Active-Set Methods for Support Vector Machines

M. Vogt[1] and V. 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

Abstract. This chapter describes an active-set algorithm for quadratic programming problems that arise from the computation of support vector machines (SVMs). Currently, most SVM optimizers implement working-set (decomposition) techniques because of their ability to handle large data sets. Although these show good results in general, active-set methods are a reasonable alternative – in particular if the data set is not too large, if the problem is ill-conditioned, or if high precision is needed. Algorithms are derived for classification and regression with both fixed and variable bias term. The material is completed by acceleration and approximation techniques as well as a comparison with other optimization methods in application examples.

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

1 Introduction

Support vector machines (SVMs) have become popular for classification and regression tasks [10, 11] since they can treat large input dimensions and show 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

$$\min_{\mathbf{a}} \quad J(\mathbf{a}) = \mathbf{a}^{\mathrm{T}}\mathbf{Q}\mathbf{a} + \mathbf{q}^{\mathrm{T}}\mathbf{a} \tag{1a}$$

$$\text{s.t.} \quad \mathbf{F}\mathbf{a} \geq \mathbf{f} \tag{1b}$$

$$\mathbf{G}\mathbf{a} = \mathbf{g} \tag{1c}$$

the sizes of which are dependent on the number N of training data. The settings for different SVM types will be derived in (10), (18), (29) and (37).

M. Vogt and V. Kecman: *Active-Set Methods for Support Vector Machines*, StudFuzz **177**, 133–158 (2005)
www.springerlink.com

1.1 Optimization Methods

The dependency on the size N of the training data set is the most critical issue of SVM optimization as N may be very large and the memory consumption is roughly $\mathcal{O}(N^2)$ if the whole QP problem (1) 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.

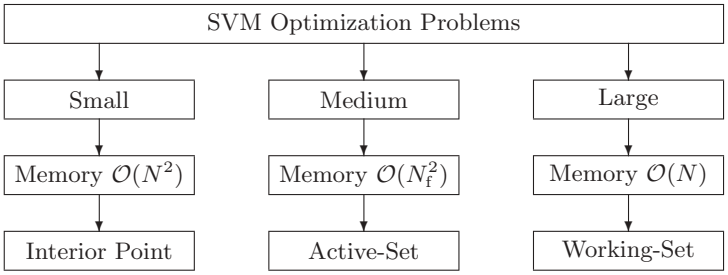


Fig. 1. QP optimization methods for different training data set sizes

If the problem is small enough to be stored completely in memory (on current PC hardware up to approximately 5000 data), *interior point* methods are suitable. They are known to be the most precise QP solvers [7, 10] but have a memory consumption of $\mathcal{O}(N^2)$. 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 $\mathcal{O}(N_f^2 + N)$ memory where N_f is the number of *free* (unbounded) variables. Although N_f is typically much smaller than the number of the data, it dominates the memory consumption for large data sets due to its quadratic dependency.

Common SVM software packages rely on working-set methods because N is often large in practical applications. However, in some situations this is not the optimal approach, e.g., if the problem is ill-conditioned, if the SVM parameters (C and ε) are not chosen carefully, or if high precision is needed. This seems to apply in particular to regression, see Sect. 5. *Active-set* algorithms are the classical solvers for QP problems. They are known to be robust, but they are sometimes slower and (as stated above) require more memory than working-set algorithms. Their robustness is in particular useful for cross-validation techniques where the SVM parameters are varied over a wide range. 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. An implementation for standard SVM classification can be found in [12], for regression problems in [13]. 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 $\mathcal{F}^0 = \{1, \ldots, N\} \backslash \mathcal{A}^0$ are *free* (lying in the interior of the feasible region). Then the following steps are performed repeatedly for $k = 1, 2, \ldots$:

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 $\mathcal{A}^k$.

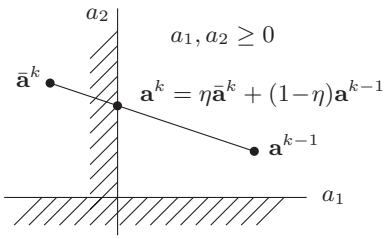


Fig. 2. Affine scaling of the non-feasible solution

This basic algorithm is used for all cases described in the next sections, only the structure of the KKT system in step A1 and the conditions in step A2 are different. Sections 2 and 3 describe how to use the algorithm for SVM classification and regression tasks. In this context the derivation of the dual problems is repeated in order to introduce the distinction between fixed and variable bias term. Section 4 considers the efficient solution of the KKT system, several acceleration techniques and the approximation of the solution with a limited

number of support vectors. Application examples for both classification and regression are given in Sect. 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.

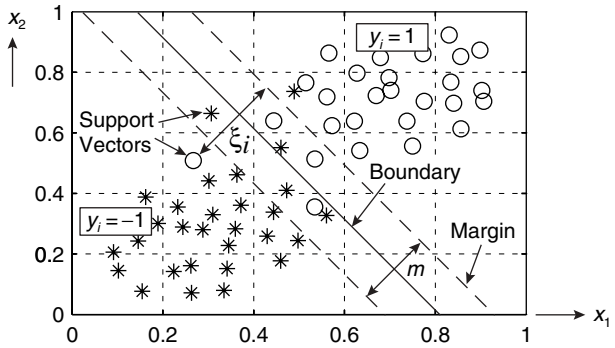


Fig. 3. Separating two overlapping classes with a linear decision function

For separable classes, a 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 in its canonical form [10, 11], the margin can be expressed as $m = 2/\mathbf{w}^\mathrm{T}\mathbf{w}$. In the case of 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 are also penalized by a factor C in the loss function to keep the deviations small [11]. These ideas lead to the *soft margin* classifier:

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

$$\text{s.t.} \quad y_i(\mathbf{w}^\mathrm{T}\mathbf{x}_i + b) \geq 1 - \xi_i \tag{2b}$$

$$\xi_i \geq 0, \quad i = 1, \ldots, N \tag{2c}$$

The parameter C describes the trade-off between maximal margin and correct classification. The *primal* problem (2) 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

$$L_{\mathrm{p}}(\mathbf{w},\boldsymbol{\xi},b,\boldsymbol{\alpha},\boldsymbol{\beta}) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}+C\sum_{i=1}^{N}\xi_i-\sum_{i=1}^{N}\alpha_i[y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i+b)-1+\xi_i]-\sum_{i=1}^{N}\beta_i\xi_i \quad (3)$$

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 (48a) the minimization is performed with respect to the primal variables in order to find the optimum:

$$\frac{\partial L_{\mathrm{p}}}{\partial \mathbf{w}} = \mathbf{0} \;\Rightarrow\; \mathbf{w} = \sum_{i=1}^{N} y_i\alpha_i\mathbf{x}_i \quad (4a)$$

$$\frac{\partial L_{\mathrm{p}}}{\partial \xi_i} = 0 \;\Rightarrow\; \alpha_i + \beta_i = C, \quad i = 1,\ldots,N \quad (4b)$$

