Chapter 3

Basic Methods of Least Squares Support Vector Machines

In this Chapter we discuss basic methods of Least Squares Support Vector Machines (LS-SVMs) for classification and nonlinear function estimation. For LS-SVM classifiers we discuss the link with a kernel version of Fisher discriminant analysis in high dimensional feature spaces. LS-SVM regression is closely related to regularization networks, Gaussian processes and reproducing kernel Hilbert spaces but emphasizes primal-dual interpretations in the context of constrained optimization problems. LS-SVM models for classification and nonlinear regression are characterized by linear Karush-Kuhn-Tucker systems. A simple method for imposing sparseness to LS-SVM models is explained by applying pruning techniques as known in neural networks.

3.1 Least Squares Support Vector Machines for classification

A nice aspect of nonlinear SVMs is that one solves nonlinear classification and regression problems by means of convex quadratic programs. Moreover, one also obtains sparseness as a result of this QP problem. A natural question that one may ask is *how much one may simplify the SVM formulation without losing any of its advantages?*

We will propose here a modification to the Vapnik SVM classifier formulation which leads to solving a set of linear equations, which is for many practitioners in different areas, easier to use than QP solvers. The following

SVM modification was originally proposed by Suykens in [235]:

$$\left[\begin{array}{ll} \boxed{\mathrm{P}}: & \min_{w,b,e} J_{\mathrm{P}}(w,e) = \frac{1}{2}w^T w + \gamma \frac{1}{2}\sum_{k=1}^{N} e_k^2 \\ & \text{such that} \quad y_k\left[w^T\varphi(x_k)+b\right] = 1 - e_k,\ k = 1,...,N \end{array}\right] \tag{3.1}$$

for a classifier in the primal space that takes the form

$$y(x) = \mathrm{sign}[w^T\varphi(x) + b] \tag{3.2}$$

where $\varphi(\cdot) : \mathbb{R}^n \to \mathbb{R}^{n_h}$ is the mapping to the high dimensional feature space as in the standard SVM case. The Vapnik formulation is modified here at two points. First, instead of inequality constraints one takes equality constraints where the value 1 at the right hand side is rather considered as a target value than a threshold value. Upon this target value an error variable e_k is allowed such that misclassifications can be tolerated in the case of overlapping distributions. These error variables play a similar role as the slack variables ξ_k in SVM formulations. Second, a squared loss function is taken for this error variable. As we will see now these modifications will greatly simplify the problem.

In the case of a linear classifier one could easily solve the primal problem, but in general w might become infinite dimensional. Therefore let us derive the dual problem for this LS-SVM nonlinear classifier formulation. The Lagrangian for the problem is

$$\mathcal{L}(w,b,e;\alpha) = J_{\mathrm{P}}(w,e) - \sum_{k=1}^{N}\alpha_k\{y_k[w^T\varphi(x_k)+b] - 1 + e_k\} \tag{3.3}$$

where the α_k values are the Lagrange multipliers, which can be positive or negative now due to the equality constraints.

The conditions for optimality yield

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 \;\rightarrow\; w = \sum_{k=1}^{N} \alpha_k y_k \varphi(x_k) \\ \frac{\partial \mathcal{L}}{\partial b} = 0 \;\rightarrow\; \sum_{k=1}^{N} \alpha_k y_k = 0 \\ \frac{\partial \mathcal{L}}{\partial e_k} = 0 \;\rightarrow\; \alpha_k = \gamma e_k, \qquad k = 1, ..., N \\ \frac{\partial \mathcal{L}}{\partial \alpha_k} = 0 \;\rightarrow\; y_k[w^T \varphi(x_k) + b] - 1 + e_k = 0, \quad k = 1, ..., N. \end{cases} \tag{3.4}$$

Defining $Z^T = [\varphi(x_1)^T y_1; ...; \varphi(x_N)^T y_N]$, $y = [y_1; ...; y_N]$, $1_v = [1; ...; 1]$, $e = [e_1; ...; e_N]$, $\alpha = [\alpha_1; ...; \alpha_N]$ and eliminating w, e, one obtains the following linear Karush-Kuhn-Tucker (KKT) system [82; 174]

$$\left[\begin{array}{l} \boxed{\text{D}}: \;\; \text{solve in } \alpha, b: \\ \\ \left[\begin{array}{c|c} 0 & y^T \\ \hline y & \Omega + I/\gamma \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline 1_v \end{array}\right] \end{array} \right] \tag{3.5}$$

where $\Omega = Z^T Z$ and the kernel trick can be applied within the Ω matrix

$$\begin{aligned} \Omega_{kl} &= y_k y_l \, \varphi(x_k)^T \varphi(x_l) \\ &= y_k y_l \, K(x_k, x_l) \;, \quad k, l = 1, ..., N. \end{aligned} \tag{3.6}$$

The classifier in the dual space takes the form

$$y(x) = \text{sign}\left[\sum_{k=1}^{N} \alpha_k y_k K(x, x_k) + b\right] \tag{3.7}$$

similar to the standard SVM case. Note that one could equally well express the solution in terms of the unknown error variables e by eliminating α instead of e.

This LS-SVM classifier has the following properties:

- *Choice of kernel function*:
  The chosen kernel function should be positive definite and satisfy the Mercer condition. All the comments on kernel functions equally well apply to the use of kernels in the LS-SVM context. For the

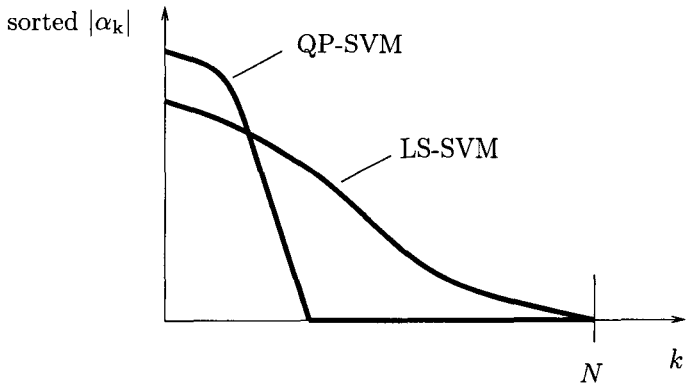

Fig. 3.1 *Qualitative comparison of the sorted $|\alpha|$ solution vectors between QP-SVMs (including the standard Vapnik SVM) and LS-SVMs. In the LS-SVM case each data point is contributing to the model and sparseness is lost.*

sequel of this book we will focus however on the use of linear, polynomial and RBF kernels.

- *Global and unique solution:*
  The dual problem for linear and nonlinear LS-SVMs corresponds to solving a linear KKT system which is a square system with a unique solution when the matrix has full rank.

- *KKT system as a core problem:*
  Solving linear KKT systems is a fundamental issue in constrained nonlinear optimization problems in general. At every iteration step KKT systems of a similar form as (3.5) are solved.
  Also we have seen in the previous Chapter that when one uses interior point methods for solving SVMs with any convex cost function, one finally obtains reduced KKT systems (2.82). Therefore, solving SVM classifiers can be considered as iteratively solving KKT systems where each takes a similar form as one single LS-SVM classifier [171].

- *Lack of sparseness and interpretation of support vectors:*
  A drawback of the simplified formulation is the lack of sparseness.

This is clear from the condition for optimality

$$\alpha_k = \gamma e_k.$$

In the LS-SVM classifier case every data point is a support vector as, normally, no α_k values will be exactly equal to zero. In Fig. 3.1 an illustrative comparison is shown between sorted $|\alpha_k|$ values in the Vapnik SVM case versus the LS-SVM classifier case after taking the absolute value and sorting the values from large to small. In the LS-SVM case this is rather a decaying spectrum without a black/white decision between non-zero and zero α_k values. Hence, one could heuristically say that in the LS-SVM case all training data points will contribute to the model, and certain data points are more important than others. The points with large $|\alpha_k|$ are located close and far from the decision boundary.

- *Non-parametric/parametric issues*:
  LS-SVM classifiers have the same primal-dual neural network interpretations as shown for SVMs in Fig. 2.5. In the primal weight space the problem is parametric with fixed size vector $w \in \mathbb{R}^{n_h}$ where n_h is the number of hidden units in the network interpretation for this space. In the dual space the problem is non-parametric as the size of the solution vector $\alpha \in \mathbb{R}^N$ grows with the number of training data N. Note that the size of the KKT system is not influenced by the dimension of the input space n, but is only determined by N. Hence, typically for large dimensional input spaces one will benefit by solving the dual problem instead of the primal one and often it is only possible to solve the dual (when the dimension of the feature space is very large).

- *Tuning parameters*:
  If one takes for example an RBF kernel $K(x, x_k) = \exp(-\|x - x_k\|_2^2/\sigma^2)$ then only α, b result from solving the linear KKT system (5.12), not the parameters (γ, σ). There are several ways then to proceed in order to determine these parameters. A simple way is to work with a training set, validation set and test set, explore a meaningful grid of possible (γ, σ) combinations and select the values in such a way that they give the best performance on this

validation set. As we discussed in the introductory chapter, the results in this case might be too sensitive with respect to the chosen validation set. In a statistical sense it is therefore better (but computationally heavier) to do n_{CV}-fold cross-validation. Further alternatives are bootstrap techniques and determination of hyperparameters at higher levels by Bayesian inference. VC bounds as discussed in the previous chapter can be applied as well. The advantage of VC bounds is that they do not take any assumptions about the underlying density of the data, except that the data are i.i.d.

In Fig. 3.2 an example is shown of an LS-SVM classifier with RBF kernel for a two spiral classification problem. This problem is known to be difficult for classical MLP classifiers as explained e.g. in [192] due to the fact that a highly nonlinear decision boundary has to be realized. The LS-SVM classifier obtains a zero error on the training set and excellent generalization with a smooth decision boundary as shown in the Figure. This problem is difficult in the sense that a complicated nonlinear decision boundary should be determined, but on the other hand is simple in the sense that the problem is separable (non-overlapping class distributions). A second illustrative example is given in Fig. 3.3 for the Ripley data set. This figure shows the decision boundary and the support values. The good performance on many UCI benchmark data sets will be discussed further in this Chapter.

3.2 Multi-class formulations

The binary LS-SVM classifier can be easily extended to multi-class problems. A traditional approach in neural networks is to take additional output variables as was illustrated e.g. on the alphabet recognition problem in the introductory chapter.

Instead of a single output value y, let us take now multiple output values $y^{(i)}$ with $i = 1, ..., n_y$ where n_y denotes the number of output values. With respect to the choice of the number of outputs the coding is important. In principle it is possible to encode 2^{n_y} classes by means of a number of n_y outputs, but usually this will certainly not be the best possible choice. A better choice is normally to take as many outputs as the number of classes. Let us postpone the discussion here on the best coding/decoding issues

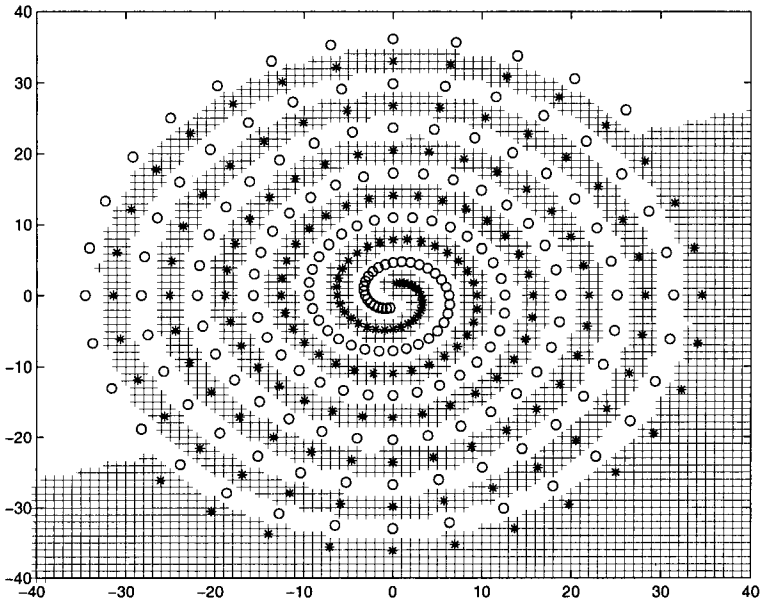

Fig. 3.2 *Excellent classification and generalization performance obtained by an LS-SVM classifier with RBF kernel on a two spiral classification problem. The given training data points for the two classes are shown as (o) and (*). The black/white regions give a visualization of the generalization performance with classification as class 1 (white) or class 2 (black) for given inputs in* $\mathbb{R}^2$ *with inputs in the range [-40,40] which are the variables given at the two axes.*

and focus now on a simple LS-SVM classifier formulation extension to the multi-class case [237].

In the primal weight space the multi-class classification system is based on the outputs of the following binary classifiers

$$\begin{cases} y^{(1)}(x) &= \text{sign}[{w^{(1)}}^T \varphi^{(1)}(x) + b^{(1)}] \\ y^{(2)}(x) &= \text{sign}[{w^{(2)}}^T \varphi^{(2)}(x) + b^{(2)}] \\ & \vdots \\ y^{(n_y)}(x) &= \text{sign}[{w^{(n_y)}}^T \varphi^{(n_y)}(x) + b^{(n_y)}] \end{cases} \tag{3.8}$$

with mappings to high dimensional feature spaces $\varphi^{(i)}(\cdot) : \mathbb{R}^n \to \mathbb{R}^{n_{h_i}}$ (for $i = 1, 2, ..., n_y$) with dimensions $n_{h_1}, n_{h_2}, ..., n_{h_{n_y}}$. One has corresponding vectors $w^{(i)} \in \mathbb{R}^{n_{h_i}}$ and bias terms $b^{(i)} \in \mathbb{R}$.

