CSCI 4588/5588: Machine Learning II.

Chapter #3: Support Vector Machine

(Ref: Ch 4, 12 of [1], Ch 6, 7 of [2])

Objectives:

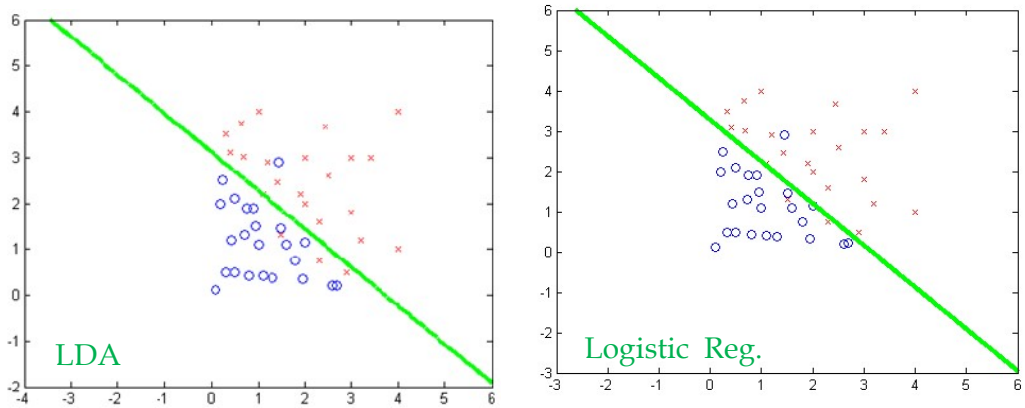
(1) Identify the best decision boundary (or class separating hyperplane), i.e. the decision boundary with maximum class margin.

(2) There may be no margin for overlapped class boundary; still we adjust the theme of decision boundary with maximum margin.

(3) For non-linearly separable classes, we would like to learn to transform the space into a linearly separable space.

Separating Hyperplanes (see 4.5 of [1])

We have seen that linear discriminant analysis (LDA) and logistic regression (LR) both estimate linear decision boundaries in similar but slightly different ways (Figure A). These procedures construct linear decision boundaries that explicitly try to separate the data into different classes as well as possible.



(a) (b)

Figure A: (a) LDA (b) LR based decision boundary for cancer {Benign, Malignant} toy problem.

Figure 4.14 shows 20 data points in two classes in $\Re^2$. These data can be separated by a linear boundary. Included in the figure (blue lines) are two of the infinitely many possible *separating hyperplanes*. The orange line is the

least-squares solution to the problem, obtained by regressing the −1/1 response Y on X (with intercept); the line is given by

$$\left\{x : \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0\right\} \tag{4.39}$$

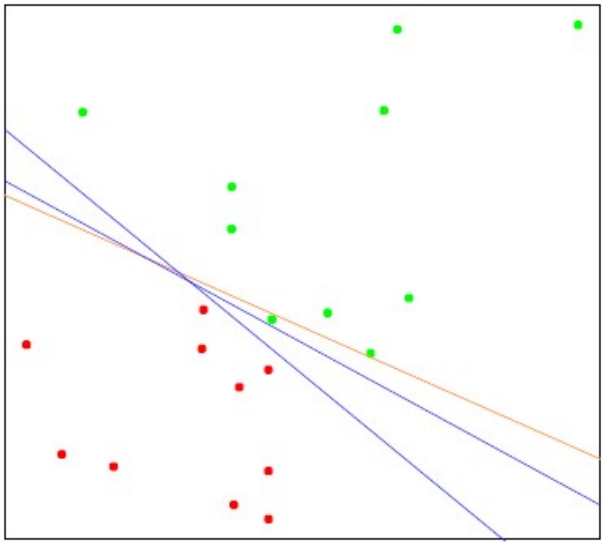


FIGURE 4.14. A toy example with two classes separable by a hyperplane. The orange line is the least-squares solution, which misclassifies one of the training points. Also shown are two blue separating hyperplanes found by the perceptron learning algorithm with different random starts.

This least-squares solution does not do a perfect job in separating the points (Figure 4.14) and makes one error. LDA would find the same boundary in light of its equivalence with linear regression in the two-class case. We also have other two decision boundaries in Figure 4.14. The questions may arise which one to choose and how to choose – we will be discussing more on this in the next section (i.e., section: Optimal Separating Hyperplanes).

Classifiers such as (4.39), that compute a linear combination of the input features and return the sign, were called perceptrons in the engineering literature in the late 1950s (Rosenblatt, 1958). Perceptrons set the foundations for the neural network models of the 1980s and 1990s.

Before we continue, let us digress slightly and review some ***vector algebra***. Figure 4.15 depicts a hyperplane or affine set L defined by the equation $f(x) => \beta_0 + \beta^T x = 0$; since we are in $\Re^2$ this is a line.

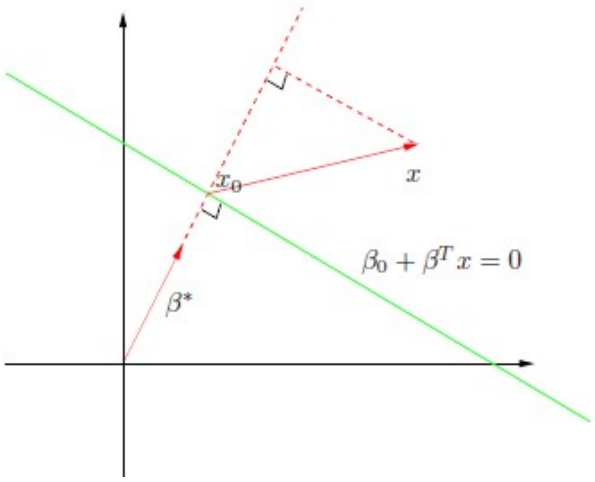


FIGURE 4.15. The linear algebra of a hyperplane (affine set).

Here we list some properties:

1. For any two points x_1 and x_2 lying in L, $\beta^T (x_1 - x_2) = 0$, and hence $\beta^* = \frac{\beta}{\|\beta\|}$ is the vector normal to the surface of L.
2. For any point x_0 in L, $\beta^T x_0 = -\beta_0$. (i)
3. The signed distance of any point x to L is given by

$$\begin{aligned}
\beta^{*T}(x - x_0) &= \frac{\beta^T}{\|\beta\|}(x - x_0) \\
&= \frac{1}{\|\beta\|}\beta^T(x - x_0) \\
&= \frac{1}{\|\beta\|}(\beta^T x - \beta^T x_0) \qquad [\because \text{using } (i),\ -\beta^T x_0 = \beta_0] \\
&= \frac{1}{\|\beta\|}(\beta^T x + \beta_0) \\
&= \frac{1}{\|f'(x)\|} f(x) \qquad (4.40)
\end{aligned}$$

Hence $f(x)$ is proportional to the signed distance from x to the hyperplane defined by $f(x) = 0$.

Optimal Separating Hyperplanes (4.5.2 of [1])

For the separating hyperplane, we have the equation (see Figure 4.15):

$$\beta^T x + \beta_0 = 0$$

Now, assume, $y_i \in \{-1,+1\}$ for $\forall_i$ we want to satisfy:

$$\beta^T x_i + \beta_0 > 0, \quad if\ y_i = 1$$
$$\beta^T x_i + \beta_0 < 0, \quad if\ y_i = -1$$

where the decision function is

$$D(x_i) = sign(\beta^T x_i + \beta_0)$$

We can have many possible sets of (β, β_0) to have the separating hyperplane. However, we are interested in the one having the maximum margin. We can define the margins as (Figure B):

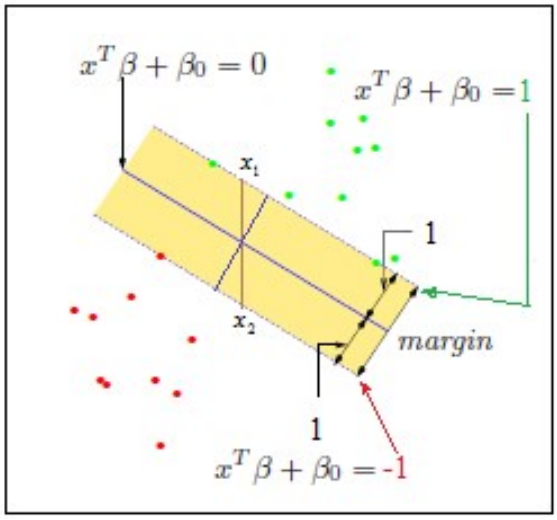


Figure B: Support vector classifier. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width, 1+1 = 2.

Margins are defined by:

$$\beta^T x + \beta_0 = 1$$
$$\beta^T x + \beta_0 = -1$$

Deducting the above two equations for points x_1 and x_2 in them respectively, we have

$$\beta^T (x_1 - x_2) = 2$$
$$=> \frac{\beta^T}{\|\beta\|}(x_1 - x_2) = \frac{2}{\|\beta\|} \qquad [\because divide\ both\ side\ by \|\beta\|]$$

$$\Rightarrow \beta^{*T}(x_1 - x_2) = \frac{2}{\|\beta\|} \qquad [\because \beta^* = \frac{\beta^T}{\|\beta\|}]$$

The LHS of the above equation (i.e., $\beta^{*T}(x_1 - x_2)$) is the projection of the line connecting x_1 and x_2, on the β. Here we want to maximize the projection on to the margins. Therefore we can have our goal as:

$$\max \frac{2}{\|\beta\|},$$

Or, we can write:

$$\max \frac{1}{\|\beta\|}, \quad \text{[Just one distance from hyperplane to support vector]}$$

Alternatively, we can also write the same goal as;

$$\min \|\beta\| \tag{ii}$$

and in general, we can define the margins as

$$(\beta^T x_i + \beta_0) \geq 1, \quad if\ y_i = 1$$
$$(\beta^T x_i + \beta_0) \leq -1, \quad if\ y_i = -1$$

Combining the above two equations, we can simply write:

$$y_i(\beta^T x_i + \beta_0) \geq 1 \tag{iii}$$

Thus, in general we write from (*ii*) and (*iii*) the following goal:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 \tag{4.48}$$
subject to $y_i(\beta^T x_i + \beta_0) \geq 1$, $i = 1, ..., N$.

