Support Vector Machines in Classification and Regression - An Introduction

This is an introductory chapter on the supervised (machine) learning from empirical data (i.e., examples, samples, measurements, records, patterns or observations) by applying support support vector machines (SVMs) a.k.a. kernel machines[1]. The parts on the semi-supervised and unsupervised learning are given later and being entirely different tasks they use entirely different math and approaches. This will be shown shortly. Thus, the book introduces the problems gradually in an order of loosing the information about the desired output label. After the supervised algorithms, the semi-supervised ones will be presented followed by the unsupervised learning methods in Chap. 6. The basic aim of this chapter is to give, as far as possible, a condensed (but systematic) presentation of a novel learning paradigm embodied in SVMs. Our focus will be on the constructive part of the SVMs' learning algorithms for both the classification (pattern recognition) and regression (function approximation) problems. Consequently, we will not go into all the subtleties and details of the statistical learning theory (SLT) and structural risk minimization (SRM) which are theoretical foundations for the learning algorithms presented below. The approach here seems more appropriate for the application oriented readers. The theoretically minded and interested reader may find an extensive presentation of both the SLT and SRM in [146, 144, 143, 32, 42, 81, 123]. Instead of diving into a theory, a quadratic programming based learning, leading to parsimonious SVMs, will be presented in a gentle way - starting with linear separable problems, through the classification tasks having overlapped classes but still a linear separation boundary, beyond the linearity assumptions to the nonlinear separation boundary, and finally to the linear and nonlinear regression problems. Here, the adjective 'parsimonious' denotes a SVM with a small number of support vectors ('hidden layer neurons'). The scarcity of the model results from a sophisticated, QP based, learning that matches the

[1] This introduction strictly follows and partly extends the School of Engineering of The University of Auckland Report 616. The right to use the material from this report is received with gratitude.

model capacity to data complexity ensuring a good generalization, i.e., a good performance of SVM on the future, previously, during the training unseen, data.

Same as the neural networks (or similarly to them), SVMs possess the well-known ability of being universal approximators of any multivariate function to any desired degree of accuracy. Consequently, they are of particular interest for modeling the unknown, or partially known, highly nonlinear, complex systems, plants or processes. Also, at the very beginning, and just to be sure what the whole chapter is about, we should state clearly when there is no need for an application of SVMs' model-building techniques. In short, whenever there exists an analytical closed-form model (or it is possible to devise one) there is no need to resort to learning from empirical data by SVMs (or by any other type of a learning machine)

2.1 Basics of Learning from Data

SVMs have been developed in the reverse order to the development of neural networks (NNs). SVMs evolved from the sound theory to the implementation and experiments, while the NNs followed more heuristic path, from applications and extensive experimentation to the theory. It is interesting to note that the very strong theoretical background of SVMs did not make them widely appreciated at the beginning. The publication of the first papers by Vapnik and Chervonenkis [145] went largely unnoticed till 1992. This was due to a widespread belief in the statistical and/or machine learning community that, despite being theoretically appealing, SVMs are neither suitable nor relevant for practical applications. They were taken seriously only when excellent results on practical learning benchmarks were achieved (in numeral recognition, computer vision and text categorization). Today, SVMs show better results than (or comparable outcomes to) NNs and other statistical models, on the most popular benchmark problems.

The learning problem setting for SVMs is as follows: there is some unknown and nonlinear dependency (mapping, function) $y = f(\mathbf{x})$ between some high-dimensional input vector $\mathbf{x}$ and the scalar output y (or the vector output $\mathbf{y}$ as in the case of multiclass SVMs). There is no information about the underlying joint probability functions here. Thus, one must perform a distribution-free learning. The only information available is a training data set $\{\mathcal{X} = [\mathbf{x}(i), y(i)] \in \Re^m \times \Re,\ i = 1, \ldots, n\}$, where n stands for the number of the training data pairs and is therefore equal to the size of the training data set $\mathcal{X}$. Often, y_i is denoted as d_i (i.e., t_i), where $d(t)$ stands for a desired (target) value. Hence, SVMs belong to the supervised learning techniques.

Note that this problem is similar to the classic statistical inference. However, there are several very important differences between the approaches and assumptions in training SVMs and the ones in classic statistics and/or NNs modeling. Classic statistical inference is based on the following three fundamental assumptions:

1. Data can be modeled by a set of linear in parameter functions; this is a foundation of a parametric paradigm in learning from experimental data.
2. In the most of real-life problems, a stochastic component of data is the normal probability distribution law, that is, the underlying joint probability distribution is a Gaussian distribution.
3. Because of the second assumption, the induction paradigm for parameter estimation is the maximum likelihood method, which is reduced to the minimization of the sum-of-errors-squares cost function in most engineering applications.

All three assumptions on which the classic statistical paradigm relied turned out to be inappropriate for many contemporary real-life problems [143] because of the following facts:

1. Modern problems are high-dimensional, and if the underlying mapping is not very smooth the linear paradigm needs an exponentially increasing number of terms with an increasing dimensionality of the input space (an increasing number of independent variables). This is known as 'the curse of dimensionality'.
2. The underlying real-life data generation laws may typically be very far from the normal distribution and a model-builder must consider this difference in order to construct an effective learning algorithm.
3. From the first two points it follows that the maximum likelihood estimator (and consequently the sum-of-error-squares cost function) should be replaced by a new induction paradigm that is uniformly better, in order to model non-Gaussian distributions.

In addition to the three basic objectives above, the novel SVMs' problem setting and inductive principle have been developed for standard contemporary data sets which are typically high-dimensional and sparse (meaning, the data sets contain small number of the training data pairs).

SVMs are the so-called 'nonparametric' models. 'Nonparametric' does not mean that the SVMs' models do not have parameters at all. On the contrary, their 'learning' (selection, identification, estimation, training or tuning) is the crucial issue here. However, unlike in classic statistical inference, the parameters are not predefined and their number depends on the training data used. In other words, parameters that define the capacity of the model are data-driven in such a way as to match the model capacity to data complexity. This is a basic paradigm of the structural risk minimization (SRM) introduced by Vapnik and Chervonenkis and their coworkers that led to the new learning algorithm. Namely, there are two basic constructive approaches possible in designing a model that will have a good generalization property [144, 143]:

1. choose an appropriate structure of the model (order of polynomials, number of HL neurons, number of rules in the fuzzy logic model) and, keep-

ing the estimation error (a.k.a. confidence interval, a.k.a. variance of the model) fixed in this way, minimize the training error (i.e., empirical risk), or
2. keep the value of the training error (a.k.a. an approximation error, a.k.a. an empirical risk) fixed (equal to zero or equal to some acceptable level), and minimize the confidence interval.

Classic NNs implement the first approach (or some of its sophisticated variants) and SVMs implement the second strategy. In both cases the resulting model should resolve the trade-off between under-fitting and over-fitting the training data. The final model structure (its order) should ideally *match the learning machines capacity with training data complexity*. This important difference in two learning approaches comes from the minimization of different cost (error, loss) functionals. Table 2.1 tabulates the basic risk functionals applied in developing the three contemporary statistical models. In Table 2.1, d_i stands for desired values, $\mathbf{w}$ is the weight vector subject to training, λ is a regularization parameter, $\mathbf{P}$ is a smoothness operator, L_ε is a SVMs' loss function, h is a VC dimension and Ω is a function bounding the capacity of the learning machine. In classification problems L_ε is typically 0-1 loss func-

Table 2.1. Basic Models and Their Error (Risk) Functionals

Multilayer perceptron (NN)	$R = \sum_{i=1}^{n} \underbrace{(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{Closeness\,to\,data}$
Regularization Network(Radial Basis Functions Network)	$R = \sum_{i=1}^{n} \underbrace{(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{Closeness\,to\,data} + \lambda \underbrace{\lVert\mathbf{P}f\rVert^2}_{Smoothness}$
Support Vector Machine	$R = \sum_{i=1}^{n} \underbrace{L_\varepsilon}_{Closeness\,to\,data} + \underbrace{\Omega(n,h)}_{Capacity\,of\,a\,machine}$

Closeness to data = training error, a.k.a. empirical risk

tion, and in regression problems L_ε is the so-called Vapnik's ε-insensitivity loss (error) function

$$L_\varepsilon = |y - f(\mathbf{x}, \mathbf{w})|_\varepsilon = \begin{cases} 0, & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \le \varepsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon, & \text{otherwise.} \end{cases} \tag{2.1}$$

where ε is a radius of a tube within which the regression function must lie, after the successful learning. (Note that for $\varepsilon = 0$, the interpolation of training data will be performed). It is interesting to note that [58] has shown that under some constraints the SV machine can also be derived from the framework of regularization theory rather than SLT and SRM. Thus, *unlike the classic* *adaptation algorithms (that work in the L_2 norm), SV machines represent novel learning techniques which perform SRM*. In this way, the SV machine creates a model with minimized VC dimension and when the VC dimension of the model is low, the expected probability of error is low as well. This means good performance on previously unseen data, i.e. a good generalization. This property is of particular interest because the model that generalizes well is a good model and not the model that performs well on training data pairs. Too good a performance on training data is also known as an extremely undesirable overfitting.

As it will be shown below, in the 'simplest' pattern recognition tasks, support vector machines use a linear separating hyperplane to create a *classifier with a maximal margin*. In order to do that, the learning problem for the SV machine will be cast as a *constrained nonlinear optimization* problem. In this setting the cost function will be quadratic and the constraints linear (i.e., one will have to solve a classic *quadratic programming problem*).

In cases when given classes cannot be linearly separated in the original input space, the SV machine first (non-linearly) transforms the original input space into a higher dimensional feature space. This transformation can be achieved by using various nonlinear mappings; polynomial, sigmoid as in multilayer perceptrons, RBF mappings having as the basis functions radially symmetric functions such as Gaussians, or multiquadrics or different spline functions. After this nonlinear transformation step, the task of a SV machine in finding the linear optimal separating hyperplane in this feature space is 'relatively trivial'. Namely, the optimization problem to solve in a feature space will be of the same kind as the calculation of a maximal margin separating hyperplane in original input space for linearly separable classes. How, after the specific nonlinear transformation, nonlinearly separable problems in input space can become linearly separable problems in a feature space will be shown later.

In a probabilistic setting, there are three basic components in all supervised learning from data tasks: a *generator* of random inputs $\mathbf{x}$, a *system* whose *training responses* y (i.e., d) are used for training the learning machine, and a *learning machine* which, by using inputs $\mathbf{x}_i$ and system's responses y_i, should learn (estimate, model) the unknown dependency between these two sets of variables (namely, $\mathbf{x}_i$ and y_i) defined by the weight vector $\mathbf{w}$ (Fig.2.1). The figure shows the most common learning setting that some readers may have already seen in various other fields - notably in statistics, NNs, control system identification and/or in signal processing. During the (successful) training phase a learning machine should be able to find the relationship between an input space X and an output space Y , by using data $\mathcal{X}$ in regression tasks (or to find a function that separates data within the input space, in classification ones). The result of a learning process is an 'approximating function' $f_a(\mathbf{x}, \mathbf{w})$, which in statistical literature is also known as, a *hypothesis* $f_a(\mathbf{x}, \mathbf{w})$. This function approximates the underlying (or true) dependency between the input and output in the case of regression, and the decision boundary, i.e.,

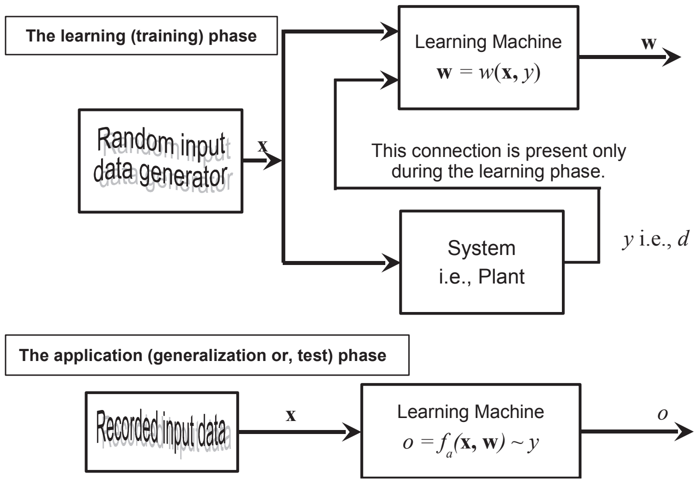

The learning (training) phase
Random input data generator
x
Learning Machine
$\mathbf{w} = w(\mathbf{x}, y)$
w
This connection is present only during the learning phase.
System i.e., Plant
y i.e., d
The application (generalization or, test) phase
Recorded input data
x
Learning Machine
$o = f_a(\mathbf{x}, \mathbf{w}) \sim y$
o

Fig. 2.1. A model of a learning machine (top) $\mathbf{w} = w(\mathbf{x}, y)$ that during *the training phase* (by observing inputs $\mathbf{x}_i$ to, and outputs y_i from, the system) estimates (learns, adjusts, trains, tunes) its parameters (weights) $\mathbf{w}$, and in this way learns mapping $y = f(\mathbf{x}, \mathbf{w})$ performed by the system. The use of $f_a(\mathbf{x}, \mathbf{w}) \sim y$ denotes that *we will rarely* try to *interpolate* training data pairs. We would rather seek an *approximating* function that can generalize well. After the training, at the *generalization* or *test phase*, the output from a machine $o = f_a(\mathbf{x}, \mathbf{w})$ is expected to be 'a good' estimate of a system's true response y.

separation function, in a classification. The chosen hypothesis $f_a(\mathbf{x}, \mathbf{w})$ belongs to a *hypothesis space of functions* $H(f_a \in H)$, and it is a function that minimizes some *risk functional* $R(\mathbf{w})$.

It may be practical to remind the reader that under the general name 'approximating function' we understand any mathematical structure that maps inputs $\mathbf{x}$ into outputs y. Hence, an 'approximating function' may be: a multilayer perceptron NN, RBF network, SV machine, fuzzy model, Fourier truncated series or polynomial approximating function. Here we discuss SVMs. A set of parameters $\mathbf{w}$ is the very subject of learning and generally these parameters are called weights. These parameters may have different geometrical and/or physical meanings. Depending upon the hypothesis space of functions H we are working with the parameters $\mathbf{w}$ are usually:

- the hidden and the output layer weights in multilayer perceptrons,
- the rules and the parameters (for the positions and shapes) of fuzzy subsets,
- the coefficients of a polynomial or Fourier series,
- the centers and (co)variances of Gaussian basis functions as well as the output layer weights of this RBF network,
- the support vector weights in SVMs.

There is another important class of functions in learning from examples tasks. A learning machine tries to capture an unknown *target function* $f_o(\mathbf{x})$ that is believed to belong to some target space T, or to a class T, that is also called a *concept class*. Note that we rarely know the target space T and that our learning machine generally does not belong to the same class of functions as an unknown target function $f_o(\mathbf{x})$. Typical examples of target spaces are continuous functions with s continuous derivatives in m variables; Sobolev spaces (comprising square integrable functions in m variables with s square integrable derivatives), band-limited functions, functions with integrable Fourier transforms, Boolean functions, etc. In the following, we will assume that the target space T is a space of differentiable functions. The basic problem we are facing stems from the fact that we know very little about the possible underlying function between the input and the output variables. All we have at our disposal is a training data set of labeled examples drawn by independently sampling $a(X \times Y)$ space according to some unknown probability distribution.

The learning-from-data problem is ill-posed. (This will be shown on Figs. 2.2 and 2.3 for a regression and classification examples respectively). The basic source of the ill-posedness of the problem is due to the infinite number of possible solutions to the learning problem. At this point, just for the sake of illustration, it is useful to remember that all functions that interpolate data points will result in a zero value for training error (empirical risk) as shown (in the case of regression) in Fig. 2.2. The figure shows a simple example of three-out-of-infinitely-many different interpolating functions of training data pairs sampled from a noiseless function $y = \sin(x)$.

In Fig. 2.2, each interpolant results in a training error equal to zero, but at the same time, each one is a very bad model of the true underlying dependency between x and y, because all three functions perform very poorly outside the training inputs. In other words, none of these three particular interpolants can generalize well. However, not only interpolating functions can mislead. There are many other approximating functions (learning machines) that will minimize the empirical risk (approximation or training error) but not necessarily the generalization error (true, expected or guaranteed risk). This follows from the fact that a learning machine is trained by using some particular sample of the true underlying function and consequently it always produces biased approximating functions. These approximants depend necessarily on the specific training data pairs (i.e., the training sample) used.

Fig. 2.3 shows an extremely simple classification example where the classes (represented by the empty training circles and squares) are linearly separable. However, in addition to a linear separation (dashed line) the learning was also performed by using a model of a high capacity (say, the one with Gaussian basis functions, or the one created by a high order polynomial, over the 2-dimensional input space) that produced a perfect separation boundary (empirical risk equals zero) too. However, such a model is overfitting the data and it will definitely perform very badly on, during the training unseen, test examples. Filled circles and squares in the right hand graph are all wrongly

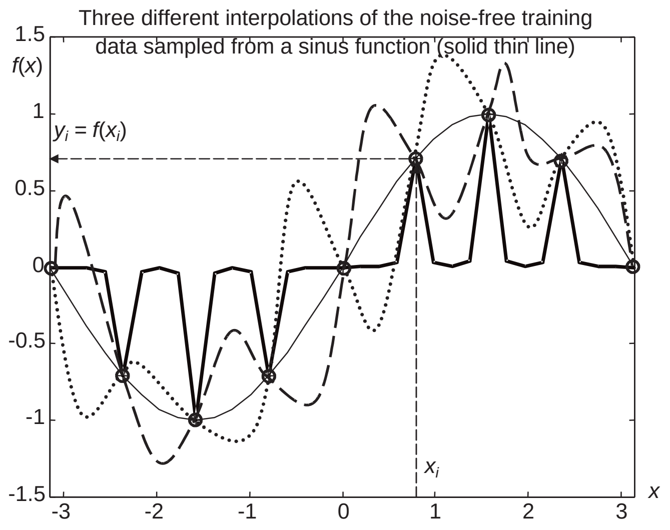


Fig. 2.2. Three-out-of-infinitely-many interpolating functions resulting in a training error equal to 0. However, a thick solid, dashed and dotted lines are bad models of a true function $y = \sin(x)$ (thin solid line).

classified by the nonlinear model. Note that a simple linear separation boundary correctly classifies both the training and the test data.

