Active-Set Methods for Support Vector Machines

Michael Vogt[1] and Vojislav Kecman[2]

[1] Darmstadt University of Technology, Institute of Automatic Control, Landgraf-Georg-Strasse 4, 64283 Darmstadt, Germany, `mvogt@iat.tu-darmstadt.de`

[2] University of Auckland, School of Engineering, Private Bag 92019 Auckland, New Zealand `v.kecman@auckland.ac.nz`

Summary. This chapter describes an *active-set* algorithm for the solution of quadratic programming problems in the context of Support Vector Machines (SVMs). Most of the common SVM optimizers implement *working-set* algorithms like the SMO method because of their ability to handle large data sets. Although they show generally good results, they may perform weakly in some situations, e.g., if the problem is ill-posed or if high precision is needed. In these cases, active-set techniques (which are robust general-purpose QP solvers) are a reasonable alternative. Algorithms are derived for classification and regression problems for both fixed and variable bias term. The approximation of the solution is considered as well as the comparison with other optimization methods.

Key words: support vector machines, classification, regression, quadratic programming, active-set algorithm

1 Introduction

In the recent years, Support Vector Machines (SVMs) have become popular for classification and regression tasks [10, 11] since they can treat large input dimensions and show a good generalization behavior. The method has its foundation in classification and has later been extended to regression. SVMs are computed by solving *Quadratic Programming* (QP) problems (see (9), (17), (28) and (36)), the sizes of which depends on the number N of training data.

1.1 Optimization methods

This dependency on N is the most critical point of SVM optimization since N may be very large and the memory consumption is roughly $\mathcal{O}(N^2)$ if the whole QP problem needs to be stored in memory. For that, the choice of an optimization method has to consider mainly the problem size and the memory consumption of the algorithm, see Fig. 1. If the problem is small enough to

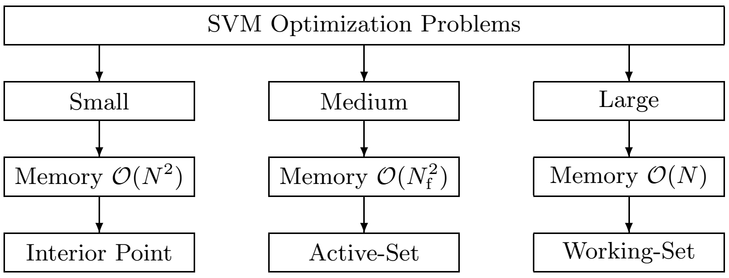

SVM Optimization Problems
Small
Medium
Large
Memory $\mathcal{O}(N^2)$
Memory $\mathcal{O}(N_\mathrm{f}^2)$
Memory $\mathcal{O}(N)$
Interior Point
Active-Set
Working-Set

Fig. 1. QP optimization methods for different training data set sizes

be stored completely in memory, *interior point* methods (having a memory consumption of $\mathcal{O}(N^2)$) are suitable, since they are known to be the most precise QP solvers [7, 10]. For very large data sets on the other hand, there is currently no alternative to *working-set* methods (*decomposition* methods) like SMO [8], ISDA [4] or similar strategies [1]. This class of methods has basically a memory consumption of $\mathcal{O}(N)$ and can therefore cope even with large scale problems. *Active-set* algorithms are appropriate for medium-size problems because they need $\mathcal{O}(N_\mathrm{f}^2)$ memory where N_f is the number of *free* support vectors that is typically only a small fraction of the data set.

Common SVM software packages rely on working-set methods because N is often large in practical applications. However, these methods may show weak results if the problem is ill-posed, if the SVM parameters (C and ε) are not chosen carefully, or if high precision is needed. This is in particular true for regression, see Sec. 5. *Active-set* algorithms are the classical solvers for QP problems. They are known to be robust, but they are often slower and (as stated above) require more memory than working-set algorithms. Only few attempts have been made to utilize this technique for SVMs. E.g., in [5] it is applied to a modified SVM classification problem. Also the *Chunking* algorithm [11] is closely related.

1.2 Active-set algorithms

The basic idea is to find the active set $\mathcal{A}$, i.e., those inequality constraints that are fulfilled with equality. If $\mathcal{A}$ is known, the Karush-Kuhn-Tucker (KKT) conditions reduce to a simple system of linear equations which yields the solution of the QP problem [7]. Because $\mathcal{A}$ is unknown in the beginning, it is constructed iteratively by adding and removing constraints and testing if the solution remains feasible.

The construction of $\mathcal{A}$ starts with an initial active set $\mathcal{A}^0$ containing the indices of the *bounded* variables (lying on the boundary of the feasible region) whereas those in $\mathcal{F}^0 = \{1, \dots, N\} \backslash \mathcal{A}^0$ are *free* (lying in the interior of the feasible region). Then the following steps are performed repeatedly for $k = 1, 2, \dots$:

A1.Solve the KKT system for all variables in $\mathcal{F}^k$.
A2.If the solution is feasible, find the variable in $\mathcal{A}^k$ that violates the KKT conditions most, move it to $\mathcal{F}^k$, then go to A1.
A3.Otherwise find an intermediate value between old and new solution lying on the border of the feasible region, move one bounded variable from $\mathcal{F}^k$ to $\mathcal{A}^k$, then go to A1.

The intermediate solution in step A3 is computed as $\mathbf{a}^k = \eta \bar{\mathbf{a}}^k - (1-\eta)\mathbf{a}^{k-1}$ with maximal $\eta \in [0, 1]$ (*affine scaling*), where $\bar{\mathbf{a}}^k$ is the solution of the linear system in step A1, i.e., the new iterate $\mathbf{a}^k$ lies on the connecting line of $\mathbf{a}^{k-1}$ and $\bar{\mathbf{a}}^k$, see Fig. 2. The optimum is found if during step A2 no violating variable is left in $\mathcal{A}^k$.

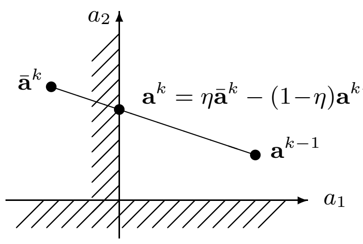

a_2
$\bar{\mathbf{a}}^k$
$\mathbf{a}^k = \eta\bar{\mathbf{a}}^k - (1-\eta)\mathbf{a}^{k-1}$
$\mathbf{a}^{k-1}$
a_1

Fig. 2. Affine scaling of the non-feasible solution

This basic algorithm is used for all cases described in the next sections, only the structures of the KKT system in step A1 and the conditions in step A2 are different. Sec. 2 and 3 describe how to use the algorithm for both classification and regression tasks. In this context we also repeat the derivations of the dual problems in order to introduce the distinction between fixed and variable bias term. In Sec. 4, the efficient solution of the KKT system and the approximation of the solution are explained. Application examples for both classification and regression are given in Sec. 5.

2 Support Vector Machine Classification

A two-class classification problem is given by the data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with the class labels $y_i \in \{-1, 1\}$. Linear classifiers aim to find a decision function $f(\mathbf{x}) = \mathbf{w}^\mathrm{T}\mathbf{x} + b$ so that $f(\mathbf{x}_i) > 0$ for $y_i = 1$ and $f(\mathbf{x}_i) < 0$ for $y_i = -1$. The decision boundary is the intersection of $f(\mathbf{x})$ and the input space, see Fig. 3.

