CloudSVM: Training an SVM Classifier in Cloud Computing Systems

F. Ozgur Catak[1] and M. Erdal Balaban[2]

[1] National Research Institute of Electronics and Cryptology (UEKAE), Tubitak
ozgur.catak@tubitak.gov.tr
[2] Quantitative Methods, Istanbul University
mebalaban@gmail.com

Abstract. In conventional distributed machine learning methods, distributed support vector machines (SVM) algorithms are trained over pre-configured intranet/internet environments to find out an optimal classifier. These methods are very complicated and costly for large datasets. Hence, we propose a method that is referred as the Cloud SVM training mechanism (CloudSVM) in a cloud computing environment with MapReduce technique for distributed machine learning applications. Accordingly, (i) SVM algorithm is trained in distributed cloud storage servers that work concurrently; (ii) merge all support vectors in every trained cloud node; and (iii) iterate these two steps until the SVM converges to the optimal classifier function. Single computer is incapable to train SVM algorithm with large scale data sets. The results of this study are important for training of large scale data sets for machine learning applications. We provided that iterative training of splitted data set in cloud computing environment using SVM will converge to a global optimal classifier in finite iteration size.

Keywords: Support Vector Machines, Distributed Computing, Cloud Computing, MapReduce.

1 Introduction

Machine learning applications generally require large amounts of computation time and storage space. Learning algorithms have to be scaled up to handle extremely large data sets. When the training set is large, not all the examples can be loaded into memory in training phase of the machine learning algorithm at one step. They are computationally expensive to process. It is required to distribute computation and memory requirements among several connected computers for scalable learning.

In machine learning field, support vector machines (SVM) offer most robust and accurate classification method due to their generalized properties. With its solid theoretical foundation and also proven effectiveness, SVM has contributed to researchers' success in many fields. But, SVM's suffer from a widely recognized scalability problem in both memory requirement and computational time [1]. SVM algorithm's computation and memory requirements increase rapidly with the number of instances in data set, many data sets are not suitable for classification [14]. The SVM algorithm is formulated as quadratic optimization problem. Quadratic optimization problem has $O(m^3)$ time and $O(m^2)$ space complexity, where m is the training set size [2]. The computation time of SVM training is quadratic in the number of training instances.

The first approach to overcome large scale data set training is to reduce feature vector size. Feature selection and feature transformation methods are basic approaches for reducing vector size [3]. Feature selection algorithms choose a subset of the features from the original feature set and feature transformation algorithms creates new data from the original feature space to a new space with reduced dimensionality. In literature, there are several methods; Singular Value Decomposition (SVD) [4], Principal Component Analysis (PCA) [5], Independent Component Analysis (ICA) [6], Correlation Based Feature Selection (CFS) [7], Sampling based data set selection. All of these methods have a big problem for generalization of final machine learning model. Generalization accuracy of supervised learning algorithms like SVM will increase as the training data set scale increases. The more training data given to SVM algorithm, the more evidence the learning algorithm has about the classification problem. Training data would contain every possible example of classification problem, and then classifier function would generalize perfectly.

Second approach for large scale data set training is chunking [13]. Collobert et al. [12] propose a parallel SVM training algorithm that each subset of whole dataset is trained with SVM and then the classifiers are combined into a final single classifier. Lu et al. [8] proposed distributed support vector machine (DSVM) algorithm that finds support vectors (SVs) on strongly connected networks. Each site within a strongly connected network classifies subsets of training data locally via SVM and passes the calculated SVs to its descendant sites and receives SVs from its ancestor sites and recalculates the SVs and passes them to its descendant sites and so on. Ruping et al. [9] proposed incremental learning with Support Vector Machine. One needs to make an error on the old Support Vectors (which represent the old learning set) more costly than an error on a new example. Syed et al. [10] proposed the distributed support vector machine (DSVM) algorithm that finds SVs locally and processes them altogether in a central processing center. Caragea et al. [11] in 2005 improved this algorithm by allowing the data processing center to send support vectors back to the distributed data source and iteratively achieve the global optimum. Graf et al. [14] had an algorithm that implemented distributed processors into cascade top-down network topology, namely Cascade SVM. The bottom node of the network is the central processing center. The distributed SVM methods in these works converge and increase test accuracy. All of these works have similar problems.

