ISSN(O): 2456–026X

Efficient Support Vector Machine for Multi-Core Systems

Rahul Kumar[1], Saumil Maheshwari[2], Apoorva Mishra[3*], Ishu Garg[4], Anupam Shukla[5]
[1]Soft Computing and Expert Systems Lab, ABV-IIITM, Gwalior, Madhya Pradesh, India.
Email: rahul73.iiitm@gmail.com
[2]Soft Computing and Expert Systems Lab, ABV-IIITM, Gwalior, Madhya Pradesh, India.
Email: saumilmaheshwari@yahoo.co.in
[3]Soft Computing and Expert Systems Lab, ABV-IIITM, Gwalior, Madhya Pradesh, India.
Email: apoorvamish1989@gmail.com
[4]Department of Computer Science and Engineering, IIT, Madras, India.
Email: Ishugarg567@gmail.com
[5]Soft Computing and Expert Systems Lab, ABV-IIITM, Gwalior, Madhya Pradesh, India.
Email: dranupamiiitm@gmail.com
*Corresponding Author

Abstract: We are in an era where scaling up the processing speed of computers by increasing the clock speed of processors had become an ineffective strategy for handling Big Data. Now, computers have increased number of processors embedded within, which has motivated various programmer to take advantage of these architectures. To increase the processing speed, parallelization of algorithms can be used as an alternative rather than optimizing the existing algorithms. In the paper, we have deployed a parallel programming framework that could take advantage of multi-core structure of processors. This framework can be used for different learning algorithms that satisfies Statistical Query Model (SQM) i.e., can be simplified in "Summation form". We have used Google's MapReduce model for achieving parallelization in learning algorithms on two datasets (Ionoshpere, Breast Cancer datasets) from UCI machine learning repository. Our implemented model gives better result than existing implementations and also verifies the linear speedup of processing with an increase in number of cores in the processors.

Keywords: Big data, Machine learning, MapReduce, Multi-core, Statistical Query Model (SQM).

I. Introduction

Multi-core processors are widely used as the most viable means for delivering high performance processing for high density of data. However, this potential of processors could only be realized, in the long run, if the application programs running on the processors are suitably parallel [1]. Parallelism obtained via mapping is highly dependent on the program while the best mapping for maximum parallelism is dependent on the hardware of the systems [2]. During mapping of programs to platform many decisions had to be taken care that includes how much parallelism has to be exploited, the number of processors have to be used, how to schedule parallelism in the hardware etc. Mapping to cores highly depends on the costs of communication between elements, computation costs, network latency and other hardware costs [3]. However, difficulties in thread-based parallel programming, utilizing multiple processing cores of hardware is still a challenging tasks. As it requires, not only synchronization but also involves load balancing and locality of references and detailed understanding of hardware implementation [4].

An alternative technique to handle different ambiguities is to use a runtime system for concurrency management. In this approach, programmer has to summarize the computation in terms of many sub-tasks and runtime system automatically manages the problem of synchronization, locality and load balancing for a complete and efficient execution of task [5]. This led to decrease in complexity of programs coded by the programmer and these load is shifted to systems. One of the frequent used parallel programming model that depends on this runtime approach of the system is Google's MapReduce [6]. In this programming model, computation can be described as a set of different map and reduce tasks. Use of parallelization techniques with modified algorithms are the key to gain scalability and higher performance for different data analysis [6][7].

According to Moore law, the density of circuits will be doubling every generation and thus enabling programmer to map parallelism of algorithms with the hardware architecture. There had been various studies on distributed learning but very little of these studies focuses on our work, how MapReduce could help in implementing different machine learning algorithms parallely on different clusters [9][10]. Many of the previous work had focused on parallelizing individual algorithms, they had not intended to form a parallel programming paradigm so that a general class of algorithms could fit in. In this paper main focus is to develop a general technique which could be

used by different programmers to parallelize SVM. The model so developed, does not possesses hardware limitation, thus enabling to parallelize algorithms with different systems, thus enabling to work in heterogeneous clusters. We had developed the model for SVM as it can be described in "Summation form". In the paper there is comparison with other traditional methods to parallelize SVM with our devised approach.