In light of (4.40), the constraints define an empty slab or margin around the linear decision boundary of thickness $1/\|\beta\|$. Hence we choose β and β_0 to maximize its thickness. This is a convex optimization problem (quadratic criterion with linear inequality constraints). The ***Lagrange*** (primal, L_p) function (see Appendix, ~pages; 23-25), to be minimized w.r.t. β and β_0 is

$$L_p(\beta,\beta_0,\alpha_i)=\frac{1}{2}\|\beta\|^2-\sum_{i=1}^{N}\alpha_i\left[y_i(x_i{}^T\beta+\beta_0)-1\right] \tag{4.49}$$

Here α_i is the Lagrange multiplier and $\alpha_i \geq 0$. It is to be noted that there is a minus sign in front of the ***Lagrange multiplier*** term because we are minimizing with respect to β and β_0 but maximizing with respect to α_i.

Setting the derivatives to zero, we obtain:

$$\beta=\sum_{i=1}^{N}\alpha_i y_i x_i \tag{4.50}$$

$$0=\sum_{i=1}^{N}\alpha_i y_i \tag{4.51}$$

and substituting these in (4.49) we obtain the so-called Wolfe dual

$$\begin{aligned}
L_p(\beta,\beta_0,\alpha_i)&=\frac{1}{2}\|\beta\|^2-\sum_{i=1}^{N}\alpha_i\left[y_i(x_i{}^T\beta+\beta_0)-1\right]\\
&=\frac{1}{2}\|\beta\|^2-\sum_{i=1}^{N}\alpha_i y_i x_i{}^T\beta-\sum_{i=1}^{N}\alpha_i y_i\beta_0+\sum_{i=1}^{N}\alpha_i\\
&=\frac{1}{2}\|\beta\|^2-\sum_{i=1}^{N}\sum_{k=1}^{N}\alpha_i\alpha_k y_i y_k x_i{}^T x_k-\beta_0\sum_{i=1}^{N}\alpha_i y_i+\sum_{i=1}^{N}\alpha_i\\
&=\frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}\alpha_i\alpha_k y_i y_k x_i{}^T x_k-\sum_{i=1}^{N}\sum_{k=1}^{N}\alpha_i\alpha_k y_i y_k x_i{}^T x_k+\sum_{i=1}^{N}\alpha_i \quad \left[\because\sum_{i=1}^{N}\alpha_i y_i=0\right]
\end{aligned}$$

Clearly, we can now express the equation as a function of α_i,

$$L_D(\alpha_i)=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}\alpha_i\alpha_k y_i y_k x_i{}^T x_k$$

$$\text{subject to } \alpha_i \geq 0. \tag{4.52}$$

The dual expression ($L_D(\alpha_i)$) is now a maximization problem with respective to α instead of β. In fact, in the equation there exists no β. Any simple optimization solver software will do to solve the equation (4.52). Besides, the solution must satisfy the Karush–Kuhn–Tucker (KKT) conditions, which include (4.50), (4.51), (4.52) and

$$\alpha_i[y_i(x_i{}^T\beta+\beta_0)-1]=0 \quad \forall i \tag{4.53}$$

From these (4.52 and 4.53) we can see that

- if $\alpha_i > 0$, then $y_i(x_i^T\beta + \beta_0) = 1$, or in other words, x_i is on the boundary of the slab;
- if $y_i(x_i^T\beta + \beta_0) > 1$, x_i is not on the boundary of the slab, and $\alpha_i = 0$.

From (4.50) we see that the solution vector β is defined in terms of a linear combination of the support points x_i—those points defined to be on the boundary of the slab via $\alpha_i > 0$. Figure 4.16 shows the optimal separating hyperplane for our toy example; there are three support points. Likewise, β_0 is obtained by solving (4.53) for any of the support points.

The optimal separating hyperplane produces a function $\hat{f}(x) = x^T\hat{\beta} + \hat{\beta}_0$ for classifying new observations:

$$\hat{G}(X) = sign\,\hat{f}(x) \tag{4.54}$$

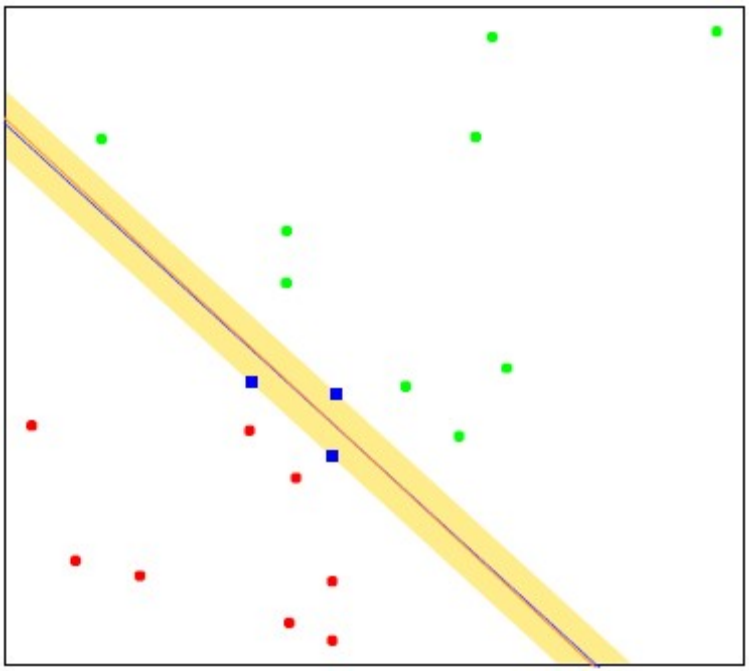


FIGURE 4.16. The same data as in Figure 4.14. The shaded region delineates the maximum margin separating the two classes. There are three support points indicated, which lie on the boundary of the margin, and the optimal separating hyperplane (blue line) bisects the slab. Included in the figure is the boundary found using logistic regression (red line), which is very close to the optimal separating hyperplane.

The intuition here is that a large margin on the training data will lead to good separation on the test data. The description of the solution in terms of support points seems to suggest that the optimal hyperplane focuses more on the points that count, and is more robust to model misspecification. The LDA solution, on the other hand, depends on all of the data, even points far away from the decision boundary. Note, however, that the identification of these support points required the use of all the data. Of course, if the classes are Gaussian, then LDA is optimal, and separating hyperplanes will pay the price for focusing on the (noisier) data at the boundaries of the classes.

Included in Figure 4.16 is the logistic regression solution to this problem, fit by maximum likelihood. Both solutions are similar in this case. When a separating hyperplane exists, logistic regression will always find it, since the log-likelihood can be driven to 0 in this case. The logistic regression solution shares some other qualitative features with the separating hyperplane solution. The coefficient vector is defined by a weighted least-squares fit of a zero-mean linearized response on the input features, and the weights are larger for points near the decision boundary than for those further away.

When the data are not separable, there will be no feasible solution to this problem, and an alternative formulation is needed. Again one can enlarge the space using basis transformations for example.

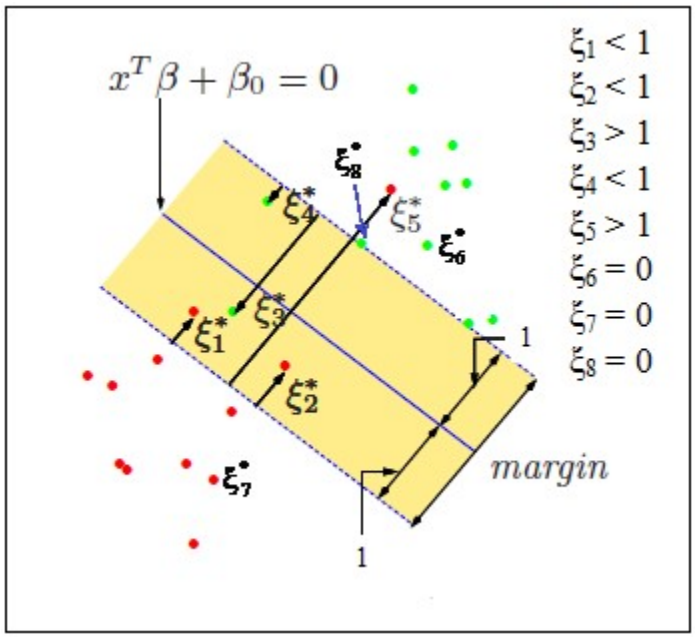


Figure 12.1: Support vector classifier shows the nonseparable (overlap) case. We have $\xi>0$ for the points, which are the wrong side of their margin and $\xi=0$ for points on the

correct side of their margin. The margin is maximized subject to a total budget $\Sigma \xi i \leq$ constant. Hence $\sum \xi_j^*$ is the total distance of points on the wrong side of their margin.

Overlapped Classes

Suppose now that the classes overlap in feature space. One way to deal with the overlap is to still maximize the margin but allow for some points to be on the wrong side of the margin. We define the slack variables $\xi = (\xi_1, \xi_2, \ldots, \xi_N)$. We modify the constraint in (*iii*):

$$y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i \tag{12.6}$$

$\forall i$, $\xi_i \geq 0$, $\sum_{i=1}^{N} \xi_i \leq$ constant. For the subsequent optimization, we make the penalty function as a linear function of the distance. Naturally, $\xi_i = 0$ for the points which are on the correct side of the margin (See Figure 12.1), $\xi_i = 1$ for the points of the decision boundary, $\xi_i < 1$ in between the decision boundary and the class-margin and $\xi_i > 1$ for misclassification. While the slack variables allow for overlapping class distributions, this framework is still sensitive to outliers because the penalty for misclassification increases linearly with ξ.

