an experimental environment. By partitioning the training dataset into smaller subsets and optimizing the partitioned subsets across a cluster of computers, we reduce the training time significantly while maintaining a high level of accuracy in both binary and multiclass classifications.**

I. Introduction

With the development of computer technology, the quantity of data is in exponential growth. Data deluge has become a salient problem to be solved. Scientists are overwhelmed with the increasing amount of data processing needs arising from the storm of data that is flowing through every science field, such as bioinformatics, biomedical, web. How to take full use of the large scale data to support decision is a big problem encountered by scientists [1].

In the past few years, the size of data sets used to build classifiers has greatly increased. Linear classification is useful in many applications, but training large-scale data remains an important research issue [2]. Existing approaches to handle large data classification can be categorized to two types [3]. The first approach considers online learning algorithms [4][5][6][7]. Since data may be used only once, this type of approaches can handle the memory issue effectively. However, even with online setting, an implementation over a distributed environment is still complicated. The second approach solves problems in distributed systems by parallelizing batch training algorithms [8][9]. However, not only writing programs on a distributed system is difficult, but also the data communication/synchronization may cause significant overheads [10].

II. Parallel Decomposition Method for SVM

Given a training set of instance-label pairs $(x_i, y_i), i = 1, ..., l$, where $x_i \in R^n$ and $y \in \{1, -1\}$, the linear SVM problem [11] seeks the minimizer of:

$$\min_{\omega} \frac{C}{2}||\omega||_2^2 + \frac{1}{m}\sum_{i=1}^{m} loss(\omega; (x_i, y_i)) \tag{1}$$

where C tradeoffs the weight given to the margin between the hyperplane ω to the closest examples (the support vectors) to it and the weight given to margin errors (example points which do not lie respectively farther from the support vectors), and

$$loss(\omega; (x_i, y_i)) = \max(0, 1 - y_i \omega^T x_i) \tag{2}$$

Usually, the parallel SVM is based on the cascade SVM model [12]. In the architecture, the sets of support vectors of two SVMs are combined into one set and to be input a new SVM. The process continues until only one set of vectors is left. In this architecture a single SVM never has to deal with the whole training set. If the filters in the first few layers are efficient in extracting the support vectors then the largest optimization, the one of the last layer, has to handle only a few more vectors than the number of actual support vectors. Therefore, the training sets of each sub-problems are much smaller than that of the whole problem when the support vectors are a small subset of the training vectors. However, it lacks of theoretical proof and experiments show that it does work very well for large scale documents classification. We need a more practical method to solve these subSVMs.

Let $S_1, .., S_k$ be subsets (not necessarily disjoint) of the training set indices: $S_j \subset \{1, ..., m\}$ such that $S_j \neq \emptyset$ and $\bigcup_j S_j = \{1, ..., m\}$ where k is a fixed number representing the number of available processors.

Now, we derive a parallel block update algorithm for the SVM problem Eq.1, which is equivalent to:

$$\min_{\omega_1 = ... = \omega_k} \frac{C}{2k}\sum_{j=1}^{k} ||\omega_j||_2^2 + \frac{1}{m}\sum_{i=1}^{k}\sum_{i \in S_j} loss(\omega; (x_i, y_i)) \tag{3}$$

where $\omega_1 = ... = \omega_k$ live in R^n as well. Let $\mathcal{H}$ be the set defined by:

$$\mathcal{H} = \bar{\omega} = (\omega_1, ..., \omega_k) \in R^n k : \omega_1 = ... = \omega_k \tag{4}$$

Let $\delta_{\mathcal{H}}(\bar{\omega})$ be the indicator function $\delta_{\mathcal{H}}(\bar{\omega}) = 0$ if $\bar{\omega} \in \mathcal{H}$ and $\delta_{\mathcal{H}}(\bar{\omega}) = \infty$ if $\bar{\omega} \notin \mathcal{H}$, and let $h_j(\omega_j) = \sum_{i \in S_j} loss(\omega_j; (x_i, y_i))$. Eq.3 is equivalent to:

$$\bar{\omega} = \min_{(\omega_1,...,\omega_k)} (\frac{1}{m}\sum_{j=1}^{k} h_j(\omega_j) + \delta_{\mathcal{H}}(\bar{\omega})) \ - (-\frac{C}{2k}\sum_{j=1}^{k} ||\omega_j||_2^2) \tag{5}$$

