A Distributed SVM for Image Annotation

Nasullah Khalid Alham, Maozhen Li, Suhel Hammoud, Yang Liu, Mahesh Ponraj
School of Engineering and Design
Brunel University, Uxbridge, UB8 3PH, UK
Nasullah.KhalidAlham@brunel.ac.uk

***Abstract*— The popularity of SVMs has grown tremendously in the last few years for many different classification problems due to its generalization properties, however training SVMs require high computational power. Platt's SMO is one the fastest algorithm for training support vector machines, which takes the decomposition technique to the extreme by selecting a set of only two points as the working set then solving them analytically. However SMO becomes slow for large size training data set. In this paper we present a MapReduce based distributed implementation of SMO using Hadoop. The distributed SMO uses multiple core processors to process the training data. By partitioning the training data set into smaller subsets and allocating each of the partitioned subsets to a single Map task, each Map task optimizes the partition in parallel and finally the reducer combine the results. Experiments show the efficiency of the distributed SMO increases with the increase of the number of processors, the training speed of distributed SMO with 12 Map task is about 11times higher than standalone SMO. There is no significant difference in accuracy between distributed and standalone SMO.**

Keywords-image anotation; machine learning; SMO; mapReduce; distributed SVM

I. INTRODUCTION

Recently Support Vector Machines (SVMs) have gained popularity for many different classification problems due to its generalization properties that is the ability of a classifier to correctly classify data that are involved in the training process. Currently SVMs are one of the most widely used classifier in image classification. In our previous paper [1] we evaluated the performances of the most representative classifiers in image classification in terms of accuracy and training overheads. The evaluation results show SVMs perform better than other classifiers in term of accuracy, however the training time of the classifier is longer than other classifier especially with larger dataset. The evaluation results confirm that SVM models are too large to be used in a practical hence the speed of training is lower.

The main problem in using SVMs is that building SVMs require solving a constrained quadratic programming problem which means its size is quadratic in the number of training examples [13]. This problem has led researcher to find ways to speed up the training of SVMs. Recently, a number of researchers have suggested using multiple computing nodes in order to increase the computational power available for solving SVMs [2]; that is to obtain results which are as accurate as standalone algorithms, while providing fast convergence and good scalability ranging from several computers in a cluster to many hundreds of computing nodes. In recent times, there has been some affords on distributed implementation of training SVMs [2]. Usually by partitioning large training data set into smaller parts, then process each parts in parallel and finally combine the results. It has been shown that parallel algorithms can provide much more efficiency than standalone algorithm for training SVM.

There are a number of algorithms available for training SVMs. Sequential minimal optimization (SMO) is one of the fastest and most popular algorithms for SVMs; however it still requires a large amount of computation time for solving large size problems [3]. In this paper we present a distributed implementation of SMO using Hadoop[4] an open-source implementation of Google's distributed file system[5] and the MapReduce framework [6] for training SVMs which deals with very large size training data with minimum network communication cost. The distributed SMO uses multiple core processors to process the training data. By partitioning the training data set into smaller subsets and allocating each of the partitioned subsets to a single Map task; each Map task optimizes the partition in parallel and finally the reducer combine the partial results. Experiments show the efficiency of the distributed SMO increases with the increase of the number of processors, the speed of training SVM with 12 Map tasks is about 11times higher than standalone SMO.

II. RELATED WORKS

Bickson et al [2] introduced a parallel implementation of an SVM solver using Message Passing Interface (MPI). Their parallel solution can be used in peer-to-peer and grid environments, where there is no central authority that allocates the work. However their implementation of the algorithm using a synchronous communication model due to MPI's lack of support for asynchronous communication which could affect the speed of training time. Collobert et al. [7] proposed a parallel SVM training algorithm in which training multiple SVMs were performed using subsets of the training data, and then combined the classifiers into a final single classifier. The training data is then reallocated to the classifiers based on their performance and the process is iterated until convergence is reached. However the frequent reallocation of training data during the optimization process may cause a reduction in the training speed. Zanghirati et al. [8] introduced a parallel implementation of SVM solver using MPI based on a decomposition technique that splits the problem into smaller quadratic programming sub problems. The outcomes of each