We can now set our goal to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary. We therefore minimize:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i \tag{12.8}$$

$$\text{subject to } \xi_i \geq 0,\ y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i\ \forall_i,$$

where the cost parameter $C > 0$, controls the trade-off between the slack variable penalty and the margin. The parameter C is similar to, but inverse of regularization parameter. For separable case, $C = \infty$. Here we can write the corresponding Lagrange (primal) function:

$$L_p(\beta, \beta_0, \alpha_i, \xi_i, \mu_i) = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left[y_i(x_i^T \beta + \beta_0) - (1 - \xi_i)\right] - \sum_{i=1}^{N} \mu_i \xi_i \tag{12.9}$$

where $(\alpha_i \geq 0)$ and $(\mu_i \geq 0)$ are Lagrange multipliers. We minimize (12.9) w.r.t β, β_0, and ξ_i. Setting the respective derivatives to zero, we get

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i \,, \tag{12.10}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i \,, \tag{12.11}$$

$$\alpha_i = C - \mu_i, \ \forall_i \tag{12.12}$$

as well as the positivity constraints $\alpha_i, \mu_i, \xi_i \geq 0 \ \forall_i$. By substituting (12.10)–(12.12) into (12.9), we obtain the Lagrangian (Wolfe) dual objective function

$$L_D(\alpha_i) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k \tag{12.13}$$

which gives a lower bound on the objective function (12.8) for any feasible point. We maximize L_D subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$. In addition to (12.10)–(12.12), the Karush–Kuhn–Tucker conditions include the constraints

$$\alpha_i [y_i (x_i^T \beta + \beta_0) - (1 - \xi_i)] = 0, \tag{12.14}$$

$$\mu_i \xi_i = 0, \tag{12.15}$$

$$y_i (x_i^T \beta + \beta_0) - (1 - \xi_i) \geq 0, \tag{12.16}$$

for $i = 1, \ldots, N$. Together, these equations (12.10)–(12.16) uniquely characterize the solution to the primal and dual problem.

From (12.10) we see that the solution for β has the form

$$\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i \tag{12.17}$$

with nonzero coefficients $\hat{\alpha}_i$ only for those observations i for which the constraints in (12.16) are exactly met (due to (12.14)). These observations

are called the support vectors since $\hat{\beta}$ is represented in terms of them alone. Among these support points, some will lie on the edge of the margin ($\xi_i = 0$) and hence from (12.15) and (12.12) will be characterized by $0 \leq \hat{\alpha}_i \leq C$; the remainder ($\hat{\xi}_i > 0$) have $\hat{\alpha}_i = C$. From (12.14) we can see that any of these margin points ($0 < \hat{\alpha}_i$, $\hat{\xi}_i = 0$) can be used to solve for β_0, and we typically use an average of all the solutions for numerical stability.

Maximizing the dual (12.13) is a simpler convex quadratic programming problem than the primal (12.9), and can be solved with standard techniques (Murray et al., 1981, for example).

Given the solutions $\hat{\beta}_0$ and $\hat{\beta}$, the decision function can be written as

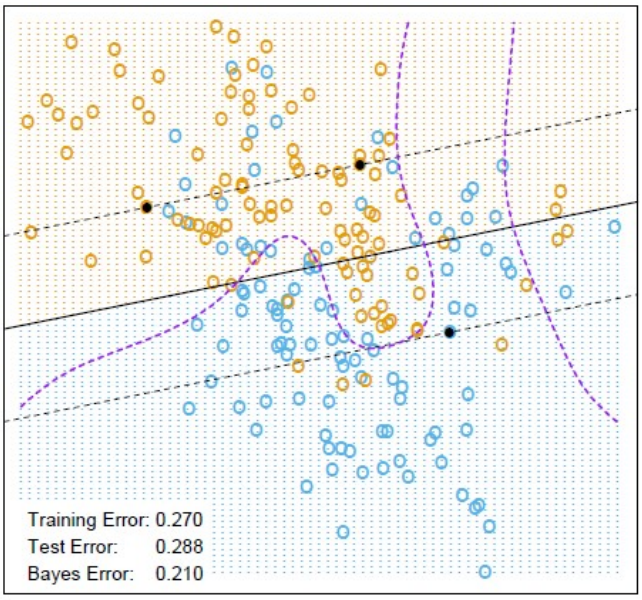
$$\begin{aligned} \hat{G}(x) &= \text{sign}\,[\hat{f}(x)] \\ &= \text{sign}\,[x^T\hat{\beta} + \hat{\beta}_0]. \end{aligned} \tag{12.18}$$

The tuning parameter of this procedure is the cost parameter C.

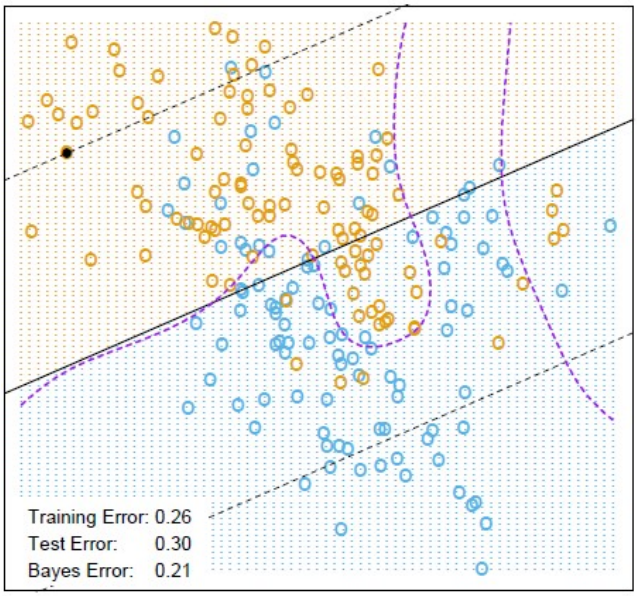
12.2.2 **Mixture Example (Continued)**

Figure 12.2 shows the support vector boundary for the mixture example of Figure 2.5 on page 21 (of [1]), with two overlapping classes, for two different values of the cost parameter C. The classifiers are rather similar in their performance. Points on the wrong side of the boundary are support vectors. Besides, points on the correct side of the boundary but close to it (in the margin), are also support vectors. The margin is larger for C = 0.01 than it is for C = 10, 000. Hence larger values of C focus attention more on (correctly classified) points near the decision boundary, while smaller values involve data further away. Either way, misclassified points are given weight, no matter how far away. In this example, the procedure is not very sensitive to choices of C, because of the rigidity of a linear boundary.

The optimal value for C can be estimated by cross-validation. Interestingly, the leave-one-out cross-validation error can be bounded above by the proportion of support points in the data. The reason is that leaving out an observation that is not a support vector will not change the solution. Hence these observations, being classified correctly by the original boundary, will be classified correctly in the cross-validation process. However, this bound tends to be too high, and not generally useful for choosing C (62% and 85%, respectively, in our examples).



$C = 10000$



$C = 0.01$

FIGURE 12.2. The linear support vector boundary for the mixture data example with two overlapping classes, for two different values of C. The broken lines indicate the margins, where $f(x) = \pm 1$. The support points ($\alpha_i > 0$) are all the points on the wrong side of their margin. The black solid dots are those support points falling exactly on the margin ($\xi_i = 0$, $\alpha_i > 0$). In the upper panel, 62% of the observations are support points, while in the lower panel 85% are. The broken purple curve in the background is the Bayes decision boundary.

12.3 **Support Vector Machines and Kernels**

The support vector classifier described so far finds linear boundaries in the input feature space. As with other linear methods, we can make the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines. Generally, linear boundaries in the enlarged space achieve better training-class separation and translate to nonlinear boundaries in the original space. Once the basis functions $h_m(x)$, $m = 1, \ldots, M$ are selected, the procedure is the same as before. We fit the Support Vector classifier using input features $h(x_i) = (h_1(x_i), h_2(x_i), \ldots, h_M(x_i))$, $i = 1, \ldots, N$, and produce the (nonlinear) function $\hat{f}(x) = h(x)^T \hat{\beta} + \hat{\beta}_0$. The classifier is $\hat{G}(x) = sign(\hat{f}(x))$ as before.

The support vector machine classifier is an extension of this idea, where the dimension of the enlarged space is allowed to get very large, infinite in some cases. It might seem that the computations would become prohibitive. It would also seem that with sufficient basis functions, the data would be separable, and overfitting would occur. We will show how the SVM technology deals with these issues.

12.3.1 **Computing the SVM for Classification**

We can represent the optimization problem (12.9) and its solution in a special way that only involves the input features via inner products. We do this directly for the transformed feature vectors $h(x_i)$. We then see that for particular choices of h, these inner products can be computed very cheaply. The Lagrange dual function (12.13) has the form

$$L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k \langle h(x_i), h(x_k) \rangle . \tag{12.19}$$

From (12.10) we see that the solution function $f(x)$ can be written

$$\begin{aligned} f(x) &= h(x)^T \beta + \beta_0 \\ &= \sum_{i=1}^{N} \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0 \end{aligned} \tag{12.20}$$

As before, given α_i, β_0 can be determined by solving $y_i f(x_i) = 1$ in (12.20) for any (or all) x_i for which $0 < \alpha_i < C$.

So both (12.19) and (12.20) involve $h(x)$ only through inner products. In fact, we need not specify the transformation $h(x)$ at all, but require only knowledge of the kernel function

$$K(x, x') = \langle h(x), h(x') \rangle \tag{12.21}$$

that computes inner products in the transformed space. K should be a symmetric positive (semi-) definite function; see Section 5.8.1. of [1] (self study for interested student).

Some popular choices for K in the SVM literature are

$$\begin{aligned} d^{\text{th}}\text{-Degree polynomial: } K(x, x') &= (1 + \langle x, x' \rangle)^d \\ \text{Radial basis: } K(x, x') &= \exp(-\gamma \|x - x'\|^2) \end{aligned} \tag{12.22}$$

Consider for example a feature space with two inputs X_1 and X_2, and a polynomial kernel of degree 2. Then

$$\begin{aligned} K(X, X') &= (1 + \langle X, X' \rangle)^2 = (1 + X^T X')^2 \qquad \left[\because\ X^T = \begin{bmatrix} X_1 & X_2 \end{bmatrix},\ X' = \begin{bmatrix} X_1' \\ X_2' \end{bmatrix}\right] \\ &= (1 + X_1 X_1' + X_2 X_2')^2 \\ &= 1 + 2X_1 X_1' + 2X_2 X_2' + (X_1 X')^2 + 2X_1 X_1' X_2 X_2' + (X_2 X_2')^2 \\ &= \begin{bmatrix} 1 & \sqrt{2}X_1 & \sqrt{2}X_2 & X_1^2 & \sqrt{2}X_1 X_2 & X_2^2 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{2}X_1' \\ \sqrt{2}X_2' \\ X_1'^2 \\ \sqrt{2}X_1' X_2' \\ X_2'^2 \end{bmatrix} \\ &= \phi(X)^T\ \phi(X') \end{aligned} \tag{12.23}$$

This kernel function therefore represents an inner product in a feature space having (a mapping from 2 dimensions to) 6 dimensions (i.e., $\Re^2 \Rightarrow \Re^6$), in which the mapping from input space to feature space is described by the vector function $\phi(X)$ for example.