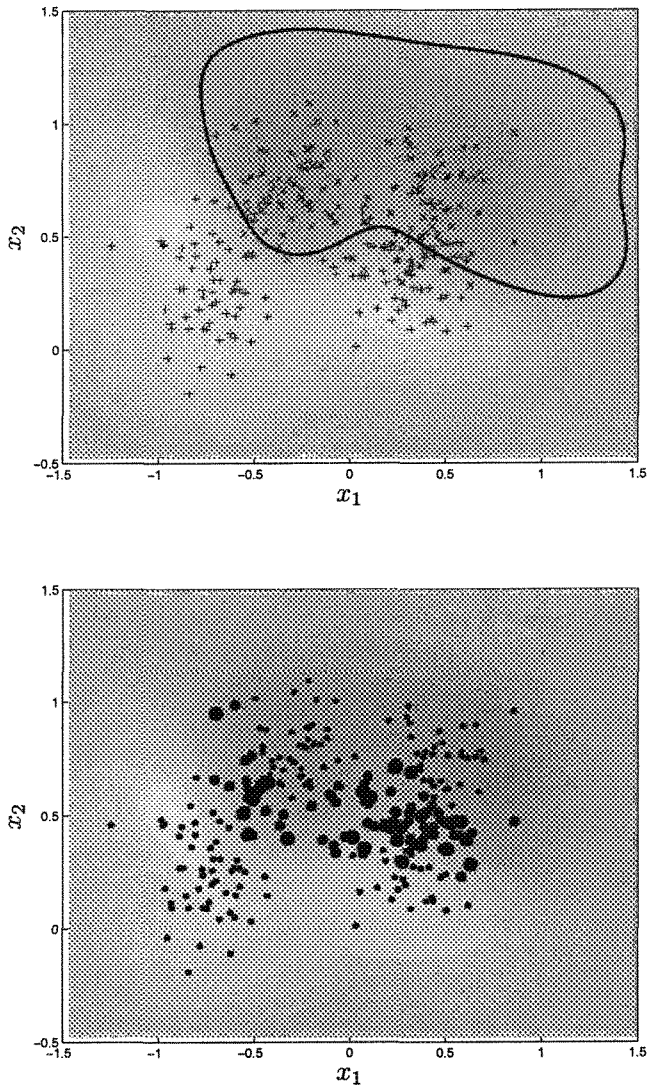

Fig. 3.3 *Illustration of the LS-SVM classifier on the Ripley binary classification data sets: (Top) Decision boundary for LS-SVM with a well-tuned RBF kernel; (Bottom) illustration of the support values α_k with size of the black dots chosen proportional to the α_k values.*

The primal optimization problem for multi-class LS-SVMs becomes then

$$\left[\begin{array}{l} \boxed{\mathrm{P}}: \quad \min_{w^{(i)},b^{(i)},e_k^{(i)}} J_{\mathrm{P}}(w^{(i)},e_k^{(i)}) = \\ \qquad\qquad \frac{1}{2}\sum_{i=1}^{n_y} {w^{(i)}}^T w^{(i)} + \frac{1}{2}\sum_{i=1}^{n_y}\gamma_i \sum_{k=1}^{N}\left(e_k^{(i)}\right)^2 \\ \text{such that} \\ y_k^{(1)}\left[{w^{(1)}}^T\varphi^{(1)}(x_k)+b^{(1)}\right] = 1-e_k^{(1)},\ k=1,...,N \\ y_k^{(2)}\left[{w^{(2)}}^T\varphi^{(2)}(x_k)+b^{(2)}\right] = 1-e_k^{(2)},\ k=1,...,N \\ \vdots \\ y_k^{(n_y)}\left[{w^{(n_y)}}^T\varphi^{(n_y)}(x_k)+b^{(n_y)}\right] = 1-e_k^{(n_y)},\ k=1,...,N. \end{array}\right] \tag{3.9}$$

The Lagrangian for this problem is

$$\begin{aligned} \mathcal{L}(w^{(i)},b^{(i)},e_k^{(i)};\alpha_k^{(i)}) = J_{\mathrm{P}}(w^{(i)},e_k^{(i)}) - \\ \sum_{i=1}^{n_y}\sum_{k=1}^{N}\alpha_k^{(i)}\left(y_k^{(i)}[{w^{(i)}}^T\varphi^{(i)}(x_k)+b^{(i)}]-1+e_k^{(i)}\right) \end{aligned} \tag{3.10}$$

with conditions for optimality

$$\left\{\begin{array}{lll} \frac{\partial\mathcal{L}}{\partial w^{(i)}}=0 & \rightarrow & w^{(i)} = \sum_{k=1}^{N}\alpha_k^{(i)}y_k^{(i)}\varphi^{(i)}(x_k) \\ \frac{\partial\mathcal{L}}{\partial b^{(i)}}=0 & \rightarrow & \sum_{k=1}^{N}\alpha_k^{(i)}y_k^{(i)}=0 \\ \frac{\partial\mathcal{L}}{\partial e_k^{(i)}}=0 & \rightarrow & \alpha_k^{(i)}=\gamma e_k^{(i)} \\ \frac{\partial\mathcal{L}}{\partial \alpha_k^{(i)}}=0 & \rightarrow & y_k^{(i)}[{w^{(i)}}^T\varphi^{(i)}(x_k)+b^{(i)}]=1-e_k^{(i)} \end{array}\right. \tag{3.11}$$

for $k=1,...,N$ and $i=1,...,n_y$.

Finally, after elimination of the variables $w^{(i)}$ and $e_k^{(i)}$ we obtain then

the following KKT system as the dual problem

$$\left[\begin{array}{l} \boxed{\mathrm{D}}: \quad \text{solve in } \alpha_k^{(i)}, b^{(i)}: \\ \\ \left[\begin{array}{c|c} 0 & Y_M^T \\ \hline Y_M & \Omega_M + D_M \end{array}\right] \left[\begin{array}{c} b_M \\ \hline \alpha_M \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline 1_v \end{array}\right] \end{array}\right] \tag{3.12}$$

where

$$Y_M = \text{blockdiag}\{\left[\begin{array}{c} y_1^{(1)} \\ \vdots \\ y_N^{(1)} \end{array}\right], ..., \left[\begin{array}{c} y_1^{(n_y)} \\ \vdots \\ y_N^{(n_y)} \end{array}\right]\}$$

$$\Omega_M = \text{blockdiag}\{\Omega^{(1)}, ..., \Omega^{(n_y)}\}, \qquad \Omega_{kl}^{(i)} = y_k^{(i)} y_l^{(i)} K^{(i)}(x_k, x_l)$$

$$D_M = \text{blockdiag}\{D^{(1)}, ..., D^{(n_y)}\}, \qquad D_{kl}^{(i)} = \delta_{kl}/\gamma_i$$

for $k, l = 1, ..., N$, $i = 1, ..., n_y$ and $b_M = [b^{(1)}; ...; b^{(n_y)}]$, $\alpha_M = [\alpha_1^{(1)}; ...; \alpha_N^{(1)}; ....; \alpha_1^{(n_y)}; ...; \alpha_N^{(n_y)}]$ and δ_{kl} denotes the Kronecker delta ($\delta_{kl} = 1$ if $k = l$ and 0 otherwise). The kernel trick is applied as follows

$$\begin{aligned} K^{(i)}(x_k, x_l) &= \varphi^{(i)}(x_k)^T \varphi^{(i)}(x_l) \\ &= \exp\left(-\|x_k - x_l\|_2^2/\sigma_i^2\right), \ i = 1, ..., n_y \end{aligned} \tag{3.13}$$

illustrated for RBF kernels. Note that for each individual (binary) subclassifier it is important to take different optimal values for (γ_i, σ_i) in the case of an RBF kernel. The resulting classifiers in the dual space are

$$y^{(i)}(x) = \text{sign}[\sum_{k=1}^{N} \alpha_k^{(i)} y_k^{(i)} K^{(i)}(x, x_k) + b^{(i)}] \tag{3.14}$$

for $i = 1, ..., n_y$.

The KKT system (3.12) also has a clear block-diagonal structure. Therefore it is clear that it is not needed to solve the system (3.12) as a whole. One can decompose it into n_y smaller subproblems thanks to the block-diagonal structure. In fact this comes as no surprise because it is well known that multiclass problems can be solved as a set of binary class problems. In general it is important then to find an optimal coding/decoding for the problem [66; 260].

Usually, multiclass categorization problems are solved by reformulating the multiclass problem with $n_{\mathcal{C}}$ classes into a set of n_y binary classification problems. To each class $\mathcal{C}_i$ a unique codeword $[y_i^{(1)}; y_i^{(2)}; \ldots; y_i^{(n_y)}] \in \{-1, +1\}^{n_y}$ for $i = 1, \ldots, n_{\mathcal{C}}$ is assigned. There are several ways then to construct the set of binary classifiers. If one uses n_y outputs to encode up to 2^{n_y} classes, one applies a minimum output coding (MOC). In one-versus-all (1vsA) coding one takes the number of outputs equal to the number of classes or $n_{\mathcal{C}} = n_y$ and makes binary decisions between each class and all other classes. The use of error correcting output codes (ECOC) [66] is motivated by information theory. One introduces redundancy by taking more binary classifiers than the number of classes ($n_y > n_{\mathcal{C}}$) in the output coding. In one-versus-one (1vs1) output coding, one uses $n_{\mathcal{C}}(n_{\mathcal{C}} - 1)/2$ binary plug-in classifiers, where each binary classifier discriminates between two opposite classes in a pairwise way. The 1vs1 output coding can also be represented by n_y length codewords belonging to $\{-1, 0, +1\}^{n_y}$ where one uses 0 for the classes that are not considered. For example, three classes can be represented using the codewords $[-1; +1; 0]$, $[-1; 0; +1]$ and $[0; -1; +1]$. While MOC uses only 2 outputs to encode 3 classes, the use of three outputs in the 1vs1 coding typically results into simpler decision boundaries. In the decoding process, class labels are usually assigned according to the codeword with minimal Hamming distance to the output, but other options are possible as well. When one considers the 1vs1 output coding scheme as a voting between each pair of classes, the codeword with minimal Hamming distance corresponds to the class with the maximal number of votes.

3.3 Link with Fisher discriminant analysis in feature space

In the previous Chapter we have explained that the standard SVM formulation conceptually starts from the margin concept according to which a unique separating hyperplane is defined, while in the separable case in fact several separating hyperplanes exist in general.

We will argue now that a natural link exists between the LS-SVM classifier formulation and Fisher discriminant analysis which is an old and well-known method in pattern recognition, but done now in the high dimensional feature space.

In order to establish and explain this link let us first review the method of linear Fisher discriminant analysis and then explain its extension to the

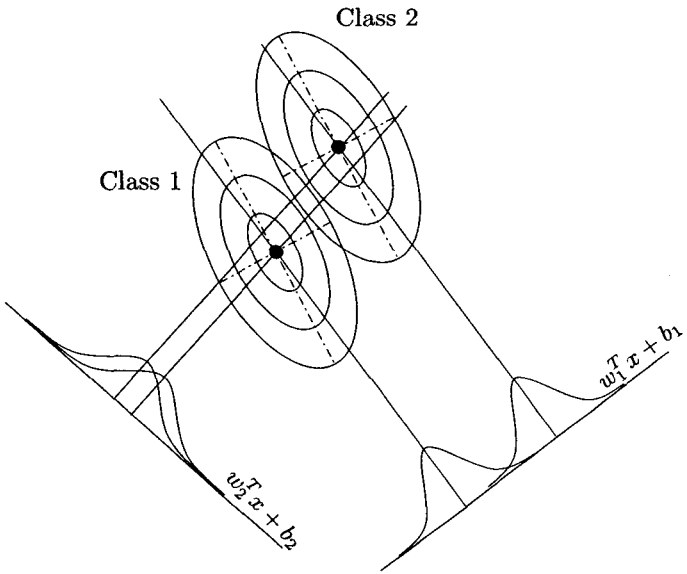

Fig. 3.4 *In Fisher discriminant analysis one aims at maximizing the between-class scatter and minimizing the within-class scatter. One finds a projection of data $x \in \mathbb{R}^n$ onto a one-dimensional space with variable $z \in \mathbb{R}$ such that a Rayleigh quotient is maximized. This figure shows that projection on the line $z = w_1^T x + b_1$ gives a much better discriminatory power than on $z = w_2^T x + b_2$.*

high dimensional feature space.

3.3.1 *Linear Fisher discriminant analysis*

A major goal of Fisher discriminant analysis [80; 81] is to project data $x_k \in \mathbb{R}^n$ from the original input space to a one-dimensional variable $z_k \in \mathbb{R}$ (in a statistical sense this means projecting multivariate data to univariate data) and make a discrimination based on this projected variable. In this one-dimensional space (in fact a straight line) one tries to achieve a high discriminatory power. One tries to maximize the between-class variances and minimize the within-class variances for the two classes (Fig. 3.4).

The data are projected as follows

$$z = f(x) = w^T x + b \tag{3.15}$$

with $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$. We are interested then in finding a line such that the

following objective of a *Rayleigh quotient* is maximized:

$$\begin{aligned}\max_{w,b} J_{\mathrm{FD}}(w,b) &= \frac{[\mathcal{E}[z^{(1)}] - \mathcal{E}[z^{(2)}]]^2}{\mathcal{E}\{[z^{(1)} - \mathcal{E}[z^{(1)}]]^2\} + \mathcal{E}\{[z^{(2)} - \mathcal{E}[z^{(2)}]]^2\}} \\ &= \frac{[w^T(\mu^{(1)} - \mu^{(2)})]^2}{w^T\mathcal{E}\{[x-\mu^{(1)}][x-\mu^{(1)}]^T\}w + w^T\mathcal{E}\{[x-\mu^{(2)}][x-\mu^{(2)}]^T\}w} \\ &= \frac{w^T\Sigma_{\mathcal{B}}w}{w^T\Sigma_{\mathcal{W}}w}\end{aligned} \tag{3.16}$$

where $z_k^{(1)} = w^T x_k^{(1)} + b$, $z_k^{(2)} = w^T x_k^{(2)} + b$ denote the transformed variables belonging to class 1 and class 2, respectively. The means of the input variables for class 1 and class 2 are $\mathcal{E}[x^{(1)}] = \mu^{(1)}, \mathcal{E}[x^{(2)}] = \mu^{(2)}$. The between and within covariance matrices related to class 1 and class 2 are $\Sigma_{\mathcal{B}} = [\mu^{(1)} - \mu^{(2)}][\mu^{(1)} - \mu^{(2)}]^T$, $\Sigma_{\mathcal{W}} = \mathcal{E}\{[x-\mu^{(1)}][x-\mu^{(1)}]^T\} + \mathcal{E}\{[x-\mu^{(2)}][x-\mu^{(2)}]^T\}$ where the latter is the sum of the two covariance matrices $\Sigma_{\mathcal{W}_1}, \Sigma_{\mathcal{W}_2}$ for the two classes. Note that the Rayleigh quotient is independent of the bias term b.

