International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:1, No:10, 2007

On the Efficient Implementation of a Serial and Parallel Decomposition Algorithm for Fast Support Vector Machine Training Including a Multi-Parameter Kernel

Tatjana Eitrich, Bruno Lang

Abstract**—** This work deals with aspects of support vector machine learning for large-scale data mining tasks. Based on a decomposition algorithm for support vector machine training that can be run in serial as well as shared memory parallel mode we introduce a transformation of the training data that allows for the usage of an expensive generalized kernel without additional costs. We present experiments for the Gaussian kernel, but usage of other kernel functions is possible, too. In order to further speed up the decomposition algorithm we analyze the critical problem of working set selection for large training data sets. In addition, we analyze the influence of the working set sizes onto the scalability of the parallel decomposition scheme. Our tests and conclusions led to several modifications of the algorithm and the improvement of overall support vector machine learning performance. Our method allows for using extensive parameter search methods to optimize classification accuracy.

Keywords**—** Support Vector Machine Training, Multi-Parameter Kernels, Shared Memory Parallel Computing, Large Data

I. Introduction

DURING the last years data mining tasks have shifted from applications on small data sets to large-scale problems usually with a lot of noise and missing values in the data. At the same time industry requires complex models with well tuned parameters and promising results for test data and the new data to be classified.

Support vector machines (SVMs) for classification and regression are powerful methods of supervised machine learning. They have been widely studied and applied to hard, but mostly small classification problems. Applications include fields like QSAR modeling [1] and text classification [2]. SVMs belong to the so-called kernel methods [3]. They have excellent generalization properties, which means that the classification function is modeled in a way that makes it to work well on data that have not been used during the training. However, SVM training methods suffer from high computational complexity when used for large and noisy data. This particular issue is treated in this paper.

Important classical SVM research issues for all kinds of data sets include

- applicability [2], [4],
- generalization abilities [5],
- convergence properties [6],
- parameter selection methods [7], [8],
- interpretational aspects [9],

and many more.

In addition to these fundamental issues, the ability to handle problems of ever increasing size is of vital interest because in many applications the amount of data grows exponentially [10]. For these problems, SVM training time becomes a major concern, particularly when applying parameter selection methods that force the user to perform dozens or hundreds of SVM trainings. Due to extreme training times complex SVM models and intelligent parameter tuning have been employed only rarely, so that users often ended up with suboptimal classifiers and started to use other parameter free data mining methods with worse generalization properties.

For these reasons, research on efficient and fast SVM classification methods has been intensified during the last years, leading to approaches for

- fast serial training [11], [12],
- efficient parameter selection methods [13],
- fast multi-class learning [14], [15],
- parallel parameter tuning [16], [17],
- parallel validation methods [18], and
- parallel training methods [19], [20].

Issues of parallel support vector machines are comparatively new. They emerged during the last few years. Really parallel implementations are rare since most of the parallel algorithms realize simple farming jobs like parallel cross validation tasks. Farming reduces overall running time but is not able to improve the performance of a single SVM training.

In our work we now try to combine aspects of efficient SVM training techniques with parallelization. Based on a decomposition algorithm for SVM classifier design that can be run in serial and parallel mode we discuss modifications of the program flow that lead to significantly faster SVM training for large data sets, both in serial and parallel mode.

The paper is organized as follows. In Sect. II we review basics of binary SVM classification. In Sect. III we describe our serial and parallel SVM algorithm. Our computing system as well as the data set used for our experiments are introduced in Sect. IV. In Sect. V we discuss the influence of kernel computations onto training time and introduce our approach of data transformation for efficient usage of the powerful multi-

T. Eitrich is with the Central Institute for Applied Mathematics, Research Centre Juelich, Germany (e-mail: t.eitrich@fz-juelich.de).

B. Lang is with the Applied Computer Science and Scientific Computing Group, Department of Mathematics, University of Wuppertal, Germany (e-mail: lang@math.uni-wuppertal.de).

parameter Gaussian kernel. The issue of optimal working set selection for SVM training is discussed in Sect. VI.

II. The Support Vector Machine Learning Approach

Support vector machine learning [21], [22] is a well known and reliable data mining method. We consider the problem of supervised binary classification which means to use a training data set

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

to learn a binary decision function

$$h(\boldsymbol{x}) = \operatorname{sgn}\left(f(\boldsymbol{x})\right),$$

where we define the signum function in a modified form as

$$\operatorname{sgn}(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases} \quad (a \in \mathbb{R}).$$

The real-valued nonlinear classification function f is defined as [23]

$$f(\boldsymbol{x}) = \sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b\ . \tag{1}$$

In Fig. 1 a linear learning problem is shown. The margin is the value of the maximal separation between the classes.