Therefore, in terms of equation (12.21), we can write: $h_1(X) = 1,\ h_2(X) = \sqrt{2}X_1,\ h_3(X) = \sqrt{2}X_2,\quad h_4(X) = X_1^2,\quad h_5(X) = \sqrt{2}X_1 X_2,$ and

$h_6(X) = X_2^2$, and thus, $K(X, X') = \langle h(X), h(X') \rangle$. From (12.20) we see that the solution can be written

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{\alpha}_i y_i K(x, x_i) + \hat{\beta}_0 \qquad (12.24)$$

The redial basis or Gaussian kernel, i.e., $\exp(-\gamma \|x - x'\|^2)$, is very commonly used. The radial basis function can also be expressed as (similar to 12.23) an inner product in infinite-dimensional space. Assuming, $x \in \Re^1$ and $\gamma > 0$, we can write:

$$\begin{aligned}
K(x, x') &= \exp(-\gamma \|x - x'\|^2) \\
&= \exp(-\gamma (x - x')^2) \\
&= \exp(-\gamma x^2 + 2\gamma x x' - \gamma x'^2) \\
&= \exp(-\gamma x^2).\exp(-\gamma x'^2).\exp(2\gamma x x') \\
&= \exp(-\gamma x^2).\exp(-\gamma x'^2).\left(1 + \frac{2\gamma x x'}{1!} + \frac{(2\gamma x x')^2}{2!} + \frac{(2\gamma x x')^3}{3!} + \ldots\right) \\
&\qquad \left[\because e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots\right]
\end{aligned}$$

$$= \exp(-\gamma x^2).\exp(-\gamma x'^2).\left(1.1 + \sqrt{\frac{2\gamma}{1!}}x.\sqrt{\frac{2\gamma}{1!}}x' + \sqrt{\frac{(2\gamma)^2}{2!}}x^2.\sqrt{\frac{(2\gamma)^2}{2!}}x'^2 + \sqrt{\frac{(2\gamma)^3}{3!}}x^3.\sqrt{\frac{(2\gamma)^3}{3!}}x'^3 + \ldots\right)$$

$$= \left(\exp(-\gamma x^2).\left[1 \quad \sqrt{\frac{2\gamma}{1!}}x \quad \sqrt{\frac{(2\gamma)^2}{2!}}x^2 \quad \ldots\right]\right) \left(\exp(-\gamma x'^2).\begin{bmatrix} 1 \\ \sqrt{\frac{2\gamma}{1!}}x' \\ \sqrt{\frac{(2\gamma)^2}{2!}}x'^2 \\ \vdots \end{bmatrix}\right)$$

$$= \phi(x)^T \phi(x')$$

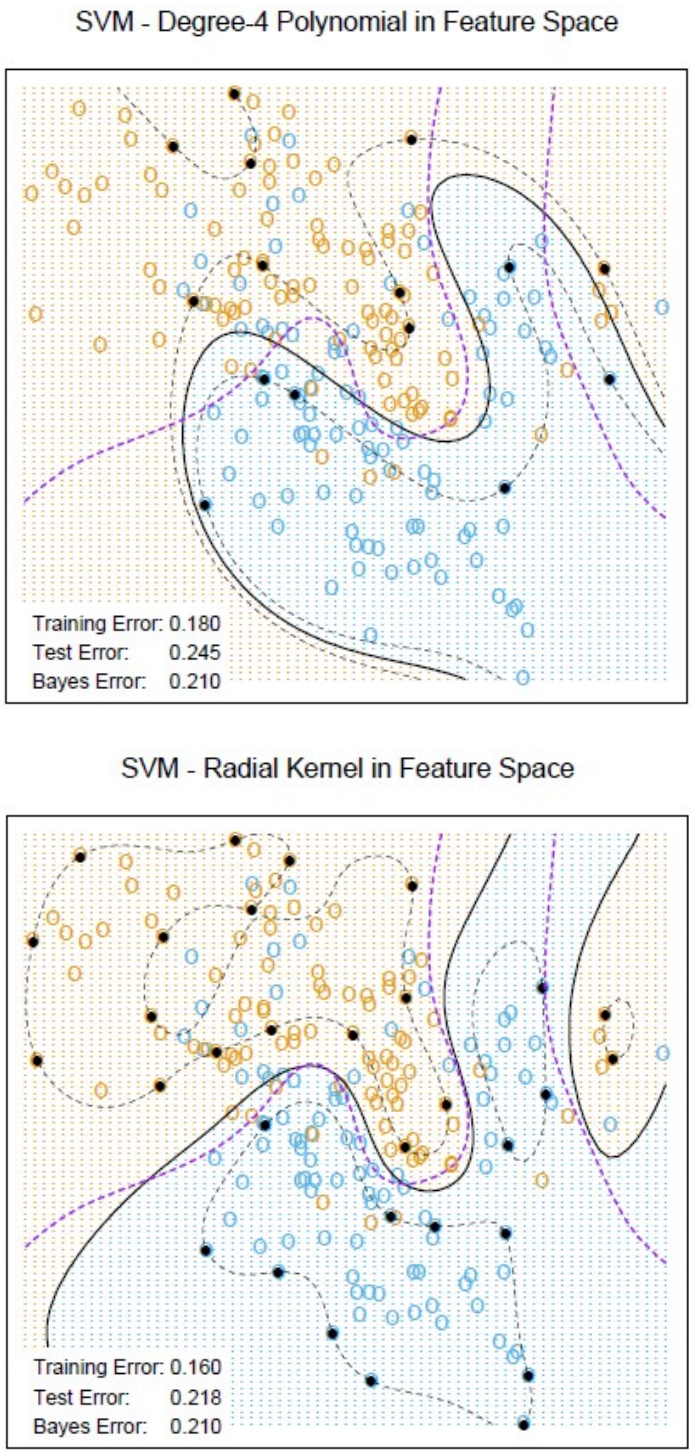


FIGURE 12.3. Two nonlinear SVMs for the mixture data. The upper plot uses a 4th-degree polynomial kernel, the lower a radial basis kernel (with $\gamma = 1$). In each case C was tuned to approximately achieve the best test error performance, and $C = 1$ worked well in both cases. The radial basis kernel performs the best (close to Bayes optimal), as might be expected given the data arise from mixtures of Gaussians. The broken purple curve in the background is the Bayes decision boundary.

The role of the parameter C is clearer in an enlarged feature space since perfect separation is often achievable there. A large value of C will discourage any positive ξ_i, and lead to an overfit wiggly boundary in the original feature space; a small value of C will encourage a small value of $\|\beta\|$, which in turn causes $f(x)$ and hence the boundary to be smoother. Figure 12.3 shows two nonlinear support vector machines applied to the mixture example of Chapter 2 of [1]. The regularization parameter was chosen in both cases to achieve good test error. The radial basis kernel produces a boundary quite similar to the Bayes optimal boundary for this example; compare Figure 2.5 of [1].

Constructing Kernels (Ch 6 of [2]**)**

In order to exploit kernel substitution, we need to be able to construct valid kernel functions. One approach is to choose a feature space mapping $\phi(x)$ and then use this to find the corresponding kernel. Here the kernel function is defined for a one-dimensional input space by

$$K(x,x') = \phi(x)^T \phi(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x') \qquad (iv)$$

where $\phi_i(x)$ are the basis functions and M is the dimension.

An alternative approach is to construct kernel functions directly. In this case, we must ensure that the function we choose is a valid kernel, in other words, that it corresponds to a scalar product in some (perhaps infinite-dimensional) feature space. As a simple example, consider a kernel function given by

$$K(\mathbf{x},\mathbf{x}') = (\mathbf{x}^{\mathbf{T}}\mathbf{x}')^2 \qquad (v)$$

If we take the particular case of a two-dimensional input space $\mathbf{x} = (x_1, x_2)$ we can expand out the terms and thereby identify the corresponding nonlinear feature mapping

$$\begin{aligned} K(x,x') &= (\mathbf{x}^{\mathbf{T}}\mathbf{x}')^2 = (x_1x_1' + x_2x_2')^2 \\ &= x_1^2x_1'^2 + 2x_1x_1'x_2x_2' + x_2^2x_2'^2 \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \quad (x_1'^2, \sqrt{2}x_1'x_2', x_2'^2)^T \\ &= \phi(x)^T \ \phi(x') \end{aligned} \qquad (vi)$$

We see that the feature mapping takes the form $\phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$ and therefore comprises all possible second order terms, with a specific weighting between them.

More generally, however, we need a simple way to test whether a function constitutes a valid kernel without having to construct the function $\phi(x)$ explicitly. A necessary and sufficient condition for a function $K(x,x')$ to be a valid kernel (Shawe-Taylor and Cristianini, 2004) is that the Gram matrix **K**, whose elements are given by $K(x_n,x_m)$, should be positive semi-definite for all possible choices of the set $\{\mathrm{x_n}\}$.

One powerful technique for constructing new kernels is to build them out of simpler kernels as building blocks. This can be done using the following properties:

Given valid kernels $K_1(x,x')$ and $K_2(x,x')$, the following new kernels will also be valid:

$$K(x,x') = cK_1(x,x') \qquad (vii)$$
$$K(x,x') = f(x)\,K_1(x,x')\,f(x') \qquad (viii)$$
$$K(x,x') = q(K_1(x,x')) \qquad (ix)$$
$$K(x,x') = \exp(K_1(x,x')) \qquad (x)$$
$$K(x,x') = K_1(x,x') + K_2(x,x') \qquad (xi)$$
$$K(x,x') = K_1(x,x')\,K_2(x,x') \qquad (xii)$$
$$K(x,x') = K_3(\phi(x),\phi(x')) \qquad (xiii)$$
$$K(x,x') = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \qquad (xiv)$$
$$K(x,x') = K_a(x_a,x_a') + K_b(x_b,x_b') \qquad (xv)$$
$$K(x,x') = K_a(x_a,x_a')\,K_b(x_b,x_b') \qquad (xvi)$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is polynomial with nonnegative coefficients, $\phi(\mathbf{x})$ is a function from $\mathbf{x}$ to $\Re^M$, $K_3(\cdot,\cdot)$ is a valid kernel in $\Re^M$, A is a symmetric positive semi-definite matrix, $\mathbf{x}_\mathrm{a}$ and $\mathbf{x}_\mathrm{b}$ are variables (not necessarily disjoint) with $\mathbf{x} = (x_\mathrm{a}, x_\mathrm{b})$, and K_a and K_b are valid kernel functions over their respective spaces.