It is well-known that a Rayleigh quotient characterizes the optimality for eigenvalue problems [114; 290]. By taking $\partial J_{\mathrm{FD}}(w)/\partial w = 0$ we indeed obtain the generalized eigenvalue problem $\Sigma_{\mathcal{W}}w = \Sigma_{\mathcal{B}}w\,(w^T\Sigma_{\mathcal{W}}w/w^T\Sigma_{\mathcal{B}}w)$. By using the expression for $\Sigma_{\mathcal{B}}$ one obtains then the following w_{FD} direction for the optimal line

$$w_{\mathrm{FD}} \propto \Sigma_{\mathcal{W}}^{-1}[\mu^{(1)} - \mu^{(2)}]. \tag{3.17}$$

The projections of the means $\mu^{(1)}, \mu^{(2)}$ to the one-dimensional space give

$$\begin{aligned} f(\mu^{(1)}) &= w_{\mathrm{FD}}^T\mu^{(1)} + b \;\propto\; [\mu^{(1)} - \mu^{(2)}]^T\Sigma_{\mathcal{W}}^{-1}\mu^{(1)} \\ f(\mu^{(2)}) &= w_{\mathrm{FD}}^T\mu^{(2)} + b \;\propto\; [\mu^{(1)} - \mu^{(2)}]^T\Sigma_{\mathcal{W}}^{-1}\mu^{(2)}. \end{aligned} \tag{3.18}$$

In practice one works with the sample means

$$\hat{\mu}^{(1)} = \frac{1}{N_1}\sum_{k=1}^{N_1} x_k^{(1)}, \quad \hat{\mu}^{(2)} = \frac{1}{N_2}\sum_{k=1}^{N_2} x_k^{(2)} \tag{3.19}$$

for $\mu^{(1)}, \mu^{(2)}$ of class 1 and 2, respectively, and the sample covariance ma-

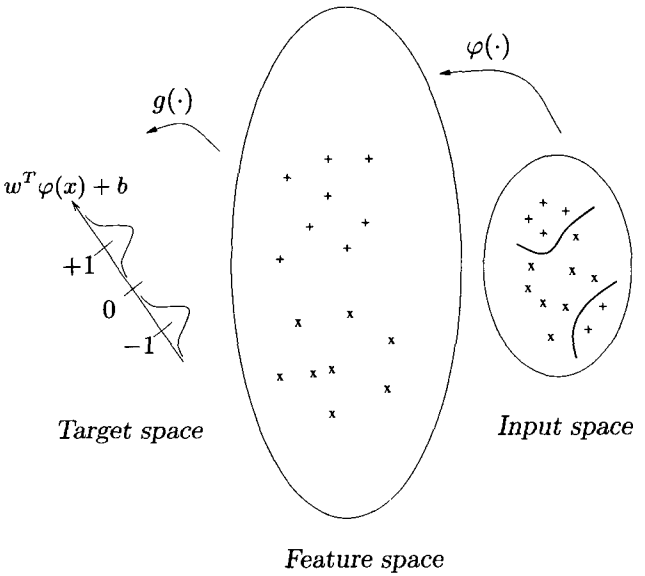

Fig. 3.5 *Fisher discriminant analysis in the feature space is closely related to LS-SVM classification. In LS-SVM classification the constraints* $y_k[w^T\varphi(x_k)+b] = 1-e_k$ *can be interpreted as having target values* $+1$ *and* -1 *on a line for which one aims at minimizing the within-class scattering, which is characterized by the term* $\sum_k e_k^2$ *in the objective function of the LS-SVM classifier.*

trices

$$
\begin{aligned}
S_{\mathcal{W}_1} &= \frac{1}{N_1-1}\sum_{i=1}^{N_1}[x_k^{(1)}-\hat{\mu}^{(1)}][x_k^{(1)}-\hat{\mu}^{(1)}]^T \\
S_{\mathcal{W}_2} &= \frac{1}{N_2-1}\sum_{i=1}^{N_2}[x_k^{(2)}-\hat{\mu}^{(2)}][x_k^{(2)}-\hat{\mu}^{(2)}]^T
\end{aligned}
\tag{3.20}
$$

as estimates for the covariance matrices. Here N_1, N_2 denote the number of data points for class 1 and 2, respectively.

As a classification rule one often computes then the midpoint of the line and checks at which side a projected data point is located in order to make a decision.

3.3.2 *Fisher discriminant analysis in feature space*

Let us now formulate the Fisher discriminant algorithm in a high dimensional feature space instead of the original input space and point out the link with LS-SVM classifiers.

The data are projected as follows

$$z = f(x) = g(\varphi(x)) = w^T\varphi(x) + b \tag{3.21}$$

with $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$ the mapping from the input space to the one-dimensional target space, $g(\cdot) : \mathbb{R}^{n_h} \to \mathbb{R}$ the mapping from the high dimensional feature space to the target space and $\varphi(\cdot) : \mathbb{R}^n \to \mathbb{R}^{n_h}$ the mapping from the input space to the high dimensional feature space, which can be infinite dimensional.

As for linear Fisher discriminant analysis we aim now at maximizing the between-class scatter and minimize the within-class scatter for the projected one-dimensional variable z. Note that the LS-SVM classifier constraints

$$y_k[w^T\varphi(x_k) + b] = 1 - e_k\,, \quad k = 1, ..., N \tag{3.22}$$

can be interpreted as having target values $+1$ and -1 upon which one wants to minimize the errors e_k (Fig. 3.5). This is done by taking a squared error in the cost function which corresponds to minimizing the within-class scatter for classes 1 and 2. Hence, in the LS-SVM classifier formulation one fixes in fact the between-class scatter by taking target values $+1$ and -1 for the z target variable on the line $z = w^T\varphi(x) + b$.

Fisher discriminant analysis in the feature space optimizes then the same Rayleigh quotient as in (3.16), where $z_k^{(1)} = w^T\varphi(x_k^{(1)}) + b$, $z_k^{(2)} = w^T\varphi(x_k^{(2)}) + b$ denote now the projected variables belonging to class 1 and class 2, respectively. The means of the input variables for class 1 and class 2 are $\mathcal{E}[\varphi(x^{(1)})] = \mu^{(1)}, \mathcal{E}[\varphi(x^{(2)})] = \mu^{(2)}$. The between and within covariance matrices related to class 1 and class 2 are $\Sigma_{\mathcal{B}} = [\mu^{(1)} - \mu^{(2)}][\mu^{(1)} - \mu^{(2)}]^T$, $\Sigma_{\mathcal{W}} = \mathcal{E}\{[\varphi(x) - \mu^{(1)}][\varphi(x) - \mu^{(1)}]^T\} + \mathcal{E}\{[\varphi(x) - \mu^{(2)}][\varphi(x) - \mu^{(2)}]^T\}$ where the latter is the sum of the two covariance matrices $\Sigma_{\mathcal{W}_1}, \Sigma_{\mathcal{W}_2}$ for the two classes.

Note that these matrices can become *infinite dimensional* now, as well as the solution vector w_{FD}. Therefore, for the nonlinear case one may better solve the dual problem and apply the kernel trick. This dual problem corresponds then to solving the LS-SVM classifier problem (5.12). Of course instead of fixing the targets at $+1$ and -1, one could generalize this to other

values and allow more freedom in the numerator of the Rayleigh quotient in this way. This would lead to a LS-SVM classifier formulation in the primal weight space as follows

$$\left[\begin{array}{l} \boxed{\mathrm{P}}: \quad \min\limits_{w,b,e} J_{\mathrm{P}}(w,e) = \frac{1}{2}w^T w + \gamma\frac{1}{2}\left(\sum\limits_{k=1}^{N_1}(e_k^{(1)})^2 + \sum\limits_{l=1}^{N_2}(e_l^{(2)})^2\right) \\ \qquad\qquad \text{such that} \quad y_k[w^T\varphi(x_k)+b] = t_1 - e_k^{(1)}, k = 1,\ldots,N_1 \\ \qquad\qquad\qquad\quad\; y_l[w^T\varphi(x_l)+b] = t_2 - e_l^{(2)}, l = 1,\ldots,N_2 \end{array}\right] \tag{3.23}$$

where t_1, t_2 are fixed and positive constant target values for class 1 and 2 and N_1, N_2 are the number of data points belonging to class 1 and 2 respectively. The indices k, l run over the elements of class 1 and 2, respectively. In a regression interpretation context one has targets $z_1 = t_1$ and $z_2 = -t_2$. It is an easy exercise to work out the dual problem of this. In [264] it was shown, however, that it is usually better to work with $t_1 = t_2 = 1$ as in the original LS-SVM classifier formulation.

As we explained, the link between LS-SVM classification and Fisher discriminant analysis is very natural. From this analysis we can state that the goal of LS-SVM classifiers is basically twofold:

(1) Maximizing the soft margin by minimizing $\|w\|_2$ for separable or non-separable data either in the linear or nonlinear case.

(2) Minimizing the within-class scatter for fixed targets on a line with a similar objective as Fisher discriminant analysis but in a high dimensional feature spaces.

Other formulations and links with kernel based methods have been discussed also in [16; 163; 206].

3.4 Solving the LS-SVM KKT system

3.4.1 *Conjugate gradient iterative methods*

The solution to linear and nonlinear LS-SVM classifiers is characterized by a square linear system of equations. For many decades several methods have been developed in numerical analysis for reliably solving sets of linear equations and many specialized algorithms have been developed that

maximally try to exploit structure of the problem [98; 100; 256; 301].

The size of the matrix grows with the number of data points. Therefore, direct elimination methods are usually restricted to data sets of size of about $N = 2000$, depending on the available computer memory and assuming that one stores the full matrix into memory. For larger data sets the use of iterative methods is recommended. In principle, various methods can be used at this point including SOR (Successive Over-Relaxation), CG (Conjugate Gradient), GMRES (Generalized Minimal Residual) etc. However, not all of these iterative methods can be applied to any kind of linear system. For example, in order to apply CG the matrix should be positive definite. Due to the presence of the b bias term in the LS-SVM model the resulting matrix is not positive definite (in fact it is well known that KKT systems as arising in constrained optimization problems are indefinite). So before we can apply such methods we have to transform the linear system into a positive definite system. According to [236] this can be done in a simple way as follows.

The LS-SVM KKT system is of the form

$$\begin{bmatrix} 0 & y^T \\ y & H \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{3.24}$$

more specifically with $H = \Omega + I/\gamma$, $\xi_1 = b$, $\xi_2 = \alpha$, $d_1 = 0$, $d_2 = 1_v$. This can be transformed into

$$\begin{bmatrix} s & 0 \\ 0 & H \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 + H^{-1} y \xi_1 \end{bmatrix} = \begin{bmatrix} -d_1 + y^T H^{-1} d_2 \\ d_2 \end{bmatrix} \tag{3.25}$$

with $s = y^T H^{-1} y > 0$ ($H = H^T > 0$). Because s is positive and H positive definite the overall matrix is positive definite. This form is very suitable because different kinds of iterative methods can be applied to problems involving positive definite matrices. This leads to the following algorithm for solving LS-SVM classifiers with application of a conjugate gradient algorithm.

LS-SVM CG large scale algorithm:

(1) Solve η, ν from $H\eta = y$ and $H\nu = 1_v$.

(2) Compute $s = y^T \eta$.

(3) Find solution: $b = \eta^T 1_v / s$ and $\alpha = \nu - \eta b$.

The system (3.25) is of the form $\mathcal{A}x = \mathcal{B}$ with $\mathcal{A} = \mathcal{A}^T > 0$. A basic *Hestenes-Stiefel CG algorithm* [98] works as follows

$$
\begin{array}{l}
i = 0; x_0 = 0; r_0 = \mathcal{B}; \\
\text{while } r_i \neq 0 \\
\quad i = i + 1 \\
\quad \text{if } i = 1 \\
\quad\quad p_1 = r_0 \\
\quad \text{else} \\
\quad\quad \beta_i = r_{i-1}^T r_{i-1} / r_{i-2}^T r_{i-2} \\
\quad\quad p_i = r_{i-1} + \beta_i p_{i-1} \\
\quad \text{end} \\
\quad \lambda_i = r_{i-1}^T r_{i-1} / p_i^T \mathcal{A} p_i \\
\quad x_i = x_{i-1} + \lambda_i p_i \\
\quad r_i = r_{i-1} - \lambda_i \mathcal{A} p_i \\
\text{end} \\
x = x_i
\end{array}
$$

Some comments on the application of this conjugate gradient method:

- The underlying cost function optimized by the CG algorithm is of the quadratic form $V(x) = \frac{1}{2} x^T \mathcal{A} x - x^T \mathcal{B}$. In the application of this algorithm it has been experienced that especially the decrease of $V(x)$ is important towards a suitable stopping criterion in this LS-SVM context. The main reason for using a measure based on the difference $V(x_i) - V(x_{i-1})$ between two iteration steps i and $i-1$, is that $V(x)$ monotonically decreases. This is not the case for the norm on the residual $\mathcal{A}x - \mathcal{B}$.

- The complexity of CG methods can be understood as follows [25; 98; 217]. Let x_i be the i-th iteration by the CG method when solving $\mathcal{A}x = \mathcal{B}$ with $\mathcal{A}$ real symmetric and positive definite, then one has

$$
\|x_i - x_*\|_{\mathcal{A}} \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \|x_0 - x_*\|_{\mathcal{A}} \tag{3.26}
$$

where the $\mathcal{A}$-norm of a vector v is defined as $\|v\|_{\mathcal{A}} = (v^T \mathcal{A} v)^{1/2}$ and $\kappa = \|\mathcal{A}\| \, \|\mathcal{A}^{-1}\|$ denotes the condition number. One can conclude

[217] that if

$$i \geq \frac{\log \frac{2}{\epsilon}}{\log\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)} = \mathcal{O}\left(\frac{\sqrt{\kappa}}{2}\log\frac{2}{\epsilon}\right) \tag{3.27}$$

then $\|x_i - x_*\|_{\mathcal{A}} \leq \epsilon \|x_0 - x_*\|_{\mathcal{A}}$ with accuracy ϵ. Hence, the time needed to solve the problem depends on the requested precision $\log \frac{1}{\epsilon}$ and the condition number of matrix $\mathcal{A}$. In contrast with direct algorithms such as Gauss elimination the computational complexity does not just depend on the size of the matrix $\mathcal{A}$. It is important to note that when using e.g. an LS-SVM classifier with RBF kernel, the matrix $\mathcal{A}$ will depend on the tuning parameters (γ, σ) and as a result also the condition number κ and the speed of convergence. Also in [293] quick convergence has been reported of conjugate gradient algorithms in the context of kernel based learning.

- The basic CG algorithm shown here might be further improved by using preconditioners. On the other hand this basic CG algorithm has shown to perform sufficiently well and fast on a large variety of problems. For the implementation of the algorithm the matrix $\mathcal{A}$ is usually not stored. The elements of the matrix are re-calculated every iteration step. Clever tricks may however be applied to optimally use the computer memory.

The CG method has also been compared with SOR methods in [103] where CG and block CG methods performed much better on several tests. Recently in [132] also squential minimal optimization (SMO) has been successfully applied to solving LS-SVM systems.

3.4.2 *LS-SVM classifiers: UCI benchmarking results*

Tuning parameter selection

Let us take a look now at some benchmarking results on UCI binary and multiclass data sets [24] as reported by Van Gestel *et al.* in [264]. The LS-SVM classifiers are tested for linear, polynomial and RBF kernels. A simple method of tuning parameter selection based on 10-fold cross-validation with

several randomizations for the data sets is applied. In the case of an RBF kernel, the hyperparameter γ, the kernel parameter σ and the test set performance of the binary LS-SVM classifier are estimated by applying the following algorithm of a shrinking grid search.

