
A Gaussian Belief Propagation Solver for Large Scale Support Vector Machines

Danny Bickson

School of Engineering and Computer Science
The Hebrew University of Jerusalem
Givat Ram, Jerusalem, 91904
Israel
danny.bickson@gmail.com

Elad Yom-Tov

IBM Haifa Research Lab
Haifa 31905
Israel
yomtov@il.ibm.com

Danny Dolev

School of Engineering and Computer Science
The Hebrew University of Jerusalem
Givat Ram, Jerusalem, 91904
Israel
dolev@cs.huji.ac.il

Abstract

Support vector machines (SVMs) are an extremely successful type of classification and regression algorithms. Building an SVM entails solving a constrained convex quadratic programming problem, which is quadratic in the number of training samples. We introduce an efficient parallel implementation of an SVM solver, based on the Gaussian Belief Propagation algorithm (GaBP). Unlike previous parallel solutions, our approach can be easily used in peer-to-peer and grid environments, where there is no central authority that allocates the work. We compare the proposed algorithm to previously proposed distributed and single-node SVM solvers. Our comparison shows that the proposed algorithm is just as accurate as these solvers, while being significantly faster, especially for large datasets. We demonstrate scalability of the proposed algorithm to hundreds of nodes and hundreds of thousands of data points using an IBM Blue Gene supercomputer.

1 Introduction

1.1 Classification Using Support Vector Machines

Support-vector machines (SVMs) are a class of algorithms that have, in recent years, exhibited superior performance compared to other pattern classification algorithms. There are several formulations of the SVM problem, depending on the specific application of the SVM (e.g., classification, regression, etc.).

One of the difficulties in using SVMs is that building an SVM requires solving a constrained quadratic programming problem, whose size is quadratic in the number of training examples. This fact has led to much research on efficient SVM solvers. Recently, several researchers have suggested using multiple computing nodes in order to increase the computational power available for solving SVMs.

In this article, we introduce a distributed SVM solver based on the Gaussian Belief Propagation (GaBP) algorithm. We improve on the original GaBP algorithm by reducing the communication load, as represented by the number of messages sent in each optimization iteration, from n^2 to

n aggregated messages, where n is the number of data points. Previously, it was known that the GaBP algorithm is very efficient for sparse matrices. Using our novel construction, we demonstrate that the algorithm exhibits very good performance for dense matrices as well. We also show that the GaBP algorithm can be used with kernels, thus making the algorithm more powerful than previously possible.

These improvements allow the proposed solver to obtain results which are as accurate as previously proposed algorithms, while demonstrating fast convergence and excellent scalability ranging from several computers in a laboratory cluster to many hundreds of computing nodes in an IBM super-computer.

2 Classification using Support Vector Machines

We begin by formulating the SVM problem. Consider a training set:

$$D = \{(\mathbf{x}_i, y_i), \quad i = 1, \dots, N, \quad \mathbf{x}_i \in \mathfrak{R}^m, \quad y_i \in \{-1, 1\}\}. \quad (1)$$

The goal of the SVM is to learn a mapping from \mathbf{x}_i to y_i such that the error in mapping, as measured on a new dataset, would be minimal. SVMs learn to find the linear weight vector that separates the two classes so that

$$y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 \quad \text{for } i = 1, \dots, N \quad (2)$$

There may exist many hyperplanes that achieve such separation, but SVMs find a weight vector \mathbf{w} and a bias term b that maximize the margin $2/\|\mathbf{w}\|$. Therefore, the optimization problem that needs to be solved is

$$\text{Minimize } J_D(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\| \quad (3)$$

$$\text{Subject to } y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 \quad \text{for } i = 1, \dots, N \quad (4)$$

Any points lying on the hyperplane $y_i (\mathbf{x}_i \cdot \mathbf{w} + b) = 1$ are called support vectors.

If the data cannot be separated using a linear separator, a slack variable $\xi \geq 0$ is introduced and the constraint is relaxed to:

$$y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \xi_i \quad \text{for } i = 1, \dots, N \quad (5)$$

The optimization problem then becomes:

$$\text{Minimize } J_D(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\| + C \sum_{i=1}^N \xi_i \quad (6)$$

$$\text{Subject to } y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 \quad \text{for } i = 1, \dots, N \quad (7)$$

$$\xi_i \geq 0 \quad \text{for } i = 1, \dots, N \quad (8)$$

The weights of the linear function can be found directly or by converting the problem into its dual optimization problem, which is usually easier to solve.