The rest of paper is divided into sections with Section 2 describing the background and basic motivation for the experiment; Section 3 describes the concepts that were used in implementing the machine learning algorithms parallely in a cluster with small explanation about the algorithms; Section 4 describes the implementations and other set-up for the experiment; Section 5 with the results obtained and detailed discussion; Section 6 with the conclusion of our experiment.

II. Background and Motivation

Google's map reduce is a parallel programming paradigm used mainly for large processing and storing large amount of data. It was basically proposed to work with a large number of nodes where there is requirement of extensive computation over Big Data [11]. The basic principle of the original MapReduce was to spawn into multiple processes to achieve parallelism.

Modifications in standard MapReduce could make it suitable for multi-core and multiprocessor systems. In this approach, shared-memory threads are used to achieve parallelism in the program [12]. The system launches multiple threads to complete the execution process. The computation occurs in two phases Map Phase, Reduce Phase [9]. In first phase, the data is split-ted into chunks depending on the user specification and hdfs (hadoop file system) blocks and individual map task runs on each chunk. In last phase, result from independent chunks are collected and merged to give the computation result.

A. Hadoop

Hadoop is an open source Java-based programming frame- work which not only supports processing of large amount of data but also provides storage of those data in a distributing environment. It was developed by Doug Cutting and Mike Cafarella in the year of 2006, to support distribution of Nutch Search engine [13]. It was inspired by the approach followed in Google's MapReduce in which application is sub-divided into numerous small parts. Hadoop has emerged as a foundation pillar for processing of Big Data that includes scientific analytics, data from sensor networks, business and sales planning, data from IOT sensors [14]. Basic architecture of Hadoop Framework is illustrated in the picture depicted which states it is consist of two parts named Hadoop File System and MapReduce.

B. Summation Form and Statistical Query Model

Statistical Query model was developed by Kearns as an initiative for handling noise in the probability approximately correct learning (PAC learning) model. Main idea in the algorithm is that instead of using random values, the learner used to gain information about the unknown function by querying from various statistics over the labeled examples.

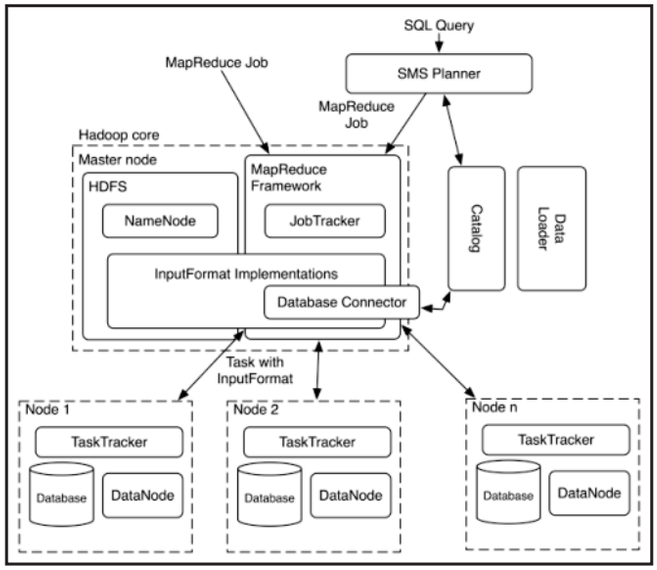

Fig. 1: Hadoop Infrastructure

Multi-core systems are best suited for concurrent applications, where there is little communication and information exchange between multiple cores.

The working of the SQ model is illustrated by given example. We have a function f(x; y) which is distributed over multiple instances, statistical query oracle returns an estimate of the expectation of f(x; y) i.e. average over training divided by test distribution. Some of the machine learning algorithms that could calculate sufficient gradients are best fitted for the model. These calculations could be made in a batch i.e. they could be expressed as a summation form over datasets. There are various algorithms that exhibits summation form which have been discussed in lower sections. However, when an algorithm possesses such a property, we could easily distribute these computations over multiple number of cores. For doing so, we have to divide the datasets into pieces which are then provided to cores and then results are aggregated at the very end. Such a form of the algorithm is described as "summation form".

