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

Technical Report

Shared Memory Parallel Support Vector Machine Learning

*Tatjana Eitrich, Bruno Lang**

FZJ-ZAM-IB-2005-11

September 2005

(last change: 21.9.2005)

Preprint: submitted for publication

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

Shared Memory Parallel Support Vector Machine Learning

Tatjana Eitrich t.eitrich@fz-juelich.de
John von Neumann Institute for Computing
Central Institute for Applied Mathematics
Research Centre Juelich, Germany

Bruno Lang Bruno.Lang@math.uni-wuppertal.de
Applied Computer Science and Scientific Computing Group
Department of Mathematics
University of Wuppertal, Germany

Editor:

Abstract

The increasing amount of data used for classification, as well as the demand for complex models with a large number of well tuned parameters, naturally lead to the search for efficient approaches making use of massively parallel systems. We describe the parallelization of support vector learning for shared memory systems. Our learning algorithm relies on a decomposition scheme, which in turn uses a special variable projection method, for solving the quadratic program associated with support vector machine learning. By using hybrid parallel programming, our parallelization approach can be combined with the parallelism of a distributed cross validation routine and parallel parameter optimization methods.

Keywords: Support Vector Machine Training, Shared Memory Parallel Computing, Large Data

1. Introduction

Support vector machines (SVMs) are important and well-known state-of-the-art machine learning methods for classification and regression. Classification is one of the most important tasks of data mining in our days. Given a training set with attributes the goal is to learn a model that later on can be used to classify unseen data in a reliable way. Much work has been done to apply support vector learning to challenging classification problems in pharmaceutical research, text mining, and many other application areas (Inoue and Abe, 2001; Kless and Eitrich, 2004; Han et al., 2003; Markowetz, 2001; Joachims, 1998; Lanckriet et al., 2004; Yu et al., 2003).

While the SVM theory is widely accepted, there still seems to be some gap between the theoretical framework given by learning theory, and the real world data to be classified (Hettich et al., 1998). For one thing, most current SVM models suffer from large, noisy and unbalanced data, and therefore the development of more robust algorithms remains an important topic for research. In addition, the data sets are becoming increasingly large, and therefore parallel processing is essential to provide the performance required by large-scale data mining tasks. The latter issue is addressed in the present paper.

Various work on parallel data mining for distributed memory systems has been done (Dhillon and Modha, 2000). Currently more and more machines are becoming available

with either global shared memory or with multi-processor shared memory nodes that are connected with some network. In Jin and Agrawal (2002) and Zaki et al. (1999) the authors present parallelization techniques for data mining algorithms on shared memory systems. Methods like decision trees, nearest neighbors and artificial neural networks have been analyzed and parallelized. Our intention is to broaden this work by the development of a parallel support vector machine for multi-processor shared memory (SMP) clusters.

The remainder of the paper is organized as follows. In Sections 2 and 3 we briefly review the basic concepts of support vector learning and our hierarchical SVM training method, as far as these issues are essential for understanding the following material. Section 4 first gives a run time analysis for the serial case, which motivates our shared memory parallelization scheme presented thereafter. Experimental results are given in Section 5. Section 6 contains a summary and points to directions for future work.

2. Support Vector Learning

Support vector learning means to determine functions that can be used to classify data points. To simplify the exposition we will discuss only binary classification. In the linear case the so-called reference data of given input–output pairs (training data)

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

are taken to find an optimal separating hyperplane (Cristianini and Shawe-Taylor, 2000; Schölkopf and Smola, 2002)

$$f_{\mathrm{l}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b = 0.$$

Using assumptions of statistical learning theory the desired classifier is then defined as

$$h(\boldsymbol{x}) = \begin{cases} +1, & \text{if } f_{\mathrm{l}}(\boldsymbol{x}) \geq 0, \\ -1, & \text{if } f_{\mathrm{l}}(\boldsymbol{x}) < 0, \end{cases}$$

with the linear decision function f_{l}. If the two classes are not linearly separable then f_{l} is replaced with a nonlinear decision function

$$f_{\mathrm{nl}}(\boldsymbol{x}) = \sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b,$$

where $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a (nonlinear) kernel function. Here the classification parameters α_i and b can be obtained as the unique global solution of a suitable (dual) quadratic optimization problem (Schölkopf and Smola, 2002)