Using the notation of Vijayakumar and Wu [7], the dual problem is thus:

$$\text{Maximize } L_D(h) = \sum_i h_i - \frac{1}{2} h' \cdot D \cdot h \quad (9)$$

$$\text{subject to } 0 \leq h_i \leq C, \quad i = 1, \dots, N \quad (10)$$

$$, \quad \sum_i h_i y_i = 0 \quad (11)$$

where \mathbf{D} is a matrix such that $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ and $K(\cdot, \cdot)$ is either an inner product of the samples or a function of these samples. In the latter case, this function is known as the kernel function, which can be any function that complies with the Mercer conditions [9]. For example, these may be polynomial functions, radial-basis (Gaussian) functions, or hyperbolic tangents. If the

data is not separable, C is a tradeoff between maximizing the margin and reducing the number of misclassifications.

The classification of a new data point is then computed using the following equation:

$$(x) = \text{sign} \left(\sum_{i \in SV} h_i y_i K(x_i, x) + b \right) \quad (12)$$

2.1 Previous Approaches for Solving Parallel SVMs

There are several main methods for finding a solution to an SVM problem on a single-node computer. (See Chapter 10 of [9]) for a taxonomy of such methods.) However, since solving an SVM is quadratic in time and cubic in memory, these methods encounter difficulty when scaling to datasets that have many examples and support vectors. The latter two are not synonymous. A large dataset with many repeated examples might be solved using sub-sampling approaches, while a highly non-separable dataset with many support vectors will require an altogether different solution strategy. The literature covers several attempts at solving SVMs in parallel, which allow for greater computational power and larger memory size. In Collobert et al. [15] the SVM solver is parallelized by training multiple SVMs, each on a subset of the training data, and aggregating the resulting classifiers into a single classifier. The training data is then redistributed to the classifiers according their performance and the process is iterated until convergence is reached. The need to re-divide the data among the SVM classifiers means that the data must be moved between nodes several times; this rules out the use of an approach where bandwidth is a concern. A more low-level approach is taken by Zanghirati et al. [14], where the quadratic optimization problem is divided into smaller quadratic programs (similar to the Active Set methods), each of which is solved on a different node. The results are aggregated and the process is repeated until convergence. The performance of this method has a strong dependence on the caching architecture of the cluster. Graf et al. [17] partition the data and solve an SVM for each partition. The support vectors from each pair of classifiers are then aggregated into a new training set for which an SVM is solved. The process continues until a single classifier remains. The aggregation process can be iterated, using the support vectors of the final classifier in the previous iteration to seed the new classifiers. One problem with this approach is that the data must be repeatedly shared between nodes, meaning that once again the goal of data distribution cannot be attained. The second problem, which might be more severe, is that the number of possible support vectors is restricted by the capacity of a single SVM solver. Recently, Yom Tov [8] proposed modifying the sequential algorithm developed in [7] to batch mode. In this way, the complete kernel matrix is held in distributed memory and the Lagrange multipliers are computed iteratively. This method has the advantage that it can efficiently solve difficult SVM problems that have many support vectors to their solution. Based on that work, we show in this paper how an SVM solution can be obtained by adapting a Gaussian Belief Propagation algorithm to the solution of the algorithm proposed in [7].

3 Gaussian Belief Propagation

This section provides a brief background of Gaussian graphical models using the model described in [5]. A Gaussian graphical model is defined by an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges, and a collection of jointly Gaussian random variables $x = (x_i, i \in V)$ with the probability density:

$$p(x) \propto \exp\left(-\frac{1}{2}x'Jx + h'x\right)$$

, where J is a symmetric, positive, definite matrix ($J \succ 0$) that is sparse relative to the graph G : If $\{i, j\} \notin E$ the $J_{ij} = 0$.

The *inference problem* for the Gaussian graphical model is defined as follows [2]: Given a Gaussian graphical model of the information filter form (h, J) , evaluate the marginal densities $x_i = N(\mu_i, P_{ii})$ for all nodes i .

