FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Technical Report

HyParSVM – A New Hybrid Parallel Software for Support Vector Machine Learning on SMP Clusters

*Tatjana Eitrich, Wolfgang Frings, Bruno Lang**

FZJ-ZAM-IB-2006-04

March 2006

(last change: 2.3.2006)

Preprint: submitted for publication

(*) Applied Computer Science and Scientific Computing Group
Department of Mathematics
University of Wuppertal, Germany

HyParSVM – A New Hybrid Parallel Software for Support Vector Machine Learning on SMP Clusters

Tatjana Eitrich[1], Wolfgang Frings[1], and Bruno Lang[2]

[1] Central Institute for Applied Mathematics, Research Centre Jülich, Germany
t.eitrich@fz-juelich.de
[2] Applied Computer Science and Scientific Computing Group, Department of Mathematics, University of Wuppertal, Germany

Abstract. In this paper we describe a new hybrid distributed/shared memory parallel software for support vector machine learning on large data sets. The support vector machine (SVM) method is a well-known and reliable machine learning technique for classification and regression tasks. Based on a recently developed shared memory decomposition algorithm for support vector machine classifier design we increased the level of parallelism by implementing a cross validation based on message passing. With this extention we obtained a flexible parallel SVM algorithm that can be used on high-end machines with SMP architectures to process the large data sets that arise more and more in bioinformatics and other fields of research.

1 Introduction

Support vector machines are well-known data mining methods for classification and regression problems [1]. Their popularity is mainly due to their applicability in various fields of data mining, such as text mining [2], biomedical research [3], and many more. Their accuracy is excellent and in many cases they outperform other machine learning methods such as neural networks. SVMs have their roots in the field of statistical learning which provides the reliable generalization theory [4]. Several properties that make this learning method successful are well-known, e.g. the kernel trick [5] for nonlinear classification and the sparse structure of the final classification function. In addition, SVMs have an intuitive geometrical interpretation, and a global minimum can be located during the SVM training phase. In comparison to genetic algorithms or neural networks, less experience is required for using them, which helps researchers to get started with SVM software quite fast. The main drawback of current SVM models is their high computational complexity for large data sets [6]. This can in fact restrict the applicability of SVMs since the amount of data for classification modeling increases dramatically. Therefore the development of highly scalable parallel SVM algorithms is a new important topic for current SVM research. Some algorithms for parallel SVM learning already do exist, but most of them

are limited to heuristics for distributed training on reduced data sets. These are not useful as stand-alone systems for high quality learning on large data.

In this paper we propose an efficient parallel support vector machine software well suited for multi-processor shared memory (SMP) clusters that become more and more available. Our algorithm can be used in serial and parallel mode. The parallel implementation provides pure MPI and OpenMP modes as well as a hybrid mode which combines fine and coarse grained parallelization aspects to a well scalable SVM learning method.

The remainder of this paper is organized as follows. In Sect. 2 we briefly review the basic concepts of support vector machine learning and describe the SVM parameter optimization problem, which leads to the enormous computational challenges we address in this paper. We limit the discussion to the issues that are essential for understanding the following sections. Since the field of parallel SVM methods is quite new and implementations are rare, we give a detailed review of existing approaches for parallel data mining and support vector machine learning in Sect. 3. One aim of this paper is therefore to present the current state-of-the-art in parallel support vector machine design. In Sect. 4 we explain the structure of our new parallel SVM software *HyParSVM*. In Sect. 5 we present first experimental results on the IBM p690 cluster JUMP at Research Centre Jülich. Finally, Sect. 6 contains a summary and shows directions for future work.

2 Theoretical Background

In this paper we consider the well-known supervised binary classification problem [7]. Given a training set

$$\left\{ \left(\boldsymbol{x}^i, y_i \right) \in \mathbb{R}^n \times \{-1, 1\}, \quad i = 1, \ldots, l \right\} \ ,$$

where $l \in \mathbb{N}$ is the number of given instances and $n \in \mathbb{N}$ the number of attributes in the data set, the task of support vector machine learning is to find a hypothesis function $h : \mathbb{R}^n \to \mathbb{R}$ that can be used to classify unseen data. The hypothesis function, the sign of which is used to classify a point $\boldsymbol{x}$, is of the form

$$h(\boldsymbol{x}) = \sum_{i:\alpha_i > 0} y_i \alpha_i K(\boldsymbol{x}^i, \boldsymbol{x}) + b^* .$$