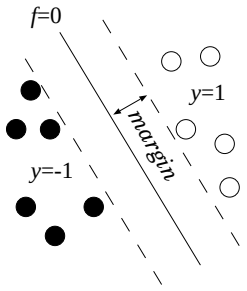

Fig. 1. Linear separation of the labeled training data by using the hyperplane $f = 0$.

The kernel function K [24], which can be interpreted as a local measure of similarity between training points, is used to avoid a so-called feature mapping of the data and to construct a nonlinear classifier based on a simple linear learning approach. In this work we analyze the so-called Gaussian kernel, which is very popular. A definition of this kernel will be given in Sect. V.

The problem specific classification parameters $\boldsymbol{\alpha} \in \mathbb{R}^l$ and $b \in \mathbb{R}$ of (1) are given implicitly through the training data and some SVM specific learning parameters [25]. These learning parameters, e.g. the kernel type and its parameter(s), have to be set before training and can be adjusted via parameter optimization [16].

It is well known [23] that the optimal vector $\boldsymbol{\alpha}^*$ can be computed via the solution of the dual quadratic program (QP)

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

with

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

constrained to

$$\begin{aligned} &\boldsymbol{\alpha}^T \boldsymbol{y} = 0, \\ &0 \leq \alpha_i \leq C \quad (i = 1, \ldots, l), \\ &\alpha_i \Big(y_i \cdot f(\boldsymbol{x}_i) - 1 + \xi_i \Big) = 0 \quad (i = 1, \ldots, l), \end{aligned}$$

the latter constraints resulting from the evaluation of the Karush–Kuhn–Tucker conditions [23]. The vector $\boldsymbol{\xi} \in \mathbb{R}^l_+$ of so called slack variables belongs to the primal problem of (2) and reflects the 1-norm soft margin approach for SVM learning, which is used in most of the available software packages to allow for training errors [26]–[29].

The parameter $C > 0$ is important for a natural weighting between the competing goals of training error minimization and generalization. For details we refer to the work [23] and the tutorial [30].

The computation of the threshold b^* is based on the so-called Karush–Kuhn–Tucker conditions [23] for the primal form of the problem (2). Given the unique and global dual solution $\boldsymbol{\alpha}^*$ (the vector of Lagrange multipliers) it is easy to show that

$$\begin{aligned} 0 &= \alpha_i^* \cdot (y_i \cdot f(\boldsymbol{x}_i) + \xi_i^* - 1) \\ 0 &= \xi_i^* \cdot (\alpha_i^* - C) \end{aligned}$$

hold for all $i = 1, \ldots, l$. Since we solve the dual problem, the slack values ξ_i^* are unknown. Therefore we have to use the so called nonbound support vectors to compute b^*. A nonbound support vector $\boldsymbol{x}_i$ is characterized by $\alpha_i^* \in (0, C)$ and must have a zero slack value. See [23] for detailed information on slack variables, support vectors and bounds. Using (1) we derive

$$0 = \alpha_i^* \left(y_i \left(\sum_{j=1}^{l} \alpha_j^* y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + b^* \right) + \xi_i - 1 \right)\ .$$

for all training points and thus

$$b^* = y_i - \sum_{j=1}^{l} \alpha_j^* y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

for all nonbound support vectors $\boldsymbol{x}_i$.

Note that (2) is a quadratic optimization problem with a dense matrix H. For large data the solution of this problem—the so called training stage—is very expensive. In the following section we shortly describe our efficient SVM training method which provides a serial and a parallel mode.

III. Efficient Serial and Parallel SVM Training

This work is based on the SVM training method described in [31]. We briefly review the most important features of the serial and parallel implementations.

We are working with the well known decomposition scheme [32] for the solution of (2). This scheme is summarized in Fig. 2. It repeatedly performs the following four steps.