They require a pre-defined network topology and computer size in their network. The performance of training depends on the special network configuration. Main idea of current distributed SVM methods is first data chunking then parallel implementation of SVM training. Global synchronization overheads are not considered in these approaches.

In this paper, we propose a Cloud Computing based novel SVM method with MapReduce [18] technique for distributed training phase of algorithm. By splitting training set over a cloud computing system's data nodes, each subset is optimized iteratively to find out a single global classifier function. The basic idea behind this approach is to collect SVs from every optimized subset of training set at each cloud node, and then merge them to save as global support vectors. Computers in cloud computing system exchange only minimum number of training set samples. Our algorithm CloudSVM is analyzed with various UCI public datasets. CloudSVM is built on the LibSVM and implemented using the Hadoop implementation of MapReduce.

This paper is organized as follows. In section 2, we will provide an overview to SVM formulations. In Section 3, presents the Map Reduce pattern in detail. Section 4 explains system model with our implementation of the Map Reduce pattern for the SVM training. In section 5, convergence of CloudSVM is explained. In section 6, simulation results with various UCI datasets are shown. Thereafter, we will give concluding remarks in Section 7.

2 Support Vector Machine

Support vector machine is a supervised learning method in statistics and computer science, to analyze data and recognize patterns, used for classification and regression analysis. SVM uses machine learning theory to maximize generalization accuracy while automatically avoiding overfit to the training dataset. The standard SVM takes a set of input data and predicts, for each given input, which of two possible classes forms the input, making the SVM a non-probabilistic binary linear classifier. Note that if the training data are linearly separable as shown in Figure 1, we can select the two hyper planes of the margin in a way that there are no points between them and then try to maximize their distance. By using geometry, we find the distance between these two hyper planes is $2/||\boldsymbol{w}||$. Given some training data, D, a set of n points of the form

$$D = \{(\boldsymbol{x}_i, y_i) | \boldsymbol{x}_i \in R^m, y_i \in \{-1,1\}\}_{i=1}^{n} \tag{1}$$

where $\boldsymbol{x}_i$ is an m-dimensional real vector, y_i is either -1 or 1 denoting the class to which point $\boldsymbol{x}_i$ belongs. SVMs aim to search a hyper plane in the Reproducing Kernel Hilbert Space (RKHS) that maximizes the margin between the two classes of

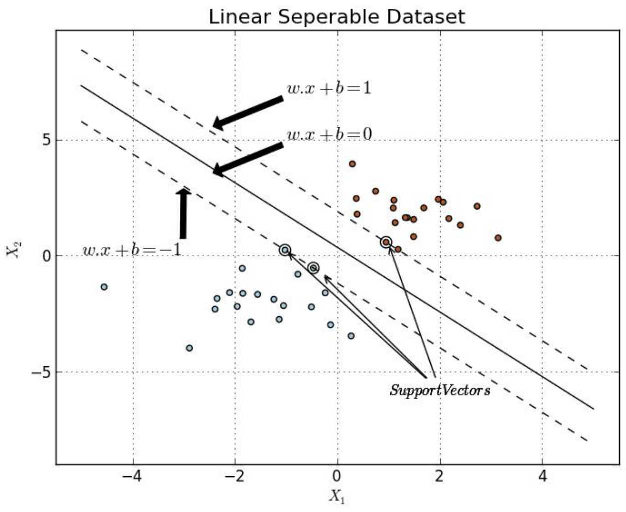


Fig. 1. Binary classification of an SVM with Maximum-margin hyper plane trained with samples from two classes. Samples on the margin are called the support vectors.

