53. Support Vector Machines for Data Modeling with Software Engineering Applications

This chapter presents the basic principles of support vector machines (SVM) and their construction algorithms from an applications perspective. The chapter is organized into three parts. The first part consists of Sects. 53.2 and 53.3. In Sect. 53.2 we describe the data modeling issues in classification and prediction problems. In Sect. 53.3 we give an overview of a support vector machine (SVM) with an emphasis on its conceptual underpinnings. In the second part, consisting of Sects. 53.4–53.9, we present a detailed discussion of the support vector machine for constructing classification and prediction models. Sections 53.4 and 53.5 describe the basic ideas behind a SVM and are the key sections. Section 53.4 discusses the construction of optimal hyperplane for the simple case of linearly separable patterns and its relationship to the Vapnik–Chervonenkis dimension. A detailed example is used for illustration. The relatively more difficult case of nonseparable patterns is discussed in Sect. 53.5. The use of inner product kernels for nonlinear classifiers is described in Sect. 53.6 and is illustrated via an example. Nonlinear regression is described in Sect. 53.7. The issue of specifying SVM hyperparameters is addressed in Sect. 53.8, and a generic SVM construction flowchart is presented in Sect. 53.9. The third part details two case studies. In Sect. 53.10 we present the results of a detailed analysis of module-level NASA data for developing classification models. In Sect. 53.11, effort data from 75 projects is used to obtain nonlinear prediction models and analyze their performance. Section 53.12 presents some concluding remarks, current activities in support vector machines, and some guidelines for further reading.

53.1 Overview

The problem of predictive data modeling is of both academic and practical interest in many engineering and scientific disciplines, including software engineering. It is the process of building a model of the input–output relationship from historical or experimental data. This model is used to predict the output of a future occurrence for which only the input will be known. Such models have their roots in traditional statistics. However, recent advances in machine learning and related disciplines have been shifting focus away from statistical methods toward these approaches. In particular, a new type of learning machine, called a support vector machine (SVM), has gained prominence within the last decade. These machines are based on statistical learn-

ing theory, possess some very nice properties, and have exhibited impressive performance in a wide range of applications.

In this chapter we present the basic principles of support vector machines and their construction algorithms, with emphasis on applications. In Sect. 53.2 we formally describe data modeling for classification and prediction in software engineering and some important considerations in model development. Support vector machines are introduced in Sect. 53.3. Section 53.4 deals with the case of developing maximal margin classifiers for linearly separable classes and their relationship to the important concept of the Vapnik–Chervonenkis (VC) dimension. An illustrative example is used to explain the computations involved. Next, the more difficult problem of nonseparable patterns is presented in Sect. 53.5. Nonlinear classifiers using inner-product kernels are discussed in Sect. 53.6, and their computational steps are illustrated via an example. The development of nonlinear prediction models using the SVM algorithm is discussed in Sect. 53.7 and some comments about selecting SVM hyperparameters are summarized in Sect. 53.8. In Sect. 53.9, a generic SVM flow chart is presented to depict the development of classification and prediction models using SVM. In Sect. 53.10 a case study for module classification is detailed using public-domain software metrics data. Software effort-prediction using SVM nonlinear regression modeling is presented in Sect. 53.11 for some commercial software projects. A summary, a mention of current activities in SVM, and suggestions for further reading are included in Sect. 53.12.

For readers who want to get a general understanding of support vector machines, Sects. 53.2, 53.3, 53.4.1, 53.4.3, and 53.10 should be adequate. Section 53.6 is useful in understanding how SVM develops classifiers for practical classification problems. The remaining sections provide a description of related issues and SVM nonlinear regression.

53.2 Classification and Prediction in Software Engineering

53.2.1 Classification

A classification model is constructed from a set of data for which the attributes and the true classes are known and is employed to assign a class to a new object on the basis of its observed attributes or features. In software engineering, metric-based classification models are employed to classify a module as fault-prone or not fault-prone. Other terms that are used for module classes are high or low risk, high or low criticality, etc. An ability to identify fault-prone modules early in their lifecycle is of practical significance in software engineering because it enables allocation of appropriate resources such as additional reviews and testing to these modules and will thus enhance the quality of the delivered system.

Formally, the problem can be stated as follows. We have available metrics data ($\boldsymbol{x}_i$) about n modules and their corresponding class labels, y_i, denoted as

$$S = \left\{(\boldsymbol{x}_i, y_i), \boldsymbol{x}_i \in \mathbb{R}^d, y_i \in (-1, +1)\right\}, i = 1, 2, \ldots, n, \quad (53.1)$$

where d is the dimensionality of the input metrics data. Here the value of y_i as -1 or $+1$ is based on some threshold value of the numbers of faults found in the module. The modeling task is to construct a support vector classifier (SVC) that captures the unknown pattern between the inputs and the outputs, based on the evidence provided by the data. The developed model is then used to predict the criticality class of a future module whose software metrics values ($\boldsymbol{x}$) would be known, but not the class (y).

The objective is to develop a model with good accuracy on both the training data and on future predictions. A common measure used for assessing accuracy is the classification error (CE), which is the ratio of the number of modules classified correctly to the total number of modules. To seek an objective measure of the future performance of the classification model, one approach is to partition the sample data randomly into three sets: training, validation, and test, as shown in Fig. 53.1. The first set is used for model development, i. e., for training.

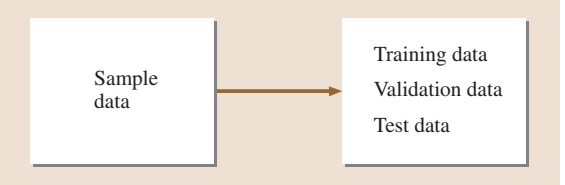


Fig. 53.1 Sample-data partitioning

The candidate models are evaluated on the validation data and usually the model with the smallest CE is selected. Then, the selected model is used to classify the data in the test set, and this error is considered to be an objective measure of future performance, or generalization error, of the selected model. The relative sizes of these sets are application-dependent. However, 50% of the data is often used for training, 25% for validation, and 25% for testing. This is called the *hold out* approach. Sometimes, only two subsets are created, training and the test, and the test set is used for model selection.

If the data set is small, the generalization error is estimated using cross validation. For k-fold cross validation (KCV), the data set is randomly divided into almost equal k sets and the model is developed from $(k-1)$ sets and tested on the omitted k-th set. The process is repeated k times, leaving a different set out each time. The average CE on the k omitted sets is called the *KCV error* of the model. Commonly used values for k are five and ten. Sometimes, we use $k=1$. Then, the estimated error is called the *leave one out* (*Loo*) error.