Although b is also a primal variable, we defer the minimization with respect to b for a moment. Instead, (4) is used to eliminate $\mathbf{w}$, $\boldsymbol{\xi}$ and $\boldsymbol{\beta}$ from the Lagrangian which leads to

$$L_{\mathrm{p}}^{*}(\boldsymbol{\alpha},b) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_iy_j\alpha_i\alpha_j\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + \sum_{i=1}^{N}\alpha_i - b\sum_{i=1}^{N} y_i\alpha_i\,. \quad (5)$$

To solve *nonlinear* classification problems, the linear SVM is applied to *features* $\boldsymbol{\Phi}(\mathbf{x})$ (instead of the inputs $\mathbf{x}$), where $\boldsymbol{\Phi}$ is a given feature map (see Fig. 4). Since $\mathbf{x}$ occurs in (5) only in scalar products $\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$, we define the *kernel function*

$$K(\mathbf{x},\mathbf{x}') = \boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{x})\boldsymbol{\Phi}(\mathbf{x}')\,, \quad (6)$$

and finally (5) becomes

$$L_{\mathrm{p}}^{*}(\boldsymbol{\alpha},b) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_iy_j\alpha_i\alpha_jK_{ij} + \sum_{i=1}^{N}\alpha_i - b\sum_{i=1}^{N} y_i\alpha_i \quad (7)$$

with the abbreviation $K_{ij} = K(\mathbf{x}_i,\mathbf{x}_j)$. In the following, kernels are always assumed to be symmetric and positive definite. This class of functions includes most of the common kernels [10], e.g.

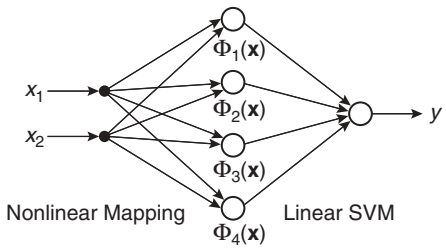


Fig. 4. Structure of a nonlinear SVM

Linear kernel (scalar product):	$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{x}'$
Inhomogeneous polynomial kernel:	$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^{\mathrm{T}}\mathbf{x}' + c)^p$
Gaussian (RBF) kernel:	$K(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2/\sigma^2)$
Sigmoidal (MLP) kernel:	$K(\mathbf{x}, \mathbf{x}') = \tanh(\mathbf{x}^{\mathrm{T}}\mathbf{x}' + d)$

The conditions (48d) and (4b) yield additional restrictions for the dual variables

$$0 \leq \alpha_i \leq C\,, \quad i = 1, \ldots, N \tag{8}$$

and from (4a) and (6) follows that

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\Phi}(\mathbf{x}) + b = \sum_{\alpha_i \neq 0} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\,. \tag{9}$$

This shows 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 10%. In Fig. 3, these are the data points lying on the margin ($\xi_i = 0$ and $0 < \alpha_i < C$) or on the "wrong" side of the margin ($\xi_i > 0$ and $\alpha_i = C$).

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 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, e.g., in the case of *positive definite* kernel functions [4, 9, 14]. 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 below in (18)):

$$\min_{\boldsymbol{\alpha}} \quad J_{\mathrm{d}}(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K_{ij} - \sum_{i=1}^{N}\alpha_i + b\sum_{i=1}^{N} y_i \alpha_i \tag{10a}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C\,, \quad i = 1, \ldots, N \tag{10b}$$

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 (10a) vanishes. The reason for the change of the sign in the objective function is that optimization algorithms usually assume *minimization* rather than *maximization* problems.

If b is kept fixed, the SVM is computed by solving the box-constrained convex QP problem (10), which is one of the most convenient QP cases. To

solve it with the active-set method described in Sect. 1, the KKT conditions of this problem must be found. Its Lagrangian is

$$\begin{aligned} L_{\mathrm{d}}(\boldsymbol{\alpha}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K_{ij} - \sum_{i=1}^{N}\alpha_i + b\sum_{i=1}^{N} y_i \alpha_i \\ & - \sum_{i=1}^{N}\lambda_i \alpha_i - \sum_{i=1}^{N}\mu_i (C - \alpha_i) \end{aligned} \tag{11}$$

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

$$\frac{\partial L_{\mathrm{d}}}{\partial \alpha_i} = y_i E_i - \lambda_i + \mu_i = 0 \tag{12a}$$

$$0 \leq \alpha_i \leq C \tag{12b}$$

$$\lambda_i \geq 0, \quad \mu_i \geq 0 \tag{12c}$$

$$\alpha_i \lambda_i = 0, \quad (C - \alpha_i)\mu_i = 0 \tag{12d}$$

According to α_i, three cases have to be considered:

$$\begin{aligned} \boxed{0 < \alpha_i < C} \quad (i \in \mathcal{F}) \quad & \Rightarrow \quad \lambda_i = \mu_i = 0 \\ & \Rightarrow \quad \sum_{j \in \mathcal{F}} y_j \alpha_j K_{ij} = y_i - \sum_{j \in \mathcal{A}_C} y_j \alpha_j K_{ij} - b \end{aligned} \tag{13a}$$

$$\begin{aligned} \boxed{\alpha_i = 0} \quad (i \in \mathcal{A}_0) \quad & \Rightarrow \quad \lambda_i = y_i E_i > 0 \\ & \Rightarrow \quad \mu_i = 0 \end{aligned} \tag{13b}$$

$$\begin{aligned} \boxed{\alpha_i = C} \quad (i \in \mathcal{A}_C) \quad & \Rightarrow \quad \lambda_i = 0 \\ & \Rightarrow \quad \mu_i = -y_i E_i > 0 \end{aligned} \tag{13c}$$

$\mathcal{A}_0$ denotes the set of lower bounded variables $\alpha_i = 0$, whereas $\mathcal{A}_{\mathrm{C}}$ comprises the upper bounded ones with $\alpha_i = C$. The above conditions are exploited in each iteration step k. Case (13a) establishes the linear system in step A1 for the currently free variables $i \in \mathcal{F}^k$. Cases (13b) and (13c) 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 $\alpha_i^k = 0$ and $\alpha_i^k = C$ cannot be met simultaneously. The variables $\alpha_i = C$ in $\mathcal{A}_C^k$ also occur on the right hand side of the linear system (13a).