We use the message passing formulation of the Gaussian Belief Propagation (GaBP) algorithm given in [5] to solve the inference problem:

Input: Adjacency matrix J (the correlation matrix), shift vector h
Iterate:

$$h_{i \setminus j} = h_i + \sum_{k \in N(i) \setminus j} \Delta h_{k \rightarrow i}, J_{i \setminus j} = J_{ii} + \sum_{k \in N(i) \setminus j} \Delta J_{k \rightarrow i} \quad (13)$$

$$\Delta h_{i \rightarrow j} = -J_{ji}(J_{i \setminus j})^{-1} h_{i \setminus j}, \Delta J_{i \rightarrow j} = -J_{ji}(J_{i \setminus j})^{-1} J_{ij} \quad (14)$$

where $h_{i \setminus j}, J_{i \setminus j}$ are intermediate terms calculated by the nodes and each message sent from node i to node j is composed of two reals: $\Delta J_{i \rightarrow j}, \Delta h_{i \rightarrow j}$. The first represents the Gaussian precision (one over the variance) and the latter represents the Gaussian mean. Finally, each node locally computes the terms:

$$\hat{h}_i = h_i + \sum_{k \in N(i)} \Delta h_{k \rightarrow i}, \hat{J}_i = J_{ii} + \sum_{k \in N(i)} \Delta J_{k \rightarrow i}$$

and extracts the variance by \hat{J}_i^{-1} and the mean: $\hat{J}_i^{-1} \hat{h}$.

4 Proposed Solution of SVM Solver Based on GaBP

For our proposed solution, we take the exponent of dual SVM formulation given in equation (9) and solve $\max \exp(L_D(h))$. Since $\exp(L_D(h))$ is convex, the solution of $\max \exp(L_D(h))$ is a global maximum that also satisfies $\max L_D(h)$ since the matrix M is symmetric and positive definite. Now we can relate to the new problem formulation as a probability density function, which is in itself Gaussian:

$$p(h) \propto \exp\left(-\frac{1}{2} h' M h + h' 1\right)$$

where 1 is a vector of ones, and find the assignment of $\hat{h} = \arg \max p(h)$. It is known [5] that in Gaussian models finding the MAP assignment is equivalent to solving the inference problem. To solve the inference problem, namely computing the marginals \hat{h} , we propose using the GaBP algorithm, which is a distributed message passing algorithm. We take the computed \hat{h} as the Lagrange multiplier weights of the support vectors of the original SVM data points and apply a threshold for choosing data points with non-zero weight as support vectors.

Finally, following [7], we remove the explicit bias term b and instead add another dimension to the pattern vector x_i such that $\mathbf{x}_i = (x_1, x_2, \dots, x_N, \lambda)$, where λ is a scalar constant. The modified weight vector, which incorporates the bias term, is written as $\hat{\mathbf{w}} = (w_1, w_2, \dots, w_N, b/\lambda)$. However, this modification causes a change to the optimized margin. Vijayakumar and Wu [7] discuss the effect of this modification and reach the conclusion that “setting the augmenting term to zero (equivalent to neglecting the bias term) in high dimensional kernels gives satisfactory results on real world data”. We did not completely neglect the bias term and in our experiments, which used the Radial Basis Kernel, set it to $1/N$, as proposed in [8].

4.1 GaBP Algorithm Convergence

A sufficient condition for the convergence of the GaBP algorithm [1], is for the matrix J to be diagonally dominant:

$$\forall_i J_{ii} > \sum_{k \neq i} |J_{ik}|$$

When converging, the GaBP algorithm converges to the right mean and approximated variance.

In order to force the algorithm to onverge, we artificially weight the main diagonal of the kernel matrix M to make it diagonally dominant. Our empirical results (in Section 6) showc that this modification did not significantly affect the error in classifications on all tested data sets.

A partial justification for weighting the main diagonal is found in [6]. In the 2-Norm soft margin formulation of the SVM problem, the sum of squared slack variables is minimized:

$$\text{Minimize}_{\xi, w, b} \|w\|_2^2 + C \sum_i \xi_i^2$$

$$s.t. y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

The dual problem is derived:

$$W(h) = \sum_{i=1}^l h_i - \frac{1}{2} \sum_{i,j=1}^l y_i y_j h_i h_j (x_i \cdot x_j + \frac{1}{C} \delta_{ij}),$$

where δ_{ij} is the Kronecker δ defined to be 1 when $i = j$, and zero elsewhere. It is shown that the only change relative to the 1-Norm soft margin SVM is the addition of $1/C$ to the diagonal of the inner product matrix associated with the training set. This has the effect of adding $1/C$ to the eigenvalues, rendering the kernel matrix (and thus the GaBP problem) better conditioned [6].