978-1-4244-5934-6/10/$26.00 ©2010 IEEE

sub problems are combined. However the performance of the parallel implementation is heavily depended on which caching strategy is used to avoid re-computation of previously used elements. Cao et al [3] proposes a parallel implementation of the modified SMO to speeding up the training of SVM with large size training data. They developed parallel SMO using MPI. They consider the KKT condition updates as the major source of parallelism. The computation of updating f_i array at each of SMO iteration which includes the kernel evaluations and is also required for every training data pattern performed in a parallel way. Their experiments show an increase in the speed of training SVM. Although MPI is a message-passing scheme designed to enable heterogeneous implementations and virtual communication channels communicate across different architectures, however MPI does not provide support for active messages, multithreading, and fault tolerance [9]. Catanzaro et al [10] describe a parallel SMO algorithm using MapReduce in graphic processor platform. Their approach is similar to Cao et al [3]; the computation of f_i is the map function, and the search for b_{low}, b_{up}, I_{low} and I_{up} is the reduction operation, which means that there is a map and reduce phase for each of the SMO algorithm's iteration. This solution might be useful for some training data. However usually there are a very large number of iterations involved in the optimization process of SMO especially with larger data set. This can cause a major overhead because launching MapReduce operation for each of the algorithm iteration is time consuming. Kun et al [11] implemented parallel SMO using Cilk [15] and Java threads. The idea is to partition the training data into more than one part, train these parts in parallel, and perform a combination of results. Cilk's main disadvantage is that it requires a shared-memory machine [16].

In contrast to the exiting solutions, our approach is simple, efficient and scalable that is using a single Map Reduce phase to distribute SMO algorithm with minimal data movement between nodes and communication overheads. Training data are partitioned into a number of parts, each partition is trained by a single Map task and the reducer combines the partial results. The affect of distributing SMO algorithm on the accuracy of classification is insignificant.

III. SUPPORT VECTOR MACHINES

The basic idea of SVM is to create a hyperplane as the decision plane, which separates the positive (+1) and negative (-1) classes with the largest margin. An optimal hyperplane is the one with the maximum margin of separation between the two classes, where the margin is the sum of the distances from the hyperplane to the closest data points of each of the two classes. These closest data points to hyperplalne are called Support Vectors (SVs) [20]. Given a set of training data D, a set of points of the type,

$$D = \{(x_i, c_i) \mid x_i \in \Re^p, c_i \in \{-1,1\}\}^n_{i=1}, \quad (1)$$

where c_i is either 1 or -1 indicates the class to which point x_i belongs. The aim is to give a maximum margin hyperplane which divide points having $c_i = 1$ from those having ci = -1.

Any hyperplane can be constructed as a set of point x satisfying

$$w.x - b = 0. \quad (2)$$

The vector w is a normal vector. We want to choose the w and threshed b to maximize the margin. The margin m is

$$m = 1/\|w\|_2 \quad (3)$$

The dual of the SVM can be shown to be the following optimization problem:

Maximize (in α_i)

$$\sum_{i=1}^{n} \alpha_i - 1/2 \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j, \quad (4)$$

Subject to $\alpha_i > 0, \quad \sum_{i=1}^{n} \alpha_i y_i = 0.$

There is a one-to-one association between each Lagrange multiplier a and each training example. Once the Lagrange multipliers are determined, the normal vector $\vec{w}$ and the threshold b can be derived from the Lagrange multipliers [13]:

$$\vec{w} = \sum_{i=1}^{n} y_i a_i \vec{x}_i. \quad (5)$$

$$b = \vec{w}.\vec{x}_k - y_k. \quad (6)$$

For some $a_k > 0$

Not all data sets are linearly separable. There may be no hyperplane exist that separate the training examples with positive signs from the examples with negative signs. SVMs can be further generalized to non-linear classifiers. The output of a non-linear SVM is computed from the Lagrange multipliers:

$$u = \sum_{i=1}^{n} y_i a_i K(X_i, X) + b. \quad (7)$$

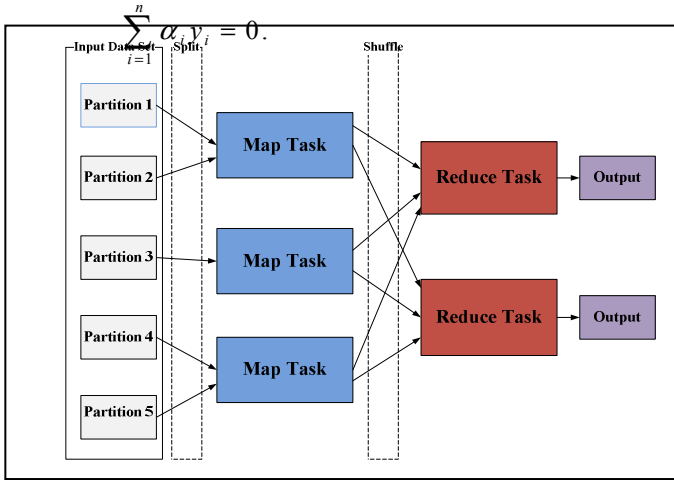
where K is a kernel function that measures the similarity or distance between the input vector $\vec{x}$ and the stored training vector $\vec{x}_i$.

A. Sequential Minimal Opimization (SMO)

The SMO algorithm was developed by Platt [18], enhanced by Keerthi [20], and based on Osuna [19]. Platt takes the decomposition to the extreme by selecting a set of only two points as the working set which is the minimum due to the following condition;

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \quad (8)$$

This allows the sub



problems to have analytical solution. Despite the need for more iterations to converge, each iteration uses so few operations that the algorithm shows an overall speed-up of some orders of magnitude [21]. Platt's idea provides improvement in efficiency, and the SMO is now one of the fastest SVM algorithms available. SMO optimize two a_i related with b_{up} and b_{low} according to the followings;

$$a_2^{new} = a_2^{old} - y_2\left(f_1^{old} - f_2^{old}\right)/\eta, \quad (9)$$

$$a_1^{new} = a_1^{old} - s\left(a_2^{old} - a_2^{new}\right), \quad (10)$$

where $\eta = 2k(X_1, X_2) - k(X_1, X1) - k(X_2, X_2)$. After optimizing two a_i, f_i which denote the error of ith training data updated according to the following;

$$f_i^{new} = f_i^{old} + (a_1^{new} - a_1^{old}) y_1 k(X_1, X_i) + (a_2^{new} - a_2^{old}) y_2 k(X_2, X_i). \quad (11)$$

To build a linear SVM, a single weight vector needs to be stored instead of all the training examples that correspond to non-zero Lagrange multipliers. If the joint optimization is successful, the stored weight vector needs to be updated to reflect the new Lagrange multiplier values. The weight vector is updated according to the following:

$$\vec{w}^{new} = \vec{w} + y_1(a_1^{new} - a)\vec{x}_1 + y_2(a_2^{new,clipped} - a_2)\vec{x}. \quad (12)$$

We check the optimality of the solution by calculating the optimality gap that is the gap between the b_{low} and b_{up}. The algorithm is terminated when $b_{low} \le b_{up} + 2\tau$.

IV. DISTRIBUTED SMO

We start this section by briefly describing the MapReduce programming model followed by a detailed description of our proposed distributed SMO.

A. MapReduce Model

Applications frequently require more resources than are available on an inexpensive machine [17]. The need for efficient and effective models of parallel and grid computing is clear due the existences very large data sets which cannot be processed without using multiple computing nodes. Google's MapReduce programming model provides an efficient framework for processing large data sets in an extremely parallel manner [6]. The Google File System (GFS) that underlie a MapReduce provide efficient and reliable distributed data storage required for applications involving large data sets [17]. The basic function of MapReduce model is to iterate over the input, compute key/value pairs from each part of input, group all intermediate values by key, then iterate over the resulting groups and finally reduce each group. The programmer can abstract from the issues of distributed and parallel programming. MapReduce implementation deals with issues such as load balancing, network performance, fault tolerance etc [6]. The Apache Hadoop [4] project is an open-source implementation of Google's MapReduce writing in java for reliable, scalable, distributed computing.

Figure 1: Workflow of MapReduce model

The MapReduce model is based on two separate steps in an application [17]:

- *Map*: An initial transformation step, in which individual input records are processed in parallel.
- *Reduce*: A summarization step, in which all associated records are processed together by a single entity.