1) Select $\hat{l}$ "active" variables from the l free variables α_i, the so-called working set. In our implementation the working set is made up from points violating the Karush–Kuhn–Tucker conditions; see [25] for more details.
2) Restrict the optimization in (2) to the active variables and fix the remaining ones. Prepare the submatrix
$$H_{\text{active}} \in \mathbb{R}^{\hat{l} \times \hat{l}}$$
for the restricted problem and the submatrix
$$H_{\text{mixed}} \in \mathbb{R}^{(l-\hat{l}) \times \hat{l}}$$
for the stopping criterion.
3) Check for convergence. The solution of (2) is found if step 1 yields an empty working set.
4) Solve the restricted problem.

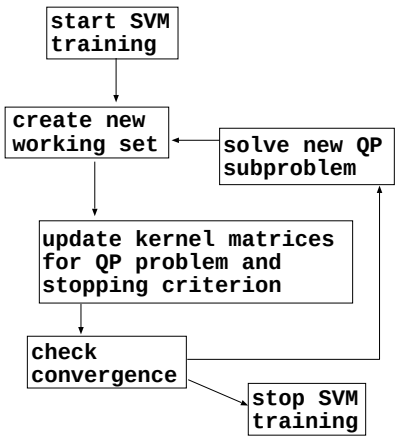

Fig. 2. Iterative decomposition scheme for the serial SVM training with a flexible working set size.

The idea of splitting the quadratic problem into active and inactive parts iteratively is not new [33]. One feature that makes this approach particularly attractive for SVM training is the flexibility concerning the size $\hat{l}$. Large values of $\hat{l}$ place high demands on memory because $\hat{l}$ columns of H (i.e., $\hat{l} \cdot l$ entries) must be stored. In the extreme case $\hat{l} = l$, the whole matrix H is required. On the other hand, choosing $\hat{l} < l$ may lead to kernel values being re-computed several times when a variable α_i switches between the "inactive" and "active" states. Therefore most SVM software packages avoid recomputation of the columns of H. They implement caching of kernel values to speed up training time. For example, [28] uses the well-known least-recently-used cache strategy. Unfortunately, the caching strategies are difficult and system dependent.

For complex learning models on large data a huge amount of time is consumed by the kernel function evaluations in the decomposition step, where the kernel matrices are updated in every iteration. It is known [31] that training time is a function of the working set size $\hat{l}$ that acts as a mediator between the alternating work in the outer decomposition loop and the inner solver. Large working sets slow down the solution of each quadratic subproblem, whereas small working sets lead to a large number of decomposition iterations until convergence is reached, which means that a lot of kernel function evaluations take place.

The SVM training time also depends on the efficiency of the algorithm that solves the subproblems. We use the generalized variable projection method introduced in [34] as an inner solver.

Usually small working sets have been used to avoid expensive subproblems [35]. However, our powerful computing systems now allow for very large working sets. Thus we have to determine the optimal value for $\hat{l}$ that minimizes the sum of times for inner solver computations and decomposition workload.

One way to improve the performance of SVM training is parallelization. Our parallel SVM training method does not implement a simple farming approach, but a real parallel flow. It is based on the observation [31] that typically more than 90% of the overall time is spent in the kernel evaluations and in the matrix–vector and vector–vector operations of the inner solver; for very large data sets this fraction is even higher. We decided to address these computational bottlenecks with a shared memory parallelization, using OpenMP work sharing for the kernel computations and relying on the parallelized numerical linear algebra kernels available in the *ESSLSMP* library for the compute-intensive parts of the inner solver. Fig. 3 shows the parallel parts (shaded) of the decomposition scheme.

However, in order to achieve optimum performance, the parallelization should be complemented with techniques that reduce the learning time also in the serial case. In Sect. V and VI we will discuss our approaches for faster SVM training and their results.

IV. Characteristics of Data and Computing System

For all tests reported here we used the so-called adult data set from [36], which is the data set with the largest number of training instances in the database. The task for this set is to predict whether someone's income exceeds a certain threshold. Thus we have a binary classification problem. The number of training points is 32561. Out of the 14 attributes, 6 are continuous, and 8 are discrete. There are plenty of missing

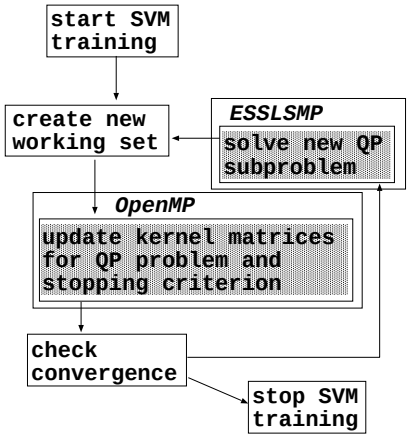

Fig. 3. Extension of the serial algorithm for shared memory parallel SVM training.

values for the discrete attributes. These were replaced with either the value that occurred most frequently for the particular attribute or with a new value, if the number of missing values for the attribute was very high.

