A MapReduce based Parallel SVM for Large Scale Spam Filtering

Godwin Caruana[1], Maozhen Li[1,3] and Man Qi[2]

[1]School of Engineering and Design, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
[2]Department of Computing, Canterbury Christ Church University, Canterbury, Kent, CT1 1QU, UK
[3]The Key Laboratory of Embedded Systems and Service Computing, Ministry of Education, Tongji University, China

***Abstract*— Spam continues to inflict increased damage. Varying approaches including Support Vector Machine (SVM) based techniques have been proposed for spam classification. However, SVM training is a computationally intensive process. This paper presents a parallel SVM algorithm for scalable spam filtering. By distributing, processing and optimizing the subsets of the training data across multiple participating nodes, the distributed SVM reduces the training time significantly. Ontology based concepts are also employed to minimize the impact of accuracy degradation when distributing the training data amongst the SVM classifiers.**

Index Terms—*Machine Learning, Classification, Ontology Semantics, Support Vector Machine, Parallel Computing*

I. INTRODUCTION

Support Vector Machine (SVM) based approaches have persistently gained popularity in terms of their application for text classification and machine learning [1], [2]. Classification in SVM based approaches is founded on the notion of hyperplanes [3]. The hyperplanes act as class segregators in common binary classification, such as spam or ham in the context of spam filtering. SVM training is a computationally intensive process. Numerous SVM formulations, solvers and architectures for improving SVM performance have been explored and proposed [4], [5] including distributed and parallel computing techniques. SVM decomposition is another widespread technique for improving the performance in SVM training [6], [7]. Decomposition approaches work on the basis of identifying a small number of optimization variables and tackling a set of fixed size problems. Another widespread and effective practice is to split the training data into smaller fragments and use a number of SVM's to process the individual data chunks. This in turn reduces overall training time. Various forms of summarizations and aggregations are then performed to process the final set of global support vectors [8]. Numerous forms of decomposition which are based on a data splitting strategy approach can suffer from issues including convergence and accuracy. Challenges related to chunk aliasing as well as outlier accumulation tend to intensify problems in a distributed SVM context. Adopting a training data set splitting strategy commonly amplifies issues related to data imbalance and data distribution instability.

In this paper, we present an ontology assisted, parallel scheme for scalable SVM training. This work supplements current approaches by focusing on a number of aspects. We prototype a parallel SVM, building on the Sequential Minimal Optimization (SMO) algorithm [6]. We utilize a distributed computing framework, namely MapReduce [9] using Hadoop's implementation [10]. We also employ ontology semantics for improving overall accuracy.

The rest of the paper is organized as follows. In Section II we briefly describe the design of a parallel SVM algorithm. This work is then employed as a baseline for extension and accuracy improvement through the application of ontology assisted techniques, as described in Section III. Section IV describes basic experimental observations and results. Section V concludes the paper and points out some future work.

II. DISTRUBUTING SVM WITH MAPREDUCE

MapReduce is a generic framework and programming model intended to abstract large scale computation challenges. Popular implementations include Mars [11], Phoenix [12], Hadoop [10] and Google's implementation [13]. MapReduce was popularized by the latter and primarily motivated by the need to be able to parallelize the processing of Internet scale datasets. Programmatically inspired from functional programming, at its core are two primary features, namely a *map* and a *reduce* operation. From a logical perspective, all data is treated as a Key (K), Value (V) pair. Multiple *mappers* and *reducers* can be employed. At an atomic level however a *map* operation takes a $\{K_1, V_1\}$ pair and emits an intermediate list of $\{K_2, V_2\}$ pairs. A *reduce* operation takes all values represented by the same key in the intermediate list and processes them accordingly, emitting a final new list. Whilst the execution of *reduce* operations cannot start before the respective *map* counterparts are finished, all *map* and *reduce* operations run independently in parallel. Each *map* function executes in parallel emitting respective values from associated input. Similarly, each *reducer* processes different keys independently and concurrently.