which is of the form $\min_{\bar{\omega}} f_1(Q\bar{\omega}) - f_2(\bar{\omega})$, where $Q = [I; I]$, i.e., $Q\bar{\omega} = (\bar{\omega}, \bar{\omega})$. From Fenchel Duality [13], we have:

$$\min_{\bar{\omega}} f_1(Q\bar{\omega}) - f_2(\bar{\omega}) = \max_{\bar{\lambda},\bar{\mu}} g_2(\bar{\lambda} + \bar{\mu}) - g_1(\bar{\lambda}, \bar{\mu}) \tag{6}$$

where g_1, g_2 are convex conjugates of f_1, f_2, respectively, and $\bar{\lambda} = (\lambda_1, ..., \lambda_k)$ and $\bar{\mu} = (\mu_1, ..., \mu_k)$. Specifically,

$$\begin{aligned} &\max_{\bar{\lambda},\bar{\mu}} g_2(\bar{\lambda} + \bar{\mu}) - g_1(\bar{\lambda}, \bar{\mu}) = \\ &\max_{\bar{\lambda},\bar{\mu}} -\frac{k}{2C}\sum_{j=1}^{k} ||\lambda_j + \mu_j||_2^2 - \sum_{j=1}^{k} h_j^*(\lambda_j) - \sigma_{\mathcal{H}}(\bar{\mu}) \end{aligned} \tag{7}$$

where $\sigma_{\mathcal{H}}(\bar{\mu})$ is the support function of $\mathcal{H}$ (convex conjugate of $\delta_{\mathcal{H}}(\bar{\mu})$). We employ a Jacobi-style block-update approach, with a time counter (t), for maximizing the dual vectors $\lambda_j, \mu_j, j = 1, ..., k$, where at each step we maximize $g_2(\bar{\lambda}+\bar{\mu}) - g_1(\bar{\lambda}, \bar{\mu})$ with respect to λ_j while keeping the other dual vectors fixed. The value of the dual vector $\lambda_j^{(t)}$ becomes:

$$\lambda_j^{(t)} = \arg\min_{\lambda} -\frac{k}{2C}||\lambda + \mu_j^{(t-1)}||_2^2 - h_j^*(\lambda) \tag{8}$$

which is of the form (employing FD again) $\max_\lambda g_2(\lambda) - g_1(\lambda)$, which is equal to $\min_\omega f_1(\omega) - f_2(\omega)$, where

$$\begin{aligned} f_2(\omega) &= -\frac{C}{2k}||\omega||_2^2 - \omega^T \mu_j^{(t-1)} \\ f_1(\omega) &= h_j(\omega) \end{aligned} \tag{9}$$

from which we obtain the solution for ω for the j^{th} subproblem:

$$\omega_J^{(t)} = \arg\min_{\omega} \frac{C}{2k}||\omega||_2^2 + \omega^T \mu_j^{(t-1)} + \frac{1}{m}\sum_{i \in S_j} loss(\omega; (x_i, y_i)) \tag{10}$$

The connection between $\lambda_j^{(t)}$ and $\omega_j^{(t)}$ follows Lagrangian optimality:

$$\omega_j^{(t)} = \arg\min_{\omega} \omega^T \lambda_j^{(t)} + \frac{C}{2k}||\omega||_2^2 + \omega^T \mu_j^{(t-1)} \tag{11}$$

from which we obtain the update equation for λ_j:

$$\lambda_j^{(t)} = -\mu_j^{(t-1)} - \frac{C}{k} P_j(\mu_i^{(t-1)}) \tag{12}$$

Next we consider the block update for $\bar{\mu}$. The value of the dual vector $\bar{\mu}^{(t)}$ is equal to:

$$\bar{\mu}^{(t)} = \arg\max_{\mu} -\frac{k}{2C}\sum_{j=1}^{k} ||\lambda_j^{(t)} + \mu_j||_2^2 - \sigma_{\mathcal{H}}(\bar{\mu}) \tag{13}$$

which is of the form $\max g_2(\bar{\mu}) - g_1(\bar{\mu})$, which from Fenchel Deuality is equal to $\min \delta_{\mathcal{H}}(\bar{\omega}) - f_2(\bar{\omega})$, where

$$f_2(\bar{\omega}) = -\frac{C}{2k}\sum_{j=1}^{k} ||\omega_j||_2^2 - \sum_{j=1}^{k} \omega_j^T \lambda_j^{(t)} \tag{14}$$