The adult data set was also used in [37], but only for 16000 training points. There, a new parallel MPI based SVM learning method for distributed memory systems has been described. We used nearly all points for the training, i.e., 30000.

Our serial and parallel experiments were made on the Juelich Multi Processor (JUMP) at Research Centre Juelich [38]. 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. We have tested on a single node of JUMP. Test results are given in the following two sections.

V. Efficient Kernel Evaluations

As we discussed in Sect. I the training of support vector machines on large data is a challenging problem [35], [39]. A vast amount of time is always consumed by the expensive kernel function evaluations [31], no matter which kernel type is used.

In this section we present our new approach of efficient kernel evaluations that includes the usage of a multi-parameter kernel.

The usual Gaussian kernel [21]

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

which is used in many data analysis tools, includes a single division operation for each kernel function evaluation. $\sigma > 0$ is the constant width of the kernel. This parameter is central for SVM learning with the Gaussian kernel. It has to be chosen carefully to avoid overfitting effects.

The division operation in (3) can be replaced with a less expensive multiplication by setting

$$\tilde{\sigma} = \frac{1}{2\cdot\sigma^2}$$

once already before SVM training and evaluating the kernel as

$$K^g(\boldsymbol{x},\boldsymbol{z}) \quad = \quad \exp\left(-\tilde{\sigma}\|\boldsymbol{x}-\boldsymbol{z}\|^2\right). \tag{4}$$

This simple modification is not possible for the generalized, multi-parameter Gaussian kernel [25]

$$K^g_{\mathrm{M}}(\boldsymbol{x},\boldsymbol{z}) \quad = \quad \exp\left(-\sum_{k=1}^{n}\frac{(x_k-z_k)^2}{2\sigma_k^2}\right). \tag{5}$$

This kernel assigns a different width $\sigma_k \geq 0$ for each feature k $(k = 1,\ldots,n)$. For unbalanced data sets this kernel can lead to significantly better SVMs than the standard kernel; cf. [16]. Unfortunately the n divisions make this kernel rather expensive and thus responsible for long SVM training times. Therefore it is used only rarely [16].

It is possible to avoid all parameter-dependent operations inside the kernel function. To this end we first rewrite the kernel (3) as

$$K^g(\boldsymbol{x},\boldsymbol{z}) \quad = \quad \exp\left(-\sum_{k=1}^{n}\left(\frac{x_k-z_k}{\sqrt{2}\sigma}\right)^2\right).$$

Thus, an initial scaling of the training points according to

$$t(\boldsymbol{x}) := \frac{\boldsymbol{x}}{\sqrt{2}\sigma} \tag{6}$$

allows the standard kernel to be evaluated as

$$K^g(\boldsymbol{x},\boldsymbol{z}) \quad = \quad \exp\left(-\|t(\boldsymbol{x})-t(\boldsymbol{z})\|^2\right). \tag{7}$$

Similarly, the scaling

$$\tilde{t}(\boldsymbol{x}) \quad := \quad \left(\frac{x_1}{\sqrt{2}\sigma_1},\ldots,\frac{x_n}{\sqrt{2}\sigma_n}\right), \tag{8}$$

leads to

$$K^g_{\mathrm{M}}(\boldsymbol{x},\boldsymbol{z}) \quad = \quad \exp\left(-\|\tilde{t}(\boldsymbol{x})-\tilde{t}(\boldsymbol{z})\|^2\right). \tag{9}$$

Note that in this formulation the generalized and the standard kernel differ only in the initial transformation of the data. The transformation step has to be done before SVM training and is independent of $\hat{l}$ and other settings of the decomposition method.

Usage of our multi-parameter approach is not limited to the Gaussian kernel. For example, the generalized polynomial kernel of degree $d \in \mathbb{N}_+$

$$K^p_{\mathrm{M}}(\boldsymbol{x},\boldsymbol{z}) \quad = \quad \left(1+\sum_{k=1}^{n}\frac{x_k-z_k}{\sigma_k^2}\right)^d \tag{10}$$

presented in [40] is suitable, too.