For separable classes, an SVM classifier computes a decision function having a *maximal margin* m with respect to the two classes, so that all data lie outside the margin, i.e., $y_i f(\mathbf{x}_i) \geq 1$. Since $\mathbf{w}$ is the normal vector of the separating hyperplane, the margin can be expressed as $m = 2/\mathbf{w}^\mathrm{T}\mathbf{w}$. In the case of

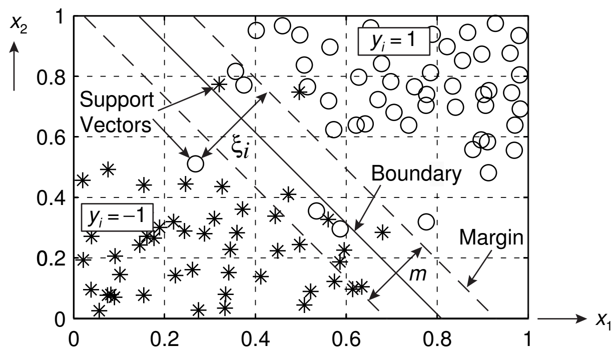


Fig. 3. Separating two overlapping classes with a linear decision function

non-separable classes, *slack variables* ξ_i are introduced measuring the distance to the data lying on the wrong side of the margin. They do not only make the constraints feasible but also occur in the loss function to be minimized [11]. These ideas lead to the *soft margin* classifier:

$$\min_{\mathbf{w},\boldsymbol{\xi}} \quad J_\mathrm{p}(\mathbf{w},\boldsymbol{\xi}) = \frac{1}{2}\mathbf{w}^\mathrm{T}\mathbf{w} + C\sum_{i=1}^{N}\xi_i \tag{1a}$$

$$\text{s.t.} \quad y_i(\mathbf{w}^\mathrm{T}\mathbf{x}_i + b) \geq 1 - \xi_i \tag{1b}$$

$$\xi_i \geq 0, \quad i = 1,\ldots,N. \tag{1c}$$

The parameter C describes the trade-off between maximal margin and correct classification. The *primal* problem (1) is now transformed into its *dual* one by introducing the *Lagrange multipliers* $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the $2N$ primal constraints. The Lagrangian is given by

$$L_\mathrm{p}(\mathbf{w},\boldsymbol{\xi},b,\boldsymbol{\alpha},\boldsymbol{\beta}) = \frac{1}{2}\mathbf{w}^\mathrm{T}\mathbf{w}+C\sum_{i=1}^{N}\xi_i-\sum_{i=1}^{N}\alpha_i\Big[y_i(\mathbf{w}^\mathrm{T}\mathbf{x}_i+b)-1+\xi_i\Big]-\sum_{i=1}^{N}\beta_i\xi_i \tag{2}$$

having a minimum with respect to the primal variables $\mathbf{w}$, $\boldsymbol{\xi}$ and b, and a maximum with respect to the dual variables $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (*saddle point condition*). According to the KKT condition (47a) the minimization is performed with respect to the primal variables in order to find the saddle point:

$$\frac{\partial L_\mathrm{p}}{\partial \mathbf{w}} = \mathbf{0} \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{N} y_i\alpha_i\mathbf{x}_i \tag{3a}$$

$$\frac{\partial L_\mathrm{p}}{\partial \xi_i} = 0 \quad \Rightarrow \quad \alpha_i + \beta_i = C, \quad i = 1,\ldots,N \tag{3b}$$

Although b is also a primal variable, we defer the minimization with respect to b for a moment. Instead, (3) is used to eliminate $\mathbf{w}$, $\boldsymbol{\xi}$ and $\boldsymbol{\beta}$ from the Lagrangian which leads to

$$L_{\mathrm{p}}^{*}(\boldsymbol{\alpha}, b) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + \sum_{i=1}^{N} \alpha_i - b \sum_{i=1}^{N} y_i \alpha_i \ . \tag{4}$$

To solve *nonlinear* classification problems, the SVM is applied to *features* $\boldsymbol{\Phi}(\mathbf{x})$ (instead of the inputs $\mathbf{x}$), where $\boldsymbol{\Phi}$ is a given feature map. Since $\mathbf{x}$ occurs in (4) only in scalar products $\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$, we define the *kernel function*

$$K(\mathbf{x}, \mathbf{x}') = \boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{x})\boldsymbol{\Phi}(\mathbf{x}') \ , \tag{5}$$

and finally (4) becomes

$$L_{\mathrm{p}}^{*}(\boldsymbol{\alpha}, b) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K_{ij} + \sum_{i=1}^{N} \alpha_i - b \sum_{i=1}^{N} y_i \alpha_i \tag{6}$$

with the abbreviation $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. We assume kernels always to be symmetric and positive definite. This class of functions includes nearly all common kernels, like linear kernels, polynomials and Gaussians [10]. The conditions (47d) and (3b) yield additional restrictions for the dual variables

$$0 \le \alpha_i \le C \quad \text{for } i = 1, \ldots, N \tag{7}$$

From (3a) and (5) we conclude that

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\Phi}(\mathbf{x}) + b = \sum_{\alpha_i \neq 0} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b \ . \tag{8}$$

This shows once more the strengths of the kernel concept: SVMs can easily handle extremely large feature spaces since the primal variables $\mathbf{w}$ and the feature map $\boldsymbol{\Phi}$ are needed neither for the optimization nor in the decision function. Vectors $\mathbf{x}_i$ with $\alpha_i \neq 0$ are called *support vectors*. Usually only a small fraction of the data set are support vectors, typically about 5%.

From the algorithmic point of view, an important decision has to be made at this stage: weather the bias term b is treated as a variable or it is kept fixed during optimization. The next two sections derive active-set algorithms for both cases.

2.1 Classification with fixed bias term

We first consider the bias term b to be fixed, including the most important case $b = 0$. This is possible if the kernel function provides an implicit bias which is true e.g. for *positive definite* kernel functions [4, 9, 12]. The only effect is that slightly more support vectors are computed. The main advantage of a fixed bias term is a simpler algorithm since no additional equality constraint needs to be imposed during optimization (like in (17)):

$$\min_{\boldsymbol{\alpha}} \quad J_{\mathrm{d}}(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K_{ij} - \sum_{i=1}^{N}\alpha_i + b\sum_{i=1}^{N} y_i \alpha_i \tag{9a}$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C\,, \quad i = 1, \dots, N \tag{9b}$$

Note that $J_{\mathrm{d}}(\boldsymbol{\alpha})$ equals $-L_{\mathrm{p}}^{*}(\boldsymbol{\alpha}, b)$ with a given b. For $b = 0$ (the "no-bias SVM") the last term of the objective function (9a) vanishes.

If b is kept fixed, the SVM is computed by solving the box-constrained convex QP problem (9), which is one of the most simple QP cases. To solve it with the active-set method described in Sec. 1, the KKT conditions of this problem must be found. Its Lagrangian is

$$\begin{aligned} L_{\mathrm{d}}(\boldsymbol{\alpha}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K_{ij} - \sum_{i=1}^{N}\alpha_i + b\sum_{i=1}^{N} y_i \alpha_i \\ & - \sum_{i=1}^{N}\lambda_i \alpha_i - \sum_{i=1}^{N}\mu_i (C - \alpha_i) \end{aligned} \tag{10}$$

where λ_i and μ_i are the Lagrange multipliers of the constraints $\alpha_i \ge 0$ and $\alpha_i \le C$, respectively. Introducing the prediction errors $E_i = f(\mathbf{x}_i) - y_i$, the KKT conditions can be derived for $i = 1, \dots, N$ (see App. A):

$$\frac{\partial L_{\mathrm{d}}}{\partial \alpha_i} = y_i E_i - \lambda_i + \mu_i = 0 \tag{11a}$$

$$0 \le \alpha_i \le C \tag{11b}$$

$$\lambda_i \ge 0, \quad \mu_i \ge 0 \tag{11c}$$

$$\alpha_i \lambda_i = 0, \quad (C - \alpha_i)\mu_i = 0 \tag{11d}$$

According to α_i, three cases have to be considered:

$$\boxed{0 < \alpha_i < C} \quad (i \in \mathcal{F}) \quad \Rightarrow \quad \lambda_i = \mu_i = 0$$
$$\Rightarrow \quad \sum_{j \in \mathcal{F}} y_j \alpha_j K_{ij} = y_i - \sum_{j \in \mathcal{A}_C} y_j \alpha_j K_{ij} - b \tag{12a}$$

$$\boxed{\alpha_i = 0} \quad (i \in \mathcal{A}_0) \quad \Rightarrow \quad \lambda_i = y_i E_i > 0$$
$$\Rightarrow \quad \mu_i = 0 \tag{12b}$$

$$\boxed{\alpha_i = C} \quad (i \in \mathcal{A}_C) \quad \Rightarrow \quad \lambda_i = 0$$
$$\Rightarrow \quad \mu_i = -y_i E_i > 0 \tag{12c}$$