Equipped with these properties, we can now embark on the construction of more complex kernels appropriate to specific applications. We require that the kernel $K(x, x')$ be symmetric and positive semi-definite and that it expresses the appropriate form of similarity between x and x' according to the intended application. Here we consider a few common

examples of kernel functions. For a more extensive discussion of 'kernel engineering', see Shawe-Taylor and Cristianini (2004).

We saw that the simple polynomial kernel $K(x, x') = (x^{\mathrm{T}} x')^2$ contains only terms of degree two. If we consider the slightly generalized kernel $K(x, x') = (x^T x' + c)^2$ with $c > 0$, then the corresponding feature mapping $\phi(\mathbf{x})$ contains constant and linear terms as well as terms of order two (see equation (12.23)). Similarly, $K(x, x') = (x^T x')^M$ contains all monomials of order M. For instance, if x and x' are two images, then the kernel represents a particular weighted sum of all possible products of M pixels in the first image with M pixels in the second image. This can similarly be generalized to include all terms up to degree M by considering $K(x, x') = (x^T x' + c)^M$ with c > 0. Using the results (*xi*) and (*xii*) for combining kernels, we see that these will all be valid kernel functions. Another commonly used kernel takes the form

Another commonly used kernel takes the form (as we have seen before):

$$K(x, x') = \exp(-\gamma \|x - x'\|^2) \qquad (xvii)$$

On page 15, we have shown that this radial basis or Gaussian kernel can be expressed as inner product and hence is a valid kernel. The feature vector that corresponds to the Gaussian kernel has infinite dimensionality.

An important contribution to arise from the kernel viewpoint has been the extension to inputs that are symbolic, rather than simply vectors of real numbers. Kernel functions can be defined over objects as diverse as graphs, sets, strings, and text documents. Consider, for instance, a fixed set and define a nonvectorial space consisting of all possible subsets of this set. If A_1 and A_2 are two such subsets then one simple choice of the kernel would be

$$K(A_1, A_2) = 2^{|A_1 \cap A_2|} \qquad (xviii)$$

where, $A_1 \cap A_2$ denotes the intersection of sets, A_1 and A_2, and $|A|$ denotes the number of elements in A. This is a valid kernel function because it can be shown to correspond to an inner product in a feature space.

One powerful approach to the construction of kernels starts from a probabilistic generative model (Haussler 1999), which allows us to apply generative models in a discriminative setting. Generative models can deal naturally with missing data and in the case of hidden Markov models can handle sequences of varying lengths. By contrast, discriminative models generally give better performance on discriminative tasks than generative

models. It is therefore of some interest to combine these two approaches (Lasserre *et al*., 2006). One way to combine them is to use a generative model to define a kernel, and then use this kernel in a discriminative approach.

Given a generative model $p(x)$ we can define a kernel by

$$K(\mathbf{x}, \mathbf{x}') = p(\mathbf{x})p(\mathbf{x}') \qquad (xix)$$

This is clearly a valid kernel function because we can interpret it as an inner product in the one-dimensional feature space defined by the mapping $p(x)$. It says that two inputs $\mathbf{x}$ and $\mathbf{x}'$ are similar if they both have high probabilities. We can use (*vii*) and (*xi*) to extend this class of kernels by considering sums over products of different probability distributions, with positive weighting coefficients $p(i)$, of the form

$$K(\mathbf{x}, \mathbf{x}') = \sum p(\mathbf{x} \mid i)p(\mathbf{x}' \mid i)p(i) \qquad (xx)$$

This is equivalent, up to an overall multiplicative constant, to a mixture distribution in which the components factorize, with the index i playing the role of a 'latent' variable. Two inputs $\mathbf{x}$ and $\mathbf{x}'$ will give a large value for the kernel function, and hence appear similar, if they have significant probability under a range of different components. Taking the limit of an infinite sum, we can also consider kernels of the form

$$K(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{x} \mid \mathbf{z})p(\mathbf{x}' \mid \mathbf{z})p(\mathbf{z})\,d\mathbf{z} \qquad (xxi)$$

where $\mathbf{z}$ is a continuous latent variable.

Now suppose that our data consists of ordered sequences of length L so that observation is given by $\mathbf{X} = \{\mathrm{x}_1, \mathrm{x}_2, ..., \mathrm{x}_L\}$. A popular generative model for sequences is the hidden Markov model, which expresses the distribution $p(\mathbf{X})$ as a marginalization over a corresponding sequence of hidden states $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_L\}$. We can use this approach to define a kernel function measuring the similarity of two sequences $\mathbf{X}$ and $\mathbf{X}'$ by extending the mixture representation (*xx*) to give

$$K(\mathbf{X}, \mathbf{X}') = \sum p(\mathbf{X} \mid \mathbf{Z})p(\mathbf{X}' \mid \mathbf{Z})p(\mathbf{Z}) \qquad (xxii)$$

so that both observed sequences are generated by the same hidden sequence **Z**. This model can easily be extended to allow sequences of differing lengths to be compared.

Multiclass SVM **(**Ch-7 of [2]**)**

The support vector machine is fundamentally a two-class classifier. In practice, however, we often have to tackle problems involving $K > 2$ classes. Various methods have therefore been proposed for combining multiple two-class SVMs in order to build a multiclass classifier.

(1) One way is to go by *one-versus-all* or *one-versus-the-rest* approach. Here, we construct K separate SVMs, where k^{th} model $y_k(\mathbf{x})$ is trained using the data from class C_k as the positive example and the data from the remaining (K-*1*) classes as the negative examples.

Unfortunately, this heuristic approach may suffer from the problem that the different classifiers were trained on different tasks, and there is no guarantee that the real-valued quantities $y_k(\mathbf{x})$ for different classifiers will have appropriate scales.

Another problem with the *one-versus-the-rest* approach is that the training sets are imbalanced. For instance, if we have ten classes each with equal numbers of training data points, then the individual classifiers are trained on data sets comprising 90% negative examples and only 10% positive examples, and the symmetry of the original problem may be lost. A variant of the *one-versus-the-rest* scheme was proposed by *Lee et al.* (2001) who modify the target values so that the positive class has the target: '+1' and the negative class has the target: '$\frac{-1}{(K-1)}$'.

Weston and Watkins (1999) define a single objective function for training all K SVMs simultaneously, based on maximizing the margin from each to remaining classes. However, this can result in much slower training because, instead of solving K separate optimization problems each over N data points with an overall cost of $O(K\,N^2)$, a single optimization problem of size (K-1) N must be solved giving an overall cost of $O(K^2\,N^2)$.

(2) Another approach is to train $\frac{K(K-1)}{2}$ different 2-class SVMs on all possible pairs of classes, and then to classify test points according to which class has the highest number of 'votes', an approach that is sometimes called *one-versus-one*. However, this approach may also suffer like a *one-versus-the-rest* approach sometimes. Also, for large K this approach requires

significantly more training time than the one-versus-the-rest approach. Similarly, to evaluate test points, significantly more computation is required.

(3) A different approach to multiclass classification, based on error-correcting output codes, was developed by Dietterich and Bakiri (1995) and applied to support vector machines by Allwein *et al.* (2000). This can be viewed as a generalization of the voting scheme of the *one-versus-one* approach in which more general partitions of the classes are used to train the individual classifiers. The K classes themselves are represented as particular sets of responses from the two-class classifiers chosen, and together with a suitable decoding scheme, this gives robustness to errors and to ambiguity in the outputs of the individual classifiers.

Although the application of SVMs to multiclass classification problems remains an open issue, in practice the *one-versus-the-rest* approach is the most widely used in spite of its ad-hoc formulation and its practical limitations.

Optional reading: ***Support Vector Machines for Regression*** **(**12.3.6, of [1]**)**.

Further reading: http://www.kernel-machines.org/ is a good source.

Appendix (A)

for chapter #5 (SVM)

Lagrange Multipliers/Lagrange Optimization (Ref: [2]):

Lagrange Multipliers, also known as *undetermined multipliers*, are used to find the stationary points of a function of several variables subject to one or more constraints.

Consider the problem of finding the maximum of a function $f(x_1, x_2)$ subject to a constraint relating x_1 and x_2, which we write in the form.

$$g(x_1, x_2) = 0 \tag{A.1}$$

To solve, a more elegant, and often simpler, approach is based on the introduction of a parameter λ called a *Lagrange multiplier*. Then, the problem can be solved by optimizing the *Lagrangian function*

$$L(\mathrm{x}, \lambda) \equiv f(\mathrm{x}) + \lambda\, g(\mathrm{x}) \tag{A.2}$$

We then differentiate A.2 with respect to the variables (i.e., x, λ) and equate the equation(s) to zero and then we get others equations, using which we can get suitable solutions.

As an example, suppose we wish to find the stationary points of the function $f(x_1, x_2) = 1 - x_1^2 - x_2^2$ subject to the constraint $g(x_1, x_2) => x_1 + x_2 - 1 = 0$. The corresponding *Lagrangian function* is given by:

$$L(\mathrm{x}, \lambda) = 1 - x_1^2 - x_2^2 + \lambda\,(x_1 + x_2 - 1) \tag{A.3}$$

The conditions for this *Lagrangian* to be stationary with respect to x_1 , x_2 and λ give the following coupled equations:

$$-2x_1 + \lambda = 0 \tag{A.4}$$
$$-2x_2 + \lambda = 0 \tag{A.5}$$
$$x_1 + x_2 - 1 = 0 \tag{A.6}$$

The solution of these equations then gives the stationary point as $(x_1^*, x_2^*) = \left(\frac{1}{2}, \frac{1}{2}\right)$ and the corresponding value for the Lagrange multiplier is $\lambda = 1$.

So far, we have considered the problem of maximizing a function subject to an ***equality constraint*** of the form $g(x) = 0$. We now consider the problem of maximizing $f(x)$ subject to an ***inequality constraint*** of the form $g(x) \geq 0$.

There are now **two** kinds of solution possible, according to whether the constrained stationary point lies in the region where $g(x) > 0$, in which case the constraint is *inactive*, or whether it lies on the boundary $g(x) = 0$, in which case the constraint is said to be *active*.