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 Sect. 3, and because it slightly accelerates the computation. With this modification, in step A1 the linear system

$$\mathbf{H}^k \bar{\mathbf{a}}^k = \mathbf{c}^k \tag{14}$$

with

$$\left.\begin{aligned} \bar{a}_i^k &= y_i \bar{\alpha}_i^k \\ h_{ij}^k &= K_{ij} \\ c_i^k &= y_i - \sum_{j \in \mathcal{A}_C^k} a_j^k K_{ij} - b \end{aligned}\right\} \quad \text{for } i, j \in \mathcal{F}^k \tag{15}$$

has to be solved. $\mathbf{H}^k$ is called *reduced* or *projected* Hessian. In the case of box constraints, it results from the complete Hessian $\mathbf{Q}$ in (1) by dropping all rows and columns belonging to constraints that are currently regarded as active. 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, (14) can be solved by the methods described in Sect. 4. Step A2 computes

$$\lambda_i^k = +y_i E_i^k \quad \text{for } i \in \mathcal{A}_0^k \tag{16a}$$

$$\mu_i^k = -y_i E_i^k \quad \text{for } i \in \mathcal{A}_C^k \tag{16b}$$

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 $\mathcal{F}^k$. In practice, the KKT conditions are checked with precision τ, so that a variable α_i is accepted as optimal if $\lambda_i^k > -\tau$ and $\mu_i^k > -\tau$.

2.2 Classification with Variable Bias Term

Most SVM algorithms do not keep the bias term fixed but compute it during optimization. In that case b is a primal variable, and the Lagrangian (3) can be minimized with respect to it:

$$\frac{\partial L_\mathrm{p}}{\partial b} = 0 \;\Rightarrow\; \sum_{i=1}^{N} y_i \alpha_i = 0 \tag{17}$$

On the one hand (17) removes the last term from (5), on the other hand it is an additional constraint that must be considered in the optimization problem:

$$\min_{\boldsymbol{\alpha}} \quad J_\mathrm{d}(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K_{ij} - \sum_{i=1}^{N} \alpha_i \tag{18a}$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C\,, \quad i = 1, \ldots, N \tag{18b}$$

$$\sum_{i=1}^{N} y_i \alpha_i = 0 \tag{18c}$$

This modification changes the Lagrangian (11) to

$$\begin{aligned} L_{\mathrm{d}}(\boldsymbol{\alpha}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \nu) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K_{ij} - \sum_{i=1}^{N} \alpha_i \\ - \sum_{i=1}^{N} \lambda_i \alpha_i - \sum_{i=1}^{N} \mu_i (C - \alpha_i) - \nu \sum_{i=1}^{N} y_i \alpha_i \end{aligned} \tag{19}$$

and its derivatives to

$$\frac{\partial L_{\mathrm{d}}}{\partial \alpha_i} = y_i \sum_{j=1}^{N} y_j \alpha_j K_{ij} - 1 - \lambda_i + \mu_i - \nu y_i = 0\,, \quad i = 1, \ldots, N \tag{20}$$

where ν is the Lagrange multiplier of the equality constraint (18c). It can be easily seen that $\nu = -b$, i.e., L_{d} is the same as (11) with the important difference that b is not fixed any more. With the additional equality constraint (18c) and again with $a_i = y_i \alpha_i$ the linear system becomes

$$\begin{pmatrix} \mathbf{H}^k & \mathbf{e} \\ \mathbf{e}^{\mathrm{T}} & 0 \end{pmatrix} \begin{pmatrix} \bar{\mathbf{a}}^k \\ b^k \end{pmatrix} = \begin{pmatrix} \mathbf{c}^k \\ d^k \end{pmatrix} \begin{matrix} \} \ p \text{ rows} \\ \} \ 1 \text{ row} \end{matrix} \tag{21}$$

with

$$d^k = -\sum_{j \in \mathcal{A}_C^k} a_j^k \quad \text{and} \quad \mathbf{e} = (1, \ldots, 1)^{\mathrm{T}}\,. \tag{22}$$

One possibility to solve this indefinite system is to use factorization methods for indefinite matrices like 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 Sect. 4), and the system (21) can be solved by exploiting its block structure. For that, a Gauss transformation 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

$$(\mathbf{u}^k)^{\mathrm{T}} \mathbf{e} b^k = (\mathbf{u}^k)^{\mathrm{T}} \mathbf{c}^k - d^k\,. \tag{23}$$

Since this is a scalar equation, it is simply divided by $(\mathbf{u}^k)^{\mathrm{T}}\mathbf{e}$ in order to find b^k. This technique is 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 $\quad (\mathbf{R}^k)^{\mathrm{T}} \mathbf{R}^k \mathbf{u}^k = \mathbf{e} \quad$ *for* $\quad \mathbf{u}^k\,.$

Compute $\quad b^k = -\left(\sum_{j \in \mathcal{A}_C^k} a_j^k + \sum_{j \in \mathcal{A}_C^k} u_j^k c_j^k \right) \Big/ \sum_{j \in \mathcal{A}_C^k} u_j^k\,.$

Solve $\quad (\mathbf{R}^k)^{\mathrm{T}} \mathbf{R}^k \bar{\mathbf{a}}^k = \mathbf{c}^k - \mathbf{e} b^k \quad$ *for* $\quad \bar{\mathbf{a}}^k\,.$

The computation of λ_i^k and μ_i^k remains the same as in (16) 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 $i = 1, \ldots, N$. Then the gradients of the active inequality constraints and the equality constraint are linear dependent. 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 regression 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. 5 (for a nonlinear example) and [10]. This concept results in the following primal problem:

$$\min_{\mathbf{w},\boldsymbol{\xi},\boldsymbol{\xi}^*} \quad J_{\mathrm{p}}(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^*) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \tag{24a}$$

$$\text{s.t.} \quad y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i - b \leq \varepsilon + \xi_i \tag{24b}$$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i^* \tag{24c}$$

$$\xi_i, \xi_i^* \geq 0\,, \quad i = 1, \ldots, N \tag{24d}$$

To apply the same technique as for classification, the Lagrangian

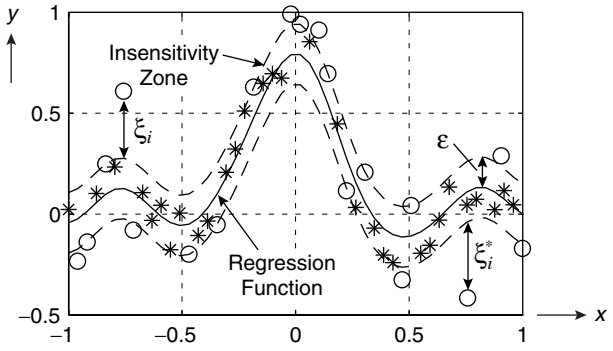


Fig. 5. Nonlinear support vector machine regression

$$L_{\mathrm{p}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}, \boldsymbol{\beta}^*) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) - \sum_{i=1}^{N}(\beta_i\xi_i + \beta_i^*\xi_i^*) - \sum_{i=1}^{N}\alpha_i(\varepsilon + \xi_i - y_i + \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) - \sum_{i=1}^{N}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i - b) \tag{25}$$

of the primal problem (24) 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 Sect. 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:

$$L_{\mathrm{p}}^*(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*, b) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K_{ij} - \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)y_i + \varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + b\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) \tag{26}$$

