Distributed SmSVM Ensemble Learning

Jeff Hajewski and Suely Oliveira

Department of Computer Science
University of Iowa
Iowa City, IA, USA
jeffrey-hajewski@uiowa.edu, suely-oliveira@uiowa.edu

Abstract. Traditional ensemble methods are typically performed with models that are fast to construct and evaluate, such as random trees and Naive Baye's. More complex models frequently suffer from increased computational load in both training and inference. In this work, we present a distributed ensemble method using SmoothSVM, a fast support vector machine (SVM) algorithm. We build and evaluate a large ensemble of SVMs in parallel, with little overhead when compared to a single SVM. The ensemble of SVMs trains in less time than a single SVM while maintaining the same test accuracy and, in some cases, even exhibits improved test accuracy. Our approach also has the added benefit of trivially scaling to much larger systems.

Keywords: Support Vector Machine · Parallel Ensemble Learning · Distributed SVM · SmoothSVM.

1 Introduction

Support Vector Machines (SVMs) are commonly known for their CPU intensive workloads and large memory requirements. In the big data setting, where it is common for data size to exceed available memory, these issues quickly become major bottlenecks. Because of their memory requirements, it is desirable to parallelize the SVM algorithm for large datasets, allowing them to take advantage of greater compute resources and a larger memory pool. In this work, we implement a distributed SmoothSVM (SmSVM) ensemble model. SmSVM [11] is an SVM algorithm that uses a smooth approximation to the hinge-loss function and an active-set approximation to the ℓ_1 norm. It has two major advantages over the standard SVM formulation: 1) the smoothness of the loss function permits use of Newton's method for optimization and 2) the active-set approximation of the ℓ_1 norm smoothes the loss function but not the ℓ_1 penalty, yielding a sparse solution. This reduces the computational requirements and memory footprint.

Efficient communication patterns in large-scale distributed systems is a complex and challenging problem in both design and implementation. In the distributed machine learning setting, unnecessary communication can consume valuable network bandwidth and memory resources with little impact on training time and test accuracy. Conversely, too little communication can result in poorly

performing models due to reduced information transfer from the dataset to the models. We circumvent these issues by scaling the amount of data distributed to each node inversely with the number of worker nodes and use a bootstrap sample of the dataset rather than disjoint subsets. This approach improves the training speed due to improved convergence during optimization while training on fewer data points per model. Our method is able to maintain test accuracy via the improved generalizability of the ensemble models (an artifact of the models being trained on different subsets of the original dataset) while decreasing training time. This work investigates the parallelization problem via a distributed ensemble of SmSVM models, each trained on different, but possibly overlapping, subsets of the original dataset (known as bagging [5]). The advantages of this technique in the distributed machine learning setting are:

- Improved generalization of the aggregated models
- Reduced network utilization via sub-sampling the data
- Reduced training time due to the concurrent construction of SVMs on sub-sampled datasets

We parallelize SmSVM using message passing via MPI, rather than a MapReduce-based framework such as Hadoop or Spark. This approach allows our system to easily scale from running locally on multiple cores to running on a large cluster without requiring reconfiguration. Additionally, it avoids the startup overhead commonly associated with MapReduce-based frameworks (see, for example, [17]). The drawback of this approach is having to handle communication between nodes at a much lower level; however, in our experience this is typically a worthwhile trade-off when performance is the primary goal.

2 Background

In this section we describe the SmSVM algorithm in detail. The standard SVM loss function with both ℓ_1 and ℓ_2 regularizers is shown in (1).

$$\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i\boldsymbol{\omega}\cdot\boldsymbol{x}_i) + \mu||\boldsymbol{\omega}||_1 + \frac{\lambda}{2}||\boldsymbol{\omega}||_2^2 \tag{1}$$

We use a different formulation, called SmoothSVM (SmSVM) [11], which uses a smooth approximation to the hinge-loss and maintains an active-set to approximate the ℓ_1 regularizer. The hinge-loss function, $\psi(\boldsymbol{x}_i, y_i, \boldsymbol{\omega}) = \max(0, 1 - y_i\boldsymbol{\omega}\cdot\boldsymbol{x}_i)$, is non-smooth and thus not globally differentiable. Defining $u = 1 - y\boldsymbol{\omega}\cdot\boldsymbol{x}$, the smoothed hinge-loss is given by equation (2).