C. MapReduce Architecture

There are many programming frameworks that could utilize the summation form, but the success of Googles functional programming paradigm MapReduce has got the attention of many developers to adopt for the technology. We use MapReduce in multicore systems for parallel programming construct.

The Fig. 2 shows abstract view of working of multiple mappers-reducers simultaneously inside the system, how it processes the data in the systems. At the beginning, Map- Reduce engine of the system will split the datasets on the basis of size of training

examples. Then the splitted data is being cached for further map-reduce invocations. In the approach, there will be a master which would be responsible for coordination between mappers and reducers. The master has the task of splitting the data to different map-reduce tasks, and collects all the intermediate processed data. These intermediate data are then sent to reducers which returns the final processed result. For additional scalar information, there is support of communication between different mapper/reducer.

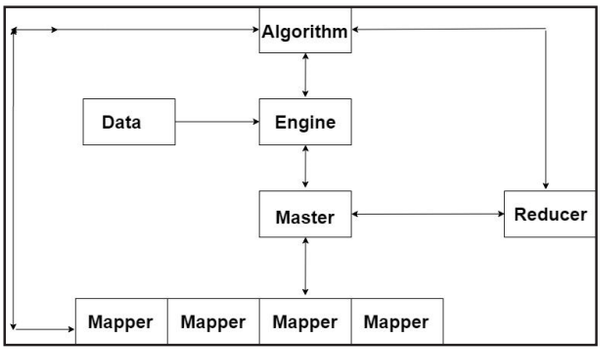

Fig. 2: MapReduce Working

III. Support Vector Machine

Support Vector Machine (SVM) was introduced in the year 1995 by Vladimir N. Vapnik. It is one of the most popular machine learning algorithm that is based on supervised learning. It is based on the idea of drawing a hyperplane which would act a separator between the classes [15]-[19]. It can only be used for binary classification i.e. could separate a set of training examples in two different classes (x_1, y_1), (x_2, y_2), ..., (x_m, y_m), where x_i R_2 denoting feature vector in d-dimensions and y_i −1, +1 is respective class label. The line separator is selected in such a way that it would give maximum distance from the support vector of both classes and thus is also known as maximum margin classifier.

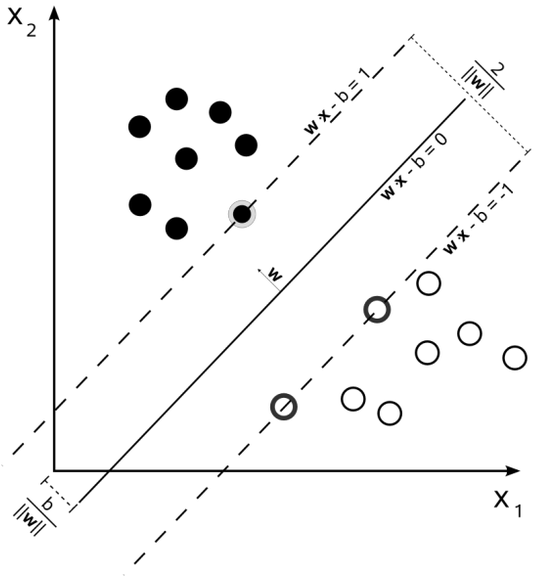

Fig. 3: Support Vector Machine

Support Vector Machine classifier performs a binary classification. Fig. 3 illustrates the functioning of linear kernel based SVM, in which there is a mapping of nonlinear input space into newly linear separable state space. In particular, all input vectors that lies on one side of the hyperplane is assigned value different from the ones lying on other side, usually these values are denoted as -1 and +1. The training instances that is present in nearest to the hyperplane are called as support vectors. These support vectors are then used to determine the margin of hyperplane and thus the decision boundary. In case of linear datasets hyperplane itself can divide them into two classes. In case of nonlinear separable datasets, SVM uses kernel functions. These kernel functions are used for mapping these non-linear datasets to high dimensional space denoted as $\varphi : Rd \rightarrow H$ f where $d < f$. Thus, an optimal separating hyperplane in the new feature space is constructed by a kernel function K (xi, xj) which is given as product of input vectors xi xj and $K(xi, xj) = \varphi(xi).\varphi(xj)$. The main disadvantages of SVM are [20]–[24]:

