UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

LEHRSTUHL VIII
KÜNSTLICHE INTELLIGENZ

Making Large-Scale SVM Learning Practical

LS–8 Report 24

Thorsten Joachims

Dortmund, 15. June, 1998

Universität Dortmund
Fachbereich Informatik

UNI DO

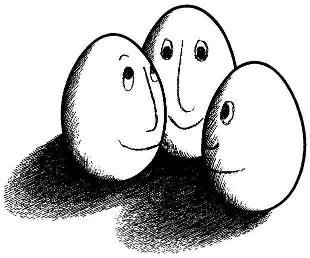
University of Dortmund
Computer Science Department

Forschungsberichte des Lehrstuhls VIII (KI)
Fachbereich Informatik
der Universität Dortmund

ISSN 0943-4135

Anforderungen an:

Universität Dortmund
Fachbereich Informatik
Lehrstuhl VIII
D-44221 Dortmund



Research Reports of the unit no. VIII (AI)
Computer Science Department
of the University of Dortmund

ISSN 0943-4135

Requests to:

University of Dortmund
Fachbereich Informatik
Lehrstuhl VIII
D-44221 Dortmund

e-mail: reports@ls8.informatik.uni-dortmund.de
ftp: ftp-ai.informatik.uni-dortmund.de:pub/Reports
www: http://www-ai.informatik.uni-dortmund.de/ls8-reports.html

Making Large-Scale SVM Learning Practical

LS–8 Report 24

Thorsten Joachims

Dortmund, 15. June, 1998

Universität Dortmund
Fachbereich Informatik

Abstract

Training a support vector machine (SVM) leads to a quadratic optimization problem with bound constraints and one linear equality constraint. Despite the fact that this type of problem is well understood, there are many issues to be considered in designing an SVM learner. In particular, for large learning tasks with many training examples, off-the-shelf optimization techniques for general quadratic programs quickly become intractable in their memory and time requirements. SVM^{light}[1] is an implementation of an SVM learner which addresses the problem of large tasks. This chapter presents algorithmic and computational results developed for SVM^{light}V2.0, which make large-scale SVM training more practical. The results give guidelines for the application of SVMs to large domains.

Also published in:
'Advances in Kernel Methods - Support Vector Learning',
Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.),
MIT Press, Cambridge, USA, 1998.

[1] SVM^{light} is available at `http://www-ai.cs.uni-dortmund.de/svm_light`

1 Introduction

Vapnik [1995] shows how training a support vector machine for the pattern recognition problem leads to the following quadratic optimization problem (QP) OP1.

$$\textbf{(OP1)}\ \text{minimize:} \quad W(\boldsymbol{\alpha}) = -\sum_{i=1}^{\ell} \alpha_i + \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} y_i y_j \alpha_i \alpha_j k(\mathbf{x_i}, \mathbf{x_j}) \tag{1}$$

$$\text{subject to:} \quad \sum_{i=1}^{\ell} y_i \alpha_i = 0 \tag{2}$$

$$\forall i : 0 \leq \alpha_i \leq C \tag{3}$$

The number of training examples is denoted by ℓ. $\boldsymbol{\alpha}$ is a vector of ℓ variables, where each component α_i corresponds to a training example $(\mathbf{x_i}, \mathbf{y_i})$. The solution of OP1 is the vector $\boldsymbol{\alpha}^*$ for which (1) is minimized and the constraints (2) and (3) are fulfilled. Defining the matrix Q as $(Q)_{ij} = y_i y_j k(\mathbf{x_i}, \mathbf{x_j})$, this can equivalently be written as

$$\text{minimize:} \quad W(\boldsymbol{\alpha}) = -\boldsymbol{\alpha}^T \mathbf{1} + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\alpha}^{\mathbf{T}}\mathbf{Q}\boldsymbol{\alpha} \tag{4}$$

$$\text{subject to:} \quad \boldsymbol{\alpha}^T \mathbf{y} = \mathbf{0} \tag{5}$$

$$\mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{C1} \tag{6}$$

The size of the optimization problem depends on the number of training examples ℓ. Since the size of the matrix Q is ℓ^2, for learning tasks with 10000 training examples and more it becomes impossible to keep Q in memory. Many standard implementations of QP solvers require explicit storage of Q which prohibits their application. An alternative would be to recompute Q every time it is needed. But this becomes prohibitively expensive, if Q is needed often.

One approach to making the training of SVMs on problems with many training examples tractable is to decompose the problem into a series of smaller tasks. SVM^{light} uses the decomposition idea of Osuna et al. [1997b]. This decomposition splits OP1 in an inactive and an active part - the so call "working set". The main advantage of this decomposition is that it suggests algorithms with memory requirements linear in the number of training examples and linear in the number of SVs. One potential disadvantage is that these algorithms may need a long training time. To tackle this problem, this chapter proposes an algorithm which incorporates the following ideas:

- An efficient and effective method for selecting the working set.
- Successive "shrinking" of the optimization problem. This exploits the property that many SVM learning problems have
  - much less support vectors (SVs) than training examples.
  - many SVs which have an α_i at the upper bound C.
- Computational improvements like caching and incremental updates of the gradient and the termination criteria.