The solution for $\bar{\omega} = (\omega_1, ..., \omega_k)$ is:

$$\bar{\omega}^{(t)} = \arg\min_{\omega_1 = ... = \omega_k} \frac{C}{2k}\sum_{j=1}^{k} ||\omega_j||_2^2 + \sum_{j=1}^{k} \omega_j^T \lambda_j^{(t)} \tag{15}$$

from which we obtain:

$$\omega^{(t)} = \arg\min_{\omega} \frac{C}{k}||\omega||_2^2 + \omega^T(\sum_{j=1}^{k} \lambda_j^{(t)}) \tag{16}$$

from which it follows:

$$C\omega^{(t)} + \sum_{j=1}^{k} \lambda_j^{(t)} = 0 \tag{17}$$

The connection between $\bar{\mu}^{(t)} = (\mu_1^{(t)}, ..., \mu_k^{(t)})$ and $\bar{\omega}^{(t)} = (\omega_1^{(t)}, ..., \omega_k^{(t)})$ follows Lagrange optimality:

$$\bar{\omega}^{(t)} = \arg\min_{\bar{\omega}} \bar{\omega}^T \bar{\mu}^{(t)} + \frac{C}{2k}\sum_{j=1}^{k} ||\omega_j||_2^2 + \sum_{j=1}^{k} \omega_j^T \lambda_j^{(t)}, \tag{18}$$

from which it follows:

$$\mu_j^{(t)} = -\lambda_j^{(t)} - \frac{C}{k}\omega^{(t)} \tag{19}$$

Substituting $\omega^{(t)}$ from Eq.17 we obtain the update formula:

$$\mu_j^{(t)} \leftarrow -\lambda_j^{(t)} + \frac{1}{k}\sum_{l=1}^{k} \lambda_l^{(t)} \tag{20}$$

From Eq.10, we have the solution to the j^{th} update for subSVM. So, we define $P_j(\mathbf{d})$ be the solution, that is:

$$P_j(\mathbf{d}) = \arg\min_{\omega} \frac{C}{2k}||\omega||_2^2 + \omega^T \mathbf{d} + \frac{1}{m}\sum_{i \in S_j} loss(\omega; (x_i, y_i)) \tag{21}$$

where $\mathbf{d} = \mu_j^{(t-1)}$ for which $\mathbf{d} = 0$ is the SVM separating hyperplane ω_j of the training data represented by S_j. So we get our parallel algorithm for SVM.

Note that λ_j holds a separating hyperplane calculated by the training examples of S_j and a weighted sum of all previous separating hyperplanes. When $t = 1$, $\lambda_j^{(1)}$ equal to the SVM solution over S_j, i.e., the classifier ω_j of the training set S_j. Then $\mu_j^{(1)}$contains a weighted sum of all those classifiers. The result of the weighted sum is passed onto the next iteration as the vector $\mathbf{d}$ in the operator $P_j(\mathbf{d})$.

We implement the PDS (Algorithm 1) on the iterative MapReduce framework to achieve better training speed. From the parallel SVM architecture, the pseudo program code based on Haloop [14] is shown in Workflow.1.

Algorithm 1 Parallel Decomposition Solver for SVM(PDS).

```
Input:
1: Set $\lambda_j^{(0)} = 0$ and $\mu_j^{(0)} = 0$.
Output:
2: For $t = 1, 2, ..., T$
3: parallel update $j \in \{i, ..., k\}$:
       $$\lambda_j^{(t)} \leftarrow -\mu_j^{(t-1)} - \frac{C}{k} P_j(\mu_j^{(t-1)})$$
4: message passing $j = 1, ..., k$:
       $$\mu_j^{(t)} \leftarrow -\lambda_j^{(t)} + \frac{1}{k}\sum_{l=1}^{k} \lambda_j^{(t)}$$
5: Return:$\omega^* = -\frac{1}{C}\sum_{j=1}^{k} \lambda_j^{(T)}$.
```

III. Experiments

Table I highlights some key results on PDS running on the Spiral data set for $k = 2, 4, 10$ processing nodes. For $k = 10$ processing nodes, for example, PDS underwent 1400 iterations where most of the work per local processor was done during the first iteration (2040 msec) with 10-20 msec per subsequent iteration, totaling 14630 msec spent on the local SVMs. The integration time spent after the first iteration is 19560 msec where most of the work goes into computing the kernels between points assigned to the node and other remaining points. Integration time on subsequent iterations takes 10-20 msec due to retrieval from RAM of pre-computed kernel computations and extensive re-use of previous samples made by SVM-light. The total time spent on integration is 36590 msec making the overall parallel speedup stand on 4.7, which is slightly less than the expected $k/2$ speedup.

