Chapter 13
Nonlinear Classification Models

The previous chapter described models that were intrinsically linear—the structure of the model would produce linear class boundaries unless nonlinear functions of the predictors were manually specified. This chapter deals with some intrinsically nonlinear models. As in the regression sections, there are other nonlinear models that use trees or rules for modeling the data. These are discussed in the next chapter.

With a few exceptions (such as FDA models, Sect. 13.3), the techniques described in this chapter can be adversely affected when a large number of non-informative predictors are used as inputs. As such, combining these models with feature selection tools (described in Chap. 19) can significantly increase performance. The analyses shown in this chapter are conducted without supervised removal of non-informative predictors, so performance is likely to be less than what could be achieved with a more comprehensive approach.

13.1 Nonlinear Discriminant Analysis

We saw in the previous chapter that the linear boundaries of linear discriminant analysis came about by making some very specific assumptions for the underlying distributions of the predictors. In this section, we will explore ways that linear discriminant methods as described in the previous chapter are modified in order to handle data that are best separated by nonlinear structures. These methods include quadratic discriminant analysis (QDA), regularized discriminant analysis (RDA), and mixture discriminant analysis (MDA).

M. Kuhn and K. Johnson, *Applied Predictive Modeling*,
DOI 10.1007/978-1-4614-6849-3_13,

Quadratic and Regularized Discriminant Analysis

Recall that linear discriminant analysis could be formulated such that the trained model minimized the total probability of misclassification. The consequence of the assumption that the predictors in each class shared a common covariance structure was that the class boundaries were linear functions of the predictors.

In quadratic discriminant models, this assumption is relaxed so that a class-specific covariance structure can be accommodated. The primary repercussion of this change is that the decision boundaries now become quadratically curvilinear in the predictor space. The increased discriminant function complexity may improve model performance for many problems. However, another repercussion of this generalization is that the data requirements become more stringent. Since class-specific covariance matrices are utilized, the inverse of the matrices must exist. This means that the number of predictors must be less than the number of cases within each class. Also, the predictors within each class must not have pathological levels of collinearity. Additionally, if the majority of the predictors in the data are indicators for discrete categories, QDA will only to able to model these as linear functions, thus limiting the effectiveness of the model.

In pure mathematical optimization terms, LDA and QDA each minimize the total probability of misclassification assuming that the data can truly be separated by hyperplanes or quadratic surfaces. Reality may be, however, that the data are best separated by structures somewhere between linear and quadratic class boundaries. RDA, proposed by Friedman (1989), is one way to bridge the separating surfaces between LDA and QDA. In this approach, Friedman advocated the following covariance matrix:

$$\widetilde{\boldsymbol{\Sigma}}_\ell(\lambda) = \lambda \boldsymbol{\Sigma}_\ell + (1-\lambda)\,\boldsymbol{\Sigma}, \tag{13.1}$$

where $\boldsymbol{\Sigma}_\ell$ is the covariance matrix of the ℓth class and $\boldsymbol{\Sigma}$ is the pooled covariance matrix across all classes. It is easy to see that the tuning parameter, λ, enables the method to flex the covariance matrix between LDA (when $\lambda = 0$) and QDA (when $\lambda = 1$). If a model is tuned over λ, a data-driven approach can be used to choose between linear or quadratic boundaries as well as boundaries that fall between the two.

RDA makes another generalization of the data: the pooled covariance matrix can be allowed to morph from its observed value to one where the predictors are assumed to be independent (as represented by an identity matrix):

$$\boldsymbol{\Sigma}(\gamma) = \gamma \boldsymbol{\Sigma} + (1-\gamma)\,\sigma^2 \mathbf{I}, \tag{13.2}$$

where σ^2 is the common variance of all predictors and $\mathbf{I}$ is the identity matrix (i.e., the diagonal entries of the matrix are 1 and all other entries are 0), which forces the model to assume that all of the predictors are independent. Recall the familiar two-class example with two predictors, last seen in Chap. 4

(p. 69). There is a high correlation between these predictors indicating that γ values near 1 are most likely to be appropriate. However, in higher dimensions, it becomes increasingly more difficult to visually recognize such patterns, so tuning an RDA model over λ and γ enables the training set data to decide the most appropriate assumptions for the model. Note, however, that unless γ is one or λ is zero, the more stringent data standards of QDA must be applied.

Mixture Discriminant Analysis

MDA was developed by Hastie and Tibshirani (1996) as an extension of LDA. LDA assumes a distribution of the predictor data such that the class-specific means are different (but the covariance structure is independent of the classes). MDA generalizes LDA in a different manner; it allows each class to be represented by *multiple* multivariate normal distributions. These distributions can have different means but, like LDA, the covariance structures are assumed to be the same. Figure 13.1 presents this idea with a single predictor. Here, each class is represented by three normal distributions with different means and common variances. These are effectively sub-classes of the data. The modeler would specify how many different distributions should be used and the MDA model would determine their optimal locations in the predictor space.

How are the distributions aggregated so that a class prediction can be calculated? In the context of Bayes' Rule (Eq. 12.4), MDA modifies $Pr[X|Y = C_\ell]$. The class-specific distributions are combined into a single multivariate normal distribution by creating a per-class mixture. Suppose $D_{\ell k}(x)$ is the discriminant function for the kth subclass in the ℓth class, the overall discriminant function for the ℓth class would be proportional to

$$D_\ell(x) \propto \sum_{k=1}^{L_\ell} \phi_{\ell k} D_{\ell k}(x),$$

where L_ℓ is the number of distributions being used for the ℓth class and the $\phi_{\ell k}$ are the mixing proportions that are estimated during training. This overall discriminant function can then produce class probabilities and predictions.

For this model, the number of distributions per class is the tuning parameter for the model (they need not be equal per class). Hastie and Tibshirani (1996) describe algorithms for determining starting values for the class-specific means required for each distribution, along with numerical optimization routines to solve the nontrivial equations. Also, similar to LDA, Clemmensen et al. (2011) describe using ridge- and lasso-like penalties to MDA, which would integrate feature selection into the MDA model.

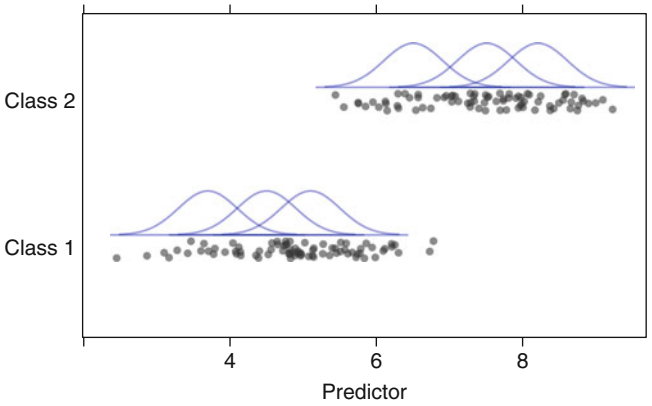

Fig. 13.1: For a single predictor, three distinct subclasses are determined within each class using mixture discriminant analysis

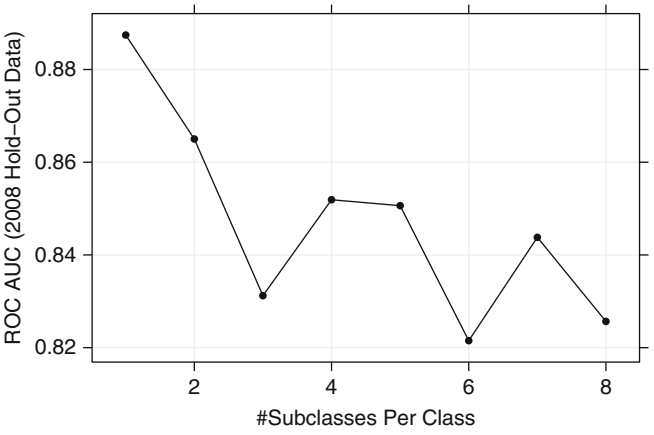

Fig. 13.2: The tuning parameter profile for the MDA model for the grants data. The optimal number of subclasses is 1, which is identical to performing LDA

For the grant data, MDA was tuned over the number of subclasses per group with possible values ranging from 1 to 8 (Fig. 13.2). The areas under the ROC curve was optimized using one subclass per group, which is the same as performing LDA. MDA may be adverse to more complex discriminant boundaries in these data due to the large number of binary predictors.

13.2 Neural Networks

As we have seen with other classification methods, such as partial least squares discriminant analysis, the C classes can be encoded into C binary columns of dummy variables and then used as the outcomes for the model. Although the previous discussion on neural networks for regression used a single response, the model can easily handle multiple outputs for both regression and classification. For neural network classification, this is the approach discussed here.

Figure 13.3 shows a diagram of the model architecture for classification. Instead of a single output (as in Fig. 7.1 for regression), the bottom layer has multiple nodes for each class. Note that, unlike neural networks for regression, an additional nonlinear transformation is used on the combination of hidden units. Each class is predicted by a linear combination of the hidden units that have been transformed to be between zero and one (usually by a sigmoidal function). However, even though the predictions are between zero and one (due the extra sigmoidal function), they aren't "probability-like" since they do not add up to one. The *softmax* transformation described in Sect. 11.1 is used here to ensure that the outputs of the neural network comply with this extra constraint:

$$f^*_{i\ell}(x) = \frac{e^{f_{i\ell}(x)}}{\sum_l e^{f_{il}(x)}},$$

where $f_{i\ell}(x)$ is the model prediction of the ℓth class and the ith sample.

What should the neural network optimize to find appropriate parameter estimates? For regression, the sum of the squared errors was the focus and, for this case, it would be altered to handle multiple outputs by accumulating the errors across samples *and* the classes:

$$\sum_{\ell=1}^{C}\sum_{i=1}^{n}\left(y_{i\ell} - f^*_{i\ell}(x)\right)^2,$$

where $y_{i\ell}$ is the 0/1 indicator for class ℓ. For classification, this can be effective method for determining parameter values. The class with the largest predicted value would be used to classify the sample.