This chapter is structured as follows. First, a generalized version of the decompositon algorithm of Osuna et al. [1997a] is introduced. This identifies the problem of selecting the working set, which is addressed in the following section. In section 4 a method for "shrinking" OP1 is presented and section 5 describes the computational and implementational approach of SVM^{light}. Finally, experimental results on two benchmark tasks, a text classification task, and an image recognition task are discussed to evaluate the approach.

2 General Decomposition Algorithm

This section presents a generalized version of the decomposition strategy proposed by Osuna et al. [1997a]. This strategy uses a decomposition similar to those used in *active set* strategies (see Gill et al. [1981]) for the case that all inequality constraints are simple bounds. In each iteration the variables α_i of OP1 are split into two categories.

- the set B of free variables
- the set N of fixed variables

Free variables are those which can be updated in the current iteration, whereas fixed variables are temporarily fixed at a particular value. The set of free variables will also be referred to as the working set. The working set has a constant size q much smaller than ℓ.

The algorithm works as follows:

- `While the optimality conditions are violated`
  - `Select` q `variables for the working set` B`. The remaining` $\ell - q$ `variables are fixed at their current value.`
  - `Decompose problem and solve QP-subproblem: optimize` $W(\boldsymbol{\alpha})$ `on` B`.`
- `Terminate and return` $\boldsymbol{\alpha}$`.`

How can the algorithm detect that it has found the optimal value for $\boldsymbol{\alpha}$? Since OP1 is guaranteed to have a positive-semidefinite Hessian Q and all constraints are linear, OP1 is a convex optimization problem. For this class of problems the following Kuhn-Tucker conditions are necessary and sufficient conditions for optimality. Denoting the Lagrange multiplier for the equality constraint 5 with λ^{eq} and the Lagrange multipliers for the lower and upper bounds 6 with $\lambda^{\mathbf{lo}}$ and $\lambda^{\mathbf{up}}$, $\boldsymbol{\alpha}$ is optimal for OP1, if there exist λ^{eq}, $\lambda^{\mathbf{lo}}$, and $\lambda^{\mathbf{up}}$, so that (Kuhn-Tucker Conditions, see Werner [1984]):

$$
\begin{aligned}
& g(\boldsymbol{\alpha}) + (\lambda^{eq}\mathbf{y} - \lambda^{\mathbf{lo}} + \lambda^{\mathbf{up}}) && = \mathbf{0} && (7)\\
\forall i \in [1..n]: \quad & \lambda_i^{lo}(-\alpha_i) && = 0 && (8)\\
\forall i \in [1..n]: \quad & \lambda_i^{up}(\alpha_i - C) && = 0 && (9)\\
& \lambda^{\mathbf{lo}} && \geq \mathbf{0} && (10)\\
& \lambda^{\mathbf{up}} && \geq \mathbf{0} && (11)\\
& \boldsymbol{\alpha}^T\mathbf{y} && = 0 && (12)\\
\mathbf{0} \leq \quad & \boldsymbol{\alpha} && \leq C\mathbf{1} && (13)
\end{aligned}
$$

$g(\boldsymbol{\alpha})$ is the vector of partial derivatives at $\boldsymbol{\alpha}$. For OP1 this is

$$g(\boldsymbol{\alpha}) = -\mathbf{1} + \mathbf{Q}\boldsymbol{\alpha} \tag{14}$$

If the optimality conditions do not hold, the algorithm decomposes OP1 and solves the smaller QP-problem arising from this. The decomposition assures that this will lead to progress in the objective function $W(\boldsymbol{\alpha})$, if the working set B fulfills some minimum requirements (see Osuna et al. [1997b]). In particular, OP1 is decomposed by separating the variables in the working set B from those which are fixed (N). Let's assume $\boldsymbol{\alpha}$, $\mathbf{y}$, and Q are properly arranged with respect to B and N, so that

$$\boldsymbol{\alpha} = \left| \begin{array}{c} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N \end{array} \right| \qquad \mathbf{y} = \left| \begin{array}{c} \mathbf{y_B} \\ \mathbf{y_N} \end{array} \right| \qquad Q = \left| \begin{array}{cc} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{array} \right| \tag{15}$$