- Selection of appropriate kernel functions.
- Estimation of values of gaussian parameters.
- Number of feature selection in case of smaller datasets.

A. Efficient Parallel SVM

Parallel version of SVM is based on idea of splitting the datasets into smaller chunks and use multiple numbers of SVM to process each fragment and then finding local support vectors for each chunks. This led to decrease in overall time of computations. Every SVM gives only the partial solution which is then used by other SVMs to find global solutions. By doing this, we can optimize the work through distribution over small optimizations. The results of first stage SVM is then given to next stage SVM thus posing a hierarchical architecture. Merging the results of two or more SVMs in one SVM is based on the fact that SVM can be expressed as summation form, thus enabling users to make it parallel. The merging process is a recursive process until we have one final SVM which would give the results using all the previously used SVMs. In the implementation, each SVM has to care for a subset of data, resulting in smaller training set thus reducing the training costs.

Google's Mapreduce framework was used for implementing this parallel version of SVM. As shown in the Fig. 4, first data is divided into chunks which is then provided to different mappers simultaneously and then these results are accumulated by the reducer in the reduce phase. The reduce phase gives the final globally updated SVM which will be then used for classification. We thus perform batch gradient descent for optimizing the objective function [25]. The mappers would calculate partial gradient which is then summed up by the reducer to update the weight vector. Initially, we had a large dataset D which is divided into n parts (depending on the number of cores used for algorithm) like (D1, D2, D3,…, Dn). These datasets are then passed to individual mappers by MapReduceDriver in which mapping tasks are done. The trained SVM from each mapper is

sent to reduce phase in which reducer perform reduce actions merging the values using all the SVMs making a global SVM which would then be used for testing purposes.

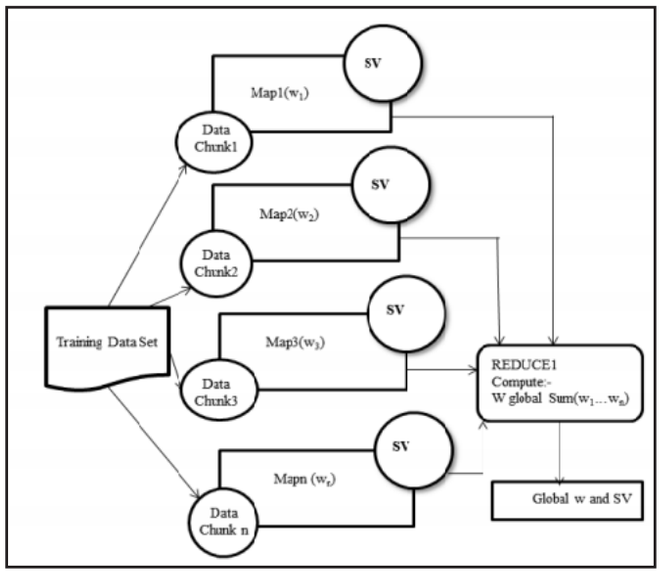

Fig. 4: SVM Using MapReduce Framework

In the experiment, for gradient calculation, stochastic gradient descent had been used as the approach gives better results for those supervised learning algorithm functions that can be expressed as summation form. Along with hindgeloss function and L2-norm regularization were used for updating the new weights at each iteration in the algorithm. L2 form is known for giving stable solution and most significantly single solution.