It is mainly controlled by the so-called Lagrange multipliers α_i $(i = 1, \ldots, l)$. They can be determined via the solution of the quadratic programming (qp) problem

$$\left.\begin{aligned} \min_{\boldsymbol{\alpha} \in \mathbb{R}^l} \quad & \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}^i, \boldsymbol{x}^j) - \sum_{i=1}^{l} \alpha_i \\ \text{s.t.} \quad & \\ & \sum_{i=1}^{l} y_i \alpha_i = 0 \ , \qquad 0 \le \alpha_i \le C \quad (1 \le i \le l) \ . \end{aligned}\right\} \tag{1}$$

The function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is known as the kernel [1] and measures similarity between input vectors in the SVM feature space. $C \in \mathbb{R}_+$ is an SVM

internal error penalization parameter which controls the trade-off between a large margin and the corresponding training errors. We refer to [1] for a detailed description of the SVM learning problem.

Usually, for SVM learning either the L_1-norm or the L_2-norm approach is used. In this paper we work with the L_1-norm approach (1) and avoid the discussion about SVM internal algorithmics. Our software is able to handle both methods. All details to our flexible serial implementation are given in [8] where we presented a comparison between these methods and observed a superiority of the L_1-norm model for unbalanced classification problems.

One of the main challenges when using SVM-based methods is parameter selection. Several data dependent parameter values need to be adjusted [9]. Different methods for tuning the parameters have been proposed [10]. One of them is a search procedure that iteratively creates new parameter values using quality results from k-fold cross validation. In Fig. 1 we explain this method for $k = 4$. A

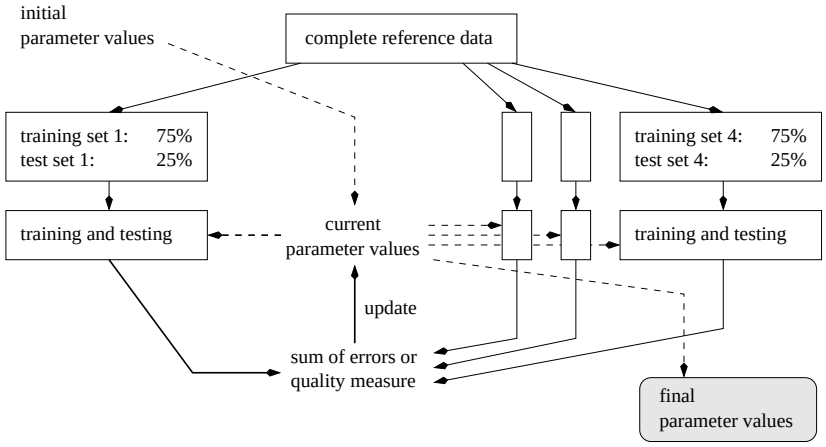

Fig. 1. Structure of parameter tuning with a 4-fold cross validation method.

k-fold cross validation includes k SVM training and test stages as well as a final combination of the results to obtain a quality measure value. We are working with our implementation of the decomposition method which includes the fast projection method proposed in [11]. However, a single SVM training is expensive for large data. Thus, a complete validation takes a very long time.

Our work is aimed at speeding up the SVM parameter optimization time. Please note that parameter tuning usually means to perform a large number of validation stages. Efficient and fast methods are of great interest since they allow for an extensive scan of the parameter space and usage of additional parameters, e.g. for sensitive classification of highly unbalanced data [9].

3 History of Parallel Support Vector Machines

Most sequential data mining algorithms have large runtimes, but the volume of data available for analysis is growing rapidly, i.e. the number of attributes as well as the number of instances both increase. In addition to improvements of the serial algorithms the development of parallel techniques may help to avoid computational bottlenecks. This section gives an overview of activities concerning large scale data mining, particularly the problem of classification using machine learning techniques like SVMs.

Parallel Data Mining

The first parallel data mining algorithms have emerged a decade ago. In [12] the general differences between parallel data mining and other numerical parallel algorithms are explained. The design of scalable data mining algorithms requires meeting several challenges, e.g., the enormous memory requirements have to be supported by the computing system. Various algorithms, especially for supervised learning methods, have been parallelized.