4.2 Convergence in Asynchronous Settings

One of the desired properties of a large scale algorithm is that it should converge in asynchronous settings as well as in synchronous settings. This is because in a large-scale communication network, clocks are not synchronized accurately and some nodes may be slower than others, while some nodes experience longer communication delays.

Recent work by Koller et. al [3] defines conditions for the convergence of belief propagation. This work shows how to create a global metric for measuring the change in BP messages; if this metric forms a max-norm construction, the BP algorithm converges under some assumptions. Using experiments on various network sizes, up to a sparse matrix of one million over one million nodes, the algorithm converged asynchronously in all cases where it converged in synchronous settings. Furthermore, as noted in [3], in asynchronous settings the algorithm converges faster as compared to synchronous settings.

5 Algorithm Optimization

This section describes our novel modification to the GaBP Algorithm. This modification significantly reduces the number of messages sent in the network in each round. Recall that in the GaBP algorithm each node out of the n nodes sends a unique message to every other node, which results in a communication overhead of n^2 messages per round (See Equation 14). Initially, when we implemented the algorithm using our cluster, we noticed that communication between all pairs is the major bottleneck of the algorithm. To improve the algorithms' performance in communication networks, we modified the GaBP algorithm an equivalent variant albeit where in each round only n sums are aggregated. Following round termination, each node updates the received messages based on information stored in the node and obtains the original GaBP messages.

This modification can be used in other settings such as communication channels that support the accumulation of transmitted signals like Code Division Multiple Access (CDMA) systems.

In our proposed changes to the algorithm, first n aggregated sums are computed in the network:

$$\bar{h}_i = h_i + \sum_{k \in N(i)} \Delta h_{k \rightarrow i}, \bar{J}_i = J_{ii} + \sum_{k \in N(i)} \Delta J_{k \rightarrow i}$$

where \bar{h}_i, \bar{J}_i is the sum of all messages sent to node i . Second, each node j computes $2n$ local corrections to retrieve the unique $2n$ messages sent to it:

$$h_{i \setminus j} = \bar{h}_i - \Delta h_{j \rightarrow i}, J_{i \setminus j} = \bar{J}_i - \Delta J_{j \rightarrow i}$$

$$\Delta h_{i \rightarrow j} = -J_{ji}(J_{i \setminus j})^{-1} h_{i \setminus j}, \Delta J_{i \rightarrow j} = -J_{ji}(J_{i \setminus j})^{-1} J_{ij}$$

This can be done locally since node j knows h_i for all i because they are broadcasted to every node. The local corrections are done by subtracting $\Delta h_{j \rightarrow i}$ for each i , which is available locally only at node j .

6 Experimental Results

We implemented our proposed algorithm using approximately 1,000 lines of code in C. We implemented communication between the nodes using the MPICH2 message passing interface ¹. Each node was responsible for d data points out of the total n data points in the dataset.

¹<http://www-unix.mcs.anl.gov/mpi/mpich/>

Dataset	Dimension	Train	Test	ERROR (%)		
				GaBP	Sequential	SVMlight
Isolet	617	6238	1559	7.06	5.84	49.97
Letter	16	20000		2.06	2.06	2.3
Mushroom	117	8124		0.04	0.05	0.02
Nursery	25	12960		4.16	5.29	0.02
Pageblocks	10	5473		3.86	4.08	2.74
Pen digits	16	7494	3498	1.66	1.37	1.57
Spambase	57	4601		16.3	16.5	6.57

Table 1: Error rates of the GaBP solver versus those of the parallel sequential solver and SVMlight

Dataset	RUN TIMES (SEC)	
	GaBP	Sequential
Isolet	228	1328
Letter	468	601
Mushroom	226	176
Nursery	221	297
Pageblocks	26	37
Pen digits	45	155
Spambase	49	79

Table 2: Running times (in seconds) of the GaBP solver (working in a distributed environment) compared to that of the IBM parallel solver

Our implementation used synchronous communication rounds because of MPI limitations. In Section 7 we further elaborate on this issue.