data in D with the smallest training error [13]. This problem can be formulated as the following quadratic optimization problem:

$$\begin{gathered} minimize : P(\boldsymbol{w}, b, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ subject\ to : y_i(\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i)\rangle + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{gathered} \tag{2}$$

for $i = 1,\ldots, m$, where ξ_i are slack variables and C is a constant denoting the cost of each slack. C is a trade-off parameter which controls the maximization of the margin and minimizing the training error. The decision function of SVM is $f(\boldsymbol{x}) = \boldsymbol{w}^T\ \phi(\boldsymbol{x}) + b$ where the $\boldsymbol{w}$ and b are obtained by solving the optimization problem P in Equation (2). By using Lagrange multipliers, the optimization problem P in Equation (2) can be expressed as

$$\begin{gathered} min: F(\alpha) = \frac{1}{2}\alpha^T\boldsymbol{Q}\ \alpha^T - \alpha^T\boldsymbol{1} \\ subject\ to : \boldsymbol{0} \leq \alpha \leq \boldsymbol{C} \\ \boldsymbol{y}^T\alpha = 0 \end{gathered} \tag{3}$$

Where $[Q]_{ij} = y_i\, y_j\ \phi^T(x_i)\phi(x_j)$ is the Lagrangian multiplier variable. It is not need to know ϕ, but it is necessary to know is how to compute the modified inner product which will be called as kernel function represented as \$ $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi^T(\boldsymbol{x}_i)\phi(\boldsymbol{x}_j)$. Thus, $[Q]_{ij} = y_i y_j K(x_i, x_j)$. Choosing a positive definite kernel

K, by Mercer's theorem, then optimization problem P is a convex quadratic programming (QP) problem with linear constraints and can be solved in polynomial time.

3 MapReduce

MapReduce is a programming model derived from the map and reduce function combination from functional programming. MapReduce model widely used to run parallel applications for large scale data sets processing. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key [18]. MapReduce is divided into two major phases called map and reduce, separated by an internal shuffle phase of the intermediate results. The framework automatically executes those functions in parallel over any number of processors [19]. Simply, a MapReduce job executes three basic operations on a data set distributed across many shared-nothing cluster nodes. First task is Map function that processes in parallel manner by each node without transferring any data with other notes. In next operation, processed data by Map function is repartitioned across all nodes of the cluster. Lastly, Reduce task is executed in parallel manner by each node with partitioned data.

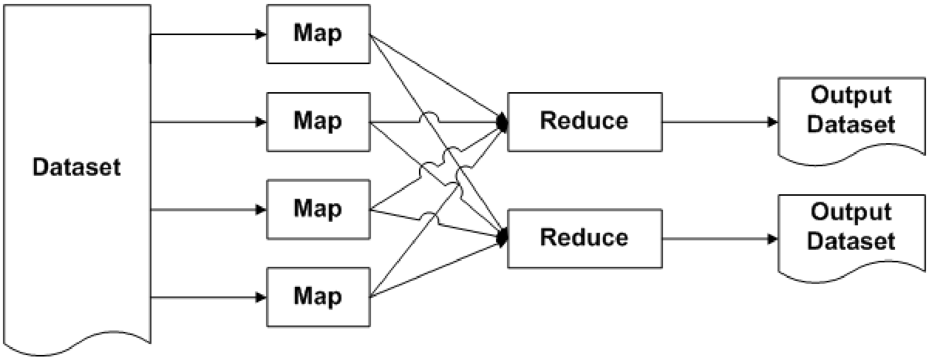


Fig. 2. Overview of MapReduce System

A file in the distributed file system (DFS) is split into multiple chunks and each chunk is stored on different data-nodes. A map function takes a key/value pair as input from input chunks and produces a list of key/value pairs as output. The type of output key and value can be different from input key and value:

$$map(key_1, value_1) \Rightarrow list(key_2, value_2)$$

A reduce function takes a key and associated value list as input and generates a list of new values as output:

$$reduce(key_2, list(value_2)) \Rightarrow list(value_3)$$

Each Reduce call typically produces either one value v_3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list. Main advantage of MapReduce system is that it allows distributed processing of submitted job on the subset of a whole dataset in the network.