A simple approach to parallelize and improve SVM training performance is thus by splitting training data and capitalizing on respective MapReduce functionality. From a Hadoop MapReduce [10] perspective, the data splitting strategy can be done according to the number of MapReduce tasks that will be employed. Each Map task ($MAP_1 \ldots MAP_n$) will process the associated data chunk ($DataChunk_1 \ldots DataChunk_n$) and generate a respective set of

978-1-61284-181-6/11/$26.00 ©2011 IEEE

Support Vectors (SV^{set}_1 … SV^{set}_n). These can then forwarded to a single (or multiple) Reducer ($REDUCE_1$) which will contribute the respectively aggregated Support Vector Set (*SV*), weight (*w*) and bias (*b*) elements of the global SVM to a final learned model. In our prototype, the aggregation of the weight (*w*) and bias (*b*) elements are performed using a sum and average strategy respectively. The final output is used as the final classification model including the necessary information for the objective function to be able to classify unseen data. This process is described pictorially in **Fig. 1**.

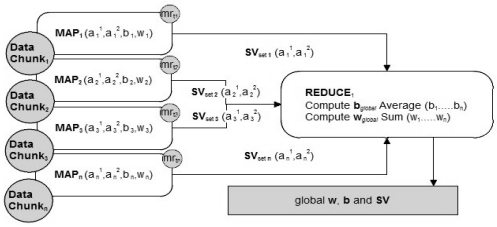

Fig 1: SVM process aggregation

Algorithm 1 presents the proposed parallel SMO based on the one described in [6]. The *map* segment of the algorithm is basically the same as the original SMO algorithm except for the fact that it is applied for each participating *mapper*. The primary difference lies in the approach that the global b threshold (b_{global}) and weight vector (w_{global}) are computed via the *reducer*, using an *average* and *sum* strategy respectively as described in the pseudo code. In Algorithm 1, for each $data_{chunk}$ we associate a Map (Map_j) operation. In this context, line 4 initializes the necessary structures, primarily the α multipliers and the objective function. Lines 5 – 10 portray the SMO optimization process. Iterations are based on the selection and optimization of two Lagrange multipliers, subsequently the objective function. Line 11 checks for the respective exit conditions, whilst line 12 updates the bias threshold accordingly. For actual implementations multiplier selection is frequently based on approaches such as heuristics, albeit strategies vary with specific implementations. Lines 13 and 14 store the Lagrange multipliers and update the local weight vector for the specific *map* (Map_j). In contrast with the sequential SMO algorithm presented in [6], we perform two additional steps using the *reduce* phase of the MapReduce prototype. Basically, the *reducer* performs an average computation on all respective b outputs emitted by the individual Map (Map_j) operations (b_{global} - Line 18) as well as a sum operation on the weight vectors emitted by the respective Map (Map_j) operations (w_{global} - Line 19).

Weka's SMO [14] implementation is employed as a baseline solver. For this work we focused on linear SVMs, although the approach can be easily extended and applied to non-linear variants as well. The base SMO algorithm is decomposed and re-structured to benefit from MapReduce. As discussed, each MapReduce *map* processes an associated data chunk in its entirety. The output of each *map* process is the localized (per data chunk) SVM weight vector (Algorithm 1: w_j) and the bias (Algorithm 1: b_j) threshold. Again, the primary role of the associated *reduce* phase is to compute the global weight vector (Algorithm 1: w_{global}) by summing the individual *maps* weight vectors. The bias thresholds from each *map* output are averaged by the respective *reduce* phase (Algorithm 1: b_{global}).

From a time complexity perspective the original sequential representation, i.e. $O(m^2n)$ can now be contextualized in a MapReduce environment and expressed as:

$$O\left((m^2n / S)+ n \log(S)\right) \quad (1)$$

where *n* is the dimension of the input, *m* are the training samples and *S* the number of MapReduce nodes.

1. **MAP**$_j$ $\forall j \in \{1 \ldots data_{chunk}\}$
2. **input**: set of training data x_i, corresponding labels y_i, $\forall i \in \{1 \ldots \ell\}$
3. **output**: weight vector w_j, α_j array, b_j and SV
4. **initialize**: $\alpha_i \leftarrow 0$, $f_i \leftarrow -y_i$ $\forall i \in \{1 \ldots \ell\}$
5. **compute** b_{high}, I_{high}, b_{low}, I_{low}
6. **update** ${}^{\alpha}I_{high}$ and ${}^{\alpha}I_{low}$
7. **repeat**
8. **update** f_i, $\forall i \in \{1 \ldots \ell\}$
9. **compute** b_{high}, I_{high}, b_{low}, I_{low}
10. **update** ${}^{\alpha}I_{high}$ and ${}^{\alpha}I_{low}$
11. **until** $b_{low} \leq b_{high}, + 2\Gamma$
12. **update** b_j bias term
13. **store** updated α_{j1} and α_{j2}
14. **update** w_j
15. **REDUCE**
16. **input**: set of $\mathbf{Map}_j$ weight vectors w_j $\forall j \in \{1 \ldots data_{chunk}\}$, set of $\mathbf{Map}_j$ bias b_j $\forall j \in \{1 \ldots data_{chunk}\}$
17. **output**: global weight vector w, average b and SV
18. **compute** $b_{global} = \sum_{j=1}^{Map_j} b^{data}_{chunk} / Map_j$
19. **compute** $w_{global} = \sum_{j=1}^{Map_j} b^{data}_{chunk}$