Since Q is symmetric (in particular $Q_{BN} = Q_{NB}^T$), we can write

$$\begin{aligned}
\textbf{(OP2)}\ \text{minimize:} \quad & W(\boldsymbol{\alpha}) = -\boldsymbol{\alpha}_B^T(\mathbf{1} - \mathbf{Q_{BN}}\boldsymbol{\alpha}_\mathbf{N}) + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\alpha}_\mathbf{B}^\mathbf{T}\mathbf{Q_{BB}}\boldsymbol{\alpha}_\mathbf{B} + \\
& \frac{1}{2}\boldsymbol{\alpha}_N^T Q_{NN}\boldsymbol{\alpha}_N - \boldsymbol{\alpha}_N^T\mathbf{1} && (16) \\
\text{subject to:} \quad & \boldsymbol{\alpha}_B^T\mathbf{y_B} + \boldsymbol{\alpha}_\mathbf{N}^\mathbf{T}\mathbf{y_N} = \mathbf{0} && (17) \\
& \mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{C1} && (18)
\end{aligned}$$

Since the variables in N are fixed, the terms $\frac{1}{2}\boldsymbol{\alpha}_N^T Q_{NN}\boldsymbol{\alpha}_N$ and $-\boldsymbol{\alpha}_N^T\mathbf{1}$ are constant. They can be omitted without changing the solution of OP2. OP2 is a positive semidefinite quadratic programming problem which is small enough be solved by most off-the-shelf methods. It is easy to see that changing the α_i in the working set to the solution of OP2 is the optimal step on B. So fast progress depends heavily on whether the algorithm can select good working sets.

3 Selecting a Good Working Set

When selecting the working set, it is desirable to select a set of variables such that the current iteration will make much progress towards the minimum of $W(\boldsymbol{\alpha})$. The following proposes a strategy based on Zoutendijk's method (see Zoutendijk [1970]), which uses a first-order approximation to the target function. The idea is to find a steepest feasible direction $\mathbf{d}$ of descent which has only q non-zero elements. The variables corresponding to these elements will compose the current working set.

This approach leads to the following optimization problem:

$$\begin{aligned}
\textbf{(OP3)}\ \text{minimize:} \quad & V(\mathbf{d}) = \mathbf{g}(\boldsymbol{\alpha}^{(\mathbf{t})})^\mathbf{T}\mathbf{d} && (19) \\
\text{subject to:} \quad & \mathbf{y}^\mathbf{T}\mathbf{d} = \mathbf{0} && (20) \\
& d_i \geq 0 \qquad \text{for i: } \alpha_i = 0 && (21) \\
& d_i \leq 0 \qquad \text{for i: } \alpha_i = C && (22) \\
& -\mathbf{1} \leq \mathbf{d} \leq \mathbf{1} && (23) \\
& |\{d_i : d_i \neq 0\}| = q && (24)
\end{aligned}$$

The objective (19) states that a direction of descent is wanted. A direction of descent has a negative dot-product with the vector of partial derivatives $g(\boldsymbol{\alpha}^{(t)})$ at the current point $\boldsymbol{\alpha}^{(t)}$. Constraints (20), (21), and (22) ensure that the direction of descent is projected along the equality constraint (5) and obeys the active bound constraints. Constraint (23) normalizes the descent vector to make the optimization problem well-posed. Finally, the last constraint (24) states that the direction of descent shall only involve q variables. The variables with non-zero d_i are included into the working set B. This way we select the working set with the steepest feasible direction of descent.

3.1 Convergence

The selection strategy, the optimality conditions, and the decomposition together specify the optimization algorithm. A minimum requirement this algorithm has to fulfill is that it

- terminates only when the optimal solution is found
- if not at the solution, takes a step towards the optimum

The first requirement can easily be fulfilled by checking the (necessary and sufficient) optimality conditions (7) to (13) in each iteration. For the second one, let's assume the current $\boldsymbol{\alpha}^{(t)}$ is not optimal. Then the selection strategy for the working set returns an optimization problem of type OP2. Since by construction for this optimization problem there exists a $\mathbf{d}$ which is a feasible direction for descent, we know using the results of Zoutendijk [1970] that the current OP2 is non-optimal. So optimizing OP2 will lead to a lower value of the objective function of OP2. Since the solution of OP2 is also feasible for OP1 and due to the decomposition (16), we also get a lower value for OP1. This means we get a strict descent in the objective function of OP1 in each iteration.

3.2 How to Solve OP3

The solution to OP3 is easy to compute using a simple strategy. Let $\omega_i = y_i g_i(\boldsymbol{\alpha}^{(t)})$ and sort all α_i according to ω_i in decreasing order. Let's futhermore require that q is an even number. Successively pick the $q/2$ elements from the top of the list for which $0 < \alpha_i^{(t)} < C$, or $d_i = -y_i$ obeys (21) and (22). Similarly, pick the $q/2$ elements from the bottom of the list for which $0 < \alpha_i^{(t)} < C$, or $d_i = y_i$ obeys (21) and (22). These q variables compose the working set.