(1) In the former case, the function $g(x)$ plays no role, and so the stationary condition is simply $\nabla f(x) = 0$. This again corresponds to a stationary point of the Lagrange function (A.2) but this time with $\lambda = 0$.

(2) The latter case, where the solution lies on the boundary, is analogous to the equality constraint discussed previously and corresponded to a stationary point of the Lagrange function (A.2) with $\lambda \neq 0$.

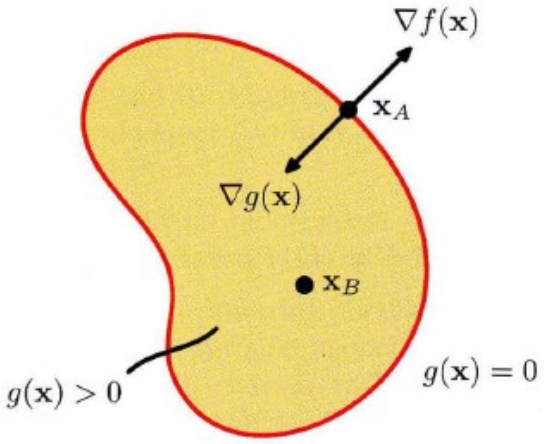


Figure A.1: Illustration of the problem of maximizing $f(x)$ subject to the inequality constraint $g(x) \geq 0$.

Now, however, the sign of the Lagrange multiplier is crucial, because the function $f(x)$ will only be at a maximum if its gradient is oriented away from the region $g(x) > 0$, as illustrated in Figure (A.1). We, therefore, have $\nabla f(x) = -\lambda \nabla g(x)$ for some value of $\lambda > 0$.

For either of these two cases, the product $\lambda g(x) = 0$. Thus the solution to the problem of maximizing $f(x)$ subject to $g(x) \geq 0$ is obtained by optimizing the Lagrange function (A.2) with respect to x and λ subject to the conditions

$$g(x) \geq 0 \quad \text{(A.7)}$$

$$\lambda \geq 0 \quad \text{(A.8)}$$

$$\lambda g(x) = 0 \quad \text{(A.9)}$$

These are known as the *Karush-Kuhn-Tucker* (**KKT**) conditions (Karush, 1939; Kuhn and Tucker, 1951).

Note that if we wish to minimize (rather than maximize) the function $f(x)$ subject to an inequality constraint $g(x) \geq 0$, then we minimize the Lagrangian function $L(x, \lambda) = f(x) - \lambda\, g(x)$ with respect to x, again subject to $\lambda \geq 0$.

Finally, it is straightforward to extend the technique of Lagrange multipliers to the case of multiple equality and inequality constraints. For a more detailed discussion of the technique of Lagrange multipliers, see Nocedal and Wright (1999).

References:

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*: Springer, 2009.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*: Springer, 2009.

Next, we attach,

"A Practical Guide to Support Vector Classification",
by Lin *et al*.

and will see the corresponding implementations using *libsvm*.

A Practical Guide to Support Vector Classification

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin

Department of Computer Science

National Taiwan University, Taipei 106, Taiwan

http://www.csie.ntu.edu.tw/~cjlin

Initial version: 2003 Last updated: April 15, 2010

Abstract

The support vector machine (SVM) is a popular classification technique. However, beginners who are not familiar with SVM often get unsatisfactory results since they miss some easy but significant steps. In this guide, we propose a simple procedure which usually gives reasonable results.

1 Introduction

SVMs (Support Vector Machines) are a useful technique for data classification. Although SVM is considered easier to use than Neural Networks, users not familiar with it often get unsatisfactory results at first. Here we outline a "cookbook" approach which usually gives reasonable results.

Note that this guide is not for SVM researchers nor do we guarantee you will achieve the highest accuracy. Also, we do not intend to solve challenging or difficult problems. Our purpose is to give SVM novices a recipe for rapidly obtaining acceptable results.

Although users do not need to understand the underlying theory behind SVM, we briefly introduce the basics necessary for explaining our procedure. A classification task usually involves separating data into training and testing sets. Each instance in the training set contains one "target value" (i.e. the class labels) and several "attributes" (i.e. the features or observed variables). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.

Given a training set of instance-label pairs $(\mathbf{x}_i, y_i), i = 1, \ldots, l$ where $\mathbf{x}_i \in R^n$ and $\mathbf{y} \in \{1, -1\}^l$, the support vector machines (SVM) (Boser et al., 1992; Cortes and Vapnik, 1995) require the solution of the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{1}$$

Table 1: Problem characteristics and performance comparisons.

Applications	#training data	#testing data	#features	#classes	Accuracy by users	Accuracy by our procedure
Astroparticle[1]	3,089	4,000	4	2	75.2%	96.9%
Bioinformatics[2]	391	0[4]	20	3	36%	85.2%
Vehicle[3]	1,243	41	21	2	4.88%	87.8%

Here training vectors $\mathbf{x}_i$ are mapped into a higher (maybe infinite) dimensional space by the function ϕ. SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. $C > 0$ is the penalty parameter of the error term. Furthermore, $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ is called the kernel function. Though new kernels are being proposed by researchers, beginners may find in SVM books the following four basic kernels:

- linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.
- polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma {\mathbf{x}_i}^T \mathbf{x}_j + r)^d$, $\gamma > 0$.
- radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$.
- sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma {\mathbf{x}_i}^T \mathbf{x}_j + r)$.

Here, γ, r, and d are kernel parameters.

1.1 Real-World Examples

Table 1 presents some real-world examples. These data sets are supplied by our users who could not obtain reasonable accuracy in the beginning. Using the procedure illustrated in this guide, we help them to achieve better performance. Details are in Appendix A.

These data sets are at `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/data/`

[1] Courtesy of Jan Conrad from Uppsala University, Sweden.

[2] Courtesy of Cory Spencer from Simon Fraser University, Canada (Gardy et al., 2003).

[3] Courtesy of a user from Germany.

[4] As there are no testing data, cross-validation instead of testing accuracy is presented here. Details of cross-validation are in Section 3.2.

1.2 Proposed Procedure

Many beginners use the following procedure now:

- Transform data to the format of an SVM package
- Randomly try a few kernels and parameters
- Test

We propose that beginners try the following procedure first:

- Transform data to the format of an SVM package
- Conduct simple scaling on the data
- Consider the RBF kernel $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}$
- Use cross-validation to find the best parameter C and γ
- Use the best parameter C and γ to train the whole training set[5]
- Test

We discuss this procedure in detail in the following sections.

2 Data Preprocessing

2.1 Categorical Feature

SVM requires that each data instance is represented as a vector of real numbers. Hence, if there are categorical attributes, we first have to convert them into numeric data. We recommend using m numbers to represent an m-category attribute. Only one of the m numbers is one, and others are zero. For example, a three-category attribute such as {red, green, blue} can be represented as (0,0,1), (0,1,0), and (1,0,0). Our experience indicates that if the number of values in an attribute is not too large, this coding might be more stable than using a single number.

[5]The best parameter might be affected by the size of data set but in practice the one obtained from cross-validation is already suitable for the whole training set.

2.2 Scaling

Scaling before applying SVM is very important. Part 2 of Sarle's Neural Networks FAQ Sarle (1997) explains the importance of this and most of considerations also apply to SVM. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. We recommend linearly scaling each attribute to the range $[-1, +1]$ or $[0, 1]$.

Of course we have to use the same method to scale both training and testing data. For example, suppose that we scaled the first attribute of training data from $[-10, +10]$ to $[-1, +1]$. If the first attribute of testing data lies in the range $[-11, +8]$, we must scale the testing data to $[-1.1, +0.8]$. See Appendix B for some real examples.

3 Model Selection

Though there are only four common kernels mentioned in Section 1, we must decide which one to try first. Then the penalty parameter C and kernel parameters are chosen.

3.1 RBF Kernel

In general, the RBF kernel is a reasonable first choice. This kernel nonlinearly maps samples into a higher dimensional space so it, unlike the linear kernel, can handle the case when the relation between class labels and attributes is nonlinear. Furthermore, the linear kernel is a special case of RBF Keerthi and Lin (2003) since the linear kernel with a penalty parameter $\tilde{C}$ has the same performance as the RBF kernel with some parameters (C, γ). In addition, the sigmoid kernel behaves like RBF for certain parameters (Lin and Lin, 2003).

The second reason is the number of hyperparameters which influences the complexity of model selection. The polynomial kernel has more hyperparameters than the RBF kernel.

Finally, the RBF kernel has fewer numerical difficulties. One key point is $0 < K_{ij} \leq 1$ in contrast to polynomial kernels of which kernel values may go to infinity $(\gamma {\mathbf{x}_i}^T \mathbf{x}_j + r > 1)$ or zero $(\gamma {\mathbf{x}_i}^T \mathbf{x}_j + r < 1)$ while the degree is large. Moreover, we must note that the sigmoid kernel is not valid (i.e. not the inner product of two

vectors) under some parameters (Vapnik, 1995).

There are some situations where the RBF kernel is not suitable. In particular, when the number of features is very large, one may just use the linear kernel. We discuss details in Appendix C.

3.2 Cross-validation and Grid-search

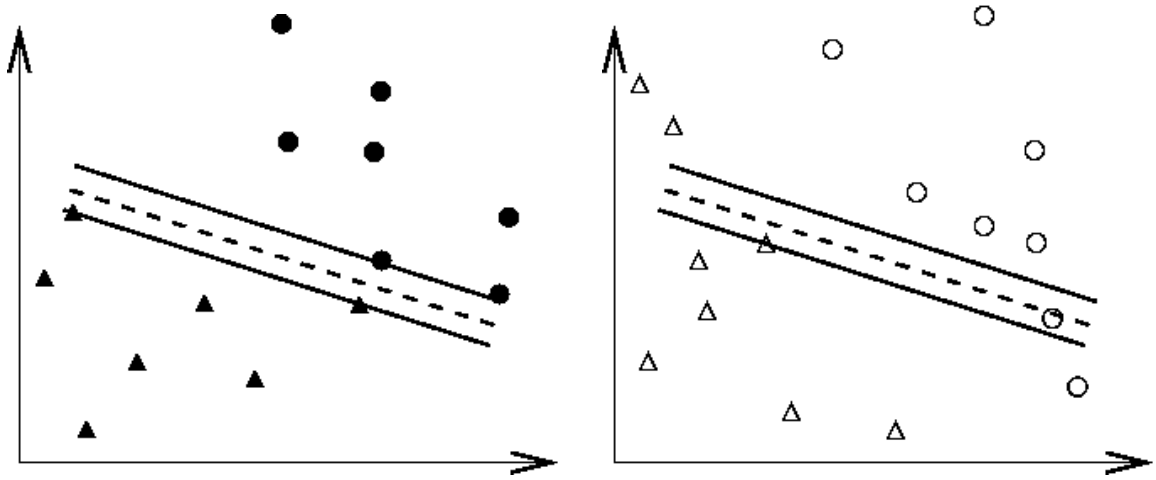
There are two parameters for an RBF kernel: C and γ. It is not known beforehand which C and γ are best for a given problem; consequently some kind of model selection (parameter search) must be done. The goal is to identify good (C, γ) so that the classifier can accurately predict unknown data (i.e. testing data). Note that it may not be useful to achieve high training accuracy (i.e. a classifier which accurately predicts training data whose class labels are indeed known). As discussed above, a common strategy is to separate the data set into two parts, of which one is considered unknown. The prediction accuracy obtained from the "unknown" set more precisely reflects the performance on classifying an independent data set. An improved version of this procedure is known as cross-validation.