$$\psi_\epsilon(u) = \frac{1}{2}(u + \sqrt{\epsilon^2 + u^2}) \tag{2}$$

The choice of ϵ is determined via a relaxation method, where we choose a larger value for ϵ initially, find an optimal solution $\boldsymbol{\omega}$, and then scale ϵ by a relaxation

factor, β. In practice we found $\beta = 100$ to work sufficiently well. Using this definition, and defining $\phi(\boldsymbol{\omega})$ as in (3)

$$\phi(\boldsymbol{\omega}) = \frac{1}{n}\sum_{i=1}^{n}\psi_\epsilon(1 - y_i(\boldsymbol{\omega}\cdot\boldsymbol{x}_i)) + \mu||\boldsymbol{\omega}||_1 + \frac{\lambda}{2}||\boldsymbol{\omega}||_2^2 \tag{3}$$

the problem we aim to solve is given by (4).

$$\min_{\boldsymbol{\omega}} \phi(\boldsymbol{\omega}) \tag{4}$$

The advantage of this formulation is that the optimization problem is smooth, which allows us to use Newton's method, rather than solving the dual problem. Solving (4) is equivalent to solving (5) under the requirement that the Hessian, $H(\boldsymbol{\omega})$, is positive semi-definite.

$$\nabla_{\boldsymbol{\omega}}\phi(\boldsymbol{\omega}) = 0 \tag{5}$$

However, SmSVM does not require the entire Hessian matrix because the active indices of $\boldsymbol{\omega}$ are maintained via the active-set. Thus, we reduce the dimensionality of $H(\boldsymbol{\omega})$ by restricting it to the non-zero components of $\boldsymbol{\omega}$. With this formulation we can use Newton's method [16], which is an iterative solver based on the recurrence relation

$$\boldsymbol{\omega}_{n+1} = \boldsymbol{\omega}_n - f'(\boldsymbol{\omega}_n)^{-1} f(\boldsymbol{\omega}_n)$$

where $f(\boldsymbol{\omega}) = \nabla\phi(\boldsymbol{\omega})$, $f'(\boldsymbol{\omega}) = H(\boldsymbol{\omega})$, and the weight update is given via

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + s\boldsymbol{d}$$

In the general Newton's method implementation we typically have $s = 1$. Computing $H(\boldsymbol{\omega})$ is expensive – we limit the cost of computing $H(\boldsymbol{\omega})$ by using an Armijo linesearch [3] algorithm to reduce the number of required steps. This done by solving (6).

$$\text{argmin}_{s\geq 0}\psi(\boldsymbol{\omega} + s\boldsymbol{d}) + \mu||\boldsymbol{\omega} + s\boldsymbol{d}||_1 \tag{6}$$

Replacing $\psi(\boldsymbol{\omega} + s\boldsymbol{d})$ via a second-order approximation yields

$$\psi(\boldsymbol{\omega} + s\boldsymbol{d}) \approx as^2 + bs + c + \mu||\boldsymbol{\omega} + s\boldsymbol{d}||_1$$

with $s \geq 0$. For $a \geq 0$, $\mu \geq 0$ this is a coercive convex function, which means it has a global minimum. We find this global minimum via binary search. The binary search is computationally efficient because it is with respect to s and thus does not require any additional function or gradient evaluations.

Bootstrap aggregation (commonly referred to as *bagging*) is a statistical technique that improves a model's ability to generalize to unseen data. Bagging consists of a bootstrap phase and an aggregation phase. During the bootstrap phase, data is sub-sampled uniformly with replacement from the original dataset. In this work, we sample a subset that is inversely proportional to the number

of nodes (i.e., for n nodes, and a dataset with cardinality equal to $|D|$, each subset has cardinality equal to $\lfloor |D|/|n| \rfloor$) and train a different model in parallel for each subset. During the inference/aggregation phase, each model generates a prediction for the same data point, the mode of these predictions is computed and used as the final prediction of the bagged model for the given data point. If there are an equal number of votes for each class, the master can choose a class at random or train a model of its own to use as a tie-breaker.

3 Related Work

Lean Yu et al. [22] propose a multiagent SVM ensemble in a sequential compute environment. Their agents are designed to favor diversity by using disjoint subsets from the training set. This approach improves the generalization of their model but inhibits its scalability. As the number of agents grows, the size of the training set per agent decreases, eventually resulting in an agent-based model that has little resemblance to the actual data distribution. Hyun-Chul Kim, et al. [7] propose a sequential, bagged SVM ensemble in the small data setting. Claesen, et al. [8] introduce an ensemble SVM library called `EnsembleSVM` with a focus on sequential efficiency via avoiding data duplication as well as duplicate support vector evaluations. While this library achieves very impressive results, it is designed for computational efficiency in the multicore setting rather than the distributed, big data setting.