4 Shrinking: Reducing the Size of OP1

For many tasks the number of SVs is much smaller than the number of training examples. If it was known a priori which of the training examples turn out as SVs, it would be sufficient to train just on those examples and still get to the same result. This would make OP1 smaller and faster to solve, since we could save time and space by not needing parts of the Hessian Q which do not correspond to SVs.

Similarly, for noisy problems there are often many SVs with an α_i at the upper bound C. Let's call these support vectors "bounded support vectors" (BSVs). Similar arguments as for the non-support vectors apply to BSVs. If it was known a priori which of the training examples turn out as BSVs, the corresponding α_i could be fixed at C leading to a new optimization problem with fewer variables.

During the optimization process it often becomes clear fairly early that certain examples are unlikely to end up as SVs or that they will be BSVs. By eliminating these variables from OP1, we get a smaller problem OP1' of size ℓ'. From OP1' we can construct the solution of OP1. Let X denote those indices corresponding to unbounded support vectors, Y those indexes which correspond to BSVs, and Z the indices of non-support vectors. The transformation from OP1 to OP1' can be done using a decomposition similar to (16). Let's assume $\boldsymbol{\alpha}$, $\mathbf{y}$, and Q are properly arranged with respect to X, Y, and Z, so that we can write

$$\boldsymbol{\alpha} = \begin{vmatrix} \boldsymbol{\alpha}_X \\ \boldsymbol{\alpha}_Y \\ \boldsymbol{\alpha}_Z \end{vmatrix} = \begin{vmatrix} \boldsymbol{\alpha}_X \\ C\mathbf{1} \\ \mathbf{0} \end{vmatrix} \qquad \mathbf{y} = \begin{vmatrix} \mathbf{y_X} \\ \mathbf{y_Y} \\ \mathbf{y_Z} \end{vmatrix} \qquad Q = \begin{vmatrix} Q_{XX} & Q_{XY} & Q_{XZ} \\ Q_{YX} & Q_{YY} & Q_{YZ} \\ Q_{ZX} & Q_{ZY} & Q_{ZZ} \end{vmatrix} \tag{25}$$

The decomposition of $W(\boldsymbol{\alpha})$ is

$$\begin{aligned} \text{minimize:} \quad & W(\boldsymbol{\alpha}_X) = -\boldsymbol{\alpha}_X^T(\mathbf{1} - (\mathbf{Q_{XY}1}) \cdot \mathbf{C}) + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\alpha}_{\mathbf{X}}^{\mathbf{T}}\mathbf{Q_{XX}}\boldsymbol{\alpha}_{\mathbf{X}} + \\ & \qquad \frac{1}{2}C\mathbf{1}^{\mathbf{T}}\mathbf{Q_{YY}C1} - |\mathbf{Y}|\mathbf{C} & (26) \\ \text{subject to:} \quad & \boldsymbol{\alpha}_X^T\mathbf{y_X} + \mathbf{C1}^{\mathbf{T}}\mathbf{y_Y} = \mathbf{0} & (27) \\ & \mathbf{0} \leq \alpha_{\mathbf{X}} \leq \mathbf{C1} & (28) \end{aligned}$$

Since $\frac{1}{2}C\mathbf{1}^{\mathbf{T}}\mathbf{Q_{YY}C1} - |\mathbf{Y}|\mathbf{C}$ is constant, it can be dropped without changing the solution. So far it is not clear how the algorithm can identify which examples can be eliminated. It is desirable to find conditions which indicate early in the optimization process that certain variables will end up at a bound. Since sufficient conditions are not known, a heuristic approach based on Lagrange multiplier estimates is used.

At the solution, the Lagrange multiplier of a bound constraint indicates, how much the variable "pushes" against that constraint. A strictly positive value of a Lagrange multiplier of a bound constraint indicates that the variable is optimal at that bound. At non-optimal points, an estimate of the Lagrange multiplier can be used. Let A be the current set of α_i fulfilling $0 < \alpha_i < C$. By solving (7) for λ^{eq} and averaging over all α_i in A, we get the estimate (29) for λ^{eq}.

$$\lambda^{eq} = \frac{1}{|A|}\sum_{i \in A}\left[y_i - \sum_{j=1}^{\ell}\alpha_j y_j k(\mathbf{x_i}, \mathbf{x_j})\right] \tag{29}$$

Note the equivalence of λ^{eq} and the threshold b in the decision function. Since variables α_i cannot be both at the upper and the lower bound simultanously, the multipliers of the bound constraints can now be estimated by

$$\lambda_i^{lo} = y_i\left(\left[\sum_{j=1}^{\ell}\alpha_j y_j k(\mathbf{x_i}, \mathbf{x_j})\right] + \lambda^{eq}\right) - 1 \tag{30}$$

for the lower bounds and by

$$\lambda_i^{up} = -y_i \left(\left[\sum_{j=1}^{\ell} \alpha_j y_j k(\mathbf{x_i}, \mathbf{x_j}) \right] + \lambda^{eq} \right) + 1 \tag{31}$$

for the upper bounds. Let's consider the history of the Lagrange multiplier estimates over the last h iterations. If the estimate (30) or (31) was positive (or above some threshold) at each of the last h iterations, it is likely that this will be true at the optimal solution, too. These variables are eliminated using the decomposition from above. This means that these variables are fixed and neither the gradient, nor the optimality conditions are computed. This leads to a substantial reduction in the number of kernel evaluations.