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^l} \quad g(\boldsymbol{\alpha}) := \frac{1}{2} \boldsymbol{\alpha}^T H \boldsymbol{\alpha} - \sum_{i=1}^{l} \alpha_i \tag{1}$$

with $H \in \mathbb{R}^{l \times l}$, $H_{ij} = y_i K(\boldsymbol{x}_i, \boldsymbol{x}_j) y_j$ $(1 \leq i, j \leq l)$, constrained to

$$\boldsymbol{\alpha}^T \boldsymbol{y} = 0, \quad 0 \leq \alpha_i \leq C.$$

The Hessian H is usually dense, and therefore the complexity of evaluating the objective function g in (1) scales quadratically with the number l of training pairs, leading to very time-consuming computations. The parameter C controls the trade-off between the width of the classifier's margin and the number of weak and wrong classifications on the training set. This parameter has to be chosen by the user. We refer to Eitrich and Lang (2005a) for details on the extension of the basic model leading to a weighted approach, which can be adjusted to highly unbalanced data sets.

In Eitrich and Lang (2005a) we also have introduced a so-called multi-parameter Gaussian kernel

$$K^{\mathrm{M}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\sum_{k=1}^{n} \frac{(\boldsymbol{x}_i(k) - \boldsymbol{x}_j(k))^2}{2\sigma_k^2}\right) \tag{2}$$

with a different width σ_k for each feature. Comparing it with the usual single-width kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right), \tag{3}$$

experiments showed that our new kernel yields better classification results if the significance of the features in the data varies heavily. The price for the improved quality is a slower kernel computation: while using (3) requires a simple norm computation and a single division for each entry of H, (2) involves divisions in the inner loop. Since kernel computations account for a major part of the overall training time, the SVM training is slowed down perceptibly.

3. A Hierarchical Approach for Support Vector Machine Training

Support vector machine training corresponds to solving (1), which is a quadratic problem (qp) with simple constraints. A well-known method for the solution of such problems is the decomposition algorithm (Hsu and Lin, 2002), where each iteration consists of four steps:

1. select a working set of $\tilde{l}$ "active" variables from the l free variables α_i,
2. solve the size-$\tilde{l}$ quadratic subproblem that results from restricting the optimization in (1) to the active variables and fixing the remaining variables,
3. update the global solution $\boldsymbol{\alpha}$, and
4. check a stopping criterion.

One advantage of the decomposition method is its flexibility concerning the size $\tilde{l} \leq l$ of the subproblems. In our implementation we rely on the usual working set selection scheme that uses a method of feasible directions originally described in Zoutendijk (1960). In contrast to other implementations our convergence criterion is not based on the fulfillment of the Karush–Kuhn–Tucker conditions but on the number of pairs given by the working set selection method (Eitrich and Lang, 2005a).

The overall SVM training time depends heavily on the efficiency of the qp solver for the resulting subproblems. Here we use an own implementation of the generalized variable projection method introduced in Serafini et al. (2005). This method again defines subproblems (with diagonal matrices) and solves these iteratively with a very fast inner solver (Pardalos and Kovoor, 1990). Figure 1 summarizes the resulting hierarchical algorithm for the SVM training.