53.2.2 Prediction

The development of such models can also be seen as finding input–output mapping, but now between the project features and the effort. This problem can be stated as follows. Given some input and output data about previous projects, find a suitable functional relationship between them. Suppose we are given a training data set,

$$S = \left\{ (\boldsymbol{x}_i, y_i), \boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R} \right\}, i = 1, 2, \ldots, n \,, \tag{53.2}$$

where $\boldsymbol{x}$ are the software project features, d is the number of features, y is the effort, and n is the number of projects in the data set. Then, the goal is to find a function or mapping that maps $\boldsymbol{x}_i$ to y_i, $i = 1, \ldots, n$, such that the training and generalization errors are small. Two error evaluation measures commonly employed in software effort research are the mean magnitude of relative error (MMRE) and PRED(25), defined as follows:

$$\text{MMRE} = \frac{1}{n} \sum_{i=1}^{n} \frac{\left| \text{Actual effort}, y_i - \text{Predicted effort}, \hat{y}_i \right|}{\text{Actual effort}, y_i} \,; \tag{53.3}$$

$$\text{PRED}\,(25) = \frac{\text{Number of estimates within 25\% of actual } y}{n} \,. \tag{53.4}$$

These are measured for the model using the training data set and evaluated on the test data, usually by the Loo cross validation, since the number of projects used for effort estimation is generally small. Note that MMRE is a measure of error, while PRED(25) is a measure of accuracy. Therefore, we seek low MMRE and high PRED(25) values. The model performance on test data is used as a measure of its generalization ability, that is, a measure of the predictive accuracy on future projects for which the effort would be estimated.

53.3 Support Vector Machines

In this section we provide a very brief introduction to support vector machines. Consider the classification and regression tasks we just discussed. The statistical techniques for dealing with these require a knowledge of the underlying distribution. In many practical situations this distribution is not known and has to be assumed. Due to this restriction we seek alternative, distribution-free procedures. A support vector machine provides such a procedure that is relatively new and has some very nice properties. In particular, SVM is a learning procedure that can be used to derive learning machines such as polynomial approximators, neural networks, and radial basis functions. The origins of SVMs are in statistical learning theory (SLT) and they represent an approximate implementation of the principle of structural risk minimization (SRM). These are discussed in-depth in [53.1, 2] and are beyond the scope of this chapter. Briefly, the SRM principle is based on the fact that the test error of a learning machine, that is, the generalization error, is bounded by the sum of two quantities. The first is the training error, that is, the error on the training set. The second quantity is a function of the Vapnik–Chervonenkis (VC) dimension [53.1] and is called the VC bound. The underlying theory shows that a learning machine with a low VC bound will have a low generalization error. Thus, a low-VC classifier will have a low CE on new data. Similarly, a prediction model with a low VC bound will have a low MMRE or a high PRED(25)

when used for future predictions. Consider the classification problem for a set of separable patterns. For this case, SVM derives a learning machine with a training error of zero. It also minimizes the second term.

However, in practice, the VC dimension of nonlinear classifiers or predictors, such as radial basis functions, cannot be accurately estimated and hence the results of this theory cannot be precisely implemented. That is why the statement above about SVM used the words approximate implementation. What is discussed in subsequent sections is a practical realization of this theory. We first show how the SVM develops classifiers for the separable case. SVM algorithms are then derived to deal with the more difficult cases of nonseparable patterns. These ideas are then extended to nonlinear prediction models.

53.4 Linearly Separable Patterns

This is the simplest classification task and was the first to be introduced for support vector machines (SVM) [53.1]. It is applicable for data that are linearly separable in the input space. Although not very useful for many real-world applications, it helps focus on the key aspects for developing SVM classifiers for more complex applications. We are given training data as in (53.1). We seek to determine a decision function D such that $D(\boldsymbol{x}) = y$, where y is the class label of data point $\boldsymbol{x}$. The goal is to find a D that minimizes generalization error. The decision boundary is given by the hyperplane

$$D(\boldsymbol{x}) : \boldsymbol{w}^T \boldsymbol{x} + b = 0 \,, \tag{53.5}$$

where $\boldsymbol{w}$ is a d-dimensional weight vector, and b is the bias term. If the data is separable, it will be correctly classified, i. e., for $i = 1, 2, \ldots, n$,

$$\begin{aligned} &\boldsymbol{w}^T \boldsymbol{x}_i + b \geq 0 \text{ for } y_i = +1 \\ \text{and} \quad &\boldsymbol{w}^T \boldsymbol{x}_i + b < 0 \text{ for } y_i = -1 \,. \end{aligned} \tag{53.6}$$

Separation between the decision function and the closest data point is called the *margin of separation* and equals $1/||\boldsymbol{w}||$. There can be many possible hyperplanes that separate the two classes. However, intuitively, the larger the margin, the lower the generalization error. This is so because points closest to the decision surface are harder to classify, and by providing a maximal separation between the two classes it should be easier to classify a new data point. For illustration, sample data points from two classes are shown in Fig. 53.2 with two margins, small and large. The boundaries with a larger margin can be seen to be preferable.

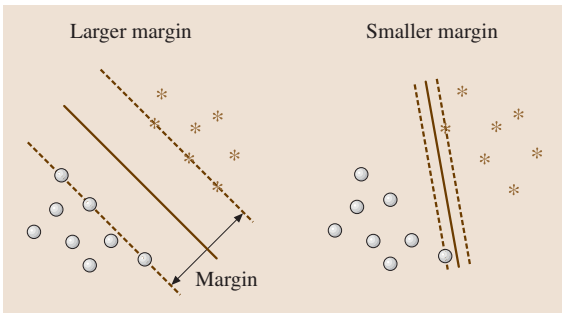


Fig. 53.2 Two linearly separable classes

53.4.1 Optimal Hyperplane

Given the linearly separable training data, the hyperplane that maximizes the margin is sought in the support vector machine formulation. Equivalently, we seek $(\boldsymbol{w}, b)$, which solves the following problem:

$$L\,(\mathrm{w}) = \min_{w\,b} \left(\frac{1}{2} \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} \right) \,. \tag{53.7}$$

$$\text{subject to} \quad y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i + b \right) \geq 1, \quad i = 1,\ 2, \ldots,\ n \,. \tag{53.8}$$

The aforementioned optimization problem may be solved by the method of Lagrange multipliers. Usually, the dual of this primal problem is solved. For details, see [53.3, 4]. The dual problem can then be stated as finding the Lagrange multipliers for the data in (53.1) that maximize the following objective function:

$$L_{\mathrm{D}} \equiv \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}} \cdot \boldsymbol{x}_j \,, \tag{53.9}$$

The constraints are

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \,, \tag{53.10}$$

$$\alpha_i \geq 0, \quad i = 1,\ 2, \ldots,\ n \,. \tag{53.11}$$

Note that in the dual only the input data appears in (53.9) in the form of dot products of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, $i, j = 1, 2, \ldots, n$.

The solution yields the Lagrange multipliers α_i^*, $i = 1, 2, \ldots, n$. The optimal weight vector and the bias term are then given by

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i \tag{53.12}$$

$$\text{and } \boldsymbol{w}^{*\mathrm{T}} \boldsymbol{x} + b \geq 1 \,. \tag{53.13}$$

The optimal decision hyperplane can now be obtained by substituting for $\boldsymbol{w}^*$ and b^* in (53.5) as

$$D(x) : \boldsymbol{w}^{*\mathrm{T}} \cdot \boldsymbol{x} + b^* = 0 \,. \tag{53.14}$$

To classify a new data point, only its $\boldsymbol{x}$ will be known, and its y is determined based on (53.12) and (53.14) as

$$\text{class of new data} = \operatorname{sgn}\left[\sum_{i=1}^{n} \alpha_i^* y_i \, (\boldsymbol{x}_i \cdot \boldsymbol{x}) + b^*\right] . \tag{53.15}$$

Note from (53.15) that only nonzero α_i^* participate in determining the class. The indices of these determine the data points that determine the class. These points are support vectors.