- A parallel algorithm for data mining of association rules was presented in [13]. It has been designed for work on shared memory multiprocessors.
- The *ScalParC* software [14], designed in 1998, was one of the first methods for parallel decision tree classification. Parallel decision tree applications are still of interest, mainly in the important field of Grid computing [15].
- Clustering is useful in various fields, i.e., pattern recognition and learning theory. The runtime complexity of a serial k-means algorithm is high for problems of large size. Therefore parallel k-means clustering methods have been developed. We refer to [16] for a master-slave approach.
- K-nearest neighbor methods have received a great deal of attention since they are applied frequently in bioinformatics, but performance is a serious problem for many implementations. In [17] a parallel algorithm was introduced to overcome the problem of runtime.
- Artificial neural networks (ANNs) are well-known data mining methods with high learning cost when the models are large. One approach for speeding up their implementation by using parallel environments is given in [18].
- Bayesian networks for unsupervised classification tasks include time consuming steps which can be parallelized. A description is given in [19].
- Boosting is a general method for improving the accuracy of any given learning algorithm [20] and is often used within the context of supervised learning. A framework for distributed boosting is presented in [21]. The method requires less memory and computational time than serial boosting packages.

Parallel Support Vector Machine Approaches

Efficient and parallel support vector machine learning is a young and emerging field of research, but the number of truly parallel implementations is small.

Most approaches just try to increase the efficiency of the serial algorithms and to overcome the problem of large scale applications by dividing the data into subsets.

- Different approaches for splitting a large data set into small subsets have been implemented [22]. Usually results of the individual training stages are merged and so finally a single SVM model is obtained. The individual optimization steps can be run in parallel.
- A fast SVM algorithm, which uses caching, digest and shrinking policies is given in [23].
- The clustering-based SVM [24] is a learning method that scans the data set before training the SVM. It selects the data which are supposed to maximize the benefit of learning and is useful for very large problems when a limited amount of resources is given. So far it is only applicable for linear problems.

In addition various projects exist where a simple parallelization scheme is used to speed up the learning process.

- In [25] a parallel optimization step is proposed. It approximates the kernel matrix by block diagonal matrices and splits the original problem into subproblems which can be solved independently from each other with standard algorithms. This step is used to remove non-support vectors before SVM training.
- A parallel training of several binary SVMs for solving multiclass problems is described in [26].
- Parallel cross validation methods exist for the *WEKA* machine learning package [27].
- Parallel parameter optimization techniques such as grid search or pattern search have been studied for SVM parameter fitting [28].

These approaches can be interpreted as coarse grained parallelization techniques for SVM methods at a high level which is independent from the inner qp solver. However, the computational bottleneck of a single SVM training on a large data set can be avoided only by implementing a fine grained parallel support vector machine training. The following methods have been proposed.

- Parallel computation of the kernel matrix for high dimensional data spaces is implemented in [29]. The speedup is limited because of high communication costs. Therefore an approximation method that reduces the kernel matrix was implemented, too. The method is applicable only for commonly used kernels which are inner product-based and requires changes in the algorithm for each kernel.
- A distributed SVM algorithm for row-wise and column-wise data distribution is described in [26], which so far can be used for linear SVMs only.
- A promising parallel MPI-based decomposition solver for training support vector machines has been implemented recently [30].
- A parallel support vector machine for multi-processor shared memory (SMP) clusters has been introduced in [31].

4 A New Hybrid Support Vector Machine Software

In [31] we have discussed a mixed library/loop-based shared memory parallelization for a single SVM training. We have continued to optimize the parallel code, i.e., in addition to the mixed parallelization we implemented two versions of the parallel SVM training that perform library- or loop-based parallelization exclusively (except for the distributed kernel computations). The first one is based on calls to the shared memory parallel version of the *ESSL* (*Engineering Scientific Subroutine Library*) [32], whereas the second one implements OpenMP loop level parallelism. This scheme was realized for the outer decomposition loop, as well as the projection method and the inner solver. The settings may be chosen independently for each routine by using C preprocessor macro names. The code is written in *Fortran90*, and the *IBM XL Fortran compiler* is used. We observed satisfactory speedups for moderate numbers of processors on the IBM supercomputer JUMP. For a larger number of processors the speedups tended to stagnate or even decreased. The training routine comprises some sequential parts that cannot be parallelized, e.g., the iterative working set selection scheme. These parts consume approximately 5% of the training time for data sets with more than 10000 points. In addition, the working set size, an important parameter for the decomposition loop that determines the size of the qp problem (1), which is solved within the parallel OpenMP mode, is limited by the available memory. Therefore the *ESSLsmp* routines have limited scalability for increasing numbers of threads. All in all, for the data we have analyzed, the attainable speedup was limited to values between 5 and 10. For a large number of threads (> 12) the speedups started to decrease. In this paper we present a parallel software which speeds up the SVM learning process to a greater extent by exploiting an additional level of parallelism.