Alternatively, parameter estimates can be found that can maximize the likelihood of the Bernoulli distribution, which corresponds to a binomial likelihood function (Eq. 12.1) with a sample size of $n = 1$:

$$\sum_{\ell=1}^{C}\sum_{i=1}^{n} y_{i\ell} \ln f^*_{i\ell}(x). \tag{13.3}$$

This function also goes by then names *entropy* or *cross-entropy*, which is used in some of the tree-based models discussed in the next chapter (Sect. 14). The likelihood has more theoretical validity than the squared error approach,

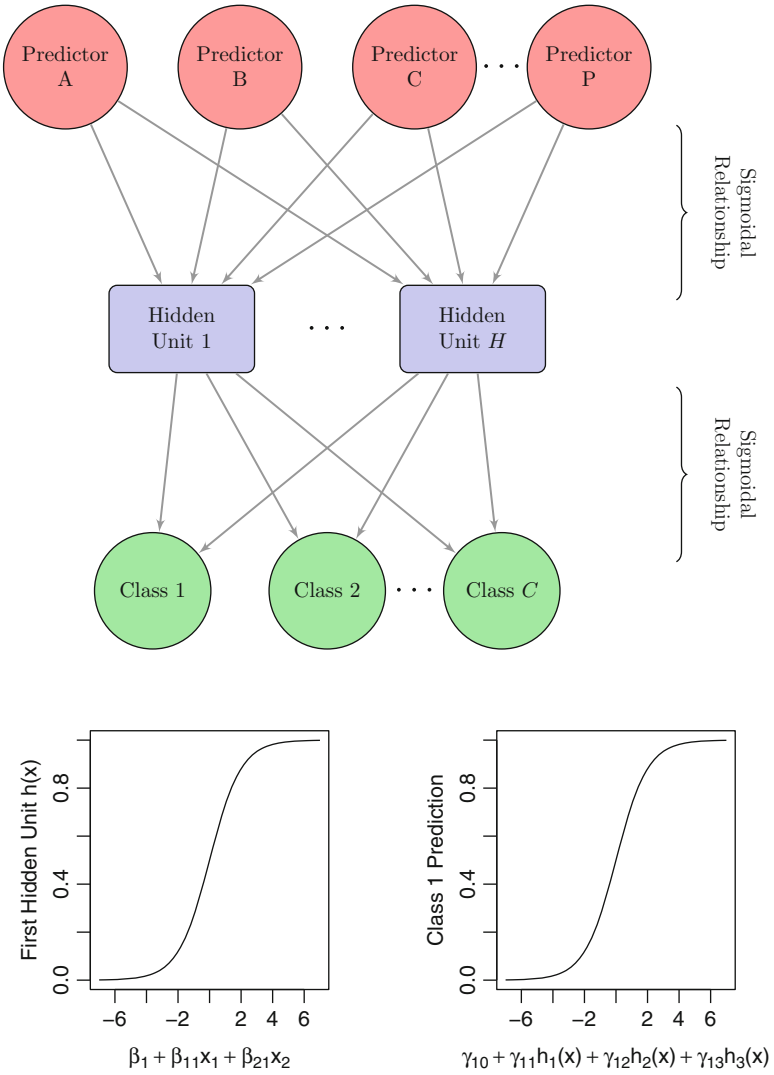

Fig. 13.3: A diagram of a neural network for classification with a single hidden layer. The hidden units are linear combinations of the predictors that have been transformed by a sigmoidal function. The output is also modeled by a sigmoidal function

although studies have shown that differences in performance tend to be negligible (Kline and Berardi 2005). However, Bishop (1995) suggests that the entropy function should more accurately estimate small probabilities than those generated by the squared-error function.

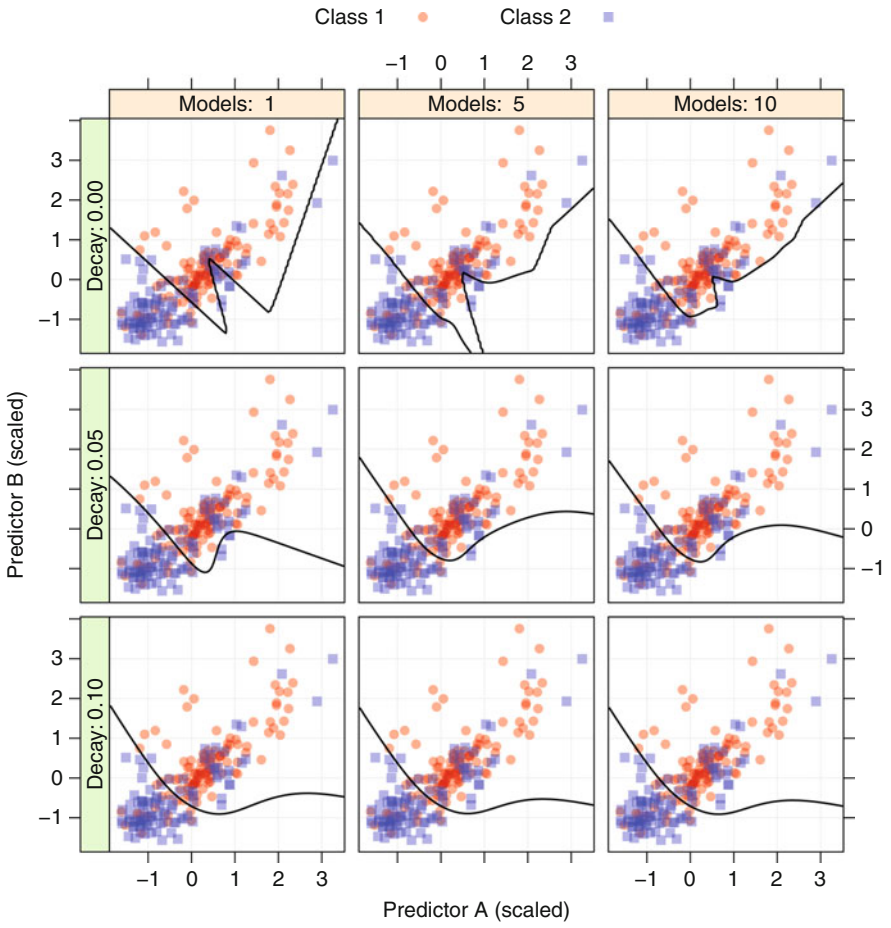

Fig. 13.4: Classification boundaries for neural networks with varying levels of smoothing and regularization. As weight decay and number of models increase, the boundaries become smoother

Like their regression counterparts, neural networks for classification have a significant potential for over-fitting. When optimizing the sums of squares error or entropy, weight decay attenuates the size of the parameter estimates. This can lead to much smoother classification boundaries. Also, as previously discussed, model averaging helps reduce over-fitting. In this case, the class probability estimates ($f^*_{i\ell}(x)$) would be averaged across networks and these average values would be used to classify samples.

Figure 13.4 shows examples of models fit with different amounts of weight decay and model averaging. Each model was initiated with the same random seed, used three hidden units, and was optimized for the sums of squared

errors. The first row of models without weight decay shows significant over-fitting, and, in these cases, model averaging has a marginal impact. The small amount of decay shown in the second row shows an improvement (as does the model averaging) but is still over-adapting to the training data when a single network is used. The highest amount of weight decay showed the best results with virtually no impact of model averaging. For these data, a single model with weight decay is probably the best choice since it is computationally least expensive.

Many other aspects of neural network classification models mirror their regression counterparts. Increasing the number of predictors or hidden units will still give rise to a large number of parameters in the model and the same numerical routines, such as back-propagation, can be used to estimate these parameters. Collinearity and non-informative predictors will have a comparable impact on model performance.

Several types of neural networks were fit to the grant data. First, single network models (i.e., no model averaging) were fit using entropy to estimate the model coefficients. The models were tuned over the number of units in the hidden layer (ranging from 1 to 10), as well as the amount of weight decay ($\lambda = 0, 0.1, 1, 2$). The best model used eight hidden units with $\lambda = 2$ and had an area under the ROC curve of 0.884. The tuning parameter profiles show a significant amount of variation, with no clear trend across the tuning parameters.

To counter this variation, the same tuning process was repeated, but 10 networks were fit to the data and their results averaged. Here, the best model had six hidden units with $\lambda = 2$ and had an area under the ROC curve of 0.884.

To increase the effectiveness of the model, various transformations of the data were evaluated. One in particular, the spatial sign transformation, had a significant positive impact on the performance of the neural networks for these data. When combined with a single network model, the area under the curve was 0.903. When model averaging was used, the area under the ROC curve was 0.911.

Figure 13.5 visualizes the tuning parameter profiles across the various models. When no data transformations are used, model averaging increases the performance of the models across all of the tuning parameters. It also has the effect of smoothing out differences between the models; the profile curves are much closer together. When the spatial sign transformation is used with the single network model, it shows an improvement over the model without the transformation. However, performance appears to be optimized when using both model averaging and the spatial sign.

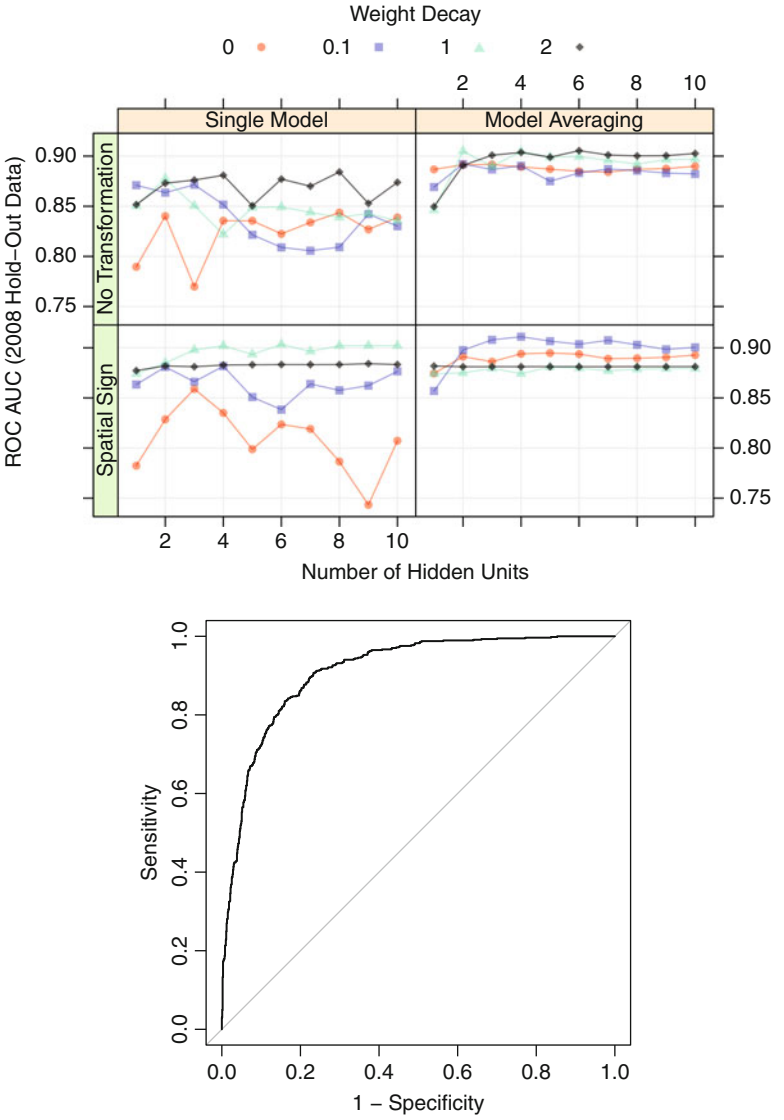

Fig. 13.5: *Top*: The models for grant success were tuned under four different conditions: with and without a transformation on the predictors and with and without model averaging. *Bottom*: The ROC curve for the 2008 holdout set when a model averaged network is used with the spatial sign transformation (area under the curve: 0.911)

1. Create a new response matrix of binary dummy variable columns for each of the C classes
2. Create a multivariate regression model using any method that generates slopes and intercepts for predictors or functions of the predictors (e.g. linear regression, MARS, etc)
3. Post-process the model parameters using the optimal scoring technique
4. Use the adjusted regression coefficients as discriminant values

Algorithm 13.1: The flexible discriminant analysis algorithm for generalizing LDA model (Hastie et al. 1994)

13.3 Flexible Discriminant Analysis

In the last chapter, the motivation for classical linear discriminant analysis was based on minimizing the total probability of misclassification. It turns out that the same model can be derived in a completely different manner. Hastie et al. (1994) describe a process where, for C classes, a set of C linear regression models can be fit to binary class indicators and show that the regression coefficients from these models can be post-processed to derive the discriminant coefficients (see Algorithm 13.1). This allows the idea of linear discriminant analysis to be extended in a number of ways. First, many of the models in Chaps. 6 and 7, such as the lasso, ridge regression, or MARS, can be extended to create discriminant variables. For example, MARS can be used to create a set of hinge functions that result in discriminant functions that are nonlinear combinations of the original predictors. As another example, the lasso can create discriminant functions with feature selection. This conceptual framework is referred to as *flexible discriminant analysis* (FDA).