53.4.2 Relationship to the SRM Principle

To see how minimization in (53.7), or equivalently maximizing the margin, is related to implementing the SRM principle, suppose that the following bound holds,

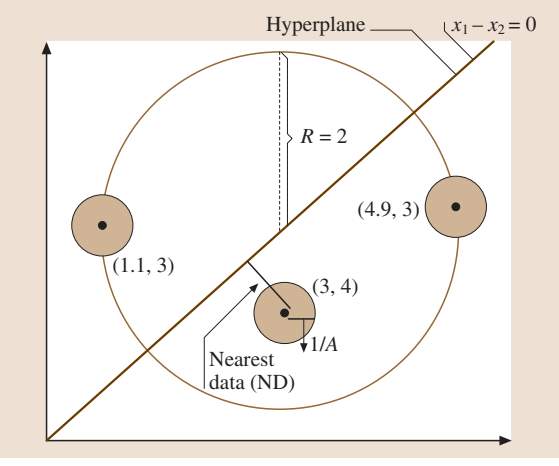


Fig. 53.3 Relationship between maximal margin and VC dimension

$||\boldsymbol{w}|| < A$, which means that the distance $d(\boldsymbol{w}, b; \boldsymbol{x})$ of a point from the hyperplane $(\boldsymbol{w}, b)$ is greater than or equal to $1/A$. The VC dimension, h, of the set of canonical hyper planes in n-dimensional space is bounded by

$$h \leq \min\left[R^2 A^2, n\right] + 1 \,,$$

where R is the radius of a hypersphere enclosing all the data points. Minimizing the above is equivalent to minimizing an upper bound on the VC dimension. Let us illustrate this by a simple numerical example.

Consider three ($n = 3$) data: (1.1, 3), (3, 4), and (4.9, 3), as shown in Fig. 53.3. The radius of a hypersphere enclosing all the data points, R, is 2. The distance between the nearest point (3, 4) and hyperplane $(x_1 - x_2) = 0$ is

$$\begin{aligned} d(\boldsymbol{w}, b; \boldsymbol{x}) &= \frac{|\boldsymbol{w} x_p \pm b|}{\|\boldsymbol{w}\|} \\ &= \frac{|w_1 x_{1p} + w_2 x_{2p} + \hbar\, w_n x_{np} \pm b|}{\sqrt{w_1^2 + w_2^2 + \hbar + w_n^2}} \\ &= \frac{|(1)\,(3) + (-1)\,(4)|}{\sqrt{1^2 + (-1)^2}} = 0.7071 \,. \end{aligned}$$

Therefore, $1/A \leq 0.7071 \approx 1.4142 \leq A$. In this example, we define $A = 1.5$ and $\min[R^2 A^2, n] + 1 = \min\left[\left(2^2\right)\left(1.5^2\right), 2\right] + 1 = \min[9, 2] + 1 = 3$. Therefore, the VC dimension, $h \leq 3$. Also, we can show that the norm of the weight vector should be equal to the inverse of the distance of the nearest point in the data set to the hyperplane. Here, the inverse of the distance between the nearest point (3, 4) and the hyperplane is $1/0.7071 = 1.4142$, which is equal to the norm of the weight vector $[1 - 1]$, $\sqrt{1^2 + (-1)^2} = 2 = 1.4142$.

53.4.3 Illustrative Example

We take a simple data set to illustrate in detail the computational steps involved in deriving the optimal separating hyperplane. Consider five normalized input data points in a two-dimensional plane along with their class labels, as shown in Table 53.1. Recall that our goal is to solve (53.9) subject to the constraints (53.10) and (53.11). We first compute the dot-product kernel of the input points and then compute $H(i, j) = y_i \cdot y_j \cdot \boldsymbol{x}_i^{\mathrm{T}} \cdot \boldsymbol{x}_j$

as shown below.

$$
\begin{aligned}
x_i^{\mathrm{T}} \cdot x_j &= \begin{pmatrix} x_1^{\mathrm{T}}x_1 & x_1^{\mathrm{T}}x_2 & x_1^{\mathrm{T}}x_3 & x_1^{\mathrm{T}}x_4 & x_1^{\mathrm{T}}x_5 \\ \hbar & & \hbar & & \hbar \\ x_5^{\mathrm{T}}x_1 & x_5^{\mathrm{T}}x_2 & x_5^{\mathrm{T}}x_3 & x_5^{\mathrm{T}}x_4 & x_5^{\mathrm{T}}x_5 \end{pmatrix} \\
&= \begin{pmatrix} \left(0\ 0\ \begin{pmatrix}0\\0\end{pmatrix}\right) & \left(0\ 0\ \begin{pmatrix}0\\1\end{pmatrix}\right) & \left(0\ 0\ \begin{pmatrix}0.3333\\0.75\end{pmatrix}\right) & \left(0\ 0\ \begin{pmatrix}0.6667\\0.75\end{pmatrix}\right) & \left(0\ 0\ \begin{pmatrix}1\\1\end{pmatrix}\right) \\ \hbar & & \hbar & & \hbar \\ \left(1\ 1\ \begin{pmatrix}0\\0\end{pmatrix}\right) & \left(1\ 1\ \begin{pmatrix}0\\1\end{pmatrix}\right) & \left(1\ 1\ \begin{pmatrix}0.3333\\0.75\end{pmatrix}\right) & \left(1\ 1\ \begin{pmatrix}0.6667\\0.75\end{pmatrix}\right) & \left(1\ 1\ \begin{pmatrix}1\\1\end{pmatrix}\right) \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ \hbar & & \hbar & & \hbar \\ 0 & 1.0000 & 1.0833 & 1.4167 & 2.0000 \end{pmatrix}, \\
H &= \begin{pmatrix} y_1 & y_1 & \hbar & y_1 \\ \hbar & \hbar & \hbar & \\ y_5 & y_5 & \hbar & y_5 \end{pmatrix} \cdot^* \begin{pmatrix} y_1 & y_2 & & y_5 \\ \hbar & & \hbar & \hbar \\ y_1 & y_2 & & y_5 \end{pmatrix} \cdot^* \boldsymbol{x}_i^{\mathrm{T}} \cdot \boldsymbol{x}_j \\
&= \begin{pmatrix} -1 & -1 & \hbar & -1 \\ \hbar & \hbar & \hbar & \\ -1 & -1 & \hbar & -1 \end{pmatrix} \cdot^* \begin{pmatrix} -1 & +1 & \hbar & -1 \\ \hbar & & \hbar & \hbar \\ -1 & +1 & \hbar & -1 \end{pmatrix} \cdot^* \begin{pmatrix} 0 & 0 & \hbar & 0 \\ \hbar & & \hbar & \hbar \\ 0 & 1.0000 & \hbar & 2.0000 \end{pmatrix}, \\
\text{or } H &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0000 & 0.7500 & -0.7500 & -1.0000 \\ 0 & 0.7500 & 0.6736 & -0.7847 & -1.0833 \\ 0 & -0.7500 & -0.7847 & 1.0069 & 1.4167 \\ 0 & -1.0000 & -1.0833 & 1.4167 & 2.0000 \end{pmatrix}.
\end{aligned}
$$

With the expression H as above, we now use the quadratic programming (QP) program, MATLAB Support Vector Machine Toolbox [53.5] to solve the dual problem given in (53.9), subject to the constraints in (53.10) and (53.11). The solution yields the optimal Lagrange multipliers given as

$$\left(\alpha^*\right)^{\mathrm{T}} = (7.11,\ 0.00,\ 32.22,\ 25.11,\ 0.00)\,.$$

Next, the optimal weight vector is obtained from (53.12) and its norm $||\boldsymbol{w}^*|| = \sqrt{64.44}$. Thus the maximal margin separating the hyperplanes is $2/||\boldsymbol{w}^*|| = 0.25$. Finally, the optimal bias from (53.13) is obtained as $b^* = -1.0$.