4 System Model

CloudSVM is a MapReduce based SVM training algorithm that runs in parallel on multiple commodity computers with Hadoop. As shown in Figure 3, the training set of the algorithm is split into subsets and each one is evaluated individually to get $\boldsymbol{\alpha}$ values (i.e. support vectors). In Map stage of MapReduce job, the subset of training set is combined with global support vectors. In Reduce step, the merged subset of training data is evaluated. The resulting new support vectors are combined with the global support vectors in Reduce step. The CloudSVM with MapReduce algorithm can be explained as follows. First, each computer within a cloud computing system reads the global support vectors, then merges global SVs with subsets of local training data and classifies via SVM. Finally, all the computed SVs in cloud computers are merged. Thus, algorithm saves global SVs with new ones. The algorithm of CloudSVM consists of the following steps.

1. As initialization the global support vector set as $t = 0, V^t = \emptyset$
2. t = t + 1;
3. For any computer in $l, l = 1, \ldots, L$ reads global SVs and merge them with subset of training data.
4. Train SVM algorithm with merged new data set
5. Find out support vectors
6. After all computers in cloud system complete their training phase, merge all calculated SVs and save the result to the global SVs
7. If $h^t = h^{t-1}$ stop, otherwise go to step 2

Pseudo code of CloudSVM Algorithm's Map and Reduce function are given in Algorithm 1 and Algorithm 2.

Algorithm 1 Map Function of CloudSVM Algorithm

```
$SV_{Global} = \emptyset$ //Empty global support vector set
while $h^t \neq h^{t-1}$
    for $l \in L$ do //For each subset loop
            $D_l^t \leftarrow D_l^t \cup SV_{Global}^t$
    end for
end while
```

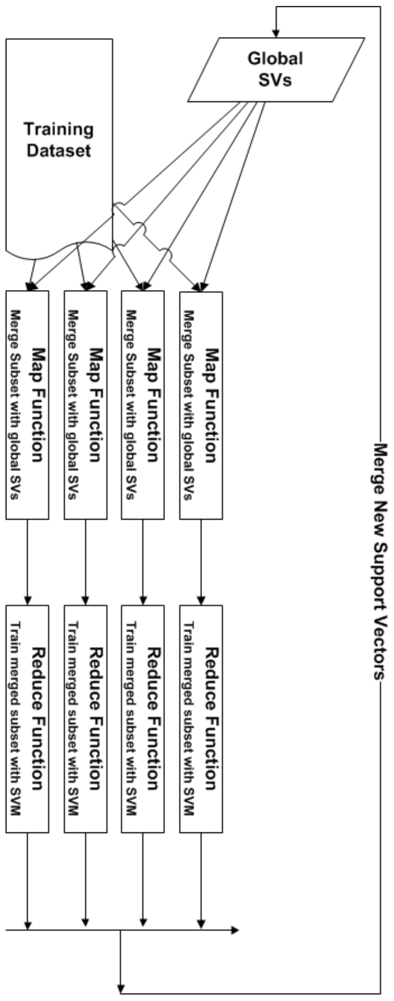

Global SVs
Training Dataset
Map Function Merge Subset with global SVs
Reduce Function Train merged subset with SVM
Merge New Support Vectors

Fig. 3. Schematic of Cloud SVM architecture

Algorithm 2 Reduce Function of CloudSVM Algorithm

```
while $h^t \neq h^{t-1}$ do
    for $l \in \boldsymbol{L}$
        $SV_l, h^t \leftarrow svm(D_l)$ // Train merged Dataset to obtain
Support Vectors and Hypothesis
    end for
    for $l \in \boldsymbol{L}$
        $SV_{Global} \leftarrow SV_{Global} \cup SV_l$
    end for
end while
```

For training SVM classifier functions, we used LibSVM with various kernels. Appropriate parameters C and γ values were found by cross validation test. We used 10-fold cross validation method. All system is implemented with Hadoop and streaming Python package mrjob library.