Each node was assigned several examples from the input file. Then, the kernel matrix M was computed by the nodes in a distributed fashion, so that each node computed the rows of the kernel matrix related to its assigned data points. After computing the relevant parts of the matrix M , the nodes weighted the diagonal of the matrix M , as discussed in Section 4.1. Then, several rounds of communication between the nodes were run. In each round, using our optimization, a total of n sums were calculated using MPI_Allreduce system call. Finally, each node output the solution x , which was the mean of the input Gaussian that matched its own data points. Each x_i signified the weight of the data point i for being chosen as a support vector.

To compare our algorithm performance, we used two algorithms: Sequential SVM (SVMSeq) [7] and SVMlight [16]. We used the SVMSeq implementation provided within the IBM Parallel Machine Learning (PML) toolbox [10]. The PML implements the same algorithm by Vijaykumar and Wu [7] that our GaBP solver is based on, but the implementation in through a master-slave architecture as described in [8]. SVMlight is a single computing node solver.

Table 1 describes the seven datasets we used to compare the algorithms and the classification accuracy obtained. These computations were done using five processing nodes (3.5GHz Intel Pentium machines, running the Linux operating system) for each of the parallel solvers. All datasets were taken from the UCI repository [11]. We used medium-sized datasets so that run-times using SVMlight would not be prohibitively high. All algorithms were run with an RBF kernel. The parameters of the algorithm (kernel width and misclassification cost) were optimized using a line-search algorithm, as detailed in [12].

Note that SVMlight is a single node solver, which we use mainly as a comparison for the accuracy in classification.

Using the Friedman test [13], we did not detect any statistically significant difference between the performance of the algorithms with regards to accuracy ($p < 10^{-3}$).

Figure 1 shows the speedup results of the algorithm when running the GaBP algorithm on a Blue Gene supercomputer. The speedup with N nodes is computed as the run time of the algorithm on a single node, divided by the run time using N nodes. Obviously, it is desirable to obtain linear

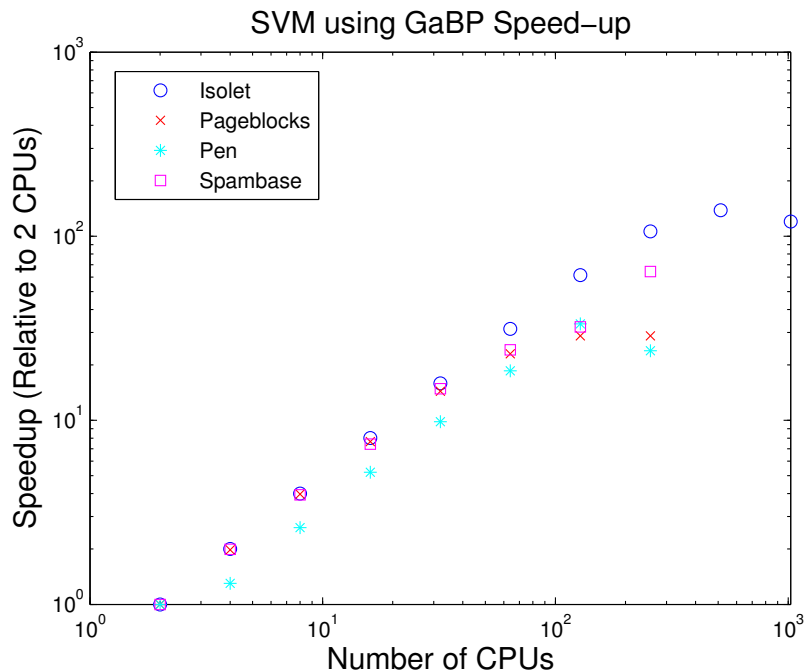


Figure 1: Speedup of the GaBP algorithm vs. 2 CPUS

Dataset	Dimension	Number of examples	Run time (sec)
Coverttype	54	150,000	468
MNIST	784	60,000	756

Table 3: Running times of the GaBP solver for large data sets using 1024 CPUs on an IBM Blue Gene supercomputer

speedup, i.e., doubling computational power halves the processing time, but this is limited by the communication load and by parts of the algorithm that cannot be parallelized. Since Blue Gene is currently limited to 0.5 GB of memory at each node, most datasets could not be run on a single node. We therefore show speedup compared to two nodes. As the figure shows, in most cases we obtain linear speedup up to 256 CPUs, which is considered a very broad range of computing power. When using 512 - 1024 CPUs, the communication overhead reduces the efficiency of the parallel computation. We identified this problem as an area for future research into optimizing the performance for larger scale grids.