Algorithm 1: MapReduce based parallel SMO

Where:

Map_j = MapReduce Map
$Data_{chunk}$ = training data associated with Map_j
x = training elements, y = class labels for x
w_j = local (Map_j) weight vector
b_j = local (Map_j) b threshold
I = training data index set
α_j = Lagrange multiplier(s)
b_{global} = global b threshold
w_{global} = global weight vector

III. Ontology For Accuracy Augmentation

Training an SVM by splitting the input data set and working on the individual sub-sets separately may reduce the overall accuracy [15]. In order to improve the overall classification accuracy of the parallel SMO algorithm, we extend it with an ontology based enhancement process. We designed *SPONTO,* short for **SP**am**ONTO**logy, which acts as a feedback loop base for the training and classification processes. This is in contrast with the work presented in [16] where the ontology itself is employed for classification. The feedback loop is then employed to re-train the parallel SVM with added intelligence to improve overall accuracy. This is performed to mitigate the accuracy degradation challenge introduced due to the training data file splitting strategy and respective separate SVM computation strategy adopted.

SPONTO reflects all the basic elements presented in the SpamBase [17] dataset as well as additional attribute assertions. SPONTO is also employed to carry additional intelligence such as mail class (Ham or Spam), whether the machine learning based (parallel SVM) classifier outcome for instance classification was correct as well as support for instance weights. The intelligence conveyed through the supplementary instance attributes via end user contribution is employed for correcting and influencing training data. The end user contributed, ontology based intelligence augmented training sets are then employed for the regeneration of the classifier by the parallel SVM.

A base RDF graph is generated from the SpamBase ARFF and based on the SPONTO ontology structure. We transform the Weka ARFF format to an equivalent RDF representation. We then apply the learned model from the parallel SVM on the instances which require classification. For each test instance, we generate a new ontology instance based on SPONTO. Ontology generation is performed via the extraction of instance data and automated generation of a respective SPARQL query, applied on the base RDF graph to identify respective misclassified elements. Misclassified nodes are identified as a set of final ontology instances that are used for user contribution. End users contribute preference and intelligence by increasing individual instance weights, removing instances or modifying instance classification outcomes for example. For the prototype, we employ Protégé, the Ontology Editor and Knowledge Acquisition System [18] for Ontology interaction and end user contribution.

The final process involves the re-generation of the Weka ARFF input files from the final ontology for subsequent processing by the parallel SVM. We increase correctly classified instance weights, correspondingly decrease the instance weights of incorrectly classified ones and merge these instances with the original input. This closes the ontology assisted feedback loop – described pictorially in **Fig. 2**.