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 (6) and Fig. 5. 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

$$0 \le \alpha_i^{(*)} \le C\,, \quad i = 1, \ldots, N \tag{27}$$

resulting from (48d) must be fulfilled. Since a data point cannot lie above *and* below the insensitivity zone simultaneously, the dual variables $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ are not independent. At least one of the primal constraints (24b) and (24c) must be met with equality for each i. The KKT conditions then imply that $\alpha_i\alpha_i^* = 0$. The output of regression SVMs is computed as

$$f(\mathbf{x}) = \sum_{\alpha_i^{(*)} \neq 0} (\alpha_i - \alpha_i^*)K(\mathbf{x}_i, \mathbf{x}) + b\,. \tag{28}$$

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 (10) is similar for regression SVMs. It is built from (26) and (27) by treating the bias term as a fixed parameter:

$$\min_{\boldsymbol{\alpha},\,\boldsymbol{\alpha}^*} \quad J_\mathrm{d}(\boldsymbol{\alpha},\boldsymbol{\alpha}^*) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K_{ij} - \sum_{i=1}^{N}(\alpha_i-\alpha_i^*)y_i + \varepsilon\sum_{i=1}^{N}(\alpha_i+\alpha_i^*) + b\sum_{i=1}^{N}(\alpha_i-\alpha_i^*) \tag{29a}$$

$$\text{s.t.} \quad 0 \le \alpha_i^{(*)} \le C\,, \quad i=1,\ldots,N \tag{29b}$$

Again, (29) is formulated as a *minimization* problem by setting $J_\mathrm{d}(\boldsymbol{\alpha},\boldsymbol{\alpha}^*) = -L_\mathrm{p}^*(\boldsymbol{\alpha},\boldsymbol{\alpha}^*,b)$ with fixed b. For $b=0$ the last term vanishes so that (29) differs from the standard problem (37) only in the absence of the equality constraint (37c). To find the steps A1 and A2 of an active-set algorithm that solves (29) its Lagrangian

$$\begin{aligned} L_\mathrm{d}(\boldsymbol{\alpha},\boldsymbol{\alpha}^*,\boldsymbol{\lambda},\boldsymbol{\lambda}^*,\boldsymbol{\mu},\boldsymbol{\mu}^*) &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K_{ij} \\ &\quad - \sum_{i=1}^{N}(\alpha_i-\alpha_i^*)y_i + \varepsilon\sum_{i=1}^{N}(\alpha_i+\alpha_i^*) + b\sum_{i=1}^{N}(\alpha_i-\alpha_i^*) \\ &\quad - \sum_{i=1}^{N}\lambda_i\alpha_i - \sum_{i=1}^{N}\mu_i(C-\alpha_i) - \sum_{i=1}^{N}\lambda_i^*\alpha_i^* - \sum_{i=1}^{N}\mu_i^*(C-\alpha_i^*) \end{aligned} \tag{30}$$

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

$$\frac{\partial L_\mathrm{d}}{\partial \alpha_i} = \varepsilon + E_i - \lambda_i + \mu_i = 0 \tag{31a}$$

$$\frac{\partial L_\mathrm{d}}{\partial \alpha_i^*} = \varepsilon - E_i - \lambda_i^* + \mu_i^* = 0 \tag{31b}$$

$$0 \le \alpha_i^{(*)} \le C \tag{31c}$$

$$\lambda_i^{(*)} \ge 0, \quad \mu_i^{(*)} \ge 0 \tag{31d}$$

$$\alpha_i^{(*)}\lambda_i^{(*)} = 0, \quad (C-\alpha_i^{(*)})\mu_i^{(*)} = 0\,. \tag{31e}$$

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

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

$$\Rightarrow\ \lambda_i=\mu_i=\mu_i^*=0,\ \lambda_i^*=2\varepsilon>0$$

$$\Rightarrow\ \sum_{j\in\mathcal{F}^{(*)}} a_jK_{ij} = y_i - \varepsilon - \sum_{j\in\mathcal{A}_C^{(*)}} a_jK_{ij} \tag{32a}$$

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

$$\begin{aligned} &\Rightarrow\ \lambda_i^* = \mu_i = \mu_i^* = 0,\ \lambda_i = 2\varepsilon > 0 \\ &\Rightarrow\ \sum_{j \in \mathcal{F}^{(*)}} a_j K_{ij} = y_i + \varepsilon - \sum_{j \in \mathcal{A}_C^{(*)}} a_j K_{ij} \end{aligned} \tag{32b}$$

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

$$\begin{aligned} &\Rightarrow\ \lambda_i = \varepsilon + E_i > 0, \quad \lambda_i^* = \varepsilon - E_i > 0 \\ &\Rightarrow\ \mu_i = 0, \quad \mu_i^* = 0 \end{aligned} \tag{32c}$$

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

$$\begin{aligned} &\Rightarrow\ \lambda_i = 0, \quad \lambda_i^* = \varepsilon - E_i > 0 \\ &\Rightarrow\ \mu_i = -\varepsilon - E_i > 0, \quad \mu_i^* = 0 \end{aligned} \tag{32d}$$

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

$$\begin{aligned} &\Rightarrow\ \lambda_i = \varepsilon + E_i > 0, \quad \lambda_i^* = 0 \\ &\Rightarrow\ \mu_i = 0, \quad \mu_i^* = -\varepsilon + E_i > 0 \end{aligned} \tag{32e}$$

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 ((32a) and (32b)) or equal to C ((32d) and (32e)), the other one must be zero. The structure of the sets $\mathcal{A}_0^*$ and $\mathcal{A}_C^*$ is identical to that of $\mathcal{A}_0$ and $\mathcal{A}_C$, but it considers the variables α_i^* instead of α_i. It follows from the reasoning above that $\mathcal{A}_C \subseteq \mathcal{A}_0^*$, $\mathcal{A}_C^* \subseteq \mathcal{A}_0$ and $\mathcal{A}_C \cap \mathcal{A}_C^* = \emptyset$. Similar to classification, the cases (32a) and (32b) form the linear system for step A1 and the cases (32c) – (32e) 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 (14):

$$\mathbf{H}^k \bar{\mathbf{a}}^k = \mathbf{c}^k \tag{33}$$

with

$$\begin{aligned} &\left.\begin{aligned} \bar{a}_i^k &= \bar{\alpha}_i^k - \bar{\alpha}_i^{*k} \\ h_{ij}^k &= K_{ij} \end{aligned}\right\} \quad \text{for } i \in \mathcal{F}^k \cup \mathcal{F}^{*k} \\ &c_i^k = y_i - \sum_{j \in \mathcal{A}_C^k \cup \mathcal{A}_C^{*k}} a_j^k K_{ij} + \begin{cases} -\varepsilon & \text{for } i \in \mathcal{F}^k \\ +\varepsilon & \text{for } i \in \mathcal{F}^{*k} \end{cases} \end{aligned} \tag{34}$$