5 Convergence of CloudSVM

Let S denotes a subset of training set D, $F(S)$ is the optimal objective function over data set S, h^* is the global optimal hypothesis for which has a minimal empirical risk $R_{emp}(h)$. Our algorithm starts with $\boldsymbol{SV}_{Global}^{0} = \boldsymbol{0}$, and generates a non-increasing sequence of positive set of vectors $\boldsymbol{SV}_{Global}^{t}$, where $\boldsymbol{SV}_{Global}^{t}$ is the vector of support vector at the t.th iteration. We used hinge loss for testing our models trained with CloudSVM algorithm. Hinge loss works well for its purposes in SVM as a classifier, since the more you violate the margin, the higher the penalty is [20]. The hinge loss function is the following:

$$l(f(x), y) = max\,\{0{,}1 - y.f(x)\}$$

Empirical risk can be computed with an approximation:

$$R_{emp}(h) = \frac{1}{n}\sum_{i=1}^{n}\big(l(h(x_i), y_i)\big)$$

According to the empirical risk minimization principle the learning algorithm should choose a hypothesis $\hat{h}$ which minimizes the empirical risk:

$$\hat{h} = \arg\max_{h \in H} R_{emp}(h)$$

A hypothesis is found in every cloud node. Let X be a subset of training data at cloud node i where $X \in R^{mxn}$, $\boldsymbol{SV}_{Global}^{t}$ is the vector of support vector at the t. th iteration, $h^{t,i}$ is hypothesis at node i with iteration t, then the optimization problem in equation 3 becomes

$$\begin{aligned} &maximize\ h^{t,l} = -\frac{1}{2}\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix}^T \begin{bmatrix}\boldsymbol{Q}_{11} & \boldsymbol{Q}_{12}\\ \boldsymbol{Q}_{21} & \boldsymbol{Q}_{22}\end{bmatrix} \begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix} + \begin{bmatrix}1\\ 1\end{bmatrix}^T \begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix} \\ &subject\ to\ : 0 \le \alpha_i \le C, \forall i\ and \sum_{i}^{I} \alpha_i y_i = 0 \end{aligned} \tag{4}$$

where $\boldsymbol{Q}_{12}$ and $\boldsymbol{Q}_{21}$ are kernel matrices with respect to

$$Q_{12} = \left\{ K_{i,j}\big(x_{ij}, SV_{Global(i,j)}^{t}\big) \middle|\ i = 1, \dots, m, j = 1, \dots n \right\}.$$

α_1 and α_2 are the solutions estimated by node i with dataset X and $\boldsymbol{SV}_{Global}$. Because of the Mercer's theorem, our kernel matrix $\boldsymbol{Q}$ is a symmetric positive-definite function on a square. Then our sub matrices $\boldsymbol{Q}_{12}$ and $\boldsymbol{Q}_{21}$ must be equal. We can define $\boldsymbol{Q}_{11}$ and $\boldsymbol{Q}_{22}$ matrices such that

$$\boldsymbol{Q}_{11} = \{ K_{i,j}(x_{i,j}, x_{i,j}) | x_{i,j} \in X, i = 1, \dots, m, j = 1, \dots, n \}$$
$$\boldsymbol{Q}_{22} = \{ K_{i,j}(SV_{Global}, SV_{Global}) | i = 1, \dots, m, j = 1, \dots, n \}$$

at iteration t.