Since this heuristic can fail, the optimality conditions for the excluded variables are checked after convergence of OP1'. If necessary, the full problem is reoptimized starting from the solution of OP1'.

5 Efficient Implementation

While the previous sections dealt with algorithmic issues, there are still a lot of open questions to be answered before having an efficient implementation. This section addresses these implementational issues.

5.1 Termination Criteria

There are two obvious ways to define termination criteria which fit nicely into the algorithmic framework presented above. First, the solution of OP3 can be used to define a necessary and sufficient condition for optimality. If (19) equals 0, OP1 is solved with the current $\boldsymbol{\alpha}^{(t)}$ as solution.

SVM^{light}goes another way and uses a termination criterion derived from the optimality conditions (7)-(13). Using the same reasoning as for (29)-(31), the following conditions with $\epsilon = 0$ are equivalent to (7)-(13).

$$\forall i \text{ with } 0 < \alpha_i < C: \quad \lambda^{eq} - \epsilon \leq y_i - [\textstyle\sum_{j=1}^{\ell} \alpha_j y_j k(\mathbf{x_i}, \mathbf{x_j})] \leq \boldsymbol{\lambda}^{\mathbf{eq}} + \epsilon \tag{32}$$

$$\forall i \text{ with } \alpha_i = 0: \quad y_i([\textstyle\sum_{j=1}^{\ell} \alpha_j y_j k(\mathbf{x_i}, \mathbf{x_j})] + \boldsymbol{\lambda}^{\mathbf{eq}}) \geq \mathbf{1} - \epsilon \tag{33}$$

$$\forall i \text{ with } \alpha_i = C: \quad y_i([\textstyle\sum_{j=1}^{\ell} \alpha_j y_j k(\mathbf{x_i}, \mathbf{x_j})] + \boldsymbol{\lambda}^{\mathbf{eq}}) \leq \mathbf{1} + \epsilon \tag{34}$$

$$\boldsymbol{\alpha}^T \mathbf{y} = \mathbf{0} \tag{35}$$

The optimality conditions (32), (33), and (34) are very natural since they reflect the constraints of the primal optimization problem. In practice these conditions need not be fulfilled with high accuracy. Using a tolerance of $\epsilon = 0.001$ is acceptable for most tasks. Using a higher accuracy did not show improved generalization performance on the tasks tried, but lead to considerably longer training time.

5.2 Computing the Gradient and the Termination Criteria Efficiently

The efficiency of the optimization algorithm greatly depends on how efficiently the "housekeeping" in each iteration can be done. The following quantities are needed in each iteration.

- The vector of partial derivatives $g(\boldsymbol{\alpha}^{(t)})$ for selecting the working set.
- The values of the expressions (32), (33), and (34) for the termination criterion.
- The matrices Q_{BB} and Q_{BN} for the QP subproblem.

Fortunately, due to the decompositon approach, all these quantities can be computed or updated knowing only q rows of the Hessian Q. These q rows correspond to the variables in the current working set. The values in these rows are computed directly after the working set is selected and they are stored throughout the iteration. It is useful to introduce $\mathbf{s^{(t)}}$

$$s_i^{(t)} = \sum_{j=1}^{\ell} \alpha_j y_j k(\mathbf{x_i}, \mathbf{x_j}) \tag{36}$$

Knowing $\mathbf{s^{(t)}}$, the gradient (14) as well as in the termination criteria (32)-(34) can be computed very efficiently. When $\boldsymbol{\alpha}^{(t-1)}$ changes to $\boldsymbol{\alpha}^{(t)}$ the vector $\mathbf{s^{(t)}}$ needs to be updated. This can be done efficiently and with sufficient accuracy as follows

$$s_i^{(t)} = s_i^{(t-1)} + \sum_{j \in B} (\alpha_j^{(t)} - \alpha_j^{(t-1)}) y_j k(\mathbf{x_i}, \mathbf{x_j}) \tag{37}$$

Note that only those rows of Q are needed which correspond to variables in the working set. The same is true for Q_{BB} and Q_{BN}, which are merely subsets of columns from these rows.

5.3 What are the Computational Resources Needed in each Iteration?

Most time in each iteration is spent on the kernel evaluations needed to compute the q rows of the Hessian. This step has a time complexity of $O(qlf)$, where f is the maximum number of non-zero features in any of the training examples. Using the stored rows of Q, updating $\mathbf{s^{(t)}}$ is done in time $O(ql)$. Setting up the QP subproblem requires $O(ql)$ as well. Also the selection of the next working set, which includes computing the gradient, can be done in $O(ql)$.