IV. Experimental Setup and Dataset

For providing fair comparisons, we conducted experiments with varying cluster size, varying the numbers of cores used per system for comparing the speed up in the performance. Datasets were then converted to Resilient Distributed Datasets (RDD) so that they could be easily divided. Multiple single node clusters were joined using the proxy server. In the architecture, one node was made as master to which other nodes connect through proxy given at the run-time execution of programs. Single node cluster for the experiment was built on Intel i5-3210M processor with dual core 2.5GHz CPUs and 4GB physical memory. Operating system used in the experiment was Ubuntu 14.04. Multi-node clusters were built on different systems with different configurations architecture consisting of i3-2370M, i5-2520M, i5-3210M processors with 2GB, 4GB, 4GB primary memory respectively.

A. DataSets Description

1) *Ionosphere Datasets*: Ionosphere dataset is a standard UCI machine learning dataset for binary classifications [26]. The dataset consists of 34 attributes which are labelled as 2 different classes. It is a standard binary classification problems in which labels are denoted as +1 or -1. During the implementation phase, rows which were labelled as -1 were renamed as 0 and +1 with 1.

2) *Breast Cancer Datasets*: Breast cancer datasets were also taken from UCI machine learning repository consisting of 11 different features [26]. Last feature indicates labelling of class, 2 for benign and 4 for malignant. All other feature except the first one is an integral value which lies between 1 and 10 [8]. In the experiment, we work on 9 features which were used to predict whether these attribute leads to class 1 (benign) or class 2 (malignant).

V. Results and Discussion

Table I shows the speed up in performance and accuracy of parallel SVM algorithms for same size datasets by varying the degree of parallelism at execution time. The values give us idea how efficiency of same algorithm could be increased by splitting the original datasets into small datasets and merging their local results to update global variables

Table I: Measurement of SVM Accuracy by Varying Number of Cores in Single Node Cluster

Number of Cores	Computation Time (sec)	Accuracy (%)
1	72.648	83.24
2	69.636	85.63
3	66.300	88.73
4	66.450	90.14
5	77.944	86.73
6	86.398	84.34

Table II shows how the new implementation of algorithm is better from earlier existing implementation of the algorithm [27]. It could be seen that the efficiency of algorithm is increased when datasets is broken into more chunks leads to better results. Loading the complete datasets into main memory at the start of computation leads to decrease in computation time in single node cluster, as a result the communication cost and I/O time decreases and thus overall execution time. In-memory computation of mapreduce tasks reduces the overall time to a great extent and thus processing very fast. The computation time slightly increases than previous implementation for multi-node clusters due to network latency, partitioning of datasets, transformations, scheduling delays and many other factors. These factors become negligible in case of large clusters but have significant for smaller size datasets. The results were obtained for Ionosphere datasets which consists of 34 attributes.

Fig. 5 gives the graphical representation of relationship of variation of accuracy between both implementations. New implementation has got better results due to use of stochastic gradient descent approach with SVM, and each stage the error is propagated using least square error methods. Fig. 6 describes that for smaller size datasets, utilizing full potential of multi-

core systems give better results than multi-node clusters. So, for smaller datasets it is advisable to use single node cluster rather than multi-node cluster.

TABLE II: MEASUREMENT OF SVM ACCURACY ON IONOSPHERE DATASETS BY VARYING NUMBER OF NODES IN MULTI-NODE CLUSTERS

Number of Nodes	Existing Implementation Computation Time (sec)	New Implementation Computation Time (sec)	Existing Implementation Accuracy (%)	New Implementation Accuracy (%)
1	123.51	72.65	86.43	83.24
2	80.46	85.53	85.07	85.36
3	74.34	98.75	84.43	86.83
4	74.61	96.36	88.10	90.14