We will now assess the savings induced by our approach, first with respect to the number of divisions and then to overall learning time.

For a training set with l instances and n attributes the number of divisions in the initial transformation is simply

given by the number of entries in the original data matrix, that is,

$$d_T = l \cdot n.$$

For our implementation of the decomposition algorithm the number of divisions in the standard kernel function evaluations is given by

$$d_E = D \cdot l \cdot \hat{l},$$

where D is the number of decomposition steps and $\hat{l}$ is the working set size. For the generalized kernel, d_E is higher by a factor of n.

Since our approach replaces the divisions of the kernel evaluations with those of the data transformation, the overall number of divisions is reduced by a factor of d_E/d_T. In Table I we show these ratios for the adult data set. We computed the number d_E for the (standard) kernel evaluations and different working set sizes. Note that for all tests we have

$$d_T = 30000 \cdot 14 = 420000.$$

TABLE I
NUMBERS OF DIVISIONS d_E FOR THE USUAL GAUSSIAN KERNEL (3) AND THE CORRESPONDING NORMALIZATIONS USING THE FIXED VALUE d_T.

$\hat{l}$	D	d_E	d_E/d_T
5000	37	$5550 \cdot 10^6$	13200
10000	16	$4800 \cdot 10^6$	11400
15000	10	$4500 \cdot 10^6$	10700
20000	5	$3000 \cdot 10^6$	7100
25000	2	$1500 \cdot 10^6$	3600
30000	1	$900 \cdot 10^6$	2100

The data indicate that the savings are highest for small values of $\hat{l}$. However, a large factor does not automatically minimize the overall time. We will analyze overall running time for different working set sizes in the next section.

Now we consider the execution time for different ways of kernel computations. We measure the time that is spent to transform the data as well as the time used to solve the quadratic optimization problem, i.e., the ensuing training, which includes the kernel computations. We consider the following five variants for kernel evaluation:

- K_1 : standard kernel (3) including a single division operation,
- K_2 : standard kernel (4) including a single multiplication operation,
- K_3 : multi-parameter kernel (5) including n division operations,
- K_4 : standard kernel (7) with the proposed pre-scaling, and
- K_5 : multi-parameter kernel (9) with the proposed pre-scaling.

In Table II we show the training time (in seconds) of our support vector machine for these kernels. We performed the tests for a working set size of 10000 to show the amount of time that can be saved. Since the number of kernel evaluations depends on the working set size, too, the effects can vary. We will analyze the influence of the working set size onto the overall training time in the next section.

TABLE II
INFLUENCE OF THE KERNEL EVALUATION METHOD ONTO THE OVERALL TRAINING TIME.

	pre-scaling	training
K_1	–	1529.2
K_2	–	1452.7
K_3	–	2412.3
K_4	0.01	1122.3
K_5	0.02	1122.3

From Table II we conclude the following:

- Replacing the division with a multiplication, i.e. replace K_1 by K_2, gives only a minor improvement on our machine [38].
- For our example the initial data transformation reduces the overall training time by 30% for the standard kernel and by more than 50% for the generalized kernel.
- The preceding discussion suggests that the training time results for K_4 and K_5 should be equal, which indeed is true. This means that the multi-parameter kernel essentially comes for free—except for the fact that it involves more learning parameters, which must be set before the training.
- The time for the initial transformation is negligible. It might be reduced even further with an easy-to-implement parallel version.

VI. OPTIMAL WORKING SET SIZE

In this section we analyze the influence of the working set size $\hat{l}$ on the number of decomposition steps, D, the number of kernel evaluations, E, and the training time. In [31] we observed that for a data set with 10000 points and working set sizes between 1000 and 7000 points there were nearly no differences between the training times. The situation is different for the much larger adult data set with its 30000 points.

In Table III we summarize the test results we achieved for serial SVM training. All tests were performed with the kernel (9), which is the most efficient one. The training times do not include the transformation times, which are negligible and do not depend on $\hat{l}$. Computation times are given in seconds as before. In addition to medium-sized working sets we also consider very small and extremely large working sets.

Comparing the graphs for E and the time in Fig. 4 confirms the dominating effect of the kernel evaluations on the training time. From the serial experiments we conclude that the largest possible working set minimizes the training time.

Now we consider the problem of working set selection for parallel SVM training. For the parallel mode we try to find out whether for a fixed number of threads the same number $\hat{l}$ also leads to the minimal consumption of time or not. Since our decomposition algorithm consists of two parallelized parts that show different behavior for varying working set sizes we cannot predict the effects for the parallel algorithm easily.