The highest memory requirements are due to storing the q rows of Q. Here $O(ql)$ floating point numbers need to be stored. Besides this, $O(q^2)$ is needed to store Q_{BB} and $O(l)$ to store $\mathbf{s^{(t)}}$.

5.4 Caching Kernel Evaluations

As pointed out in the last section, the most expensive step in each iteration is the evaluation of the kernel to compute the q rows of the Hessian Q. Throughout the optimization process, eventual support vectors enter the working set multiple times. To avoid recomputation of these rows, SVM^{light}uses caching. This allows an elegant trade-off between memory consumption and training time.

SVM^{light}uses a least-recently-used caching strategy. When the cache is full, the element which has not been used for the greatest number of iterations, is removed to make room for the current row.

Only those columns are computed and cached which correspond to active variables. After shrinking, the cache is reorganized accordingly.

5.5 How to Solve OP2 (QP Subproblems)

Currently a primal-dual interior-point solver (see Vanderbei [1994]) implemented by A. Smola is used to solve the QP subproblems OP2. Nevertheless, other optimizers can easily be incorporated into SVM^{light} as well.

6 Related Work

The first approach to splitting large SVM learning problems into a series of smaller optimization tasks was proposed by Boser et al. [1992]. It is known as the "chunking" algorithm (see also Kaufman [1998]). The algorithm starts with a random subset of the data, solves this problem, and iteratively adds examples which violate the optimality conditions. Osuna et al. [1997b] prove formally that this strategy converges to the optimal solution. One disadvantage of this algorithm is that it is necessary to solve QP-problems scaling with the number of SVs. The decomposition of Osuna et al. [1997a], which is used in the algorithm presented here, avoids this.

Currently, an approach called Sequential Minimal Optimization (SMO) is explored for SVM training (see Platt [1998a] and Platt [1998b]). It can be seen a special case of the algorithm presented in this chapter, allowing only working sets of size 2. The algorithms differ in their working set selection strategies. Instead of the steepest feasible descent approach presented here, SMO uses a set of heuristics. Nevertheless, these heuristics are likely to produce similar decisions in practice. Another difference is that SMO treats linear SVMs in a special way, which produces a great speedup for training linear separators. Although possible, this is not implemented in SVM^{light}. On the other hand, SVM^{light} uses caching, which could be a valuable addition to SMO.

7 Experiments

The following experiments evaluate the approach on four datasets. The experiments are conducted on a SPARC Ultra/167Mhz with 128MB of RAM running Solaris II. If not stated otherwise, in the following experiments the cache size is 80 megabytes, the number of iterations h for the shrinking heuristic is 100, and OP1 is solved up to a precision of $\epsilon = 0.001$ in (32)-(34).

7.1 How does Training Time Scale with the Number of Training Examples?

7.1.1 Income Prediction

This task was compiled by John Platt (see Platt [1998a]) from the UCI "adult" data set. The goal is to predict whether a household has an income greater than \$50,000. After discretization of the continuous attributes, there are 123 binary features. On average, there are $\approx$14 non-zero attributes per example.

Table 1 and the left graph in figure 1 show training times for an RBF-kernel

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2/(2\,\sigma^2)\right), \tag{38}$$

with $\sigma = 10$ and $C = 1$. The results for SMO and Chunking are taken from Platt [1998a]. When comparing absolute training times, one should keep in mind that SMO and Chunking were run on a faster computer (266Mhz Pentium II)[2].

Examples	SVM^{light}	SMO	Chunking	Minimum	total SV	BSV
1605	7.8	15.8	34.8	4.2	691	585
2265	16.8	32.1	144.7	9.0	1007	849
3185	30.6	66.2	380.5	6.8	1293	1115
4781	68.4	146.6	1137.2	38.4	1882	1654
6414	120.6	258.8	2530.6	70.2	2475	2184
11221	430.8	781.4	11910.6	215.4	4182	3763
16101	906.0	1784.4	N/A	436.2	5894	5398
22697	1845.6	4126.4	N/A	862.8	8263	7574
32562	3850.2	7749.6	N/A	1795.8	11572	10740
Scaling	2.1	2.1	2.9	2.0		

Table 1: Training times and number of SVs for the income prediction data.

Both $SVM^{light}$and SMO are substantially faster than the conventional chunking algorithm, whereas $SVM^{light}$is about twice as fast as SMO. The best working set size is $q = 2$. By fitting lines to the log-log plot we get an empirical scaling of $\ell^{2.1}$ for both $SVM^{light}$and SMO. The scaling of the chunking algorithm is $\ell^{2.9}$.