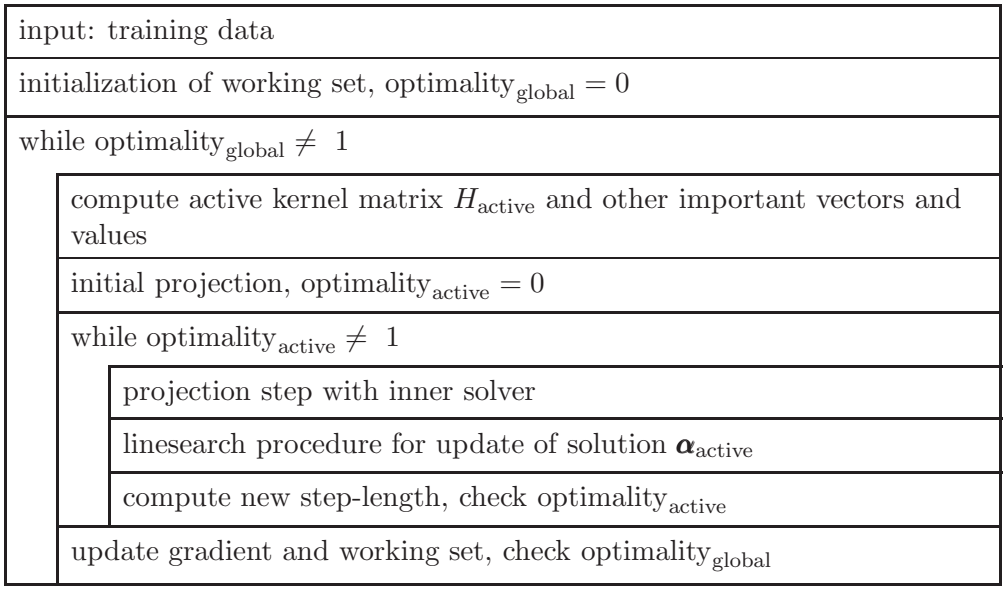

Figure 1: Structure chart (slightly simplified) for the hierarchical SVM training with the decomposition scheme (main loop) and the projection based qp solver (inner loop).

4. Shared Memory Parallelization of the SVM Training

There are three ways to insert parallelism into SVM methods: parallelizing the training of a single SVM, training several SVMs in parallel, and using a parallel algorithm for optimizing the learning parameters, such as C and the σ_k. The second of these options has been addressed with mostly straight-forward approaches, for example, parallel mixture of SVMs (Collobert et al., 2002), parallel training of binary SVMs for multiclass problems (Selikoff, 2003), parallel training of SVMs on splitted data (Graf et al., 2005) and parallel cross validation models (Celis and Musicant, 2002). Parallel parameter optimization has been discussed in Runarsson and Sigurdsson (2004) and Eitrich and Lang (2005b). Concerning the first option, a promising technique for parallelizing the SVM training on distributed memory systems has been described in Zanghirati and Zanni (2003). It uses standard C and MPI communication routines.

We believe that shared memory parallelization is better suited for speeding up the training of a single SVM, for two reasons. First, most of today's high-end machines are built from "fat nodes," each of which contains multiple processors with access to a rather large shared memory. Thus, while it is possible to use all processors of such machines under the message passing paradigm, a hybrid distributed/shared memory parallelization is more natural, relying on message passing for very coarse-grained parallelism (training of several SVMs in parallel, parameter optimization), and running the training of each single SVM with its finer grained parallelism on a few processors within a shared memory node. Thus, the shared memory parallelization needs not scale to high numbers of processors. Note that

shared memory multi-processors are becoming increasingly popular even in workstations and PCs, in particular in the form of multi-core processors offering functionality of multiple CPUs on a single chip. The second reason for preferring shared memory parallelization is its ease of use. The hierarchical training algorithm is quite involved and comprises dozens of routines with several thousands of lines of code. Distributing the data affects almost all of these routines, whereas shared memory parallelization can be done incrementally, addressing the performance hot spots one-by-one until a satisfactory speedup is reached.

Our numerical experiments were made on the Juelich Multi Processor (JUMP) at Research Centre Juelich (Detert, 2004). 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. All in all the 1312 processors have an aggregate peak performance of 8.9 TFlop/s.