Algorithm's stop point is reached when the hypothesis' empirical risk is same with previous iteration. That is:

$$R_{emp}(h^t) = R_{emp}(h^{t-1}) \tag{5}$$

Lemma : Accuracy of the decision function of CloudSVM classifier at iteration t is always greater or equal to the maximum accuracy of the decision function of SVM classifier at iteration $t-1$. That is

$$R_{emp}(h^t) \leq arg \min_{h \in H^{t-1}} R_{emp}(h) \tag{6}$$

Proof: Without loss of generality, Iterated CloudSVM monotonically converges to optimum classifier.

$$\boldsymbol{SV}^t_{Global} = \boldsymbol{SV}^{t-1}_{Global} \cup \{ \boldsymbol{SV}^{t-1}_i \mid i = 1, \dots n \}$$

where n is the data set split size(or cloud node size). Then, training set for svm algorithm at node i is

$$d = X \cup \boldsymbol{SV}^t_{Global}$$

Adding more samples cannot decrease the optimal value. Generalization accuracy of the sub problem in each node monotonically increases in each step.

6 Simulation Results

We have selected several data sets from the UCI Machine Learning Repository, namely, German, Heart, Ionosphere, Hand Digit and Satellite. The data sets length and input dimensions are shown in Table 1. We test our algorithm over a real-word data sets to demonstrate the convergence. Linear kernels were used with optimal parameters (γ, C). Parameters were estimated by cross-validation method.

Table 1. The datasets used in experiments

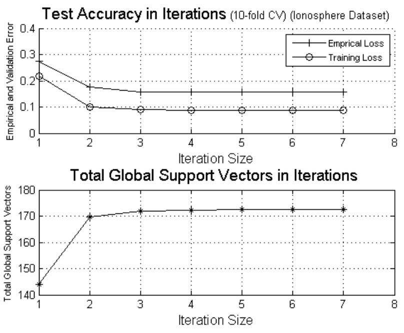
Dataset Name	Train. Data	Dim.
German	1000	24
Heart	270	13
Ionosphere	351	34
Satellite	4435	36

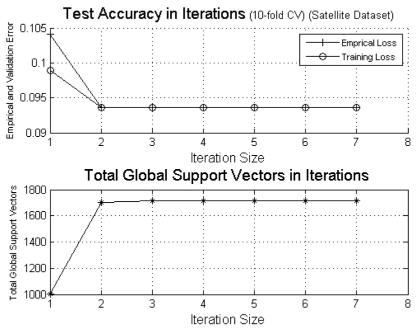
We used 10-fold cross-validation, dividing the set of samples at random into 10 approximately equal-size parts. The 10 parts were roughly balanced, ensuring that the classes were distributed uniformly to each of the 10 parts. Ten-fold cross-validation works as follows: we fit the model on 90% of the samples and then predict the class labels of the remaining 10% (the test samples). This procedure is repeated 10 times, with each part playing the role of the test samples and the errors on all 10 parts added together to compute the overall error.

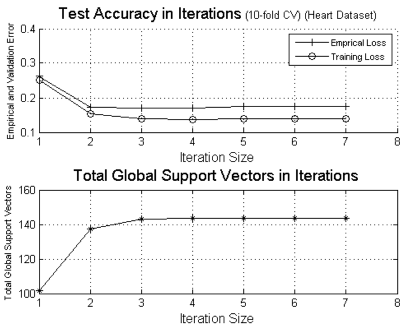
Table 2. Performance Results of CloudSVM algorithm with various UCI datasets γ

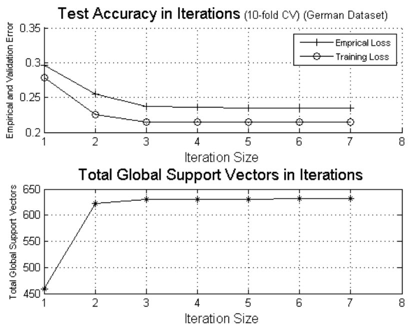
Dataset Name	γ	C	No. Of Iteration	No. Of SVs	Accuracy	Kernel Type
German	10^0	1	5	606	0.7728	Linear
Heart	10^0	1	3	137	0.8259	Linear
Ionosphere	10^8	1	3	160	0.8423	Linear
Satellite	10^0	1	2	1384	0.9064	Linear

Table 3. Data set prediction accuracy with iterations









To analyse the CloudSVM, we randomly distributed all the training data to a cloud computing system with 10 computers with pseudo distributed Hadoop. Data set prediction accuracy with iterations and total number of SVs are shown in Table 3. When iteration size become 3 - 5, test accuracy values of all data sets reach to the highest values. If the iteration size is increased, the value of test accuracy falls into a steady state. The value of test accuracy is not changed for large enough number of iteration size. As a result, the CloudSVM algorithm is useful for large size training data.

