Chapter 12
Discriminant Analysis and Other Linear Classification Models

In general, discriminant or classification techniques seek to categorize samples into groups based on the predictor characteristics, and the route to achieving this minimization is different for each technique. Some techniques take a mathematical path [e.g., linear discriminant analysis (LDA)], and others take an algorithmic path (e.g., k-nearest neighbors).

Classical methods such as LDA and its closely related mathematical cousins (partial least squares discriminant analysis (PLSDA), logistic regression, etc.) will be discussed in this chapter and will focus on separating samples into groups based on characteristics of predictor variation.

12.1 Case Study: Predicting Successful Grant Applications

These data are from a 2011 Kaggle competition sponsored by the University of Melbourne where there was interest in predicting whether or not a grant application would be accepted. Since public funding of grants had decreased over time, triaging grant applications based on their likelihood of success could be important for estimating the amount of potential funding to the university. In addition to predicting grant success, the university sought to understand factors that were important in predicting success. As we have discussed throughout the regression chapters, there is often a trade-off between models that are developed for understanding and models that are developed for prediction. The same is true for classification models; this will be illustrated in this and the following chapters.

In the contest, data on 8,708 grants between the years 2005 and 2008 were available for model building and the test set contained applications from 2009 to 2010. The winning entry achieved an area under the ROC curve of 0.968

M. Kuhn and K. Johnson, *Applied Predictive Modeling*,
DOI 10.1007/978-1-4614-6849-3_12,

on the test set. The first and second place winners discuss their approaches to the data and modeling on the Kaggle blog.[1]

The data can be found at the Kaggle web site,[2] but only the training set data contain the outcomes for the grants. Many pieces of information were collected across grants, including whether or not the grant application was successful. The original data contained many predictors such as:

- The role of each individual listed on the grant. Possible values include chief investigator (shortened to "CI" in the data), delegated researcher (DR), principal supervisor (PS), external advisor (EA), external chief investigator (ECI), student chief investigator (SCI), student researcher (SR), honorary visitor (HV), or unknown (UNK). The total number of individuals listed on the grant ranged from 1 to 14.
- Several characteristics of each individual on the grant, such as their date of birth, home language, highest degree, nationality, number of prior successful (and unsuccessful) grants, department, faculty status, level of seniority, length of employment at the university, and number of publications in four different grades of journals.
- One or more codes related to Australia's research fields, courses and disciplines (RFCD) classification. Using this, the grant can be classified into subgroups, such as Applied Economics, Microbiology, and Librarianship. There were 738 possible values of the RFCD codes in the data. If more than one code was specified for a grant, their relative percentages were recorded. The RFCD codes listed by the Australian Bureau of Statistics[3] range from 210,000 to 449,999. There were many grants with nonsensical codes (such as 0 or 999,999) that were grouped into an unknown category for these analyses.
- One or more codes corresponding to the socio-economic objective (SEO) classification. This classification describes the intended purpose of the grant, such as developing construction activities or health services. If more than one code was specified for a grant, their relative percentages were recorded. Like the RFCD codes, there were some values in the data that did not map to any of the codes listed by the Australian government and were grouped into an unknown category.
- The submission date of the grant
- The monetary value of the grant, binned into 17 groups
- A grant category code which describes the type sponsor as well as a code for the specific sponsor

One of the first steps in the model building process is to transform, or encode, the original data structure into a form that is most informative for the model (i.e., *feature engineering*). This encoding process is critical and must be done

[1] http://blog.kaggle.com/.

[2] http://www.kaggle.com/c/unimelb.

[3] The RFCD codes can be found at http://tinyurl.com/25zvts while the SEO codes can be found at http://tinyurl.com/8435ae4.

with foresight into the analyses that will be performed so that appropriate predictors can be elucidated from the original data. Failure to appropriately format the predictors can prevent developing effective predictive models.

The original form of the grant data is not conducive to modeling. For example, many fields are broken down for each individual involved in the grant. As such, there are 15 columns in the data for each individual. Since there could be as many as 14 individuals associated with a grant, there are a large number of columns for a grant, many of which have no data.

How to encode these data is a primary first question. For example, since there are often multiple individuals associated with the grant, how should this information be represented in the data? Similarly, when there are multiple RFCD codes and associated percentages, in what manner should these data enter the models? Additionally, these data contain many missing values which we must also handle before building a predictive model. We must think through all of these questions while keeping in mind the goal of predicting the success of a grant application.