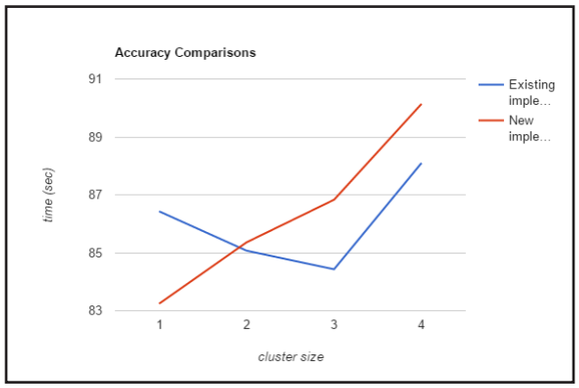

Fig. 5: Accuracy Variation between New Implementation and Existing Implementation

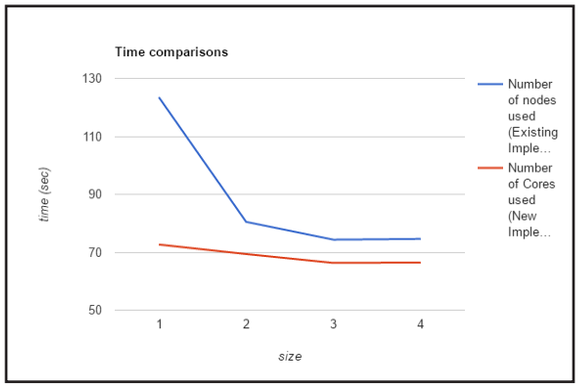

Fig. 6: Time Variation by Varying Number of Cores in Single Node Cluster and by Varying Nodes in Cluster

VI. CONCLUSION

According to AMD and Intel products roadmaps, there would be multiplicative increase in processing cores numbers on a single chip over the next decade. This would eventually lead to increase in processing power of each individual core. Thus there is need that these machine learning could take advantage of bounty of Moore's law for larger datasets and problems. For this, it has become important to adopt a parallel programming paradigm which could exploit the full potential of multi-cores. In the paper, we tried to convert the problem into summation form, so that it could take advantage of map-reduce framework. Using the architecture, we get a linear speedup in processing with increase in used number of cores, additional constraints that prove to be bottleneck in the performance. By loading the datasets into main memory before computation makes the computation faster. L2-norm regularization updater when used with hindgeloss function for SVM gives better results than combination of shrinking approach and epsilon loss functions. For a small size datasets if we run more map- reduce task simultaneously it would degrade the performance rather than increasing the speedup. We have experimentally showed that by just throwing more cores to the problems also leads to speeding up machine learning algorithms apart from other optimizations. Thus there is always a tradeoff involved between selection of number of cores for a particular datasize i.e. for smaller datasize increasing cores degrades the performance but for larger datasize it is advisable to go for more cores based on number of partition of datasets.

REFERENCES

[1] Z. Sun, F. Chen, M. Chi, and Y. Zhu, "A spark-based big data platform for massive remote sensing data processing," In International Conference on Data Science, Springer, pp. 120-126, 2015. J. C. Maxwell, *A Treatise on Electricity and Magnetism*, 3rd ed., vol. 2. Oxford: Clarendon, pp. 68-73, 1892.

[2] O. Y. Al-Jarrah, P. D. Yoo, S. Muhaidat, G. K. Karagiannidis, and K. Taha, "Efficient machine learning for big data: A Review," *Big Data Research*, vol. 3, no. 2, pp. 87-93, 2015.

[3] S. Bardhan, and D. A. Menasce´, "Predicting the effect of memory contention in multi-core computers using analytic performance models," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2279-2292, 2015.

[4] J. Rosen, N. Polyzotis, V. Borkar, Y. Bu, M. J. Carey, M. Weimer, T. Condie, and R. Ramakrishnan, "Iterative mapreduce for large scale machine learning," arXiv preprint arXiv:1303.3517, 2015.

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, USENIX Association, pp. 265-283, 2016.

[6] M. Boehm, S. Tatikonda, B. Reinwald, P. Sen, Y. Tian, D. R. Burdick, and S. Vaithyanathan, "Hybrid paral-

lelization strategies for large-scale machine learning in system ml," In *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 553-564, 2014.