LS-SVM tuning - grid search algorithm:

(1) Divide data set into 2/3 for training and validation and 1/3 for a test set.

(2) Start $i = 0$. Apply 10-fold cross-validation on the training/validation data for each (σ, γ) combination from the initial candidate tuning sets at iteration 0:

$$\begin{aligned} \Sigma_0 &= \{0.5, 5, 10, 15, 25, 50, 100, 250, 500\} \text{ (elements } \times \sqrt{n}) \\ \Gamma_0 &= \{0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000\}. \end{aligned}$$

The square root $\sqrt{n}$ of the number of inputs n is considered in the grid since the value of $||x - x_k||_2^2$ in the RBF kernel grows with the value of n.

(3) Choose optimal (σ, γ) from the tuning sets Σ_i and Γ_i by looking at the best cross-validation performance.

(4) If $i = i_{max}$, go to step 5; else $i := i + 1$, construct a locally refined grid $\Sigma_i \times \Gamma_i$ around the optimal hyperparameters (σ, γ) and go to step 3.

(5) Construct the LS-SVM classifier using the total training/validation set for the optimal choice of the tuned hyperparameters (σ, γ).

(6) Assess the test set accuracy by means of the independent test set.

In this benchmark study, i_{max} was chosen as $i_{max} = 3$. For the polynomial kernels (γ, τ) was tuned following a similar procedure, while the γ parameter for the linear kernel was selected from a refined set Γ based upon the cross-validation performance. For multiclass problems, this cross-validation procedure is applied to each binary subproblem and the different coding/decoding schemes.

Benchmark data sets

The UCI datasets can be obtained from the UCI benchmark repository [24] at `http://kdd.ics.uci.edu/`. The US postal service is available from `http://www.kernel-machines.org/`. These datasets have been referred numerous times in the literature, which makes them very suitable for benchmarking purposes. As a preprocessing step, all records containing unknown values are removed and all data sets were normalized for the benchmarking process. The following binary datasets were retrieved from [24]: Statlog Australian credit (`acr`), Bupa liver disorders (`bld`), Statlog German credit (`gcr`), Statlog heart disease (`hea`), Johns Hopkins university ionosphere (`ion`), Pima Indians diabetes (`pid`), sonar (`snr`), tic-tac-toe endgame (`ttt`), Wisconsin breast cancer (`wbc`) and adult (`adu`) dataset. The main characteristics of these datasets are summarized in Table 3.1. The following multiclass datasets were used: balance scale (`bal`), contraceptive method choice (`cmc`), image segmentation (`ims`), iris (`iri`), LED display (`led`), thyroid disease (`thy`), US postal service (`usp`), Statlog vehicle silhouette (`veh`), waveform (`wav`) and wine recognition (`win`) dataset. The main characteristics of the multiclass datasets are summarized in Table 3.2.

Compared methods

According to [264] the following methods are compared in Tables 3.3 & 3.4: linear, polynomial and RBF kernels for LS-SVM classifiers, standard SVM classifiers with linear and RBF kernel; the decision tree algorithm C4.5 [188], Holte's one-rule classifier (oneR) [113]; linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), logistic regression (Logit) [23; 71; 193]; instance based learners (IB) [3] and Naive Bayes [126]. The Matlab SVM toolbox `http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox` with SMO solver was used to train and evaluate the Vapnik SVM classifier. The C4.5, IB1, IB10, Naive Bayes and oneR algorithms were implemented using the Weka workbench [295], while the Discriminant Analysis Toolbox (M. Kiefte) for Matlab was applied for LDA, QDA and Logit. The oneR, LDA, QDA, Logit, NBk and NBn require no parameter tuning. Both standard Naive Bayes with the normal approximation (NB_n) [71] and the kernel approximation (NB_k) for numerical attributes [126] were used. The default classifier or majority rule (Maj. Rule) was included as a baseline in the comparison tables. All comparisons were made

on the same randomizations. For a further comparison study among 22 decision tree, 9 statistical and 2 neural network algorithms, see [141]. For each algorithm, the average test set performance and sample standard deviation on 10 randomizations in each domain is reported [18; 62; 68; 141]. Averaging over all domains, the Average Accuracy (AA) and Average Rank (AR) are shown for each algorithm [141]. A Wilcoxon signed rank test of equality of medians is used on both AA and AR to check whether the performance of an algorithm is significantly different from the algorithm with the highest accuracy. A Probability of a Sign Test (P_{ST}) is also reported comparing each algorithm to the algorithm with best accuracy.

LS-SVM classifier performance on binary class problems

The benchmarking results for the binary class problem are shown in Table 3.3. For the kernel types, RBF kernels, linear (Lin) and polynomial (Pol) kernels (with degree $d = 2, \ldots, 10$) were tried. Both the performance of LS-SVM targets $\{-1, +1\}$ and Regularized Kernel Fisher Discriminant Analysis (LS-SVM_F) targets $\{-N/N_2, +N/N_1\}$ are reported.

The experimental results indicate that the RBF kernel yields the best validation and test set performance, while also polynomial kernels yield good performances (if one allows τ values non equal to 1). LS-SVMs with polynomial kernels ($\tau = 1$) of degrees $d = 2$ and $d = 10$ yielded on all domains average test set performances of 84.3% and 65.9%, respectively. Comparing this performance with the average test set performance of 85.6% and 85.5% obtained when using scaling ($\tau \neq 1$), this clearly motivates the use of bandwidth or kernel parameters. This is especially important for polynomial kernels with degree $d \geq 5$. The regularization parameter c and kernel parameter σ of the SVM classifiers with linear and RBF kernels were selected in a similar way as for the LS-SVM classifier using the 10-fold cross-validation procedure.

The LS-SVM classifier with RBF kernel achieves the best average test set performance on 3 of the 10 benchmark domains, while its accuracy is not significantly worse than the best algorithm in 3 other domains. LS-SVM classifiers with polynomial and linear kernel yield the best performance on two and one datasets, respectively. Also RBF SVM, IB1, NB_k and C4.5 achieve the best performance on one dataset each. Comparison of the accuracy achieved by the nonlinear polynomial and RBF kernel with the accuracy of the linear kernel illustrates that most domains are only

weakly nonlinear. The LS-SVM formulation with targets $\{-1, +1\}$ yields a better performance than the LS-SVM$_F$ regression formulation related to regularized kernel Fisher's discriminant with targets $\{-N/N_2, +N/N_1\}$, although not all tests report a significant difference. Noticing that the LS-SVM with linear kernel without regularization ($\gamma \to \infty$) corresponds to the LDA classifier, we also remark that a comparison of both accuracies indicates that the use of regularization slightly improves the generalization behaviour. Considering the Average Accuracy (AA) and Average Ranking (AR) over all domains [18; 62; 68], the RBF SVM gets the best average accuracy and the RBF LS-SVM yields the best average rank. There is no significant difference between the performance of both classifiers. The average performance of Pol LS-SVM and Pol LS-SVM$_F$ is not significantly different with respect to the best algorithms. The performances of many other advanced SVM algorithms are in line with the above results [30; 163; 202]. The significance tests on the average performances of the other classifiers do not always yield the same results. Generally speaking, the performance of Lin LS-SVM, Lin SVM, Logit, NB$_k$ and IB1 are not significantly different at the 1% level. Also the performances of LS-SVMs with Fisher targets (LS-SVM$_F$) are not significantly different at the 1%. From these results we may conclude that both the SVM and LS-SVM classifiers achieve very good test set performances in comparison with the other reference classification algorithms.

LS-SVM classifier performance on multiclass problems

The benchmarking results for the binary class problem are shown in Table 3.4. Each multiclass problem is decomposed into a set of binary classification problems using minimum output coding (MOC) and one-versus-one (1vs1) output coding. The same kernel types as for the binary domain were considered: RBF kernels, linear (Lin) and polynomial (Pol) kernels with degrees $d = 2, \ldots, 10$. Both the performance of LS-SVM and LS-SVM$_F$ classifiers are reported. The MOC and 1vs1 output coding were also applied to SVM classifiers with linear and RBF kernels.

The average test set accuracies of the different LS-SVM and LS-SVM$_F$ classifiers, with RBF, Lin and Pol kernel ($d = 2, \ldots, 10$) and using MOC and 1vs1 output coding, are reported in Table 3.4. The use of QDA yields the best average test set accuracy on two domains, while LS-SVMs with 1vs1 coding using a RBF and Lin kernel and LS-SVM$_F$ with Lin kernel each

yield the best performance on one domain. SVMs with RBF kernel with MOC and 1vs1 coding yield the best performance on one domain each. Also C4.5, Logit and IB1 each achieve one time the best performance. The use of 1vs1 coding generally results in a better classification accuracy. Averaging over all 10 multiclass domains, the LS-SVM classifier with RBF kernel and 1vs1 output coding achieves the best average accuracy (AA) and average ranking, while its performance is only on three domains significantly worse at 1% than the best algorithm. This performance is not significantly different from the SVM with RBF kernel and 1vs1 output coding. The results illustrate that the SVM and LS-SVM classifier with RBF kernel using 1vs1 output coding consistently yield very good test set accuracies on the multiclass domains.

	acr	bld	gcr	hea	ion	pid	snr	ttt	wbc	adu
N_{CV}	460	230	666	180	234	512	138	638	455	33000
N_{test}	230	115	334	90	117	256	70	320	228	12222
N	690	345	1000	270	351	768	208	958	683	45222
n_{num}	6	6	7	7	33	8	60	0	9	6
n_{cat}	8	0	13	6	0	0	0	9	0	8
n	14	6	20	13	33	8	60	9	9	14

Table 3.1 *Characteristics of binary classification UCI datasets, where N_{CV} stands for the number of data points used in the cross-validation based tuning procedure, N_{test} for the number of observations in the test set and N for the total dataset size. The number of numerical and categorical attributes is denoted by n_{num} and n_{cat} respectively, n is the total number of attributes.*

	bal	cmc	ims	iri	led	thy	usp	veh	wav	win
N_{CV}	416	982	1540	100	2000	4800	6000	564	2400	118
N_{test}	209	491	770	50	1000	2400	3298	282	1200	60
N	625	1473	2310	150	3000	7200	9298	846	3600	178
n_{num}	4	2	18	4	0	6	256	18	19	13
n_{cat}	0	7	0	0	7	15	0	0	0	0
n	4	9	18	4	7	21	256	18	19	13
M	3	3	7	3	10	3	10	4	3	3
$n_{y,\text{MOC}}$	2	2	3	2	4	2	4	2	2	2
$n_{y,\text{1vs1}}$	3	3	21	3	45	3	45	6	2	3

Table 3.2 *Characteristics of the multiclass datasets, where N_{CV} stands for the number of data points used in the cross-validation based tuning procedure, N_{test} for the number of data in the test set and N for the total amount of data. The number of numerical and categorical attributes is denoted by n_{num} and n_{cat} respectively, n is the total number of attributes. The M row denotes the number of classes for each dataset, encoded by $n_{y,\text{MOC}}$ and $n_{y,\text{1vs1}}$ bits for MOC and 1vs1 output coding, respectively.*

	acr	bld	gcr	hea	ion	pid	snr	ttt	wbc	adu	AA	AR	P_{ST}
N_{test}	230	115	334	90	117	256	70	320	228	12222			
n	14	6	20	13	33	8	60	9	9	14			
RBF LS-SVM	**87.0**(2.1)	**70.2**(4.1)	**76.3**(1.4)	**84.7**(4.8)	**96.0**(2.1)	**76.8**(1.7)	*73.1*(4.2)	*99.0*(0.3)	*96.4*(1.0)	*84.7*(0.3)	**84.4**	**3.5**	**0.727**
RBF LS-SVM$_F$	**86.4**(1.9)	*65.1*(2.9)	*70.8*(2.4)	83.2(5.0)	*93.4*(2.7)	*72.9*(2.0)	*73.6*(4.6)	*97.9*(0.7)	96.8(0.7)	*77.6*(1.3)	*81.8*	8.8	**0.109**
Lin LS-SVM	**86.8**(2.2)	*65.6*(3.2)	75.4(2.3)	**84.9**(4.5)	*87.9*(2.0)	76.8(1.8)	*72.6*(3.7)	*66.8*(3.9)	*95.8*(1.0)	*81.8*(0.3)	79.4	7.7	**0.109**
Lin LS-SVM$_F$	**86.5**(2.1)	*61.8*(3.3)	*68.6*(2.3)	*82.8*(4.4)	*85.0*(3.5)	*73.1*(1.7)	73.3(3.4)	*57.6*(1.9)	**96.9**(0.7)	*71.3*(0.3)	*75.7*	12.1	**0.109**
Pol LS-SVM	**86.5**(2.2)	**70.4**(3.7)	**76.3**(1.4)	**83.7**(3.9)	*91.0*(2.5)	**77.0**(1.8)	**76.9**(4.7)	**99.5**(0.5)	*96.4*(0.9)	*84.6*(0.3)	**84.2**	**4.1**	**0.727**
Pol LS-SVM$_F$	**86.6**(2.2)	*65.3*(2.9)	*70.3*(2.3)	*82.4*(4.6)	*91.7*(2.6)	*79.0*(1.8)	**77.3**(2.6)	*98.1*(0.8)	**96.9**(0.7)	*77.9*(0.2)	*82.0*	**8.2**	**0.344**
RBF SVM	*86.9*(1.8)	**70.4**(3.2)	**75.9**(1.4)	**84.7**(4.8)	95.4(1.7)	**77.3**(2.2)	**75.0**(6.6)	*98.6*(0.5)	*96.4*(1.0)	*84.4*(0.3)	**84.4**	**4.0**	**1.000**
Lin SVM	**86.7**(2.4)	*67.7*(2.6)	75.4(1.7)	*83.2*(4.2)	*87.1*(3.4)	77.0(2.4)	*74.1*(4.2)	*66.2*(3.6)	*96.3*(1.0)	*83.9*(0.2)	*79.8*	**7.5**	0.021
LDA	*85.9*(2.2)	*65.4*(3.2)	**75.9**(2.0)	**83.9**(4.3)	*87.1*(2.3)	76.7(2.0)	*67.9*(4.9)	*68.0*(3.0)	*95.6*(1.1)	*82.2*(0.3)	*78.9*	*9.6*	*0.004*
QDA	*80.1*(1.9)	*62.2*(3.6)	*72.5*(1.4)	*78.4*(4.0)	*90.6*(2.2)	*74.2*(3.3)	*53.6*(7.4)	*75.1*(4.0)	*94.5*(0.6)	*80.7*(0.3)	*76.2*	*12.6*	*0.002*
Logit	**86.8**(2.4)	*66.3*(3.1)	**76.3**(2.1)	*82.9*(4.0)	*86.2*(3.5)	**77.2**(1.8)	*68.4*(5.2)	*68.3*(2.9)	*96.1*(1.0)	*83.7*(0.2)	79.2	7.8	**0.109**
C4.5	*85.5*(2.1)	*63.1*(3.8)	*71.4*(2.0)	*78.0*(4.2)	*90.6*(2.2)	*73.5*(3.0)	*72.1*(2.5)	*84.2*(1.6)	*94.7*(1.0)	**85.6**(0.3)	*79.9*	*10.2*	0.021
oneR	*85.4*(2.1)	*56.9*(4.4)	*66.0*(3.0)	*71.7*(3.6)	*83.6*(4.8)	*71.3*(2.7)	*62.6*(5.5)	*70.7*(1.5)	*91.8*(1.4)	*80.4*(0.3)	*74.0*	*15.5*	*0.002*
IB1	*81.1*(1.9)	*61.5*(6.2)	*69.3*(2.6)	*74.3*(4.2)	*87.2*(2.8)	*69.6*(2.4)	**77.7**(4.4)	*82.3*(3.3)	*95.3*(1.1)	*78.9*(0.2)	*77.7*	12.5	0.021
IB10	**86.4**(1.3)	*60.5*(4.4)	*72.6*(1.7)	*80.0*(4.3)	*85.9*(2.5)	*73.6*(2.4)	*69.4*(4.3)	*94.8*(2.0)	*96.4*(1.2)	*82.7*(0.3)	*80.2*	*10.4*	0.039
NB$_k$	*81.4*(1.9)	*63.7*(4.5)	*74.7*(2.1)	83.9(4.5)	*92.1*(2.5)	75.5(1.7)	*71.6*(3.5)	*71.7*(3.1)	**97.1**(0.9)	*84.8*(0.2)	*79.7*	**7.3**	**0.109**
NB$_n$	*76.9*(1.7)	*56.0*(6.9)	74.6(2.8)	**83.8**(4.5)	*82.8*(3.8)	75.1(2.1)	*66.6*(3.2)	*71.7*(3.1)	*95.5*(0.5)	*82.7*(0.2)	*76.6*	*12.3*	*0.002*
Maj. Rule	*56.2*(2.0)	*56.5*(3.1)	*69.7*(2.3)	*56.3*(3.8)	*64.4*(2.9)	*66.8*(2.1)	*54.4*(4.7)	*66.2*(3.6)	*66.2*(2.4)	*75.3*(0.3)	*63.2*	*17.1*	*0.002*