We can illustrate the nonlinear nature of the flexible discriminant algorithm using MARS with the example data in Fig. 4.1 (p. 63). Recall that MARS has two tuning parameters: the number of retained terms and the degree of predictors involved in the hinge functions. If we use an additive model (i.e., a first-degree model), constrain the maximum number of retained terms to 2 and have a binary response of class membership, then discriminant function is

$$D(A, B) = 0.911 - 19.1 \times h(0.2295 - B)$$

In this equation, $h(\cdot)$ is the hinge function described in Eq. 7.1 on p. 146. If the discriminant function is greater than zero, the sample would be predicted to be the first class. In this model, the prediction equation only used the one variable, and the left-hand panel in Fig. 13.6 shows the resulting class boundaries. The class boundary is a horizontal line since predictor B is the only predictor in the split.

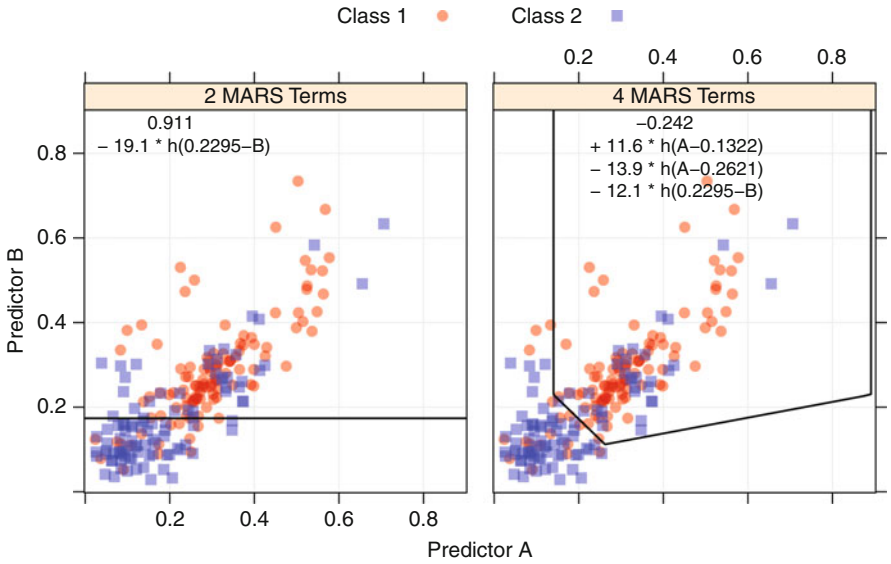

Fig. 13.6: Classification boundaries for two FDA models of different complexities

The effectiveness of FDA is not apparent when MARS is so severely restricted. If the maximum number of retained terms is relaxed to 4, then the discriminant equation is estimated to be

$$\begin{aligned} D(A,B) = &-0.242 \\ &+ 11.6 \times h(A - 0.1322) \\ &- 13.9 \times h(A - 0.2621) \\ &- 12.1 \times h(0.2295 - B). \end{aligned}$$

This FDA model uses both predictors and its class boundary is shown in the right-hand panel of Fig. 13.6. Recall that the MARS hinge function h sets one side of the breakpoint to zero. Because of this, the hinge functions isolate certain regions of the data. For example, if $A < 0.1322$ and $B > 0.2295$, none of the hinge functions affect the prediction and the negative intercept in the model indicates that all points in this region correspond to the second class. However, if $A > 0.2621$ and $B < 0.2295$, the prediction is a function of all three hinge functions. Essentially, the MARS features isolate multidimensional polytopal regions of the predictor space and predict a common class within these regions.

An FDA model was tuned and trained for the grant application model. First-degree MARS hinge functions were evaluated where the number of retained terms ranged from 2 to 25. Performance increases as the number of

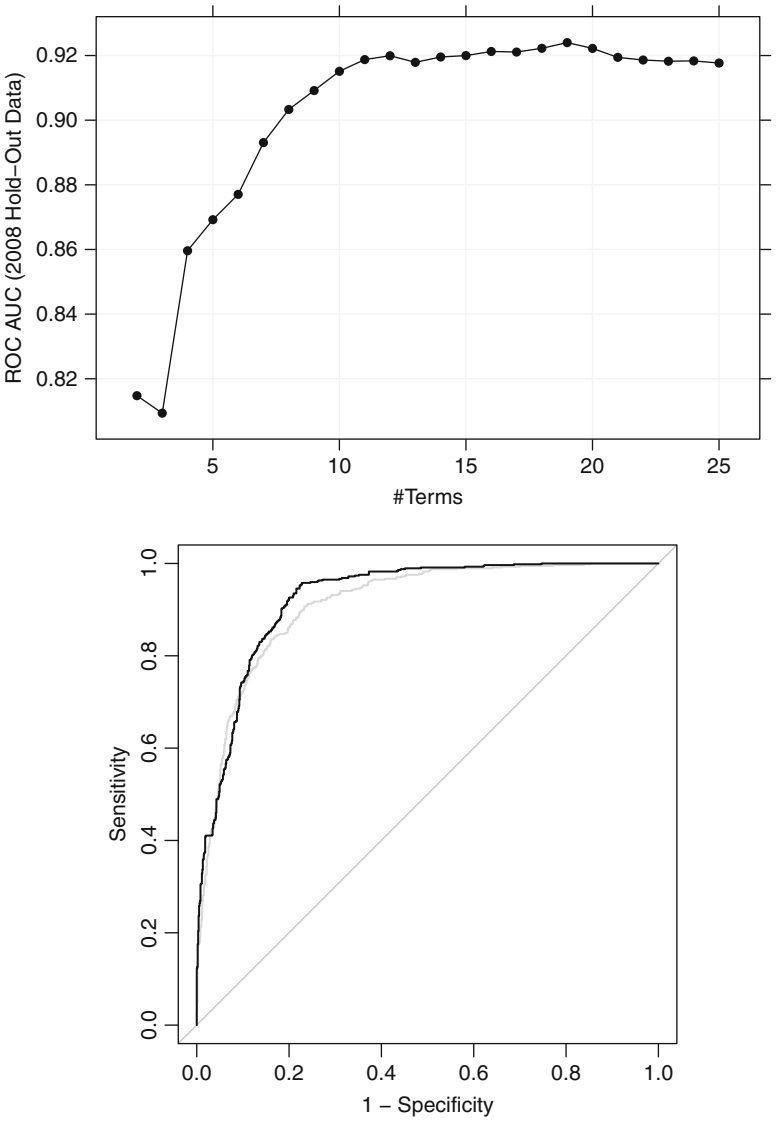

Fig. 13.7: *Top*: The parameter tuning profile for the FDA model. *Bottom*: The FDA ROC curve (area under the curve: 0.924) is shown in relation to the curve for the previous neural network model (in *grey*)

terms increases and plateaus around 15 terms (see Fig. 13.7). The numerically optimal value was 19 although there is clearly some flexibility in this parameter. For this model, the area under the ROC curve for the 2008 data was estimated to be 0.924, with a sensitivity of 82.5 % and a specificity of

86.4 %. Although the FDA model contained 19 terms, 14 unique predictors were used (of a possible 1,070). Also, nine of the model terms were simple linear functions of binary categorical predictors. The discriminant equation for the model is

$$
\begin{aligned}
D(x) = {} & 0.85 \\
& - 0.53 \times h(1 - \text{number of chief investigators}) \\
& + 0.11 \times h(\text{number of successful grants by chief investigators} - 1) \\
& - 1.1 \times h(1 - \text{number of successful grants by chief investigators}) \\
& - 0.23 \times h(\text{number of unsuccessful grants by chief investigators} - 1) \\
& + 1.4 \times h(1 - \text{number of unsuccessful grants by chief investigators}) \\
& + 0.18 \times h(\text{number of unsuccessful grants by chief investigators} - 4) \\
& - 0.035 \times h(8 - \text{number of A journal papers by all investigators}) \\
& - 0.79 \times \text{sponsor code 24D} \\
& - 1 \times \text{sponsor code 59C} \\
& - 0.98 \times \text{sponsor code 62B} \\
& - 1.4 \times \text{sponsor code 6B} \\
& + 1.2 \times \text{unknown sponsor} \\
& - 0.34 \times \text{contract value band B} \\
& - 1.5 \times \text{unknown contract value band} \\
& - 0.34 \times \text{grant category code 30B} \\
& + 0.3 \times \text{submission day of Saturday} \\
& + 0.022 \times h(54 - \text{numeric day of the year}) \\
& + 0.076 \times h(\text{numeric day of the year} - 338).
\end{aligned}
$$

From this equation, the exact effect of the predictors on the model can be elucidated. For example, as the number of chief investigators increases from zero to one, the probability of a successful grant increases. Having more than one chief investigator does not affect the model since the opposite hinge function was eliminated. Also, the probability of success increases with the number of successful grants by chief investigators and decreases with the number of unsuccessful grants by chief investigators; this is a similar result to what was found with previous models. For the day of the year, the probability of a successful grant decreases as the year proceeds and has no affect on the model until late in the year when the probability of success increases.

The discriminant function shown above can be additionally transformed to produce class probability estimates. Visually, the probability trends for the continuous predictors are shown in Fig. 13.8. Recall that since an additive model was used, the probability profile for each variable can be considered independently of the others. Here, the terms for the number of chief investigators and the number of publications in *A*-level journals only affect the prediction up to a point. This is the result of the pruning algorithm elimi-

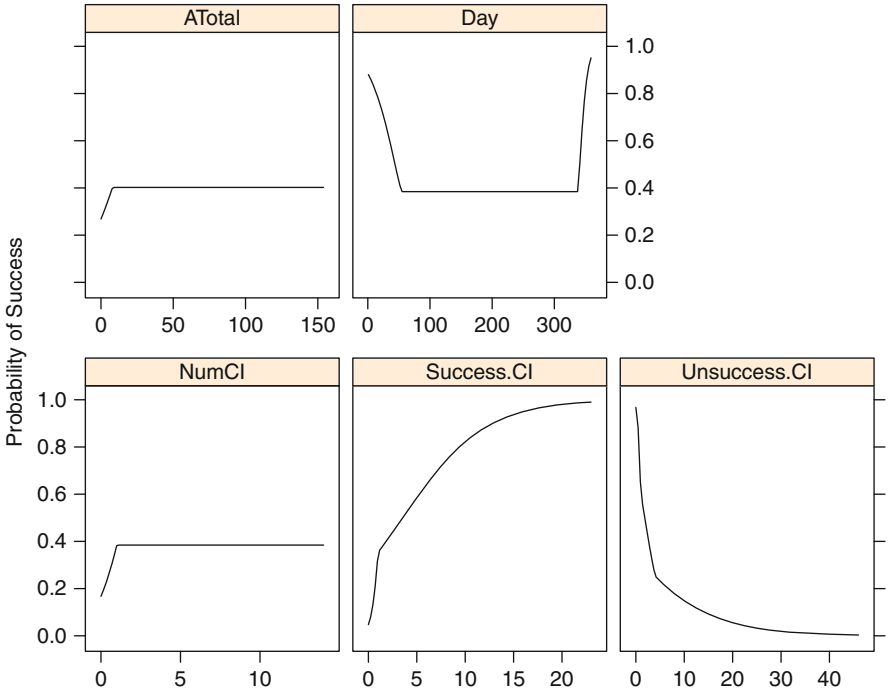

Fig. 13.8: Probability profiles for each of the continuous predictors used in the additive FDA model

nating one of each predictor's reflective pairs. The profile for the day of the year has two terms that remain from two different reflected pairs. As a result, this predictor only affects the model in the early and late periods of the year. In the last chapter, there was good evidence that this predictor had a nonlinear relationship with the outcome that was approximated by adding a quadratic function of the predictor. Here, FDA also tries to approximate the same relationship. One predictor, the number of unsuccessful grants by chief investigators, has multiple terms in the model, which is reflected in the smoother probability profile. Of the binary terms, the predictors for contract value band B, unknown contract value band, grant category code 30B, sponsor code 24D, sponsor code 59C, sponsor code 62B, and sponsor code 6B had a positive effect on the probability of success while the terms for submission day of Saturday and unknown sponsor were associated with a decrease in the success rate.