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

$$\left.\begin{aligned} \lambda_i^k &= \varepsilon + E_i^k \\ \lambda_i^{*k} &= \varepsilon - E_i^k \end{aligned}\right\} \quad \text{for} \;\; i \in \mathcal{A}_0^k \cup \mathcal{A}_0^{*k} \tag{35a}$$

and

$$\left.\begin{aligned} \mu_i^k &= -\varepsilon - E_i^k \\ \mu_i^{*k} &= -\varepsilon + E_i^k \end{aligned}\right\} \quad \text{for} \;\; i \in \mathcal{A}_C^k \cup \mathcal{A}_C^{*k} \; . \tag{35b}$$

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

3.2 Regression with Variable Bias Term

If the bias term is treated as a variable, (26) can be minimized with respect to b (i.e., $\partial L_\mathrm{d}^*/\partial b = 0$) resulting in

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

Like in classification, this condition removes the last term from (29a) but must be treated as additional equality constraint:

$$\begin{aligned} \min_{\boldsymbol{\alpha},\, \boldsymbol{\alpha}^*} \quad & J_\mathrm{d}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K_{ij} \\ & \qquad - \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) y_i + \varepsilon \sum_{i=1}^{N} (\alpha_i + \alpha_i^*) \end{aligned} \tag{37a}$$

$$\text{s.t.} \quad 0 \le \alpha_i^{(*)} \le C \, , \quad i = 1, \ldots, N \tag{37b}$$

$$\sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0 \tag{37c}$$

The Lagrangian of this QP problem is nearly identical to (30):

$$\begin{aligned} L_\mathrm{d}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\lambda}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}, \boldsymbol{\mu}^*, \nu) &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K_{ij} \\ &- \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) y_i + \varepsilon \sum_{i=1}^{N} (\alpha_i + \alpha_i^*) - \nu \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \\ &- \sum_{i=1}^{N} \lambda_i \alpha_i - \sum_{i=1}^{N} \mu_i (C - \alpha_i) - \sum_{i=1}^{N} \lambda_i^* \alpha_i^* - \sum_{i=1}^{N} \mu_i^* (C - \alpha_i^*) \end{aligned} \tag{38}$$

Classification has already shown 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, (31) also comprises the equality constraint (37c), but the five cases (32) 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 (21):

$$\begin{pmatrix} \mathbf{H}^k & \mathbf{e} \\ \mathbf{e}^{\mathrm{T}} & 0 \end{pmatrix} \begin{pmatrix} \bar{\mathbf{a}}^k \\ b^k \end{pmatrix} = \begin{pmatrix} \mathbf{c}^k \\ d^k \end{pmatrix} \begin{matrix} \} \ p \text{ rows} \\ \} \ 1 \text{ row} \end{matrix} \tag{39}$$

with

$$d^k = -\sum_{j \in \mathcal{A}_C^k \cup \mathcal{A}_C^{*k}} a_j^k \quad \text{and} \quad \mathbf{e} = (1, \ldots, 1)^{\mathrm{T}} \tag{40}$$

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 Sect. 2. The KKT conditions in step A2 remain exactly the same as (35).

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,
- $N_{\mathrm{f}}(N_{\mathrm{f}} + 3)/2$ floating point elements for the triangular matrix and the right hand side of the linear system,

where N_{f} is the number of free variables in the optimum, 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 Sect. 4.1. Since most of the coefficients a_i will be zero in the optimum, the initial feasible solution is chosen as $a_i = 0$ for $i = 1, \ldots, N$. If shrinking and/or caching is activated, additional memory must be provided, see Sects. 4.4 and 4.5 for details.

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 then found by simple backsubstitution. For variable bias term, the block-algorithm described in Sect. 2.2 is used.

4.1 Cholesky Decomposition with Pivoting

Although the Cholesky decomposition is numerically 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 Gaussian kernels 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.

Usually the Cholesky decomposition is computed element-wise using `axpy` operations defined in the BLAS [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 only updates the diagonal elements and makes use of `axpy` operations otherwise:

Compute for $i = 1, \ldots, p$*:*
Find $k = \arg\max\{|\bar{h}_{ii}|, \ldots, |\bar{h}_{pp}|\}$.
Swap rows and columns i *and* k *symmetrically.*
Compute $r_{ii} = \sqrt{\bar{h}_{ii}}$.
Compute for $j = i + 1, \ldots, p$*:*

$$r_{ij} = \left(h_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right) \Big/ r_{ii}$$
$$\bar{h}_{jj} \leftarrow \bar{h}_{jj} - r_{ij}^2$$

where p is the size of the system, $\bar{h}_{jj}$ are the updated diagonal elements and "$\leftarrow$" indicates the update process. The i-th step

$$\begin{array}{c} \begin{array}{cc} i & j \\ \downarrow & \longrightarrow \end{array} \\ \left(\begin{array}{cccccc} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ & \bullet & \bullet & \bullet & \bullet & \bullet \\ & & \circ & \circ & \circ & \circ \\ & & & \times & * & * \\ & & j \searrow & & \times & * \\ & & & & & \times \end{array} \right) \leftarrow i \end{array}$$

of the algorithm computes the i-th row ($\circ$) of the matrix from the already finished elements ($\bullet$). The diagonal elements ($\times$) are updated whereas the rest ($*$) remains untouched. The result can be written as

$$\mathbf{P}\mathbf{H}\mathbf{P}^{\mathrm{T}} = \mathbf{R}^{\mathrm{T}}\mathbf{R} \tag{41}$$

with the permutation matrix $\mathbf{P}$. Of course the implementation uses the pivot vector described above instead of the complete matrix. Besides that, only the upper triangular part of $\mathbf{R}$ is stored, so that 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 changes the active set by 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 Sect. 4.1, because they cannot be used with pivoting. The only way to cope 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 $\mathbf{H}$. As any element r_{ij} of the Cholesky decomposition is calculated solely from the diagonal element r_{ii} and the sub-columns i and j above the i-th row (see Sect. 4.1), only the last column needs to be computed:

Compute for i = 1,. . . ,p:

$$r_{ip} = \left(h_{ip} - \sum_{k=1}^{i-1} r_{ki} r_{kp} \right) \Big/ r_{ii}$$

The columns $1, \ldots, p-1$ 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