In v-fold cross-validation, we first divide the training set into v subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining $v - 1$ subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data which are correctly classified.

The cross-validation procedure can prevent the overfitting problem. Figure 1 represents a binary classification problem to illustrate this issue. Filled circles and triangles are the training data while hollow circles and triangles are the testing data. The testing accuracy of the classifier in Figures 1a and 1b is not good since it overfits the training data. If we think of the training and testing data in Figure 1a and 1b as the training and validation sets in cross-validation, the accuracy is not good. On the other hand, the classifier in 1c and 1d does not overfit the training data and gives better cross-validation as well as testing accuracy.

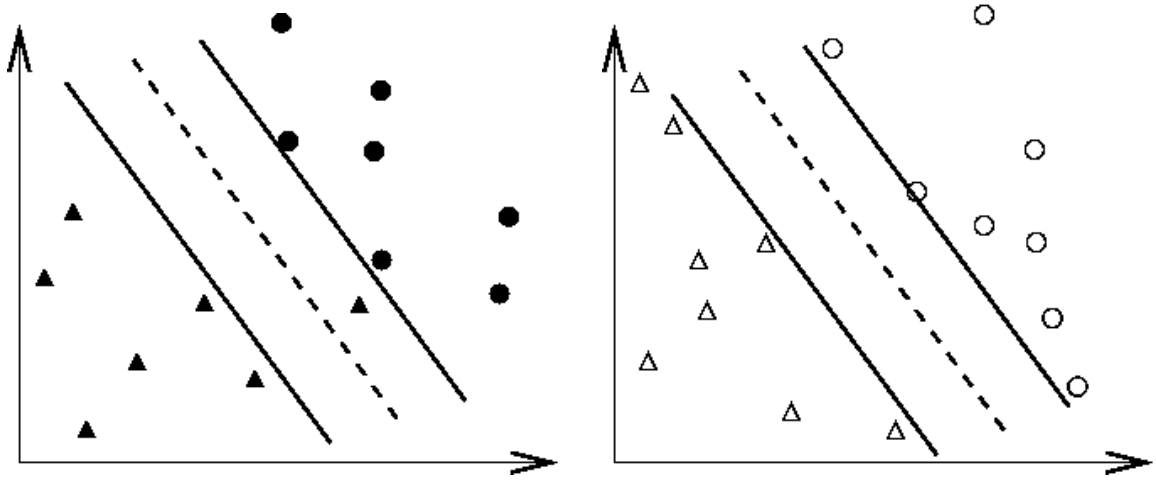
We recommend a "grid-search" on C and γ using cross-validation. Various pairs of (C, γ) values are tried and the one with the best cross-validation accuracy is picked. We found that trying exponentially growing sequences of C and γ is a practical method to identify good parameters (for example, $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$, $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$).

The grid-search is straightforward but seems naive. In fact, there are several advanced methods which can save computational cost by, for example, approximating the cross-validation rate. However, there are two motivations why we prefer the simple



(a) Training data and an overfitting classifier

(b) Applying an overfitting classifier on testing data



(c) Training data and a better classifier

(d) Applying a better classifier on testing data

Figure 1: An overfitting classifier and a better classifier (● and ▲: training data; ○ and △: testing data).

grid-search approach.

One is that, psychologically, we may not feel safe to use methods which avoid doing an exhaustive parameter search by approximations or heuristics. The other reason is that the computational time required to find good parameters by grid-search is not much more than that by advanced methods since there are only two parameters. Furthermore, the grid-search can be easily parallelized because each (C, γ) is independent. Many of advanced methods are iterative processes, e.g. walking along a path, which can be hard to parallelize.

Since doing a complete grid-search may still be time-consuming, we recommend

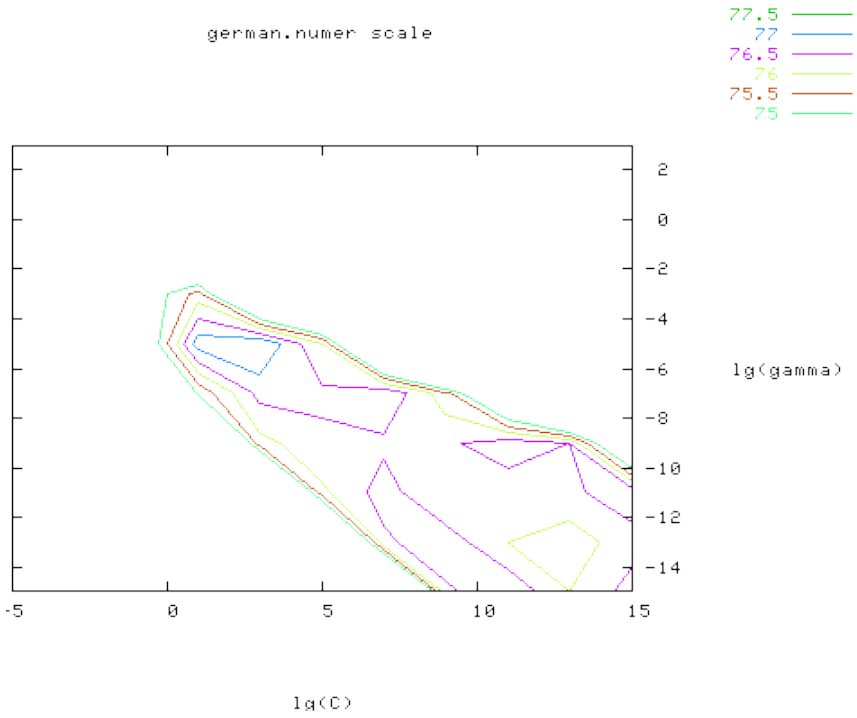


Figure 2: Loose grid search on $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$ and $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$.

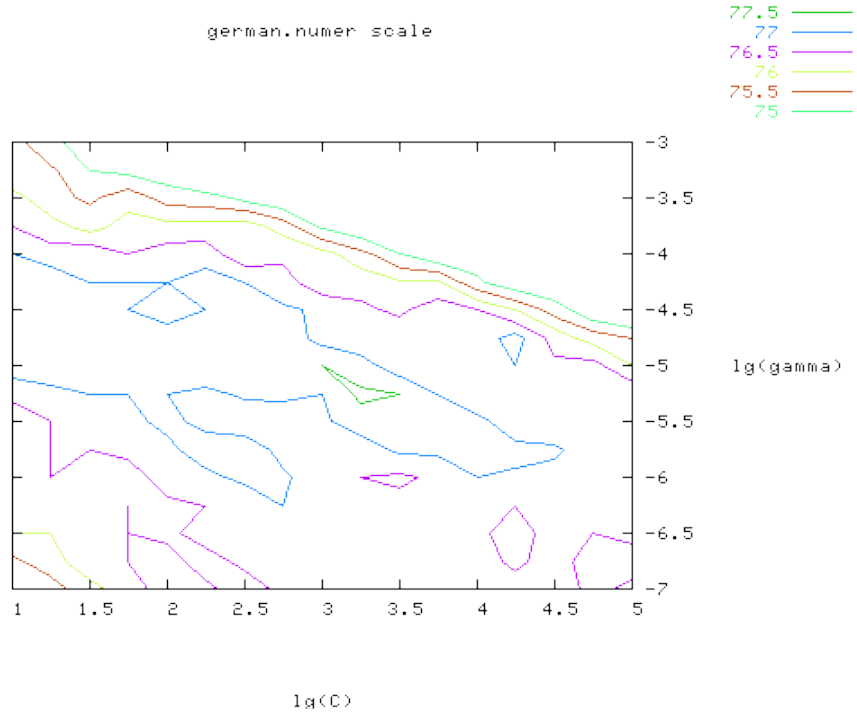


Figure 3: Fine grid-search on $C = 2^{1}, 2^{1.25}, \ldots, 2^{5}$ and $\gamma = 2^{-7}, 2^{-6.75}, \ldots, 2^{-3}$.

using a coarse grid first. After identifying a "better" region on the grid, a finer grid search on that region can be conducted. To illustrate this, we do an experiment on the problem `german` from the Statlog collection (Michie et al., 1994). After scaling this set, we first use a coarse grid (Figure 2) and find that the best (C, γ) is $(2^3, 2^{-5})$ with the cross-validation rate 77.5%. Next we conduct a finer grid search on the neighborhood of $(2^3, 2^{-5})$ (Figure 3) and obtain a better cross-validation rate 77.6% at $(2^{3.25}, 2^{-5.25})$. After the best (C, γ) is found, the whole training set is trained again to generate the final classifier.

The above approach works well for problems with thousands or more data points. For very large data sets a feasible approach is to randomly choose a subset of the data set, conduct grid-search on them, and then do a better-region-only grid-search on the complete data set.

4 Discussion

In some situations the above proposed procedure is not good enough, so other techniques such as feature selection may be needed. These issues are beyond the scope of this guide. Our experience indicates that the procedure works well for data which do not have many features. If there are thousands of attributes, there may be a need to choose a subset of them before giving the data to SVM.

Acknowledgments

We thank all users of our SVM software LIBSVM and BSVM, who helped us to identify possible difficulties encountered by beginners. We also thank some users (in particular, Robert Campbell) for proofreading the paper.