Bagging the model coerces FDA to produce smoother relationships between the predictors and the outcome. MARS models are moderately unstable predictors since they use exhaustive searches of the data and the splits

are based on specific data points in the training set.[1] Bagging the FDA model will have the effect of adding more splits for the important predictors, leading to a better approximation. However, our experience is that bagging MARS or FDA models has a marginal impact on model performance and increased number of terms diminishes the interpretation of the discriminant equation (similar to the trend shown in Fig. 8.16).

Since many of the predictors in the FDA model are on different scales, it is difficult to use the discriminant function to uncover which variables have the most impact on the outcome. The same method of measuring variable importance described in Sect. 7.2 can be employed here. The five most important predictors are, in order: unknown contract value band, the number of unsuccessful grants by chief investigators, the number of successful grants by chief investigators, unknown sponsor, and numeric day of the year.

As an alternative to using MARS within the FDA framework, Milborrow (2012) describes a two-phase approach with logistic regression when there are two classes. Here, an initial MARS model is created to predict the binary dummy response variable (i.e., the first two steps in Algorithm 13.1). After this, a logistic regression model is created with the MARS features produced by the original dummy variable model. Our preliminary experiences with this approach are that it yields results very similar to the FDA model.

13.4 Support Vector Machines

Support vector machines are a class of statistical models first developed in the mid-1960s by Vladimir Vapnik. In later years, the model has evolved considerably into one of the most flexible and effective machine learning tools available, and Vapnik (2010) provides a comprehensive treatment. The regression version of these models was previously discussed in Sect. 7.3, which was an extension of the model from its original development in the classification setting. Here we touch on similar concepts from SVM for regression and layout the case for classification.

Consider the enviable problem shown in the left panel of Fig. 13.9 where two variables are used to predict two classes of samples that are completely separable. As shown on the left, there are a multitude (in fact an infinite) number of linear boundaries that perfectly classify these data. Given this, how would we choose an appropriate class boundary? Many performance measures, such as accuracy, are insufficient since all the curves would be deemed equivalent. What would a more appropriate metric be for judging the efficacy of a model?

Vapnik defined an alternate metric called the *margin*. Loosely speaking, the margin is the distance between the classification boundary and the closest

[1] However, MARS and FDA models tend to be more stable than tree-based models since they use linear regression to estimate the model parameters.

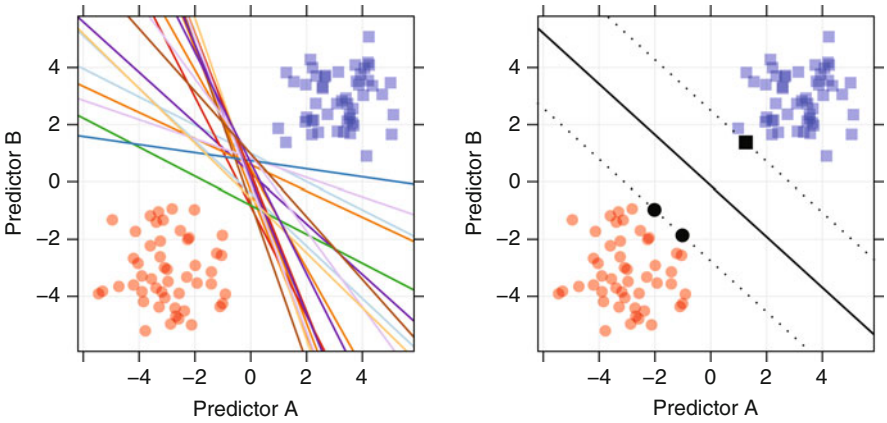

Fig. 13.9: *Left*: A data set with completely separable classes. An infinite number of linear class boundaries would produce zero errors. *Right*: The class boundary associated with the linear maximum margin classifier. The solid black points indicate the support vectors

training set point. For example, the right-hand panel of Fig. 13.9 shows one possible classification boundary as a solid line. The dashed lines on both sides of the boundary are at the maximum distance from the line to the closest training set data (equidistant from the boundary line). In this example the three data points are equally closest to the classification boundary and are highlighted with solid black symbols. The margin defined by these data points can be quantified and used to evaluate possible models. In SVM terminology, the slope and intercept of the boundary that maximize the buffer between the boundary and the data is known as the maximum margin classifier.

Let's explore a few of the mathematical constructs of SVM in the context of a simple example in order to better understand the inner workings of the method. Suppose we have a two-class problem and we code the class #1 samples with a value of 1 and the class #2 samples with -1. Also, let the vectors $\mathbf{x}_i$ contain the predictor data for a training set sample. The maximum margin classifier creates a decision value $D(\mathbf{x})$ that classifies samples such that if $D(\mathbf{x}) > 0$ we would predict a sample to be class #1, otherwise class #2. For an unknown sample $\mathbf{u}$, the decision equation can be written in a similar form as a linear discriminant function that is parameterized in terms of an intercept and slopes as

$$\begin{aligned} D(\mathbf{u}) &= \beta_0 + \boldsymbol{\beta}'\mathbf{u} \\ &= \beta_0 + \sum_{j=1}^{P} \beta_j u_j. \end{aligned}$$

Notice that this equation works from the viewpoint of the predictors. This equation can be transformed so that the maximum margin classifier can be written in terms of each data point in the sample. This changes the equation to

$$\begin{aligned} D(\mathbf{u}) &= \beta_0 + \sum_{j=1}^{P} \beta_j u_j \\ &= \beta_0 + \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i' \mathbf{u} \end{aligned} \tag{13.4}$$

with $\alpha_i \geq 0$ (similar to Eq. 7.2). It turns out that, in the completely separable case, the α parameters are exactly zero for all samples that are not on the margin. Conversely, the set of nonzero α values are the points that fall on the boundary of the margin (i.e., the solid black points in Fig. 13.9). Because of this, the predictor equation is a function of only a subset of the training set points and these are referred to as the *support vectors*. Interestingly, the prediction function is only a function of the training set samples that are closest to the boundary and are predicted with the least amount of certainty.[2] Since the prediction equation is *supported* solely by these data points, the maximum margin classifier is the usually called the *support vector machine*.

On first examination, Eq. 13.4 may appear somewhat arcane. However, it can shed some light on how support vector machines classify new samples. Consider Fig. 13.10 where a new sample, shown as a solid grey circle, is predicted by the model. The distances between each of the support vectors and the new sample are as grey dotted lines.

For these data, there are three support vectors, and therefore contain the only information necessary for classifying the new sample. The meat of Eq. 13.4 is the summation of the product of: the sign of the class, the model parameter, and the dot product between the new sample and the support vector predictor values. The following table shows the components of this sum, broken down for each of the three support vectors:

	True class	Dot product	y_i	α_i	Product
SV 1	Class 2	−2.4	−1	1.00	2.40
SV 2	Class 1	5.1	1	0.34	1.72
SV 3	Class 1	1.2	1	0.66	0.79

The dot product, $\mathbf{x}_i'\mathbf{u}$, can be written as a product of the distance of $\mathbf{x}_i$ from the origin, the distance of $\mathbf{u}$ from the origin, and the cosine of the angle between $\mathbf{x}_i$ and $\mathbf{u}$ (Dillon and Goldstein 1984).

[2] Recall a similar situation with support vector regression models where the prediction function was determined by the samples with the largest residuals.

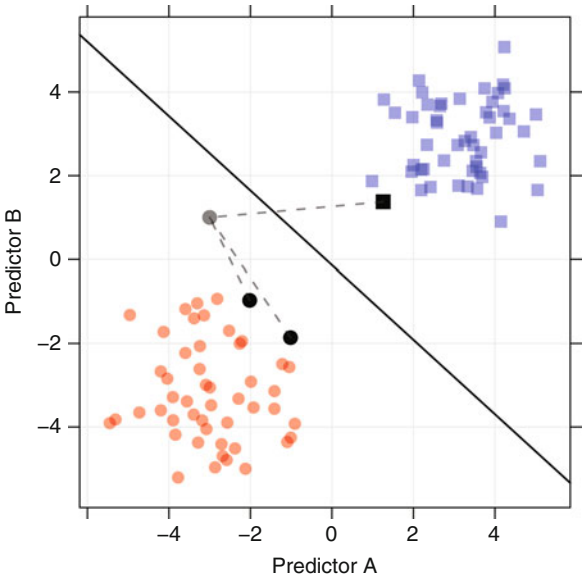

Fig. 13.10: Prediction of a new sample using a support vector machine. The final value of the decision equation is $D(u) = 0.583$. The grey lines indicate the distance of the new sample to the support vectors

Based on the parameter estimates α_i, the first support vector has the largest single effect on the prediction equation (all other things being equal) and it has a negative slope. For our new sample, the dot product is negative, so the total contribution of this point is positive and pushes the prediction towards the first class (i.e., a positive value of the decision function $D(u)$). The remaining two support vectors have positive dot products and an overall product that increases the decision function value for this sample. For this model, the intercept is -4.372; $D(u)$ for the new sample is therefore 0.583. Since this value is greater than zero, the new sample has the highest association with the first class.

What happens when the classes are not completely separable? Cortes and Vapnik (1995) develop extensions to the early maximum margin classifier to accommodate this situation. Their formulation puts a cost on the sum of the training set points that are on the boundary or on the wrong side of the boundary. When determining the estimates of the α values, the margin is penalized when data points are on the wrong side of the class boundary or inside the margin. The cost value would be a tuning parameter for the model and is the primary mechanism to control the complexity of the boundary. For example, as the cost of errors increases, the classification boundary will shift and contort itself so that it correctly classifies as many of the training

set points as possible. Figure 4.2 in Chap. 4 demonstrated this; the panel on the right-hand side of this figure used an inappropriately high cost value, resulting in severe over-fitting.

Echoing the comments in Sect. 7.3, most of the regularization models discussed in this book add penalties to the coefficients, to prevent over-fitting. Large penalties, similar to costs, impose limits on the model complexity. For support vector machines, cost values are used to penalize number of *errors*; as a consequence, larger cost values induce higher model complexity rather than restrain it.

Thus far, we have considered linear classification boundaries for these models. In Eq. 13.4, note the dot product $\mathbf{x}_i'\mathbf{u}$. Since the predictors enter into this equation in a linear manner, the decision boundary is correspondingly linear. Boser et al. (1992) extended the linear nature of the model to nonlinear classification boundaries by substituting the kernel function instead of the simple linear cross product:

$$\begin{aligned} D(\mathbf{u}) &= \beta_0 + \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i'\mathbf{u} \\ &= \beta_0 + \sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}_i, \mathbf{u}), \end{aligned}$$

where $K(\cdot, \cdot)$ is a *kernel function* of the two vectors. For the linear case, the kernel function is the same inner product $\mathbf{x}_i'\mathbf{u}$. However, just as in regression SVMs, other nonlinear transformations can be applied, including:

$$\begin{aligned} \text{polynomial} &= (scale\ (\mathbf{x}'\mathbf{u}) + 1)^{degree} \\ \text{radial basis function} &= \exp(-\sigma \|\mathbf{x} - \mathbf{u}\|^2) \\ \text{hyperbolic tangent} &= \tanh\left(scale\ (\mathbf{x}'\mathbf{u}) + 1\right). \end{aligned}$$

Note that, due to the dot product, the predictor data should be centered and scaled prior to fitting so that attributes whose values are large in magnitude do not dominate the calculations.