3.1 Hadoop and Spark

Distributed SVM architectures are widely studied. Perhaps the most popular approach is a MapReduce-based architecture, using either Hadoop [18] or Spark [23]. A number of implementations use a MapReduce framework via Hadoop[1,2,9,12]. While MapReduce works well for many big data applications, it has two bottlenecks: (i) data is stored on disk during intermediate steps and (ii) a shuffle stage where data is shuffled between servers in a distributed sort These two issues lead to decreased performance when running the iterative style algorithms common to many machine learning algorithms. Specifically, highly iterative algorithms will go through several phases of reading data from disk, processing, writing data to disk, and shuffling *for each iteration*. Other work utilizes the MapReduce framework via Spark[13,15,20,21]. Spark is built on top of Hadoop and attempts to keep its data in memory, which alleviates the disk IO bottleneck. However, Spark can still incur communication overhead during the shuffle stage of MapReduce and, if the data is large enough or there are too few nodes, Spark may spill excessive data to disk (when there is more data than available RAM). While this prior work shows promising results, the use of a MapReduce framework can be sub-optimal. Reyes-Ortiz, Oneto, et al. [17] show that MPI generally outperforms Spark in the distributed SVM setting, seeing as much as a 50× speedup.

3.2 Message passing-based approaches

Chang, Zhu, et al. [6] show that two core bottlenecks in solving the SVM optimization problem (via interior point methods) are the required computational and memory resources. Graf, Cosatto, et al. [10] develop a parallel Incomplete Cholesky Factorization to reduce memory usage, and then solve the dual SVM problem via a parallel interior point method [14]. Although they achieve promising results, their communication overhead scales with the number of nodes. In our approach, communication overhead is constant with respect to the number of nodes.

The advantage of message passing-based approaches over MapReduce-based approaches is the avoidance of the unnecessary disk IO, the shuffle stage, and start-up overhead. Message passing frameworks are typically lower-level than a MapReduce type framework such as Hadoop. The benefit of this lower-level approach is the ability to explicitly control how and when communication occurs, avoiding unnecessary communication. However, the lower-level nature leads to longer development times, more bugs, and generally more complex software.

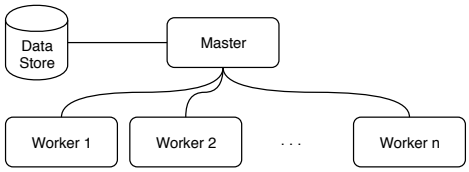


Fig. 1. Overview of system architecture.

4 Distributed Ensemble SmSVM

Distributed ensemble SmSVM is a message-passing based distributed algorithm, detailed in Algorithms 1, 2, 3, and 4. Figure 1 shows an overview of the system architecture.

Algorithm 1 and Algorithm 2 detail the training stage of the distributed ensemble model for the master and worker nodes, respectively. Algorithm 3 and Algorithm 4 detail the inference phase for the master and worker nodes, respectively. The core intuition behind this algorithm is to train a number of different SVM models using the SmSVM algorithm in parallel with bootstrapped subsets of the data and use the resulting models as voters during the inference phase. Consider the case where $n_{\text{workers}} = 5$, after the training phase the master node will have five different weight vectors $\boldsymbol{\omega}$. During the inference phase the master node will (possibly) send a weight vector to each of the worker nodes along with the prediction data, each worker will send its prediction(s) back to the master node, and finally the master node will compute the mode prediction for each data point, using this as the final output. If there are an even number of workers

Algorithm 1 Distributed ensemble training algorithm for master node.

Master Node

Require: $X \in \mathbb{R}^{m \times n}$, $y \in \mathbb{R}^m$ - Training data
Require: n_{nodes} - Number of nodes
for $i = 1$ **to** n_{nodes} **do**
  $\mathcal{I}_i = \{j : j \sim \mathcal{U}(0, m)\}$, with $|\mathcal{I}_i| = m/n_{\text{node}}$
  `send_to`$(X[\mathcal{I}_i, :], y[\mathcal{I}_i], i)$
end for
for $i = 1$ **to** n_{nodes} **do**
  $\boldsymbol{\omega}^{(i)} \leftarrow$ `receive_from`(i)
end for
return $W \in \mathbb{R}^{n_{\text{nodes}} \times n}$

Algorithm 2 Distributed ensemble training algorithm for worker node.