Figure 1 shows the basic model of MapReduce in which the input is split into logical chunks, and each chunk is processed independently, by a map task. The results of these processing chunks can be physically partitioned into separate sets, which are then sorted. Each sorted chunk is passed to a reduce task. HDFS is a file system is designed for MapReduce jobs that read input in large chunks of inputs, process it, and write chunks of output. For reliability, file data is mirrored to multiple storage nodes [17]. HDFS services are provided by two processes:

- *NameNode*: Controls the file system metadata, and provides management and control services.
- *DataNode*: Offers block storage and retrieval services.

B. Distributed Algorithm

We have implemented distributed SMO using Hadoop and the WEKA package [12]. The basic idea of our distributed implementation is similar to Kun et al [11] idea. Caching scheme is used to error cache the current predicted output and update each successful step also Kernel cache is used for Kernel value between 2 points which is used during error cache updates. The algorithm partitioning the entire training data set into smaller subsets *m* partition and allocating each of the partitioned is allocated to a single Map task; number of Map tasks is equal to the number data partitions. Each Map task optimizes a partition in parallel. The output of each Map task is the alpha array for local partition and the value of *b*. Reducer simply joins the partial alpha arrays to produce the global alpha array. The reducer has to deal with the value of *b* because this value is different for each partition therefore the reducer takes the average value of *b* for all partitions to obtain global *b*. Multi-class classification with MapReduce based distributed SMO was performed by one against all technique [13].

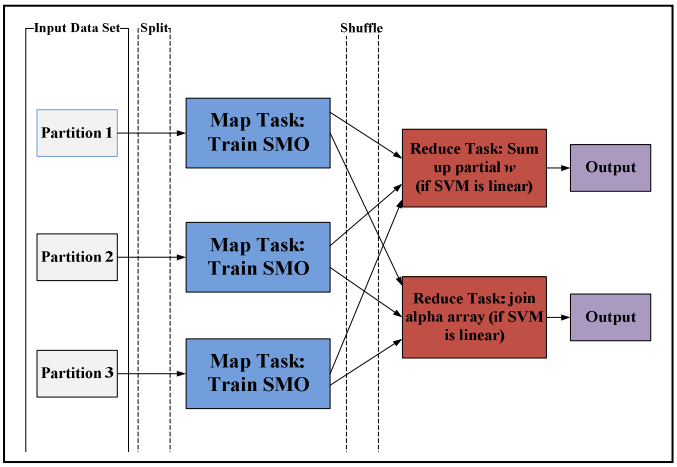


Figure 2: The MapReduce based SMO Architecture.

V. EXPERIMENTS

In order to test the efficacy of our distributed SMO algorithm, we have built an image annotation system to compare the performances of Distributed SMO with standalone SMO. The image annotation systems classify visual features into pre-defined classes. First images are segmented into blocks. Then, the low-level features are extracted from the segmented images. Each segmented block is represented by feature vectors. Next stage is to assign the low-level feature vectors to pre-defined categories. The system learns the correspondence between low level visual features and image labels. Combination of Low-level MPEG-7 descriptors such as scalable color [14] and edge histogram [14] are used. The classifier is fed with a set of training images in the form of attribute vectors with the associated labels After a model is trained, it is able to classify an unknown instance, into one of the learned class labels in the training set. Figure 3 shows user interface of the prototype system which supports automatic annotation of images using standalone and distributed SMO.

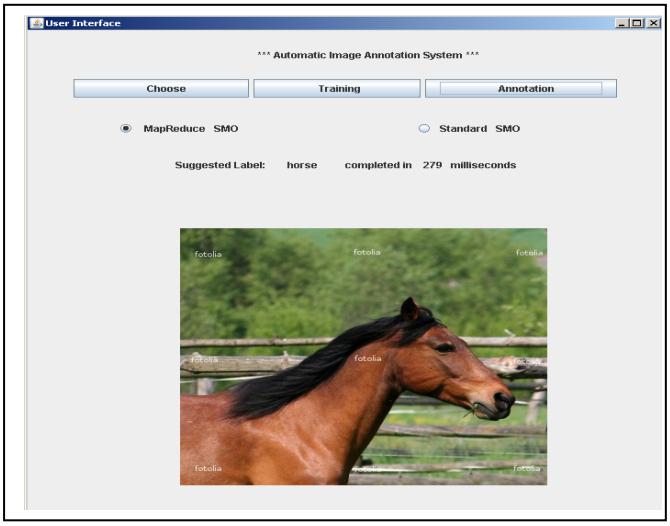


Figure 3: A snapshot of the image annotation system.