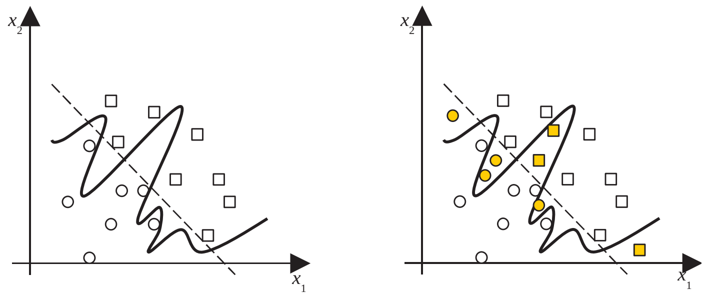


Fig. 2.3. Overfitting in the case of linearly separable classification problem. *Left*: The perfect classification of the training data (empty circles and squares) by both low order linear model (dashed line) and high order nonlinear one (solid wiggly curve). *Right*: Wrong classification of all the test data shown (filled circles and squares) by a high capacity model, but correct one by the simple linear separation boundary.

A solution to this problem proposed in the framework of the SLT is restricting the hypothesis space H of approximating functions to a set smaller than that of the target function T while simultaneously controlling the flexibility (complexity) of these approximating functions. This is ensured by an introduction of a novel induction principle of the SRM and its algorithmic realization through the SV machine. The Structural Risk Minimization principle [141] tries to minimize an expected risk (the cost function) R comprising two terms as given in Table 2.1 for the SVMs $R = \Omega(n,h) + \sum_{i=1}^{n} L_{\varepsilon} = \Omega(n,h) + R_{emp}$ and it is based on the fact that for the classification learning problem with a probability of at least $1-\eta$ the bound

$$R(\mathbf{w}_m) \leqslant \Omega(\frac{h}{n}, \frac{\ln(\eta)}{n}) + R_{emp}(\mathbf{w}_m) \tag{2.2a}$$

holds. The first term on the right hand side is named a VC confidence (confidence term or confidence interval) that is defined as

$$\Omega(\frac{h}{n}, \frac{\ln(\eta)}{n}) = \sqrt{\frac{h\left[\ln(\frac{2n}{h}) + 1\right] - \ln(\frac{\eta}{4})}{n}}. \tag{2.2b}$$

The parameter h is called the VC (Vapnik-Chervonenkis) dimension of a set of functions. It describes the capacity of a set of functions implemented in a learning machine. For a binary classification h is the maximal number of points which can be separated (shattered) into two classes in all possible 2^h ways by using the functions of the learning machine.

A SV (learning) machine can be thought of as

- a set of functions implemented in a SVM,
- an induction principle and,
- an algorithmic procedure for implementing the induction principle on the given set of functions.

The notation for risks given above by using $R(\mathbf{w}_m)$ denotes that an expected risk is calculated over a set of functions $f_{an}(\mathbf{x}, \mathbf{w}_m)$ of increasing complexity. Different bounds can also be formulated in terms of other concepts such as *growth function* or *annealed VC entropy*. Bounds also differ for regression tasks. More detail can be found in ([144], as well as in [32]). However, the general characteristics of the dependence of the confidence interval on the number of training data n and on the VC dimension h is similar and given in Fig 2.4.

Equations (2.2) show that when the number of training data increases, i.e., for $n \to \infty$(with other parameters fixed), an expected (true) risk $R(\mathbf{w}_n)$ is very close to empirical risk $R_{emp}(\mathbf{w}_n)$ because $\Omega \to 0$. On the other hand, when the probability $1-\eta$ (also called a confidence level which should not be confused with the confidence term Ω) approaches 1, the generalization bound grows large, because in the case when $\eta \to 0$ (meaning that the confidence level $1-\eta \to 1$), the value of $\Omega \to \infty$. This has an obvious intuitive interpretation

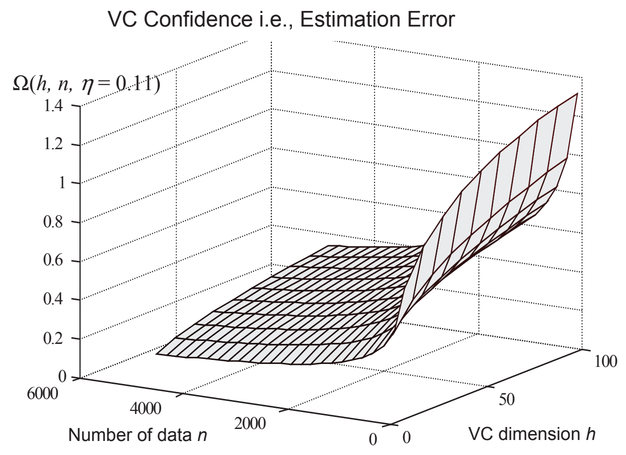


Fig. 2.4. The dependency of VC confidence interval $\Omega(h, n, \eta)$ on the number of training data n and the VC dimension $h(h < n)$ for a fixed confidence level $1 - \eta = 1 - 0.11 = 0.89$.

[32] in that any learning machine (model, estimates) obtained from a finite number of training data cannot have an arbitrarily high confidence level. There is always a trade-off between the accuracy provided by bounds and the degree of confidence (in these bounds). Fig 2.4 also shows that the VC confidence interval increases with an increase in a VC dimension h for a fixed number of the training data pairs n.

The SRM is a novel inductive principle for learning from finite training data sets. It proved to be very useful when dealing with small samples. The basic idea of the SRM is to choose (from a large number of possibly candidate learning machines), a model of the right capacity to describe the given training data pairs. As mentioned, this can be done by restricting the hypothesis space H of approximating functions and simultaneously by controlling their flexibility (complexity). Thus, learning machines will be those parameterized models that, by increasing the number of parameters (typically called weights w_i here), form a nested structure in the following sense

$$H_1 \subset H_2 \subset H_3 \subset \ldots H_{n-1} \tag{2.3}$$

In such a nested set of functions, every function always contains a previous, less complex, function. Typically, H_n may be: a set of polynomials in one variable of degree n; fuzzy logic model having n rules; multilayer perceptrons, or RBF network having n HL neurons, SVM structured over n support vectors. The goal of learning is one of a *subset selection* that matches training data complexity with approximating model capacity. In other words, a learning algorithm chooses an optimal polynomial degree or, an optimal number of HL neurons or, an optimal number of FL model rules, for a polynomial model or NN or FL model respectively. For learning machines linear in parameters, this complexity (expressed by the VC dimension) is given by the number of weights, i.e., by the number of 'free parameters'. For approximating models nonlinear in parameters, the calculation of the VC dimension is often not an easy task. Nevertheless, even for these networks, by using simulation experiments, one can find a model of appropriate complexity.

2.2 Support Vector Machines in Classification and Regression

Below, we focus on the algorithm for implementing the SRM induction principle on the given set of functions. It implements the strategy mentioned previously - it keeps the training error fixed and minimizes the confidence interval. We first consider a 'simple' example of linear decision rules (i.e., the separating functions will be hyperplanes) for binary classification (dichotomization) of linearly separable data. In such a problem, we are able to perfectly classify data pairs, meaning that an empirical risk can be set to zero. It is the easiest classification problem and yet an excellent introduction of all relevant and important ideas underlying the SLT, SRM and SVM.

Our presentation will gradually increase in complexity. It will begin with a *Linear Maximal Margin Classifier for Linearly Separable Data* where there is no sample overlapping. Afterwards, we will allow some degree of overlapping of training data pairs. However, we will still try to separate classes by using linear hyperplanes. This will lead to the *Linear Soft Margin Classifier for Overlapping Classes*. In problems when linear decision hyperplanes are no longer feasible, the mapping of an input space into the so-called feature space (that 'corresponds' to the HL in NN models) will take place resulting in the *Nonlinear Classifier*. Finally, in the subsection on *Regression by SV Machines* we introduce same approaches and techniques for solving regression (i.e., function approximation) problems.

2.2.1 Linear Maximal Margin Classifier for Linearly Separable Data

Consider the problem of binary classification or dichotomization. Training data are given as

$$(\mathbf{x}_1, y), (\mathbf{x}_2, y), \ldots, (\mathbf{x}_n, y_n), \ \mathbf{x} \in \Re^m, \quad y \in \{+1, -1\}. \tag{2.4}$$

For reasons of visualization only, we will consider the case of a two-dimensional input space, i.e., ($\mathbf{x} \in \Re^2$). Data are linearly separable and there are many different hyperplanes that can perform separation (Fig. 2.5). (Actually, for $\mathbf{x} \in \Re^2$, the separation is performed by 'planes' $w_1x_1 + w_2x_2 + b = d$. In other words, the decision boundary, i.e., the separation line in input space is defined by the equation $w_1x_1 + w_2x_2 + b = 0$.). How to find 'the best' one? The difficult part is that all we have at our disposal are sparse training data. Thus, we want to find the optimal separating function without knowing the underlying probability distribution $P(\mathbf{x}, y)$. There are many functions that can solve given pattern recognition (or functional approximation) tasks. In such a problem setting, the SLT (developed in the early 1960s by Vapnik and Chervonenkis [145]) shows that it is crucial to restrict the class of functions implemented by a learning machine to one with a complexity that is suitable for the amount of available training data.

In the case of a classification of linearly separable data, this idea is transformed into the following approach - among all the hyperplanes that minimize the training error (i.e., empirical risk) find the one with the largest margin. This is an intuitively acceptable approach. Just by looking at Fig 2.5 we will find that the dashed separation line shown in the *right graph* seems to promise *probably* good classification while facing previously unseen data (meaning, in the generalization, i.e. test, phase). Or, at least, it seems to probably be better in generalization than the dashed decision boundary having smaller margin shown in the left graph. This can also be expressed as that a classifier with smaller margin will have higher expected risk. By using given

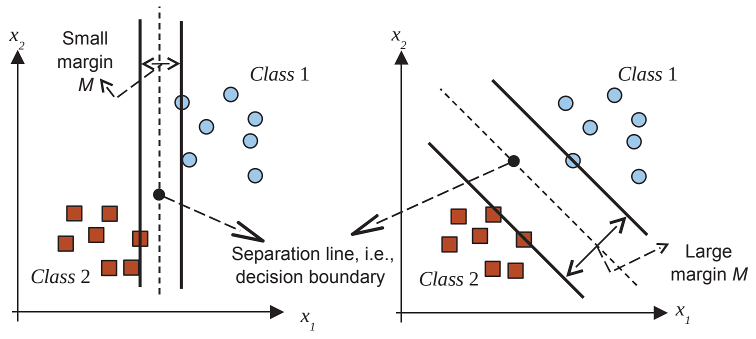


Fig. 2.5. Two-out-of-many separating lines: a good one with a large margin (right) and a less acceptable separating line with a small margin, (left).

training examples, during the learning stage, our machine finds parameters $\mathbf{w} = [w_1 \ w_2 \ \ldots w_m]^T$ and b of a discriminant or decision function $d(\mathbf{x}, \mathbf{w}, b)$ given as

$$d(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^{m} w_i x_i + b, \tag{2.5}$$

where $\mathbf{x}, \mathbf{w} \in \Re^m$, and the scalar b is called a *bias*.(Note that the dashed separation lines in Fig. 2.5 represent the line that follows from $d(\mathbf{x}, \mathbf{w}, b) = 0$). After the successful training stage, by using the weights obtained, the learning machine, given previously unseen pattern $\mathbf{x}_p$, produces output o according to an *indicator function* given as

$$i_F = o = \operatorname{sign}(d(\mathbf{x}_p, \mathbf{w}, b)). \tag{2.6}$$

where o is the standard notation for the *output* from the learning machine. In other words, *the decision rule* is:

- if $d(\mathbf{x}_p, \mathbf{w}, b) > 0$, the pattern $\mathbf{x}_p$ belongs to a class 1 (i.e., $o = y_p = +1$),
- and if $d(\mathbf{x}_p, \mathbf{w}, b) < 0$ the pattern $\mathbf{x}_p$, belongs to a class 2 (i.e., $o = y_p = -1$).

The *indicator function* i_F given by (2.6) is a step-wise (i.e., a stairs-wise) function (see Figs. 2.6 and 2.7). At the same time, the decision (or discriminant) function $d(\mathbf{x}, \mathbf{w}, b)$ is a hyperplane. Note also that both a decision hyperplane d and the indicator function i_F live in an $n + 1$-dimensional space or they lie 'over' a training pattern's n-dimensional input space. There is one more mathematical object in classification problems called a separation boundary that lives in the same n-dimensional space of input vectors $\mathbf{x}$. Separation boundary separates vectors $\mathbf{x}$ into two classes. Here, in cases of linearly separable data, the boundary is also a (separating) hyperplane but of a lower order than $d(\mathbf{x}, \mathbf{w}, b)$. The decision (separation) *boundary* is an intersection of a decision *function* $d(\mathbf{x}, \mathbf{w}, b)$ and a space of input features. It is given by

$$d(\mathbf{x}, \mathbf{w}, b) = 0. \tag{2.7}$$

All these functions and relationships can be followed, for two-dimensional inputs $\mathbf{x}$, in Fig. 2.6. In this particular case, the decision boundary i.e., separating (hyper)plane is actually a separating line in a $x_1 - x_2$ plane and, a decision function $d(\mathbf{x}, \mathbf{w}, b)$ is a plane over the 2-dimensional space of features, i.e., over a $x_1 - x_2$ plane. In the case of 1-dimensional training patterns x (i.e., for 1-dimensional inputs x to the learning machine), decision function $d(\mathbf{x}, \mathbf{w}, b)$ is a straight line in an $x - y$ plane. An intersection of this line with an x-axis defines a point that is a separation boundary between two classes. This can be followed in Fig. 2.7. Before attempting to find an optimal separating hyperplane having the largest margin, we introduce the concept of the *canonical hyperplane*. We depict this concept with the help of the 1-dimensional example shown in Fig. 2.7. Not quite incidentally, the decision plane $d(\mathbf{x}, \mathbf{w}, b)$ shown in Fig. 2.6 is also a *canonical* plane. Namely, the values of d and of i_F are the same and both are equal to $|1|$ for the support vectors depicted by stars. At the same time, for all other training patterns $|d| > |i_F|$.

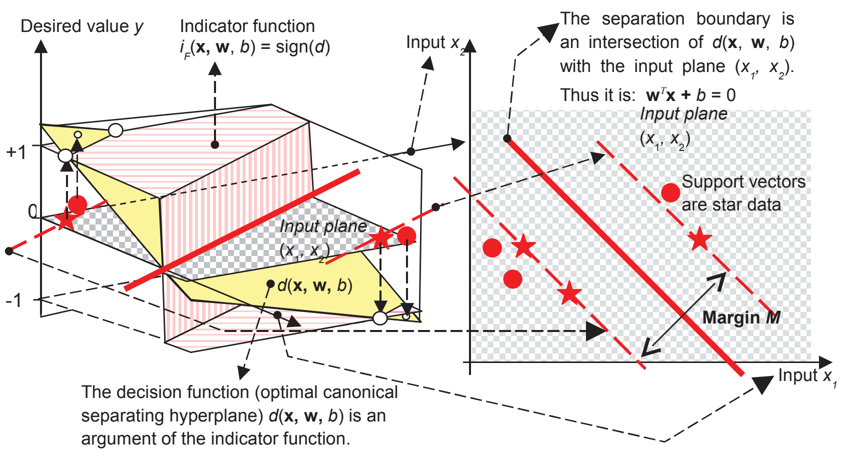

Desired value y
Indicator function $i_F(\mathbf{x}, \mathbf{w}, b) = \text{sign}(d)$
Input x_2
The separation boundary is an intersection of $d(\mathbf{x}, \mathbf{w}, b)$ with the input plane (x_1, x_2). Thus it is: $\mathbf{w}^T\mathbf{x} + b = 0$
Input plane (x_1, x_2)
Support vectors are star data
+1
0
-1
Input plane (x_1, x_2)
$d(\mathbf{x}, \mathbf{w}, b)$
Margin M
Input x_1
The decision function (optimal canonical separating hyperplane) $d(\mathbf{x}, \mathbf{w}, b)$ is an argument of the indicator function.

Fig. 2.6. The definition of a decision (discriminant) *function* or hyperplane $d(\mathbf{x}, \mathbf{w}, b)$, a decision boundary $d(\mathbf{x}, \mathbf{w}, b) = 0$ and an indicator function $i_F = \text{sign}(d(\mathbf{x}, \mathbf{w}, b))$ whose value represents a learning machine's output o.