As mentioned before, only the nonzero values of the Lagrange multipliers participate in the solution, and they are $\alpha_1^* = 7.11$, $\alpha_3^* = 32.22$, $\alpha_4^* = 25.11$.

The input data points (1, 3, and 4) corresponding to the indices of these become the support vectors; the other two data points (2 and 5) can be ignored for classification decision. Thus the decision function of (53.15) can be written as

$$D(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{5} y_i \alpha_i^* \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x} + b^*\right). \qquad (53.16)$$

Next, we illustrate how a new data point for which only the $\boldsymbol{x}$ values are known is labeled by the above classifier. We first normalize the new data point according to the same scheme as used for the training data. Let the new point be (0.67, 1.00). The class computations proceed as follows. First, compute the dot product and H for the

Table 53.1 Data points for the illustrative example

X		Y
0.00	1.00	−1
0.00	1.00	+1
0.33	0.75	+1
0.67	0.75	−1
1.00	1.00	−1

new point and then substitute in (53.16), which yields the class as +1.

A graphical representation of the five data points used to develop the classifier, the optimal separating hyperplane, and the decision boundaries for classes −1 and +1 are shown in Fig. 53.4 in the normalized input space. We note that data points 1 and 4 are on the boundary for class −1, and data 3 is on the boundary of class +1. These are the three support vectors that participate in classification decisions. The other two points, 2 and 5, play no role in classification decisions.

53.5 Linear Classifier for Nonseparable Classes

In real-world applications, it is not realistic to construct a linear decision function without encountering errors. If the data are noisy, in general there will be no linear separation in the input space. Two situations may arise. In the first, data points fall in the region of separation, but on the right side so that the classification is correct. In the second case, data points fall on the wrong side and misclassification of points occurs. To accommodate such situations, the problem of the separable case is modified as follows. First, the classification constraints for the separable case are revised by adding slack variables (ξ_i). Next, the cost of constraint violation is set to C. With these modifications the function to be minimized becomes

$$L(\boldsymbol{w}, \xi_i) = \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i \,, \tag{53.17}$$

$$\text{subject to } y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geq 1 - \xi_I \tag{53.18}$$

$$\text{and } \xi_i \geq 0, i = 1, 2, \ldots, n \,. \tag{53.19}$$

Minimizing $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$ is related to minimizing the Vapnik–Chervonenkis (VC) dimension of the SVM, and the second term in (53.17) is an upper bound on the number of test errors. Thus, C controls the tradeoff between maximum margin and classification error, and ξ_i measures the deviation of a data point from the ideal condition of separability. If $0 \leq \xi_i \leq 1$, the data point is inside the decision region, but on the right side. If $\xi_i > 1$, the data point is on the wrong side. The data points that satisfy the above constraints are the support vectors. Proceeding as for the separable case with Lagrange multipliers α_i, the new dual problem is to

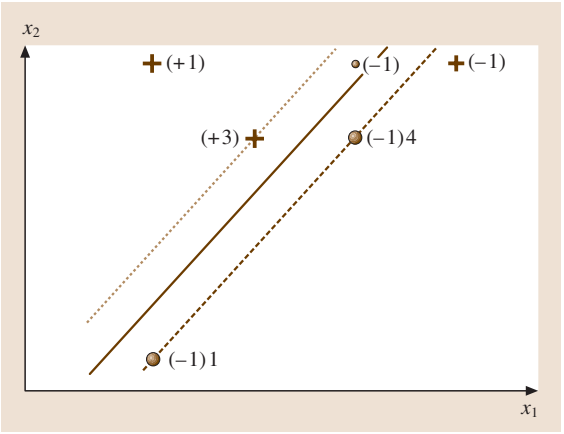


Fig. 53.4 Graphical representation of illustrative example

maximize

$$L(\alpha) = \sum \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j \tag{53.20}$$

$$\text{subject to } \sum_{i=1}^{n}\alpha_i y_i = 0 \tag{53.21}$$

$$\text{and } 0 \leq \alpha_i \leq C, i = 1, 2, \ldots, n \,. \tag{53.22}$$

Then the optimal weights are

$$\boldsymbol{w}^* = \sum_{i=1}^{n}\alpha_i^* y_i \boldsymbol{x}_i \,. \tag{53.23}$$

Again, only those points with nonzero α_i^* contribute to $\boldsymbol{w}^*$, i. e., only those points which are the support vectors.

53.6 Nonlinear Classifiers

This situation arises when the data are linearly nonseparable in the input space and the separation lines are nonlinear hypersurfaces. This is a common case in practical applications. Here, we seek nonlinear decision boundaries in the input space. For this, the approach described above is extended to derive nonlinear decision

boundaries. This is achieved by first mapping the input data into a higher-dimensional feature space using an inner-product kernel $K(\boldsymbol{x}, \boldsymbol{x}_i)$. Then, an optimum separating hyperplane is constructed in the feature space. This hyperplane is defined as a linear combination of the feature space vectors and solves a linear classification problem in the feature space. Together, these two steps produce the solution for the case of nonlinear classifiers.

53.6.1 Optimal Hyperplane

We provide below a brief justification for the above two-step procedure. According to Cover's theorem [53.3, 6], a nonseparable space may be nonlinearly transformed into a new feature space where the patterns are very likely separable. Three inner-product kernels employed for SVMs are listed in Table 53.2. Among these, the Gaussian kernel is most commonly used in practical applications.

Finally, the dual of the constrained optimization problem for this case can be obtained as

$$Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right), \tag{53.24}$$

where $Q(\alpha)$ has to be maximized with respect to the α_i subject to

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{53.25}$$

$$\text{and } 0 \le \alpha_i \le C, i = 1, 2, \ldots, n . \tag{53.26}$$

Here the parameter C is to be specified by the user. In the above, $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the ij-th element of the symmetric $n \times n$ matrix $\mathbf{K}$. A solution of the above problem yields the optimum Lagrange multipliers α_i^*, which yield the optimal weights as

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{\phi}(\boldsymbol{x}_i) , \tag{53.27}$$

where $\boldsymbol{\phi}(\boldsymbol{x}_i)$ represents the mapping of the input vector $\boldsymbol{x}_i$ into the feature space.

53.6.2 Illustrative Example

We illustrate the development of nonlinear classifiers using a Gaussian kernel for a small data set. The step-by-step solution procedure can be stated as follows:

- Preprocess input data
- Specify C, the kernel and its parameter
- Compute the inner-product matrix H
- Perform optimization using quadratic programming and compute the optimum α^*
- Compute the optimum weight vector $\boldsymbol{w}^*$
- Obtain support vectors and the decision boundary

Consider a data set of five points that consists of a 5×2 input matrix and a 5×1 vector y. Here, $\boldsymbol{x}_1$ is the normalized fan-in, $\boldsymbol{x}_2$ is the normalized module size in lines of code, and y is the module class.

$$\begin{array}{cc} X & y \\ \left(\begin{array}{cc:c} 0.29 & 0.00 & +1 \\ 1.00 & 0.02 & -1 \\ 0.00 & 0.19 & -1 \\ 0.06 & 1.00 & +1 \\ 0.02 & 0.17 & -1 \end{array}\right) & \end{array}$$

We use the radial basis function (rbf) kernel with $\sigma = 1.3$, and C is taken to be 100. Note that selection of C and σ is an important problem, as will be discussed later. These values are selected for illustrative purpose and are based on some preliminary analysis.