So far, the parallel shared memory SVM training had been embedded into the serial validation loop as it is shown in Fig. 2. At this higher level we added a new parallelization scheme. A pure extension of the shared memory approach was not reasonable since usage of more than 32 processors on the JUMP supercomputer would mean to assign the validation tasks to different nodes which do not share the same memory and can communicate with MPI-based functions only. Therefore we implemented a hybrid parallel support vector machine with an MPI-based cross validation routine. Using a coarse grained parallelization scheme the k validation steps for a k-fold cross validation are distributed to p processes, each of which performs a training-and-testing step for k/p data sets. Each training may in turn be executed by multiple threads, as shown in Fig. 2. Since I/O is necessary only at the beginning of the program, we could use a simple data distribution scheme. A single ("master") process reads the complete training data, preprocesses it and then calls MPI collective broadcast operations to distribute the validation matrix to the other processes. Inside the validation loop each process uses the matrix k/p times to extract training and test data. Each process accumulates results of the local validation tests during execution of the program. At the end of the validation loop, MPI collective reduction operations compute the overall results, and the master process calculates the overall

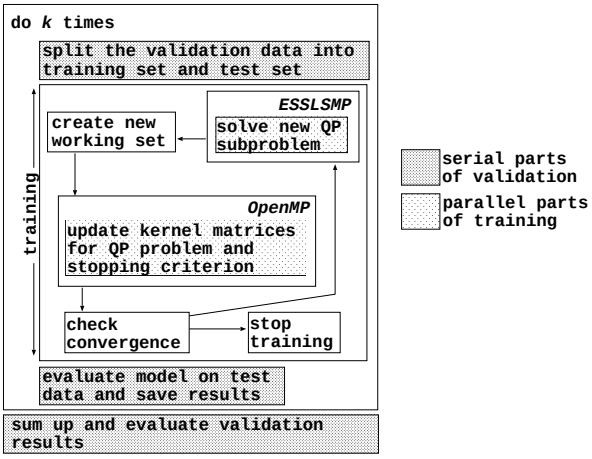

Fig. 2. Shared memory parallel SVM training as part of the validation loop to be parallelized.

quality measure. Each validation step consists of a single SVM training on a data matrix with n features and $l \cdot (1 - 1/k)$ instances. It is known that training time is quadratic in the number of instances and linear in the number of features and does not heavily depend on other parameters except the outer SVM parameters which do not change during a single validation process. Due to this relatively balanced load and the fact that variances in time are data dependent and unpredictable, the assignment of validation jobs to processes was implemented in a straight forward way. As it can be seen in Fig. 2 each step of the cross validation method previously comprised some non-parallel parts (dark grey), which we have parallelized now with a distributed memory approach to increase the efficiency of the overall scheme. The additional speedup obtained by the hybrid parallelization is particularly useful in the context of parameter search, since a large number of validation steps may be necessary here. Sophisticated parameter search is usually performed iteratively and new paths in the parameter space are defined based on former results. For simple approaches like grid search, where even the whole validation steps are independent, the MPI-parallel part of our software may be turned off.

5 Experimental Results

We performed our tests on the Juelich Multi Processor (JUMP) at Research Centre Jülich [33]. JUMP is a distributed shared memory parallel computer consisting of 41 frames (nodes). Each node contains 32 IBM Power4+ processors running at 1.7 GHz, and 128 GB shared main memory. The 1312 processors have an aggregate peak performance of 8.9 TFlop/s. For our tests we have used a QSAR data set from pharmaceutical industry with 40000 instances and 50 features. We show results for an SVM with the Gaussian kernel. However, any other

Table 1. Comparison of (running time in seconds : speedup : efficiency) for 8-fold cross validation using the L_1-norm approach with a Gaussian kernel. A data set with 40000 instances and 50 features was tested.