In order to present a notion of this new concept of the canonical plane, first note that there are many hyperplanes that can correctly separate data. In Fig. 2.7 three different decision functions $d(\mathbf{x}, \mathbf{w}, b)$ are shown. There are infinitely many more. In fact, given $d(\mathbf{x}, \mathbf{w}, b)$, all functions $d(\mathbf{x}, k\mathbf{w}, kb)$, where k is a positive scalar, are correct decision functions too. Because parameters $(\mathbf{w}, b)$ describe the same separation hyperplane as parameters $(k\mathbf{w}, kb)$ there is a need to introduce the notion of a *canonical hyperplane*:

A hyperplane is in the canonical form with respect to the training data $x_i,\ i = 1, \dots, n$, if

$$\underbrace{\min}_{\mathbf{x}_i \in \mathcal{X}} \left| \mathbf{w}^T \mathbf{x}_i + b \right| = 1. \tag{2.8}$$

The solid line $d(\mathbf{x}, \mathbf{w}, b) = -2x + 5$ in Fig. 2.7 fulfills (2.8) because its minimal *absolute value for the given six training patterns* belonging to two classes is 1. It achieves this value for two patterns, chosen as support vectors, namely for $x_3 = 2$, and $x_4 = 3$. For all other patterns, $|d| > 1$. Note an interesting detail regarding the notion of a canonical hyperplane that is easily checked. There are many different hyperplanes (planes and straight lines for 2-D and 1-D problems in Figs. 2.6 and 2.7 respectively) that have the same separation boundary (solid line and a dot in Figs. 2.6 (right) and 2.7 respectively). At the same time there are far fewer hyperplanes that can be defined as canonical ones fulfilling (2.8). In Fig. 2.7, i.e., for a 1-dimensional input vector x, the canonical hyperplane is unique. This is not the case for training patterns of higher dimension. Depending upon the configuration of class' elements, various canonical hyperplanes are possible.

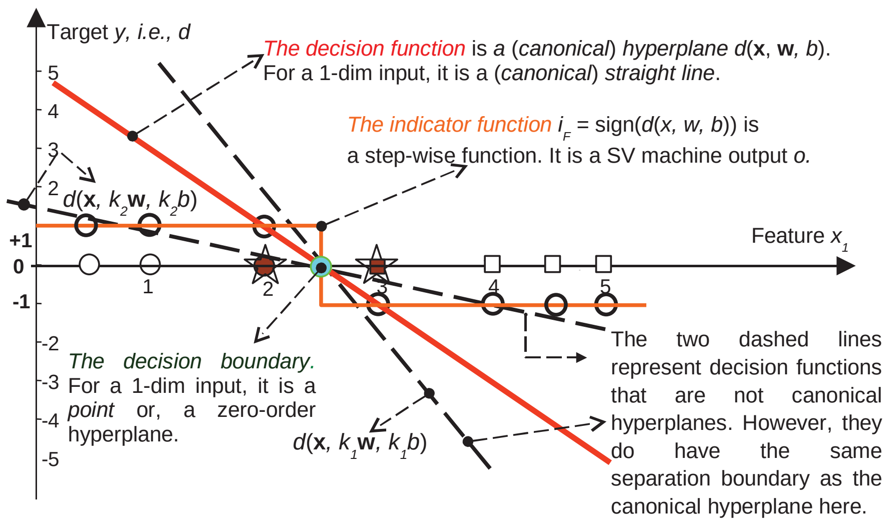

Target y, i.e., d
The decision function is a (canonical) hyperplane d(x, w, b).
For a 1-dim input, it is a (canonical) straight line.
The indicator function i_F = sign(d(x, w, b)) is
a step-wise function. It is a SV machine output o.
$d(\mathbf{x}, k_2\mathbf{w}, k_2 b)$
Feature x_1
The decision boundary. For a 1-dim input, it is a point or, a zero-order hyperplane.
$d(\mathbf{x}, k_1\mathbf{w}, k_1 b)$
The two dashed lines represent decision functions that are not canonical hyperplanes. However, they do have the same separation boundary as the canonical hyperplane here.
+1
0
-1
5
4
3
2
1
-2
-3
-4
-5

Fig. 2.7. SV classification for 1-dimensional inputs by the linear decision function. Graphical presentation of a canonical hyperplane. For 1-dimensional inputs, it is actually a canonical straight line (depicted as a thick straight solid line) that passes through points (+2, +1) and (+3, -1) defined as the support vectors (stars). The two dashed lines are the two other decision hyperplanes (i.e., straight lines). The training input patterns $\{x_1 = 0.5, x_2 = 1, x_3 = 2\} \in$ $Class1$ have a desired or target value (label) $y_1 = +1$. The inputs $\{x_4 = 3, x_5 = 4, x_6 = 4.5, x_7 = 5\} \in$ $Class2$ have the label $y_2 = -1$.

Therefore, there is a need to define an *optimal* separation canonical hyperplane (OCSH) as a canonical hyperplane having a *maximal margin*. This search for a separating, maximal margin, canonical hyperplane is the ultimate learning goal in statistical learning theory underlying SV machines. Carefully note the adjectives used in the previous sentence. The hyperplane obtained from a limited training data must have a *maximal margin* because it will *probably* better classify new data. It must be in *canonical* form because this will ease the quest for significant patterns, here called support vectors. The canonical form of the hyperplane will also simplify the calculations. Finally, the resulting hyperplane must ultimately *separate* training patterns.

We avoid the derivation of an expression for the calculation of a distance (margin M) between the closest members from two classes for its simplicity here. Instead, the curious reader can find a derivation of (2.9) in the Appendix A. There are other ways to get (2.9) which can be found in other books or monographs on SVMs. The margin M can be derived by both the geometric and algebraic argument and is given as

$$M = \frac{2}{\|\mathbf{w}\|}. \tag{2.9}$$

This important result will have a great consequence for the constructive (i.e., learning) algorithm in a design of a maximal margin classifier. It will lead to solving a quadratic programming (QP) problem which will be shown shortly. Hence, the 'good old' gradient learning in NNs will be replaced by solution of the QP problem here. This is the next important difference between the NNs and SVMs and follows from the implementation of SRM in designing SVMs, instead of a minimization of the sum of error squares, which is a standard cost function for NNs. Equation (2.9) is a very interesting result showing that minimization of a norm of a hyperplane normal weight vector $\|\mathbf{w}\| = \sqrt{\mathbf{w}^T\mathbf{w}} = \sqrt{w_1^2 + w_1^2 + \ldots + w_m^2}$ leads to a maximization of a margin M. Because a minimization of $\sqrt{f}$ is equivalent to the minimization of f, the minimization of a norm $\|\mathbf{w}\|$ equals a minimization of $\mathbf{w}^T\mathbf{w} = \sum_{i=1}^{m} w_1^2 + w_1^2 + \ldots + w_m^2$, and this leads to a maximization of a margin M. Hence, the learning problem is

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{2.10a}$$

subject to constraints introduced and given in (2.10b) below. (A multiplication of $\mathbf{w}^T\mathbf{w}$ by 0.5 is for numerical convenience only, and it doesn't change the solution). Note that in the case of linearly separable classes empirical error equals zero ($R_{emp} = 0$ in (2.2a)) and minimization of $\mathbf{w}^T\mathbf{w}$ corresponds to a minimization of a confidence term Ω. The OCSH, i.e., a separating hyperplane with the largest margin defined by $M = 2/\|\mathbf{w}\|$, specifies *support vectors*, i.e., training data points closest to it, which satisfy $y_j[\mathbf{w}^T\mathbf{x}_j + b] \equiv 1,\ j = 1, N_{SV}$. For all the other (non-SVs data points) the OCSH satisfies inequalities $y_i[\mathbf{w}^T\mathbf{x}_i + b] > 1$. In other words, for all the data, OCSH should satisfy the following constraints

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad i = 1, \ldots, n. \tag{2.10b}$$

where n denotes a number of training data points, and N_{SV} stands for a number of SVs. The last equation can be easily checked visually in Figs. 2.6 and 2.7 for 2-dimensional and 1-dimensional input vectors $\mathbf{x}$ respectively. Thus, in order to find the OCSH having a maximal margin, a learning machine should minimize $\|\mathbf{w}\|^2$ subject to the inequality constraints (2.10b). This is a *classic quadratic optimization problem with inequality constraints*. Such an optimization problem is solved by the *saddle point* of the Lagrange functional (Lagrangian) [2].

$$L_p(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} \alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1\}. \tag{2.11}$$

[2] In forming the Lagrangian, for constraints of the form $f_i > 0$, the inequality constraints equations are multiplied by *nonnegative* Lagrange multipliers (i.e., $\alpha_i \geq 0$) and *subtracted* from the objective function.

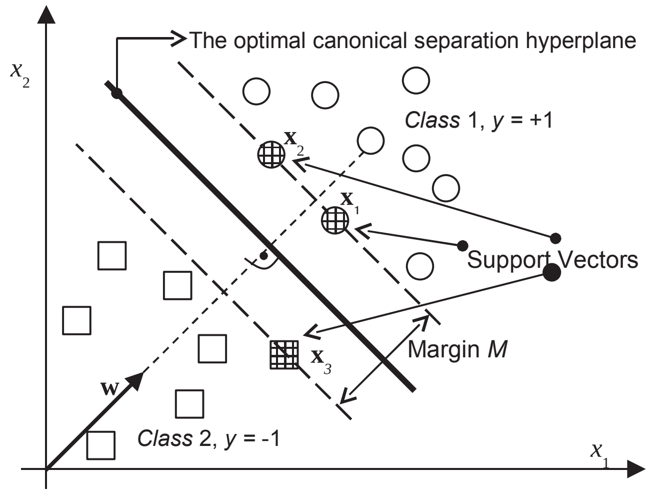


Fig. 2.8. The optimal canonical separation hyperplane with the largest margin intersects halfway between the two classes. The points closest to it (satisfying $y_j \left|\mathbf{w}^T\mathbf{x}_j + b\right| = 1, j = 1, N_{SV}$) are *support vectors* and the OCSH satisfies $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1,\ i = 1, n$ (where n denotes the number of training data and N_{SV} stands for the number of SV). Three support vectors ($\mathbf{x}_1$ and $\mathbf{x}_2$ from class 1, and $\mathbf{x}_3$ from class 2) are the textured training data.

where the α_i are Lagrange multipliers. The search for an optimal *saddle point* $(\mathbf{w}_o, b_o, \boldsymbol{\alpha}_o)$ is necessary because Lagrangian L_p must be *minimized* with respect to $\mathbf{w}$ and b, and has to be *maximized* with respect to nonnegative α_i (i.e., $\alpha_i \geq 0$ should be found). This problem can be solved either in a *primal space* (which is the space of parameters $\mathbf{w}$ and b) or in a *dual space* (which is the space of Lagrange multipliers α_i). The second approach gives insightful results and we will consider the solution in a dual space below. In order to do that, we use Karush-Kuhn-Tucker (KKT) conditions for the optimum of a constrained function. In our case, both the objective function (2.11) and constraints (2.10b) are *convex* and KKT conditions are *necessary* and *sufficient* conditions for a maximum of (2.11). These conditions are: at the saddle point $(\mathbf{w}_o, b_o, \boldsymbol{\alpha}_o)$, derivatives of Lagrangian L_p with respect to primal variables should vanish which leads to,

$$\frac{\partial L_p}{\partial \mathbf{w}_o} = 0 \quad \mathbf{w}_o = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\,, \tag{2.12a}$$

$$\frac{\partial L_p}{\partial b} = 0 \quad \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{2.12b}$$

and the KKT complementarity conditions below (stating that at the solution point the products between dual variables and constraints equals zero) must also be satisfied,

$$\alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1\} = 0, \quad i = 1, \ldots, n \tag{2.13}$$

This also means that the condition (2.13) must also be satisfied to ensure that the solution of the primal Lagrangian L_p is the same as the solution of the original optimization problem (2.10). The standard change to a dual Lagrangian problem is to first substitute $\mathbf{w}$ from (2.12a) into the primal Lagrangian (2.11) and this leads to a dual Lagrangian problem below

$$L_d(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,\,j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T\mathbf{x}_j - \sum_{i=1}^{n} \alpha_i y_i b, \tag{2.14a}$$

$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \ldots, n \tag{2.14b}$$

subject to the inequality constraints (2.14b). In a standard SVMs formulation, (2.12b) is also used to eliminate the last term of (2.14a), so one only needs to maximize L_d with respect to α_i. As a result, the dual function to be maximized is (2.15a) with inequality constraints (2.15b) and an equality constraint (2.15c).

$$\max\ L_d(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T\mathbf{x}_j \tag{2.15a}$$

$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \ldots, n \quad \text{and} \tag{2.15b}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{2.15c}$$

Note that the dual Lagrangian $L_d(\boldsymbol{\alpha})$ is expressed in terms of training data and depends *only* on the *scalar products* of input patterns $(\mathbf{x}_i^T\mathbf{x}_j)$. The dependency of $L_d(\boldsymbol{\alpha})$ on a scalar product of inputs will be very handy later when analyzing nonlinear decision boundaries and for general nonlinear regression. Note also that the number of unknown variables equals the number of training data n. After learning, the number of free parameters is equal to the number of SVs but it does not depend on the dimensionality of input space. Such *a standard quadratic optimization problem* can be expressed in a *matrix notation* and formulated as follows:

$$\max\ \ L_d(\boldsymbol{\alpha}) = -0.5\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} + \mathbf{p}^T\boldsymbol{\alpha}, \tag{2.16a}$$

$$\text{s.t. } \ \mathbf{y}^T\boldsymbol{\alpha} = 0, \tag{2.16b}$$

$$\alpha_i \geq 0, \quad i = 1, \ldots, n, \tag{2.16c}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$, $\mathbf{H}$ denotes a symmetric Hessian matrix (with elements $H_{ij} = y_i y_j \mathbf{x}_i^T\mathbf{x}_j$), and $\mathbf{p}$ is an $n \times 1$ unit vector $\mathbf{p} = \mathbf{1} = [1\ 1 \ldots 1]^T$. (Note that maximization of (2.16a) equals a minimization of $L_d(\boldsymbol{\alpha}) = 0.5\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} - \mathbf{p}^T\boldsymbol{\alpha}$, subject to the same constraints).The Hessian matrix has a size of n by n and it is always a dense matrix. This means that the learning of SVMs scales with the size of training data. This is the main reason for the development of a fast iterative learning algorithm which does not need to store the complete Hessian matrix in Chap. 3.1 . Solution $\boldsymbol{\alpha}_o$ of the dual optimization (2.15) determines the parameters of the optimal hyperplane $\mathbf{w}_o$ and b_o according to (2.12a) and (2.13) as follows,

$$\mathbf{w}_o = \sum_{i=1}^{n} \alpha_{oi} y_i \mathbf{x}_i \tag{2.17a}$$

$$b_o = \frac{1}{N_{SV}} \sum\nolimits_{s=1}^{N_{SV}} \left(\frac{1}{y_s} - \mathbf{x}_s^T \mathbf{w}_o\right) = \frac{1}{N_{SV}} \sum\nolimits_{s=1}^{N_{SV}} (y_s - \mathbf{x}_s^T \mathbf{w}_o) \quad s = 1, \dots, N_{sv} \tag{2.17b}$$

In deriving (2.17b) the fact that y can be either +1 or -1, and $1/y = y$ is used. N_{SV} denotes the number of support vectors. There are two important observations about the calculation of $\mathbf{w}_o$. First, an optimal weight vector $\mathbf{w}_o$, is obtained in (2.17a) as a linear combination of the training data points and second, $\mathbf{w}_o$ (same as the bias term b_o) is calculated by using only the selected data points called support vectors (SVs) . It is because they have nonzero α_{oi} and they are the data which support forming the decision function. Thus, the data having $\alpha_{oi} = 0$ are called non-SVs here. The fact that the summation in (2.17a) goes over all training data (i.e., from 1 to n) is irrelevant because the Lagrange multipliers α_{oi} for all non-SVs are equal to zero. Furthermore, if now all the non-SVs are removed from the training data set and only the SVs are used for training, then the same solution (i.e., the same values of $\mathbf{w_o}$ and b_o) will be obtained as the ones obtained by using the complete training data set. This is a very pleasing property of SVMs, because the solutions of good models are generally sparse (only 10-20% of the complete data set are SVs). For linearly separable training data, all support vectors lie on the margin and they are generally just a small portion of all training data (typically, $N_{SV} << n$). Figs. 2.5, 2.7 and 2.8 show the geometry of standard results for non-overlapping classes. Also, in order to satisfy the KKT complementarity conditions (2.13), the output of the decision function ($\mathbf{w}^T\mathbf{x}_s + b$) at SVs must have a magnitude of 1. This fact is used in the derivation of (2.17b). Equation (2.17b) tries to find out the optimal bias term b_o by taking the average deviation between the desired output y_s (+1 or -1) and $\mathbf{x}_s\mathbf{w}_o$ over all SVs. After calculating $\mathbf{w}_o$ and b_o, a decision hyperplane and an indicator function are obtained as follows

$$d(\mathbf{x}) = \sum\nolimits_{i=1}^{n} w_{oi}x_i + b_o = \sum\nolimits_{i=1}^{n} y_i \alpha_i \mathbf{x}_i^T \mathbf{x} + b_o, \quad i_F = o = \text{sign}(d(\mathbf{x})). \tag{2.18}$$

Before presenting a derivation of an OCSH for both overlapping classes and classes having nonlinear decision boundaries, we will comment only on whether and how SV based linear classifiers actually implement the SRM principle. The more detailed presentation of this important property can be found in [81, 123]. First, it can be shown that an increase in margin reduces the number of points that can be shattered i.e., the increase in margin reduces the VC dimension, and this leads to the decrease of the SVM capacity. In short, by minimizing $\|\mathbf{w}\|$ (i.e., maximizing the margin) the SV machine training actually minimizes the VC dimension and consequently a generalization error (expected risk) at the same time. This is achieved by imposing a structure on the set of canonical hyperplanes and then, during the training, by choosing the one with a minimal VC dimension. A structure on the set of canonical hyperplanes is introduced by considering various hyperplanes having different $\|\mathbf{w}\|$. In other words, we analyze sets S_A such that $\|\mathbf{w}\| \leq A$. Then, if $A_1 \leq A_2 \leq \ldots \leq A_m$, we introduced a nested set $S_{A1} \subset S_{A2} \subset S_{A3} \ldots \subset S_{Am}$. Thus, if we impose the constraint $\|\mathbf{w}\| \leq A$, then the canonical hyperplane cannot be closer than $1/A$ to any of the training points $\mathbf{x}_i$. Vapnik in [144] states that the VC dimension h of a set of canonical hyperplanes in $\Re^m$ such that $\|w\| \leq A$ is

$$H \leq min[R^2, A^2, m] + 1, \tag{2.19}$$

where all the training data points (vectors) are enclosed by a sphere of the smallest radius R. Therefore, a small $\|\mathbf{w}\|$ results in a small h, and minimization of $\|\mathbf{w}\|$ is an implementation of the SRM principle. In other words, a minimization of the canonical hyperplane weight norm $\|\mathbf{w}\|$ minimizes the VC dimension according to (2.19). See also Fig. 2.4 that shows how the estimation error, meaning the expected risk (because the empirical risk, due to the linear separability, equals zero) decreases with a decrease of a VC dimension. Finally, there is an interesting, simple and powerful result [144] connecting the generalization ability of learning machines and the number of support vectors. Once the support vectors have been found, we can calculate the bound on the expected probability of committing an error on a test example as follows

$$E_n \left[P(\text{error})\right] \leq \frac{E\left[\text{number of support vectors}\right]}{n}, \tag{2.20}$$

where E_n denotes expectation over all training data sets of size n. Note how easy it is to estimate this bound that is independent of the dimensionality of the input space. Therefore, an SV machine having a small number of support vectors will have good generalization ability even in a very high-dimensional space.