[7] B. Wang, S. Huang, J. Qiu, Y. Liu, and G. Wang, "Parallel online sequential extreme learning machine based on mapreduce," *Neurocomputing*, vol. 149, Part A, pp. 224-232, 2015.

[8] M. W. Huang, C. W. Chen, W. C. Lin, S. W. Ke, and C. F. Tsai, "SVM and SVM ensembles in breast cancer prediction," *PloS ONE*, vol. 12, no. 1, 2017.

[9] W. Romsaiyud, and W. Premchaiswadi, "An adaptive machine learning on map-reduce framework for improving performance of large-scale data analysis on ec2," In *2013 Eleventh International Conference on ICT and Knowledge Engineering*, pp. 1-7, 2013.

[10] W. Jiang and G. Agrawal, "Mate-cg: A map reduce-like framework for accelerating data-intensive computations on heterogeneous clusters," In *2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, IEEE, pp. 644-655, 2012.

[11] J. Lin, and A. Kolcz, "Large-scale machine learning at twitter," 2012.

[12] D. Harnie, M. Saey, A. E. Vapirev, J. K. Wegner, A. Gedich, M. Steijaert, H. Ceulemans, R. Wuyts, and W. De Meuter, "Scaling machine learning for target prediction in drug discovery using apache spark," *Future Generation Computer Systems*, vol. 67, pp. 409-417, 2017.

[13] S. R. Upadhyaya, "Parallel approaches to machine learning a comprehensive survey," *Journal of Parallel and Distributed Computing*, vol. 73, no. 3, pp. 284-292, 2013.

[14] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new framework for parallel machine learning," arXiv:1006.4990.

[15] N. Cristianini, and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, Cambridge, 2000.

[16] S. Agarwal, G. Pandey, *et al.*, "SVM based context awareness using body area sensor network for pervasive healthcare monitoring," In *Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia*, ACM, pp. 271-278, 2010.

[17] D. Tomar, and S. Agarwal, "A comparison on multi-class classification methods based on least squares twin support vector machine," *Knowledge-Based Systems*, vol. 81, no. C, pp. 131-147, June 2015. [Online] Available: http://dx.doi.org/10.1016/j.knosys.2015.02.009

[18] S. Agarwal, *et al.*, "Weighted support vector regression approach for remote healthcare monitoring," In *2011 International Conference on Recent Trends in Information Technology (ICRTIT),* IEEE, pp. 969-974, 2011.

[19] D. Tomar, and S. Agarwal, "An effective weighted multi-class least squares twin support vector machine for imbalanced data classification," *International Journal of Computational Intelligence Systems*, vol. 8, no. 4, pp. 761-778, 2015.

[20] R. Sangeetha, and B. Kalpana, "Performance evaluation of kernels in multiclass support vector machines," *International Journal of Soft Computing and Engineering*, vol. 1, no. 5, pp. 138-145, 2011.

[21] S. Agarwal, "Classification of countries based on macro-economic variables using fuzzy support vector machine," *International Journal of Computer Applications*, vol. 27, no. 6, 2011.

[22] J. Che, "Support vector regression based on optimal training subset and adaptive particle swarm optimization algorithm," *Applied Soft Computing*, vol. 13, no. 8, pp. 3473-3481, 2013.

[23] C. W. Hsu, and C. J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415-425, 2002.

[24] D. Tomar, R. Arya, and S. Agarwal, "Prediction of profitability of industries using weighted svr," *International Journal on Computer Science and Engineering*, vol. 3, no. 5, pp. 1938-1945, 2011.

[25] U. Patel, "Performance analysis of parallel support vector machines on a mapreduce architecture," Ph.D. dissertation, Wake Forest University, 2016.

[26] M. Lichman, "UCI machine learning repository," 2013. [Online] Available: http://archive.ics.uci.edu/ml

[27] A. Priyadarshini, *et al.*, "A map reduce based support vector machine for big data classification," *International Journal of Database Theory and Application*, vol. 8, no. 5, pp. 77-98, 2015.