We also tested the ability to build classifiers for larger datasets. Table 3 shows the run times of the GaBP algorithm using 1024 CPUs on two larger datasets, both from the UCI repository. Due to memory restrictions of Blue Gene we used only the first 150,000 samples of the Coverttype dataset. The execution times obtained using the proposed algorithm are relatively short (under 15 minutes). This demonstrates the ability of the algorithm to process very large datasets in a reasonably short amount of time.

7 Discussion

In this paper we demonstrated the application of the Gaussian Belief Propagation to the solution of SVM problems. Our experiments demonstrate the usefulness of this solver, being both accurate and scalable.

We implemented our algorithm using a synchronous communication model mainly because MPICH2 does not support asynchronous communication. While synchronous communication is

the mode of choice for supercomputers such as Blue Gene, in many cases such as heterogeneous grid environments, asynchronous communication will be preferred. We believe that the next challenging goal will be to implement the proposed algorithm in asynchronous settings, where algorithm rounds will no longer be synchronized.

Our initial experiments with very large sparse kernel matrices (millions of data points) show that asynchronous settings converge faster. Recent work by Koller [3] supports this claim by showing that in many cases the BP algorithm converges faster in asynchronous settings.

Another challenging task would involve scaling to data sets of millions of data points. Currently the full kernel matrix is computed by the nodes. While this is effective for problems with many support vectors [8], it is not required in many problems which are either easily separable or else where the classification error is less important compared to the time required to learn the mode. Thus, solvers scaling to much larger datasets may have to diverge from the current strategy of computing the full kernel matrix and instead sparsify the kernel matrix as is commonly done in single node solvers.

Finally, it remains an open question whether SVMs can be solved efficiently in Peer-to-Peer environments, where each node can (efficiently) obtain data from only several close peers. Future work will be required in order to verify how the GaBp algorithm performs in such an environment, where only partial segments of the kernel matrix can be computed by each node.

References

- [1] Y. Weiss and W. T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. In NIPS-12, 1999
- [2] J.K. Johnson. Walk-summable Gauss-Markov random fields. Technical Report, February 2002. (Corrected, November 2005).
- [3] G. Elidan and I. McGraw and D. Koller, Residual Belief Propagation: Informed Scheduling for Asynchronous Message Passing, Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI), Boston, Massachusetts, 2006
- [4] J.K. Johnson, D.M. Malioutov, A.S. Willsky. Walk-sum interpretation and analysis of Gaussian belief propagation, In Advances in Neural Information Processing Systems, vol. 18, pp. 579-586, 2006.
- [5] D.M. Malioutov, J.K. Johnson, A.S. Willsky. Walk-sums and belief propagation in Gaussian graphical models, Journal of Machine Learning Research, vol. 7, pp. 2031-2064, October 2006.
- [6] Nello Cristianini and John Shawe-Taylor. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, 2000. ISBN 0-521-78019-5.
- [7] Sethu Vijayakumar and Si Wu (1999), Sequential Support Vector Classifiers and Regression. Proc. International Conference on Soft Computing (SOCO'99), Genoa, Italy, pp.610-619.
- [8] Elad Yom-Tov (2007), A distributed sequential solver for large scale SVMs. In: O. Chapelle, D. DeCoste, J. Weston, L. Bottou: Large scale kernel machines. MIT Press, pp. 141-156.
- [9] B. Schölkopf and A. J. Smola. Learning with kernels: Support vector machines, regularization, optimization, and beyond. MIT Press, Cambridge, MA, USA, 2002.
- [10] <http://www.alphaworks.ibm.com/tech/pml>
- [11] Catherine L. Blake, Eamonn J. Keogh, and Christopher J. Merz. UCI repository of machine learning databases, 1998. URL <http://www.ics.uci.edu/~lmsim/learn/MLRepository.html>.
- [12] R. Rifkin and A. Klautau. In defense of One-vs-All classification. Journal of Machine Learning Research, 5:101-141, 2004.
- [13] J. Demšar. Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research, 7:1-30, 2006.
- [14] G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. Parallel computing, 29:535-551, 2003.

- [15] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems. In *Advances in Neural Information Processing Systems*. MIT Press, 2002.
- [16] T. Joachims. Making large-scale svm learning practical. In "B. Schölkopf, C. Burges, A. Smola" (Editors), *Advances in Kernel Methods - Support Vector Learning*,
- [17] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. In *Advances in Neural Information Processing Systems*, 2004.