The *kernel trick* allows the SVM model produce extremely flexible decision boundaries. The choice of the kernel function parameters and the cost value control the complexity and should be tuned appropriately so that the model does not over-fit the training data. Figure 13.11 shows examples of the classification boundaries produced by several models using different combinations of the cost and tuning parameter values. When the cost value is low, the models clearly underfit the data. Conversely, when the cost is relatively high (say a value of 16), the model can over-fit the data, especially if the kernel parameter has a large value. Using resampling to find appropriate estimates of these parameters tends to find a reasonable balance between under- and over-fitting. Section 4.6 used the radial basis function support vector machine as an example for model tuning.

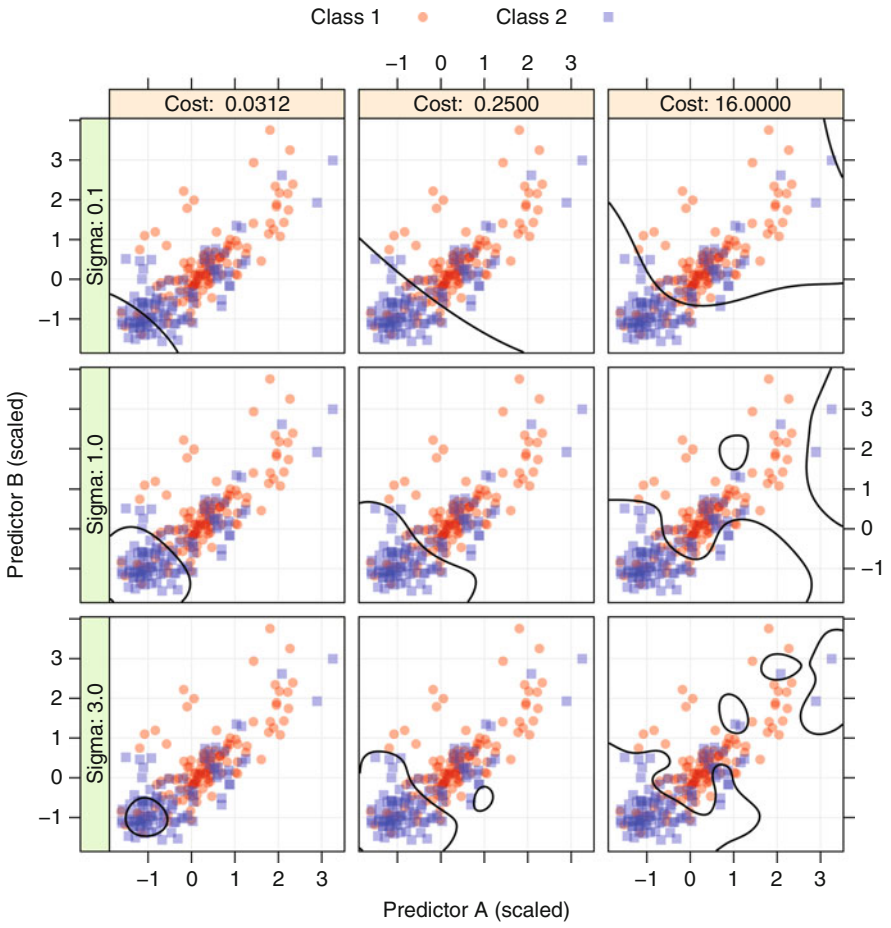

Fig. 13.11: Classification boundaries for nine radial basis function support vector machine models varied over the cost parameter and the kernel parameter (σ)

Support vector machines fall into a more general category of *kernel methods* and this has been an extremely active area of research for some time. Here, we have discussed extensions to the original model to allow for misclassified samples and nonlinear class boundaries. Still more extensions have been developed for support vector machines, such as handling more than two classes (Hsu and Lin 2002; Duan and Keerthi 2005). Also, the original motivation of the model is to create a hard decision boundary for the purpose of classifying samples, as opposed to estimating class probabilities. However, Platt (2000) describes methods of post-processing the output of the SVM model to estimate class probabilities. Alternate

versions of the support vector machine model also exist, such as least squares support vector machines (Suykens and Vandewalle 1999), relevance vector machines (Tipping 2001), and import vector machines (Zhu and Hastie 2005).

Specialized kernels have also been developed. For example, the QSAR application discussed in Sect. 6.1 and used throughout the regression chapters used chemical descriptors as predictors. Figure 6.1 shows the chemical formula of aspirin. Rather than deriving descriptors from a molecular formula, the formula can be converted to a graph (or network) representation. A specialized class of kernel functions, called *graph kernels*, can directly relate the content of the chemical formula to the model without deriving descriptor variables (Mahé et al. 2005; Mahé and Vert 2009). Similarly, there are different kernels that can be employed in text mining problems. The "bag-of-words" approach summarizes a body of text by calculating frequencies of specific words. These counts are treated as predictor variables in classification models. There are a few issues with this approach. First, the additional computational burden of deriving the predictor variables can be taxing. Secondly, this term-based approach does not consider the ordering of the text. For example, the text "Miranda ate the bear" and "the bear ate Miranda" would score the same in the bag-of-words model but have very different meanings. *String kernels* (Lodhi et al. 2002; Cancedda et al. 2003) can use the entire text of a document directly and has more potential to find important relationships than the bag-of-words approach.

For the grant data, there are several approaches to using SVMs. We evaluated the radial basis function kernel as well as the polynomial kernel (configured to be either linear or quadratic). Also, both the full and reduced predictor sets were evaluated. As will be shown in Chap. 19, support vector machines can be negatively affected by including non-informative predictors in the model.

For the radial basis function kernel, the analytical approach for determining the radial basis function parameter was assessed. For the full set of predictors, the estimate was $\sigma = 0.000559$ and for the reduced set, the value was calculated to be $\sigma = 0.00226$. However, these models did not show good performance, so this parameter was varied over values that were smaller than analytical estimates. Figure 13.12 shows the results of these models. The smaller predictor set yields better results than the more comprehensive set, with an optimal area under the ROC curve of 0.895, a sensitivity of 84 %, and a specificity of 80.4 %. Also, for the reduced set, smaller values of σ produced better results, although values below 0.001167 did not improve the model fit.

For the polynomial models, a fair amount of trial and error was used to determine appropriate values for this kernel's scaling factor. Inappropriate values would result in numerical difficulties for the models and feasible values of this parameter depended on the polynomial degree and the cost parameter. Figure 13.13 shows the results for the holdout set. Models built with the reduced set of predictors did uniformly better than those utilizing the full set. Also, the optimal performance for linear and quadratic models was about

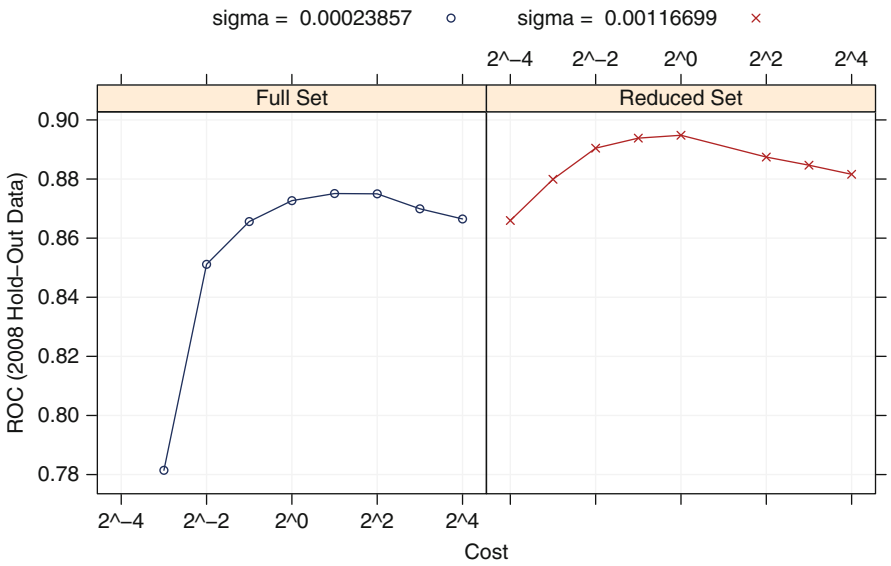

Fig. 13.12: Tuning parameter profile of the radial basis function SVM model for the grant data

the same. This suggests that the models are mostly picking up on linear relationships in the data. Given that many of the predictors are binary, this makes sense. Of these models, the best area under the ROC curve was 0.898.

Overall, the support vector machine models did not have competitive performance in comparison to models created thus far. Many of the linear models shown in Chap. 12 had similar (or better) performance; the FDA model in this chapter, so far, is more effective. However, in our experience, SVM models tend to be very competitive for most problems.

13.5 *K*-Nearest Neighbors

We first met the K-nearest neighbors (KNNs) model for classification in Sect. 4.2 when discussing model tuning and the problem of over-fitting. We have also learned extensively about KNN in the context of regression in Sect. 7.4. While many of the ideas from KNN for regression directly apply here, we will highlight the unique aspects of how this method applies to classification.

The classification methods discussed thus far search for linear or nonlinear boundaries that optimally separate the data. These boundaries are then used to predict the classification of new samples. KNN takes a different approach

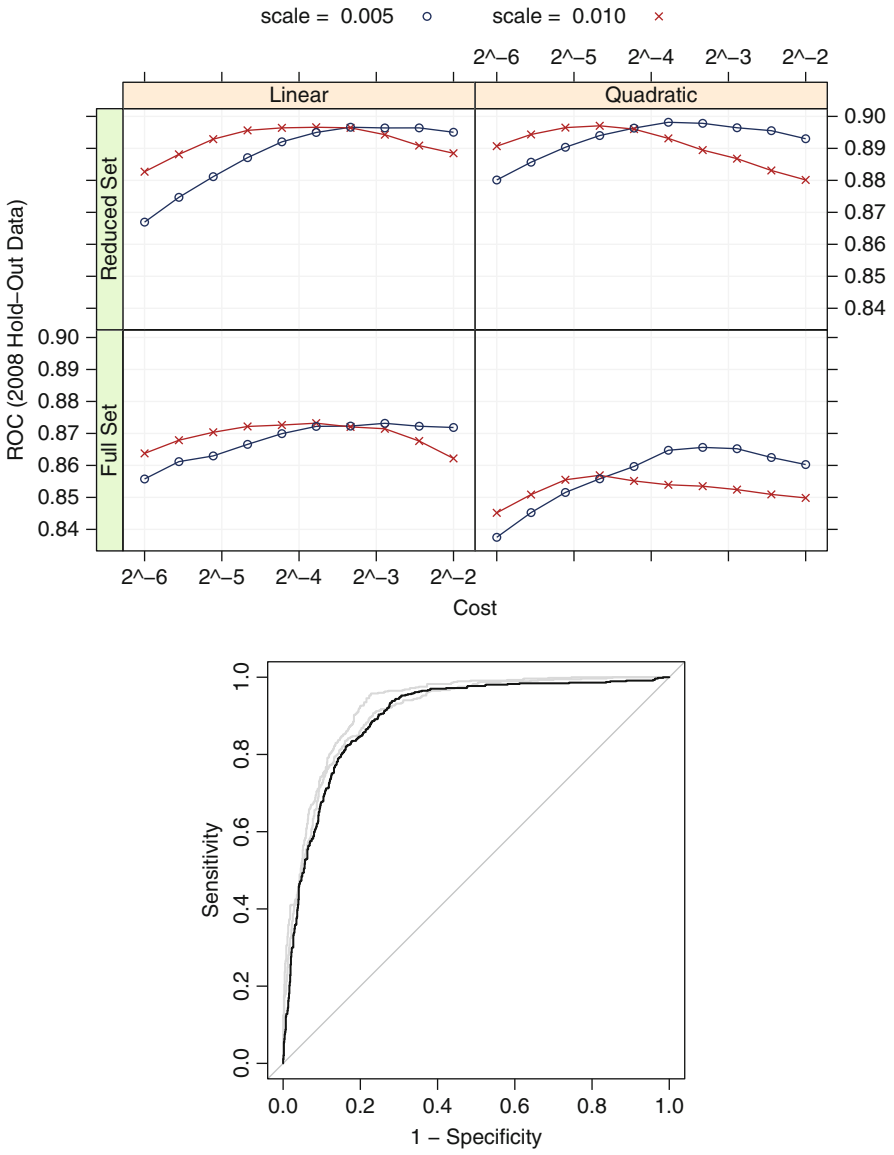

Fig. 13.13: *Top*: Performance profiles for the quadratic SVM model. *Bottom*: The ROC curve for the optimal model (area under the curve: 0.895)

by using a sample's geographic neighborhood to predict the sample's classification. Similar to the regression context, KNN for classification predicts a new sample using the K-closest samples from the training set. "Closeness" is determined by a distance metric, like Euclidean and Minkowski (Sect. 7.4),

and choice of metric depends on predictor characteristics. For any distance metric, it is important to recall that the original measurement scales of the predictors affect the resulting distance calculations. This implies that if predictors are on widely different scales, the distance value between samples will be biased towards predictors with larger scales. To allow each predictor to contribute equally to the distance calculation, we recommend centering and scaling all predictors prior to performing KNN.