Given this goal, we took the following steps. First, a group of predictors was created that describe how many investigators were on the grant broken up by role (e.g., chief investigator). Second, groups of role-specific count variables were also created for the home language, nationality, degree, birth year, department, and grant history. For example, one variable counts the number of chief investigators from Australia while another counts the total number of successful grants from all delegated researchers on the grant. For publication data, the total number of publications in the four tiers of journals was aggregated across all roles. The duration of employment was similarly aggregated across all roles.

Indicator variables for each sponsor code and grant category were also created. For the RFCD and SEO codes, the number of non-zero percentages for each grant was used. Finally, indicators were generated for the month and day or the week that the grant was submitted. In all, 1,784 possible predictors were created using this encoding scheme.

As a result, the vast majority of these predictors are discrete in nature (i.e., either 0/1 dummy variables or counts) with many 0 values. Since many of the predictors are categorical in nature, missing values were encoded as "unknown." For example, 912 grants had missing category codes. A binary predictor for missing grant categories was created to capture this information.

As described in Chap. 3, some predictive models have different constraints on the type of predictors that they can utilize. For example, in these data, a significant number of predictors had pair-wise absolute correlations that were larger than 0.99. Because of this, a high-correlation filter was used on the predictor set to remove these highly redundant predictors from the data. In the end, 56 predictors were eliminated from the data for this reason. The binary nature of many of predictors also resulted in many cases where the data were very sparse and unbalanced. For the RFCD codes, 95 % of the predictors had less than 50 non-zero indicators. This high degree of class imbalance

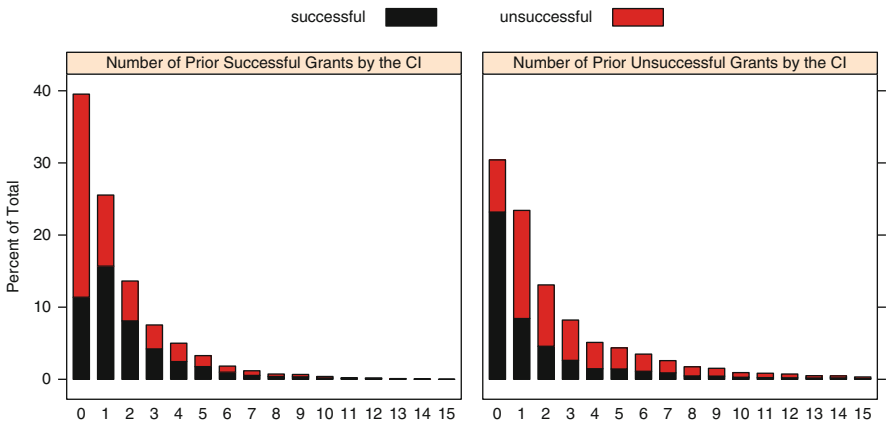

Fig. 12.1: The top two continuous predictors associated with grant success based on the pre-2008 data. Prior success in receiving a grant by the chief investigator as well as prior failure in receiving a grant are most highly associated with the success or failure of receiving a future grant. The x-axis is truncated to 15 grants so that the long tails of the distribution do not obfuscate the differences

indicates that many of the predictors could be classified as *near-zero variance predictors* described in Chap. 3, which can lead to computational issues in many of the models.

Since not all models are affected by this issue, two different sets of predictors were used, depending on the model. The "full set" of predictors included all the variables regardless of their distribution (1,070 predictors). The "reduced set" was developed for models that are sensitive to sparse and unbalanced predictors and contained 252 predictors. In subsequent chapters and sections, the text will describe which predictor set was used for each model.[4] As a reminder, the process of removing predictors without measuring their association with the outcome is *unsupervised feature selection*. Although a few models that use supervised feature selection are described in this chapter, a broader discussion of feature selection is tabled until Chap. 19.

A cursory, univariate review of the newly encoded data uncovers a few interesting relationships with the response. Two continuous predictors, the number of prior successful and unsuccessful grant applications by the chief investigator, were highly associated with grant application success. The distributions of these predictors by current grant application success are displayed in Fig. 12.1. Not surprisingly these histograms suggest that prior success or

[4] However, there are several tree-based methods described in Chap. 14 that are more effective if the categorical predictors are not converted to dummy variables. In these cases, the full set of categories are used.