4.1 Performance Characteristics of Serial SVM Training

In order to obtain a highly efficient parallel SMV training routine, the serial version must be optimized as well. That is, in addition to using adequate algorithms, the computations must be performed at maximum speed. This can be achieved in a portable way by relying on the Level–1 and 2 Basic Linear Algebra Subroutines (BLAS) (Lawson et al., 1979; Dongarra et al., 1988), which comprise routines for computing norms (*DNORM*) or linear combinations of vectors (*DAXPY*), matrix–vector products (*DGEMV*), and others. While the calling sequences for the BLAS routines are standardized, most vendors provide optimized implementations for their machines, so that optimum performance can be obtained by simply linking with an appropriate library. In the case of IBM, their Engineering Scientific Subroutine Library (ESSL) includes Power4-tuned versions of the BLAS and of many other basic mathematical operations. We have also made use of the latter, where appropriate (for example, *DYAX*, *DVEA*, *DVES*).

overall time	1672.0
time for decomposition (A)	97.4
time for kernel evaluations (B)	644.9
time for projection (C)	0.4
time for inner solver (D)	9.6
sum of times for ESSL routines (E)	912.1

Table 1: Run time analysis for a serial SVM training (20000 instances, working set size 10000).

We used the GNU profiler *gprof* to determine the main computational bottlenecks during SVM training. In Table 1 the individual costs are given for the decomposition routine (A), the evaluations of the multi-parameter kernel function (B), the variable projection method (C), the inner solver (D), and all calls to ESSL routines (E). Please note that these times comprise only the computations carried out *within* the respective routines and thus do not include calls to lower-level routines. This is important since (E) is called from (A), (C) and

(D), (D) is called from (C), (C) and (B) are called from (A). The timings were obtained for a data set with $l = 20000$ instances and $n = 10$ features and a working set size of 10000.

The left picture in Figure 2 shows the corresponding call graph, the right picture gives a more detailed view on the most time-consuming non-ESSL routines.

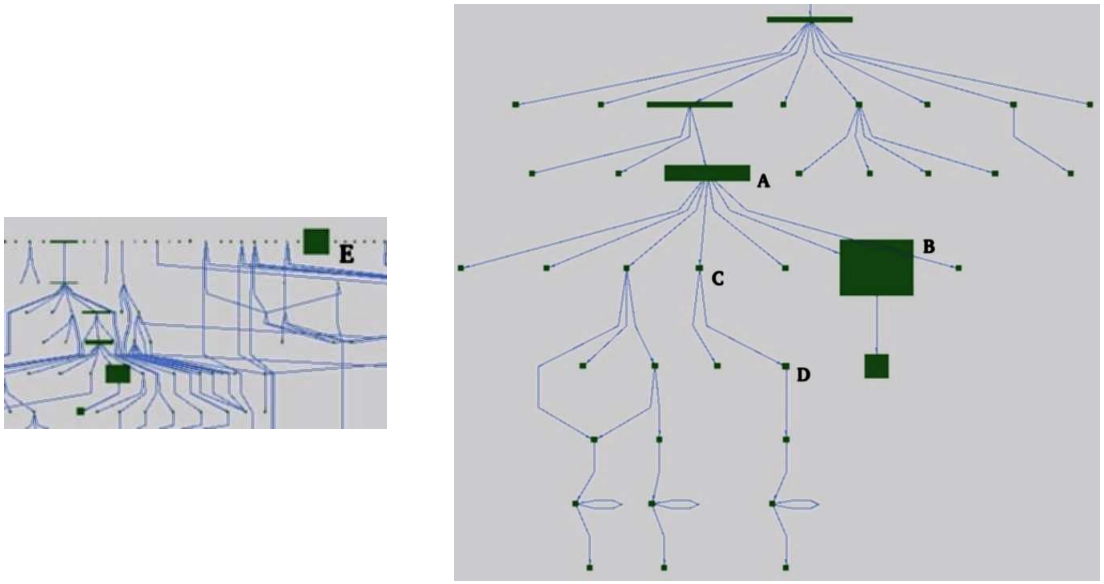

Figure 2: Call graph for the SVM training (left) and detailed view excluding ESSL calls (right).

Depending on the dimensions of the data and on the working set sizes the relative contributions of (A) through (E) to the overall time may vary, but for complex models on large data the vast majority of time is always consumed by

- the kernel function evaluations in the decomposition routine and
- the matrix–vector multiplications called from the qp solver.

By contrast, the decomposition and projection steps themselves (not counting their calls to ESSL routines) and the inner solver do not consume a significant amount of time.