TABLE III
NUMBER OF DECOMPOSITION STEPS AND OF KERNEL EVALUATIONS, AND TRAINING TIMES FOR DIFFERENT WORKING SET SIZES.

$\hat{l}$	D	E	time
50	5831	$8.747 \cdot 10^9$	1869.3
100	2828	$8.484 \cdot 10^9$	1779.6
500	497	$7.455 \cdot 10^9$	1618.1
1000	238	$7.140 \cdot 10^9$	1508.2
2000	113	$6.780 \cdot 10^9$	1449.6
5000	37	$5.550 \cdot 10^9$	1212.1
10000	16	$4.800 \cdot 10^9$	1108.0
15000	10	$4.500 \cdot 10^9$	1099.1
20000	5	$3.000 \cdot 10^9$	780.2
25000	2	$1.500 \cdot 10^9$	430.4
30000	1	$0.900 \cdot 10^9$	267.8

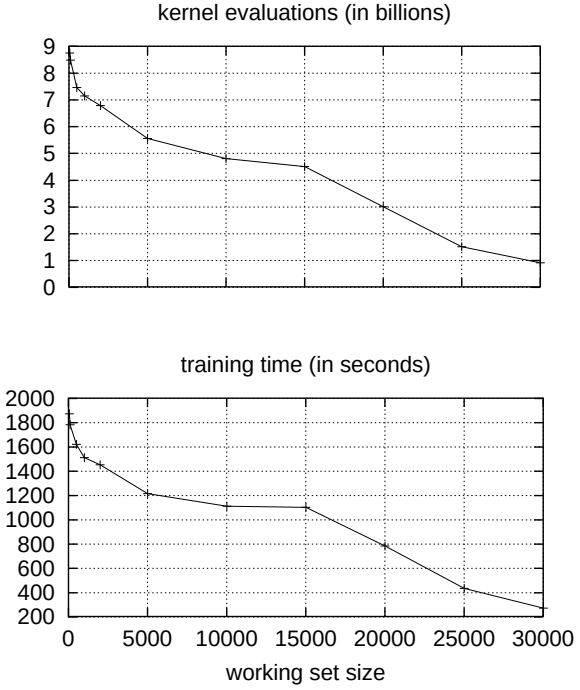

Fig. 4. Number of kernel evaluations and training time for different values of $\hat{l}$.

However, some aspects are already known. The efficiency of the parallel numerical linear algebra kernels (*ESSLSMP* routines on the JUMP) is low for small working set sizes since the problem sizes within the QP solver solely correspond to the working set size and not to the overall problem size l. The definition of "small" is somewhat vague and depends on the computing system to be used as well as the number of threads, but of course a value $\hat{l} = 1000$ is not sufficient for satisfactory speedups of *ESSLSMP* routines.

In Table IV we show results of parallel SVM training for two large working set sizes and different numbers of threads. For $\hat{l} = 15000$ the speedups tend to be slightly larger than for the extreme case $\hat{l} = l$. This is due to the fact that for $\hat{l} = 30000$ the influence of sequential parts of the code is higher. For example, since the number of decomposition steps—and therefore of kernel matrix updates—is only one, the relative contribution of this perfectly scalable routine is smaller.

TABLE IV
SPEEDUP VALUES FOR TWO DIFFERENT WORKING SET SIZES USING UP TO 8 THREADS.

	$\hat{l} = 15000$		$\hat{l} = 30000$	
	time	speedup	time	speedup
serial	1108.0	–	267.8	–
2 threads	535.6	2.1	144.6	1.9
3 threads	345.1	3.2	106.9	2.5
4 threads	263.4	4.2	78.6	3.4
5 threads	223.2	5.0	66.8	4.0
6 threads	231.9	4.8	56.8	4.7
7 threads	220.7	5.0	48.8	5.5
8 threads	229.7	4.8	49.9	5.4

Our shared memory parallelization yields satisfactory speedups for small numbers of processors, but it does not scale to high numbers of processors. Indeed, the speedups did not exceed 5 and 5.5, and these were obtained with 5 and 7 processors. Note that the SVM training involves at most level-2 numerical linear algebra kernels, which can make only very limited use of the processors' caches. Therefore the number of data accesses increases with the number of threads, until the maximum bandwidth of the memory is reached.