The PDS profile of behavior is dramatically different for an "easy" problem. Table II shows the same performance entries but on a 2D training set generated by two Gaussians where the distance between the means is ten times their variance. The data set contains $m = 20,000$ points; the kernel used is an RBF with $\sigma = 1$; $C = 1/m$ and the percentage of support vectors stands on 4%. Considering the case of $k = 10$, we see that the number of iterations is small (14 compared to 1400 for the Spiral) and the overall parallel speedup stands on 3.54. The speedup is smaller than for the difficult scenario of the Spiral data set mainly because the SVM-light on a single processor samples much less than m points in order to converge making the integration cycles of PDS somewhat non-efficient.

Tables III, IV show the PDS results (using the same format as the previous tables) of the two data sets. The MNIST stopped when generalization error flattened on 0.44% and Cover-Type stopped when the dual energy flattened. Cover-Type had significantly a higher percentage of support vectors (45%) than MNIST (5.7%) and thus achieved a higher parallel speedup for $k = 10$ nodes (6.36 compared to 3.45).

Workflow 1 Implementation PDS based on Haloop.

```
Preparation:
  Computation environment configuration
  Data partition and distribution to the computation nodes
  Create partition file
  function MainClass
      JobConf;//configure the MapReduce parameters and classnames
      HaloopDriver; //to initiate the MapReduce tasks
      while (condition) do
          JobConf;//reconfigure the MapReduce parameters;
          HaloopDriver;// initiate new MapReduce tasks, Broadcast combined support vectors to each computation node;
      end while
      Support vectors to each computation node;
      Get feedback results;// initiate new MapReduce tasks, Broadcast combined support vectors to each computation node;
      if (condition) then
          break; //if one SVM obtained, program finished
      end if
  end function
  function MapClass
      if (the first layer SVM) then
          Load data from local file system;
      else
          Read data broadcasted by Main class;
      end if
      Svm_train(); //the parameters of the SVM model are transformed through jobConf.
      Collector; //sent the training result to Reduce job through message.
  end function
  function ReduceClass
      Read data transformed from Map job;
      Combine support vectors of each two subSVM into one sample set.
      Collect; //feedback all the trained support vectors
  end function
```

IV. Conclusion

Classical SVM model is difficult to analyze large scale practical problems. Parallel SVM can improve the computation speed greatly. In this paper, parallel SVM model based on iterative MapReduce is proposed. We derived the algorithm from the principles of convex conjugate duality as a parallel block update coordinate ascent. The algorithm is mostly appropriate for computing clusters of independent nodes with independent memory and limited communication bandwidth. Experimental results on Kernel-SVM on synthetic and real data show an approximate parallel speedup of $k/2$ when using k processing nodes. Further study is required for evaluating the effects of RAM size on the performance of PDS. Generally, the smaller

TABLE I

PERFORMANCE ANALYSIS OF PDS ON THE "SPIRAL" SET OF 20, 000 POINTS (80% OF THE POINTS ARE SUPPORT VECTORS). (A) NUMBER OF PROCESSORS, (B) NUMBER OF ITERATIONS T ; (C),(D) TIME IN MSEC SPENT ON THE LOCAL SVM-LIGHT DURING THE FIRST TWO ITERATIONS, (E) TOTAL TIME SPENT SPENT ON LOCAL SVM-LIGHT OVER T ITERATIONS; (F),(G) TIME SPENT ON THE INTEGRATION PHASE (COMMUNICATION AND KERNEL EVALUATIONS) DURING THE FIRST TWO ITERATIONS, (H) TOTAL TIME SPENT SPENT ON INTEGRATION OVER T ITERATIONS, AND (I) THE PARALLEL SPEED ACHIEVED (SHOULD BE COMPARED TO $k/2$).

Processors	Iterations	SVM-light($t=1$)	SVM-light($t=2$)	Total SVM	Integration($t=1$)	Integration($t=2$)	Total Integration	Parallel Speed-up
$k=2$	900	60000	1200	107440	50000	20	71630	1.34
$k=4$	1120	14410	30	40760	45390	20	64220	2.29
$k=10$	1400	2040	10	14630	19560	10	36590	4.7
(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)

TABLE II

PERFORMANCE ANALYSIS OF PDS ON THE "GAUSSIANS" SET OF 20, 000 POINTS (4% OF THE POINTS ARE SUPPORT VECTORS). THE COLUMN DESCRIPTION FOLLOW THE SAME FORMAT AS TABLE I.