Example below shows the SVM's learning of the weights for a simple separable data problem in both the primal and the dual domain. The small number and low dimensionality of data pairs is used in order to show the optimization steps analytically and graphically. The same reasoning will be in the case of high dimensional and large training data sets but for them, one has to rely on computers and the insight in solution steps is necessarily lost.

Example 2.1. Consider a design of SVM classifier for 3 data shown in Fig. 2.9 below. First we solve the problem in the primal domain: From the constraints (2.10b) it follows

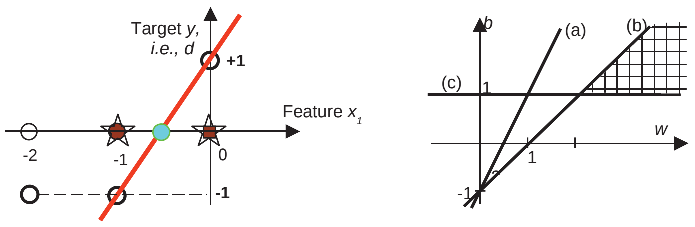


Fig. 2.9. *Left:* Solving SVM classifier for 3 data shown. SVs are star data. *Right:* Solution space $w - b$.

$$\begin{aligned} &2w - 1 \geq b && (a)\\ &w - 1 \geq b && (b)\\ &b \geq 1 && (c) \end{aligned} \tag{2.21}$$

The three straight lines corresponding to the equalities above are shown in Fig. 2.9 right. The textured area is a feasible domain for the weight w and bias b. Note that the area is not defined by the inequality (a), thus pointing to the fact that the point -2 is not a support vector. Points -1 and 0 define the textured area and they will be the supporting data for our decision function. The task is to minimize (2.10a), and this will be achieved by taking the value $w = 2$. Then, from (b), it follows that $b = 1$. Note that (a) must not be used for the calculation of the bias term b.

Because both the cost function (2.10a) and the constraints (2.10b) are convex, the primal and the dual solution must produce same w and b. Dual solution follows from maximizing (2.15a) subject to (2.15b) and (2.15c) as follows

$$L_d = \alpha_1 + \alpha_2 + \alpha_3 - \frac{1}{2}[\alpha_1\ \alpha_2\ \alpha_3] \begin{bmatrix} 4\ 2\ 0 \\ 2\ 1\ 0 \\ 0\ 0\ 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix},$$

$$\text{s.t.} \qquad -\alpha_1 - \alpha_2 + \alpha_3 = 0,$$
$$\alpha_1 \geqslant 0, \quad \alpha_2 \geqslant 0, \quad \alpha_3 \geqslant 0,$$

The dual Lagrangian is obtained in terms of α_1 and α_2 after expressing α_3 from the equality constraint and it is given as $L_d = 2\alpha_1 + 2\alpha_2 - 0.5(4\alpha_1^2 + 4\alpha_1\alpha_2 + \alpha_2^2)$. L_d will have maximum for $\alpha_1 = 0$, and it follows that we have to find the maximum of $L_d = 2\alpha_2 - 0.5\alpha_2^2$ which will be at $\alpha_2 = 2$. Note that the Hessian matrix **H** is extremely bad conditioned and, if the QP problem is to be solved by computer, **H** should be regularized first. From the equality constraint it follows that $\alpha_3 = 2$ too. Now, we can calculate the weight vector w and the bias b from (2.17a) and (2.17b) as follows,

$$w = \sum_{i=1}^{3} \alpha_i y_i \mathbf{x}_i = 0(-1)(-2) + 2(-1)(-1) + 2(1)0 = 2$$

The bias can be calculated by using SVs only, meaning from either point -1 or point 0. Both result in same value as shown below

$$b = -1 - 2(-1) = 1, \text{ or } b = 1 - 2(0) = 1$$

2.2.2 Linear Soft Margin Classifier for Overlapping Classes

The learning procedure presented above is valid for linearly separable data, meaning for training data sets without overlapping. Such problems are rare in practice. At the same time, there are many instances when linear separating hyperplanes can be good solutions even when data are overlapped (e.g., normally distributed classes having the same covariance matrices have a linear separation boundary). However, quadratic programming solutions as given above cannot be used in the case of overlapping because the constraints $y_i[\mathbf{w}^T\mathbf{x}_i + b] \geq 1, i = 1, n$ given by (2.10b) cannot be satisfied. In the case of an overlapping (see Fig. 2.10), the overlapped data points cannot be correctly classified and for any misclassified training data point $\mathbf{x}_i$, the corresponding α_i will tend to infinity. This particular data point (by increasing the corresponding α_i value) attempts to exert a stronger influence on the decision boundary in order to be classified correctly. When the α_i value reaches the maximal bound, it can no longer increase its effect, and the corresponding point will stay misclassified. In such a situation, the algorithm introduced above chooses all training data points as support vectors. To find a classifier with a maximal margin, the algorithm presented in the Sect. 2.2.1, must be changed allowing some data to be unclassified. Better to say, we must leave some data on the 'wrong' side of a decision boundary. In practice, we allow a *soft* margin and all data inside this margin (whether on the correct side of the separating line or on the wrong one) are neglected. The width of a soft margin can be controlled by a corresponding penalty parameter C (introduced below) that determines the trade-off between the training error and VC dimension of the model.

The question now is how to measure the degree of misclassification and how to incorporate such a measure into the hard margin learning algorithm given by (2.10). The simplest method would be to form the following learning problem

$$\min \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\text{number of misclassified data}) \qquad (2.22)$$

where C is a penalty parameter, trading off the margin size (defined by $\|\mathbf{w}\|$, i.e., by $\mathbf{w}^T\mathbf{w}$) for the number of misclassified data points. Large C leads to small number of misclassifications, bigger $\mathbf{w}^T\mathbf{w}$ and consequently to the smaller margin and vice versa. Obviously taking $C = \infty$ requires that the number of misclassified data is zero and, in the case of an overlapping this is not possible. Hence, the problem may be feasible only for some value $C < \infty$.

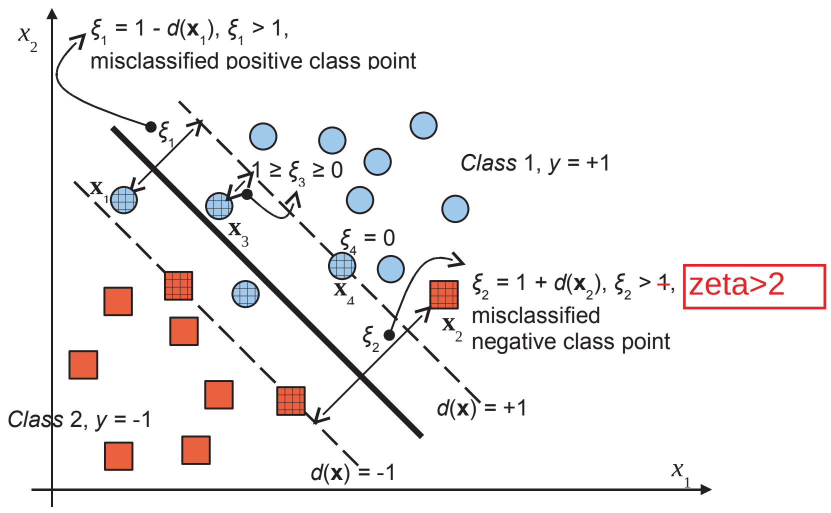


Fig. 2.10. The soft decision boundary for a dichotomization problem with data overlapping. Separation line (solid), margins (dashed) and support vectors (textured training data points).). 4 SVs in positive class (circles) and 3 SVs in negative class (squares). 2 misclassifications for positive class and 1 misclassification for negative class.

However, the serious problem with (2.22) is that the error's counting can't be accommodated within the handy (meaning reliable, well understood and well developed) quadratic programming approach. Also, the counting only can't distinguish between huge (or disastrous) errors and close misses! The possible solution is to measure the distances ξ_i of the points crossing the margin from the corresponding margin and trade their sum for the margin size as given below

$$\min \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\text{sum of distances of the wrong side points}), \tag{2.23}$$

In fact this is exactly how the problem of the data overlapping was solved in [39, 40] - by generalizing the optimal 'hard' margin algorithm. They introduced the nonnegative *slack variables* $\xi_i (i = 1,\ n)$ in the statement of the optimization problem for the overlapped data points. Now, instead of fulfilling (2.10a) and (2.10b), the separating hyperplane must satisfy

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i \tag{2.24a}$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \quad \text{and} \tag{2.24b}$$

$$\xi_i \geq 0. \tag{2.24c}$$

i.e., subject to

$$\mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i, \quad \text{for } y_i = +1,\ \xi_i \geq 0 \tag{2.24d}$$
$$\mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i, \quad \text{for } y_i = -1,\ \xi_i \geq 0 \tag{2.24e}$$

Hence, for such a generalized optimal separating hyperplane, the functional to be minimized comprises an extra term accounting the cost of overlapping errors. In fact the cost function (2.24a) can be even more general as given below

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i^k \tag{2.24f}$$

subject to same constraints. This is a convex programming problem that is usually solved only for $k = 1$ or $k = 2$, and such soft margin SVMs are dubbed *L1* and *L2 SVMs* respectively. By choosing exponent $k = 1$, neither slack variables ξ_i nor their Lagrange multipliers β_i appear in a dual Lagrangian L_d. Same as for a linearly separable problem presented previously, for *L1 SVMs* ($k = 1$) here, the solution to a quadratic programming problem (2.24), is given by the saddle point of the primal Lagrangian $L_p(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ shown below

$$L_p(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\sum_{i=1}^{n}\xi_i) - \sum_{i=1}^{n}\alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i\ +\ b] - 1 + \xi_i\} - \sum_{i=1}^{n}\beta_i\xi_i. \tag{2.25}$$

where α_i and β_i are the Lagrange multipliers. Again, we should find an *optimal* saddle point $(\mathbf{w}_o, b_o, \boldsymbol{\xi}_o, \boldsymbol{\alpha}_o, \boldsymbol{\beta}_o)$ because the Lagrangian L_p has to be *minimized* with respect to $\mathbf{w}$, b and ξ_i and *maximized* with respect to nonnegative α_i and β_i. As before, this problem can be solved in either a primal space or dual space (which is the space of Lagrange multipliers α_i and β_i.). Again, we consider a solution in a dual space as given below by using - standard conditions for an optimum of a constrained function

$$\frac{\partial L}{\partial \mathbf{w}_o} = 0, \text{ i.e., } \quad \mathbf{w}_o = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i \tag{2.26a}$$
$$\frac{\partial L}{\partial b_o} = 0, \text{ i.e., } \quad \sum_{i=1}^{n}\alpha_i y_i = 0 \tag{2.26b}$$
$$\frac{\partial L}{\partial \xi_{io}} = 0, \text{ i.e., } \quad \alpha_i + \beta_i = C \tag{2.26c}$$

and the KKT complementarity conditions below,

$$\alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1 + \xi_i\} = 0, \quad i = 1, \ldots, n, \tag{2.26d}$$
$$\beta_i\xi_i = (C - \alpha_i)\xi_i = 0, \qquad i = 1, \ldots, n. \tag{2.26e}$$

At the optimal solution, due to the KKT conditions (2.26d) and (2.26e), the last two terms in the primal Lagrangian L_p given by (2.25) vanish and the *dual variables Lagrangian* $L_d(\boldsymbol{\alpha})$, for *L1 SVM*, is not a function of β_i . In fact, it is same as the hard margin classifier's L_d given before and repeated here for the soft margin one,

$$\max\, L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j \tag{2.27a}$$

In order to find the optimal hyperplane, a dual Lagrangian $L_d(\boldsymbol{\alpha})$ has to be *maximized* with respect to nonnegative and (unlike before) smaller than or equal to C, α_i. In other words with

$$0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n \tag{2.27b}$$

and under the constraint (2.26b), i.e., under

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{2.27c}$$

Thus, the final quadratic optimization problem is practically same as for the separable case the only difference being in the modified bounds of the Lagrange multipliers α_i. The penalty parameter C, which is now the upper bound on α_i, is determined by the user. The selection of a 'good' or 'proper' C is always done experimentally by using some cross-validation technique. Note that in the previous linearly separable case, without data over-lapping, this upper bound $C = \infty$. We can also readily change to the matrix notation of the problem above as in (2.16). Most important of all is that the learning problem is expressed only in terms of unknown Lagrange multipliers α_i, and known inputs and outputs. Furthermore, optimization does not solely depend upon inputs $\mathbf{x}_i$ which can be of a very high (inclusive of an infinite) dimension, but it depends upon a scalar product of input vectors $\mathbf{x}_i$. It is this property we will use in the next section where we design SV machines that can create nonlinear separation boundaries. Finally, expressions for both a *decision function* $d(\mathbf{x})$ and an indicator function $i_F = \text{sign}(d(\mathbf{x}))$ for a soft margin classifier are same as for linearly separable classes and are also given by (2.18).

From (2.26d) and (2.26e) follows that there are only three possible solutions for α_i (see Fig. 2.10)

1. α_i, $\xi_i = 0$, $\rightarrow$ data point $\mathbf{x}_i$ is correctly classified,
2. $C > \alpha_i > 0$, $\rightarrow$ then, the two complementarity conditions must result in $y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1 + \xi_i = 0$, and $\xi_i = 0$. Thus, $y_i[\mathbf{w}^T\mathbf{x}_i + b] = 1$ and $\mathbf{x}_i$ is a support vector. The support vectors with $C \geq \alpha_i \geq 0$ are called *unbounded* or free support vectors. They lie on the two margins,
3. $\alpha_i = C$, $\rightarrow$ then, $y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1 + \xi_i = 0$, and $\xi_i \not\geq 0$, and $\mathbf{x}_i$ is a support vector. The support vectors with $\alpha_i = C$ are called *bounded support vectors*. They lie on the 'wrong' side of the margin. For $1 > \xi_i \geq 0$, $\mathbf{x}_i$ is still correctly classified, and if $\xi_i \geq 1$, $\mathbf{x}_i$ is misclassified. $>$

xxx

After the learning, the parameter $\mathbf{w}_o$ of the optimal hyperplane is calculated using the same expression (2.17a) as in the linearly separable case. For computing the optimal bias term b_o, the same philosophy as in (2.17b) is used, but the bounded support vectors BSV must not be included because they are not supposed to be on the margin, i.e. the ξ_i term for BSV should be greater than 0. Therefore, the formulation for working out the optimal bias b_o does not include BSV and it is given as follow,

$$b_o = \frac{1}{N_{FSV}} \sum\nolimits_{s=1}^{N_{FSV}} (y_s - \mathbf{x}_s^T \mathbf{w}_o), \quad s = 1, \ldots, N_{FSV} \tag{2.28}$$

where N_{FSV} is the number of free support vectors. The same indicator function (2.18) is used for the soft margin SVMs as in the hard margin ones.

For *L2 SVM* the second term in the cost function (2.24f) is quadratic, i.e., $C \sum\nolimits_{i=1}^{n} \xi_i^2$, and this leads to changes in a dual optimization problem which is now,

$$L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,\, j=1}^{n} y_i y_j \alpha_i \alpha_j \left(\mathbf{x}_i^T \mathbf{x}_j + \frac{\delta_{ij}}{C} \right) \tag{2.29a}$$

subject to

$$\alpha_i \geq 0, \quad i = 1, n \tag{2.29b}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.29c}$$

where, $\delta_{ij} = 1$ for $i = j$, and it is zero otherwise. Note the change in Hessian matrix elements given by second terms in (2.29a), as well as that there is no upper bound on α_i. The detailed analysis and comparisons of the *L1* and *L2 SVMs* is presented in [1]. Derivation of (2.29) is given in the Appendix A. We use the most popular *L1 SVMs* here, because they usually produce more sparse solutions, i.e., they create a decision function by using less SVs than the L2 SVMs.

