

A Large-Scale Gaussian Belief Propagation Solver for Kernel Ridge Regression

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 University Campus
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

We introduce an efficient parallel implementation of a Kernel Ridge Regression solver, based on the Gaussian Belief Propagation algorithm (GaBP). Our approach can be easily used in Peer-to-Peer and grid environments, where there is no central authority that allocates work. Empirically, our solver has high accuracy in solving classification problems. We have tested our distributed implementation on large scale datasets using up to 1,024 CPUs of an IBM Blue Gene supercomputer.

1. KERNEL RIDGE REGRESSION

Kernel Ridge Regression (KRR) implements a regularized form of the least squares method useful for both regression and classification. The non-linear version of KRR is similar to the Support-Vector Machine (SVM) problem. However, in the latter, special emphasis is given to points close to the decision boundary, which is not provided by the cost function used by KRR.

Given training data

$$\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^l, \mathbf{x}_i \in R^d, y_i \in R$$

the KRR algorithm determines the parameter vector $\mathbf{w} \in R^d$ of a non-linear model (using the “kernel trick”), via minimization of the following objective function: [2]:

$$\min \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^l (y_i - \mathbf{w}^T \Phi(\mathbf{x}_i))^2$$

where λ is a tradeoff parameter between the two terms of the optimization function, and $\Phi(\cdot)$ is a (possible non-linear) mapping of the training patterns.

One can show that the dual form of this optimization problem is given by:

$$\max W(\alpha) = \mathbf{y}^T \mathbf{a} + 1/4\lambda\alpha^T \mathbf{K}\alpha - 1/4\alpha^T \mathbf{a} \quad (1)$$

where \mathbf{K} is a matrix whose (i, j) -th entry is the kernel function $\mathbf{K}_{i,j} = \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)$.

The optimal solution to this optimization problem is:

$$\alpha = 2\lambda(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

The corresponding prediction function is given by:

$$f(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) = \mathbf{y}^T (\mathbf{K} + \lambda\mathbf{I})^{-1} \mathbf{K}(\mathbf{x}_i, \mathbf{x}).$$

2. SOLVING A KRR PROBLEM USING GABP

The contributions of this paper are twofold: First, we utilize Gaussian Belief propagation, an efficient distributed algorithm that has been shown to converge empirically faster [6] than other methods proposed in the literature for solving this kind of problem. These methods are those used for solving Gaussian Processes (for a review see [8]) and include block Gauss-Seidel, Conjugate Gradient, and others. Additionally, we present a very large scale implantation (tested up to 1,024 computing nodes) of the parallel GaBP solver and demonstrate its applicability for solving support vector regression and classification problems.

The GaBP algorithm is a special case of the sum-product algorithm when the underlying distribution is Gaussian. Our aim is to find α , a solution to the quadratic cost function 1. As the kernel matrix \mathbf{K} is symmetric we can define a jointly Gaussian probability $p(\alpha) = \exp(W(\alpha))$ and infer $\max p(\alpha)$ which is an optimal solution for the KRR problem.

The formulation above allows us to shift the rating problem from an algebraic to a probabilistic domain. Instead of solving a deterministic vector-matrix linear equation, we now solve an inference problem in a graphical model describing a certain Gaussian distribution function. The move to the probabilistic domain calls for the utilization of belief propagation as an efficient inference engine. In [6] we demonstrated how to derive the GaBP update rules by substituting Gaussian distributions in the continuous BP equations. The output of this derivation are update rules that are computed locally by each node, thus allowing an efficient distributed implementation.

3. CONVERGENCE OF THE GABP ALGORITHM

A sufficient condition for the convergence of the GaBP algorithm [1], is for the matrix $\mathbf{J} = (\mathbf{K} + \lambda\mathbf{I})$ to be diagonally dominant:

$$\forall_i \mathbf{J}_{ii} > \sum_{k \neq i} |\mathbf{J}_{ik}|$$

Under these conditions the GaBP algorithm converges to the right mean and approximated variance.