The above conditions are exploited in each iteration step k. Case (12a) establishes the linear system in step A1 for the currently free variables $i \in \mathcal{F}^k$. Cases (12b) and (12c) are the conditions that must be fulfilled for the variables in $\mathcal{A}^k = \mathcal{A}_0^k \cup \mathcal{A}_C^k$ in the optimum, i.e., step A2 of the algorithm searches for the worst violator among these variables. Note that $\mathcal{A}_0^k \cap \mathcal{A}_C^k = \emptyset$ because $\alpha_i = 0$ and $\alpha_i = C$ cannot be true simultaneously. The variables in $\mathcal{A}_C^k$ are the *bounded support vectors* and also occur in the linear system of case (12a).

The implementation uses the *coefficients* $a_i = y_i \alpha_i$ instead of the Lagrange multipliers α_i. This is done to keep the same formulation for the regression algorithm in Sec. 3, and because it is slightly faster in computation. With this modification, in step A1 the linear system

$$\mathbf{H}^k \mathbf{a}^k = \mathbf{c}^k \tag{13}$$

with

$$\left.\begin{aligned} a_i^k &= y_i \alpha_i^k \\ h_{ij}^k &= K_{ij} \\ c_i^k &= y_i - \sum_{j \in \mathcal{A}_C^k} a_j^k K_{ij} - b \end{aligned}\right\} \quad \text{for } \; i,j \in \mathcal{F}^k \tag{14}$$

has to be solved. If $\mathcal{F}^k$ contains p free variables, then $\mathbf{H}^k$ is a $p \times p$ matrix. It is positive definite since positive definite kernels are assumed for all algorithms. For that (13) can be solved by the methods described in Sec. 4. Step A2 computes

$$\lambda_i^k = +y_i E_i^k \quad \text{for } \; i \in \mathcal{A}_0^k \tag{15a}$$

$$\mu_i^k = -y_i E_i^k \quad \text{for } \; i \in \mathcal{A}_C^k \tag{15b}$$

and checks if they are positive, i.e. if the KKT conditions are valid for $i \in \mathcal{A}^k = \mathcal{A}_0^k \cup \mathcal{A}_C^k$. Among the negative multipliers, the most negative one is selected and moved to $\mathcal{F}^k$.

2.2 Classification with variable bias term

Most implementations do not keep the bias term fixed but compute it during optimization. In that case b is a primal variable, and the Lagrangian (2) can be minimized with respect to it:

$$\frac{\partial L_\mathrm{P}}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^N y_i \alpha_i = 0 \tag{16}$$

On the one hand (16) removes the last term from (4), on the other hand it is an additional constraint that must be considered in the optimization problem:

$$\min_{\boldsymbol{\alpha}} \quad J_\mathrm{d}(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j K_{ij} - \sum_{i=1}^N \alpha_i \tag{17a}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \,, \quad i = 1, \ldots, N \tag{17b}$$

$$\sum_{i=1}^N y_i \alpha_i = 0 \tag{17c}$$

This modification changes the Lagrangian (10) to

$$\begin{aligned} L_{\mathrm{d}}(\boldsymbol{\alpha},\boldsymbol{\lambda},\boldsymbol{\mu},\nu) =& \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K_{ij} - \sum_{i=1}^{N}\alpha_i \\ & - \sum_{i=1}^{N}\lambda_i\alpha_i - \sum_{i=1}^{N}\mu_i(C-\alpha_i) - \nu\sum_{i=1}^{N} y_i\alpha_i \end{aligned} \tag{18}$$

and its derivatives to

$$\frac{\partial L_{\mathrm{d}}}{\partial \alpha_i} = y_i \sum_{j=1}^{N} y_j \alpha_j K_{ij} - 1 - \lambda_i + \mu_i - \nu y_i = 0\,, \quad i = 1,\ldots,N \tag{19}$$

where ν is the Lagrange multiplier of the equality constraint (17c). It can be easily seen that $\nu = -b$, i.e., L_{d} is the same as (10) with the important difference that b is not fixed any more. With this and again with $a_i = y_i\alpha_i$ the linear system becomes

$$\begin{pmatrix} \mathbf{H}^k & \mathbf{e} \\ \mathbf{e}^{\mathrm{T}} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{a}^k \\ b^k \end{pmatrix} = \begin{pmatrix} \mathbf{c}^k \\ d^k \end{pmatrix} \begin{matrix} \}\ p \text{ rows} \\ \}\ 1 \text{ row} \end{matrix} \tag{20}$$

with

$$d^k = -\sum_{j\in\mathcal{A}_C^k} a_j^k \quad \text{and} \quad \mathbf{e} = (1,\ldots,1)^{\mathrm{T}} \tag{21}$$

One possibility to solve this indefinite system is to use factorization methods for indefinite matrices, e.g., the Bunch-Parlett decomposition [3]. But since we retain the assumption that $K(\mathbf{x}_i,\mathbf{x}_j)$ is positive definite, the Cholesky decomposition $\mathbf{H} = \mathbf{R}^{\mathrm{T}}\mathbf{R}$ is available (see Sec. 4), and the system (20) can be solved by exploiting its block structure. For that, a Gauss transform is applied to the blocks of the matrix, i.e., the first block row is multiplied by $(\mathbf{u}^k)^{\mathrm{T}} := \mathbf{e}^{\mathrm{T}}(\mathbf{H}^k)^{-1}$. Subtracting the second row yields

$$(\mathbf{u}^k)^{\mathrm{T}}\mathbf{e}b^k = (\mathbf{u}^k)^{\mathrm{T}}\mathbf{c}^k - d^k \tag{22}$$

Since this is a scalar equation, it can be simply divided by $(\mathbf{u}^k)^{\mathrm{T}}\mathbf{e}$ in order to find b^k. This technique is very effective here because only *one* additional row/column has been appended to $\mathbf{H}^k$. The complete solution of the block system is done by the following procedure:

- Solve $(\mathbf{R}^k)^{\mathrm{T}}\mathbf{R}^k\mathbf{u}^k = \mathbf{e}$ for $\mathbf{u}^k$.
- $b^k = -\Big(\sum_{j\in\mathcal{A}_C^k} a_j^k + \sum_{j\in\mathcal{A}_C^k} u_j^k c_j^k\Big)\Big/\sum_{j\in\mathcal{A}_C^k} u_j^k$
- Solve $(\mathbf{R}^k)^{\mathrm{T}}\mathbf{R}^k\mathbf{a}^k = \mathbf{c}^k - \mathbf{e}b^k$ for $\mathbf{a}^k$.