Table 3.3 *Comparison of the 10 times randomized* **test set** *performance of LS-SVM and LS-SVM$_F$ (linear, polynomial and Radial Basis Function kernel) with the performance of LDA, QDA, Logit, C4.5, oneR, IB1, IB10, NB$_k$, NB$_n$ and the Majority Rule classifier on 10 binary domains. The Average Accuracy (AA), Average Rank (AR) and Probability of equal medians using the Sign Test (P_{ST}) taken over all domains are reported in the last three columns. Best performances are underlined and denoted in bold face, performances not significantly different at the 5% level are denoted in bold face, performances significantly different at the 1% level are emphasized. LS-SVMs with RBF kernels are performing very well in this comparative study.*

	bal	cmc	ims	iri	led	thy	usp	veh	wav	win	AA	AR	P_{ST}
N_{test}	209	491	770	50	1000	2400	3298	282	1200	60			
n	4	9	18	4	7	21	256	18	19	13			
RBF LS-SVM (MOC)	*92.7*(1.0)	**54.1**(1.8)	*95.5*(0.6)	*96.6*(2.8)	*70.8*(1.4)	*96.6*(0.4)	*95.3*(0.5)	81.9(2.6)	**99.8**(0.2)	**98.7**(1.3)	**88.2**	**7.1**	**0.344**
RBF LS-SVM$_F$ (MOC)	*86.8*(2.4)	*43.5*(2.6)	*69.6*(3.2)	**98.4**(2.1)	*36.1*(2.4)	*22.0*(4.7)	*86.5*(1.0)	*66.5*(6.1)	*99.5*(0.2)	*93.2*(3.4)	*70.2*	17.8	**0.109**
Lin LS-SVM (MOC)	*90.4*(0.8)	*46.9*(3.0)	*72.1*(1.2)	*89.6*(5.6)	*52.1*(2.2)	*93.2*(0.6)	*76.5*(0.6)	*69.4*(2.3)	*90.4*(1.1)	**97.3**(2.0)	*77.8*	*17.8*	*0.002*
Lin LS-SVM$_F$ (MOC)	*86.6*(1.7)	*42.7*(2.0)	*69.8*(1.2)	*77.0*(3.8)	*35.1*(2.6)	*54.1*(1.3)	*58.2*(0.9)	*69.1*(2.0)	*55.7*(1.3)	*85.5*(5.1)	*63.4*	*22.4*	*0.002*
Pol LS-SVM (MOC)	*94.0*(0.8)	53.5(2.3)	*87.2*(2.6)	**96.4**(3.7)	*70.9*(1.5)	*94.7*(0.2)	*95.0*(0.8)	*81.8*(1.2)	*99.6*(0.3)	**97.8**(1.9)	87.1	**9.8**	**0.109**
Pol LS-SVM$_F$ (MOC)	*93.2*(1.9)	*47.4*(1.6)	*86.2*(3.2)	96.0(3.7)	*67.7*(0.8)	*69.9*(2.8)	*87.2*(0.9)	*81.9*(1.3)	*96.1*(0.7)	*92.2*(3.2)	*81.8*	*15.7*	*0.002*
RBF LS-SVM (1vs1)	*94.2*(2.2)	**55.7**(2.2)	**96.5**(0.5)	**97.6**(2.3)	**74.1**(1.3)	*96.8*(0.3)	94.8(2.5)	83.6(1.3)	*99.3*(0.4)	**98.2**(1.8)	**89.1**	**5.9**	**1.000**
RBF LS-SVM$_F$ (1vs1)	*71.4*(15.5)	*42.7*(3.7)	*46.2*(6.5)	*79.8*(10.3)	*58.9*(8.5)	*92.6*(0.2)	*30.7*(2.4)	*24.9*(2.5)	*97.3*(1.7)	*67.3*(14.6)	*61.2*	*22.3*	*0.002*
Lin LS-SVM (1vs1)	*87.8*(2.2)	*50.8*(2.4)	*93.4*(1.0)	**98.4**(1.8)	**74.5**(1.0)	*93.2*(0.3)	*95.4*(0.3)	*79.8*(2.1)	*97.6*(0.9)	**98.3**(2.5)	**86.9**	**9.7**	**0.754**
Lin LS-SVM$_F$ (1vs1)	*87.7*(1.8)	*49.6*(1.8)	*93.4*(0.9)	**98.6**(1.3)	**74.5**(1.0)	*74.9*(0.8)	*95.3*(0.3)	*79.8*(2.2)	*98.2*(0.6)	**97.7**(1.8)	85.0	11.1	**0.344**
Pol LS-SVM (1vs1)	*95.4*(1.0)	*53.2*(2.2)	*95.2*(0.6)	96.8(2.3)	72.8(2.6)	88.8(14.6)	**96.0**(2.1)	82.8(1.8)	*99.0*(0.4)	**99.0**(1.4)	**87.9**	**8.9**	**0.344**
Pol LS-SVM$_F$ (1vs1)	*56.5*(16.7)	*41.8*(1.8)	*30.1*(3.8)	*71.4*(12.4)	*32.6*(10.9)	*92.6*(0.7)	95.8(1.7)	*20.3*(6.7)	*77.5*(4.9)	*82.3*(12.2)	*60.1*	*21.9*	0.021
RBF SVM (MOC)	**99.2**(0.5)	*51.0*(1.4)	*94.9*(0.9)	96.6(3.4)	*69.9*(1.0)	*96.6*(0.2)	*95.5*(0.4)	*77.6*(1.7)	**99.7**(0.1)	**97.8**(2.1)	**87.9**	**8.6**	**0.344**
Lin SVM (MOC)	*98.3*(1.2)	*45.8*(1.6)	*74.1*(1.4)	**95.0**(10.5)	*50.9*(3.2)	*92.5*(0.3)	*81.9*(0.3)	*70.3*(2.5)	*99.2*(0.2)	97.3(2.6)	80.5	16.1	0.021
RBF SVM (1vs1)	*98.3*(1.2)	**54.7**(2.4)	*96.0*(0.4)	97.0(3.0)	*64.6*(5.6)	*98.3*(0.3)	**97.2**(0.2)	83.8(1.6)	*99.6*(0.2)	**96.8**(5.7)	**88.6**	**6.5**	**1.000**
Lin SVM (1vs1)	*91.0*(2.3)	*50.8*(1.6)	*95.2*(0.7)	**98.0**(1.9)	**74.4**(1.2)	*97.1*(0.3)	*95.1*(0.3)	*78.1*(2.4)	*99.6*(0.2)	**98.3**(3.1)	**87.8**	**7.3**	**0.754**
LDA	*86.9*(2.1)	*51.8*(2.2)	*91.2*(1.1)	**98.6**(1.0)	73.7(0.8)	*93.7*(0.3)	*91.5*(0.5)	*77.4*(2.7)	*94.6*(1.2)	**98.7**(1.5)	85.8	11.0	**0.109**
QDA	*90.5*(1.1)	*50.6*(2.1)	*81.8*(9.6)	**98.2**(1.8)	**73.6**(1.1)	*93.4*(0.3)	*74.7*(0.7)	**84.8**(1.5)	*60.9*(9.5)	**99.2**(1.2)	**80.8**	**11.8**	**0.344**
Logit	*88.5*(2.0)	*51.6*(2.4)	*95.4*(0.6)	**97.0**(3.9)	**73.9**(1.0)	*95.8*(0.5)	*91.5*(0.5)	*78.3*(2.3)	**99.9**(0.1)	*95.0*(3.2)	*86.7*	**9.8**	0.021
C4.5	*66.0*(3.6)	*50.9*(1.7)	96.1(0.7)	96.0(3.1)	73.6(1.3)	**99.7**(0.1)	*88.7*(0.3)	*71.1*(2.6)	**99.8**(0.1)	*87.0*(5.0)	82.9	**11.8**	**0.109**
oneR	*59.5*(3.1)	*43.2*(3.5)	*62.9*(2.4)	*95.2*(2.5)	*17.8*(0.8)	*96.3*(0.5)	*32.9*(1.1)	*52.9*(1.9)	*67.4*(1.1)	*76.2*(4.6)	*60.4*	*21.6*	*0.002*
IB1	*81.5*(2.7)	*43.3*(1.1)	**96.8**(0.6)	*95.6*(3.6)	**74.0**(1.3)	*92.2*(0.4)	*97.0*(0.2)	*70.1*(2.9)	*99.7*(0.1)	*95.2*(2.0)	**84.5**	**12.9**	**0.344**
IB10	*83.6*(2.3)	*44.3*(2.4)	*94.3*(0.7)	*97.2*(1.9)	**74.2**(1.3)	*93.7*(0.3)	*96.1*(0.3)	*67.1*(2.1)	*99.4*(0.1)	*96.2*(1.9)	84.6	**12.4**	**0.344**
NB_k	*89.9*(2.0)	*51.2*(2.3)	*84.9*(1.4)	**97.0**(2.5)	**74.0**(1.2)	*96.4*(0.2)	*79.3*(0.9)	*60.0*(2.3)	*99.5*(0.1)	**97.7**(1.6)	*83.0*	12.2	0.021
NB_n	*89.9*(2.0)	*48.9*(1.8)	*80.1*(1.0)	**97.2**(2.7)	**74.0**(1.2)	*95.5*(0.4)	*78.2*(0.6)	*44.9*(2.8)	*99.5*(0.1)	97.5(1.8)	*80.6*	13.6	0.021
Maj. Rule	*48.7*(2.3)	*43.2*(1.8)	*15.5*(0.6)	*38.6*(2.8)	*11.4*(0.0)	*92.5*(0.3)	*16.8*(0.4)	*27.7*(1.5)	*34.2*(0.8)	*39.7*(2.8)	*36.8*	*24.8*	*0.002*

Table 3.4 *Comparison of the 10 times randomized* **test set** *performance of LS-SVM and LS-SVM$_F$ (linear, polynomial and Radial Basis Function kernel) with the performance of LDA, QDA, Logit, C4.5, oneR, IB1, IB10, NB_k, NB_n and the Majority Rule classifier on 10 binary domains. The Average Accuracy (AA), Average Rank (AR) and Probability of equal medians using the Sign Test (P_{ST}) taken over all domains are reported in the last three columns. Best performances are underlined and denoted in bold face, performances not significantly different at the 5% level are denoted in bold face, performances significantly different at the 1% level are emphasized. Good results are obtained by LS-SVM 1vs1 with RBF kernel.*

3.5 Least Squares Support Vector Machines for function estimation

So far we have discussed LS-SVMs for classification together with its link to Fisher discriminant analysis in high dimensional feature spaces. Let us now derive the LS-SVM formulation for the case of nonlinear function estimation. The derivation is similar to the LS-SVM classifier case which can also be interpreted as a regression problem with targets $+1$ and -1 with squared loss function taken on these targets.

Consider first a model in the primal weight space of the following form:

$$y(x) = w^T \varphi(x) + b \tag{3.28}$$

where $x \in \mathbb{R}^n, y \in \mathbb{R}$ and $\varphi(\cdot) : \mathbb{R}^n \to \mathbb{R}^{n_h}$ is the mapping to the high dimensional and potentially infinite dimensional feature space. Given a training set $\{x_k, y_k\}_{k=1}^N$ we can formulate then the following optimization problem in the primal weight space

$$\left[\boxed{\text{P}}: \quad \min_{w,b,e} J_{\text{P}}(w,e) = \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{k=1}^{N} e_k^2 \right.$$
$$\left. \text{such that} \quad y_k = w^T \varphi(x_k) + b + e_k, \quad k = 1, ..., N. \right] \tag{3.29}$$

Note that this is in fact nothing else but a ridge regression [98] cost function formulated in the feature space. However, one should be aware that when w becomes infinite dimensional, one cannot solve this primal problem. Therefore, let us proceed by constructing the Lagrangian and derive the dual problem. This problem was studied in [198] without the use of a bias term.