The restriction of our shared memory parallelization to small numbers of processors is, however, not a severe limitation. If large numbers of processors are available, then two additional levels of parallelism may be exploited with the message passing paradigm: The k-fold cross validation, which requires training of k SVMs on different data, is easily parallelized with a farming approach, and a parallel optimizer can be used to determine adequate settings for the learning parameters, such as C and the σ_i.

The setting $\hat{l} = 30000$ and 7 threads led to the minimal training time. Since the data transformation needed 0.02 seconds, the overall time for the generalized kernel is also 48.8 seconds. Comparing this value with the 2412.3 seconds given in Table II we observe that combining the pre-scaling, an optimal working set size, and a moderate degree of parallelism may result in an overall speedup of almost 50.

Based on the results in Sect. V and the Tables III and IV we propose the following settings for efficient training of support vector machines:

- a priori transformation of the training data according to (6) or (8),
- implementation of the modified kernel (7) and (9) respectively,
- choice of working sets as large as the available memory allows, and
- usage of an appropriate number of threads for parallel training; 6 may be a reasonable upper bound, as it can be seen in Table IV.

VII. CONCLUSIONS AND FUTURE WORK

We proposed several ideas and techniques for the efficient serial and parallel training of support vector machines for

large data sets. We introduced an a priori transformation of the training data that heavily reduces training time for large data sets. In addition, this transformation allows for usage of multiple kernel parameters without additional costs. In the extreme case the user may assign a single kernel parameter to each feature in the data. We analyzed the behavior of our serial and parallel support vector machine learning method for varying numbers of working set sizes. In combination with the choice of a reasonable working set size the improvement of support vector learning performance can be substantial.

In the future we plan to combine our parallel support vector learning algorithm with efficient parameter optimization methods [16]. This combination would lead to a fully automated approach for fast and reliable support vector machine learning for the classification of large data sets. In addition we work on the comparison and improvement of different quality measures. A good quality measure is the crucial component of the SVM parameter optimization stage, since the values of the quality measure are the only information given iteratively to the optimizer. In an automated approach where the user is not allowed to take any corrective actions a well defined quality measure may ensure success.

Acknowledgements

We would like to acknowledge the support by Research Centre Jülich. We are thankful to the ZAM team at Juelich for generous technical support. Thanks also to several co-workers for careful reading and a couple of improvements.

References

[1] A. Kless and T. Eitrich, "Cytochrome p450 classification of drugs with support vector machines implementing the nearest point algorithm," in *Knowledge Exploration in Life Science Informatics, International Symposium, KELSI 2004, Milan, Italy*, ser. Lecture Notes in Computer Science, J. A. López, E. Benfenati, and W. Dubitzky, Eds., vol. 3303. Springer, 2004, pp. 191–205.

[2] T. Joachims, "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, 1998, pp. 137–142.

[3] B. Schölkopf, "The kernel trick for distances," in *NIPS*, 2000, pp. 301–307.

[4] E. Osuna, R. Freund, and F. Girosi, "Support vector machines: training and applications," Massachusetts Institute of Technology, AI Memo 1602, 1997.

[5] V. N. Vapnik, *Statistical learning theory*. New York: John Wiley & Sons, 1998.

[6] C.-J. Lin, "On the convergence of the decomposition method for support vector machines," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1288–1298, 2001.

[7] M. Momma and K. P. Bennett, "A pattern search method for model selection of support vector regression," in *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*, R. L. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, Eds. SIAM, 2002.

[8] C. Staelin, "Parameter selection for support vector machines," HP Laboratories, Israel, Technical Report HPL-2002-354, 2002.

[9] H. Nunez, C. Angulo, and A. Catala, "Support vector machines with symbolic interpretation," *VII Simpsio Brasileiro de Redes Neurais - Recife, PE, Brasil*, pp. 142–147, 2002. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SBRN.2002.1181456

[10] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, "Parallel support vector machines: the cascade SVM," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 521–528.

[11] V. Ruggiero and L. Zanni, "On the efficiency of splitting and projection methods for large strictly convex quadratic programs," *Applied Optimization*, pp. 401–413, 1999.

[12] H. Yu, J. Yang, and J. Han, "Classifying large data sets using SVMs with hierarchical clusters," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, Eds. ACM, 2003, pp. 306–315.

[13] S. S. Keerthi, "Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms," *IEEE Transactions on Neural Networks*, vol. 13, pp. 1225–1229, 2002.