The computation of λ_i^k and μ_i^k remains the same as in (15) for fixed bias term.

An additional topic has to be considered here: For a variable bias term, the *Linear Independence Constraint Qualification (LICQ)* [7] is violated when for each α_i one inequality constraint is active, e.g., when the algorithm is initialized with $\alpha_i = 0$ for $i = 1,\ldots,N$. The algorithm uses *Bland's rule* to avoid *cycling* in these cases.

3 Support Vector Machine Regression

Like in classification, we start from the linear problem. The goal is to fit a linear function $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$ to a given data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Whereas most other learning methods minimize the sum of squared errors, SVMs try to find a maximal flat function, so that all data lie within an *insensitivity zone* of size ε around the function. Outliers are treated by *two* sets of slack variables ξ_i and ξ_i^* measuring the distance above and below the insensitivity zone, respectively, see Fig. 4 (for a nonlinear example) and [10]. This concept results in the following primal problem:

$$\begin{aligned}
\min_{\mathbf{w},\boldsymbol{\xi},\boldsymbol{\xi}^*} \quad & J_{\mathrm{p}}(\mathbf{w},\boldsymbol{\xi},\boldsymbol{\xi}^*) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) && \text{(23a)}\\
\text{s.t.} \quad & y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i - b \le \varepsilon + \xi_i && \text{(23b)}\\
& \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b - y_i \le \varepsilon + \xi_i^* && \text{(23c)}\\
& \xi_i, \xi_i^* \ge 0, \quad i = 1,\dots,N. && \text{(23d)}
\end{aligned}$$

To apply the same technique as for classification, the Lagrangian

$$\begin{aligned}
&L_{\mathrm{p}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}, \boldsymbol{\beta}^*) = \\
&\quad \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) - \sum_{i=1}^{N}(\beta_i\xi_i + \beta_i^*\xi_i^*) \\
&\quad - \sum_{i=1}^{N}\alpha_i(\varepsilon + \xi_i + y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i - b) - \sum_{i=1}^{N}\alpha_i^*(\varepsilon + \xi_i^* - y_i + \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b)
\end{aligned} \tag{24}$$

of the primal problem (23) is needed. $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}^*$, $\boldsymbol{\beta}$ and $\boldsymbol{\beta}^*$ are the dual variables, i.e., the Lagrange multipliers of the primal constraints. As in Sec. 2, the saddle point condition can be exploited to minimize L_{p} with respect to the primal variables $\mathbf{w}$, $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^*$, which results in a function that only contains $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}^*$ and b:

$$\begin{aligned}
L_{\mathrm{p}}^*(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*, b) =& \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K_{ij} \\
& - \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)y_i + \varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + b\sum_{i=1}^{N}(\alpha_i - \alpha_i^*)
\end{aligned} \tag{25}$$

The scalar product $\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$ has already been substituted by the kernel function $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ to introduce nonlinearity to the SVM, see (5) and Fig. 4. The bias term b is untouched so far because the next sections offer again two possibilities (fixed and variable b) that lead to different algorithms. In both cases, the inequality constraints

$$0 \le \alpha_i^{(*)} \le C\,, \quad i = 1,\dots,N \tag{26}$$

resulting from (47d) must be fulfilled. Since a data sample cannot lie above *and* below the insensitivity zone simultaneously, the dual variables $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$

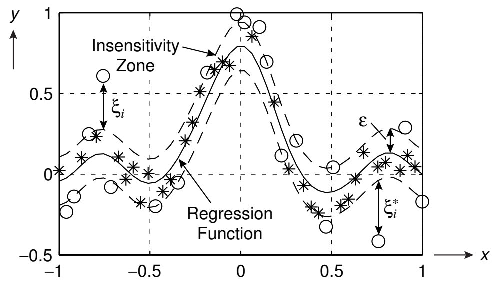


Fig. 4. Nonlinear support vector machine regression

are not independent. At least one of the primal constraints (23b) and (23c) must be fulfilled with equality for each i, i.e., the KKT conditions imply that $\alpha_i \alpha_i^* = 0$. The output of regression SVMs is computed as

$$f(\mathbf{x}) = \sum_{\alpha_i^{(*)} \neq 0} (\alpha_i - \alpha_i^*) K(\mathbf{x}_i, \mathbf{x}) + b \tag{27}$$

In the following, the notation $\alpha_i^{(*)}$ is used as an abbreviation if an (in-) equality is valid for both α_i and α_i^*.

3.1 Regression with fixed bias term

The kernel function is still assumed to be positive definite so that b can be kept fixed or even omitted. The QP problem (9) is similar for regression SVMs. It is built from (25) and (26) by treating the bias term as a fixed parameter:

$$\min_{\boldsymbol{\alpha},\, \boldsymbol{\alpha}^*} \quad J_\mathrm{d}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K_{ij} - \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) y_i + \varepsilon \sum_{i=1}^{N} (\alpha_i + \alpha_i^*) + b \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \tag{28a}$$

$$\text{s.t.} \quad 0 \leq \alpha_i^{(*)} \leq C\,, \quad i = 1, \ldots, N \tag{28b}$$

Since optimization methods usually assume *minimization* problems rather than *maximization* problems, we set $J_\mathrm{d}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = -L_\mathrm{p}^*(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*, b)$ with fixed b. For $b = 0$ the last term vanishes so that (28) differs from the standard problem (36) only in the absence of the equality constraint (36c). To find the steps A1 and A2 of an active-set algorithm that solves (28) its Lagrangian

$$L_{\mathrm{d}}(\boldsymbol{\alpha},\boldsymbol{\alpha}^*,\boldsymbol{\lambda},\boldsymbol{\lambda}^*,\boldsymbol{\mu},\boldsymbol{\mu}^*) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K_{ij} - \sum_{i=1}^{N}(\alpha_i-\alpha_i^*)y_i + \varepsilon\sum_{i=1}^{N}(\alpha_i+\alpha_i^*) + b\sum_{i=1}^{N}(\alpha_i-\alpha_i^*) - \sum_{i=1}^{N}\lambda_i\alpha_i - \sum_{i=1}^{N}\mu_i(C-\alpha_i) - \sum_{i=1}^{N}\lambda_i^*\alpha_i^* - \sum_{i=1}^{N}\mu_i^*(C-\alpha_i^*) \tag{29}$$

is required. Compared to classification, two additional sets of multipliers λ_i^* (for $\alpha_i^* \geq 0$) and μ_i^* (for $\alpha_i^* \leq C$) are needed here. Using the prediction errors $E_i = f(\mathbf{x}_i) - y_i$, the KKT conditions for $i = 1, \ldots, N$ are

$$\frac{\partial L_{\mathrm{d}}}{\partial \alpha_i} = \varepsilon + E_i - \lambda_i + \mu_i = 0 \tag{30a}$$

$$\frac{\partial L_{\mathrm{d}}}{\partial \alpha_i^*} = \varepsilon - E_i - \lambda_i^* + \mu_i^* = 0 \tag{30b}$$

$$0 \leq \alpha_i^{(*)} \leq C \tag{30c}$$

$$\lambda_i^{(*)} \geq 0, \quad \mu_i^{(*)} \geq 0 \tag{30d}$$

$$\alpha_i^{(*)}\lambda_i^{(*)} = 0, \quad (C-\alpha_i^{(*)})\mu_i^{(*)} = 0 \tag{30e}$$

According to α_i and α_i^*, five cases have to be considered:

$\boxed{0 < \alpha_i < C,\ \alpha_i^* = 0}$ $(i \in \mathcal{F})$

$$\begin{aligned} &\Rightarrow\quad \lambda_i = \mu_i = \mu_i^* = 0,\ \lambda_i^* = 2\varepsilon > 0 \\ &\Rightarrow\quad \sum_{j\in\mathcal{F}^{(*)}} a_jK_{ij} = y_i - \varepsilon - \sum_{j\in\mathcal{A}_C^{(*)}} a_jK_{ij} \end{aligned} \tag{31a}$$

$\boxed{0 < \alpha_i^* < C,\ \alpha_i = 0}$ $(i \in \mathcal{F}^*)$

$$\begin{aligned} &\Rightarrow\quad \lambda_i^* = \mu_i = \mu_i^* = 0,\ \lambda_i = 2\varepsilon > 0 \\ &\Rightarrow\quad \sum_{j\in\mathcal{F}^{(*)}} a_jK_{ij} = y_i + \varepsilon - \sum_{j\in\mathcal{A}_C^{(*)}} a_jK_{ij} \end{aligned} \tag{31b}$$

$\boxed{\alpha_i = \alpha_i^* = 0}$ $(i \in \mathcal{A}_0 \cap \mathcal{A}_0^*)$

$$\begin{aligned} &\Rightarrow\quad \lambda_i = \varepsilon + E_i > 0, \quad \lambda_i^* = \varepsilon - E_i > 0 \\ &\Rightarrow\quad \mu_i = 0, \quad \mu_i^* = 0 \end{aligned} \tag{31c}$$

$\boxed{\alpha_i = C,\ \alpha_i^* = 0}$ $(i \in \mathcal{A}_C)$

$$\begin{aligned} &\Rightarrow\quad \lambda_i = 0, \quad \lambda_i^* = \varepsilon - E_i > 0 \\ &\Rightarrow\quad \mu_i = -\varepsilon - E_i > 0, \quad \mu_i^* = 0 \end{aligned} \tag{31d}$$

$\boxed{\alpha_i = 0,\ \alpha_i^* = C}\quad (i \in \mathcal{A}_C^*)$

$$\begin{aligned} &\Rightarrow \quad \lambda_i = \varepsilon + E_i > 0, \quad \lambda_i^* = 0 \\ &\Rightarrow \quad \mu_i = 0, \quad \mu_i^* = -\varepsilon + E_i > 0 \end{aligned} \tag{31e}$$

Obviously, there are more than five cases but only these five can occur due to $\alpha_i \alpha_i^* = 0$: If one of the variables is free ((31a) and (31b)) or equal to C ((31d) and (31e)), the other one must be zero. Similar to classification, the cases (31a) and (31b) form the linear system for step A1 and the cases (31c) – (31e) are the conditions to be checked in step A2 of the algorithm.

The regression algorithm uses the SVM coefficients $a_i = \alpha_i - \alpha_i^*$. With this abbreviation, the number of variables reduces from $2N$ to N and many similarities to classification can be observed. The linear system is almost the same as (13):

$$\mathbf{H}^k \mathbf{a}^k = \mathbf{c}^k \tag{32}$$

with

$$\begin{aligned} &\left.\begin{aligned} a_i^k &= \alpha_i^k - \alpha_i^{*k} \\ h_{ij}^k &= K_{ij} \end{aligned}\right\} \quad \text{for } \ i \in \mathcal{F}^k \cup \mathcal{F}^{*k} \\ &c_i^k = y_i - \sum_{j \in \mathcal{A}_C^k \cup \mathcal{A}_C^{*k}} a_j^k K_{ij} + \begin{cases} -\varepsilon & \text{for } \ i \in \mathcal{F}^k \\ +\varepsilon & \text{for } \ i \in \mathcal{F}^{*k} \end{cases} \end{aligned} \tag{33}$$

only the right hand side has been modified by $\pm\varepsilon$. Step A2 of the algorithm computes

$$\begin{aligned} \lambda_i^k &= \varepsilon + E_i^k \\ \lambda_i^{*k} &= \varepsilon - E_i^k \end{aligned} \quad \text{for } \ i \in \mathcal{A}_0^k \cup \mathcal{A}_0^{*k} \tag{34a}$$

and

$$\begin{aligned} \mu_i^k &= -\varepsilon - E_i^k \\ \mu_i^{*k} &= -\varepsilon + E_i^k \end{aligned} \quad \text{for } \ i \in \mathcal{A}_C^k \cup \mathcal{A}_C^{*k} \tag{34b}$$

Again, these multipliers are checked for positiveness, and the variable with the most negative multiplier is transferred to $\mathcal{F}^k$ or $\mathcal{F}^{*k}$, respectively.

3.2 Regression with variable bias term

If the bias term is treated as a variable, (25) can be minimized ($\partial L_\mathrm{d}^* / \partial b = 0$) with respect to b resulting in

$$\sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0\,. \tag{35}$$

Like in classification, this condition removes the last term from (28a) but must be treated as additional equality constraint:

$$\begin{aligned}\min_{\boldsymbol{\alpha},\,\boldsymbol{\alpha}^*} \quad J_\mathrm{d}(\boldsymbol{\alpha},\boldsymbol{\alpha}^*) =& \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K_{ij} \\ &- \sum_{i=1}^{N}(\alpha_i-\alpha_i^*)y_i + \varepsilon\sum_{i=1}^{N}(\alpha_i+\alpha_i^*)\end{aligned} \tag{36a}$$

$$\text{s.t.} \quad 0 \le \alpha_i^{(*)} \le C\,, \quad i=1,\dots,N \tag{36b}$$

$$\sum_{i=1}^{N}(\alpha_i-\alpha_i^*) = 0 \tag{36c}$$

The Lagrangian of this QP problem is nearly identical to (29):

$$\begin{aligned}&L_\mathrm{d}(\boldsymbol{\alpha},\boldsymbol{\alpha}^*,\boldsymbol{\lambda},\boldsymbol{\lambda}^*,\boldsymbol{\mu},\boldsymbol{\mu}^*,\nu) = \\ &\quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K_{ij} \\ &\quad - \sum_{i=1}^{N}(\alpha_i-\alpha_i^*)y_i + \varepsilon\sum_{i=1}^{N}(\alpha_i+\alpha_i^*) - \nu\sum_{i=1}^{N}(\alpha_i-\alpha_i^*) \\ &\quad - \sum_{i=1}^{N}\lambda_i\alpha_i - \sum_{i=1}^{N}\mu_i(C-\alpha_i) - \sum_{i=1}^{N}\lambda_i^*\alpha_i^* - \sum_{i=1}^{N}\mu_i^*(C-\alpha_i^*)\end{aligned} \tag{37}$$

We already observed for classification that the Lagrange multiplier ν of the equality constraint is basically the bias term ($\nu = -b$) that is treated as a variable. Compared to fixed b, (30) also comprises the equality constraint (36c), but the five cases (31) do not change. Consequently, the coefficients $a_i = \alpha_i - \alpha_i^*$ with $i \in \mathcal{F} \cup \mathcal{F}^*$ and the bias term b are computed by solving a block system having the same structure as (20):

$$\begin{pmatrix}\mathbf{H}^k & \mathbf{e} \\ \mathbf{e}^\mathrm{T} & 0\end{pmatrix}\begin{pmatrix}\mathbf{a}^k \\ b^k\end{pmatrix} = \begin{pmatrix}\mathbf{c}^k \\ d^k\end{pmatrix} \begin{matrix}\}\ p \text{ rows} \\ \}\ 1 \text{ row}\end{matrix} \tag{38}$$

with

$$d^k = -\sum_{j\in\mathcal{A}_C^k\cup\mathcal{A}_C^{*k}} a_j^k \quad \text{and} \quad \mathbf{e} = (1,\dots,1)^\mathrm{T} \tag{39}$$

i.e., the only difference is d^k which considers the indices in both $\mathcal{A}_C^k$ and $\mathcal{A}_C^{*k}$. This system can be solved by the algorithm derived in Sec. 2. The KKT conditions in step A2 remain exactly the same as (34).