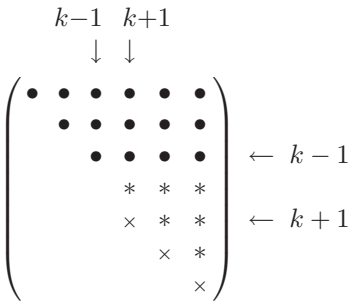
Removing variables from an existing Cholesky decomposition is a more sophisticated task [2, 3]. For that, we introduce an unknown matrix $\mathbf{A} \in \mathbb{R}^{M \times p}$ with

$$\mathbf{H} = \mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{A}^{\mathrm{T}}\mathbf{A} \quad \text{and} \quad \mathbf{QA} = \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}, \tag{42}$$

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 $\mathbf{A}$:

$$\mathbf{Q}(\mathbf{a}_1 \ldots \mathbf{a}_{k-1}, \mathbf{a}_{k+1} \ldots \mathbf{a}_p) = \begin{pmatrix} \mathbf{r}_1 \ldots \mathbf{r}_{k-1}, \mathbf{r}_{k+1} \ldots \mathbf{r}_p \\ \mathbf{0} \end{pmatrix} \tag{43}$$

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 $k+1, \ldots, p$ has one element below the diagonal:



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

$$\underbrace{\mathbf{\Omega}_p \cdots \mathbf{\Omega}_{k+1} \mathbf{Q}}_{\tilde{\mathbf{Q}}} \tilde{\mathbf{A}} = \begin{pmatrix} \tilde{\mathbf{R}} \\ \mathbf{0} \end{pmatrix} \tag{44}$$

$\tilde{\mathbf{R}}$ is the Cholesky factor of the reduced matrix $\tilde{\mathbf{H}}$, see [2, 3] for details.

However, it should be mentioned that modification techniques often do not lead to a strong acceleration. As long as N_f remains small, most of the computation time is spent to check the KKT conditions in $\mathcal{A}$ (during step A2 of the algorithm). For that, 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. Since only few reduction steps are performed repeatedly, round-off errors will not be propagated too much.

4.4 Shrinking the Problem Size

As pointed out above, checking the KKT conditions is the dominating factor of the computation time because the function values (9) or (28) need to be computed for all variables of the active set in each step. For that, the active-set algorithm uses two heuristics to accelerate the KKT check: shrinking the set of variables to be checked, and caching kernel function values (which will be described in the next section).

By default, step A2 of the algorithm checks *all* bounded variables. However, it can be observed that a variable fulfilling the KKT conditions for a number of iterations is likely to stay in the active set [1, 10]. The shrinking heuristic uses this observation to reduce number of KKT checks. It counts the number of consecutive successful KKT checks for each variable. If this number exceeds a given number s, then the variable is not checked again. Only if there are no variables left to be checked, a check of the complete active set is performed and the shrinking procedure starts again.

In experiments, small values of s (e.g., $s = 1, \ldots, 5$) have caused an acceleration up to a factor of 5. This shrinking heuristic requires an additional vector of N integer elements to count the KKT checks of each variable. If the

correct active set is identified, shrinking does not change the solution. However, for low precisions τ it may happen that the algorithm chooses a different approximation of the active set, i.e., different support vectors.

4.5 Caching Kernel Values

Whereas the shrinking heuristic tries to *reduce* the number of function evaluations, the goal of a kernel cache is to *accelerate* the remaining ones. For that, as much kernel function values K_{ij} as possible are stored in a given chunk of memory to avoid re-calculation. Some algorithms also use a cache for the function values f_i (or prediction error values $E_i = f_i - y_i$, respectively), e.g. [8]. However, since the active-set algorithm changes the values of *all* free variables in each step, this type of cache would only be useful when the number of free variables remains small.

The kernel cache has a given maximum size and is organized as *row cache* [1, 10]. It stores a complete row of the kernel matrix for each support vector – as long as space is available. The row entries corresponding to the active set are exploited to compute (9) or (28) for the KKT check, whereas the remaining elements are used to rebuild the system matrix $\mathbf{H}$ when necessary. The following caching strategy has been implemented:

- If a variable becomes $a_i = 0$, then the according row is marked as free in the cache but not deleted.
- If a variable becomes $a_i \neq 0$, the algorithm first checks if the according row is already in the cache (possibly marked as free). Otherwise, the row is completely calculated and stored as long as space is available.
- When a row should be added, the algorithm first checks if the maximum number of rows is reached. Only if the cache is full, it starts to overwrite those rows that have been marked as free.

The kernel cache allows a trade-off between computation time and memory consumption. It requires $N \times m$ floating point elements for the kernel values (where m is the maximum number of rows that can be cached), and N integer elements for indexing purposes. It is most effective for kernel functions having high computational demands, e.g., Gaussians in high-dimensional input spaces. In these cases it usually speeds up the algorithm by a factor 5 or even more, see Sect. 5.2

4.6 Approximating the Solution

Active-set methods check the KKT conditions of the *complete* active set (apart from the shrinking heuristics) 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, avoid this complete check and follow the opposite strategy: They perform

only small steps and therefore need to reduce the number of KKT evaluations to a minimum by additional 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 are associated with

- N_{f} free variables $0 < \alpha_i^{(*)} < C$ (i.e., those with $i \in \mathcal{F}^{(*)}$).
- $N_{\mathrm{SV}} - N_{\mathrm{f}}$ upper bounded variables $\alpha_i^{(*)} = C$ (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 Sect. 5.2) 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 with the well-established working-set method LIBSVM [1] for different problem settings. LIBSVM (Version 2.6) is chosen as a typical representative of working-set methods – other implementations like SMO [8] or ISDA [4] show similar characteristics. Both algorithms are available as MEX functions under MATLAB and were compiled with Microsoft Visual C/C++ 6.0. All experiments were done on a 800 MHz Pentium-III PC having 256 MB RAM.

Since the environmental conditions are identical for both algorithms, mainly the *computation time* is considered to measure the performance. By default, both use shrinking heuristics and have enough cache to store the complete kernel matrix if necessary. The influence of these acceleration techniques is examined in Sect. 5.2.

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 before. In order to limit the computation time in the critical cases, a subset of 1000 samples has been selected as training 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, e.g., to find the optimal C by cross-validation. Whereas the active-set method is nearly insensitive with respect to C, the computation time of LIBSVM differs by several magnitudes. Working-set methods typically perform better when the number N_f of *free* variables is small. The computation time of active-set methods mainly depends on the *complete* number N_SV of support vectors which roughly determines the number of iterations.