The column "minimum" gives a lower bound on the training time. This bound makes the conjecture that in the general case any optimization algorithms needs to at least once look at the rows of the Hessian Q which correspond to the support vectors. The column "minimum" shows the time to compute those rows once (exploiting symmetry). This time scales with $\ell^{2.0}$, showing the complexity inherent in the classification task. For the training set sizes considered, $SVM^{light}$is both close to this minimum scaling as well as within a factor of approximately two in terms of absolute runtime.

7.1.2 Classifying Web Pages

The second data set - again compiled by John Platt (see Platt [1998a]) - is a text classification problem with a binary representation based on 300 keyword features. This representation is extremely sparse. On average there are only ≈12 non-zero features per example.

Table 2 shows training times on this data set for an RBF-kernel (38) with $\sigma = 10$ and $C = 5$. Again, the times for SMO and Chunking are taken from Platt [1998a]. $SVM^{light}$is faster than SMO and Chunking on this data set as well, scaling with $\ell^{1.7}$. The best working set size is $q = 2$.

[2]The Pentium II takes only ≈65% of the time for running SVM^{light}. Many thanks to John Platt for the comparison.

Examples	SVM^{light}	SMO	Chunking	Minimum	total SV	BSV
2477	18.0	26.3	64.9	3.6	431	47
3470	28.2	44.1	110.4	7.8	571	69
4912	46.2	83.6	372.5	13.2	671	96
7366	102.0	156.7	545.4	27.0	878	138
9888	174.6	248.1	907.6	46.8	1075	187
17188	450.0	581.0	3317.9	123.6	1611	363
24692	843.0	1214.0	6659.7	222.6	1994	506
49749	2834.4	3863.5	23877.6	706.2	3069	948
Scaling	1.7	1.7	2.0	1.7		

Table 2: Training times and number of SVs for the Web data.

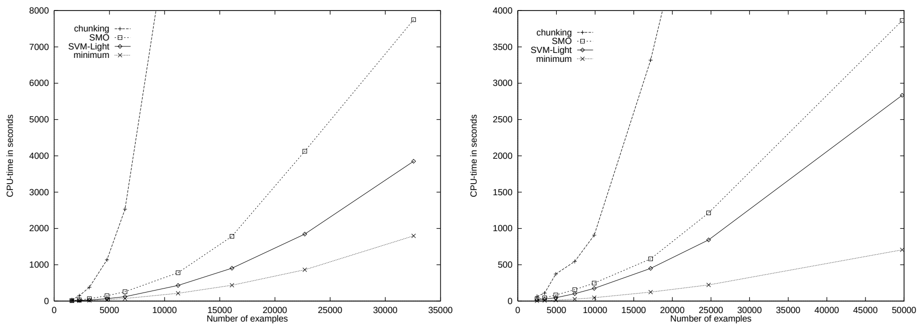


Figure 1: Training times from tables 1 (left) and 2 (right) as graphs.

7.1.3 Ohsumed Data Set

The task in this section is a text classification problem which uses a different representation. Support vector machines have shown very good generalisation performance using this representation (see Joachims [1998]). Documents are represented as high dimensional vectors, where each dimension contains a (TFIDF-scaled) count of how often a particular word occurs in the document. More details can be found in Joachims [1998]. The particular task is to learn "Cardiovascular Diseases" category of the Ohsumed dataset. It involves the first 46160 documents from 1991 using 15000 features. On average, there are ≈63 non-zero features per example. An RBF-kernel with $\sigma = 0.91$ and $C = 50$ is used.

Table 3 shows that this tasks involves many SVs which are not at the upper bound. Relative to this high number of SVs the cache size is small. To avoid frequent recomputations of the same part of the Hessian Q, an additional heuristic is incorporated here. The working set is selected with the constraint that at least for half of the selected variables the kernel values are already cached. Unlike for the previous tasks, optimum performance is achieved with a working set size of $q = 20$. For the training set sizes considered here, runtime is within a factor of 4 from the minimum.

Examples	SVM^{light}	Minimum	total SV	BSV
9337	18.8	7.1	4037	0
13835	46.3	14.4	5382	0
27774	185.7	50.8	9018	0
46160	509.5	132.7	13813	0
Scaling	2.0	1.8		

Table 3: Training time (in minutes) and number of SVs for the Ohsumed data.

7.1.4 Dectecting Faces in Images

In this last problem the task is to classify images according to whether they contain a human face or not. The data set was collected by Shumeet Baluja. The images consist of 20x20 pixels of continuous gray values. So the average number of non-zero attributes per example is 400. An RBF-kernel with $\sigma = 7.1$ and $C = 10$ is used. The working set size is $q = 20$.

Examples	SVM^{light}	Minimum	total SV	BSV
512	10.8	8.4	340	0
1025	37.2	31.2	559	0
2050	129.0	111.0	930	0
4100	443.4	381.0	1507	0
8200	1399.2	1170.6	2181	0
Scaling	1.7	1.7		

Table 4: Training time and number of SVs for the face detection data.