Worker Node

$X, y \leftarrow$ `receive_from`(i_{master})
$\boldsymbol{\omega} \leftarrow \text{SmSVM}(X, y)$
`send_to`$(\boldsymbol{\omega}, i_{\text{master}})$

Algorithm 3 Distributed ensemble inference algorithm for master node.

Master Node

Require: $W \in \mathbb{R}^{n_{\text{nodes}} \times n}$
Require: $X \in \mathbb{R}^{m \times n}$ - Input data (to be classified)
for $i = 1$ **to** n_{nodes} **do**
  $\boldsymbol{\omega} \leftarrow W[i, :]$
  `send_to`(X, i) {Send X to process i}
  `send_to`$(\boldsymbol{\omega}, i)$ {Send $\boldsymbol{\omega}$ to process i}
end for
for $i = 1$ **to** n_{nodes} **do**
  $\mathbf{y}_i \leftarrow$ `receive_from`(i)
end for
results $\leftarrow \text{mode}_i(\mathbf{y}_i)$
return results

Algorithm 4 Distributed ensemble inference algorithm for worker node.

Worker Node

Require: $X \in \mathbb{R}^{m \times n}$
$X \leftarrow$`receive_from`(i_{master})
$\boldsymbol{\omega} \leftarrow$ `receive_from`(i_{master})
for $i = 1$ **to** m **do**
  $z \leftarrow \max\{0, 1 - X_i\boldsymbol{\omega}\}$
  $y_i \leftarrow 1$ **if** $z > 0$ **else** -1
end for
`send_to`$(\mathbf{y}, i_{\text{master}})$

and the results during voting are split, the master may randomly select a class as the final prediction or train its own model (during the training phase) and to use as a tie-breaker. It is not strictly required that the master node uses workers during the inference stage of the algorithm. Specifically, for small enough inference datasets it is more efficient for the master to evaluate the models and voting itself. For large inference datasets, however, it is more efficient for the master to distribute the data and weight vectors to the workers.

The SmSVM algorithm was chosen over other SVM algorithms due to its efficient usage of system resources. A core benefit of the active-set approach to ℓ_1 regularization used by SmSVM is that it allows us to create an optimized implementation of the algorithm. Because the active indices are tracked throughout the computation, we are able to avoid unnecessary multiplications by reducing the data matrix and weight vector to only the active dimensions. This results in reduced memory and CPU usage. The low compute and memory requirements of the SmSVM algorithm allows our distributed ensemble algorithm to run just as effectively on a single many-core machine as it does in a multi-node setting while improving training times.

Table 1. Datasets used in experiments

Dataset	Dimension	Size (GB)	Source
Synthetic	$2,500,000 \times 1,000$	20	N/A
Epsilon	$400,000 \times 2,000$	12	[19]
CoverType	$581,000 \times 54$	0.25	[4]