2.2.3 The Nonlinear SVMs Classifier

The linear classifiers presented in two previous sections are very limited. Mostly, classes are not only overlapped but the genuine separation functions are nonlinear hypersurfaces. A nice and strong characteristic of the approach presented above is that it can be easily (and in a relatively straightforward manner) extended to create nonlinear decision boundaries. The motivation for such an extension is that an SV machine that can create a nonlinear decision hypersurface will be able to classify nonlinearly separable data. This will be achieved by considering a linear classifier in the so-called feature space that will be introduced shortly. A very simple example of a need for designing nonlinear models is given in Fig. 2.11 where the true separation boundary is quadratic. It is obvious that no errorless linear separating hyperplane can be found now. The best linear separation function shown as a dashed straight line would make six misclassifications (textured data points; 4 in the negative class and 2 in the positive one). Yet, if we use the nonlinear separation boundary we are able to separate two classes without any error. Generally, for n-dimensional input patterns, instead of a nonlinear curve, an SV machine will create a nonlinear separating hypersurface.

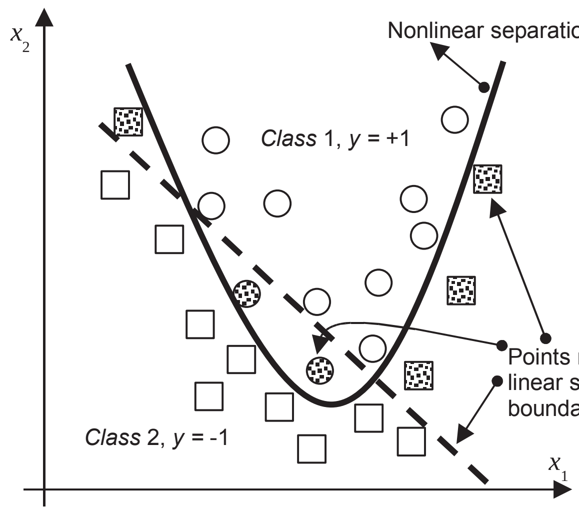

x_2
Nonlinear separation boundary
Class 1, *y* = +1
Points misclassified by linear separation boundary are textured
Class 2, *y* = -1
x_1

Fig. 2.11. A nonlinear SVM without data overlapping. A true separation is a quadratic curve. The nonlinear separation line (solid), the linear one (dashed) and data points misclassified by the linear separation line (the textured training data points) are shown. There are 4 misclassified negative data and 2 misclassified positive ones. SVs are not shown.

The basic idea of designing nonlinear SVMs is to map the input vectors $\mathbf{x}_i \in \Re^m$ into vectors $\boldsymbol{\Phi}(\mathbf{x}_i) \in \Re^s$ of a high dimensional feature space $\mathcal{S}$ (where $\boldsymbol{\Phi}$ represents mapping: $\Re^m \rightarrow \Re^s$) and to solve a linear classification problem in this feature space:

$$\mathbf{x} \in \Re^m \rightarrow \boldsymbol{\Phi}(\mathbf{x}) = [\phi_1(\mathbf{x})\ \phi_2(\mathbf{x}), \ldots, \phi_s(\mathbf{x})]^T \in \Re^s. \tag{2.30}$$

A mapping $\boldsymbol{\Phi}$ is chosen in advance, i.e., it is a fixed function. Note that an input space ($\mathbf{x}$-space) is spanned by components x_i of an input vector $\mathbf{x}$ and a feature space $\mathcal{S}$ ($\boldsymbol{\Phi}$-space) is spanned by components $\phi_i(\mathbf{x})$ of a vector $\boldsymbol{\Phi}(\mathbf{x})$. By performing such a mapping, we hope that in a $\boldsymbol{\Phi}$-space, our learning algorithm will be able to linearly separate images of $\mathbf{x}$ by applying the linear SVM formulation presented above. (In fact, it can be shown that for a whole class of mappings the linear separation in a feature space is always possible. Such mappings will correspond to the positive definite kernels that will be shown shortly). We also expect this approach to again lead to solving a quadratic optimization problem with similar constraints in a $\boldsymbol{\Phi}$-space. The solution for an indicator function $i_F(\mathbf{x}) = \text{sign}(\mathbf{w}^T\boldsymbol{\Phi}(\mathbf{x}) + b) = \text{sign}\left(\sum_{i=1}^{n} y_i\alpha_i\mathbf{\Phi}^T(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x}) + b\right)$, which is a linear classifier in a feature space, will create a nonlinear separating hypersurface in the original input space given by (2.31) below. (Compare this solution with (2.18) and note the appearances of scalar products in both the original X-space and in the feature space $\mathcal{S}$).

The equation for an $i_F(\mathbf{x})$ just given above can be rewritten in a 'neural networks' form as follows

$$
\begin{aligned}
i_F(\mathbf{x}) = i_F(d(\mathbf{x})) = \text{sign}(\mathbf{w}^T\boldsymbol{\Phi}(\mathbf{x}) + b) &= \text{sign}(\sum_{i=1}^{n} y_i\alpha_i K(\mathbf{x}_i, \mathbf{x}) + b) \\
&= \text{sign}(\sum_{i=1}^{n} v_i K(\mathbf{x}_i, \mathbf{x}) + b).
\end{aligned}
\tag{2.31}
$$

where v_i corresponds to the output layer weights of the 'SVM's network' and $K(\mathbf{x}_i, \mathbf{x})$ denotes the value of the kernel function that will be introduced shortly. (v_i equals $y_i\alpha_i$ in the classification case presented above and it is equal to $(\alpha_i - \alpha_i*)$ in the regression problems). Note the difference between the weight vector $\mathbf{w}$ which norm should be minimized and which is the vector of the same dimension as the feature space vector $\boldsymbol{\Phi}(\mathbf{x})$ and the weightings $v_i = \alpha_i y_i$ that are scalar values composing the weight vector $\mathbf{v}$ which dimension equals the number of training data points n. The $(n - N_{SVs})$ of v_i components are equal to zero, and only N_{SVs} entries of $\mathbf{v}$ are nonzero elements.

A simple example below (Fig. 2.12) should exemplify the idea of a nonlinear mapping to (usually) higher dimensional space and how it happens that the data become linearly separable in the $\mathcal{S}$-space.

Example 2.2. Consider solving the simplest nonlinear 1-D classification problem in Fig. 2.12 given the three input and output (desired) values as follows: $\mathbf{x} = [-1\ 0\ 1]^T$ and $\mathbf{y} = [-1\ 1\ -1]^T$. The following mapping is chosen to form the feature space here: $\boldsymbol{\Phi}(x) = [x^2\ \sqrt{2}x\ 1]^T = [\phi_1(x)\ \phi_2(x)\ \phi_3(x)]^T$. The mapping produces the following three points in the feature space.

$$
\begin{array}{lll}
x_1 = -1\ y_1 = -1 & & \boldsymbol{\Phi}(x_1) = [\ 1\ -\sqrt{2}\ \ 1]^T \\
x_2 = 0\ \ y_2 = +1 & \rightarrow & \boldsymbol{\Phi}(x_2) = [\ 0\ \ \ 0\ \ \ 1]^T \\
x_3 = 1\ \ y_3 = -1 & & \boldsymbol{\Phi}(x_3) = [\ 1\ \ \sqrt{2}\ \ 1]^T
\end{array}
\tag{2.32}
$$

These three points are shown in Fig. 2.13 and they are now linearly separable in the 3-D feature space. The figure also shows that the separating boundary

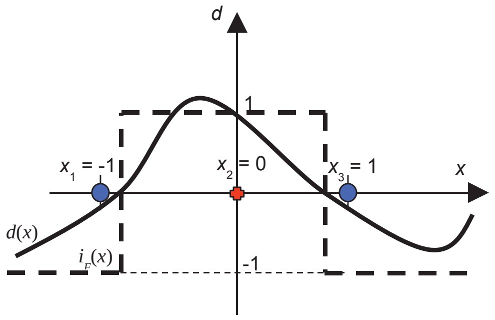


Fig. 2.12. A simple nonlinear 1-D classification problem. A linear classifier cannot be used to separate the data points successfully. One possible solution is given by the decision function $d(x)$ (solid curve). Its corresponding indicator function sign($d(x)$) is also given as a dash line.

from the optimal separating (hyper)plane is perpendicular to the x^2 direction and it has the biggest margin. Note that the decision hyperplane cannot be visualized in Fig. 2.13, because it exists in the space which is for one dimension higher (namely, in a 4-D space).

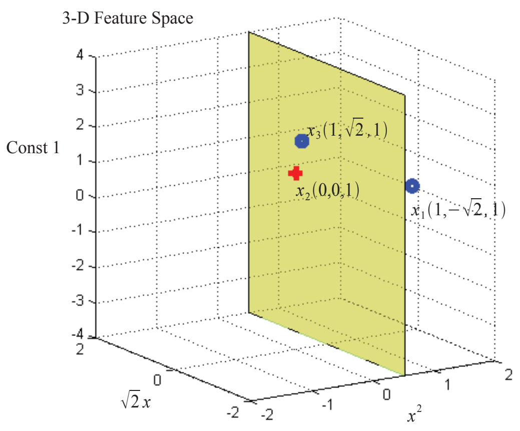


Fig. 2.13. The three points of the problem in Fig. 2.12 are linearly separable in the feature space (obtained by the mapping $\boldsymbol{\Phi}(\mathbf{x}) = [\phi_1(\mathbf{x})\ \phi_2(\mathbf{x})\ \phi_3(\mathbf{x})]^T = [x^2\ \sqrt{2}x\ 1]^T$). The separation boundary from SVMs which gives maximal margin is given by the plane $\phi_1(x) = 0.5$ as shown in the figure. Note that the decision function is in the 4-D space.

Although the use of feature mapping $\boldsymbol{\Phi}$ allows learning machines to deal with nonlinear dependency, there are two basic problems when performing such a mapping:

1. the choice of mapping $\boldsymbol{\Phi}$ that should result in a "rich" class of decision hyperplane.
2. the calculation of the scalar product $\boldsymbol{\Phi}(\mathbf{x})^T\boldsymbol{\Phi}(\mathbf{x})$ can be computationally very challenging if the number of features s (i.e., dimensionality s of a feature space) is very large.

The second problem is connected with a phenomenon called the '*curse of dimensionality*'. For example, to construct a decision surface corresponding to a polynomial of degree two in an m-D input space, a dimensionality of a feature space $s = m(m+3)/2$. In other words, a feature space is spanned by s coordinates of the form

$$\begin{aligned} z_1 = x_1, \ldots, z_m = x_m \ (m \text{ coordinates}),\\ z_{m+1} = (x_1)^2, \ldots, z_{2m} = (x_m)^2 \ (\text{next } m \text{ coordinates}),\\ z_{2m+1} = x_1x_2, \ldots, \ z_f = x_m x_{m-1} \ (m(m-1)/2\text{coordinates}), \end{aligned}$$

and the separating hyperplane created in this space, is a second-degree polynomial in the input space [143]. Thus, constructing a polynomial of degree two only, in a 256-dimensional input space, leads to a dimensionality of a feature space $s = 33,152$. Performing a scalar product operation with vectors of such, or higher, dimensions, is not a cheap computational task. The problems become serious (and fortunately only seemingly unsolvable) if we want to construct a polynomial of degree 4 or 5 in the same 256-dimensional space leading to the construction of a decision hyperplane in a billion-dimensional feature space.

This explosion in dimensionality can be avoided by noticing that in the quadratic optimization problem given by (2.15) and (2.27a), as well as in the final expression for a classifier, *training data only appear in the form of scalar products* $\mathbf{x}_i^T\mathbf{x}_j$. These products will be replaced by scalar products $\boldsymbol{\Phi}^T(\mathbf{x})\boldsymbol{\Phi}(\mathbf{x})_i = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_m(\mathbf{x})][\phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \ldots, \phi_m(\mathbf{x})_i]^T$ in a feature space $\mathcal{S}$, and the latter can be and will be expressed by using the *kernel function* $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)$.

Note that a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ is a function in input space. Thus, the basic advantage in using kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ is in avoiding performing a mapping $\boldsymbol{\Phi}(\mathbf{x})$ at all. Instead, the required scalar products in a feature space $\boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)$, are calculated directly by computing kernels $K(\mathbf{x}_i, \mathbf{x}_j)$ for given training data vectors in an input space. In this way, we bypass a possibly extremely high dimensionality of a feature space $\mathcal{S}$. Thus, by using the chosen kernel $K(\mathbf{x}_i, \mathbf{x}_j)$, we can construct an SVM that operates in an infinite dimensional space (such a kernel function is a Gaussian kernel function given in Table 2.2). In addition, as will be shown below, by applying kernels we do not even have to know what the actual mapping $\boldsymbol{\Phi}(\mathbf{x})$ is. A kernel is a function K such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j). \tag{2.33}$$

There are many possible kernels, and the most popular ones are given in Table 2.2. All of them should fulfill the so-called Mercer's conditions. The Mercer's kernels belong to a set of reproducing kernels. For further details see [101, 4, 129, 143, 81]. The simplest is a linear kernel defined as $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$. Below we show a few more kernels:

POLYNOMIAL KERNELS:
Let $x \in \Re^2$ i.e., $\mathbf{x} = [x_1 \ \ x_2]^T$, and if we choose $\boldsymbol{\Phi}(\mathbf{x}) = [x_1^2 \ \ \sqrt{2}x_1x_2 \ \ x_1^2]^T$ (i.e., there is an $\Re^2 \to \Re^3$ mapping), then the dot product

$\boxed{\text{x\_2}}$

$$\begin{aligned}\boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j) &= [x_{i1}^2 \ \ \sqrt{2}x_{i1}x_{i2} \ \ x_{i1}^2][x_{j1}^2 \ \ \sqrt{2}x_{j1}x_{j2} \ \ x_{j1}^2]^T \\ &= [x_{i1}^2x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{i2} + x_{i2}^2x_{j2}^2] = (\mathbf{x}_i^T\mathbf{x}_j)^2 = K(\mathbf{x}_i, \mathbf{x}_j), \text{or} \\ K(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i^T\mathbf{x}_j)^2 = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)\end{aligned}$$

Note that in order to calculate the scalar product in a feature space $\boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)$, we do not need to perform the mapping $\boldsymbol{\Phi}(\mathbf{x}) = [x_1^2 \ \sqrt{2}x_1x_2 \ x_1^2]^T$ at all. Instead, we calculate this product directly in the input space by computing $(\mathbf{x}_i^T\mathbf{x}_j)^2$. This is very well known under the popular name of *the kernel trick*. Interestingly, note also that other mappings such as an

$$\begin{aligned}&\Re^2 \to \Re^3 \text{ mapping given by } \boldsymbol{\Phi}(\mathbf{x}) = [x_1^2 - x_2^2 \ \ 2x_1x_2 \ \ x_1^2 + x_2^2], \text{ or an} \\ &\Re^2 \to \Re^4 \text{ mapping given by } \boldsymbol{\Phi}(\mathbf{x}) = [x_1^2 \ \ x_1x_2 \ \ x_1x_2 \ \ x_2^2]\end{aligned}$$

also accomplish the same task as $(\mathbf{x}_i^T\mathbf{x}_j)^2$.

Now, assume the following mapping

$$\boldsymbol{\Phi}(\mathbf{x}) = [1 \ \ \sqrt{2}x_1 \ \ \sqrt{2}x_2 \ \ \sqrt{2}x_1x_2 \ \ x_1^2 \ \ x_2^2],$$

i.e., there is an $\Re^2 \to \Re^5$ mapping plus bias term as the constant 6th dimension's value. Then the dot product in a feature space $\mathcal{S}$ is given as

$$\begin{aligned}\boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j) &= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 2x_{i1}x_{i2}x_{j1}x_{i2} + x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 \\ &= 1 + 2(\mathbf{x}_i^T\mathbf{x}_j) + (\mathbf{x}_i^T\mathbf{x}_j)^2 = (\mathbf{x}_i^T\mathbf{x}_j + 1)^2 = K(\mathbf{x}_i, \mathbf{x}_j), \text{ or} \\ K(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i^T\mathbf{x}_j + 1)^2 = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)\end{aligned}$$

Thus, the last mapping leads to the second order *complete* polynomial.

Many candidate functions can be applied to a convolution of an inner product (i.e., for kernel functions) $K(\mathbf{x}, \mathbf{x}_i)$ in a SV machine. Each of these functions constructs a different nonlinear decision hypersurface in an input space. In the first three rows, the Table 2.2 shows the three most popular kernels in SVMs' in use today, and the inverse multiquadrics one as an interesting

Table 2.2. Popular Admissible Kernels

Kernel Functions	Type of Classifier
$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T \mathbf{x}_i)$	Linear, dot product, kernel, CPD[a]
$K(\mathbf{x}, \mathbf{x}_i) = [(\mathbf{x}^T \mathbf{x}_i) + 1]^d$	Complete polynomial of degree d, PD[b]
$K(\mathbf{x}, \mathbf{x}_i) = \exp(-[\|\mathbf{x} - \mathbf{x}_i\|^2]/2\sigma^2)$	Gaussian RBF, PD[b]
$K(\mathbf{x}, \mathbf{x}_i) = \tanh[(\mathbf{x}^T \mathbf{x}_i) + b]^*$	Multilayer perceptron, CPD
$K(\mathbf{x}, \mathbf{x}_i) = 1/\sqrt{\|\mathbf{x} - \mathbf{x}_i\|^2 + \beta}$	Inverse multiquadric function, PD

[a] Conditionally positive definite [b] Positive definite
[*] only for certain values of b

Note that over the *n* dimensional input space, the number of monomial terms of degree less than or equal to *d* (i.e. the feature space dimensionality) explodes, and it can be calculated as
Feature space dimension
(*n+d* over *d*)
n+d
d

nd powerful kernel to be proven yet. The positive definite (PD) kernels are he kernels which Gramm matrix $\mathbf{G}$ (a.k.a. Grammian) calculated by using ll the n training data points is positive definite (meaning all its eigenvalues are strictly positive, i.e., $\lambda_i > 0, i = 1, n$)

$$\mathbf{G} = \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \vdots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \vdots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{2.34}$$

The $\mathbf{G}$ is a symmetric one. Even more, any symmetric positive definite matrix can be regarded as a kernel matrix, that is - as an inner product matrix in some space.