Table 4 shows the training time (in seconds). For this task, the training time is very close to the minimum. This shows that the working set selection strategy is very well suited for avoiding unnecessary kernel evaluations. The scaling is very close to the optimum scaling.

Let's now evaluate, how particular strategies of the algorithm influence the performance.

7.2 What is the Influence of the Working Set Selection Strategy?

The left of figure 2 shows training time dependent on the size of the working set q for the smallest Ohsumed task. The selection strategy from section 3 (lower curve) is compared to a basic strategy similar to that proposed in Osuna et al. [1996] (upper curve). In each iteration the basic strategy simply replaces half of the working set with variables that do not fulfill the optimality conditions. The graph shows that the new selection strategy reduces time by a factor of more than 3.

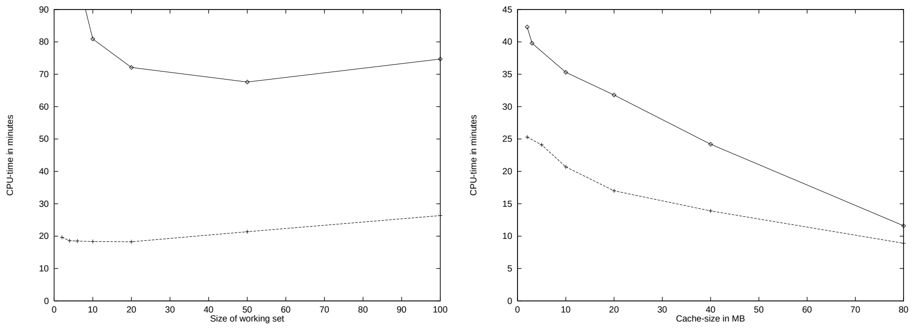


Figure 2: Training time dependent on working set size and cache size for the Ohsumed task.

7.3 What is the Influence of Caching?

The curves in the graph on the right hand side of figure 2 shows that caching has a strong impact on training time. The lower curve shows training time (for an RBF-kernel with $\sigma = 10$ and $C = 50$ on the 9337 examples of the Ohsumed data) dependent on the cache size when shrinking is used. With the cache size ranging from 2 megabytes to 80 megabytes a speedup factor of 2.8 is achieved. The speedup generally increases with an increasing density of the feature vectors $\mathbf{x_i}$.

7.4 What is the Influence of Shrinking?

All experiments above use the shrinking strategy from section 4. The upper curve in figure 2 (right) shows training time without shrinking. It can be seen that shrinking leads to a substantial improvement when the cache is small in relation to the size of the problem. The gain generally increases the smaller the fraction of unbounded SVs is compared to the number of training examples ℓ (here 2385 unbounded SVs, 110 BSVs, and a total of 9337 examples).

8 Conclusions

This chaper presents an improved algorithm for training SVMs on large-scale problems and describes its efficient implementation in SVM^{light}. The algorithm is based on a decomposition strategy and addresses the problem of selecting the variables for the working set in an effective and efficient way. Furthermore, a technique for "shrinking" the problem during the optimization process is introduced. This is found particularly effective for large learning tasks where the fraction of SVs is small compared to the sample size, or when many SVs are at the upper bound. The chapter also describes how this algorithm is efficiently implemented in SVM^{light}. It has a memory requirement linear in the number of training examples and in the number of SVs. Nevertheless, the algorithms can benefit from additional storage space, since the caching strategy allows an elegant trade-off between training time and memory consumption.

9 Acknowledgements

This work was supported by the DFG Collaborative Research Center on Complexity Reduction in Multivariate Data (SFB475). Thanks to Alex Smola for letting me use his solver. Thanks also to Shumeet Baluja and to John Platt for the data sets.

References

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, 1981.

T. Joachims. Text categorization with support vector machines. In *European Conference on Machine Learning (ECML)*, 1998.

L. Kaufman. Solving the quadratic programming problem arising in support vector classification. In B. Schölkopf, C. Burges, and A Smola, editors, *Advances in Kernel Methods - Support Vector Learning.* MIT Press, Cambridge, USA, 1998.

E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. A.I. Memo (in press), MIT A. I. Lab., 1996.

E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 276 – 285, New York, 1997a. IEEE.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In , editor, *Proceedings CVPR'97*, , 1997b. .

J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998a.

J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines,. In B. Schölkopf, C. Burges, and A Smola, editors, *Advances in Kernel Methods - Support Vector Learning.* MIT Press, Cambridge, USA, 1998b.

R. Vanderbei. Loqo: An interior point code for quadratic programming. Technical Report SOR 94-15, Princeton University, 1994.

V. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, New York, 1995.

J. Werner. *Optimization - Theory and Applications.* Vieweg, 1984.

G. Zoutendijk. *Methods of Feasible Directions: a Study in Linear and Non-linear Programming.* Elsevier, 1970.