4 Implementation details

The active-set algorithm has been implemented as C MEX-file under MATLAB for classification and regression problems. It can handle both fixed and variable bias terms. Approximately the following memory is required:

- N floating point elements for the coefficient vector,
- N integer elements for the index vector,
- $p(p+3)/2$ floating point elements for the triangular matrix and the right hand side of the linear system,

where p is the number of free variables, i.e., those with $0 < \alpha_i^{(*)} < C$. As this number is unknown in the beginning, the algorithm starts with an initial amount of memory and increases it whenever variables are added. The index vector is needed to keep track of the stets $\mathcal{F}^{(*)}$, $\mathcal{A}_C^{(*)}$ and $\mathcal{A}_0^{(*)}$. It is also used as pivot vector for the Cholesky decomposition described in Sec. 4.1. Since most of the coefficients a_i are zero in the optimum, we start with $a_i = 0$ for $i = 1, \dots, N$ as initial feasible solution.

Since all algorithms assume positive definite kernel functions, the kernel matrix has a Cholesky decomposition $\mathbf{H} = \mathbf{R}^{\mathrm{T}}\mathbf{R}$, where $\mathbf{R}$ is an upper triangular matrix. For a fixed bias term, the solution of the linear system in step A1 is found by simple backsubstitution. For variable bias term, the block-algorithm described in Sec. 2 is used.

4.1 Cholesky decomposition with pivoting

Although the Cholesky decomposition is numerically very stable, the active-set algorithm uses diagonal pivoting by default, because $\mathbf{H}$ may be "nearly indefinite", i.e., it may become indefinite by round-off errors during the computation. This occurs e.g. for Gaussians having large widths. There are two ways to cope with this problem: First, to use Cholesky decomposition with pivoting, and second, to slightly enlarge the diagonal elements to make $\mathbf{H}$ "more definite". The first case allows to extract the largest positive definite part of $\mathbf{H} = (h_{ij})$. All variables corresponding to the rest of the matrix are set to zero then.

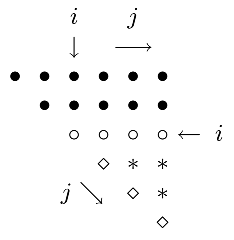


Fig. 5. The i-th step of the Cholesky decomposition computes the i-th row ($\circ$) of the matrix from the already finished elements ($\bullet$). The diagonal elements ($\diamond$) are updated whereas the rest ($*$) remains untouched.

Usually the Cholesky decomposition is computed using `axpy` operations [3]. However, the pivoting strategy needs the updated diagonal elements in each step, as they would be available if *outer product* updates were applied. Since these require many accesses to matrix elements, a mixed procedure is implemented that updates only the diagonal elements and uses `axpy` operations otherwise:

```
Compute for $i = 1, \dots, p$:
    Find $k = \arg\max\{|\bar{h}_{ii}|, \dots, |\bar{h}_{pp}|\}$.
    Swap lines and columns $i$ and $k$ symmetrically.
    Compute $r_{ii} = \sqrt{\bar{h}_{ii}}$.
    Compute for $j = i+1, \dots, p$:
        $r_{ij} = \big(h_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\big) / r_{ii}$
        $\bar{h}_{jj} \leftarrow \bar{h}_{jj} - r_{ij}^2$
```

where $\bar{h}_{jj}$ are the updated diagonal elements and "$\leftarrow$" indicates the update process. The result can be written as

$$\mathbf{PHP}^{\mathrm{T}} = \mathbf{R}^{\mathrm{T}}\mathbf{R} \tag{40}$$

with the permutation matrix $\mathbf{P}$. Of course the implementation uses an pivot vector instead of the complete matrix. Besides that, only the upper triangular part of $\mathbf{R}$ is stored, i.e., only memory for $p(p+1)/2$ elements is needed. This algorithm is almost as fast as the standard Cholesky decomposition.

4.2 Adding variables

Since the active-set algorithm adds or removes only one variable per step, it is reasonable to modify the existing Cholesky decomposition instead of computing it form scratch [2]. These techniques are faster but less accurate than the method described in Sec. 4.1, because they cannot be used with pivoting. The only way to get along with definiteness problems is to slightly enlarge the diagonal elements h_{jj}.

If a p-th variable is added to the linear system, a new column and a new row are appended to $\mathbf{H}$. Since an arbitrary element r_{ij} of the Cholesky decomposition is completely computed from the diagonal element r_{ii} and the sub-columns i and j above the i-th row (see Sec. 4.1), only the last column needs to be computed:

```
Compute for $i = 1, \dots, p$:
    $r_{ip} = \big(h_{ip} - \sum_{k=1}^{i-1} r_{ki} r_{kp}\big) / r_{ii}$
```

The columns $1, \dots, p-1$ remain unchanged. This technique is only effective if the *last* column is appended. If an arbitrary column is inserted, elements of $\mathbf{R}$ need to be re-computed.

4.3 Removing variables

Removing variables from an existing Cholesky decomposition is a more sophisticated process. For that, we introduce an unknown matrix $\mathbf{A} \in \mathbb{R}^{M \times p}$ with

$$\mathbf{H} = \mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{A}^{\mathrm{T}}\mathbf{A} \quad \text{and} \quad \mathbf{QA} = \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}, \tag{41}$$

i.e., $\mathbf{R}$ also results from the QR decomposition of $\mathbf{A}$. Removing a variable from the Cholesky decomposition is equivalent to removing a column from $\mathbf{A}$:

$$\mathbf{Q}\tilde{\mathbf{A}} = \begin{pmatrix} \mathbf{r}_1 \dots \mathbf{r}_{k-1}, \mathbf{r}_{k+1} \dots \mathbf{r}_p \\ \mathbf{0} \end{pmatrix}, \tag{42}$$

The non-zero part of the right hand side matrix is of size $p \times (p-1)$ now because the k-th column is missing. It is “nearly” an upper triangular matrix, only each of the columns $k+1, \dots, p$ has one element below the diagonal:

$$\begin{array}{cccccc}
 & & k-1 & k+1 & & \\
 & & \downarrow & \downarrow & & \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
 & \bullet & \bullet & \bullet & \bullet & \bullet \\
 & & \bullet & \bullet & \bullet & \bullet \\
 & & & * & * & * \\
 & & & \circ & * & * \\
 & & & & \circ & * \\
 & & & & & \circ
\end{array}$$

The sub-diagonal elements are removed by Givens rotations $\mathbf{\Omega}_{k+1}, \dots, \mathbf{\Omega}_p$:

$$\underbrace{\mathbf{\Omega}_p \cdots \mathbf{\Omega}_{k+1}\mathbf{Q}}_{\tilde{\mathbf{Q}}}\,\tilde{\mathbf{A}} = \begin{pmatrix} \tilde{\mathbf{R}} \\ \mathbf{0} \end{pmatrix} \tag{43}$$

$\tilde{\mathbf{R}}$ is the Cholesky factor of the reduced matrix $\tilde{\mathbf{H}}$.

However, it should be mentioned that modification techniques do not lead to a strong acceleration since most of the computation time is currently spent to check the KKT conditions in $\mathcal{A}$ (during step A2 of the algorithm). By default, the algorithm uses Cholesky decomposition with pivoting when a variable is added to its linear system, and the above modification strategy when a variable is removed.