Finally, we arrive at the point of presenting the learning in nonlinear classifiers (in which we are ultimately interested here). The learning algorithm for a nonlinear SV machine (classifier) follows from the design of an optimal separating hyperplane in a feature space. This is the same procedure as the construction of a 'hard' (2.15) and 'soft' (2.27a) margin classifiers in an $\mathbf{x}$-space previously. In a $\boldsymbol{\Phi}(\mathbf{x})$-space, the dual Lagrangian, given previously by (2.15) and (2.27a), is now

$$L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,\, j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{\Phi}_i^T \boldsymbol{\Phi}_j, \tag{2.35}$$

and, according to (2.33), by using chosen kernels, we should maximize the following dual Lagrangian

$$\max L_d(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.36a}$$

$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \ldots, n \quad \text{and} \tag{2.36b}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{2.36c}$$

In a more general case, because of a noise or due to generic class' features, there will be an overlapping of training data points. Nothing but constraints for α_i change. Thus, constraints (2.36b) will be replaced by

$$0 \leq \alpha_i \leq C \quad i = 1, \ldots, n. \tag{2.36d}$$

Again, the only difference to the separable nonlinear classifier is the upper bound C on the Lagrange multipliers α_i. In this way, we limit the influence of training data points that will remain on the 'wrong' side of a separating nonlinear hypersurface. After the dual variables are calculated, the decision hypersurface $d(\mathbf{x})$ is determined by

$$d(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b = \sum_{i=1}^{n} v_i K(\mathbf{x}, \mathbf{x}_i) + b, \tag{2.37}$$

and the indicator function is $i_F(\mathbf{x}) = \text{sign}[d(\mathbf{x})] = \text{sign}\left[\sum_{i=1}^{n} v_i K(\mathbf{x}, \mathbf{x}_i) + b\right]$.

Note that the summation is not actually performed over all training data but rather over the support vectors, because only for them do the Lagrange multipliers differ from zero. The existence and calculation of a bias b is now not a direct procedure as it is for a linear hyperplane. Depending upon the applied kernel, the bias b can be implicitly part of the kernel function. If, for example, Gaussian RBF is chosen as a kernel, it can use a bias term as the $s + 1^{\text{st}}$ feature in $\mathcal{S}$-space with a constant output $= +1$, but not necessarily. In short, all PD kernels do not necessarily need an explicit bias term b, but b can be used. More on this can be found in [84] as well as in the [150]. Same as for the linear SVM, (2.36) can be written in a matrix notation as

$$\max L_d(\boldsymbol{\alpha}) = -0.5\boldsymbol{\alpha}^T \mathbf{H}\boldsymbol{\alpha} + \mathbf{p}^T \boldsymbol{\alpha}, \tag{2.38a}$$

$$\text{s.t.} \quad \mathbf{y}^T \boldsymbol{\alpha} = 0, \tag{2.38b}$$

$$0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n, \tag{2.38c}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$, $\mathbf{H}$ denotes the Hessian matrix $(H_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j))$ of this problem, and $\mathbf{p}$ is an $(n,1)$ unit vector $\mathbf{p} = \mathbf{1} = [1\ 1 \ldots 1]^T$. Note that the Hessian matrix is a dense n by n matrix. As a result, the amount of the computer memory required to solve the optimization problem is n^2. This is why the next part of the book is focused on solving the problem in an iterative way. The optimization problem (2.38) can be solved without the equality constraint (2.38b) when the Hessian matrix is positive definite (Note that if $K(\mathbf{x}_i, \mathbf{x}_j)$ is the positive definite matrix, then so is the matrix $y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ too.). This fact is also used extensively in next chapter for deriving faster iterative learning algorithm for SVMs.

Example 2.3. The following 1-D example (just for the sake of graphical presentation) will show the creation of a linear decision function in a feature space and a corresponding nonlinear (quadratic) decision function in an input space.

Suppose we have 4 1-D data points given as $x_1 = 1,\ x_2 = 2,\ x_3 = 5,\ x_4 = 6$, with data at 1, 2, and 6 as class 1 and the data point at 5 as class 2, i.e., $y_1 = -1,\ y_2 = -1,\ y_3 = 1,\ y_4 = -1$. We use the polynomial kernel of degree 2, $K(x, y) = (xy + 1)^2$. C is set to 50, which is of lesser importance because the constraints will not be imposed in this example due to the fact that the maximal value of the dual variables alpha will be smaller than $C = 50$.

Case 1: Working with a bias term b as given in (2.37)
We first find $\alpha_i (i = 1, \dots, 4)$ by solving dual problem (2.38) having a Hessian matrix

$$\mathbf{H} = \begin{bmatrix} 4 & 9 & -36 & 49 \\ 9 & 25 & -121 & 169 \\ -36 & -121 & 676 & -961 \\ 49 & 169 & -961 & 1369 \end{bmatrix}$$

Alphas are $\alpha_1 = 0,\ \alpha_2 = 2.499999,\ \alpha_3 = 7.333333\ \alpha_4 = 4.833333$ and the bias b will be found by using (2.17b), or by fulfilling the requirements that the values of a decision function at the support vectors should be the given y_i. The model (decision function) is given by

$$d(x) = \sum_{i=1}^{4} y_i \alpha_i K(x,\, x_i) + b = \sum_{i=1}^{4} v_i (x x_i + 1)^2 + b, \quad \text{or by}$$

$$d(x) = 2.4999(-1)(2x + 1)^2 + 7.3333(1)(5x + 1)^2 + 4.8333(-1)(6x + 1)^2 + b$$

$$d(x) = -0.666667x^2 + 5.333333x + b$$

Bias b is determined from the requirement that at the SV points 2, 5 and 6, the outputs must be -1, 1 and -1 respectively. Hence, $b = -9$, resulting in the decision function

$$d(x) = -0.666667x^2 + 5.333333x - 9.$$

The nonlinear (quadratic) decision function and the indicator one are shown in Fig. 2.14. Note that in calculations above 6 decimal places have been used for alpha values. The calculation is numerically very sensitive, and working with fewer decimals can give very approximate or wrong results.

The complete polynomial kernel as used in the case 1, is positive definite and there is no need to use an explicit bias term b as presented above. Thus, one can use the same second order polynomial model without the bias term b. Note that in this particular case there is no equality constraint equation that originates from an equalization of the primal Lagrangian derivative in respect to the bias term b to zero. Hence, we do not use (2.38b) while using a positive definite kernel without bias as it will be shown below in the case 2.

Case 2: Working without a bias term b

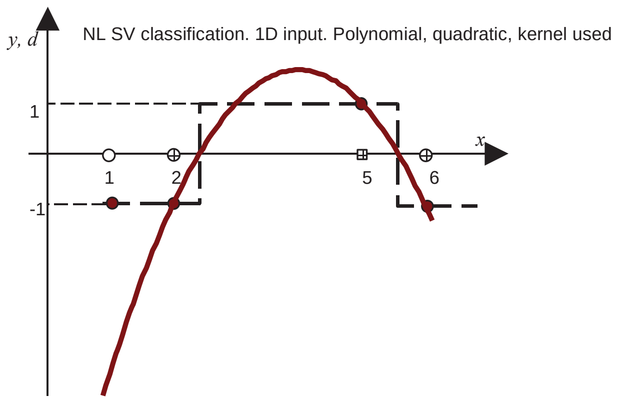


Fig. 2.14. The nonlinear decision function (solid) and the indicator function (dashed) for 1-D overlapping data. By using a complete second order polynomial the model with and without a bias term b are same.

Because we use the same second order polynomial kernel, the Hessian matrix $\mathbf{H}$ is same as in the case 1. The solution without the equality constraint for alphas is: $\alpha_1 = 0$, $\alpha_2 = 24.999999$, $\alpha_3 = 43.333333$, $\alpha_4 = 27.333333$. The model (decision function) is given by

$$d(x) = \sum_{i=1}^{4} y_i \alpha_i K(x, x_i) = \sum_{i=1}^{4} v_i (x x_i + 1)^2, \quad \text{or by}$$

$$d(x) = 24.9999(-1)(2x+1)^2 + 43.3333(1)(5x+1)^2 + 27.3333(-1)(6x+1)^2$$

$$d(x) = -0.666667x^2 + 5.333333x - 9.$$

Thus the nonlinear (quadratic) decision function and consequently the indicator function in the two particular cases are equal.

Example 2.4. XOR problems: In the next example shown by Figs. 2.15 and 2.16 we present all the important mathematical objects of a nonlinear SV classifier by using a classic XOR (*exclusive-or*) problem. The graphs show all the mathematical functions (objects) involved in a nonlinear classification. Namely, the nonlinear decision function $d(\mathbf{x})$, the NL indicator function $i_F(\mathbf{x})$, training data $(\mathbf{x}_i)$, support vectors $(\mathbf{x}_{SV})_i$ and separation boundaries.

The same objects will be created in the cases when the input vector $\mathbf{x}$ is of a dimensionality $n > 2$, but the visualization in these cases is not possible. In such cases one talks about the decision hyperfunction (hypersurface) $d(\mathbf{x})$, indicator hyperfunction (hypersurface) $i_F(\mathbf{x})$, training data $(\mathbf{x}_i)$, support vectors $(\mathbf{x}_{SV})_i$ and separation hyperboundaries (hypersurfaces).

Note the different character of a $d(\mathbf{x})$, $i_F(\mathbf{x})$ and separation boundaries in the two graphs given below. However, in both graphs all the data are correctly classified. Fig. 2.15 shows the resulting functions for the Gaussian kernel functions, while Fig. 2.16 presents the solution for a complete second order polynomial kernel. Below, we present the analytical derivation of the (saddle like) decision function in the later (polynomial kernel) case. The ana-

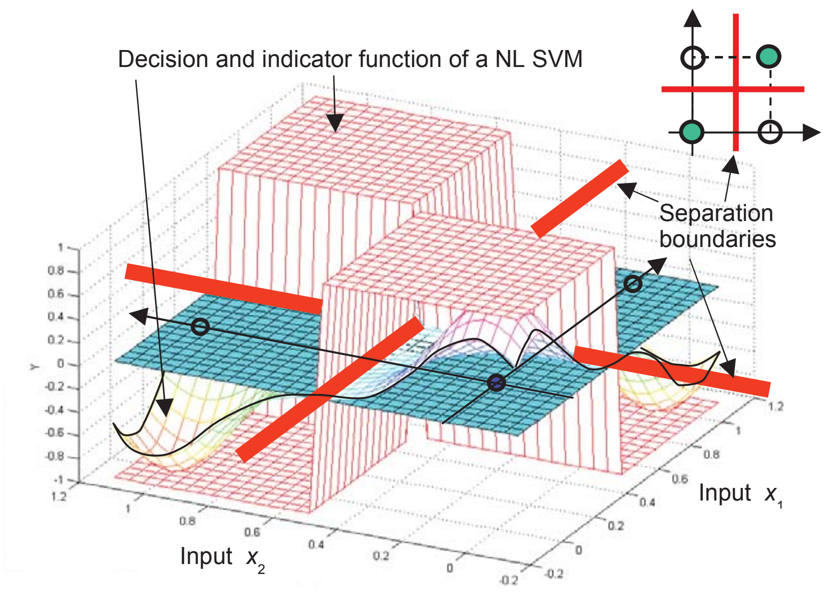


Fig. 2.15. XOR problem. Kernel functions (*2-D Gaussians*) are not shown. The nonlinear decision function, the nonlinear indicator function and the separation boundaries are shown. All four data are chosen as support vectors.

lytic solution to the Fig. 2.16 for the second order polynomial kernel (i.e., for $(\mathbf{x}_i^T\mathbf{x}_j + 1)^2 = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)$, where

$$\boldsymbol{\Phi}(\mathbf{x}) = [1 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2 \;\; \sqrt{2}x_1x_2 \;\; x_1^2 \; x_2^2],$$

no explicit bias and $C = \infty$) goes as follows. Inputs and desired outputs are,

$$\mathbf{x} = \begin{bmatrix} 0\;1\;1\;0 \\ 0\;1\;0\;1 \end{bmatrix}^T, \quad \mathbf{y} = \mathbf{d} = [1\;1\;-1\;-1]^T.$$

The dual Lagrangian (2.36a) has the Hessian matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 9 & -4 & -4 \\ -1 & -4 & 4 & 1 \\ -1 & -4 & 1 & 4 \end{bmatrix}$$

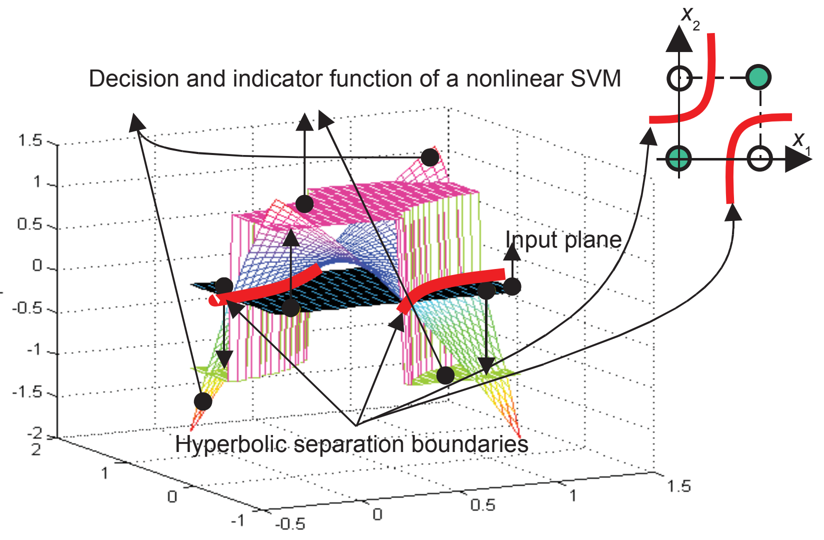


Fig. 2.16. XOR problem. Kernel function is a *2-D polynomial*. The nonlinear decision function, the nonlinear indicator function and the separation boundaries are shown. All four data are support vectors.

The optimal solution can be obtained by taking the derivative of L_d with respect to dual variables $\alpha_i (i = 1, 4)$ and by solving the resulting linear system of equations taking into account the constraints, see [84]. The solution to

$$\begin{aligned} \alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 &= 1, \\ \alpha_1 + 9\alpha_2 - 4\alpha_3 - 4\alpha_4 &= 1, \\ -\alpha_1 - 4\alpha_2 + 4\alpha_3 + \alpha_4 &= 1, \\ -\alpha_1 - 4\alpha_2 + \alpha_3 + 4\alpha_4 &= 1, \end{aligned}$$

subject to $\alpha_i > 0, (i = 1, 4)$, is $\alpha_1 = 4.3333, \alpha_2 = 2.0000, \alpha_3 = 2.6667$ and $\alpha_4 = 2.6667$. The decision function in a 3-D space is

$$\begin{aligned} d(\mathbf{x}) &= \sum\nolimits_{i=1}^{4} y_i \alpha_i \mathbf{\Phi}^T(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x}) \\ &= (4.3333 \left[1\ 0\ 0\ 0\ 0\ 0\right] + 2 \left[1\ \sqrt{2}\ \sqrt{2}\ \sqrt{2}\ 1\ 1\right] \\ &\quad -2.6667 \left[1\ \sqrt{2}\ 0\ 0\ 1\ 0\right] - 2.6667 \left[1\ 0\ \sqrt{2}\ 0\ 0\ 1\right])\mathbf{\Phi}(\mathbf{x}) \end{aligned}$$

$$= [1\ \ \text{- } 0.942\ \ \text{- } 0.942\ \ 2.828\ \ \text{- } 0.667\ \ \text{- } 0.667]\ [1\ \sqrt{2}x_1\ \sqrt{2}x_2\ \sqrt{2}x_1x_2\ x_1^2\ x_2^2]^T$$

and finally

$$d(\mathbf{x}) = 1 - 1.3335x_1 - 1.3335x_2 + 4x_1x_2 - 0.6667x_1^2 - 0.6667x_2^2$$

It is easy to check that the values of $d(\mathbf{x})$ for all the training inputs in $\mathbf{x}$ equal the desired values in $\mathbf{d}$. The $d(\mathbf{x})$ is the saddle-like function shown in Fig. 2.16.

Here we have shown the derivation of an expression for $d(\mathbf{x})$ by using explicitly a mapping $\boldsymbol{\Phi}$. Again, we do not have to know what mapping $\boldsymbol{\Phi}$ is at all. By using kernels in input space, we calculate a *scalar product* required in a (*possibly high dimensional*) *feature space* and we avoid mapping $\boldsymbol{\Phi}(\mathbf{x})$. This is known as kernel 'trick'. It can also be useful to remember that the way in which the kernel 'trick' was applied in designing an SVM can be utilized in all other algorithms that depend on the scalar product (e.g., in principal component analysis or in the nearest neighbor procedure) .