One constructs the Lagrangian

$$\mathcal{L}(w, b, e; \alpha) = \mathcal{J}_{\text{P}}(w, e) - \sum_{k=1}^{N} \alpha_k \{ w^T \varphi(x_k) + b + e_k - y_k \} \tag{3.30}$$

where α_k are Lagrange multipliers. The conditions for optimality are given

by

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 \;\rightarrow\; w = \sum_{k=1}^{N} \alpha_k \varphi(x_k) \\ \frac{\partial \mathcal{L}}{\partial b} = 0 \;\rightarrow\; \sum_{k=1}^{N} \alpha_k = 0 \\ \frac{\partial \mathcal{L}}{\partial e_k} = 0 \;\rightarrow\; \alpha_k = \gamma e_k, \qquad k = 1, ..., N \\ \frac{\partial \mathcal{L}}{\partial \alpha_k} = 0 \;\rightarrow\; w^T \varphi(x_k) + b + e_k - y_k = 0, \quad k = 1, ..., N. \end{cases} \tag{3.31}$$

After elimination of the variables w and e one gets the following solution

$$\left[\begin{array}{l} \boxed{\text{D}}: \quad \text{solve in } \alpha, b: \\ \\ \left[\begin{array}{c|c} 0 & 1_v^T \\ \hline 1_v & \Omega + I/\gamma \end{array} \right] \left[\begin{array}{c} b \\ \hline \alpha \end{array} \right] = \left[\begin{array}{c} 0 \\ \hline y \end{array} \right] \end{array} \right] \tag{3.32}$$

where $y = [y_1; ...; y_N]$, $1_v = [1; ...; 1]$ and $\alpha = [\alpha_1; ...; \alpha_N]$. The kernel trick is applied here as follows

$$\begin{aligned} \Omega_{kl} &= \varphi(x_k)^T \varphi(x_l) \\ &= K(x_k, x_l) \qquad k, l = 1, ..., N. \end{aligned} \tag{3.33}$$

The resulting LS-SVM model for function estimation becomes then

$$y(x) = \sum_{k=1}^{N} \alpha_k K(x, x_k) + b \tag{3.34}$$

where α_k, b are the solution to the linear system (3.32). The large scale algorithm as outlined for LS-SVM classifiers can be applied in a similar way to the function estimation case in order to handle large data sets. Note that in the case of RBF kernels, one has only two additional tuning parameters (γ, σ), which is less than for standard SVMs.

In Fig. 3.6 an illustration is given on a problem of time-series prediction of the Santa Fe chaotic laser data set [288]. An LS-SVM model with RBF kernel was taken to train an NARX model of the form $\hat{y}_{k+1} = f(y_k, y_{k-1}, ..., y_{k-q})$ with $q = 50$. Note that the value of q determines the number of inputs of the LS-SVM and that the size of the linear system to be solved is independent of the choice of q (while for MLPs the number of interconnections weights would grow with the value of q). The (γ, σ)

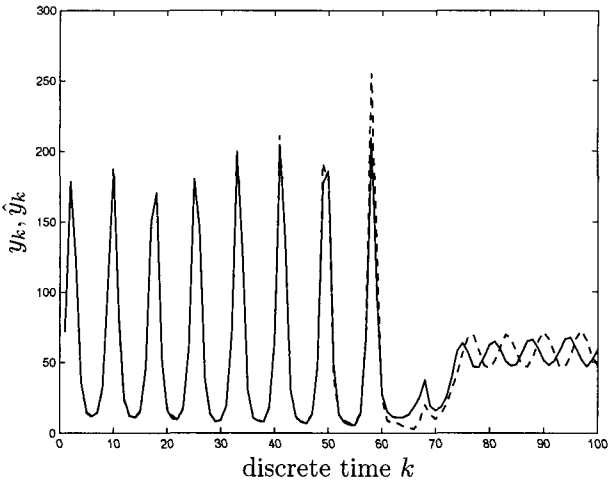

Fig. 3.6 *Time-series prediction by an LS-SVM with RBF kernel for the Santa Fe chaotic laser data. The figure shows the iterative prediction of an NARX LS-SVM model over a time horizon of 100 points, that was trained on the previous 1000 given data points; true data y_k (solid line), iterative prediction $\hat{y}_k$ (dashed line).*

parameters were tuned by 10-fold cross-validation. More sophisticated hyperparameter tuning methods may be further applied at this point, such as special cross-validation techniques for time-series.

3.6 Links with regularization networks and Gaussian processes

The LS-SVM solution (3.32) for the nonlinear function estimation case that we derived is closely related to results on regularization networks, reproducing kernel Hilbert spaces (RKHS), Gaussian processes, kriging and kernel ridge regression [1; 73; 153; 183; 286; 291]. However, a bias term b is included in the formulation as in the standard SVM case. Except for regularization networks, inclusion of a bias term is often not considered or treated in a different manner by including a constant in the kernel. The emphasis in LS-SVMs is on primal-dual optimization theory and neural networks interpretations (Fig. 3.7). A recent overview on the links between regularization networks and SVMs has been presented by Evgeniou, Pontil & Poggio in [73].

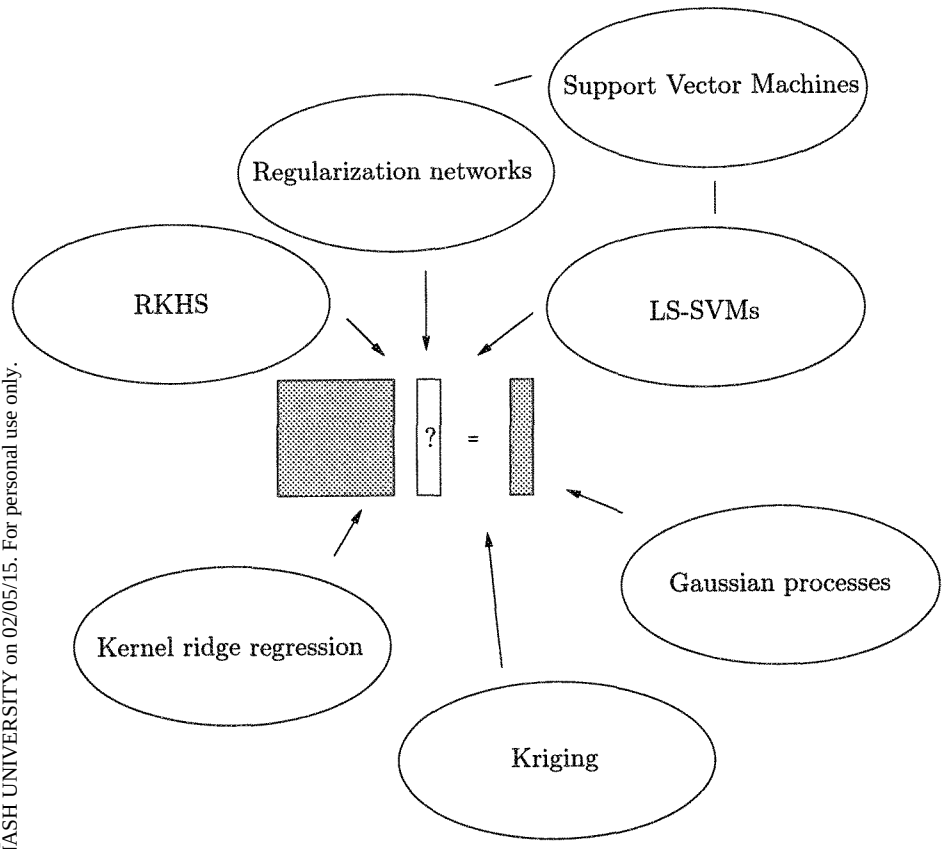

Fig. 3.7 *LS-SVMs are closely related to (and in certain cases lead to equivalent solutions as) regularization networks, estimation in RKHS, Gaussian processes, kriging and kernel ridge regression. The emphasis in LS-SVMs is mainly on primal-dual insights from optimization theory and neural networks interpretations and aims at bridging many different areas in an interdisciplinary manner.*

3.6.1 *RKHS and reproducing property*

According to Kailath in [129] the theory of Reproducing Kernel Hilbert Spaces (RKHS) was first applied to detection and estimation problems by Parzen and first studied in 1910-1920 by Moore [168], in connection with a general theory of integral equations. Krein used them in his fundamental studies on the extension of positive definite functions. It was Aronszajn [13] who made a systematic abstract development of the theory in the 1940's,

though it was discovered that many results were independently obtained in the USSR by Povzner [185]. Loeve [145] was the first to note that RKHS could be used to provide a representation for second-order random processes. On the other hand it was the work of Parzen that brought the RKHS to the fore in statistical problems. In particular in [177] the RKHS was used to clarify some relationships between time series, control theory and approximation theory. For an early review paper on links with linear filtering theory, see e.g. Kailath in [130].

A Reproducing Kernel Hilbert Space (RKHS) is a Hilbert space $\mathcal{H}$ of functions defined over a bounded domain $X \subset \mathbb{R}^n$ with the property that for each $x \in X$ the evaluation functionals $\mathcal{F}_x$ defined as $\mathcal{F}_x[f] = f(x)$ for all $f \in \mathcal{H}$ are linear bounded functionals. The boundedness means that there exists a $U = U_x \in \mathbb{R}^+$ such that $|\mathcal{F}_x[f]| = |f(x)| \leq U\|f\|$ for all f in the RKHS.

It can be proved that to every RKHS $\mathcal{H}$ there corresponds a unique positive definite function $K(x, z)$ of two variables in X (Moore-Aronszajn Theorem [13]), called the reproducing kernel of $\mathcal{H}$, that has the following reproducing property:

$$f(x) = \langle f(z), K(z, x)\rangle_{\mathcal{H}} \ , \ \forall f \in \mathcal{H} \tag{3.35}$$

where $\langle ., . \rangle_{\mathcal{H}}$ denotes the inner product in $\mathcal{H}$ and K is called a reproducing kernel for $\mathcal{H}$. The function K behaves in $\mathcal{H}$ as the delta function does in L_2. In particular one has

$$\langle K(x, \cdot), K(\cdot, z)\rangle_{\mathcal{H}} = K(x, z). \tag{3.36}$$

Consider now the following set of functions

$$f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x). \tag{3.37}$$

This Hilbert space is a RKHS because

$$\langle f(z), K(z, x)\rangle_{\mathcal{H}} = \sum_{i=1}^{\infty} \frac{c_i \lambda_i \phi_i(x)}{\lambda_i} = f(x) \tag{3.38}$$

where the scalar product between two functions $f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x)$ and

$g(x) = \sum_{i=1}^{\infty} d_i \phi_i(x)$ is defined as

$$\langle f(x), g(x) \rangle_{\mathcal{H}} = \langle \sum_{i=1}^{\infty} c_i \phi_i(x), \sum_{i=1}^{\infty} d_i \phi_i(x) \rangle_{\mathcal{H}} = \sum_{i=1}^{\infty} \frac{c_i d_i}{\lambda_i} \tag{3.39}$$

with kernel

$$K(x, z) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(z) \tag{3.40}$$

where a sequence of positive numbers λ_i and linearly independent basis functions $\phi_i(x)$ are assumed and the series converges. As an orthogonal basis one has the eigenfunctions $\{\phi_i(x)\}_{i=1}^{\infty}$ of the integral operator Γ_K : $L_2(X) \rightarrow L_2(X)$ with $(\Gamma_K f)(x) = \int K(x, z) f(z) dz$, $\forall f \in L_2(X)$ and $\lambda_i > 0$ are the corresponding eigenvalues.

One can verify then that this kernel is positive definite. Note that when working with complex functions $\phi_i(x)$ one has $K(x, z) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i^*(z)$, where $\phi_i^*(z)$ denotes the complex conjugate of $\phi_i(z)$. In relation to the LS-SVM formulations when having a function representation $f(x) = w^T \varphi(x)$ in the primal space and kernel trick $K(z, x) = \varphi(z)^T \varphi(x)$ with $K(\cdot, \cdot)$ symmetric and positive definite, the reproducing property means

$$\langle f(z), K(z, x) \rangle_{\mathcal{H}} = \langle w^T \varphi(z), \varphi(z)^T \varphi(x) \rangle_{\mathcal{H}} = w^T \varphi(x) = f(x). \tag{3.41}$$

For functions $f(x) = w^T \varphi(x)$, $g(x) = v^T \varphi(x)$ we have $\langle f(x), g(x) \rangle_{\mathcal{H}} = \langle w^T \varphi(x), v^T \varphi(x) \rangle_{\mathcal{H}} = w^T v$. The norm is

$$\|f\|_K^2 = \langle f(x), f(x) \rangle_{\mathcal{H}} = \langle w^T \varphi(x), w^T \varphi(x) \rangle_{\mathcal{H}} = w^T w. \tag{3.42}$$

Note that for a Gaussian RBF kernel the RKHS is infinite dimensional, while for a polynomial kernel with a certain degree the RKHS is finite. The RBF kernel is translation invariant with $K(x, z) = K(x - z)$ and radial with $K(x, z) = K(\|x - z\|)$. A radial positive definite K defines a RKHS in which the basis functions $\{\phi_s(x)\}_{s=0}^{\infty}$ in the high dimensional feature space are Fourier components [73]:

$$K(x, z) = \sum_{s=0}^{\infty} \lambda_s \phi_s(x) \phi_s(z) = \sum_{s=0}^{\infty} \lambda_s \exp(j 2\pi s x) \exp(-j 2\pi s z) \tag{3.43}$$

with $j = \sqrt{-1}$. Hence any positive definite radial kernel defines a RKHS

with a scalar product equal to

$$\langle f(x), g(x) \rangle_{\mathcal{H}} = \sum_{s=0}^{\infty} \frac{\tilde{f}(s)\tilde{g}^*(s)}{\lambda_s} \tag{3.44}$$

where $\tilde{f}$ denotes the Fourier transform of f. The RKHS then becomes a subspace of $L_2([0,1]^n)$ of the functions such that

$$\|f\|_K^2 = \sum_{s=0}^{\infty} \frac{|\tilde{f}(s)|^2}{\lambda_s} < \infty. \tag{3.45}$$

Functionals of this form are known to be smoothness functionals.

3.6.2 *Representer theorem and regularization networks*

In order to see the link now with LS-SVM regression, let us consider the following variational problem as explained by Girosi in [95]:

$$\min_{f \in \mathcal{H}} H[f] = \gamma \sum_{k=1}^{N} L\left(y_k - f(x_k)\right) + \frac{1}{2}\|f\|_K^2. \tag{3.46}$$

Assume that $\mathcal{H}$ is a RKHS with kernel K. This is equivalent to assuming that the functions in $\mathcal{H}$ have a unique expansion of the form $f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x)$ with norm $\|f\|_K^2 = \sum_{i=1}^{\infty} c_i^2/\lambda_i$. The solution to the problem follows from $\partial H[f]/\partial c_i = 0$ which gives

$$\gamma \sum_{k=1}^{N} L'\left(y_k - f(x_k)\right) \phi_i(x_k) + \frac{c_i}{\lambda_i} = 0. \tag{3.47}$$

One defines then new unknowns $a_k = \gamma L'(y_k - f(x_k))$. Expressing the coefficients c_i in terms of a_k one gets $c_i = \lambda_i \sum_{k=1}^{N} a_k \phi_i(x_k)$ such that the solution to the variational problem becomes

$$f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x) = \sum_{i=1}^{\infty} \sum_{k=1}^{N} a_k \lambda_i \phi_i(x_k) \phi_i(x) = \sum_{k=1}^{N} a_k K(x, x_k). \tag{3.48}$$

Hence, independently of the form of L, the solution to the problem is always a linear superposition of kernel functions [95]. This is also essentially what the representer Theorem according to Kimeldorf and Wahba states [133].