4.4 Approximating the solution

Active-set methods check the KKT conditions of the *complete* active set in each step. As pointed out above, this is a huge computational effort which is only reasonable for algorithms that make enough progress in each step. Typical working-set algorithms, on the other hand, follow the opposite strategy: They perform only simple steps and therefore try to reduce the number of KKT evaluations to a minimum by certain heuristics.

The complete KKT check of active-set methods can be exploited to approximate the solution with a given number N_{SVmax} of support vectors. Remember that the N_{SV} support vectors comprise

- N_{f} free support vectors $0 < \alpha_i^{(*)} < C$ (i.e., those with $i \in \mathcal{F}^{(*)}$).
- $N_{\mathrm{SV}} - N_{\mathrm{f}}$ bounded support vectors $\alpha_i^{(*)} = C$ (i.e., those with $i \in \mathcal{A}_C^{(*)}$).

The algorithm simply stops when at the end of step A3 a solution with more than N_{SVmax} support vectors is computed for the first time:

- If $N_{\mathrm{SV}}^k > N_{\mathrm{SVmax}}$ then stop with the previous solution.
- Otherwise accept the new solution and go to step A1.

The first case can only happen if in step A2 an $i \in \mathcal{A}_0^{(*)}$ was selected and in step A3 no variable is moved back to $\mathcal{A}_0^{(*)}$. All other cases do not increase the number of support vectors.

This heuristic approach does not always lead to a better approximation if more support vectors are allowed. However, experiments (like in Sec. 5) show that typically only a small fraction of support vectors significantly reduces the approximation error.

5 Results

This section shows experimental results for classification and regression. The proposed active-set method is compared to the well-known working-set method LIBSVM [1] for different problem settings. Since the optimization strategies are quite different, mainly the *computation time* is considered to measure the performance. For a better understanding of the results it must be admitted that the implementation of LIBSVM contains some acceleration techniques (e.g., the use of BLAS routines) that are not yet implemented for the active-set method. Additionally, it had always 40 MB cache available because it gets significantly slower if the cache size is chosen too small. All experiments were done on a 800 MHz Pentium-III PC having 256 MB RAM.

5.1 Classifying demographic data

The first example considers the "Adult" database from the *UCI machine learning repository* [6] that has been studied in several publications. The goal is to determine from 14 demographic features weather a person earns more than $ 50,000 per year. All features have been normalized to $[-1, 1]$; nominal features were converted to numeric values. In order to limit the computation time in the critical cases, a subset to 1000 samples has been selected as training

Table 1. Classification: Variation of C

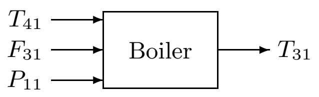
	C	10^0	10^1	10^2	10^3	10^4	10^5	10^6	10^7
Active Set	Time	8.7 s	7.0 s	6.9 s	6.5 s	6.9 s	8.0 s	9.3 s	10.2 s
	N_{SV}	237	214	196	186	176	162	149	141
	N_{f}	16	24	52	88	117	130	137	138
	Bias	0.9726	1.3980	4.8049	9.7528	29.165	31.678	−6.433	−87.149
Active Set ($b = 0$)	Time	7.6 s	7.6 s	6.8 s	6.4 s	7.6 s	7.1 s	9.3 s	10.2 s
	N_{SV}	237	213	199	184	176	163	149	141
	N_{f}	13	25	56	87	114	131	137	138
LIB SVM	Time	0.2 s	0.2 s	0.3 s	0.8 s	4.0 s	27.2 s	263.5 s	1082.7 s
	N_{SV}	237	214	196	186	176	162	148	142
	N_{f}	16	24	52	88	118	131	137	139
	Bias	0.9724	1.3976	4.8049	9.7403	29.156	32.146	−1.182	−110.72

data set. The SVMs use Gauss kernels with width $\sigma = 3$ and a precision of $\tau = 10^{-3}$ to check the KKT conditions.

Table 1 shows the results when the upper bound C is varied. Whereas the active-set method is nearly insensitive with respect to C, the computation time of LIBSVM differs by several magnitudes. Typically working-set methods perform better when the number N_{f} of free support vectors is small. Also comparison between the standard SVM and the no-bias SVM (i.e., with bias term fixed at $b = 0$) can be found in Tab. 1. It shows that there is no need for a bias term when positive definite kernels are used. Although a missing bias generally leads to more support vectors, the results are very close to the standard SVM here – even if the bias term takes large values.

5.2 Estimating the outlet temperature of a boiler

The following example shows how to use the regression algorithm in system identification. We use a data set described in detail in [12]. The task is to estimate the outlet temperature T_{31} of a *high efficiency (condensing) boiler*. The inputs are the system temperature T_{41}, the water flow F_{31} and the burner



$$\begin{aligned} T_{31}(k) = f(&T_{41}(k), T_{41}(k-1), T_{41}(k-2),\\ &F_{31}(k), F_{31}(k-1), F_{31}(k-2),\\ &P_{11}(k), P_{11}(k-1), P_{11}(k-2),\\ &T_{31}(k-1), T_{31}(k-2)) \end{aligned} \qquad (44)$$

Fig. 6. Block diagram and regression model of the boiler

output P_{11}. Based on a theoretical analysis, second order dynamics are assumed for the output and all inputs, so the model has 11 regressors, see Fig. 6. The training data set consists of 3344 samples, the validation data set of 2926 samples. Table 2 compares the active-set algorithm and LIBSVM when the upper bound C is varied. The SVM uses Gauss kernels having a width of $\sigma = 3$. The insensitivity zone is properly set to $\varepsilon = 0.01$, the precision used to check the KKT conditions is $\tau = 10^{-4}$. Both methods compute SVMs with variable bias term in order to make the results comparable.

Table 2. Regression: Variation of C

	C	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}	10^{5}
Active Set	Time	2776 s	713.0 s	159.6 s	28.0 s	11.3 s	11.7 s	17.6 s	27.3 s
	RMSE	0.0330	0.0164	0.0097	0.0068	0.0062	0.0061	0.0064	0.0069
	N_{SV}	1938	954	426	142	92	85	92	117
	N_{f}	4	10	20	34	51	74	91	117
LIB SVM	Time	8.5 s	4.9 s	3.3 s	3.6 s	8.0 s	55.6 s	314.6 s	∞
	RMSE	0.0330	0.0164	0.0097	0.0068	0.0062	0.0062	0.0063	?
	N_{SV}	1939	964	432	147	93	92	102	?
	N_{f}	6	23	34	45	54	82	100	?

Concerning computation time, Tab. 2 shows that LIBSVM can efficiently handle a large number N_{SV} of support vectors (here with only few free α_i) whereas the active-set method shows its strength if N_{SV} is small. For $C = 10^5$ LIBSVM did not converge; it was aborted after 12 hours. In this example, $C = 10^3$ is the optimal setting concerning support vectors and error (RMSE on the validation data set). Also the active-set algorithm's dependency on N_{f}^2 (see Sec. 1) is not critical: If the number of support vectors increases, typically most of the Lagrange multipliers are bounded at C so that N_{f} remains small. A comparison with Tab. 1 implies that the computation time for the active-set method mainly depends on the *number* of support vectors, whereas the *ratio* of free and bounded support vectors has strong influence on working-set methods.