		# processes			
		1	2	4	8
# threads	1	6105 : 1.0 : 1.00	3074 : 2.0 : 1.00	1566 : 3.9 : 0.98	834 : 7.3 : 0.91
	2	3157 : 1.9 : 0.95	1599 : 3.8 : 0.95	815 : 7.5 : 0.94	453 : 13.5 : 0.84
	3	2168 : 2.8 : 0.93	1109 : 5.5 : 0.92	577 : 10.6 : 0.88	348 : 17.5 : 0.73
	4	1641 : 3.7 : 0.93	847 : 7.2 : 0.90	444 : 13.7 : 0.86	284 : 21.5 : 0.67
	5	1362 : 4.5 : 0.90	703 : 8.7 : 0.87	366 : 16.7 : 0.84	187 : 32.7 : 0.82
	6	1172 : 5.2 : 0.87	609 : 10.0 : 0.83	326 : 18.7 : 0.78	165 : 37.0 : 0.77
	7	1054 : 5.8 : 0.83	549 : 11.1 : 0.79	299 : 20.4 : 0.73	155 : 39.4 : 0.70
	8	978 : 6.2 : 0.78	518 : 11.9 : 0.74	290 : 21.9 : 0.68	158 : 42.9 : 0.67

kernel function is applicable, since the kernel function itself is not parallelized. The user may integrate his own kernel function into the software. We believe that this concept of a non-parallel kernel function is crucial for a flexible usage of the parallel SVM software as it allows for the classification of data sets with widely differing characteristics. Due to the fact that we focus on a parallelization scheme, no accuracy results for the data in this paper are given. Concerning verification and improvement of SVM quality we refer to our work [8, 9, 28].

In the following we present the results for an 8-fold cross validation task using the hybrid software with the *ESSLsmp*-based inner parallelization. The working set size of the decomposition method was set to the largest possible value of $40000 \cdot 7/8 = 35000$, which is the size of the qp problems to be solved in the cross validation loop. For the allocation of matrices and vectors during computation each process needed 12 GB of memory, which was then used by the threads assigned to each process. In cases where only a smaller amount of memory is available the working set size may be reduced. This will cause the decomposition method to optimize the vector $\boldsymbol{\alpha}$ iteratively. As we mentioned in the last chapter, each validation step is expected to consume approximately the same amount of time. For our data set the timings were between 751 and 778 seconds with a mean value of 763. These results were obtained with one thread and a single process on JUMP. Thus, the time differences between the steps are negligible and the assignment of steps to the available processes may indeed be implemented without a special mapping method.

In Table 1 we show speedup and efficiency values for various combinations of processes and threads. The additional level of parallelism successfully increased the achievable speedup. Most interesting it the last column. The efficiency decreases from 0.91 down to 0.67 for 32 processors. If additional 8 processors are added, the efficiency increases to 0.82 and decreases again for further more processors. For tests with more than 32 processors two nodes of JUMP are used; all other tests were run on a single node. With using two nodes, memory bandwidth limitations become visible. However, our speedup values are promising – for 64

processors the SVM validation time decreased with a factor of 43 by using 8 processes with 8 threads each.

6 Summary and Future Work

In this paper we presented the new *HyParSVM* software for parallel SVM learning. This software, which is under development at the Research Centre Jülich, helps speeding up the data mining pipeline in various fields of classification applications. The hybrid implementation is very flexible and shows promising results on the JUMP supercomputer. In addition to the hybrid SVM software the user may increase the level of parallelism even more by using a parallel parameter tuning method which calls the *HyParSVM* cross validation routine, e.g. on different nodes of a SMP cluster.

Our future work will be aimed at further improvement of the *HyParSVM* software. The shared memory parallelization of the training routine will be enhanced and tested for larger data sets. We will analyze which parallel scheme – *ESSLsmp* or OpenMP-based constructs – gives the best speedups. The influence of the working set size onto the scalability will be investigated.

References

1. Cristianini, N., Shawe-Taylor, J.: An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press (2000)
2. Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In: Proceedings of ECML-98, 10th European Conference on Machine Learning, Chemnitz, Germany, Springer Verlag (1998) 137–142
3. Yu, H., Yang, J., Wang, W., Han, J.: Discovering compact and highly discriminative features or feature combinations of drug activities using support vector machines. In: 2nd IEEE Computer Society Bioinformatics Conference (CSB 2003), Stanford, CA, USA, IEEE Computer Society (2003) 220–228
4. Vapnik, V.N.: Statistical learning theory. John Wiley & Sons, New York (1998)
5. Schölkopf, B.: The kernel trick for distances. In: NIPS. (2000) 301–307
6. Chen, N., Lu, W., Yang, J., Li, G.: Support vector machine in chemistry. World Scientific Pub Co Inc (2004)
7. Thrun, S., Mitchell, T.M.: Learning one more thing. In: IJCAI. (1995) 1217–1225
8. Eitrich, T., Lang, B.: On the advantages of weighted L_1-norm support vector learning for unbalanced binary classification problems. Preprint FZJ-ZAM-IB-2005-16, Research Centre Jülich (2005) submitted.
9. Eitrich, T., Lang, B.: Efficient optimization of support vector machine learning parameters for unbalanced datasets. JCAM (2005) in press.
10. Chapelle, O., Vapnik, V.N., Bousquet, O., Mukherjee, S.: Choosing multiple parameters for support vector machines. Machine Learning **46**(1) (2002) 131–159
11. Serafini, T., Zanghirati, G., Zanni, L.: Gradient projection methods for quadratic programs and applications in training support vector machines. Optimization Methods and Software **20**(2-3) (2005) 353–378
12. Skillicorn, D.: Strategies for parallelizing data mining. In: Proceedings of the Workshop on High-Performance Data Mining at IPPS/SPDP. (1998)