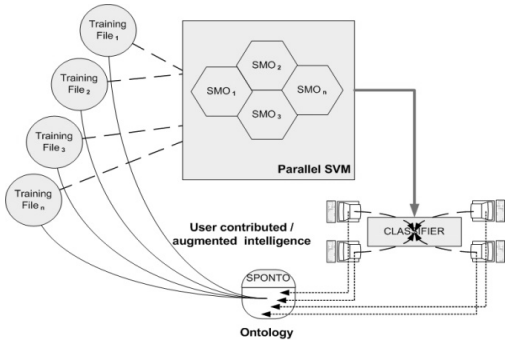

Fig 2: Ontology assisted feedback loop

IV. EXPERIMENTAL RESULTS

A number of experiments were carried out to identify the accuracy and performance of the parallel SMO, comparing it with the Sequential counterpart. For all classification experiments carried out, the SpamBase [17] dataset was employed. There are 4601 instances in the original SpamBase dataset. A baseline experiment intended to identify typical sequential SMO performance using the SpamBase dataset on a typical desktop computer was performed. Weka's SMO classification scheme was employed [19], using a number of unlabeled instances and varying the number of training instances. The time required to train the SMO sequentially using 128,000 instances on a single computer node was ≈ 563 seconds. A sequential SMO test with 327,750 instances failed. The same numbers of instances were processed in ≈ 134 seconds using a simple Hadoop MapReduce cluster with 4 computing nodes of similar processing capabilities as the desktop machine employed for the sequential tests. **Fig. 3** shows a comparison of the sequential and parallel SMO efficiency in training, using a varying number of nodes. It is also believed, as shown in **Table I** that the baseline parallel SMO classifier accuracy compares favorably with the figures identified when the model was trained using the sequential approach. Given an appropriate number of processing nodes and map tasks, training the SVM using the proposed MapReduce approach reduces training time considerably. This also provides increased scope for possible re-training.

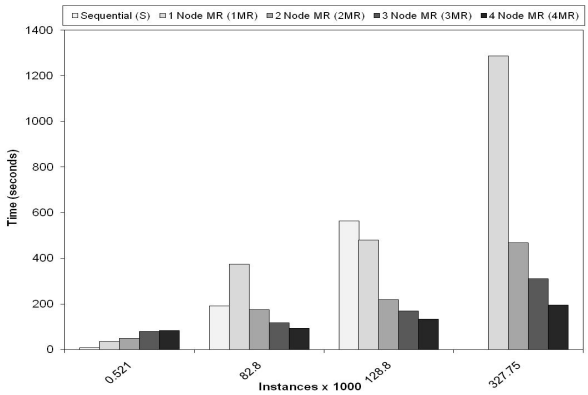

Fig 3: Efficiency of the Parallel SMO

TABLE I. TRAINING TIME AND ACCURACY COMPARISON

	Sequential	**4 Node MapReduce Avg.**
Correctly Classified	≈ 94.03 %	≈ 92.04 %
Incorrectly Classified	≈ 5.97 %	≈ 7.96 %
Training time 128,000 instances	≈ 563.70 s	≈ 134.50 s

As indicated earlier, the baseline SMO is inherently sequential making use of singular global data structures. On the other hand the parallel version employs numerous separate support vector machines based on the specific file splits and training sets. This specific approach is the primary influencer of the accuracy difference between the sequential and the parallel versions. Evaluating the accuracy of the parallel SVM computed classifier using a *random* set of 267 instances yields 226 correct and 41 incorrectly classified instances respectively. To further improve the accuracy of the parallel SMO, we augment the base training sets with additional intelligence through ontology based end user contribution as described in Section III. This is performed by influencing individual misclassified elements as well as attributing increased weighting to user identified and selected instances.

With this approach, we identified an average of ≈ 5 % accuracy improvement. The rate of accuracy degradation over the number of file splits is significantly slower as shown in **Fig. 4**. The slower rate of accuracy degradation is attributed to the ontology based intelligence augmentation. This is achieved by merging end user optimized instances to the respective training split files which are also weighted to influence overall relevancy. The minimum accuracy increases by 1.7% which reflects a scenario where the number of file splits is minimal, 4 chunks in this case, whilst the maximum of 7.5 % occurs when there is the largest number of splits, namely 48 in this case. Based on an average accuracy improvement of 4.6 % over the baseline parallel SMO, **Fig. 5** shows that using the ontology intelligence augmented approach, the MapReduce based SMO achieves an accuracy of 96% on average, which is better than the original sequential SMO.

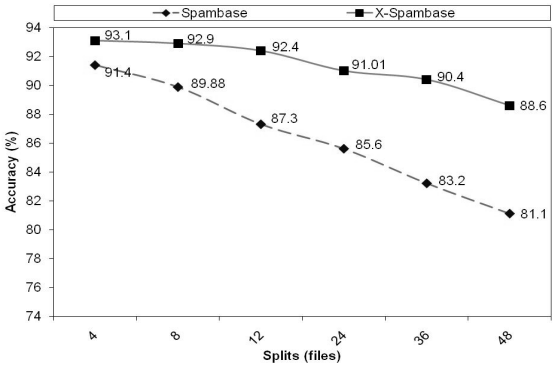

Fig 4: Accuracy degradation rate comparison.