The coefficients a_k follow as the solution to the following set of equations:

$$a_k = \gamma L'(y_k - \sum_{l=1}^{N} a_l K(x_k, x_l)) \ , \ k = 1, ..., N. \tag{3.49}$$

When taking the least squares loss function $L(e) = \frac{1}{2}e^2$ this becomes the linear system

$$[\Omega + I/\gamma]a = y \tag{3.50}$$

with $y = [y_1; ...; y_N]$ and $\Omega_{kl} = K(x_k, x_l)$ for $k, l = 1, ..., N$ the elements of the kernel matrix. This linear system corresponds to the regularization network solution as derived by Poggio & Girosi [183]. As explained in [73] one can also handle a bias term in this framework. Furthermore one should also note that both parametric and semiparametric versions of the representer theorem exist [133; 206].

In a support vector machines primal-dual optimization formulation context the problem (3.46) corresponds to the following

$$\left[\begin{array}{lll} \boxed{\text{P}}: & \min\limits_{w,b,e} J_{\text{P}}(w, e) = & \frac{1}{2} w^T w + \gamma \sum\limits_{k=1}^{N} L(e_k) \\ & \text{such that} & y_k = w^T \varphi(x_k) + b + e_k, \ \ k = 1, ..., N \end{array} \right] \tag{3.51}$$

where $L(e)$ is a general and differentiable cost function. From the Lagrangian $\mathcal{L}(w, b, e; \alpha) = \mathcal{J}_{\text{P}}(w, e) - \sum_{k=1}^{N} \alpha_k \{w^T \varphi(x_k) + b + e_k - y_k\}$ with Lagrange multipliers α_k, one has the following conditions for optimality:

$$\left\{ \begin{array}{lll} \frac{\partial \mathcal{L}}{\partial w} = 0 & \rightarrow & w = \sum\limits_{k=1}^{N} \alpha_k \varphi(x_k) \\ \frac{\partial \mathcal{L}}{\partial b} = 0 & \rightarrow & \sum\limits_{k=1}^{N} \alpha_k = 0 \\ \frac{\partial \mathcal{L}}{\partial e_k} = 0 & \rightarrow & \alpha_k = \gamma L'(e_k), \qquad\qquad k = 1, ..., N \\ \frac{\partial \mathcal{L}}{\partial \alpha_k} = 0 & \rightarrow & w^T \varphi(x_k) + b + e_k - y_k = 0, \quad k = 1, ..., N. \end{array} \right. \tag{3.52}$$

Note that $\alpha_k = \gamma L'(e_k)$ is now part of the conditions for optimality, while in the RKHS context this is rather by definition. After elimination of the variables w and e and application of the kernel trick $K(x_k, x_l) = \varphi(x_k)^T \varphi(x_l)$

one gets

$$\left[\begin{array}{l} \boxed{\mathrm{D}}: \quad \text{solve in } \alpha, b: \\ \\ \qquad \alpha_k = \gamma L'(y_k - \sum_{l=1}^{N} \alpha_l K(x_l, x_k) + b) \ , \quad k = 1, ..., N \end{array}\right] \tag{3.53}$$

which is a set of nonlinear equations to be solved in α, b. Alternatively, one may also eliminate α instead of e and solve the nonlinear equations in e. The resulting dual representation of the model is $y(x) = \sum_{k=1}^{N} \alpha_k K(x, x_k) + b$. For the case $L(e) = \frac{1}{2}e^2$ one has the linear system (3.32).

3.6.3 *Gaussian processes*

Link between LS-SVM regression and Gaussian processes

In the RKHS context it is also known that there is a link between the solution to variational problems in RKHS and Gaussian processes, according to Kimeldorf & Wahba [133; 202; 286]. To every RKHS $\mathcal{H}_K$ there corresponds a zero mean Gaussian stochastic process with covariance $K(s,t)$ and there is a one-to-one inner product preserving map between the Hilbert space spanned by the stochastic process and $\mathcal{H}_K$ where the random variable $X(t)$ corresponds to the representer $K_t \in \mathcal{H}_K$.

Gaussian processes have also been studied in the neural networks area within the Bayesian learning context. According to MacKay, one has the following methodology. Consider a given set of N data points $\{x_k, y_k\}_{k=1}^{N}$ and denote $y_{1\_N} = [y_1; ...; y_N] \in \mathbb{R}^N$. Given the input data one can construct a covariance matrix C with $C_{kl} = C(x_k, x_l)$ the kl-entry of the matrix.

Having formed the covariance matrix C the task is then to infer y_{N+1} given the observed vector y_{1_N}. From Bayes rule one has the joint density $P(y_{N+1}, y_{1\_N}) = P(y_{N+1}|y_{1\_N})P(y_{1\_N})$ or

$$P(y_{N+1}|y_{1\_N}) = \frac{P(y_{N+1}, y_{1\_N})}{P(y_{1\_N})} \tag{3.54}$$

with joint density and conditional density considered to be Gaussian. The covariance matrix $C_{N+1} \in \mathbb{R}^{(N+1)\times(N+1)}$ can then be written as a function of $C_N \in \mathbb{R}^{N\times N}$ as follows:

$$C_{N+1} = \left[\begin{array}{cc} C_N & \theta(x_{N+1}) \\ \theta(x_{N+1})^T & \nu \end{array}\right] \tag{3.55}$$

where $\theta(x_{N+1}) = [K(x_{N+1}, x_1); ...; K(x_{N+1}, x_N)]$. One has

$$P(y_{N+1}, y_{1\_N}) \propto \exp\left(-\frac{1}{2}\begin{bmatrix} y_{1\_N} & y_{N+1} \end{bmatrix} C_{N+1}^{-1} \begin{bmatrix} y_{1\_N} \\ y_{N+1} \end{bmatrix}\right) \tag{3.56}$$

with the following expressions for the inverse matrix

$$C_{N+1}^{-1} = \begin{bmatrix} M & m \\ m^T & \varrho \end{bmatrix} \tag{3.57}$$

and

$$\begin{aligned} \varrho &= \left(\nu - \theta^T C_N^{-1} \theta\right)^{-1} \\ m &= -\varrho C_N^{-1} \theta \\ M &= C_N^{-1} + \frac{1}{\varrho} m m^T. \end{aligned} \tag{3.58}$$

This gives the posterior distribution

$$P(y_{N+1}|y_{1\_N}) \propto \exp\left(-\frac{1}{2}\frac{(y_{N+1} - \hat{y}_{N+1})^2}{\sigma^2_{\hat{y}_{N+1}}}\right) \tag{3.59}$$

with

$$\begin{cases} \hat{y}_{N+1} &= \theta^T C_N^{-1} y_{1\_N} \\ \sigma^2_{\hat{y}_{N+1}} &= \nu - \theta^T C_N^{-1} \theta. \end{cases} \tag{3.60}$$

The predictive mean at a new point x_{N+1} is given by $\hat{y}_{N+1}$ and $\sigma_{\hat{y}_{N+1}}$ which defines the error bar on the prediction for this new point. Note that one only needs to invert C_N not C_{N+1}.

The link between this Gaussian process formulation and LS-SVM regression can be understood as follows. Take an LS-SVM model with zero bias term $b = 0$. Instead of denoting a new point in a sequential manner, let us denote it as x instead of x_{N+1}. The LS-SVM model can be written as

$$\begin{aligned} \hat{y}(x) &= \sum_{k=1}^{N} \alpha_k K(x, x_k) \\ &= \theta(x)^T \alpha \end{aligned} \tag{3.61}$$

where $\alpha = [\alpha_1; ...; \alpha_N]$ and $\theta(x) = [K(x, x_1); ...; K(x, x_N)]$. Let us define $C(x_k, x_l) = K(x_k, x_l) + \delta_{kl}/\gamma$. In the zero bias term case the LS-SVM

system is

$$[\Omega + I/\gamma]\alpha = y \tag{3.62}$$

with solution $\alpha = [\Omega + I/\gamma]^{-1}y$, where $y = y_{1\_N}$. The LS-SVM model becomes

$$\hat{y}(x) = \theta(x)^T[\Omega + I/\gamma]^{-1}y \tag{3.63}$$

which is the same as (3.60) if one takes $C_N = \Omega + I/\gamma$ where $\Omega_{kl} = K(x_k, x_l)$ for $k, l = 1, ..., N$ in order to have a compatible definition of θ with respect to C_N in (3.55).

Stationary and non-stationary random fields, spectral representation and kriging

The predictions produced by Gaussian processes depend entirely on the covariance matrix C. As for positive definite kernels in SVMs, one has several possible choices for the covariance function and the two are closely related. The Gaussian processes framework is also related to methods of *kriging* (named after D.G. Krige a South African mining engineer), developed within the geostatistics field [2; 50; 137; 153]. For an overview with respect to Gaussian random fields, see e.g. [1; 50]. Random field models are basically specified by expectations and covariances. One makes a distinction between stationary and non-stationary covariance functions. *Stationarity in the wide sense* for a random field means a constant expectation and covariance function $C(t, s) = C(t - s)$ with $\tau = t - s$ and typically $t, s \in \mathbb{R}^n$ [70]. The covariance function $C(t, s)$ is defined as $\mathrm{Cov}(X_t, X_s)$ with $X_t = X(t, \omega)$ a random field on $\mathbb{R}^n$ with $t \in \mathbb{R}^n$, for a given probability space and parameter set T and $X(t, \omega)$ a real valued and measurable function of ω for every $t \in T$. The correlation function is defined as $\rho(t, s) = \mathrm{Corr}(X_t, X_s) = C(t, s)/(\sigma(t)\sigma(s))$ with variance $\sigma^2(t) = C(t, t) = \mathrm{Var}(X_t, X_t)$. Any stationary covariance function has a constant variance such that $C(\tau) = \sigma^2\rho(\tau)$. A stationary random field is called *isotropic* if the covariance function depends on the distance only $C(t, s) = C(\|\tau\|)$ and anisotropic if a weighted norm is used $C(t, s) = C(\|\tau\|_W)$ with matrix W where $\|\tau\|_W = \sqrt{\tau^T W \tau}$. A correlation function is called *separable* if $\rho(\tau) = \rho_1(\tau_1)...\rho_n(\tau_n)$. For stationary correlation functions one has a spectral representation [91].

According to the multidimensional Bochner's Theorem a real function $r(\tau)$ on $\mathbb{R}^n$ is positive definite if and only if it can be represented in the form $r(\tau) = \int_{\mathbb{R}^n} \exp(j\tau k) d^n F(k)$ where $F(\cdot)$ is a non-negative bounded measure. Bochner's theorem says that all positive definite functions have a unique spectral representation. The integral is a Fourier transform of the non-negative F. Because the class of positive definite functions coincides with the class of covariance functions the Wiener-Khintchine theorem can be derived, which states that a real function $\rho(\tau)$ on $\mathbb{R}^n$ is a correlation function if and only if it can be represented in the form

$$\rho(\tau) = \int_{\mathbb{R}^n} \exp(j\tau k) d^n F(k) \tag{3.64}$$

where the function $F(k)$ on $\mathbb{R}^n$ has the properties of an n-dimensional distribution function. According to Matérn one can link correlation functions to characteristic functions. When F is continuous the following spectral density function $f(k) = \partial^n F(k)/\partial k_1 ... \partial k_n$ exists. One can then write $\rho(\tau) = \int_{\mathbb{R}^n} \exp(j\tau k) f(k) d^n k$. As a result a function $f(k)$ in $\mathbb{R}^n$ is the spectral density function of a stationary correlation function on $\mathbb{R}^n$ if and only if $f(k) \geq 0$ and $\int_{\mathbb{R}^n} f(k) d^n k = 1$, i.e. $f(k)$ has the properties of an n-dimensional probability density. The spectral density function is obtained from the correlation function as

$$f(k) = (2\pi)^{-n} \int_{\mathbb{R}^n} \exp(-j\tau k) \rho(\tau) d^n \tau. \tag{3.65}$$

This gives an explicit method for verifying positive definiteness of a stationary correlation function on $\mathbb{R}^n$ by evaluating the spectral density $f(k)$ and check whether it is non-negative for all $k \in \mathbb{R}^n$.

Stationarity and in particular isotropy impose strong restrictions on the possible definition of a Gaussian random field. However, from stationary random fields it is simple to obtain a wide range of non-stationary random fields. For example, consider an isotropic Gaussian random field X_t with properties

$$\begin{aligned} \mathcal{E}[X_t] &= 0 \\ \mathrm{Cov}(X_t, X_s) &= \rho(\tau) \end{aligned} \tag{3.66}$$

with ρ an isotropic correlation function. One can then consider the Gaus-

sian random field

$$Z_t = \sum_i a_i f_i(t) + X_t \tag{3.67}$$

where a_i are Gaussian random variables independent of X_t and $f_i(t)$ are real valued functions. The expectation and covariance function are

$$\begin{aligned} \mathcal{E}[Z_t] &= \sum_i \mathcal{E}[a_i]\, f_i(t) \\ \mathrm{Cov}(Z_t, Z_s) &= \sum_{i,j} \mathrm{Cov}(a_i, a_j)\, f_i(t) f_j(s) + \rho(\tau). \end{aligned} \tag{3.68}$$

Both the expectation and the covariance function have become non-stationary in this case. Here a general Gaussian random function Z_t is modelled in terms of the stationary random field X_t. These insights are used in the kriging method, where one estimates the predictive mean and error bars given a set of observations, resulting into a linear system to be solved.

Automatic relevance determination in Gaussian processes

An interesting choice for C as discussed by MacKay is

$$C(x, z) = \theta_1 \exp\left(-\frac{1}{2}\sum_{i=1}^{n} \frac{(x_i - z_i)^2}{\sigma_i^2}\right) + \theta_2 \tag{3.69}$$

where $x, z \in \mathbb{R}^n$ and x_i, z_i denote the i-th component of these vectors. This can be used for Automatic Relevance Determination (ARD), where σ_i are considered then as length scale parameters to be determined. A very large length scale σ_i means that the output is expected to behave as a constant with respect to the i-th input, and hence would be irrelevant to the model. An extension of this to a non-stationary covariance function is to take spatially varying length scales and work with parameterized functions $\sigma_i(x)$. The following covariance function

$$C(x, z) = \theta_1 \prod_{i=1}^{n} \left(\frac{2\sigma_i(x)\sigma_i(z)}{\sigma_i^2(x) + \sigma_i^2(z)}\right)^{1/2} \exp\left(-\sum_{i=1}^{n} \frac{(x_i - z_i)^2}{\sigma_i^2(x) + \sigma_i^2(z)}\right) \tag{3.70}$$

is positive definite.