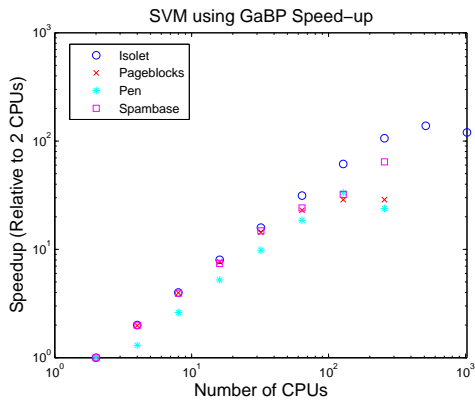


Figure 1: Speedup of the GaBP algorithm vs. 2 CPUS

We force the algorithm to converge by adding a small constant λ to weight the main diagonal. Our empirical results shows that this modification did not significantly affect the error in classifications on all tested data sets.

4. EXPERIMENTAL RESULTS

To compare our algorithm performance, we used two algorithms: Sequential SVM (SVMSeq) [7] and SVMlight [5]. We used the Sequential SVM implementation provided within the IBM Parallel Machine Learning (PML) toolbox [3].

Each node is assigned a subset of the data points. Then, the kernel matrix \mathbf{K} is computed by the nodes in a distributed fashion, i.e. each node computes the rows of the kernel matrix related to its assigned data points. After computing a subset of the kernel matrix \mathbf{K} , the nodes tune the parameter λ . Then the GaBP algorithm is run. We have set the number of rounds empirically to 3 rounds, for reaching convergence.

Dataset	ERROR (%)		
	GaBP	Sequential	SVMlight
Isolet	7.06	5.84	49.97
Letter	2.06	2.06	2.3
Mushroom	0.04	0.05	0.02
Nursery	4.16	5.29	0.02
Pageblocks	3.86	4.08	2.74
Pen digits	1.66	1.37	1.57
Spambase	16.3	16.5	6.57

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

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 [4]. 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.

Figure 1 shows the speedup results of the algorithm when running the GaBP algorithm on a Blue Gene supercomputer. The speedup

Dataset	Dim	N	Run time (sec)
Covertypes	54	150,000	468
MNIST	784	60,000	756

Table 2: Running times of the GaBP solver for large data sets using 1024 CPUs on an IBM Blue Gene supercomputer

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 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 400 MB 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 tested the ability to build classifiers for larger datasets. Table 2 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 Covertypes 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.

We believe that our preliminary results are exciting and demonstrate the feasibility of our approach. We plan to extend our work by implementing kernel matrix caching and selection of active subsets as done suggested by Zanni et al. [9].

5. REFERENCES

- [1] Y. Weiss and W. T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. In NIPS-12, 1999
- [2] 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.
- [3] <http://www.alphaworks.ibm.com/tech/pml>
- [4] Catherine L. Blake, Eamonn J. Keogh, and Christopher J. Merz. UCI repository of machine learning databases, 1998. URL [http://www.ics.uci.edu/~sim\\$mllearn/MLRepository.html](http://www.ics.uci.edu/~sim$mllearn/MLRepository.html).
- [5] T. Joachims. Making large-scale svm learning practical. In "B. Schölkopf, C. Burges, A. Smola" (Editors), *Advances in Kernel Methods - Support Vector Learning*,
- [6] Danny Bickson, Danny Dolev, Ori Shental, Paul H. Siegel and Jack K. Wolf. Linear Detection via Belief Propagation. In the 45th Annual Allerton Conference on Communication, Control, and Computing, Allerton House, Illinois, Sept. 07'.

- [7] Sethu Vijayakumar and Si Wu, Sequential Support Vector Classifiers and Regression. Proc. International Conference on Soft Computing (SOCO'99), Genoa, Italy, pp.610-619.
- [8] Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian Processes for Machine Learning. MIT Press, 2006. ISBN 0-262-18253-X. Chapter 2.1.1.
- [9] L. Zanni, T. Serafini, G. Zanghirati, Parallel Software for Training Large Scale Support Vector Machines on Multiprocessor Systems, JMLR 7(Jul), 1467-1492, 2006.