As in the regression context, to determine the classification of a new sample, the K-closest training set samples are determined via the distance metric. Class probability estimates for the new sample are calculated as the proportion of training set neighbors in each class. The new sample's predicted class is the class with the highest probability estimate; if two or more classes are tied for the highest estimate, then the tie is broken at random or by looking ahead to the $K + 1$ closest neighbor.

Any method with tuning parameters can be prone to over-fitting, and KNN is especially susceptible to this problem as was shown in Fig. 4.2. Too few neighbors leads to highly localized fitting (i.e., over-fitting), while too many neighbors leads to boundaries that may not locate necessary separating structure in the data. Therefore, we must take the usual cross-validation or resampling approach for determining the optimal value of K.

For the grant data the neighborhood range evaluated for tuning was between 1 and 451. Figure 13.14 illustrates the tuning profile for area under the ROC curve for the 2008 holdout data. There is a distinct jump in predictive performance from 1 to 5 neighbors and a continued steady increase in performance through the range of tuning. The initial jump in predictive performance indicates that local geographic information is highly informative for categorizing samples. The steady incremental increase in predictive performance furthermore implies that neighborhoods of informative information for categorizing samples are quite large. This pattern is somewhat unusual for KNN in that as the number of neighbors increases we begin to underfit and a corresponding decrease in predictive performance occurs like was illustrated by Fig. 7.10. In most data sets, we are unlikely to use this many neighbors in the prediction. This example helps to identify a numerical instability problem with KNN: as the number of neighbor increases, the probability of ties also increases. For this example, a neighborhood size greater than 451 leads to too many ties. The optimal area under the ROC curve was 0.81, which occurred at $K = 451$. The bottom plot in Fig. 13.14 compares the KNN ROC profile with those of SVM and FDA. For these data, the predictive ability of KNN is inferior to the other tuned nonlinear models. While geographic information is predictive, it is not as useful as models that seek to find global optimal separating boundaries.

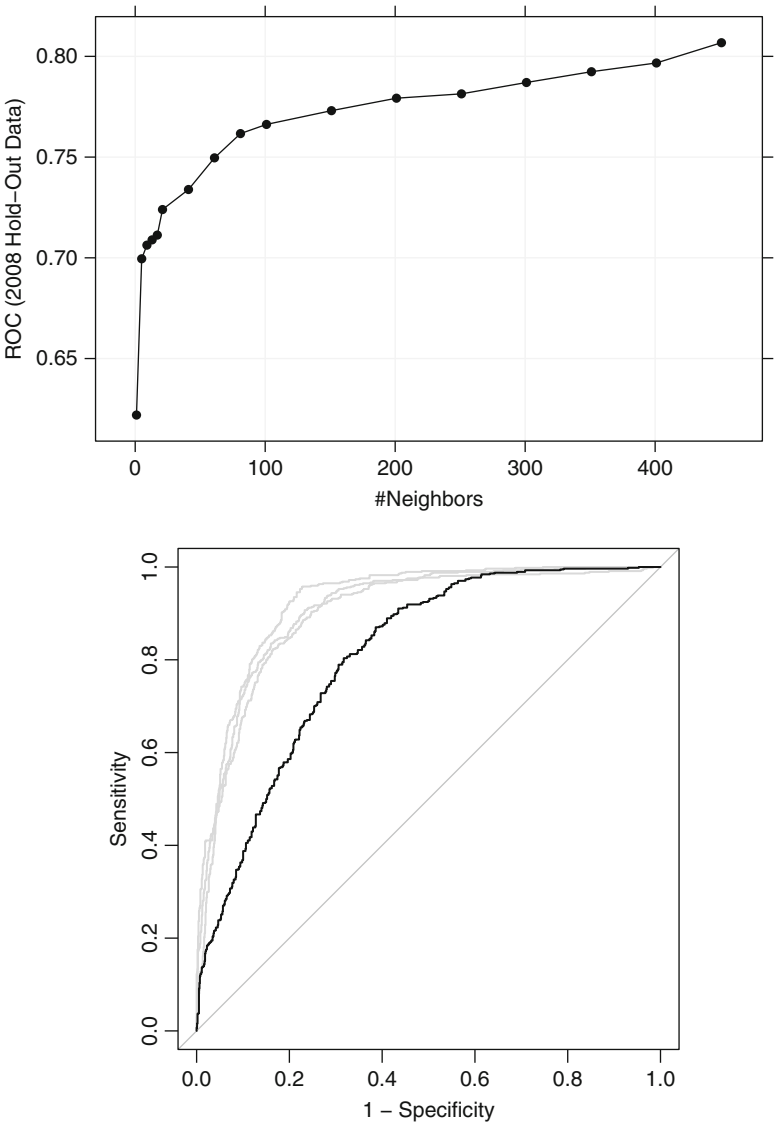

Fig. 13.14: *Top*: The parameter tuning profile for the KNN model. *Bottom*: The ROC curve for the test set data. The area under the curve was 0.81

13.6 Naïve Bayes

Bayes' Rule was previously discussed in the context of linear discriminant analysis in a previous chapter. This section expands on that discussion and focuses on a specific classification model that, like the previous LDA, QDA,

and RDA models, is defined in terms of how the multivariate probability densities are created.

Bayes' Rule answers the question "based on the predictors that we have observed, what is the probability that the outcome is class C_ℓ?" More mathematically, let Y be the class variable and X represent the collection of predictor variables. We are trying to estimate $Pr[Y = C_\ell|X]$, which is "given X, what is the probability that the outcome is the ℓth class?" Bayes' Rule provides the machinery to answer this:

$$Pr[Y = C_\ell|X] = \frac{Pr[Y]Pr[X|Y = C_\ell]}{Pr[X]} \tag{13.5}$$

$Pr[Y = C_\ell|X]$ is typically referred to as the *posterior probability* of the class. The components are:

- $Pr[Y]$ is the *prior* probability of the outcome. Essentially, based on what we know about the problem, what would we expect the probability of the class to be? For example, when predicting customer churn, companies typically have a good idea of the overall turnover rate of customers. For problems related to diseases, this prior would be the disease prevalence rate in the population (see Sect. 11.2 on p. 254 for a discussion).
- $Pr[X]$ is the probability of the predictor values. For example, if a new sample is being predicted, how likely is this pattern in comparison to the training data? Formally, this probability is calculated using a multivariate probability distribution. In practice, significant assumptions are usually made to reduce the complexity of this calculation.
- $Pr[X|Y = C_\ell]$ is the *conditional probability*. For the data associated with class C_ℓ, what is the probability of observing the predictor values? Similar to $Pr[X]$, this can be a complex calculation unless strict assumptions are made.

The naïve Bayes model simplifies the probabilities of the predictor values by assuming that all of the predictors are independent of the others. This is an extremely strong assumption. For most of the case studies and illustrative examples in this text, it would be difficult to claim that this assumption were realistic. However, the assumption of independence yields a significant reduction in the complexity of the calculations.

For example, to calculate the conditional probability $Pr[X|Y = C_\ell]$, we would use a product of the probability densities for each individual predictor:

$$Pr[X|Y = C_\ell] = \prod_{j=1}^{P} Pr[X_j|Y = C_\ell]$$

The unconditional probability $Pr[X]$ results in a similar formula when assuming independence. To estimate the individual probabilities, an assumption of

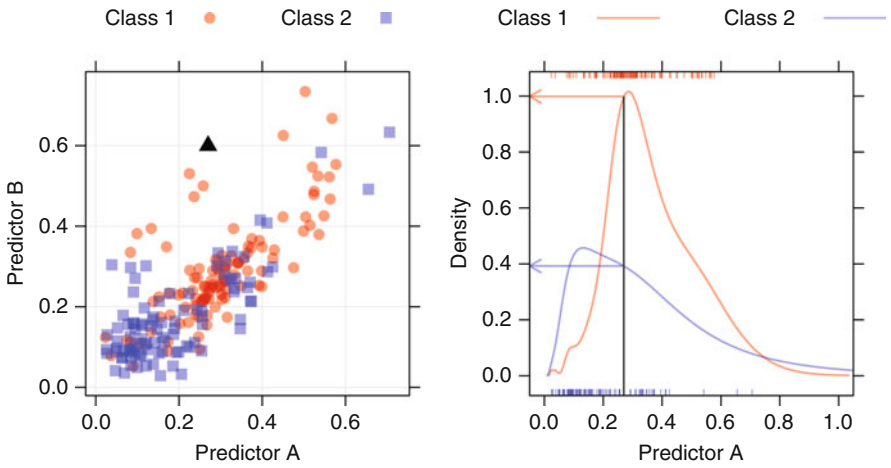

Fig. 13.15: *Left*: A plot of two class illustrative data where a new sample (the *solid triangle*) is being predicted. *Right*: Conditional density plots of predictor A created using a nonparametric density estimate. The value of predictor A for the new sample is shown by the *vertical black line*

normality might be made for continuous predictors (using the sample mean and variance from the training set). Other methods, such as nonparametric kernel density estimators (Hardle et al. 2004), can more flexibly estimate the probability densities. For categorical predictors, the probability distribution can be determined with the observed frequencies in the training set data.

For example, Fig. 13.15 shows the familiar two-class illustrative example. In the left panel, the training data are shown. Clearly, the two predictors are unlikely to be independent (their correlation is 0.78). Suppose a new sample (shown as a solid black triangle) requires prediction. To compute the overall conditional probability $Pr[X|Y = C_\ell]$, each predictor is considered separately. For predictor A, the two conditional densities are shown in the right panel of Fig. 13.15 with a vertical black line indicating the value of the new sample for this predictor. For the training set data, using this predictor alone, the first class appears to be much more likely.

To produce the class probability $Pr[X|Y = C_\ell]$ for the first class, two conditional probability values are determined for predictors A and B then multiplied together to calculate the overall conditional probability for the class.