A. Speed

A number of tests were carried out on 3 computers, Linux Fedora 11, RAM- 3.00 GB, CPU Q6600. The experiment shows that the efficiency of the distributed SMO increases with the increase of the number of processors, the speed of training SVM with 12 Map tasks is 11times higher than standalone SMO. Figure 4 shows the comparison of training times of SMO and distributed SMO in binary classification.

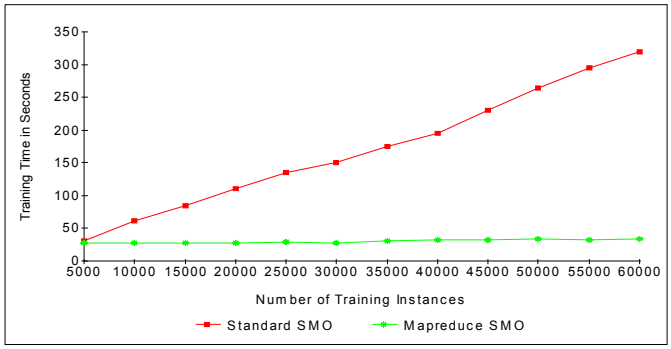


Figure 4: Comparison of training times of SMO and Distributed SMO in binary classification.

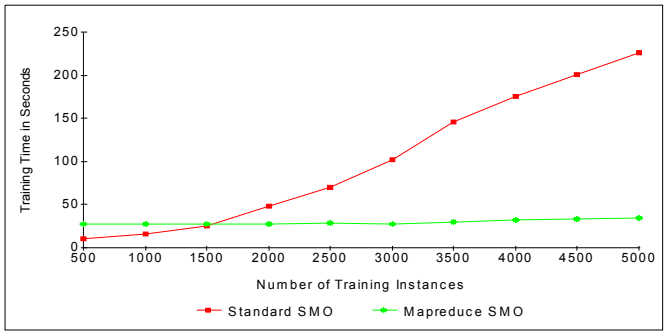


Figure 5: Comparison of training times of SMO and Distributed SMO in multiclass classification.

B. Accuracy

Distributed SMO algorithm was evaluated from the aspects of accuracy in annotating images. In total 50 unlabeled images were tested (10 images at a time), the average accuracy level was considered. Figure 6 shows comparison of accuracy level of the distributed SMO and standalone SMO algorithms in binary and multiclass classification, it clear that the parallelization of SMO has not affected the accuracy level significantly. We have compared the accuracy level of SMO and Distributed SMO in binary classification; we found out that parallel SMO had 93% accuracy level compare to 94% of standalone SMO. The differences between the generalization performances achieved by the standalone SMO and the proposed MapReduce based distributed SMO are very small and could be neglected in many cases.

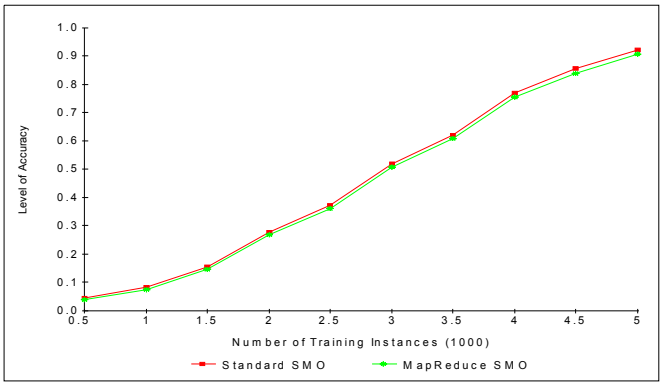


Figure 6: Comparison of accuracy level of SMO and distributed SMO in multiclass classification.