Before we discuss the parallelization we briefly touch on the issue of choosing a reasonable working set size, which affects the serial and parallel performance of the SVM training.

4.2 The Influence of the Working Set Size

The size of the working set, $\tilde{l}$, influences the SVM training time in two ways. Small working sets lead to fast qp solver computations, but to a large number of decomposition steps and therefore to an increasing number of kernel function evaluations. Large working sets, on the other hand, tend to reduce the number of decomposition steps, but they slow down each solution of a qp subproblem since the matrix–vector multiplications become more expensive.

The decomposition routine includes two updates of kernel matrices, the so-called active kernel matrix $H_{\text{active}} \in \mathbb{R}^{\tilde{l} \times \tilde{l}}$ that defines the quadratic subproblem to be solved and the

mixed kernel matrix $H_{\text{mixed}} \in \mathbb{R}^{\tilde{l} \times (l-\tilde{l})}$ that is necessary for the gradient update. Thus, the number N of kernel evaluations for SVM training with the decomposition technique is given by

$$N = D \cdot \tilde{l}^2 + D \cdot \tilde{l} \cdot \left(l - \tilde{l}\right) = D \cdot \tilde{l} \cdot l,$$

where l and D denote the number of training points and the number of decomposition steps, respectively. As pointed out above, the number of decomposition steps decreases if larger working sets are used, but no explicit formula can be given because D also depends on the working set selection scheme and on the parameters controlling convergence.

data set with 2000 points and 10 features				
working set size	200	600	1000	1400
decomposition steps	40	14	8	6
millions of kernel function evaluations	16.0	16.8	16.0	16.8
time in seconds	6.9	7.6	7.5	8.0
data set with 10000 points and 10 features				
working set size	1000	3000	5000	7000
decomposition steps	63	18	10	6
millions of kernel function evaluations	630.0	540.0	500.0	420.0
time in seconds	276.0	255.2	293.1	322.6

Table 2: Training times for different working set sizes.

In Table 2 we compare experimental results obtained by varying $\tilde{l}$ in the SVM training on two different data sets. The smaller data set consists of 2000 and the larger data set of 10000 instances; both have 10 features. In both cases the decreasing number of decomposition steps for larger working sets is compensated by slower inner solver operations, so that the overall time is almost invariant with respect to $\tilde{l}$. Note the huge number of kernel function evaluations, in particular for the large data set.

4.3 Mixed Library/Loop Parallelization

The performance analysis in Sect. 4.1 marks the ESSL operations and the kernel matrix computations as the primary targets for an incremental shared memory parallelization.

Since IBM also provides a shared memory parallel version of the ESSL, it is trivial to achieve multi-processor execution for these routines by simply linking to another version of the library. This does not require any changes in the source code.

The remaining parallelization was done using OpenMP (OpenMP Architecture Review Board, 1999). OpenMP is a standardized API defining a set of Fortran compiler directives (or C pragmas), library routines, and environment variables that can be used to describe and exploit shared memory parallelism. The directives allow the programmer to mark areas of the code, the so-called "parallel regions", that are suitable for parallel processing. On entering a parallel region, additional threads are created (or available active threads are bound to the "master thread"), and they are freed again when the parallel region is left. The statements within the parallel region are executed by all threads, except for

those statements occurring in so-called work-sharing constructs, such as parallel loops, etc. Only these constructs lead to true parallelism. For example, the passes of parallel loops are distributed *at run-time* to all available threads. The scheduling mechanism and the number of threads can be controlled via environment variables. Since all threads originating from a single process share the latter's memory space, OpenMP also provides directives for defining which variables should be accessible to all threads and for which variables each thread should have a private copy in order to avoid write conflicts. The directives are written as a special kind of comment, and thus the same program can easily be run in serial or parallel mode on a given computer, or even on a computer that does not have an OpenMP-aware compiler at all.