Next, the matrix H is obtained as

$$H(i, j) = \sum_{i=1}^{n} \sum_{j=1}^{n} y_i * y_j * \text{kernel}\left(rbf, \boldsymbol{x}_i, \boldsymbol{x}_j\right)$$
$$\equiv (5 \times 5)^* . (5 \times 5)^* . (5 \times 5) .$$

Table 53.2 Three common inner-product kernels

Type	$K(\boldsymbol{x}, \boldsymbol{x}_i), i = 1, 2, \ldots, n$	Comments
Linear	$\boldsymbol{x}^{\mathrm{T}} \boldsymbol{x}_i$	
Polynomial	$(\boldsymbol{x}^{\mathrm{T}} \boldsymbol{x}_i + 1)^b$	b is user-specified
Gaussian	$\exp\left[-(1/2\sigma^2)(\|\|\boldsymbol{x} - \boldsymbol{x}_i\|\|^2)\right]$	σ^2 is user-specified

By substituting the appropriate values, the computations for H proceed as shown below. Here the symbol $\hbar$ represents entries not shown but obtained similar to the shown entries.

$$
\begin{aligned}
H = &\begin{pmatrix} y_1 & \hbar & y_1 \\ y_2 & \hbar & y_2 \\ y_3 & \hbar & y_3 \\ y_4 & \hbar & y_4 \\ y_5 & \hbar & y_5 \end{pmatrix} .* \begin{pmatrix} y_1 & y_2 & y_3 & y_4 & y_5 \\ \hbar & \hbar & \hbar & \hbar & \hbar \\ y_1 & y_2 & y_3 & y_4 & y_5 \end{pmatrix} \\
&.* \begin{pmatrix} rbf(\sigma), x_1, x_1 & rbf(\sigma), x_1, x_2 & rbf(\sigma), x_1, x_3 & rbf(\sigma), x_1, x_4 & rbf(\sigma), x_1, x_5 \\ rbf(\sigma), x_2, x_1 & rbf(\sigma), x_1, x_2 & rbf(\sigma), x_1, x_3 & rbf(\sigma), x_1, x_4 & rbf(\sigma), x_1, x_5 \\ rbf(\sigma), x_3, x_1 & rbf(\sigma), x_1, x_2 & rbf(\sigma), x_1, x_3 & rbf(\sigma), x_1, x_4 & rbf(\sigma), x_1, x_5 \\ rbf(\sigma), x_4, x_1 & rbf(\sigma), x_1, x_2 & rbf(\sigma), x_1, x_3 & rbf(\sigma), x_1, x_4 & rbf(\sigma), x_1, x_5 \\ rbf(\sigma), x_4, x_1 & rbf(\sigma), x_1, x_2 & rbf(\sigma), x_1, x_3 & rbf(\sigma), x_1, x_4 & rbf(\sigma), x_1, x_5 \end{pmatrix} .* \\
= &\begin{pmatrix} 1 & \hbar & 1 \\ -1 & \hbar & -1 \\ -1 & \hbar & -1 \\ 1 & \hbar & 1 \\ -1 & \hbar & -1 \end{pmatrix} .* \begin{pmatrix} 1 & -1 & -1 & 1 & -1 \\ \hbar & \hbar & \hbar & \hbar & \hbar \\ 1 & -1 & -1 & 1 & -1 \end{pmatrix} \\
&.* \begin{pmatrix} \exp\left(-\frac{\left\|\begin{pmatrix}0.29\\0.00\end{pmatrix}-\begin{pmatrix}0.29\\0.00\end{pmatrix}\right\|^2}{2(1.3)^2}\right) & \hbar & \exp\left(-\frac{\left\|\begin{pmatrix}0.29\\0.00\end{pmatrix}-\begin{pmatrix}0.02\\0.17\end{pmatrix}\right\|^2}{2(1.3)^2}\right) \\ \hbar & \hbar & \hbar \\ \exp\left(-\frac{\left\|\begin{pmatrix}0.02\\0.17\end{pmatrix}-\begin{pmatrix}0.29\\0.00\end{pmatrix}\right\|^2}{2(1.3)^2}\right) & \hbar & \exp\left(-\frac{\left\|\begin{pmatrix}0.02\\0.17\end{pmatrix}-\begin{pmatrix}0.02\\0.17\end{pmatrix}\right\|^2}{2(1.3)^2}\right) \end{pmatrix} \\
= &\begin{pmatrix} 1 & -0.86 & -0.97 & 0.73 & -0.97 \\ -0.86 & 1 & 0.74 & -0.58 & 0.75 \\ -0.97 & 0.74 & 1 & -0.82 & 1.00 \\ 0.73 & -0.58 & -0.82 & 1 & -0.82 \\ -0.97 & 0.75 & 1.00 & -0.82 & 1 \end{pmatrix} .
\end{aligned}
$$

After performing optimization by quadratic programming, the optimal α values are obtained. Then the optimal vector $\boldsymbol{w}$ is computed from α^* and H. Its squared length is computed from these as below, where $\alpha^{*\prime}$ is the transpose of α^*

$$
\begin{aligned}
w^2 &= \alpha^{*\prime} * H * \alpha^* \\
&= \begin{pmatrix} 100 & 29.40 & 0 & 20.60 & 92.90 \end{pmatrix} \\
&\quad * \begin{pmatrix} 1 & -0.86 & -0.97 & 0.73 & -0.97 \\ -0.86 & 1 & 0.74 & -0.58 & 0.75 \\ -0.97 & 0.74 & 1 & -0.82 & 1.00 \\ 0.73 & -0.58 & -0.82 & 1 & -0.82 \\ -0.97 & 0.75 & 1.00 & -0.82 & 1 \end{pmatrix} * \begin{pmatrix} 100 \\ 29.40 \\ 0 \\ 20.60 \\ 92.90 \end{pmatrix} \\
&= 102.
\end{aligned}
$$

The indices of the support vectors are those α^* that satisfy $0 < \alpha^* \leq C$. Here, $\alpha_3^* = 0$ so that points 1, 2, 4, and 5 become the support vectors for this classification problem; point 3 plays no role and can be ignored.