7 Conclusion and Further Research

We have proposed distributed support vector machine implementation in cloud computing systems with MapReduce technique that improves scalability and parallelism of split data set training. The performance and generalization property of our algorithm are evaluated in Hadoop. Our algorithm is able to work on cloud computing systems without knowing how many computers connected to run parallel. SVM algorithm's training problem for large scale data set is solved with the designed algorithm referred to as CloudSVM. It is empirically shown that the generalization performance and the risk minimization of our algorithm are better than the previous results.

References

1. Chang, E.Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., Cui, H.: PSVM: Parallelizing Support Vector Machines on Distributed Computers. In: Advances in Neural Information Processing Systems, vol. 20 (2007)
2. Tsang, I.W., Kwok, J.T., Cheung, P.M.: Core Vector Machines: Fast SVM Training on Very Large Data Sets. J. Mach. Learn. Res. 6, 363–392 (2005)
3. Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., Vapnik, V.: Feature selection for SVMs. In: Advances in Neural Information Processing Systems, vol. 13, pp. 668–674 (2000)
4. Golub, G., Reinsch, C.E.: Singular value decomposition and least squares solutions. Numerische Mathematik 14, 403–420 (1970)
5. Jolliffe, I.T.: Principal Component Analysis, 2nd edn., New York. Springer Series in Statistics (2002)
6. Comon, P.: Independent Component Analysis, a new concept? Signal Processing 36, 287–314 (1994)
7. Hall, M.A.: Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In: Proceedings of the Seventeenth International Conference on Machine Learning, pp. 359–366. Morgan Kaufmann Publishers Inc., San Francisco (2000)
8. Lu, Y., Roychowdhury, V., Vandenberghe, L.: Distributed parallel support vector machines in strongly connected networks. IEEE Trans. Neural Networks 19, 1167–1178 (2008)

9. Stefan, R.: Incremental Learning with Support Vector Machines. In: IEEE International Conference on Data Mining, p. 641. IEEE Computer Society, Los Alamitos (2001)
10. Syed, N.A., Liu, H., Sung, K.: Incremental learning with support vector machines. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), San Diego, California (1999)
11. Caragea, C., Caragea, D., Honavar, V.: Learning support vector machine classifiers from distributed data sources. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI), Student Abstract and Poster Program, pp. 1602–1603. AAAI Press, Pittsburgh (2005)
12. Collobert, R., Bengio, S., Bengio, Y.: A parallel mixture of SVMs for very large scale problems. Neural Computation 14, 1105–1114 (2002)
13. Vapnik, V.N.: The nature of statistical learning theory. Springer, NY (1995)
14. Graf, H.P., Cosatto, E., Bottou, L., Durdanovic, I., Vapnik, V.: Parallel support vector machines: The cascade SVM. In: Proceedings of the Eighteenth Annual Conference on Neural Information Processing Systems (NIPS), pp. 521–528. MIT Press, Vancouver (2004)
15. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27–27 (2011)
16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324 (1998)
17. Bertsekas, D.P.: Nonlinear Programming, 2nd edn. Athena Scientific, Cambridge (1999)
18. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation(OSDI), p. 10. USENIX Association, Berkeley (2004)
19. Schatz, M.C.: CloudBurst: highly sensitive read mapping with MapReduce. Bioinformatics 25, 1363–1369 (2009)
20. Rosasco, L., De Vito, E., Caponnetto, A., Piana, M., Verri, A.: Are loss functions all the same. Neural Computation 16, 1063–1076 (2011)