The most important parallel OpenMP loops in our decomposition scheme compute the active kernel matrix $H_{\text{active}} \in \mathbb{R}^{\tilde{l} \times \tilde{l}}$ and the mixed kernel matrix $H_{\text{mixed}} \in \mathbb{R}^{\tilde{l} \times (l-\tilde{l})}$ by assigning a set of columns of the matrix to each thread. We also parallelized loops in the decomposition routine, the variable projection method, and the inner solver. The latter is not very important because the algorithm of Pardalos and Kovoor (1990) is extremely fast even in serial mode.

Note that in our mixed approach the parallelism is provided alternatingly by work-sharing constructs within parallel OpenMP regions and by the shared memory parallelized ESSL routines, implying frequent re-binding of the active threads.

Up to now our focus has been on speeding up the learning step itself. Thus, routines used for the validation (for example, counting the number of training errors and the number of support vectors) and also for the classification of unseen data have not been parallelized yet. The mixed library/loop-based parallelization approach seems appropriate for these computations as well.

5. Experimental Evaluation

We have tested the parallel SVM training method together with our multi-parameter kernel for several data sets with varying numbers of instances, numbers of features, and working set sizes. Table 3 summarizes the results for two training sets with $l = 20000$ points and $n = 10$ or $n = 15$ features. Three different working set sizes corresponding to 25%, 50%, and 75% of the training points have been used to show the influence of the subproblem sizes upon the serial and parallel overall time. These working sets are significantly larger than those used in the work of Zanghirati and Zanni (2003). The speedups are satisfactory for all numbers of threads. Some of the speedups even exceed the respective numbers of threads in use. These super-linear speedups are not yet fully understood, but probably they are caused by cache effects. Sometimes the speedups decrease for increasing numbers of threads. This is partly due to having to share the nodes (and in particular their memory bandwidth) with other users. We observed performance fluctuations up to 10% for other applications, too. To mitigate these effects, most of the tests have been run several times, and the times reported in the table are those of the fastest respective runs.

working set size	5000		10000		15000	
features	10	15	10	15	10	15
serial	2066 : —	2259 : —	1672 : —	2425 : —	1525 : —	2618 : —
2 threads	787 : 2.6	930 : 2.4	710 : 2.4	821 : 3.0	700 : 2.2	1013 : 2.6
3 threads	874 : 2.4	973 : 2.3	525 : 3.2	571 : 4.2	458 : 3.3	762 : 3.4
4 threads	430 : 4.8	589 : 3.8	356 : 4.7	441 : 5.5	363 : 4.2	494 : 5.3
5 threads	379 : 5.5	382 : 5.9	339 : 4.9	372 : 6.5	312 : 4.9	500 : 5.2
6 threads	310 : 6.7	339 : 6.7	341 : 4.9	448 : 5.4	295 : 5.2	456 : 5.7
7 threads	295 : 7.0	333 : 6.8	316 : 5.3	285 : 8.5	276 : 5.5	385 : 6.8
8 threads	254 : 8.1	354 : 6.4	226 : 7.4	305 : 8.0	270 : 5.6	331 : 7.9

Table 3: Comparison of (training time in seconds : speedup) for different scenarios.

6. Conclusions and Future Directions

We have discussed a mixed library/loop-based shared memory parallelization for SVM training. Numerical experiments show that our approach can yield rather satisfactory speedups for moderate numbers of processors, as available, for example, in high-performance workstations and PCs. On high-end machines with an SMP cluster architecture our shared memory parallelization can be complemented with message passing-based approaches to increase the level of parallelism in, for example, the training of multiple SVMs or the optimization of learning parameters. The potential of such hybrid strategies has to be investigated in the future.

Our implementation still includes some serial parts that limit the attainable speedup. For example the decomposition routine still comprises the serial time consuming working set selection method. If good speedups are desired for larger numbers of processors then the serial sections of the code have to be reduced further. In addition it is important to study the influence of the working set size on the behavior of the parallel ESSL routines.

Acknowledgments

The authors would like to thank the ZAM team at Juelich for generous technical support.

References

Sebastian Celis and David R. Musicant. Weka-parallel: machine learning in parallel. Computer Science Technical Report 2002b, Carleton College, 2002.

Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002.

Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning Methods*. Cambridge University Press, Cambridge, UK, 2000.