Table 1. Classification: Variation of C

	C	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}	10^{5}	10^{6}
Active Set	Time	8.7 s	7.4 s	4.3 s	5.4 s	8.4 s	12.3 s	11.1 s	10.8 s
	N_SV	494	480	422	389	379	364	357	337
	N_f	11	20	37	81	139	190	242	271
	Bias	0.9592	0.7829	0.0763	2.0514	3.5940	2.0549	−26.63	−95.56
Active Set ($b = 0$)	Time	7.9 s	6.1 s	3.7 s	5.7 s	8.6 s	11.5 s	11.9 s	9.6 s
	N_SV	510	481	422	391	378	364	360	339
	N_f	14	17	37	78	139	190	245	273
LIB SVM	Time	0.6 s	0.5 s	0.5 s	0.9 s	3.1 s	21.1 s	156.2 s	1198 s
	N_SV	496	481	422	390	379	366	356	334
	N_f	16	22	38	82	139	192	243	268
	Bias	0.9592	0.7826	0.0772	2.0554	3.5927	2.0960	−26.11	−107.56
Error	Train	24.4%	19.0%	16.6%	13.8%	11.4%	8.0%	5.1%	3.4%
	Test	24.6%	18.2%	16.9%	16.6%	17.9%	19.4%	21.5%	23.2%

Also a comparison between the standard SVM and the no-bias SVM (i.e., with bias term fixed at $b = 0$) can be found in Table 1. It shows that there is no need for a bias term when positive definite kernels are used. Although a missing bias usually leads to more support vectors, the results are very close to the standard SVM – even if the bias term takes large values. The errors on the training and testing data set are nearly identical for all three methods. Although the training error can be further reduced by in increasing C, the best generalization performance is achieved with $C = 10^2$ here. In that case LIBSVM is finds the solution very quickly as N_f is still small.

5.2 Estimating the Outlet Temperature of a Boiler

The following example applies the regression algorithm to a system identification problem. The goal is to estimate the outlet temperature T_{31} of a *high*

$$\begin{aligned} T_{31}(k) = f(&T_{41}(k), T_{41}(k-1), T_{41}(k-2),\\ &F_{31}(k), F_{31}(k-1), F_{31}(k-2),\\ &P_{11}(k), P_{11}(k-1), P_{11}(k-2),\\ &T_{31}(k-1), T_{31}(k-2)) \end{aligned} \tag{45}$$

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

efficiency (condensing) boiler from the system temperature T_{41}, the water flow F_{31} and the burner output P_{11} as inputs. Details about the data set under investigation can be found in [13] and [14]. 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. For a sampling time of 30 s 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. The RMSE is the *root-mean-square error* of the *predicted* output on the validation data set. The *simulation* error is not considered because models can be unstable for extreme settings of C.

Table 2. Regression: Variation of C for $\sigma = 3$ and $\tau = 10^{-4}$

	C	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}	10^{5}
Active Set	Time	211.8 s	46.9 s	8.3 s	1.5 s	0.7 s	0.7 s	0.9 s	1.2 s
	RMSE	0.0330	0.0164	0.0097	0.0068	0.0062	0.0062	0.0064	0.0069
	N_{SV}	1938	954	427	143	91	87	92	116
	N_{f}	4	10	25	36	52	77	91	116
LIB SVM	Time	7.9 s	4.5 s	2.7 s	3.0 s	7.7 s	39.1 s	163.2 s	?
	RMSE	0.0330	0.0164	0.0097	0.0068	0.0062	0.0092	0.0064	?
	N_{SV}	1943	963	433	147	95	90	98	?
	N_{f}	10	23	35	45	56	80	97	?

Concerning computation time, Table 2 shows that LIBSVM can efficiently handle a large number N_{SV} of support vectors (with only few free ones) whereas the active-set method shows its strength if N_{SV} is small. For $C = 10^5$ LIBSVM converged extremely slow so that it was aborted after 12 hours. In this example, $C = 10^3$ is the optimal setting concerning support vectors and error. Also the active-set algorithm's memory consumption $\mathcal{O}(N_{\mathrm{f}}^2)$ (see Sect. 1) is not critical: When the number of support vectors increases, typically most of the Lagrange multipliers are bounded at C so that N_{f} remains small.

Table 3. Regression: Variation of σ for $C = 10^3$ and $\tau = 10^{-4}$

	σ	0.5	1	2	3	4	5	6	7
Active Set	Time	2.3 s	1.2 s	0.8 s	0.7 s	0.8 s	0.9 s	0.8 s	1.0 s
	RMSE	0.0278	0.0090	0.0064	0.0062	0.0061	0.0059	0.0059	0.0057
	N_{SV}	184	129	96	87	88	94	96	108
	N_{f}	184	129	94	77	60	49	40	39
LIB SVM	Time	4.1 s	13.7 s	25.1 s	38.2 s	31.8 s	22.4 s	15.4 s	13.1 s
	RMSE	0.0278	0.0091	0.0064	0.0062	0.0061	0.0070	0.0058	0.0057
	N_{SV}	196	134	96	90	92	102	99	110
	N_{f}	196	134	95	80	66	60	42	44
	cond($\mathbf{H}$)	$3{\cdot}10^5$	$8{\cdot}10^6$	$2{\cdot}10^8$	$3{\cdot}10^8$	$2{\cdot}10^8$	$1{\cdot}10^8$	$2{\cdot}10^8$	$2{\cdot}10^8$

A comparison with Table 1 confirms that the computation time for the active-set method mainly depends on the *number* N_{SV} of support vectors, whereas the *ratio* $N_{\mathrm{f}}/N_{\mathrm{SV}}$ has strong influence on working-set methods.

Table 3 examines a variation of the Gaussians' width σ for $C = 10^3$ and $\tau = 10^{-4}$. As expected, the computation time of the active-set algorithm is solely dependent on the number of support vectors. For large σ the computation times of LIBSVM decrease because the fraction of free variables gets smaller, whereas for small σ another effect can be observed: If the condition number the system matrix $\mathbf{H}$ in (33) or (39) decreases, the change in one variable has less effect on the other ones. For that, the computation time decreases although there are only free variables and their number even increases.

Table 4 compares the algorithms for different precisions τ in case of $\sigma = 5$ and $C = 10^2$. Both do not change the active set for precisions smaller then 10^{-5}. Whereas LIBSVM's computation time strongly increases, the active-set method does not need more time to meet a higher precision. Once the active set is found, active-set methods compute the solution with "full" precision, i.e., a smaller τ does not change the solution any more. For low precisions,

Table 4. Regression: Variation of τ for $\sigma = 5$ and $C = 10^2$

	τ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
Active Set	Time	0.1 s	0.3 s	0.9 s	1.0 s	1.1 s	1.1 s	1.1 s	1.1 s
	RMSE	0.0248	0.0063	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059
	N_{SV}	8	49	108	118	122	122	122	122
	N_{f}	7	17	23	33	39	39	39	39
LIB SVM	Time	0.2 s	2.9 s	4.5 s	4.9 s	7.7 s	9.9 s	18.9 s	25.9 s
	RMSE	0.0220	0.0060	0.0059	0.0058	0.0058	0.0058	0.0058	0.0058
	N_{SV}	30	90	119	121	123	123	123	123
	N_{f}	30	73	49	40	39	39	39	39

Table 5. Regression: Influence of shrinking and caching on the computation time for $\sigma = 5$, $C = 10^2$, $\tau = 10^{-4}$

Cached Rows	0	50	100	120	150	200
No shrinking	16.97 s	8.65 s	3.45 s	2.78 s	2.76 s	2.76 s
$s = 10$	5.91 s	3.30 s	1.63 s	1.37 s	1.35 s	1.35 s
$s = 3$	3.80 s	2.22 s	1.24 s	1.07 s	1.05 s	1.05 s
$s = 2$	3.45 s	2.13 s	1.18 s	1.05 s	1.02 s	1.02 s
$s = 1$	2.75 s	1.73 s	1.05 s	0.93 s	0.90 s	0.90 s

the active-set method produces more compact solutions, because it is able to stop earlier due to its complete KKT check in each iteration.