Table 12.1: Statistics for the three categorical predictors with highest univariate association with the success funding of a grant

	Grant success					
	Yes	No	N	Percent	Odds	Odds ratio
Contract value band						
A	1,501	818	2,319	64.7	1.835	2.84
Other bands	2,302	3,569	5,871	39.2	0.645	
Sponsor						
Unknown	732	158	890	82.2	4.633	6.38
Known	3,071	4,229	7,300	42.1	0.726	
Month						
January	480	45	525	91.4	10.667	13.93
Other months	3,323	4,342	7,665	43.4	0.765	

failure shifts the respective distribution towards current success or failure. Given this knowledge, we would expect these predictors to play significant roles for most any classification model.

Three categorical predictors (Contract Value Band A, Sponsor Unknown, and January) had the highest univariate associations with grant application success. The associations for these three predictors were not strong but do reveal some useful patterns. Table 12.1 shows the data and suggests that grant submissions with a large monetary value, an unknown sponsor, or a submission in January are associated with greater funding success. Looking at the problem a different way, unsuccessful grant applications are likely to have a smaller monetary value, to have a known sponsor, and are submitted in a month other than January. The table has the success rates for each group and also the *odds*, which is ratio of the probability of a success grant over the probability of an unsuccessful grant. One common method for quantifying the predictive ability of a binary predictor (such as these) is the *odds ratio*. For example, when a grant is submitted in January the odds are much higher (10.7) than other months (0.8). The ratio of the odds for this predictor suggests that grants submitted in January are 13.9 times more likely to be successful than the other months. Given the high odds ratios, we would expect that these predictors will also have impact on the development of most classification models.

Finally, we must choose how to split the data which is not directly obvious. Before deciding on the splitting approach, it is important to note that the percentage of successful grants varied over the years: 45 % (2005), 51.7 % (2006), 47.2 % (2007), and 36.6 % (2008). Although 2008 had the lowest percentage in the data, there is not yet enough information to declare a downward trend. The data splitting scheme should take into account the *application domain* of the model: how will it be used and what should the criterion be to assess if it

is fit for purpose? The purpose of the model exercise is to create a predictive model to quantify the likelihood of success for new grants, which is why the competition used the most recent data for testing purposes.

If the grant success rate were relatively constant over the years, a reasonable data splitting strategy would be relatively straightforward: take all the available data from 2005 to 2008, reserve some data for a test set, and use resampling with the remainder of the samples for tuning the various models. However, a random test sample across all of the years is likely to lead to a substantially less *relevant* test set; in effect, we would be building models that are focused on the *past* grant application environment.

An alternative strategy would be to create models using the data before 2008, but tune them based on how well they fit the 2008 data. Essentially, the 2008 data would serve as a single test set that is more relevant in time to the original test set of data from 2009 to 2010. However, this is a single "look" at the data that do not provide any real measure of uncertainty for model performance. More importantly, this strategy may lead to substantial over-fitting to this particular set of 2008 data and may not generalize well to subsequent years. For example, as with regression models, there are a number of classification models that automatically perform feature selection while building the model. One potential methodology error that may occur with a single test set that is evaluated many times is that a set of predictors may be selected that work only for these particular 2008 grant applications. We would have no way of knowing if this is the case until another set of recent grant applications are evaluated.

How do these two approaches compare for these data? Figure 12.2 shows the results for a support vector machine classification model discussed in detail in Sect. 13.4 but is similar to the support vector regression model described in Sect. 7.3. Using the radial basis function kernel previously discussed, the tuning parameters are the kernel parameter, σ, and the cost value, C, used to control for over-fitting. Several values of the radial basis function kernel parameter were evaluated as well as several values of the cost function.

Figure 12.2 shows both approaches to tuning the model. Two tuning parameter profiles for the support vector machine are shown:

- The first model is built on 8,189 grants that include all the pre-2008 data and 25 % of the 2008 data ($n = 290$). To choose the regularization and kernel parameter(s), 10-fold cross-validation is used. The performance profile across the cost parameter is shown as a blue line in Fig. 12.2 (this profile uses the optimal value of the kernel parameter). A set of 2008 grants ($n = 1,785$) is held back to validate the choice of the final tuning parameter (blue profile).
- The second model is exclusively built on pre-2008 data, and the value of the tuning parameter is chosen to maximize the area under the ROC curve for the 2008 grants. No additional samples are held back for verifying the parameter choice (red profile).