Table 3 compares both algorithms for different precisions τ when the Gaussians' width is small ($\sigma = 1$) and C is set to 10^3. In this setting all support vectors are free, which is an extreme case but not unusual [8]. Both algorithms

Table 3. Regression: Variation of τ for $\sigma = 1$

	τ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
Active Set	Time	0.1 s	6.8 s	20.0 s	25.4 s	26.1 s	26.2 s	26.3 s	26.2 s
	RMSE	0.0581	0.0104	0.0091	0.0090	0.0090	0.0090	0.0090	0.0090
	N_{SV}	9	71	107	128	133	133	133	133
	N_{f}	0	0	0	0	0	0	0	0
LIB SVM	Time	0.5 s	2.9 s	8.2 s	18.0 s	55.8 s	134.7 s	197.1 s	282.7 s
	RMSE	0.0315	0.0114	0.0092	0.0091	0.0090	0.0090	0.0090	0.0090
	N_{SV}	45	156	130	131	132	131	131	131
	N_{f}	0	0	0	0	0	0	0	0

do not change the number of support vectors for precisions smaller then 10^{-5}. Whereas the active-set method does not need more time to fulfill a higher precision, LIBSVM's computation time strongly increases. This effect is even more significant if the Gaussians are broader ($\sigma = 5$, see Tab. 4), because the kernel matrix is ill-conditioned then. Table 4 also confirms the observation that working-set methods perform worse when the number of *free* support vectors increases. Generally, LIBSVM seems to produce more support vectors, in particular for low precisions and in the ill-conditioned cases. I.e., the active-set method leads to more compact models for a given precision.

Table 4. Regression: Variation of τ for $\sigma = 5$

	τ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
Active Set	Time	0.1 s	1.6 s	10.4 s	12.1 s	12.4 s	12.4 s	12.4 s	12.5 s
	RMSE	0.0175	0.0077	0.0060	0.0059	0.0059	0.0059	0.0059	0.0059
	N_{SV}	9	34	89	94	96	96	96	96
	N_{f}	0	4	48	45	45	45	45	45
LIB SVM	Time	0.5 s	6.6 s	17.8 s	32.1 s	118.9 s	431.4 s	471.2 s	1217 s
	RMSE	0.0220	0.0059	0.0059	0.0060	0.0060	0.0060	0.0060	0.0060
	N_{SV}	30	86	99	103	102	102	102	102
	N_{f}	30	85	66	61	57	57	57	57

A final experiment demonstrates the approximation method described in Sec. 4.4. For $C = 10^3$ and $\sigma = 1$ the complete model contains 131 support vectors. However, Fig. 7 shows that the solution can be approximated with much less support vectors, e.g. 10 – 15 %. Whereas the objective function is still decreasing, more support vector do not significantly reduce the approximation error.

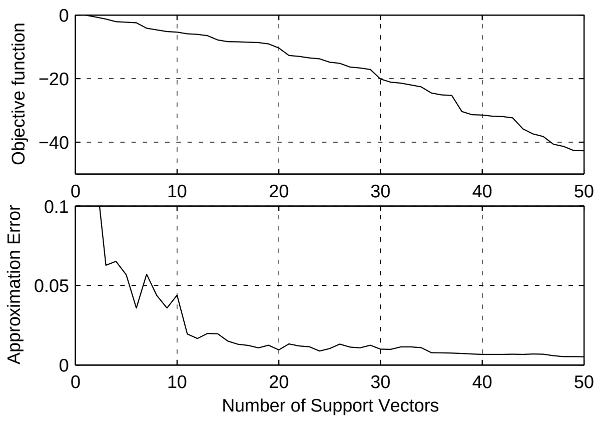


Fig. 7. Approximation of the solution

6 Conclusions

An active-set algorithm has been proposed for SVM classification and regression. The general strategy has been adapted to these problems for both fixed and variable bias term. The result is a robust algorithm that requires approximately $2N + \frac{1}{2}N_\mathrm{f}^2$ elements of memory, where N is the number of data and N_f the number of free support vectors. Simulation results show that active-set algorithms are advantageous

- when the number of SVs is small.
- when the fraction of bounded support vectors is small.
- when high precision is needed.
- when solving regression problems.

Additionally, the algorithm's KKT check can be exploited to approximate the solution with less support vectors. Although the method is very robust to changes in the settings, we admit that working-set techniques are often faster in standard cases.

Currently, the algorithm changes the active set by only one variable per step, and most of the computation time is spent to calculate the prediction errors E_i. Both problems can be significantly improved by introducing *gradient projection* steps. If this technique is combined with iterative solvers, also a large number of free support vectors is possible. This is a promising direction of future work in SVM optimization methods.

References

1. Chang CC, Lin CJ (2003) LIBSVM: A library for support vector machines. Technical report. National Taiwan University, Taipei, Taiwan
2. Gill PE et al. (1974) Methods for Modifying Matrix Computations. Mathematics of Computation 28(126):505–535
3. Golub GH, van Loan CF (1996) Matrix Computations. 3rd ed. The Johns Hopkins University Press, Baltimore, MD
4. Kecman V, Vogt M, Huang TM (2003) On the equality of Kernel AdaTron and Sequential Minimal Optimization in classification and regression tasks and alike algorithms for kernel machines. In: Proceedings of the 11th European Symposium on Artificial Neural Networks (ESANN 2003), pp. 215–222, Bruges, Belgium
5. Mangasarian OL, Musicant DR (2001) Active set support vector machine classification. In: Leen TK, Tresp V, Dietterich TG (eds) Advances in Neural Information Processing Systems (NIPS 2000) Vol. 13, pp. 577–583. MIT Press, Cambridge, MA
6. Blake CL, Merz CJ (1998) UCI repository of machine learning databases. University of California, Irvine, http://www.ics.uci.edu/∼mlearn/
7. Nocedal J, Wright SJ (1999) Numerical Optimization. Springer-Verlag, New York

8. Platt JC (1999) Fast training of support vector machines using sequential minimal optimization. In: Schölkopf B, Burges CJC, Smola AJ (eds) Advances in Kernel Methods – Support Vector Learning. MIT Press, Cambridge, MA
9. Poggio T et al. (2002) b. In: Winkler J, Niranjan M (eds) Uncertainty in Geometric Computations, pp. 131–141. Kluwer Academic Publishers, Boston
10. Schölkopf B, Smola AJ (2002) Lerning with Kernels. The MIT Press, Cambridge, MA
11. Vapnik VN (1995) The Nature of Statistical Learning Theory. Springer-Verlag, New York
12. Vogt M, Spreitzer K, Kecman V (2003) Identification of a high efficiency boiler by Support Vector Machines without bias term. In: Preprints of the 13th IFAC Symposium on System Identification (SYSID 2003), pp. 485–490. Rotterdam, The Netherlands

A The Karush-Kuhn-Tucker conditions

A general constrained optimization problem is given by

$$F(\mathbf{x}) = \min_{\mathbf{x}} \tag{45a}$$
$$\text{s.t.} \quad \mathbf{G}(\mathbf{x}) = \mathbf{0} \tag{45b}$$
$$\mathbf{H}(\mathbf{x}) \geq \mathbf{0} \tag{45c}$$

The Lagrangian of this problem is defined as

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = F(\mathbf{x}) - \sum_i \lambda_i G_i(\mathbf{x}) - \sum_i \mu_i H_i(\mathbf{x}) \tag{46}$$

In the constrained optimum $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ the following first-order necessary conditions are satisfied [7]:

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = 0 \tag{47a}$$
$$G_i(\mathbf{x}^*) = 0 \tag{47b}$$
$$H_i(\mathbf{x}^*) \geq 0 \tag{47c}$$
$$\mu_i^* \geq 0 \tag{47d}$$
$$\lambda_i^* G_i(\mathbf{x}^*) = 0 \tag{47e}$$
$$\mu_i^* H_i(\mathbf{x}^*) = 0 \tag{47f}$$

These are commonly referred to as *Karush-Kuhn-Tucker* conditions.