A graphical illustration of the decision boundaries in the input space is shown in Fig. 53.5. Also, note that the decision boundaries are nonlinear, while the decision hyperplane in the feature space computed from the feature vector is expected to be linear. Finally, for this problem, data point 1 is misclassified as being in class -1 rather than in class $+1$, i. e., the classification error of the model derived here is 20%. To classify a new point, suppose its normalized values using the same normalization as for the training data are

(0.03, –0.03). The classification computations proceed as follows:

$$
\begin{aligned}
H(i,j) &= \sum_{i=1}^{m}\sum_{j=1}^{n} y_j * \text{kernel}\left(rbf, xt_i, x_j\right) \\
&= \begin{pmatrix} y_1 & y_2 & y_3 & y_4 & y_5 \end{pmatrix} \\
&\quad * \begin{pmatrix} (rbf(\sigma), xt_1, x_1) \\ (rbf(\sigma), xt_1, x_2) \\ (rbf(\sigma), xt_1, x_3) \\ (rbf(\sigma), xt_1, x_4) \\ (rbf(\sigma), xt_1, x_5) \end{pmatrix}^{\top} \\
&= \begin{pmatrix} 1 & -1 & -1 & 1 & -1 \end{pmatrix} \\
&\quad .* \begin{pmatrix} \exp\left(-\dfrac{\left\|\begin{pmatrix}0.03\\-0.03\end{pmatrix}-\begin{pmatrix}0.29\\0.00\end{pmatrix}\right\|^2}{2.(1.3)^2}\right) \\ \hbar \\ \exp\left(-\dfrac{\left\|\begin{pmatrix}0.03\\-0.03\end{pmatrix}-\begin{pmatrix}0.02\\0.17\end{pmatrix}\right\|^2}{2.(1.3)^2}\right) \end{pmatrix}^{\top} \\
&= \begin{pmatrix} 0.98 & -0.76 & -0.98 & 0.73 & -0.99 \end{pmatrix} ; \\
y &= \text{sgn}\left(H * \alpha + \text{bias}\right)
\end{aligned}
$$

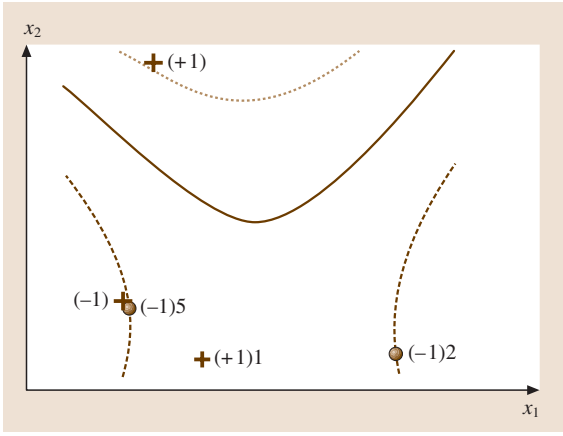


Fig. 53.5 Graphical illustration of nonlinear decision boundaries

$$
\begin{aligned}
&= \text{sgn}\left(\begin{pmatrix} 0.98 \\ -0.76 \\ -0.98 \\ 0.73 \\ -0.99 \end{pmatrix}^{\top} * \begin{pmatrix} 100.00 \\ 29.40 \\ 0.00 \\ 20.60 \\ 92.90 \end{pmatrix} + 0\right) \\
&= \text{sgn}\left(-0.92\right) = -1 .
\end{aligned}
$$

The new module belongs to class –1.

53.7 SVM Nonlinear Regression

As mentioned earlier, the support vector technique was initially developed for classification problems. This approach has been extended to nonlinear regression where the output y are real-valued. A general nonlinear regression model for y based on $\boldsymbol{x}$ can be written as

$$y = f(\boldsymbol{x}, \boldsymbol{w}) + \delta \,, \tag{53.28}$$

where f represents a function, $\boldsymbol{w}$ is a set of parameters, and δ represents noise. In terms of some nonlinear basis functions, as discussed earlier, we can write $\hat{y}$, an estimate of y, as

$$\hat{y} = \boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{x}) \,. \tag{53.29}$$

Next, we employ the commonly used Vapnik's ε-loss function and estimate $\hat{y}$ via support vector regression as summarized below. The ε-loss is defined as

$$\text{Loss}_{\varepsilon}\left(y, \hat{y}\right) = \begin{cases} |y - \hat{y}| - \varepsilon, & \text{if } |y - \hat{y}| \geq \varepsilon \,. \\ 0 \quad\quad\quad , & \text{otherwise} \end{cases}$$

The dual problem for regression can be formulated using an approach similar to that for classification. The optimization problem now is to maximize Q:

$$
\begin{aligned}
Q\left(\alpha_i, \alpha_i'\right) &= \sum_{i=1}^{n} y_i \left(\alpha_i - \alpha_i'\right) - \varepsilon \sum_{i=1}^{n} \left(\alpha_i + \alpha_i'\right) \\
&\quad - \frac{1}{2} \sum_{i=1}^{n}\sum_{j=1}^{n} \left(\alpha_i - \alpha_i'\right) \\
&\quad \times \left(\alpha_j - \alpha_j'\right) K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)
\end{aligned} \tag{53.30}
$$

$$\text{subject to} \quad \sum_{i=1}^{n} \left(\alpha_i - \alpha_i'\right) = 0 \,, \tag{53.31}$$

$$0 \le \alpha_i, \alpha_i' \le C, i = 1, 2, \ldots, n \,. \tag{53.32}$$

In (53.30–53.32), ε and C are user-specified values. The optimal values of α_i and α_i' are used to find the optimal value of the weight vector. The estimated $\hat{y}$ is given by

$$\hat{y}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = \sum_{i=1}^{n} \left(\alpha_i - \alpha_i'\right) K(\boldsymbol{x}, \boldsymbol{x}_i) \,.$$

53.8 SVM Hyperparameters

The classification and regression modeling problems using SVM are formulated as quadratic programming (QP) optimization problems. Many algorithms are available for solving QP problems and are commonly used for support vector modeling applications. However, there are some parameters that are to be specified by the user. These are called *the SVM hyperparameters* and are briefly described below.

For the linearly nonseparable case, we need to specify the penalty parameter C. It controls the tradeoff between the small function norm and empirical risk minimization. Further, for nonlinear classifiers, we also need to specify the kernel and its parameters. For example, for the radial basis function kernel, its width, σ, needs to be specified by the user. In practical applications, there are no easy answers for choosing C and σ. In general, to find the best values, different combinations are tried and their performances are compared, usually via an independent data set, known as the validation set. However, some empirical rules for their determination have been proposed in the literature [53.7, 8].

For nonlinear regression, a loss function is specified. In support vector machine applications, a commonly used loss function is the so-called *ε-loss function* as indicated above. Thus, for regression problems this additional hyperparameter is to be specified by the user. Generally, a trial-and-error approach is used to evaluate the performance of different hyperparameter combinations on some validation data set.

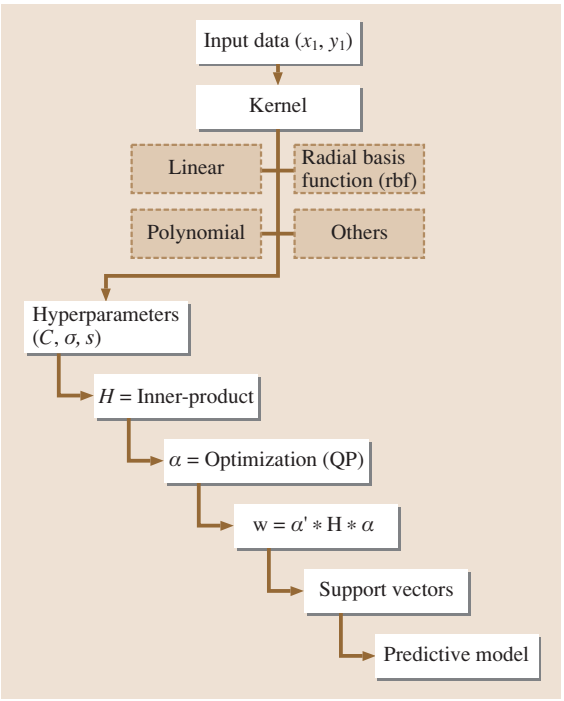


Fig. 53.6 Support vector modeling flow chart

53.9 SVM Flow Chart

A generic flow chart depicting the development of support vector classification and regression models is shown in Fig. 53.6. For given data $(\boldsymbol{x}, y)$, the kernel is selected by the user, followed by the appropriate hyperparameters. Computation of several intermediate quantities and optimization by quadratic programming yields the weights and support vectors. Finally, these values are used to define the classification or regression model.