For $Pr[X]$ a similar procedure would occur except the probabilities for predictors A and B would be determined from the entire training set (i.e., both classes). For the example in Fig. 13.15, the correlation between the predictors is fairly strong, which indicates that the new sample is highly unlikely.

Table 13.1: The frequencies and conditional probabilities $Pr[X|Y = C_\ell]$ for the day of the week

Day	Count		Percent of total	
	Successful	Unsuccessful	Successful	Unsuccessful
Mon	749	803	9.15	9.80
Tues	597	658	7.29	8.03
Wed	588	752	7.18	9.18
Thurs	416	358	5.08	4.37
Fri	606	952	7.40	11.62
Sat	619	861	7.56	10.51
Sun	228	3	2.78	0.04

However, using the assumption of independence, this probability is likely to be overestimated.

The prior probability allows the modeler to *tilt* the final probability towards one or more classes. For example, when modeling a rare event, it is common to selectively sample the data so that the class distribution in the training set is more balanced. However, the modeler may wish to specify that the event is indeed rare by assigning it a low prior probability. If no prior is explicitly given, the convention is to use the observed proportions from the training set to estimate the prior.

Given such a severe and unrealistic assumption, why would one consider this model? First, the naïve Bayes model can be computed quickly, even for large training sets. For example, when the predictors are all categorical, simple lookup tables with the training set frequency distributions are all that are required. Secondly, despite such a strong assumption, the model performs competitively in many cases.

Bayes' Rule is essentially a probability statement. Class probabilities are created and the predicted class is the one associated with the largest class probability. The meat of the model is the determination of the conditional and unconditional probabilities associated with the predictors. For continuous predictors, one might choose simple distributional assumptions, such as normality. The nonparametric densities (such as those shown in Fig. 13.16) can produce more flexible probability estimates. For the grant application data, the predictor for the numeric day of the year has several time frames where an inordinate number of grants were submitted. In this figure, the black curve for the normal distribution is extremely broad and does not capture the nuances of the data. The red curve is the nonparametric estimate and appears produce the trends in the data with higher fidelity.

For categorical predictors, the frequency distribution of the predictor in the training set is used to estimate $Pr[X]$ and $Pr[X|Y = C_\ell]$. Table 13.1

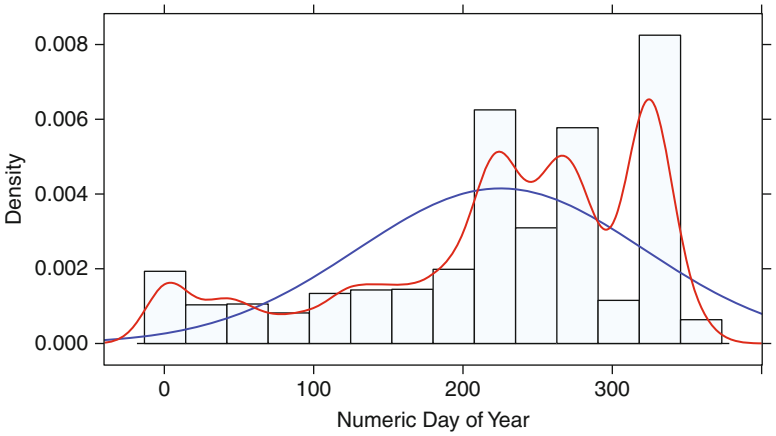

Fig. 13.16: Two approaches to estimating the density function $Pr[X]$ for the day of the year. The *blue line* is based on a normal distribution while the *red line* is generated using a nonparametric density estimator

shows the observed frequencies for the day of the week in which the grant was submitted. The columns showing the percent of total are the estimates of $Pr[X|Y = C_\ell]$ for each class. When a new sample is predicted, a simple lookup on this table is used to estimate the probabilities.

An obvious issue, especially for small samples sizes, occurs when one or more frequencies are zero. If a predictor has no training set samples for a specific class, the conditional probability would be zero and, since the probabilities are multiplied together, one predictor would coerce the posterior probability to be zero. One method for avoiding this issue is to use a *Laplace correction* or *Laplace smoothing* (Niblett 1987; Zadrozny and Elkan 2001; Provost and Domingos 2003) where the same correction factor, usually between one and two, is added to the numerator. For the denominator, the frequencies are increase by the correction factor times the number of values of the predictor. For example, there are very low frequencies for grants submitted on Sunday. To correct for the extreme probabilities, a correction factor of one would changes the observed frequencies to 229 and 4, but the denominator would be increased by seven. Given the large sample size for the training set, this correction only has a small impact (the estimated success rate on Sunday is increased from 2.78 % to 2.79 %). However, all of the three unsuccessful grants in the table were submitted after 2008. Training on pre-2008 data would generate zero probabilities. In this case, a correction of value of one would change the probability for grants to 0.02 % while a correction factor of two would increase the value to 0.03 %. For smaller training set sizes, the correction can have a substantial positive effect on the missing cells in the table.

For the grant data, many of the predictors were counts. Although these are numbers, they are discrete values and could be treated as categories. In many cases, the observed frequency distribution is compact. For example, in the training set, the number of chief investigators in department 2,678 takes on the four values between 0 and 3 and has a very right-skewed distribution. Treating such a granular predictor as if it was generated by a symmetric normal distribution may produce poor probability estimates. For this analysis, the reduced set of predictors was evaluated such that all predictors with less than 15 possible values were treated as discrete and their probabilities were calculated using their frequency distribution (such as the day of the week shown in Table 13.1. There were 14 predictors with more than 15 unique values, including the number of successful grants by chief investigators, the number of A* journal papers by chief investigators, and numeric day of the year.

These predictors were modeled using either a normal distribution or a nonparametric density (the density type was treated as a tuning parameter), and a Laplace correction of 2 was used. When using a normal distribution for the continuous predictors, the area under the curve was estimated to be 0.78, a sensitivity of 58.8 %, and a specificity of 79.6 %. Using nonparametric estimation of the probability densities, the area under the ROC curve improves to 0.81, which corresponding increases in sensitivity (64.4 %) and specificity (82.4 %). Unfortunately, the performance for this model is on par with KNNs, which is substantially below the results of the other models in this chapter.

Section 11.1 showed that Bayes' Rule can be used to calibrate class probability estimates. To do this, the true classes are used as Y, but the class probability values for the training set are used as the "predictor" and $Pr[X|Y = C_\ell]$ is determined from the model predictions on the training set. When new samples are predicted, the class probabilities that are generated by the model are post-processed using Bayes' Rule to improve the calibration. Ironically, class probabilities created by apply Bayes' Rule in the normal fashion tend not to be well-calibrated themselves. As the number of predictors increases (relative to the sample size), the posterior probabilities will become more extreme (similar to the observation related to linear discriminant analysis shown in Fig. 12.11). Recall that QDA is based on Bayes' Rule (using multivariate normality for the predictors) and the QDA results shown in Fig. 11.1 showed poor calibration with two predictor variables (but was improved by recalibrating using another application of Bayes' Rule).

13.7 Computing

The following R packages are discussed in this chapter: caret, earth, kernlab, klaR, MASS, mda, nnet, and rrcov. This section also uses the same R objects created in the last chapter that contain the data (such as the data frame `training`).

Nonlinear Discriminant Analysis

A number of packages are available to perform the varieties of nonlinear discriminant analysis described earlier in this chapter. QDA is implemented in the `qda` function in the MASS as well as an outlier-resistant version in the `QdaCov` function in the rrcov package. RDA is available in the `rda` function in the klaR package, and MDA can be found in the mda package. The syntax for these models is very similar and we will demonstrate their usage by fitting an MDA model to the grant data.

The `mda` function has a model formula interface. The tuning parameter is the number of subclasses per class, which do not have to be the same for each class. For example, to fit an MDA model to the grant data with three subpopulations per class:

```
> library(mda)
> mdaModel <- mda(Class ~ .,
+                 ## Reduce the data to the relevant predictors and the
+                 ## class variable to use the formula shortcut above
+                 data = training[pre2008, c("Class", reducedSet)],
+                 subclasses = 3)
> mdaModel
Call:
mda(formula = Class ~ ., data = training[pre2008, c("Class",
    reducedSet)], subclasses = 3)

Dimension: 5

Percent Between-Group Variance Explained:
    v1     v2     v3     v4     v5
 72.50  92.57  96.10  98.66 100.00

Degrees of Freedom (per dimension): 253

Training Misclassification Error: 0.18709 ( N = 6633 )

Deviance: 6429.499
> predict(mdaModel,
+         newdata = head(training[-pre2008, reducedSet]))
[1] successful successful successful successful successful successful
Levels: successful unsuccessful
```

Each of these nonlinear discriminant models can be built and optimal tuning parameters can be found using the caret package. The `trControl` option for the grants data is set as described in Sect. 12.7 and will be used here:

```
> set.seed(476)
> mdaFit <- train(training[,reducedSet], training$Class,
+                 method = "mda",
+                 metric = "ROC",
+                 tuneGrid = expand.grid(.subclasses = 1:8),
+                 trControl = ctrl)
```

Similar syntax can be used for RDA (using `method = "rda"`) and QDA (method values of either `"rda"` or `"QdaCov"` for the outlier-resistant version in the rrcov package).

A penalized version of MDA is also available in the sparseLDA package with the `smda` function. See Clemmensen et al. (2011) for more details.

Neural Networks

There are many R packages for neural networks, including nnet, RSNNS, qrnn, and neuralnet. Two resources for using neural networks in R are Venables and Ripley (2002) and Sect. 7 of Bergmeir and Benitez (2012).

The analyses here focus on the nnet package. The syntax is extremely similar to that of the regression models with a few exceptions. The `linout` argument should be set to `FALSE` since most classification models use a sigmoidal transformation to relate the hidden units to the outputs. The sums of squared errors or entropy estimates model parameters and the logical arguments `softmax` and `entropy` toggle between the two.

The package has both a formula interface and an interface for passing matrices or data frames for the predictors and the outcome. For the latter, the outcome cannot be a factor variable and must be converted to a set of C binary indicators. The package contains a function, `class.ind`, that is useful in making this conversion:

```
> head(class.ind(training$Class))
     successful unsuccessful
[1,]          1            0
[2,]          1            0
[3,]          1            0
[4,]          1            0
[5,]          0            1
[6,]          1            0
```

Using the formula interface to fit a simple model:

```
> set.seed(800)
> nnetMod <- nnet(Class ~ NumCI + CI.1960,
+                 data = training[pre2008,],
+                 size = 3, decay = .1)
  # weights:  13
  initial  value 4802.892391
  iter  10 value 4595.629073
  iter  20 value 4584.893054
  iter  30 value 4582.614616
  iter  40 value 4581.010289
  iter  50 value 4580.866146
  iter  60 value 4580.781092
  iter  70 value 4580.756342
```

```
  final  value 4580.756133
  converged
> nnetMod
  a 2-3-1 network with 13 weights
  inputs: NumCI CI.1960
  output(s): Class
  options were - entropy fitting  decay=0.1
> predict(nnetMod, newdata = head(testing))
           [,1]
6641 0.5178744
6647 0.5178744
6649 0.5138892
6650 0.5837029
6655 0.4899851
6659 0.5701479
> predict(nnetMod, newdata = head(testing), type = "class")
  [1] "unsuccessful" "unsuccessful" "unsuccessful" "unsuccessful"
  [5] "successful"   "unsuccessful"
```

When three or more classes are modeled, the basic call to `predict` produces columns for each class.

As before, `train` provides a wrapper to this function to tune the model over the amount of weight decay and the number of hidden units. The same model code is used (`method = "nnet"`) and either model interface is available, although `train` does allow factor vectors for the classes (using `class.ind` internally do encode the dummy variables). Also, as in regression, model averaging can be used via the stand-alone `avNNet` function or using `train` (with `method = "avNNet"`).