A Examples of the Proposed Procedure

In this appendix we compare accuracy by the proposed procedure with that often used by general beginners. Experiments are on the three problems mentioned in Table 1 by using the software LIBSVM (Chang and Lin, 2001). For each problem, we first list the accuracy by direct training and testing. Secondly, we show the difference in accuracy with and without scaling. From what has been discussed in Section 2.2, the range of training set attributes must be saved so that we are able to restore them while scaling the testing set. Thirdly, the accuracy by the proposed procedure

(scaling and then model selection) is presented. Finally, we demonstrate the use of a tool in LIBSVM which does the whole procedure automatically. Note that a similar parameter selection tool like the `grid.py` presented below is available in the R-LIBSVM interface (see the function `tune`).

A.1 Astroparticle Physics

- Original sets with default parameters

```
$ ./svm-train svmguide1
$ ./svm-predict svmguide1.t svmguide1.model svmguide1.t.predict
 → Accuracy = 66.925%
```

- Scaled sets with default parameters

```
$ ./svm-scale -l -1 -u 1 -s range1 svmguide1 > svmguide1.scale
$ ./svm-scale -r range1 svmguide1.t > svmguide1.t.scale
$ ./svm-train svmguide1.scale
$ ./svm-predict svmguide1.t.scale svmguide1.scale.model svmguide1.t.predict
 → Accuracy = 96.15%
```

- Scaled sets with parameter selection (change to the directory `tools`, which contains `grid.py`)

```
$ python grid.py svmguide1.scale
...
2.0 2.0 96.8922
```

(Best C=2.0, γ=2.0 with five-fold cross-validation rate=96.8922%)

```
$ ./svm-train -c 2 -g 2 svmguide1.scale
$ ./svm-predict svmguide1.t.scale svmguide1.scale.model svmguide1.t.predict
 → Accuracy = 96.875%
```

- Using an automatic script

```
$ python easy.py svmguide1 svmguide1.t
Scaling training data...
Cross validation...
```

```
Best c=2.0, g=2.0
Training...
Scaling testing data...
Testing...
Accuracy = 96.875% (3875/4000) (classification)
```

A.2 Bioinformatics

- Original sets with default parameters

```
$ ./svm-train -v 5 svmguide2
 → Cross Validation Accuracy = 56.5217%
```

- Scaled sets with default parameters

```
$ ./svm-scale -l -1 -u 1 svmguide2 > svmguide2.scale
$ ./svm-train -v 5 svmguide2.scale
 → Cross Validation Accuracy = 78.5166%
```

- Scaled sets with parameter selection

```
$ python grid.py svmguide2.scale
...
2.0 0.5 85.1662
 → Cross Validation Accuracy = 85.1662%
```

(Best C=2.0, γ=0.5 with five fold cross-validation rate=85.1662%)

- Using an automatic script

```
$ python easy.py svmguide2
Scaling training data...
Cross validation...
Best c=2.0, g=0.5
Training...
```

A.3 Vehicle

- Original sets with default parameters

```
$ ./svm-train svmguide3
$ ./svm-predict svmguide3.t svmguide3.model svmguide3.t.predict
 → Accuracy = 2.43902%
```

- Scaled sets with default parameters

```
$ ./svm-scale -l -1 -u 1 -s range3 svmguide3 > svmguide3.scale
$ ./svm-scale -r range3 svmguide3.t > svmguide3.t.scale
$ ./svm-train svmguide3.scale
$ ./svm-predict svmguide3.t.scale svmguide3.scale.model svmguide3.t.predict
 → Accuracy = 12.1951%
```

- Scaled sets with parameter selection

```
$ python grid.py svmguide3.scale
...
128.0 0.125 84.8753
```

(Best C=128.0, γ=0.125 with five-fold cross-validation rate=84.8753%)

```
$ ./svm-train -c 128 -g 0.125 svmguide3.scale
$ ./svm-predict svmguide3.t.scale svmguide3.scale.model svmguide3.t.predict
 → Accuracy = 87.8049%
```

- Using an automatic script

```
$ python easy.py svmguide3 svmguide3.t
Scaling training data...
Cross validation...
Best c=128.0, g=0.125
Training...
Scaling testing data...
Testing...
Accuracy = 87.8049% (36/41) (classification)
```

B Common Mistakes in Scaling Training and Testing Data

Section 2.2 stresses the importance of using the same scaling factors for training and testing sets. We give a real example on classifying traffic light signals (courtesy of an anonymous user) It is available at LIBSVM Data Sets.

If training and testing sets are separately scaled to $[0, 1]$, the resulting accuracy is lower than 70%.

```
$ ../svm-scale -l 0 svmguide4 > svmguide4.scale
$ ../svm-scale -l 0 svmguide4.t > svmguide4.t.scale
$ python easy.py svmguide4.scale svmguide4.t.scale
Accuracy = 69.2308% (216/312) (classification)
```

Using the same scaling factors for training and testing sets, we obtain much better accuracy.

```
$ ../svm-scale -l 0 -s range4 svmguide4 > svmguide4.scale
$ ../svm-scale -r range4 svmguide4.t > svmguide4.t.scale
$ python easy.py svmguide4.scale svmguide4.t.scale
Accuracy = 89.4231% (279/312) (classification)
```

With the correct setting, the 10 features in `svmguide4.t.scale` have the following maximal values:

0.7402, 0.4421, 0.6291, 0.8583, 0.5385, 0.7407, 0.3982, 1.0000, 0.8218, 0.9874

Clearly, the earlier way to scale the testing set to $[0, 1]$ generates an erroneous set.

C When to Use Linear but not RBF Kernel

If the number of features is large, one may not need to map data to a higher dimensional space. That is, the nonlinear mapping does not improve the performance. Using the linear kernel is good enough, and one only searches for the parameter C. While Section 3.1 describes that RBF is at least as good as linear, the statement is true only after searching the (C, γ) space.

Next, we split our discussion to three parts:

C.1 Number of instances $\ll$ number of features

Many microarray data in bioinformatics are of this type. We consider the Leukemia data from the LIBSVM data sets (`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`). The training and testing sets have 38 and 34 instances, respectively. The number of features is 7,129, much larger than the number of instances. We merge the two files and compare the cross validation accuracy of using the RBF and the linear kernels:

- RBF kernel with parameter selection

  ```
  $ cat leu leu.t > leu.combined
  $ python grid.py leu.combined
  ...
  8.0 3.0517578125e-05 97.2222
  ```

  (Best C=8.0, $\gamma = 0.000030518$ with five-fold cross-validation rate=97.2222%)

- Linear kernel with parameter selection

  ```
  $ python grid.py -log2c -1,2,1 -log2g 1,1,1 -t 0 leu.combined
   ...
  0.5 2.0 98.6111
  ```

  (Best C=0.5 with five-fold cross-validation rate=98.61111%)

  Though `grid.py` was designed for the RBF kernel, the above way checks various C using the linear kernel (`-log2g 1,1,1` sets a dummy γ).

The cross-validation accuracy of using the linear kernel is comparable to that of using the RBF kernel. Apparently, when the number of features is very large, one may not need to map the data.

In addition to LIBSVM, the LIBLINEAR software mentioned below is also effective for data in this case.

C.2 Both numbers of instances and features are large

Such data often occur in document classification. LIBSVM is not particularly good for this type of problems. Fortunately, we have another software LIBLINEAR (Fan et al., 2008), which is very suitable for such data. We illustrate the difference between

LIBSVM and LIBLINEAR using a document problem `rcv1_train.binary` from the LIBSVM data sets. The numbers of instances and features are 20,242 and 47,236, respectively.

```
$ time libsvm-2.85/svm-train -c 4 -t 0 -e 0.1 -m 800 -v 5 rcv1_train.binary
Cross Validation Accuracy = 96.8136%
345.569s
$ time liblinear-1.21/train -c 4 -e 0.1 -v 5 rcv1_train.binary
Cross Validation Accuracy = 97.0161%
2.944s
```

For five-fold cross validation, LIBSVM takes around 350 seconds, but LIBLINEAR uses only 3. Moreover, LIBSVM consumes more memory as we allocate some spaces to store recently used kernel elements (see `-m 800`). Clearly, LIBLINEAR is much faster than LIBSVM to obtain a model with comparable accuracy.

LIBLINEAR is efficient for large-scale document classification. Let us consider a large set `rcv1_test.binary` with 677,399 instances.

```
$ time liblinear-1.21/train -c 0.25 -v 5 rcv1_test.binary
Cross Validation Accuracy = 97.8538%
68.84s
```

Note that reading the data takes most of the time. The training of each training/validation split is *less than four seconds*.

C.3 Number of instances $\gg$ number of features

As the number of features is small, one often maps data to *higher dimensional spaces* (i.e., using nonlinear kernels). However, if you really would like to use the linear kernel, you may use LIBLINEAR with the option `-s 2`. When the number of features is small, it is often faster than the default `-s 1`. Consider the data `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/covtype.libsvm.binary.scale.bz2`. The number of instances 581,012 is much larger than the number of features 54. We run LIBLINEAR with `-s 1` (default) and `-s 2`.

```
$ time liblinear-1.21/train -c 4 -v 5 -s 2 covtype.libsvm.binary.scale
Cross Validation Accuracy = 75.67%
67.224s
$ time liblinear-1.21/train -c 4 -v 5 -s 1 covtype.libsvm.binary.scale
```

```
Cross Validation Accuracy = 75.6711%
452.736s
```

Clearly, using `-s 2` leads to shorter training time.

References

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

J. L. Gardy, C. Spencer, K. Wang, M. Ester, G. E. Tusnady, I. Simon, S. Hua, K. deFays, C. Lambert, K. Nakai, and F. S. Brinkman. PSORT-B: improving protein subcellular localization prediction for gram-negative bacteria. *Nucleic Acids Research*, 31(13):3613–3617, 2003.

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

H.-T. Lin and C.-J. Lin. A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. Technical report, Department of Computer Science, National Taiwan University, 2003. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/tanh.pdf`.

D. Michie, D. J. Spiegelhalter, C. C. Taylor, and J. Campbell, editors. *Machine learning, neural and statistical classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994. ISBN 0-13-106360-X. Data available at `http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/`.

W. S. Sarle. Neural Network FAQ, 1997. URL `ftp://ftp.sas.com/pub/neural/FAQ.html`. Periodic posting to the Usenet newsgroup comp.ai.neural-nets.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY, 1995.