VI. CONLUSION AND FUTURE WORK

In this paper we proposed a distributed implementation of SMO based on MapReduce framework. The distributed SMO uses multiple processors in a cluster to speed up SMO algorithm for training SVM with very large size data set. By partitioning the training data set into smaller subsets and allocating each of the partitioned to a single Map task; each Map task optimizes the partition in parallel. The experiments show the efficiency of the distributed SMO increases with the increase of the number of processors. Currently we are working on the implementation of a MapReduce simulator to accurately simulate the Hadoop environment. The simulator on one hand will allow us to measure the scalability of our MapReduced based SMO easily and quickly, on the other hand will capture the effects of different configurations of Hadoop setup on the algorithm's behavior in terms of speed. As part of future research work we are planning to introduce an advanced load balancing technique targeting the Hadoop environment in order to achieve optimal resource utilization hence shorter Hadoop job completion time.

REFERENCES

[1] N. Khalid Alham, M. Li, S. Hammoud and H. Qi, Evaluating Machine Learning Techniques for Automatic Image Annotations, Proc. of FSKD09, IEEE CS, Aug. 2009

[2] D. Bickson, E. Yom-Tov, and D. Dolev, "A Gaussian Belief Propagation Solver for Large Scale Support Vector Machines," 2008.

[3] L. Cao, S. Keerthi, Chong-Jin Ong, J. Zhang, U. Periyathamby, Xiu Ju Fu, and H. Lee, "Parallel sequential minimal optimization for the training of support vector machines," *Neural Networks, IEEE Transactions on*, vol. 17, 2006, pp. 1039-1049.

[4] Apache Hadoop! Available at: http://hadoop.apache.org/ [Accessed November 2, 2009

[5] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, 2003, pp. 29-43.

[6] R. Lämmel, "Google's MapReduce programming model — Revisited," *Sci. Comput. Program.*, vol. 68, 2007, pp. 208-237.

[7] R. Collobert, S. Bengio, and Y. Bengio, "A Parallel Mixture of SVMs for Very Large Scale Problems," 2002.

[8] G. Zanghirati and L. Zanni, "A parallel solver for large quadratic programs in training support vector machines," *Parallel Comput.*, vol. 29, 2003, pp. 535-551.

[9] L. Smith, M. Bull, Development of mixed mode MPI / OpenMP applications, IOS Press Amsterdam, The Netherlands, The Netherlands.

[10] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," *Proceedings of the 25th international conference on Machine learning*, Helsinki, Finland: ACM, 2008, pp. 104-111.

[11] D. Kun. L. Yih. A. Perera, Parallel SMO for Training Support Vector Machines. SMA 5505 Project Final Report. May 2003. Available at www.geocities.com/asankha/Contents/Education/APC-SVM.pdf [accessed June 29, 2009]

[12] Weka 3 - Data Mining with Open Source Machine Learning Software in Java. Available at: http://www.cs.waikato.ac.nz/ml/weka/ [Accessed November 2, 2009].

[13] S. Abe, *Support Vector Machines for Pattern Classification (Advances in Pattern Recognition)*, Springer-Verlag New York, Inc., 2005.

[14] T. Sikora: The MPEG-7 visual standard for content description-an overview. IEEE Trans. Circuits Syst. Video Techn. (2001) 11(6): 696-702

[15] The Cilk Project. Available at: http://supertech.csail.mit.edu/cilk/ [Accessed November 29, 2009].

[16] M. Frigo, C. Leiserson, K. Randall, The Implementation of the Cilk-5 Multithreaded Language. *Proceedings of the SIGPLAN '98 Conference on Programming Language Design and Implementation,* pp. 212-223, Montreal, Quebec, Canada.

[17] J. Venner, *Pro Hadoop*, Springer 2009, pp 1-89.

[18] J.C. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," 1998.

[19] E. Osuna, R. Freund, and F. Girosit, "Training support vector machines: an application to face detection," *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, 1997, pp. 130-136.

[20] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy, "Improvements to Platt's SMO Algorithm for SVM Classifier Design," *Neural Comput.*, vol. 13, 2001, pp. 637-649.

[21] Nello Cristianini and J. Shawe-Taylor, An introduction to support vector machines and Other Kernel-based Learning Methods, Cambridge University Press, 2000.