The influence of shrinking and caching is examined in Table 5 for $\sigma = 5$, $C = 10^2$ and $\tau = 10^{-4}$, which yields a SVM having $N_{\mathrm{SV}} = 118$ support vectors. It confirms the estimates given in Sects. 4.4 and 4.5: Both shrinking and caching accelerate the algorithm by a factor of 6 in this example. Used in combination, they lead to a speed-up by nearly a factor of 20. If shrinking is activated, the cache has minor influence because less KKT checks have to be performed. Table 5 also shows that it is not necessary to spend cache for much more than N_{SV} rows, because this only saves the negligible time to search for free rows.

A final experiment demonstrates the approximation method described in Sect. 4.6. With the same settings as above ($\sigma = 5$, $C = 10^2$, $\tau = 10^{-4}$) the complete model contains 118 support vectors. However, Fig. 7 shows that the solution can be approximated with much less support vectors, e.g. 10–15 %.

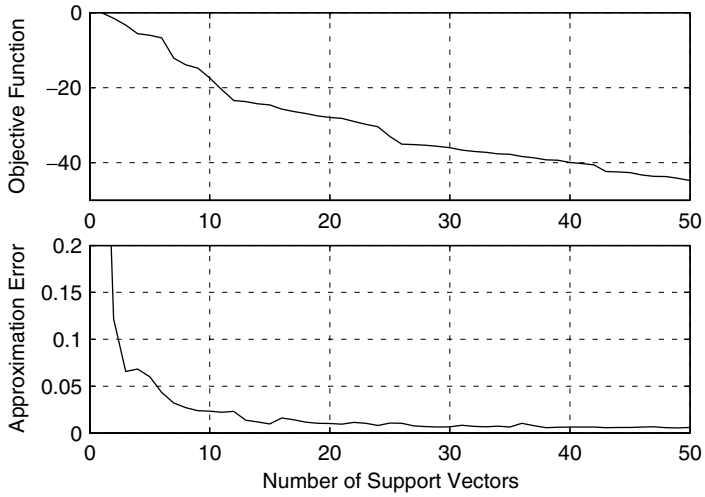


Fig. 7. Regression: Approximation of the solution

Whereas the objective function is still decreasing, more support vectors do not significantly reduce the approximation error.

6 Conclusions

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

- when the number of support vectors is small.
- when the fraction of bounded variables is small.
- when high precision is needed.
- when the problem is ill-conditioned.

Shrinking and caching heuristics can significantly accelerate the algorithm. Additionally, its KKT check can be exploited to approximate the solution with a reduced number of support vectors. Whereas the method is very robust to changes in the settings, it not should be overseen that working-set techniques like LIBSVM are still faster in certain cases and can handle larger data sets.

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

References

1. Chang CC, Lin CJ (2003) LIBSVM: A library for support vector machines. Technical report. National Taiwan University, Taipei, Taiwan 134, 150, 151, 152
2. Gill PE et al. (1974) Methods for Modifying Matrix Computations. Mathematics of Computation 28(126):505–535 149, 150
3. Golub GH, van Loan CF (1996) Matrix Computations. 3rd ed. The Johns Hopkins University Press, Baltimore, MD 141, 148, 149, 150
4. Huang TM, Kecman V (2004) Bias Term b in SVMs again. In: Proceedings of the 12th European Symposium on Artificial Neural Networks (ESANN 2004), pp. 441–448, Bruges, Belgium 134, 138, 152
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 134
6. Blake CL, Merz CJ (1998) UCI repository of machine learning databases. University of California, Irvine, http://www.ics.uci.edu/∼mlearn/ 152

7. Nocedal J, Wright SJ (1999) Numerical Optimization. Springer-Verlag, New York 134, 135, 142, 158
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 134, 151, 152
9. Poggio T et al. (2002) b. In: Winkler J, Niranjan M (eds) Uncertainty in Geometric Computations, pp. 131–141. Kluwer Academic Publishers, Boston 138
10. Schölkopf B, Smola AJ (2002) Lerning with Kernels. The MIT Press, Cambridge, MA 133, 134, 136, 137, 142, 150, 151
11. Vapnik VN (1995) The Nature of Statistical Learning Theory. Springer-Verlag, New York 133, 134, 136
12. Vishwanathan SVN, Smola AJ and Murty MN (2003) SimpleSVM. In: Proceedings of the 20th International Conference on Machine Learning (ICML 2003), pp. 760–767. Washington, DC 134
13. Vogt M, Kecman V (2004) An active-set algorithm for Support Vector Machines in nonlinear system identification. In: Proceedings of the 6th IFAC Symposium on Nonlinear Control Systems (NOLCOS 2004), pp. 495–500. Stuttgart, Germany 134, 154
14. Vogt M, Spreitzer K, Kecman V (2003) Identification of a high efficiency boiler by Support Vector Machines without bias term. In: Proceedings of the 13th IFAC Symposium on System Identification (SYSID 2003), pp. 485–490. Rotterdam, The Netherlands 138, 154

A The Karush-Kuhn-Tucker Conditions

A general constrained optimization problem is given by

$$\min_{\mathbf{a}} \quad J(\mathbf{a}) \tag{46a}$$

$$\text{s.t.} \quad \mathbf{F}(\mathbf{a}) \geq \mathbf{0} \tag{46b}$$

$$\mathbf{G}(\mathbf{a}) = \mathbf{0}\,. \tag{46c}$$

The Lagrangian of this problem is defined as

$$L(\mathbf{a}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = J(\mathbf{a}) - \sum_i \lambda_i F_i(\mathbf{a}) - \sum_i \nu_i G_i(\mathbf{a})\,. \tag{47}$$

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

$$\nabla_{\mathbf{a}} L(\mathbf{a}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = 0 \tag{48a}$$

$$F_i(\mathbf{a}^*) \geq 0 \tag{48b}$$

$$G_i(\mathbf{a}^*) = 0 \tag{48c}$$

$$\lambda_i^* \geq 0 \tag{48d}$$

$$\lambda_i^* F_i(\mathbf{a}^*) = 0 \tag{48e}$$

$$\nu_i^* G_i(\mathbf{a}^*) = 0 \tag{48f}$$

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