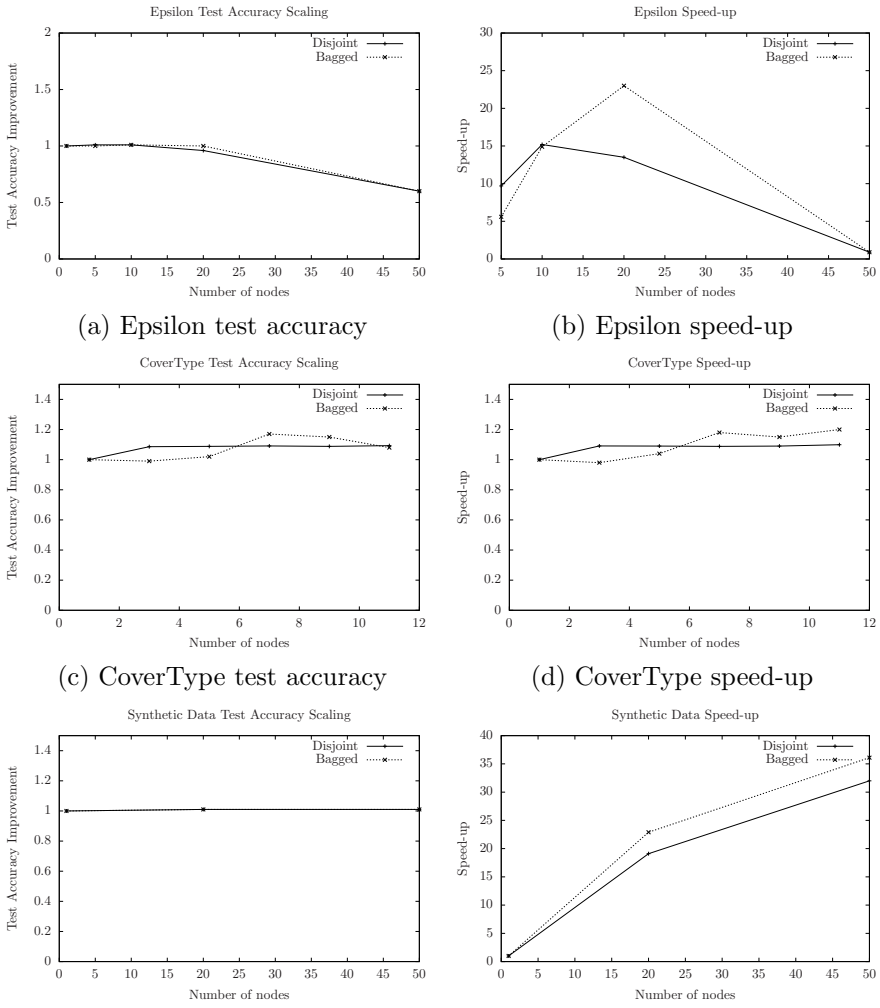
5 Discussion

We evaluate our model by measuring the speed-up scale factor (t_0/t_n) and the test accuracy, as a function of node count, using the three datasets described in Table 1. The test accuracy is evaluated on a hold-out set and the model hyper-parameters are tuned using a validation set. Table 1 lists the datasets used in our experiments. For all experiments, each node is sent a subset of the original dataset D. The cardinality of this subset is equal to $\lfloor |D|/n \rfloor$ where n is the number of worker nodes. For each node count, we run an experiment where the bootstrap step is performed with replacement (referred to as "bagged") and another experiment where the bootstrap step is performed without replacement (referred to as "disjoint").

5.1 Results

Figures 2(a), 2(c), and 2(e) show the change in test accuracy compared to a single node (core) run as a function of the number of nodes used to train the distributed SmSVM model. Figure 2(a) shows a drop-off in test accuracy for

the Epsilon dataset as the number of nodes increases. This is a result of the training subsets containing too little information (due to their small size) to accuractely capture the prior probability distribution – at 50 nodes, each node training the Epsilon dataset gets 8,000 data points while each node under the Synthetic dataset gets 50,000 data points. Figures 2(c) and 2(e), CoverType and Synthetic, respectively, do not exhibit this behavior at the given node counts because there were not enough compute resources (cores or nodes) to create small enough training sets.



(a) Epsilon test accuracy (b) Epsilon speed-up

(c) CoverType test accuracy (d) CoverType speed-up

(e) Synthetic test accuracy (f) Synthetic speed-up

Fig. 2. Scaling and test accuracy results.

Figures 2(b), 2(d), and 2(f) show the training speed-up results for the Epsilon, CoverType, and Synthetic datasets. Figure 2(b) shows the optimal node count for the Epsilon dataset is 20 nodes using the bagged model, achieving about a 24× speed-up compared to a single core. The disjoint model exhibits slower training times for larger node counts due to the reduced training set size. Because the sub-sampled datasets are disjoint, at a certain point the datasets fail to accurately capture the prior distribution of the aggregate dataset, which results in an increased training time. The CoverType results, seen in Figure 2(d) show a similar speed-up behavior to the Epsilon dataset, with the bagged model achieving the best speed-up results. Figure 2(f) shows the most impressive training speed-up results of the three datasets. Due to compute resource constraints, we were only able to run the synthetic dataset on a maximum of 50 nodes. Despite the limitation on resources, the synthetic dataset shows very strong scalability, which is due to information-rich subsets. The synthetic data requires fewer data points than the Epsilon or CoverType datasets to accurately reconstruct its prior distribution. Figure 2(f) clearly shows a decrease in slope from one node to 20 nodes and 20 nodes to 50 nodes, indicating at a larger node count the training speed-up will eventually plateau.

6 Conclusion

In this paper we introduce distributed ensemble SmSVM – a fast, robust distributed SVM model. We take advantage of SmSVM's performance and resource efficiency to achieve significant speed-ups in training time will maintaining test accuracy. These characteristics make our framework suitable for many-core single machines as well as large clusters of machines.