2.2.4 Regression by Support Vector Machines

In the regression , we estimate the functional dependence of the dependent (output) variable $y \in \Re$ on an m-dimensional input variable $\mathbf{x}$. Thus, unlike in pattern recognition problems (where the desired outputs y_i are discrete values e.g., Boolean) we deal with *real valued* functions and we model an $\Re^m$ to $\Re^1$ mapping here. Same as in the case of classification, this will be achieved by training the SVM model on a training data set first. Interestingly and importantly, a learning stage will end in the same shape of a dual Lagrangian as in classification, only difference being in a dimensionalities of the Hessian matrix and corresponding vectors which are of a double size now e.g., $\mathbf{H}$ is a $(2n, 2n)$ matrix. Initially developed for solving classification problems, SV techniques can be successfully applied in regression, i.e., for a functional approximation problems [45, 142]. The general regression learning problem is set as follows - the learning machine is given n training data from which it attempts to learn the input-output relationship (dependency, mapping or function) $f(\mathbf{x})$. A training data set $\mathcal{X} = [\mathbf{x}(i), y(i)] \in \Re^m \times \Re, i = 1, ..., n$ consists of n pairs $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$, where the inputs $\mathbf{x}$ are m-dimensional vectors $\mathbf{x} \in \Re^m$ and system responses $y \in \Re$, are continuous values. We introduce all the relevant and necessary concepts of SVM's regression in a gentle way starting again with a *linear regression hyperplane* $f(\mathbf{x}, \mathbf{w})$ given as

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} + b. \tag{2.39}$$

In the case of SVM's regression, we measure the *error of approximation* instead of the margin used in classification. The most important difference in respect to classic regression is that we use a novel loss (error) functions here. This is the Vapnik's *linear loss function* with ε-insensitivity zone defined as

$$E(\mathbf{x}, y, f) = |y - f(\mathbf{x}, \mathbf{w})|_\varepsilon = \begin{cases} 0 & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \le \varepsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon & \text{otherwise} \end{cases}, \tag{2.40a}$$

or as,

$$E(\mathbf{x}, y, f) = \max(0, |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon). \tag{2.40b}$$

Thus, the loss is equal to zero if the difference between the predicted $f(\mathbf{x}_i, \mathbf{w})$ and the measured value y_i is less than ε. In contrast, if the difference is larger than ε, this difference is used as the error. Vapnik's ε-insensitivity loss function (2.40) defines an ε tube as shown in Fig. 2.18. If the predicted value is within the tube, the loss (error or cost) is zero. For all other predicted points outside the tube, the loss equals the magnitude of the difference between the predicted value and the radius ε of the tube. The two classic error

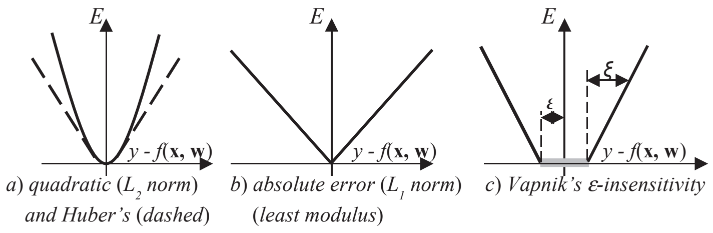


Fig. 2.17. Loss (error) functions.

functions are: a square error, i.e., L_2 norm $(y - f)^2$, as well as an absolute error, i.e., L_1 norm, least modulus $|y - f|$ introduced by Yugoslav scientist Rudjer Boskovic in 18th century [48]. The latter error function is related to Huber's error function. An application of Huber's error function results in a robust regression. It is the most reliable technique if nothing specific is known about the model of a noise. We do not present Huber's loss function here in analytic form. Instead, we show it by a dashed curve in Fig. 2.17a. In addition, Fig. 2.17 shows typical shapes of all mentioned error (loss) functions above.

Note that for $\varepsilon = 0$, Vapnik's loss function equals a least modulus function. Typical graph of a (nonlinear) regression problem as well as all relevant mathematical variables and objects required in, or resulted from, a learning unknown coefficients w_i are shown in Fig. 2.18.

We will formulate an SVM regression's algorithm for the linear case first and then, for the sake of a NL model design, we will apply mapping to a feature space, utilize the kernel 'trick' and construct a nonlinear regression hypersurface. This is actually the same order of presentation as in classification tasks. Here, for the regression, we 'measure' the empirical error term R_{emp} by Vapnik's ε-insensitivity loss function given by (2.40) and shown in Fig. 2.17c (while the minimization of the confidence term Ω will be realized through a minimization of $\mathbf{w}^T\mathbf{w}$ again). The empirical risk is given as

$$R^{\varepsilon}_{emp}(\mathbf{w}, b) = \frac{1}{n}\sum\nolimits_{i=1}^{n} \left|y_i - \mathbf{w}^T\mathbf{x}_i - b\right|_{\varepsilon} \tag{2.41}$$

As in classification, we try to minimize both the empirical risk R^{ε}_{emp} and $\|\mathbf{w}\|^2$

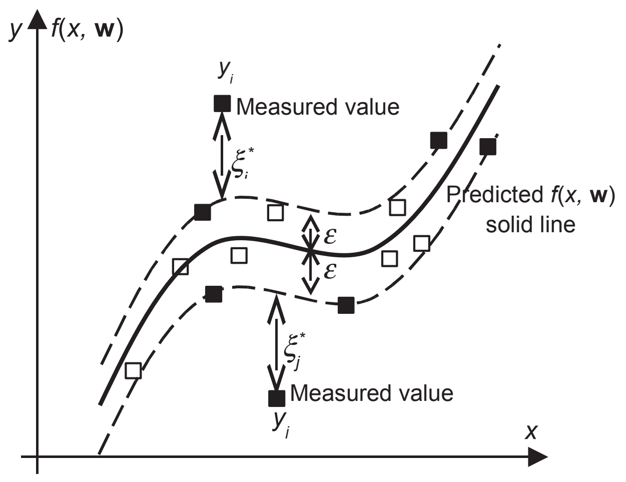
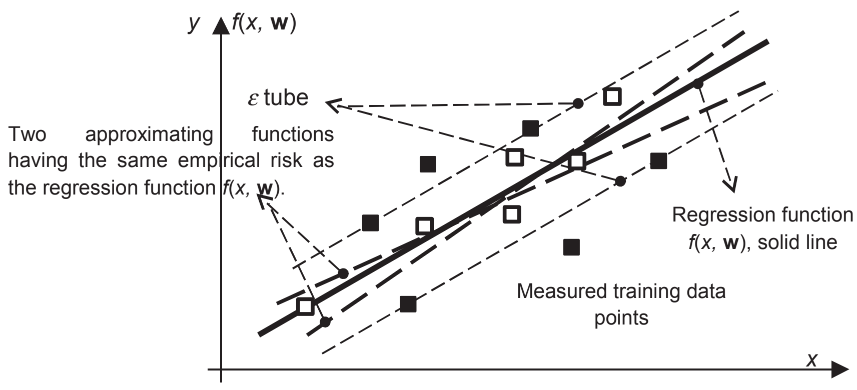


Fig. 2.18. The parameters used in (1-D) support vector regression. Filled squares data are support vectors. Hence, SVs can appear only on the tube boundary or outside the tube.



y
$f(x, \mathbf{w})$
ε tube
Two approximating functions having the same empirical risk as the regression function $f(x, \mathbf{w})$.
Regression function $f(x, \mathbf{w})$, solid line
Measured training data points
x

Fig. 2.19. Two linear approximations inside an ε tube (dashed lines) have the same empirical risk R^{ε}_{emp} on the training data as the regression function (solid line).

simultaneously. Thus, we construct a linear regression hyperplane $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} + b$ by minimizing

$$R = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} |y_i - f(\mathbf{x}_i, \mathbf{w})|_\varepsilon. \tag{2.42}$$

Note that the last expression resembles the ridge regression scheme. However, we use Vapnik's ε-insensitivity loss function instead of a squared error now. From (2.40) and Fig. 2.18 it follows that for all training data outside an ε-tube,

$$|y - f(\mathbf{x}, \mathbf{w})| - \varepsilon = \xi \text{ for data 'above' an } \varepsilon\text{-tube, or} \tag{2.43a}$$

$$|y - f(\mathbf{x}, \mathbf{w})| - \varepsilon = \xi^* \text{ for data 'below' an } \varepsilon\text{-tube, or} \tag{2.43b}$$

Thus, minimizing the risk R above equals the minimization of the following risk

$$R_{\mathbf{w},\xi,\xi^*} = \left[\frac{1}{2}||\mathbf{w}||^2 + C\left(\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \xi_i^*\right)\right] \tag{2.44a}$$

under constraints

$$y_i - \mathbf{w}^T\mathbf{x}_i - b \leq \varepsilon + \xi_i, \quad i = 1, \dots, n \tag{2.44b}$$

$$\mathbf{w}^T\mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i^*, \quad i = 1, \dots, n \tag{2.44c}$$

$$\xi_i \geq 0, \quad \xi_i^* \geq 0 \quad i = 1, \dots, n \tag{2.44d}$$

where ξ_i and ξ_i^* are slack variables shown in Fig. 2.18 for data points 'above' or 'below' the ε-tube respectively. Both slack variables are positive values. Lagrange multipliers α_i and α_i^* (that will be introduced during the minimization below) related to the first two sets of inequalities above, will be nonzero values for training points 'above' and 'below' an ε-tube respectively. Because no training data can be on both sides of the tube, either α_i or α_i^* will be nonzero. For data points inside the tube, both multipliers will be equal to zero. Thus $\alpha_i\alpha_i^* = 0$.

Note also that the constant C that influences a trade-off between an approximation error and the weight vector norm $\|\mathbf{w}\|$ is a design parameter that is chosen by the user. An increase in C penalizes larger errors i.e., it forces ξ_i and ξ_i^* to be small. This leads to an approximation error decrease which is achieved only by increasing the weight vector norm $\|\mathbf{w}\|$. However, an increase in $\|\mathbf{w}\|$ increases the confidence term Ω and does not guarantee a small generalization performance of a model. Another design parameter which is chosen by the user is the required precision embodied in an ε value that defines the size of an ε-tube. The choice of ε value is easier than the choice of C and it is given as either maximally allowed or some given or desired percentage of the output values y_i (say, $\varepsilon = 0.1$ of the mean value of $\mathbf{y}$).

Similar to procedures applied in the SV classifiers' design, we solve the constrained optimization problem above by forming a *primal variables* Lagrangian as follows,

$$\begin{aligned} L_p(\mathbf{w}, b, \xi_i, \xi_i^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*) = & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum\nolimits_{i=1}^n (\xi_i + \xi_i^*) \\ & - \sum\nolimits_{i=1}^n (\beta_i^*\xi_i^* + \beta_i\xi_i) \\ & - \sum\nolimits_{i=1}^n \alpha_i \left[\mathbf{w}^T\mathbf{x}_i + b - y_i + \varepsilon + \xi_i\right] \\ & - \sum\nolimits_{i=1}^n \alpha_i^* \left[y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i - b + \varepsilon + \xi_i^*\right] \end{aligned} \tag{2.45}$$

A primal variables Lagrangian $L_p(\mathbf{w}, b, \xi_i, \xi_i^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*)$ has to be *minimized* with respect to primal variables $\mathbf{w}, b, \xi_i$ and ξ_i^* and *maximized* with respect to nonnegative Lagrange multipliers $\alpha, \alpha_i^*, \beta_i$ and β_i^*. Hence, the function has the saddle point at the optimal solution $(\mathbf{w}_o, b_o, \xi_{io}, \xi_{io}^*)$ to the original problem. At the optimal solution the partial derivatives of L_p in respect to primal variables vanishes. Namely,

$$\frac{\partial L_p(\mathbf{w}_o, b_o, \xi_{io}, \xi_{io}^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*)}{\partial \mathbf{w}} = \mathbf{w}_o - \sum\nolimits_{i=1}^n (\alpha_i - \alpha_i^*)\mathbf{x}_i = 0, \tag{2.46}$$

$$\frac{\partial L_p(\mathbf{w}_o, b_o, \xi_{io}, \xi_{io}^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*)}{\partial b} = \sum\nolimits_{i=1}^n (\alpha_i - \alpha_i^*) = 0, \tag{2.47}$$

$$\frac{\partial L_p(\mathbf{w}_o, b_o, \xi_{io}, \xi_{io}^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*)}{\partial \xi_i} = C - \alpha_i - \beta_i = 0, \tag{2.48}$$

$$\frac{\partial L_p(\mathbf{w}_o, b_o, \xi_{io}, \xi_{io}^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*)}{\partial \xi_i^*} = C - \alpha_i^* - \beta_i^* = 0. \tag{2.49}$$

Substituting the KKT above into the primal L_p given in (2.45), we arrive at the problem of the *maximization of a dual variables Lagrangian* $L_d(\alpha, \alpha^*)$ below,

$$\begin{aligned} L_d(\alpha_i, \alpha_i^*) = & -\frac{1}{2}\sum_{i,j=1}^n (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\mathbf{x}_i^T\mathbf{x}_j - \varepsilon\sum_{i=1}^n (\alpha_i + \alpha_i^*) \\ & + \sum_{i=1}^n (\alpha_i - \alpha_i^*)y_i \\ = & -\frac{1}{2}\sum_{i,j=1}^n (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\mathbf{x}_i^T\mathbf{x}_j - \sum_{i=1}^n (\varepsilon - y_i)\alpha_i \\ & - \sum_{i=1}^n (\varepsilon + y_i)\,\alpha_i^* \end{aligned} \tag{2.50}$$

subject to constraints

$$\sum\nolimits_{i=1}^n \alpha_i^* = \sum\nolimits_{i=1}^n \alpha_i \text{ or } \sum\nolimits_{i=1}^n (\alpha_i - \alpha_i^*) = 0, \tag{2.51a}$$

$$0 \le \alpha_i \le C \quad i = 1, \ldots, n, \tag{2.51b}$$

$$0 \le \alpha_i^* \le C \quad i = 1, \ldots, n. \tag{2.51c}$$

Note that the dual variables Lagrangian $L_d(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*)$ is expressed in terms of Lagrange multipliers α_i and α_i^* only. However, the size of the problem, with respect to the size of an SV classifier design task, is doubled now. There are $2n$ unknown dual variables (n α_i-s and n α_i^*-s) for a linear regression and the Hessian matrix $\mathbf{H}$ of the quadratic optimization problem in the case of regression is a $(2n, 2n)$ matrix. The standard quadratic optimization problem above can be expressed in a *matrix notation* and formulated as follows:

$$\min L_d(\boldsymbol{\alpha}) = 0.5\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} + \mathbf{p}\boldsymbol{\alpha}, \tag{2.52}$$

subject to (2.51) where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n, \alpha_1^*, \alpha_2^*, \ldots, \alpha_n^*]^T$, $\boldsymbol{H} = [\mathbf{G} \quad -\mathbf{G}; -\mathbf{G} \quad \mathbf{G}]$, $\mathbf{G}$ is an (n,n) matrix with entries $G_{ij} = [\mathbf{x}_i^T\mathbf{x}_j]$ in a linear regression and $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ for the nonlinear one, and $\mathbf{p} = [\varepsilon - y_1, \varepsilon - y_2, \ldots, \varepsilon - y_n, \varepsilon + y_1, \varepsilon + y_2, \ldots, \varepsilon + y_n]$ (Note that G_{ij}, as given above, is a badly conditioned matrix and we rather use $G_{ij} = [\mathbf{x}_i^T\mathbf{x}_j + 1]$ instead). Equation (2.52) is written in the form of a standard optimization routine that typically *minimizes* given objective function subject to the same constraints (2.51).

The learning stage results in n Lagrange multiplier pairs (α_i, α_i^*). After the learning, the number of SVs is equal to the number of nonzero α_i and α_i^*. However, this number does not depend on the dimensionality of input space and this is particularly important when working in very high dimensional spaces. Because at least one element of each pair (α_i, α_i^*), $i = 1,\ n$, is zero, the product of α_i and α_i^* is always zero,i.e. $\alpha_i\alpha_i^* = 0$. At the optimal solution the following KKT complementarity conditions must be fulfilled

$$\alpha_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i\ +\ b\ -\ y_i\ +\ \varepsilon\ +\ \xi_i\right) = 0, \tag{2.53a}$$

$$\alpha_i^*\left(\text{-}\ \mathbf{w}^{\mathrm{T}}\mathbf{x}_i\ \text{-}\ b + y_i\ +\ \varepsilon\ +\ \xi_i^*\right) = 0, \tag{2.53b}$$

$$\beta_i\,\xi_i = (C - \alpha_i)\,\xi_i = 0, \tag{2.53c}$$

$$\beta_i^*\,\xi_i^* = (C - \alpha_i^*)\,\xi_i^* = 0. \tag{2.53d}$$

(2.53c) states that for $0 < \alpha_i < C,\ \xi_i = 0$ holds. Similarly, from (2.53d) follows that for $0 < \alpha_i^* < C,\ \xi_i^* = 0$ and, for $0 < \alpha_i, \alpha_i^* < C$, from (2.53a) and (2.53b) follows,

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_i\ +\ b\ -\ y_i\ +\ \varepsilon = 0, \tag{2.54a}$$

$$\text{-}\ \mathbf{w}^{\mathrm{T}}\mathbf{x}_i\ \text{-}\ b + y_i\ +\ \varepsilon\ = 0. \tag{2.54b}$$

Thus, for all the data points fulfilling $y - f(\mathbf{x}) = +\varepsilon$, dual variables α_i must be between 0 and C, or $0 < \alpha_i < C$, and for the ones satisfying $y - f(\mathbf{x}) = -\varepsilon$, α_i^* take on values $0 < \alpha_i^* < C$. These data points are called the *free* (or *unbounded*) support vectors. They allow computing the value of the bias term b as given below

$$b = y_i - \mathbf{w}^T\mathbf{x}_i - \varepsilon \text{ for } 0 < \alpha_i < C, \tag{2.55}$$

$$b = y_i - \mathbf{w}^T\mathbf{x}_i + \varepsilon \text{ for } 0 < \alpha_i^* < C. \tag{2.56}$$

The calculation of a bias term b is numerically very sensitive, and it is better to compute the bias b by averaging over all the *free* support vector data points.