Processors	Iterations	SVM-light($t=1$)	SVM-light($t=2$)	Total SVM	Integration($t=1$)	Integration($t=2$)	Total Integration	Parallel Speed-up
$k=2$	6	380	5	403	303	2	314	1.49
$k=4$	10	135	2	150	317	2	335	2.2
$k=10$	14	32	1	43	233	2	259	3.54

TABLE III

PERFORMANCE ANALYSIS OF PDS ON THE MNIST SET OF 60, 000 POINTS (5.7% OF THE POINTS ARE SUPPORT VECTORS). THE COLUMN DESCRIPTION FOLLOW THE SAME FORMAT AS TABLE I.

Processors	Iterations	SVM-light($t=1$)	SVM-light($t=2$)	Total SVM	Integration($t=1$)	Integration($t=2$)	Total Integration	Parallel Speed-up
$k=2$	23	192360	8460	209122	162666	4980	176342	1.29
$k=4$	70	62215	2376	75933	160014	3792	186420	1.9
$k=10$	106	13169	522	18534	99730	224	126918	3.45

TABLE IV

PERFORMANCE ANALYSIS OF PDS ON THE COVERTYPE SET OF 300, 000 POINTS (45% OF THE POINTS ARE SUPPORT VECTORS). THE COLUMN DESCRIPTION FOLLOW THE SAME FORMAT AS TABLE I.

Processors	Iterations	SVM-light($t=1$)	SVM-light($t=2$)	Total SVM	Integration($t=1$)	Integration($t=2$)	Total Integration	Parallel Speed-up
$k=2$	1000	62321803	813354	71202282	11125508	125476	13065440	1.58
$k=4$	2000	32693808	535729	42807721	5429803	126784	8333806	2.6
$k=10$	3000	9622014	307999	16742959	1605412	92042	4196778	6.36

the RAM size compared to the size of the data set less caching can be done on pre-computed kernel computations which invariably will have a detrimental effect on the performance of PDS.

ACKNOWLEDGEMENT

This work is supported by Wuhan Chenguang Program under grant No.20127103136907, National Science & Technology Pillar Program under grant No. 2013BAK11B03.

REFERENCES

[1] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. Murthy, "Improvements to Platt's SMO Algorithm for SVM Classifier Design", in *Neural Computation*, vol.13, no.3, pp.637-649, March 2001.

[2] H. F. Yu, C. J. Hsieh, K. W. Chang, and C. J. Lin, "Large linear classification when data cannot fit in memory," in *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol.5, no.4, pp.1-23, February 2012.

[3] L. Diosan, A. Rogozan, and J. P. Pecuchet, "Improving classification performance of Support Vector Machine by genetically optimising kernel shape and hyper-parameters", in *Applied Intelligence*, vol.36, no.2, pp.280-294, March 2012.

[4] R. Caruana, T. Joachims, and L. Backstrom, "KDD-Cup 2004: results and analysis", in *ACM SIGKDD Explorations Newsletter*, vol.6, no.2, December 2004.

[5] T. Joachims, "Training linear SVMs in linear time", in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 20-23, 2006, Philadelphia, PA, USA.

[6] Z. Zhu, D. Bernhard, and Iryna Gurevych, "A monolingual tree-based translation model for sentence simplification", in *Proceedings of the 23rd International Conference on Computational Linguistics*, pp.1353-1361, August 23-27, 2010, Beijing, China.

[7] K. Crammer, O. Dekel, J. Keshet, S. S. Shwartz, and Y. Singer, "Online Passive-Aggressive Algorithms", in *The Journal of Machine Learning Research*, 7, pp.551-585, 2006.

[8] L. H. Lee, C. H. Wan, R. Rajkumar, and D. Isa, "An enhanced Support Vector Machine classification framework by using Euclidean distance function for text document categorization", in *Applied Intelligence*, vol.37, no.1, pp.80-99, July 2012

[9] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. "Linear classification and selective sampling under low noise conditions", in *NIPS* 21, pp.249–256, 2008.

[10] M. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent", in *NIPS*, pp.23–31, 2010.

[11] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines", in *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol.2,no.3, pp.1-27, April 2011.

[12] H. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, "Parallel Support Vector Machines: The Cascade SVM", in *Advances in Neural Information Processing Systems*, pp.33-41, 2005.

[13] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[14] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "The HaLoop approach to large-scale iterative data analysis," in *The International Journal on Very Large Data Bases*, vol.21, no.2, pp.169-190, April 2012.