53.10 Module Classification

There are several reasons for developing software classification models. One is the limited availability of resources. Not all modules can be treated in the same way. The potentially critical modules require a more time-consuming and costly development process that involves activities such as more rigorous design and code reviews, automated test-case generation and unit testing, etc. Another reason is that faults found later in the development life-cycle are more expensive to correct than those found earlier.

The problem of software module classification has been addressed in the software engineering literature for more than thirty years. Different techniques have been applied by many authors with varying degrees of predictive accuracy [53.9, 10]. Most of the early work on this topic used statistical techniques such as discriminant analysis, principle-component analysis, and factor analysis, as well as decision or classification trees [53.11]. In recent years, machine learning techniques and fuzzy logic have also been used for software module classification. Typical of these are classification and regression trees (CART), case-based reasoning (CBR), expert judgment, and neural networks [53.10]. The main problem with most of the current models is their low predictive accuracy. Since the published results vary over a wide range, it is not easy to give a specific average accuracy value achieved by current models.

In this section, we develop support vector classification models for software data obtained from the public-domain National Aeronautics and Space Administration (NASA) software metrics database [53.12]. It contains several product and process metrics for many software systems and their subsystems. The fifteen module-level metrics used here and a brief description of each are given in Table 53.3. The metric x_7 is the number of faults, while x_9 to x_{22} are module-level product metrics which include the design-based metrics (x_9, x_{10}, x_{18}) and primarily coding metrics (x_{13}, x_{14}, x_{15}). Metrics x_{19} to x_{22} are Halstead's software science measures; x_{19} and x_{20} are the vocabulary, while x_{21} and x_{22} are measures of program size. Other metrics are self-explanatory. These represent typical metrics used in module classification studies. This system consists of 67 modules with a size of about 40k lines of code.

Table 53.3 List of metrics from NASA database

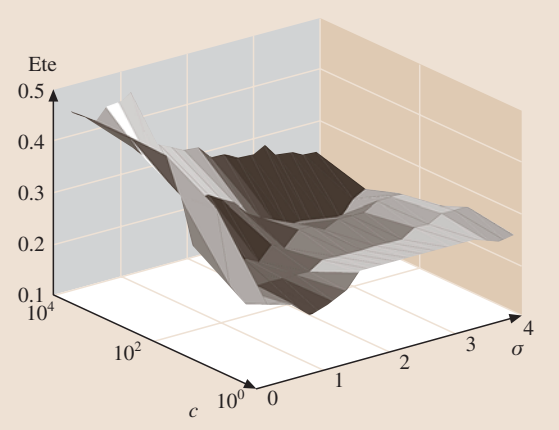
X_7	Faults
X_9	Fan out
X_{10}	Fan in
X_{11}	Input–output statements
X_{12}	Total statements
X_{13}	Size of component in program lines
X_{14}	Number of comment lines
X_{15}	Number of decisions
X_{16}	Number of assignment statements
X_{17}	Number of formal statements
X_{18}	Number of input–output parameters
X_{19}	Number of unique operators
X_{20}	Number of unique operands
X_{21}	Total number of operators
X_{22}	Total number of operands

Here, faults are the outputs and the others are the inputs. Referring to (53.1), we have $n = 67$ and $d = 14$. We first preprocess the input data. After transformation, this data set resides in a fourteen-dimensional unit cube. To determine module class, we use a threshold value of 5 so that, if $x_7 \leq 5$, the class is -1, and $+1$ otherwise.

We now develop nonlinear classifiers for this data set using the SVM algorithm of Sect. 53.6 [53.13]. The

Table 53.4 Classification results

Set	σ	C	SV	Classification error (average)	
				Training	5CV
I	0.8	1	30.6	0.18	0.21
II	1.2	100	26.8	0.10	0.20
III	1.6	1000	27.2	0.12	0.20
IV	2.0	100	26.6	0.13	0.21
V	2.4	1000	25.8	0.09	0.21
VI	2.8	1000	26.4	0.11	0.19
VII	3.2	1000	25.8	0.12	0.17
VIII	3.6	1000	26.2	0.12	0.17
IX	4.0	1000	25.4	0.12	0.21



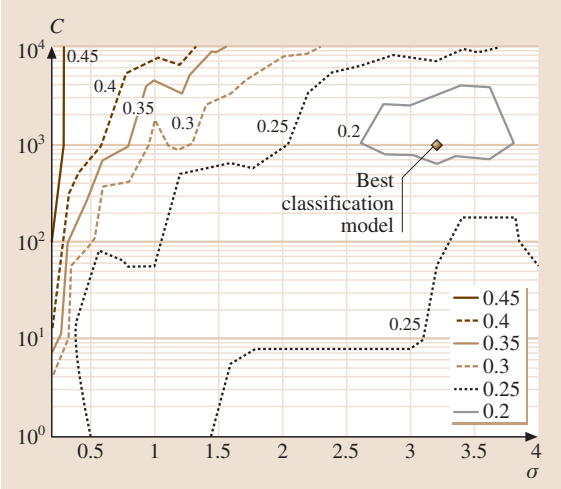


Fig. 53.6 Test error; *left panel*: surface; *right panel*: contours

optimization problem to be solved is given by (53.24–53.26). First, we choose a kernel. Since Gaussian is a popular choice, we also choose it. Next we need to specify the hyperparameters, that is, the width of the Gaussian and the penalty parameter C. To determine the best combination of these two, we follow the common practice of performing a grid search. In this case the grid search is on the two parameters C and σ. We took $C = 10^{-2}(10)10^4$ and $\sigma = 0.8(0.4)4.0$ for a total of 56 grid points. For each combination, we use the SVM algorithm for nonlinear classifiers of Sect. 53.6. Further, we used five-fold cross validation (5CV) as a criterion for selecting the best hyperparameter combination. Thus, we are essentially doing a search for the best set of hyperparameters among the 56 potential candidates by constructing $56 \times 5 = 280$ classifiers, using the approach described in Sect. 53.6.

As an example, a list of nine sets and their corresponding classification errors is given in Table 53.4. The average number of input points that became support vectors for the given model is also included. Data such as that in Table 53.4 is used to select the best set.

From this we select set VII with $C = 1000$ and kernel width 3.2. To further study the behavior of the five-fold cross validation test error versus (C, σ), its surface and the contours are shown in Fig. 53.6. We note that the surface for this data is relatively flat for high (σ, C) values and sharp for low σ and high C. This behavior is quite typical for many applications. Also shown in the contour plot is the chosen set with $\sigma = 3.2$ and $C = 1000$. Finally, these values are used as the SVM hyperparameters to solve the optimization problem of (53.24–53.26) using quadratic programming. This gives the desired, possibly best, classification model for this data set. The developed model is likely to have an error of about 17% on future classification tasks.