13. Parthasarathy, S., Zaki, M., Ogihara, M., Li, W.: Parallel data mining for association rules on shared-memory systems. Knowledge and Information Systems **3**(1) (2001) 1–29
14. Joshi, M.V., Karypis, G., Kumar, V.: ScalparC: a new scalable and efficient parallel classification algorithm for mining large datasets. In: IPPS: 11th International Parallel Processing Symposium, IEEE Computer Society Press (1998)
15. Hofer, J.: Distributed induction of decision tree classifier within the grid data mining framework: Gridminer-core. AURORA Technical Report 2004-04, Institute for Software Science, University of Vienna, Vienna (2004)
16. Kantabutra, S., Couch, A.L.: Parallel k-means clustering algorithm on NOWs. NECTEC Technical Journal **1** (2000) 243–248
17. Callahan, P.B.: Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In: Proceedings of the 34th Symp. Foundations of Computer Science, IEEE. (1993) 332–340
18. Misra, M.: Parallel environments for implementing neural networks. Neural Computing Surveys **1** (1997) 48–60
19. Jin, R., Yang, G., Agrawal, G.: Shared memory parallelization of data mining algorithms: techniques, programming interface, and performance. IEEE Transactions on Knowledge and Data Engineering **17**(1) (2005) 71–89
20. Schapire, R.E.: A brief introduction to boosting. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence. (1999)
21. Lazarevic, A., Obradovic, Z.: The distributed boosting algorithm. In: KDD '01: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining, New York, NY, USA, ACM Press (2001) 311–316
22. Graf, H.P., Cosatto, E., Bottou, L., Dourdanovic, I., Vapnik, V.: Parallel support vector machines: the cascade SVM. In: Advances in Neural Information Processing Systems 17. MIT Press, Cambridge, MA (2005) 521–528
23. Dong, J.X., Suen, C.Y.: A fast SVM training algorithm. International Journal of Pattern Recognition **17**(3) (2003) 367–384
24. Yu, H., Yang, J., Han, J.: Classifying large data sets using SVMs with hierarchical clusters. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, ACM (2003) 306–315
25. Dong, J.X., Krzyzak, A., Suen, C.Y.: A fast parallel optimization for training support vector machines. In Perner, P., Rosenfeld, A., eds.: Proceedings of 3rd International Conference on Machine Learning and Data Mining. (2003) 96–105
26. Poulet, F.: Multi-way distributed SVM algorithms. In: Proc. of ECML/PKDD 2003 Int. Workshop on Parallel and Distributed Algorithms for Data Mining. (2003)
27. Celis, S., Musicant, D.R.: Weka-parallel: machine learning in parallel. Computer Science Technical Report 2002b, Carleton College (2002)
28. Eitrich, T., Lang, B.: Parallel tuning of support vector machine learning parameters for large and unbalanced data sets. In: Computational Life Sciences (CompLife 2005). Volume 3695 of LNCS., Springer (2005) 253–264
29. Qiu, S., Lane, T.: Parallel computation of RBF kernels for support vector classifiers. In: SDM. (2005)
30. Serafini, T., Zanghirati, G., Zanni, L.: Parallel decomposition approaches for training support vector machines. In: Proceedings of ParCo2003, Elsevier (2004) 259–266
31. Eitrich, T., Lang, B.: Shared memory parallel support vector machine learning. Preprint FZJ-ZAM-IB-2005-11, Research Centre Jülich (2005) submitted.
32. IBM: ESSL - engineering and scientific subroutine library for aix version 4.1 (2003)
33. Detert, U.: Introduction to the JUMP architecture. (2004)