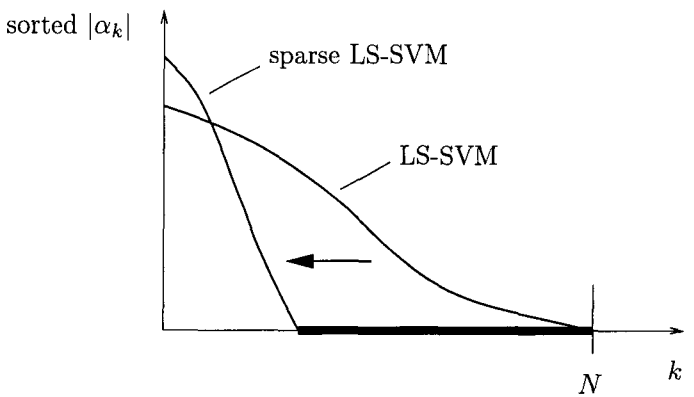

Fig. 3.8 *A simple way for imposing sparseness to LS-SVM models is to apply pruning techniques as known in the neural networks literature.*

3.7 Sparseness by pruning

We have discussed the links between LS-SVM classifiers and a kernel version of Fisher discriminant analysis together with links between LS-SVM regression and regularization networks, estimation in RKHS and Gaussian processes.

One drawback of LS-SVMs in comparison with standard SVMs is the lack of sparseness in the solution vector which is clear from the fact that $\alpha_k = \gamma e_k$. However, we will illustrate here that there are several possible ways to sparsify the LS-SVM system. A simple way is to apply insight of *pruning* from the neural networks literature. For MLPs it is well known that by starting from a huge network and deleting interconnection weights which are less relevant one can improve the generalization performance [23], e.g. by applying optimal brain damage (LeCun in [140]) or optimal brain surgeon (Hassibi & Stork in [106]).

One can proceed as follows. From the linear system one finds the vector of support values. One may apply the simple heuristic that data corresponding to small $|\alpha_k|$ values are less relevant for the construction of the model, in analogy with standard SVMs where zero α_k values do not contribute to the model. One works then in several steps where in each step typically 5% of the data with the smallest support values are removed. One gradually prunes the support value spectrum (Fig. 3.8). In each of these steps one re-estimates the linear system.

Simple LS-SVM pruning algorithm:

(1) Train an LS-SVM based on N points.

(2) Remove a small amount of points (e.g. 5% of the set) with smallest values in the sorted $|\alpha_k|$ spectrum.

(3) Re-train the LS-SVM based on the reduced training set.

(4) Go to (2), unless the user-defined performance index degrades. If the performance becomes worse, one checks whether an additional modification of (γ, σ) (in the RBF kernel case) might improve the performance and modifies the tuning parameters.

This procedure is applicable both to classification and function estimation problems. Note that omitting points implicitly correspond to creating an ϵ-insensitive zone in the underlying cost function which leads to sparseness, which is clear from the condition for optimality $\alpha_k = \gamma L'(e_k)$ in (3.52), because omitting a point is equivalent to imposing a zero value for that point which should correspond to a zero derivative for $L'(e_k)$ around the origin.

The differences with this procedure and obtaining sparseness in standard SVMs are the following: in the LS-SVM pruning case the size of the system shrinks at every stage and one has the possibility to modify the tuning parameters at certain stages while in standard SVMs one has to iterate for the same (γ, σ) until the minimum is reached on the same size of the system. Also note that interior point algorithms can be applied in the case of a general convex cost function as explained in the previous Chapter. In each iteration step one also solves a reduced KKT system which has the same form as one single LS-SVM (see (2.82)).

The simple pruning algorithm shown above is certainly not the most optimal one. More sophisticated pruning techniques can be used which also take into account the kernel matrix. However, it is also easy to apply to larger data sets and shows that in relation to least squares methods there are also ways to impose sparseness and one can overcome the lack of sparseness.

In Figs. 3.9-3.10 some easy toy problems are given of the simple LS-

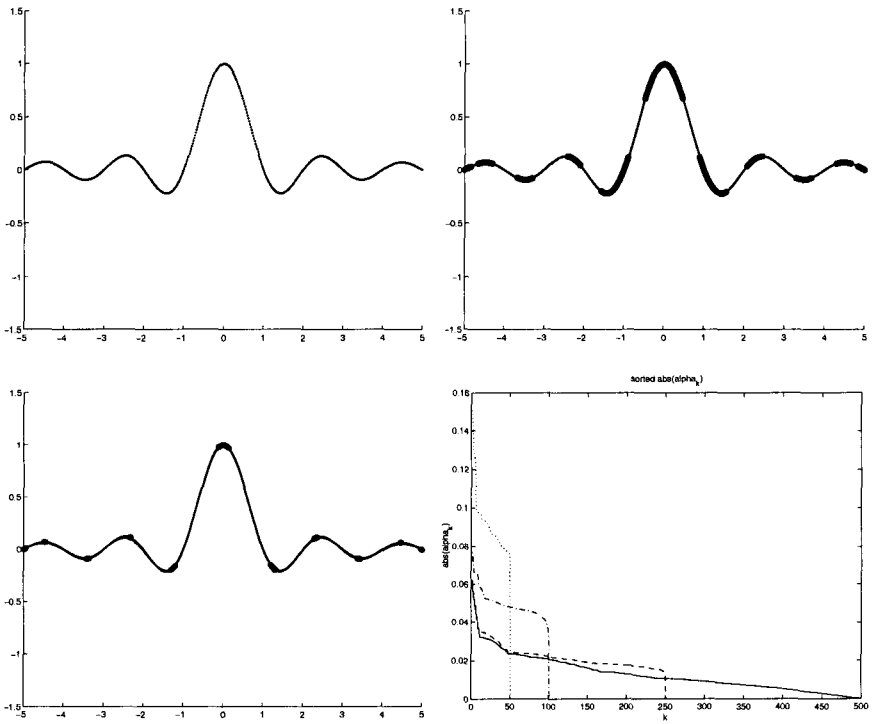

Fig. 3.9 *LS-SVM sparse approximation of a noiseless sinc function using an RBF kernel: 500 SV → 250 SV → 50 SV and sorted $|\alpha_k|$ values at the corresponding stages during the pruning process while omitting each time 5% of the data.*

SVM pruning algorithm for a noiseless and a noisy sinc function. The figures show a few snapshots during the pruning process while gradually removing 5 % of the training data and re-estimate. In these examples the values of γ and σ for an RBF kernel were kept constant. These values might be additionally optimized at certain stages in the pruning process, in order to better ensure a good generalization when re-estimation is done on the datasets that are becoming smaller. One also observes that higher degrees of sparseness are usually achieved in the noiseless case.

In Fig. 3.11 an illustration of the simple sparse approximation procedure is given for a binary classification problem where the training data have been generated by a Gaussian distribution for each class with the same covariance matrix. According to basic pattern recognition theory the

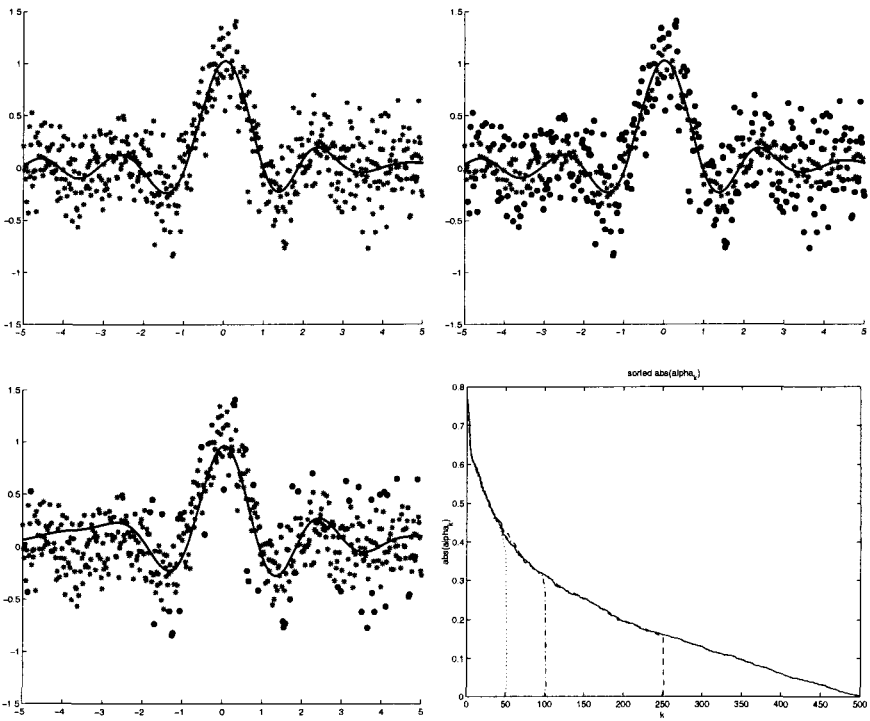

Fig. 3.10 *LS-SVM sparse approximation of a noisy sinc function with RBF kernel: 500 SV → 250 SV → 50 SV and sorted $|\alpha_k|$ values. Less support vectors can be pruned by means of this simple pruning method in the case of noisy data.*

optimal decision line is a straight line. The LS-SVM classifier is constructed here by means of an RBF kernel. The figure shows several stages of the pruning process. While standard SVMs retain support vectors that are close to decision boundary, in this LS-SVM pruning process one obtains support vectors that are located both close to and far from the decision boundary. This is because the sorting and ranking of the support values is done here on the basis of the absolute values of α. One might also prune the negative values and obtain qualitatively different support vectors with good test set ROC curve performance as observed in [146].

In [264] the sparse approximation method has been successfully applied to 5 binary and 5 multiclass UCI benchmark problems. In Table 3.5 these benchmarking results on UCI datasets are shown.

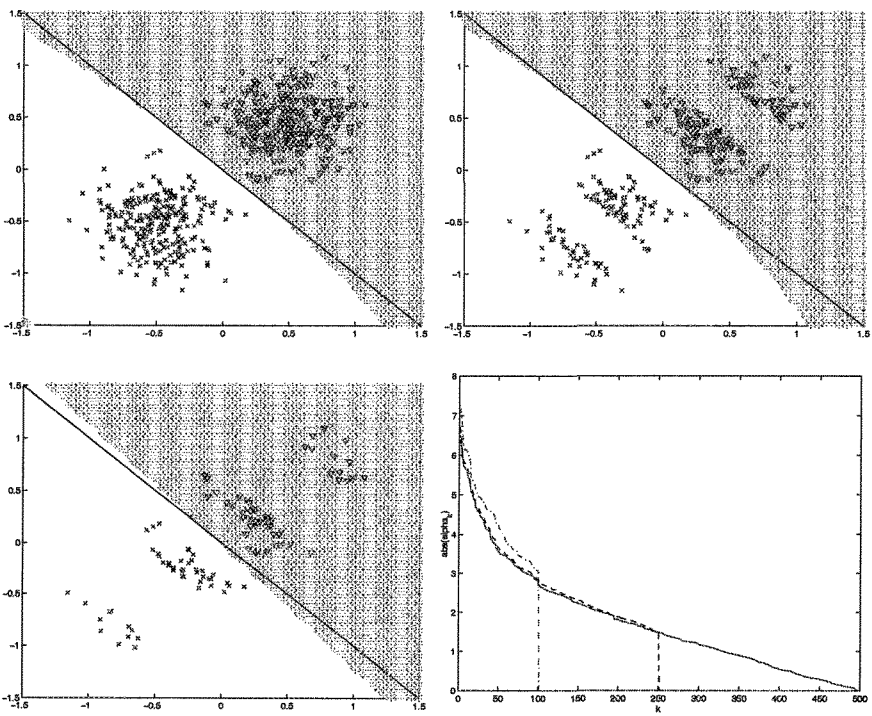

Fig. 3.11 *LS-SVM sparse approximation with RBF kernel illustrated on a classification problem: 500 SV → 250 SV → 50 SV and sorted $|\alpha_k|$ values.*

	acr	bld	gcr	hea	ion	bal	cmc	ims	iri	led	AA	AR	P_{ST}
N_{test}	230	115	334	90	117	209	491	770	50	1000			
n	14	6	20	13	33	4	9	18	4	7			
LS-SVM	**87.0**(2.1)	**70.2**(4.1)	**76.3**(1.4)	**84.7**(4.8)	**96.0**(2.1)	94.2(2.2)	**55.7**(2.2)	**96.5**(0.5)	**97.6**(2.3)	**74.1**(1.3)	**83.2**	**1.3**	**1.000**
LS-SVM$_{\text{SL5\%}}$	*85.6*(2.3)	67.3(4.1)	**76.2**(1.9)	83.6(4.0)	*89.8*(1.6)	**95.7**(1.2)	**54.4**(1.6)	*90.7*(1.7)	96.8(2.3)	**73.6**(2.5)	81.4	**2.2**	0.021
LS-SVM$_{\text{SL1\%}}$	*85.6*(2.3)	67.3(4.1)	**75.9**(1.5)	82.8(3.6)	*89.8*(1.6)	**95.5**(1.6)	**54.4**(1.6)	*90.7*(1.7)	**97.8**(2.6)	71.0(4.2)	81.1	2.5	**0.109**
$N_{CV}^{(1:L)}$	460	230	666	180	234	832	1964	9240	200	18000			
$N_{\text{SL5\%}}^{(1:L)}$	149	128	344	59	56	104	1418	860	96	4360			
$N_{\text{SL1\%}}^{(1:L)}$	149	128	327	57	56	60	1418	860	66	3640			
PPTE$_{\text{SL5\%}}$	68%	44%	48%	67%	76%	88%	28%	91%	67%	76%			
PPTE$_{\text{SL1\%}}$	68%	44%	51%	68%	76%	93%	28%	91%	52%	80%			

Table 3.5 *Sparse approximation of LS-SVMs with RBF kernel by gradually pruning the support value spectrum according to Van Gestel* et al. *Pruning is stopped when the cross-validation accuracy decreases significantly at the 5% and 1% significance level (SL 5% and SL 1%), respectively. The following randomized test set performances are reported: no pruning (LS-SVM), pruning till SL 5% (LS-SVM$_{\text{SL5\%}}$) and pruning till SL 1% (LS-SVM$_{\text{SL1\%}}$). The corresponding number of training examples N_{CV}, $N_{\text{SL5\%}}^{(1:L)}$ and $N_{\text{SL1\%}}^{(1:L)}$ are shown, together with Percentage of Pruned Training Examples PPTE$_{\text{SL1\%}}$ and PPTE$_{\text{SL5\%}}$, respectively.*