53.11 Effort Prediction

Development of software effort-prediction models has been an active area of software engineering research for over 30 years. The basic premise underlying these models is that historical data about similar projects can be employed as a basis for predicting efforts for future projects. For both engineering and management reasons, an accurate prediction of effort is of significant importance in software engineering, and improving estimation accuracy is an important goal of most software development organizations. There is a continuing search for better models and tools to improve predictive performance.

The so-called general-purpose models are generally algorithmic models developed from a relatively large collection of projects that capture a functional relationship between effort and project characteristics [53.14, 15]. The statistical models are derived from historical data using statistical methods, mostly regression analysis. The statistical models were some of the earliest to be developed. Exam-

Table 53.5 Performance of effort prediction models

Data set	LooMMRE		LooPred(25)	
	expected training	expected test	average training	average test
D	0.54	0.52	0.33	0.46
$D-1$	0.28	0.35	0.72	0.54
$D-2$	0.24	0.24	0.72	0.64
$D-3$	0.12	0.25	0.44	0.80

ples of such models are the meta-model [53.16] and the MERMAID [53.17]. Recently, machine learning techniques have been used for software effort prediction modeling. These include neural networks [53.18], rule induction, and case-based reasoning.

The effort-modeling problem can be restated as follows from Sect. 53.2 above. We are given data about n projects $\{\boldsymbol{x}_i, y_i\} \in \mathbb{R}^n \times \mathbb{R}$, $i = 1, \ldots, n$, each consisting of d software features the y are the effort values. A general nonlinear regression model for y based on $\boldsymbol{x}$ was given in (53.28).

In this section we summarize the prediction model development by support vector nonlinear regression from [53.13]. The effort data and the project features are taken from [53.19]. The data were collected from a Canadian software house. It consists of 75 projects developed in three different environments. The data is grouped by each environment ($D-1$, $D-2$, $D-3$) and as combined projects (D). There were six features collected for each project. Thus, for data set D, $n = 75$ and $d = 6$ in (53.2).

The methodology for developing SVM nonlinear regression models is very similar to that used for module classification in Sect. 53.10. The optimization problem to be solved now is given in (53.30–53.32). Further, assuming a Gaussian kernel, three hyperparameters have to be specified. Therefore, a three-dimensional grid search has to be performed for selecting σ, C, and ε. The criterion for this selection can be MMRE or Pred(25). Note that we seek a low MMRE error and a high Pred(25) accuracy.

The final results of SVM modeling for the above data sets are summarized in Table 53.5 for both selection criteria. For projects in D, the best model obtained has test values of LooMMRE $= 0.52$ and LooPred(25) $= 0.46$. However, for $D-1$, $D-2$ and $D-3$, model performance is much better due to the fact that the projects in these data sets were developed in more homogeneous environments than those in D.

53.12 Concluding Remarks

We have presented an introduction to support vector machines, their conceptual underpinnings and the main computational techniques. We illustrated the algorithmic steps via examples and presented a generic SVM flowchart. Results from two software engineering case studies using SVM were summarized.

SVM is a very active area of research and applications. An impressive body of literature on this topic has evolved during the last decade. Many open problems, theoretical and applied, are currently being pursued. These include hyperparameter selection, Bayesian relevance vector machines, reduced SVM, multiclass SVM, etc. Applications include intrusion detection, data mining, text analysis, medical diagnosis and bioinformatics. Papers on these aspects regularly appear in the machine learning and related literature.

For further reading, chapters in *Kecman* [53.4], *Cherkassky* et al. [53.20] and *Haykin* [53.3] provide good insights. Books on SVM include *Cristianini* et al. [53.21], *Scholkopf* et al. [53.22], and *Vapnik* [53.1]. Tutorials, such as *Burges* [53.23], and other useful information is available at websites dealing with support vector machines and kernel machines. Software packages are also available from several websites.

References

53.1 V. N. Vapnik: *Statistical Learning Theory* (Wiley, New York 1998)

53.2 V. N. Vapnik: An overview of statistical learning theory, IEEE Trans. Neural Netw. **10(5)**, 988–1000 (1999)

53.3 S. Haykin: *Neural Networks – A Comprehensive Foundation*, 2nd edn. (Prentice Hall, Upper Saddle River, NJ 1999)

53.4 V. Kecman: *Learning and Soft Computing* (MIT Press, Cambridge, MA 2001)

53.5 S. R. Gunn: *MATLAB Support Vector Machine Toolbox* (1998), http://www.isis.ecs.soton.ac.uk/resources/svminfo/

53.6 T. Hastie, R. Tibshirani, J. H. Friedman: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer, Berlin Heidelberg New York 2001)

53.7 S. S. Keerthi, C-J. Lin: Asymptotic behaviors of support vector machines with Gaussian kernels, Neural Comput. **15**(7), 1667–1689 (2003)

53.8 O. Chapelle, V. Vapnik: *Model Selection for Support Vector Machines. Advances in Neural Information Processing Systems* (AT&T Labs-Research, Lyone 1999)

53.9 A. L. Goel: *Software Metrics Statistical Analysis Techniques and Final Technical Report* (U. S. Army, 1995)

53.10 T. M. Khoshgoftaar, N. Seliya: Comparative assessment of software quality classification techniques: An empirical case study, Empir. Softw. Eng. **9**, 229–257 (2004)

53.11 C. Ebert, T. Liedtke, E. Baisch: Improving Reliability of Large Software Systems. In: *Annals of Software Engineering*, Vol. 8, ed. by A. L. Goel (Baltzer Science, Red Bank, NJ 1999) pp. 3–51

53.12 NASA IV & V Metrics Data Program. http://mdp.ivv.nasa.gov/

53.13 H. Lim: Support Vector Parameter Selection Using Experimental Design Based Generating Set Search (SVEG) with Application to Predictive Software Data Modeling. Ph.D. Thesis (Syracuse Univ., New York 2004)

53.14 B. W. Boehm: *Software Engineering Economics* (Prentice Hall, New York 1981)

53.15 L. H. Putnam: A general empirical solution to the macro sizing and estimating problem, IEEE Trans. Softw. Eng. **4**, 345–361 (1978)

53.16 J. W. Bailey, V. R. Basili: A Meta-Model for Software Development Resource Expenditures, Proceedings of the 5th International Conference on Software Engineering, San Diego, CA (IEEE Press, Piscataway, NJ 1981) 107–116

53.17 P. Kok, B. A. Kitchenham, J. Kirakowski: The MERMAID Approach to Software Cost Estimation. In: *Proceedings ESPRIT Technical Week* (Kluwer Academic, Brussels 1990)

53.18 M. Shin, A. L. Goel: Empirical data modeling in software engineering using radial basis functions, IEEE Trans. Softw. Eng. **36**(5), 567–576 (2000)

53.19 J. M. Desharnais: *Analyse Statistique de la Productivitie des Projets Informatique a Partie de la Technique des Point des Fonction. MSc Thesis* (Univ. of Quebec, Montreal 1988)

53.20 V. Cherkassky, F. Mulier: *Learning from Data: Concepts, Theory, and Methods* (Wiley-Interscience, New York 1998)

53.21 N. Cristianini, J. Shawe-Taylor: *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods* (Cambridge Univ. Press, Cambridge 2000)

53.22 P. Scholkopf, A. Smola: *Learning with Kernels* (MIT Press, Cambridge, MA 2002)

53.23 C. J. C. Burges: A Tutorial on SVM for Pattern Recognition, Data Mining and Knowledge Discovery **2**, 167–212 (1998)