The final model for the grant data has the following syntax:

```
> nnetGrid <- expand.grid(.size = 1:10,
+                         .decay = c(0, .1, 1, 2))
> maxSize <- max(nnetGrid$.size)
> numWts <- 1*(maxSize * (length(reducedSet) + 1) + maxSize + 1)
> set.seed(476)
> nnetFit <- train(x = training[,reducedSet],
+                  y = training$Class,
+                  method = "nnet",
+                  metric = "ROC",
+                  preProc = c("center", "scale", "spatialSign"),
+                  tuneGrid = nnetGrid,
+                  trace = FALSE,
+                  maxit = 2000,
+                  MaxNWts = numWts,
+                  ## ctrl was defined in the previous chapter
+                  trControl = ctrl)
```

Flexible Discriminant Analysis

The mda package contains a function (`fda`) for building this model. The model accepts the formula interface and has an option (`method`) that specifies the exact method for estimating the regression parameters. To use FDA with MARS, there are two approaches. `method = mars` uses the MARS implementation in the mda package. However, the earth package, previously described in Sect. 7.5, fits the MARS model with a wider range of options. Here, load the earth package and then specify `method = earth`. For example, a simple FDA model for the grant application data could be created as

```
> library(mda)
> library(earth)
> fdaModel <- fda(Class ~ Day + NumCI, data = training[pre2008,],
+                 method = earth)
```

Arguments to the `earth` function, such as `nprune`, can be specified when calling `fda` and are passed through to `earth`. The MARS model is contained in a sub-object called `fit`:

```
> summary(fdaModel$fit)
  Call: earth(x=x, y=Theta, weights=weights)

              coefficients
  (Intercept)   1.41053449
  h(Day-91)    -0.01348332
  h(Day-202)    0.03259400
  h(Day-228)   -0.02660477
  h(228-Day)   -0.00997109
  h(Day-282)   -0.00831905
  h(Day-319)    0.17945773
  h(Day-328)   -0.51574151
  h(Day-332)    0.50725158
  h(Day-336)   -0.20323060
  h(1-NumCI)    0.11782107

  Selected 11 of 12 terms, and 2 of 2 predictors
  Importance: Day, NumCI
  Number of terms at each degree of interaction: 1 10 (additive model)
  GCV 0.8660403    RSS 5708.129    GRSq 0.1342208    RSq 0.1394347
```

Note that the model coefficients shown here have not been post-processed. The final model coefficients can be found with `coef(fdaModel)`. To predict:

```
> predict(fdaModel, head(training[-pre2008,]))
  [1] successful successful successful successful successful successful
  Levels: successful unsuccessful
```

The `train` function can be used with `method = "fda"` to tune this model over the number of retained terms. Additionally, the `varImp` function from this package determines predictor importance in the same manner as for MARS models (described in Sect. 7.2).

Support Vector Machines

As discussed in the regression chapter, there are a number of R packages with implementations for support vector machine and other kernel methods, including e1071, kernlab, klaR, and svmPath. The most comprehensive of these is the kernlab package.

The syntax for SVM classification models is largely the same as the regression case. Although the `epsilon` parameter is only relevant for regression, a few other parameters are useful for classification:

- The logical `prob.model` argument triggers `ksvm` to estimate an additional set of parameters for a sigmoidal function to translate the SVM decision values to class probabilities using the method of Platt (2000). If this option is not set to `TRUE`, class probabilities cannot be predicted.
- The `class.weights` argument assigns asymmetric costs to each class (Osuna et al. 1997). This can be especially important when one or more specific types of errors are more harmful than others or when there is a severe class imbalance that biases the model to the majority class (see Chap. 16). The syntax here is to use a *named* vector of weights or costs. For example, if there was a desire to bias the grant model to detect unsuccessful grants, then the syntax would be

  ```
  class.weights = c(successful = 1, unsuccessful = 5)
  ```

  This makes a false-negative error five times more costly than a false-positive error. Note that the implementation of class weights in `ksvm` affects the predicted class, but the class probability model is unaffected by the weights (in this implementation). This feature is utilized in Chap. 17.

The following code fits a radial basis function to the reduced set of predictors in the grant data:

```
> set.seed(202)
> sigmaRangeReduced <- sigest(as.matrix(training[,reducedSet]))
> svmRGridReduced <- expand.grid(.sigma = sigmaRangeReduced[1],
+                                .C = 2^(seq(-4, 4)))
> set.seed(476)
> svmRModel <- train(training[,reducedSet], training$Class,
+                    method = "svmRadial",
+                    metric = "ROC",
+                    preProc = c("center", "scale"),
+                    tuneGrid = svmRGridReduced,
+                    fit = FALSE,
+                    trControl = ctrl)


> svmRModel

  8190 samples
   252 predictors
     2 classes: 'successful', 'unsuccessful'
```

```
Pre-processing: centered, scaled
Resampling: Repeated Train/Test Splits (1 reps, 0.75%)

Summary of sample sizes: 6633

Resampling results across tuning parameters:

  C       ROC    Sens   Spec
  0.0625  0.866  0.775  0.787
  0.125   0.88   0.842  0.776
  0.25    0.89   0.867  0.772
  0.5     0.894  0.851  0.784
  1       0.895  0.84   0.804
  2       NaN    0.814  0.814
  4       0.887  0.814  0.812
  8       0.885  0.804  0.814
  16      0.882  0.805  0.818

Tuning parameter 'sigma' was held constant at a value of 0.00117
ROC was used to select the optimal model using  the largest value.
The final values used for the model were C = 1 and sigma = 0.00117.
```

When the outcome is a factor, the function automatically uses `prob.model = TRUE`.

Other kernel functions can be defined via the `kernel` and `kpar` arguments. Prediction of new samples follows the same pattern as other functions:

```
> library(kernlab)
> predict(svmRModel, newdata = head(training[-pre2008, reducedSet]))
  [1] successful successful successful successful successful successful
  Levels: successful unsuccessful
> predict(svmRModel, newdata = head(training[-pre2008, reducedSet]),
+         type = "prob")
    successful unsuccessful
  1  0.9522587   0.04774130
  2  0.8510325   0.14896755
  3  0.8488238   0.15117620
  4  0.9453771   0.05462293
  5  0.9537204   0.04627964
  6  0.5009338   0.49906620
```

K-Nearest Neighbors

Fitting a KNN classification model has similar syntax to fitting a regression model. In this setting, the caret package with `method` set to `"knn"` generates the model. The syntax used to produce the top of Fig. 13.14 is

```
> set.seed(476)
> knnFit <- train(training[,reducedSet], training$Class,
```

```
+                  method = "knn",
+                  metric = "ROC",
+                  preProc = c("center", "scale"),
+                  tuneGrid = data.frame(.k = c(4*(0:5)+1,
+                                               20*(1:5)+1,
+                                               50*(2:9)+1)),
+                  trControl = ctrl)
```

The following code predicts the test set data and the corresponding ROC curve:

```
> knnFit$pred <- merge(knnFit$pred,  knnFit$bestTune)
> knnRoc <- roc(response = knnFit$pred$obs,
+               predictor = knnFit$pred$successful,
+               levels = rev(levels(knnFit$pred$obs)))
> plot(knnRoc, legacy.axes = TRUE)
```

Naïve Bayes

The two main functions for fitting the naïve Bayes models in R are `naiveBayes` in the e1071 package and `NaiveBayes` in the klaR package. Both offer Laplace corrections, but the version in the klaR package has the option of using conditional density estimates that are more flexible.

Both functions accept the formula and non-formula approaches to specifying the model terms. However, feeding these models binary dummy variables (instead of a factor variable) is problematic since the individual categories will be treated as numerical data and the model will estimate the probability density function (i.e., $Pr[X]$) from a continuous distribution, such as the Gaussian.

To follow the strategy described above where many of the predictors are converted to factor variables, we create alternate versions of the training and test sets:

```
> ## Some predictors are already stored as factors
> factors <- c("SponsorCode", "ContractValueBand", "Month", "Weekday")
> ## Get the other predictors from the reduced set
> nbPredictors <- factorPredictors[factorPredictors %in% reducedSet]
> nbPredictors <- c(nbPredictors, factors)


> ## Leek only those that are needed
> nbTraining <- training[, c("Class", nbPredictors)]
> nbTesting <- testing[, c("Class", nbPredictors)]


> ## Loop through the predictors and convert some to factors
> for(i in nbPredictors)
+   {
+     varLevels <- sort(unique(training[,i]))
```

```
+     if(length(varLevels) <= 15)
+       {
+         nbTraining[, i] <- factor(nbTraining[,i],
+                                   levels = paste(varLevels))
+         nbTesting[, i] <- factor(nbTesting[,i],
+                                  levels = paste(varLevels))
+       }
+   }
```

Now, we can use the `NaiveBayes` function's formula interface to create a model:

```
> library(klaR)
> nBayesFit <- NaiveBayes(Class ~ .,
+                         data = nbTraining[pre2008,],
+                         ## Should the non-parametric estimate
+                         ## be used?
+                         usekernel = TRUE,
+                         ## Laplace correction value
+                         fL = 2)
> predict(nBayesFit, newdata = head(nbTesting))
$class
      6641       6647       6649       6650       6655       6659
successful successful successful successful successful successful
Levels: successful unsuccessful

$posterior
     successful unsuccessful
6641  0.9937862 6.213817e-03
6647  0.8143309 1.856691e-01
6649  0.9999078 9.222923e-05
6650  0.9992232 7.768286e-04
6655  0.9967181 3.281949e-03
6659  0.9922326 7.767364e-03
```

In some cases, a warning appears: "Numerical 0 probability for all classes with observation 1." The `predict` function for this model has an argument called `threshold` that replaces the zero values with a small, nonzero number (0.001 by default).

The `train` function treats the density estimate method (i.e., `usekernel`) and the Laplace correction as tuning parameters. By default, the function evaluates probabilities with the normal distribution and the nonparametric method (and no Laplace correction).

Exercises

13.1. Use the hepatic injury data from the previous exercise set (Exercise 12.1). Recall that the matrices `bio` and `chem` contain the biological assay and chemical fingerprint predictors for the 281 compounds, while the vector `injury` contains the liver damage classification for each compound.

(a) Work with the same training and testing sets as well as pre-processing steps as you did in your previous work on these data. Using the same classification statistic as before, build models described in this chapter for the biological predictors and separately for the chemical fingerprint predictors. Which model has the best predictive ability for the biological predictors and what is the optimal performance? Which model has the best predictive ability for the chemical predictors and what is the optimal performance? Does the nonlinear structure of these models help to improve the classification performance?
(b) For the optimal models for both the biological and chemical predictors, what are the top five important predictors?
(c) Now combine the biological and chemical fingerprint predictors into one predictor set. Re-train the same set of predictive models you built from part (a). Which model yields best predictive performance? Is the model performance better than either of the best models from part (a)? What are the top 5 important predictors for the optimal model? How do these compare with the optimal predictors from each individual predictor set? How do these important predictors compare the predictors from the linear models?
(d) Which model (either model of individual biology or chemical fingerprints or the combined predictor model), if any, would you recommend using to predict compounds' hepatic toxicity? Explain.

13.2. Use the fatty acid data from the previous exercise set (Exercise 12.2).

(a) Use the same data splitting approach (if any) and pre-processing steps that you did in the previous chapter. Using the same classification statistic as before, build models described in this chapter for these data. Which model has the best predictive ability? How does this optimal model's performance compare to the best linear model's performance? Would you infer that the data have nonlinear separation boundaries based on this comparison?
(b) Which oil type does the optimal model most accurately predict? Least accurately predict?