The final observation follows from (2.53c) and (2.53d) and it tells that for all the data points outside the ε-tube, i.e., when both $\xi_i > 0$ and $\xi_i^* > 0$, both α_i and α_i^* equal C, i.e., $\alpha_i = C$ for the points above the tube and $\alpha_i^* = C$ for the points below it. These data are the so-called *bounded* support vectors. Also, for all the training data points within the tube, or when $|y - f(\mathbf{x})| < \varepsilon$, both α_i and α_i^* equal zero and they are neither the support vectors nor do they construct the decision function $f(\mathbf{x})$.

After calculation of Lagrange multipliers α_i and α_i^*, using (2.46) we can find an optimal (desired) weight vector of the *regression hyperplane* as

$$\mathbf{w}_o = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*)\mathbf{x}_i. \tag{2.57}$$

The best regression hyperplane obtained is given by

$$f(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}_o^T \mathbf{x} + b = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*)\mathbf{x}_i^T \mathbf{x} + b. \tag{2.58}$$

More interesting, more common and the most challenging problem is to aim at solving the *nonlinear regression tasks.* A generalization to nonlinear regression is performed in the same way the nonlinear classifier is developed from the linear one, i.e., by carrying the mapping to the feature space, or by using kernel functions instead of performing the complete mapping which is usually of extremely high (possibly of an infinite) dimension. Thus, the nonlinear regression function in an input space will be devised by considering a linear regression hyperplane in the *feature space.*

We use the same basic idea in designing SV machines for creating a *nonlinear regression function.* First, a mapping of input vectors $\mathbf{x} \in \Re^m$ into vectors $\boldsymbol{\Phi}(x)$ of a higher dimensional *feature space* $\mathcal{S}$ (where $\boldsymbol{\Phi}$ represents mapping: $\Re^m \to \Re^s$) takes place and then, we solve a linear regression problem in this feature space. A mapping $\boldsymbol{\Phi}(\mathbf{x})$ is again the chosen in advance, or fixed, function. Note that an input space ($\mathbf{x}$-space) is spanned by components x_i of an input vector $\mathbf{x}$ and a feature space $\mathcal{S}$ ($\boldsymbol{\Phi}$-space) is spanned by components $\phi_i(\mathbf{x})$ of a vector $\boldsymbol{\Phi}(\mathbf{x})$. By performing such a mapping, we hope that in a $\boldsymbol{\Phi}$-space, our learning algorithm will be able to perform a linear regression hyperplane by applying the linear regression SVM formulation presented above. We also expect this approach to again lead to solving a quadratic optimization problem with inequality constraints in the feature space. The (linear in a feature space $\mathcal{S}$) solution for the regression hyperplane $f = \mathbf{w}^T \boldsymbol{\Phi}(\mathbf{x}) + b$, will create a nonlinear regressing hypersurface in the original input space. The most popular kernel functions are polynomials and RBF with Gaussian kernels. Both kernels are given in Table 2.2.

In the case of the nonlinear regression, the learning problem is again formulated as the maximization of a dual Lagrangian (2.52) with the Hessian matrix $\mathbf{H}$ structured in the same way as in a linear case, i.e. $\mathbf{H} = [\mathbf{G} \quad -\mathbf{G}; -\mathbf{G} \quad \mathbf{G}]$ but with the changed Grammian matrix $\mathbf{G}$ that is now given as

$$\mathbf{G} = \begin{bmatrix} G_{11} & \cdots & G_{1n} \\ \vdots & G_{ii} & \vdots \\ G_{n1} & \cdots & G_{nn} \end{bmatrix}$$

where the entries $G_{ij} = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j), i, j = 1,\ n$.

After calculating Lagrange multiplier vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$, we can find an optimal weighting vector of the *kernels expansion* as

$$\mathbf{v}_o = \boldsymbol{\alpha} - \boldsymbol{\alpha}^* \tag{2.59}$$

Note however the difference in respect to the linear regression where the expansion of a regression function is expressed by using the optimal weight vector $\mathbf{w}_o$. Here, in a NL SVMs' regression, the optimal weight vector $\mathbf{w}_o$ could often be of infinite dimension (which is the case if the Gaussian kernel is used). Consequently, we neither calculate $\mathbf{w}_o$ nor we have to express it in a closed form. Instead, we create the best nonlinear regression function by using the weighting vector $\mathbf{v}_o$ and the kernel (Grammian) matrix $\mathbf{G}$ as follows,

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{G}\mathbf{v}_0 + b \tag{2.60}$$

In fact, the last result follows from the very setting of the learning (optimizing) stage in a feature space where, in all the equations above from (2.44b) to (2.58), we replace $\mathbf{x}_i$ by the corresponding feature vector $\boldsymbol{\Phi}(\mathbf{x}_i)$. This leads to the following changes:

- instead $G_{ij} = \mathbf{x}_i^T\mathbf{x}_j$ we get $G_{ij} = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)$ and, by using the kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}_j)$, it follows that $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$.
- similarly, (2.57) and (2.58) change as follows:

$$\mathbf{w}_o = \sum\nolimits_{i=1}^{n} (\alpha_i - \alpha_i^*)\boldsymbol{\Phi}(\mathbf{x}_i) \quad \text{and,} \tag{2.61}$$

$$\begin{aligned} f(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}_o^T\boldsymbol{\Phi}(\mathbf{x}) + b &= \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{\Phi}^T(\mathbf{x}_i)\boldsymbol{\Phi}(\mathbf{x}) + b \\ &= \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(\mathbf{x}_i, \mathbf{x}) + b. \end{aligned} \tag{2.62}$$

If the bias term b is explicitly used as in (2.60) then, for a NL SVMs' regression, it can be calculated from the upper SVs as,

$$\begin{aligned} b &= y_i - \sum\nolimits_{j=1}^{N free\, upper\, SVs} (\alpha_j - \alpha_j^*)\mathbf{\Phi}^T(\mathbf{x}_j)\mathbf{\Phi}(\mathbf{x}_i) - \varepsilon \\ &= y_i - \sum\nolimits_{j=1}^{N free\, upper\, SVs} (\alpha_j - \alpha_j^*)K(\mathbf{x}_i, \mathbf{x}_j) - \varepsilon, \text{ for } 0 < \alpha_i < C \end{aligned} \tag{2.63}$$

or from the lower ones as,

$$\begin{aligned} b &= y_i - \sum\nolimits_{j=1}^{N free\, lower\, SVs} (\alpha_j - \alpha_j^*) \mathbf{\Phi}^T(\mathbf{x}_j)\mathbf{\Phi}(\mathbf{x}_i) + \varepsilon \\ &= y_i - \sum\nolimits_{j=1}^{N free\, lower\, SVs} (\alpha_j - \alpha_j^*) K(\mathbf{x}_i, \mathbf{x}_j) + \varepsilon, \text{ for } 0 < \alpha_i^* < C. \end{aligned} \tag{2.64}$$

Note that $\alpha_j^* = 0$ in (2.63) and so is $\alpha_j = 0$ in (2.64). Again, it is much better to calculate the bias term b by an averaging *over all* the *free* support vector data points.

There are a few learning parameters in constructing SV machines for regression. The three most relevant are the insensitivity zone ε, the penalty parameter C (that determines the trade-off between the training error and VC dimension of the model), and the shape parameters of the kernel function (variances of a Gaussian kernel, order of the polynomial, or the shape parameters of the inverse multiquadrics kernel function). All three parameters' sets should be selected by the user. To this end, the most popular method for their selection is a cross-validation. Unlike in a classification, for not too noisy data (primarily without huge outliers), the penalty parameter C could be set to infinity and the modeling can be controlled by changing the insensitivity zone ε and shape parameters only.

The *example* below shows how an increase in an insensitivity zone ε has smoothing effects on modeling highly noise polluted data. Increase in ε means a reduction in requirements on the accuracy of approximation. It decreases the number of SVs leading to higher data compression too. This can be readily followed in the lines and Fig. 2.20 below.

Example 2.5. The task here is to construct an SV machine for modeling measured data pairs. The underlying function (known to us but, not to the SVM) is a sinus function multiplied by the square one (i.e., $f(x) = x^2 \sin(x)$) and it is corrupted by 25% of normally distributed noise with a zero mean. Analyze the influence of an insensitivity zone ε on modeling quality and on a compression of data, meaning on the number of SVs. Fig. 2.20 shows that for a very noisy data a decrease of an insensitivity zone ε (i.e., shrinking of the tube shown by dashed line) approximates the noisy data points more closely. The related more and more wiggly shape of the regression function can be achieved only by including more and more support vectors. However, being good on the noisy training data points easily leads to an overfitting. The cross-validation should help in finding correct ε value, resulting in a regression function that filters the noise out but not the true dependency and which, consequently, approximate the underlying function as close as possible. The approximation function shown in Fig. 2.20 is created by 9 and 18 weighted Gaussian basis functions for $\varepsilon = 1$ and $\varepsilon = 0.75$ respectively. These supporting functions are not shown in the figure. However, the way how the learning algorithm selects SVs is an interesting property of support vector machines and in Fig. 2.21 we also present the supporting Gaussian functions.

Note that the selected Gaussians lie in the 'dynamic area' of the function in Fig. 2.21. Here, these areas are close to both the left hand and the right

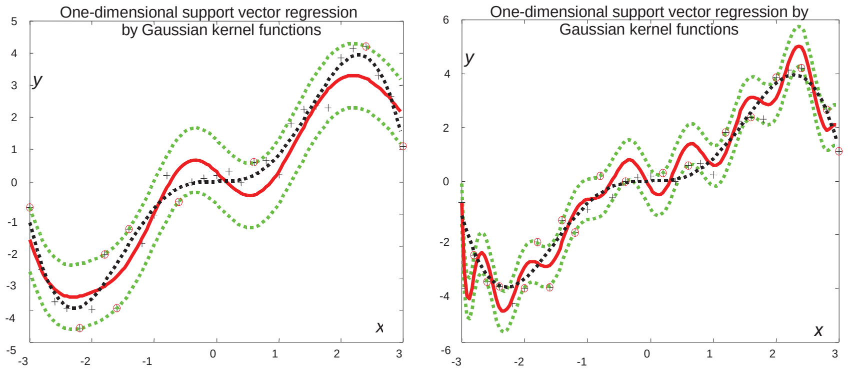


Fig. 2.20. The influence of an insensitivity zone ε on the model performance. A nonlinear SVM creates a regression function f with Gaussian kernels and models a highly polluted (25% noise) function $x^2 \sin(x)$ (dotted). 31 training data points (plus signs) are used. *Left*: $\varepsilon = 1$; 9 SVs are chosen (encircled plus signs). *Right*: $\varepsilon = 0.75$; the 18 chosen SVs produced a better approximation to noisy data and, consequently, there is the tendency of overfitting.

hand boundary. In the middle, the original function is pretty flat and there is no need to cover this part by supporting Gaussians. The learning algorithm realizes this fact and simply, it does not select any training data point in this area as a support vector. Note also that the Gaussians are not weighted in Fig. 2.21 , and they all have the peak value of 1. The standard deviation of Gaussians is chosen in order to see Gaussian supporting functions better. Here, in Fig. 2.21, $\sigma = 0.6$. Such a choice is due the fact that for the larger σ values the basis functions are rather flat and the supporting functions are covering the whole domain as the broad umbrellas. For very big variances one can't distinguish them visually. Hence, one can't see the true, bell shaped, basis functions for the large variances.

2.3 Implementation Issues

In both the classification and the regression the learning problem boils down to solving the QP problem subject to the so-called 'box-constraints' and to the equality constraint in the case that a model with a bias term b is used. The SV training works almost perfectly for not too large data basis. However, when the number of data points is large (say $n > 2,000$) the QP problem becomes extremely difficult to solve with standard QP solvers and methods. For example, a classification training set of 50,000 examples amounts to a Hessian matrix $\mathbf{H}$ with $2.5 * 10^9$ (2.5 billion) elements. Using an 8-byte floating-point representation we need 20,000 Megabytes = 20 Gigabytes of memory [109].

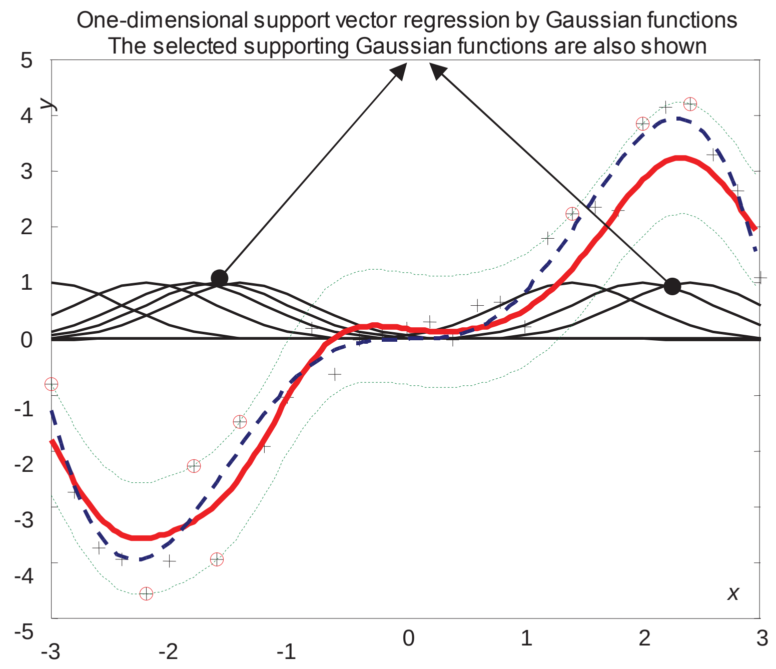


Fig. 2.21. Regression function f created as the sum of 8 weighted Gaussian kernels. A standard deviation of Gaussian bells $\sigma = 0.6$. Original function (dashed line) is $x^2 \sin(x)$ and it is corrupted by 25% noise. 31 training data points are shown as plus signs. Data points selected as the SVs are encircled. The 8 selected supporting Gaussian functions are centered at these data points.

This cannot be easily fit into memory of present standard computers, and this is the single basic disadvantage of the SVM method. There are three approaches that resolve the QP for large data sets. Vapnik in [144] proposed the chunking method that is the decomposition approach. Another decomposition approach is suggested in [109]. The sequential minimal optimization [115] algorithm is of different character and it seems to be an 'error back propagation' for an SVM learning. A systematic exposition of these various techniques is not given here, as all three would require a lot of space. However, the interested reader can find a description and discussion about the algorithms mentioned above in next chapter and [84, 150]. The Vogt and Kecman's chapter [150] discusses the application of an active set algorithm in solving small to medium sized QP problems. For such data sets and when the high precision is required the active set approach in solving QP problems seems to be superior to other approaches (notably to the interior point methods and to the sequential minimal optimization (SMO) algorithm). Next chapter introduces the efficient iterative single data algorithm (ISDA) for solving huge data sets (say more than 100,000 or 500,000 or over 1 million training data pairs). It seems that ISDA is the fastest algorithm at the moment for such large data sets still ensuring the convergence to the global minimum (see the comparisons with SMO in Sect.3.4). This means that the ISDA provides the exact, and not the approximate, solution to original dual problem.

Let us conclude the presentation of SVMs part by summarizing the basic constructive steps that lead to the SV machine.

A training and design of a support vector machine is an iterative algorithm and it involves the following steps:

1. define your problem as the classification or as the regression one,
2. preprocess your input data: select the most relevant features, scale the data between [-1, 1], or to the ones having zero mean and variances equal to one, check for possible outliers (strange data points),
3. select the kernel function that determines the hypothesis space of the decision and regression function in the classification and regression problems respectively,
4. select the 'shape', i.e., 'smoothing' parameter of the kernel function (for example, polynomial degree for polynomials and variances of the Gaussian RBF kernels respectively),
5. choose the penalty factor C and, in the regression, select the desired accuracy by defining the insensitivity zone ε too,
6. solve the QP problem in n and $2n$ variables in the case of classification and regression problems respectively,
7. validate the model obtained on some previously, during the training, unseen test data, and if not pleased iterate between steps 4 (or, eventually 3) and 7.

The optimizing part 6 is computationally extremely demanding. First, the Hessian matrix $\mathbf{H}$ scales with the size of a data set - it is an (n,n) and an $(2n, 2n)$ matrix in classification and regression respectively. Second, unlike in classic original QP problems $\mathbf{H}$ is very dense matrix and it is usually badly conditioned requiring a regularization before any numeric operation. Regularization means an addition of a small number to the diagonal elements of $\mathbf{H}$. Luckily, there are many reliable and fast QP solvers. A simple search on an Internet will reveal many of them. Particularly, in addition to the classic ones such as MINOS or LOQO for example, there are many more free QP solvers designed specially for the SVMs. The most popular ones are - the LIBSVM, SVMlight, SVM Torch, mySVM and SVM Fu. All of them can be downloaded from their corresponding sites.A user friendly software implementation of the ISDA that can handle huge data set can be download from the web site of this book `www.learning-from-data.com` . Good educational software in MATLAB named LEARNSC, with a very good graphic presentations of all relevant objects in a SVM modeling, can be downloaded from the second author's book site `www.support-vector.ws` too.

Finally we mention that there are many alternative formulations and approaches to the QP based SVMs described above. Notably, they are the linear programming SVMs [94, 53, 128, 59, 62, 83, 81, 82], μ-SVMs [123] and least squares support vector machines [134]. Their description is far beyond this chapter and the curious readers are referred to references given above.