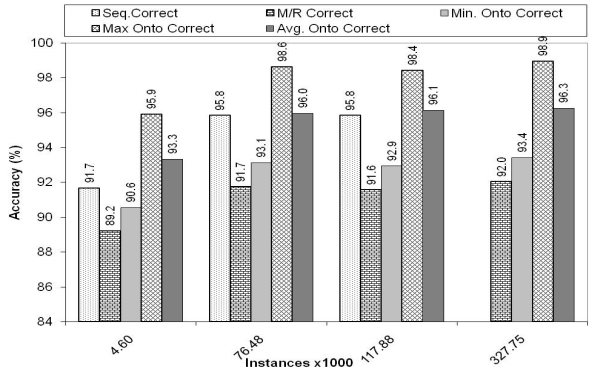

Fig 5: The accuracy of the ontology augmented parallel SMO.

V. CONCLUSIONS AND FUTURE WORK

In this paper we presented a parallel SVM algorithm for scalable spam filtering. By splitting the training set and applying distributed computing techniques such as MapReduce we can improve the training time considerably. However, this has varying yet noticeable degrees of accuracy degradation. In our work, we employ ontology based semantics to improve the accuracy of the parallel SVM. For future work, we intend to research appropriate schemes to extract additional intelligence from annotated instances and employ this within the machine learning, Parallel SVM feedback loop process. We believe that accuracy can be also further improved via automated annotation similar to [20].

REFERENCES

[1] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," *Artif. Intell. Rev.*, vol. 29, no. 1, p. 63–92, 2008.

[2] A. Blanco, A. M. Ricket, and M. Martin-Merino, "Combining SVM classifiers for email anti-spam filtering," in *Computational and Ambient Intelligence. Proceedings 9th International Work-Conference on Artificial Neural Networks, IWANN 2007*, Berlin, Germany, 2007, pp. 903 - 10.

[3] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[4] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik, "Parallel Support Vector Machines: The Cascade SVM," in *NIPS*, 2004.

[5] T.-N. Do and F. Poulet, "Classifying one billion data with a new distributed svm algorithm," in *RIVF*, 2006, pp. 59-66.

[6] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," Cambridge, MA, USA: MIT Press, 1999, p. 185–208.

[7] F. R. Bach and M. I. Jordan, "Predictive low-rank decomposition for kernel methods," in *ICML*, 2005, pp. 33-40.

[8] G. Zanghirati and L. Zanni, "A parallel solver for large quadratic programs in training support vector machines," *Parallel Comput.*, vol. 29, no. 4, p. 535–551, 2003.

[9] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, p. 107–113, 2008.

[10] "Welcome to Apache Hadoop!," *Welcome to Apache Hadoop!* [Online]. Available: http://hadoop.apache.org/. [Accessed: 09-May-2011].

[11] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a MapReduce framework on graphics processors," in *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, New York, NY, USA, 2008, p. 260–269.

[12] K. Taura, T. Endo, K. Kaneda, and A. Yonezawa, "Phoenix□: a Parallel Programming Model for Accommodating Dynamically Joining Resources," in *In Proc. of PPoPP*, 2003, p. 216–229.

[13] T. Aarnio, "Parallel Data Processing with Mapreduce." [Online]. Available: http://www.cse.tkk.fi/en/publications/B/5/papers/Aarnio_final.pdf. [Accessed: 29-Apr-2010].

[14] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005.

[15] N. A. Syed, S. Huan, L. Kah, and K. Sung, "Incremental Learning with Support Vector Machines," *Incremental Learning with Support Vector Machines*, 1999. [Online]. Available: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.6367. [Accessed: 03-Aug-2010].

[16] S. Youn and D. McLeod, "Efficient spam email filtering using adaptive ontology," in *International Conference on Information Technology, Proceedings*, 2007, pp. 249-254.

[17] A. Asuncion and D. J. Newman, *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences, 2007.

[18] "About the Protégé Team," *Protege*. [Online]. Available: http://protege.stanford.edu/aboutus/aboutus.html. [Accessed: 15-Aug-2010].

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, p. 10–18, 2009.

[20] C. Posse, P. Paulson, B. Baddeley, R. Hohimer, and A. White, "Automating Ontological Annotation with WordNet," in *Proceedings of the 3 rd Global WordNet Conference, Jeju Island, South Korea*, 2006.