References

1. Alham, N.K., Li, M., Liu, Y., Hammoud, S.: A mapreduce-based distributed svm algorithm for automatic image annotation. Computers & Mathematics with Applications **62**(7), 2801 – 2811 (2011), computers & Mathematics in Natural Computation and Knowledge Discovery
2. Alham, N.K., Li, M., Liu, Y., Qi, M.: A mapreduce-based distributed svm ensemble for scalable image classification and annotation. Computers & Mathematics with Applications **66**(10), 1920 – 1934 (2013)
3. Armijo, L.: Minimization of functions having lipschitz continuous first partial derivatives. Pacific J. Math. **16**(1), 1–3 (1966)
4. Blackard, J.A., Dean, D.J.: Comparative accuracies of neural networks and discriminant analysis in predicting forest cover types from cartographic variables. In: Second Southern Forestry GIS Conference (1998), https://archive.ics.uci.edu/ml/datasets/covertype, taken from UCI Machine Learning Repository
5. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (Aug 1996)
6. Chang, E.Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., Cui, H.: Psvm: Parallelizing support vector machines on distributed computers. In: NIPS (2007)

7. Chen, S., Wang, W., van Zuylen, H.: Construct support vector machine ensemble to detect traffic incident. Expert Syst. Appl. **36**(8), 10976–10986 (Oct 2009)
8. Claesen, M., De Smet, F., Suykens, J.A.K., De Moor, B.: Ensemblesvm: A library for ensemble learning using support vector machines. J. Mach. Learn. Res. **15**(1), 141–145 (Jan 2014)
9. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (Jan 2008)
10. Graf, H.P., Cosatto, E., Bottou, L., Dourdanovic, I., Vapnik, V.: Parallel support vector machines: The cascade svm. In: Saul, L.K., Weiss, Y., Bottou, L. (eds.) Advances in Neural Information Processing Systems 17, pp. 521–528. MIT Press (2005)
11. Hajewski, J., Oliveira, S., Stewart, D.E.: Smoothed hinge loss and l1 support vector machines. In: 2018 Workshop on Optimization Based Techniques for Emerging Data Mining Problems (OEDM). 2018 International Conference on Data Mining (2018)
12. Ke, X., Jin, H., Xie, X., Cao, J.: A distributed svm method based on the iterative mapreduce. In: Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015). pp. 116–119 (Feb 2015)
13. Liu, C., Wu, B., Yang, Y., Guo, Z.: Multiple submodels parallel support vector machine on spark. In: 2016 IEEE International Conference on Big Data (Big Data). pp. 945–950 (Dec 2016)
14. Mehrotra, S.: On the implementation of a primal-dual interior point method. SIAM Journal on Optimization **2**(4), 575 – 601 (1991)
15. Nguyen, T.D., Nguyen, V., Le, T., Phung, D.: Distributed data augmented support vector machine on spark. In: 2016 23rd International Conference on Pattern Recognition (ICPR). pp. 498–503 (Dec 2016)
16. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, New York, NY, USA, second edn. (2006)
17. Reyes-Ortiz, J.L., Oneto, L., Anguita, D.: Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf. Procedia Computer Science **53**, 121 – 130 (2015), iNNS Conference on Big Data 2015 Program San Francisco, CA, USA 8-10 August 2015
18. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). pp. 1–10. MSST '10, IEEE Computer Society (2010)
19. Sonnenburg, S., Franc, V., Yom-Tov, E., Sebag, M.: Pascal large scale learning challenge **10**, 1937–1953 (01 2008)
20. Wang, H., Xiao, Y., Long, Y.: Research of intrusion detection algorithm based on parallel svm on spark. In: 2017 7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC). pp. 153–156 (July 2017)
21. Yan, B., Yang, Z., Ren, Y., Tan, X., Liu, E.: Microblog sentiment classification using parallel svm in apache spark. In: 2017 IEEE International Congress on Big Data (BigData Congress). pp. 282–288 (June 2017)
22. Yu, L., Yue, W., Wang, S., Lai, K.K.: Support vector machine based multiagent ensemble learning for credit risk evaluation. Expert Syst. Appl. **37**(2), 1351–1360 (Mar 2010)
23. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. In: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing. pp. 10–10. HotCloud'10, USENIX Association, Berkeley, CA, USA (2010)