Ulrich Detert. *Introduction to the JUMP architecture*, 2004. http://jumpdoc.fz-juelich.de.

Inderjit S. Dhillon and Dharmendra S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, pages 245–260, 2000.

Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14(1): 1–17, March 1988.

Tatjana Eitrich and Bruno Lang. Efficient optimization of support vector machine learning parameters for unbalanced datasets. Preprint BUW-SC 2005/2, University of Wuppertal, May 2005a.

Tatjana Eitrich and Bruno Lang. Parallel tuning of support vector machine learning parameters for large and unbalanced data sets. Preprint BUW-SC 2005/7, University of Wuppertal, May 2005b.

Hans Peter Graf, Eric Cosatto, Léon Bottou, Igor Dourdanovic, and Vladimir Vapnik. Parallel support vector machines: the cascade svm. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 521–528. MIT Press, Cambridge, MA, 2005.

Hui Han, C. Lee Giles, Eren Manavoglu, Hongyuan Zha, Zhenyue Zhang, and Edward A. Fox. Automatic document metadata extraction using support vector machines. In *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*, pages 37–48, Washington, DC, USA, 2003. IEEE Computer Society.

Seth Hettich, Catherine L. Blake, and Christopher J. Merz. *UCI Repository of machine learning databases*, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Chih-Wei Hsu and Chih-Jen Lin. A simple decomposition method for support vector machines. *Machine Learning*, 46(1-3):291–314, 2002.

Takuya Inoue and Shigeo Abe. Fuzzy support vector machines for pattern classification. In *Proc. Intl. Joint Conf. Neural Networks (IJCNN'01)*, pages 1449–1454, 2001.

Ruoming Jin and Gagan Agrawal. Shared memory parallelization of data mining algorithms: techniques, programming interface, and performance. In *Proceedings of the Second SIAM International Conference on Data Mining*, Arlington, VA, 2002.

Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Chemnitz, Germany, 1998. Springer Verlag.

Achim Kless and Tatjana Eitrich. Cytochrome p450 classification of drugs with support vector machines implementing the nearest point algorithm. In Jesús A. López, Emilio Benfenati, and Werner Dubitzky, editors, *Knowledge Exploration in Life Science Informatics, International Symposium,KELSI 2004, Milan, Italy, November 25-26, 2004, Proceedings*, volume 3303 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2004.

Gert R. G. Lanckriet, Minghua Deng, Nello Cristianini, Michael I. Jordan, and William Stafford Noble. Kernel-based data fusion and its application to protein function prediction in yeast. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 300–311, 2004.

Charles L. Lawson, Richard J. Hanson, David R. Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Softw.*, 5(3):308–323, September 1979.

Florian Markowetz. Support vector machines in bioinformatics. Master's thesis, University of Heidelberg, 2001.

OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface, Version 2.0, 1999.

Panos M. Pardalos and Nainan Kovoor. An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. *Mathematical Programming*, 46 (3):321–328, 1990.

Thomas Philip Runarsson and Sven Sigurdsson. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing - Letters and Reviews*, 3(3):59–67, june 2004.

Bernhard Schölkopf and Alexander J. Smola. *Learning with kernels*. MIT Press, Cambridge, MA, 2002.

Scott Selikoff. The SVM-tree algorithm, 2003. http://scott.selikoff.net/papers/ CS678_-_Final_Report.pdf.

Thomas Serafini, Gaetano Zanghirati, and Luca Zanni. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods and Software*, 20(2-3):353–378, 2005.

Hwanjo Yu, Jiong Yang, Wei Wang, and Jiawei Han. 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), 11-14 August 2003, Stanford, CA, USA*, pages 220–228. IEEE Computer Society, 2003.

Mohammed Javeed Zaki, Ching-Tien Ho, and Rakesh Agrawal. Parallel classification for data mining on shared-memory multiprocessors. In *ICDE*, pages 198–205, 1999.

Gaetano Zanghirati and Luca Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Computing*, 29(4):535–551, 2003.

Guus Zoutendijk. *Methods of feasible directions: a study in linear and non-linear programming*. Elsevier, 1960.