[14] R. Collobert, S. Bengio, and Y. Bengio, "A parallel mixture of SVMs for very large scale problems," *Neural Computation*, vol. 14, no. 5, pp. 1105–1114, 2002.

[15] C. Hsu and C. Lin, "A comparison of methods for multi-class support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, pp. 415–425, 2002.

[16] T. Eitrich and B. Lang, "Parallel tuning of support vector machine learning parameters for large and unbalanced data sets," in *Computational Life Sciences, First International Symposium, CompLife 2005, Konstanz, Germany*, ser. Lecture Notes in Computer Science, M. R. Berthold, R. Glen, K. Diederichs, O. Kohlbacher, and I. Fischer, Eds., vol. 3695. Springer, 2005, pp. 253–264.

[17] T. P. Runarsson and S. Sigurdsson, "Asynchronous parallel evolutionary model selection for support vector machines," *Neural Information Processing - Letters and Reviews*, vol. 3, no. 3, pp. 59–67, june 2004.

[18] S. Celis and D. R. Musicant, "Weka-parallel: machine learning in parallel," Carleton College, Computer Science Technical Report 2002b, 2002.

[19] J.-X. Dong, A. Krzyzak, and C. Y. Suen, "A fast parallel optimization for training support vector machines," in *Proceedings of 3rd International Conference on Machine Learning and Data Mining*, P. Perner and A. Rosenfeld, Eds., 2003, pp. 96–105.

[20] T. Serafini, G. Zanghirati, and L. Zanni, "Parallel decomposition approaches for training support vector machines," in *Proceedings of the International Conference ParCo2003, Dresden, Germany*. Elsevier, 2004, pp. 259–266.

[21] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge, UK: Cambridge University Press, 2000.

[22] V. N. Vapnik, *The nature of statistical learning theory*. New York: Springer, 1995.

[23] B. Schölkopf and A. J. Smola, *Learning with kernels*. Cambridge, MA: MIT Press, 2002.

[24] R. Herbrich, *Learning kernel classifiers: theory and algorithms*. Cambridge, MA, USA: MIT Press, 2001.

[25] T. Eitrich and B. Lang, "Efficient optimization of support vector machine learning parameters for unbalanced datasets," *Journal of Computational and Applied Mathematics*, 2005, in press.

[26] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

[27] R. Collobert and S. Bengio, "SVMTorch: a support vector machine for large-scale regression and classification problems," *Journal of Machine Learning Research*, vol. 1, pp. 143–160, 2001. [Online]. Available: http://www.idiap.ch/learning/SVMTorch.html

[28] T. Joachims, "SVM-light support vector machine," Webpage, Mar 2002, http://www.cs.cornell.edu/People/tj/svm_light/, Version: 5.00, Date: 03.07.2002. [Online]. Available: http://www.cs.cornell.edu/People/tj/svm_light

[29] S. Rüping, "mySVM-manual," Webpage, 2000. [Online]. Available: http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM

[30] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[31] T. Eitrich and B. Lang, "Shared memory parallel support vector machine learning," Research Centre Jülich, Preprint FZJ-ZAM-IB-2005-11, September 2005, submitted for publication.

[32] P. Laskov, "Feasible direction decomposition algorithms for training support vector machines," *Machine Learning*, vol. 46, no. 1-3, pp. 315–349, 2002.

[33] S. Leyffer, "The return of the active set method," 2005, to appear in Oberwolfach Report.

[34] T. Serafini, G. Zanghirati, and L. Zanni, "Gradient projection methods for quadratic programs and applications in training support vector machines," *Optimization Methods and Software*, vol. 20, no. 2-3, pp. 353–378, 2005.

[35] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 185–208.
[36] S. Hettich, C. L. Blake, and C. J. Merz, *UCI repository of machine learning databases*, 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.html.
[37] G. Zanghirati and L. Zanni, "A parallel solver for large quadratic programs in training support vector machines," *Parallel Computing*, vol. 29, no. 4, pp. 535–551, 2003.
[38] U. Detert, *Introduction to the JUMP architecture*, 2004. [Online]. Available: http://jumpdoc.fz-juelich.de
[39] T. Joachims, "Making large-scale SVM learning practical," in *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. MIT Press, 1998, pp. 169–185.
[40] O. Chapelle, V. N. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Machine Learning*, vol. 46, no. 1, pp. 131–159, 2002.