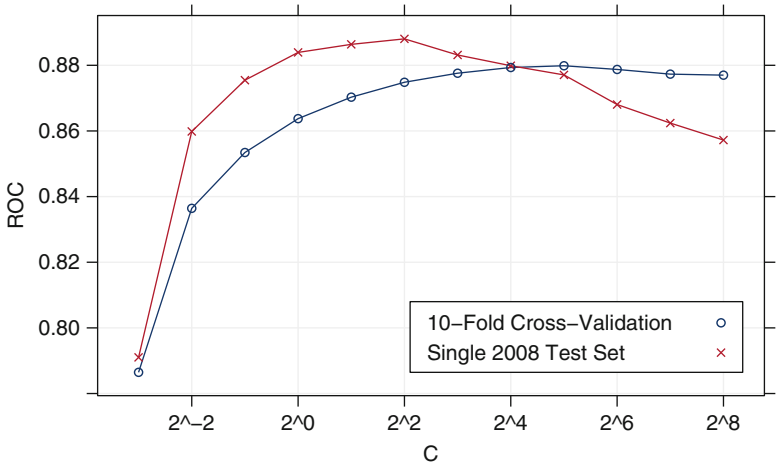

Fig. 12.2: Two models for grant success based on the pre-2008 data but with different data sets used to tune the model

The blue profile suggests that a value of 32 for the cost parameter will yield an area under the ROC curve of 0.88. The red profile shows the results of evaluating only the 2008 data (i.e., no resampling). Here, the tuning process suggests that a smaller cost value is needed (4) to achieve an optimal model with an area under the ROC curve of 0.89. Firstly, given the amount of data to evaluate the model, it is problematic that the curves suggest different tuning parameters. Secondly, when the cross-validated model is evaluated on the 2008 data, the area under the ROC curve is substantially smaller (0.83) than the cross-validation results indicate.

The compromise taken here is to build models on the pre-2008 data and tune them by evaluating a random sample of 2,075 grants from 2008. Once the optimal parameters are determined, final model is built using these parameters and the entire training set (i.e., the data prior to 2008 and the additional 2,075 grants). A small holdout set of 518 grants from 2008 will be used to ensure that no gross methodology errors occur from repeatedly evaluating the 2008 data during model tuning. In the text, this set of samples is called the *2008 holdout set*. This small set of year 2008 grants will be referred to as the *test set* and will not be evaluated until set of candidate models are identified (in Chap. 15). These strategies are summarized in Table 12.2.

To be clear, there is no single, clean approach to handling this issue for data that appear to be evolving over time. Therefore the practitioner must understand the modeling objectives and carefully craft a plan for training and testing models. In this case, the grant data have the luxury of a moderate amount of recent data; there are enough data to split out a small holdout set

Table 12.2: A schematic for the data splitting strategy for the grant application data used in this and subsequent chapters

	Model tuning		Final model	
	Training	Holdout	Training	Holdout
Pre-2008 ($n = 6,633$)	×		×	
2008 ($n = 1,557$)		×	×	
2008 ($n = 518$)				×

without significantly impairing the tuning process. The disadvantages of this approach are:

1. An assumption is being made that the model parameters derived from the tuning process will be appropriate for the final model, which uses pre-2008 data as well as the 2,075 grants from 2008.
2. Since the final model uses some 2008 grants, the performance on the test set is likely to be better than the results generated in the tuning process (where the model parameters were not exposed to year 2008 grants).

In Chap. 15, the test set results will be compared to those generated during model tuning.

12.2 Logistic Regression

Linear regression (Sect. 6.2) forms a model that is linear in the parameters, and these parameters are obtained by minimizing the sum of the squared residuals. It turns out that the model that minimizes the sum of the squared residuals also produces *maximum likelihood estimates* of the parameters when it is reasonable to assume that the model residuals follow a normal (i.e., Gaussian) distribution.

Maximum likelihood parameter estimation is a technique that can be used when we are willing to make assumptions about the probability distribution of the data. Based on the theoretical probability distribution and the observed data, the likelihood function is a probability statement that can be made about a particular set of parameter values. If two sets of parameters values are being identified, the set with the larger likelihood would be deemed more consistent with the observed data.

The probability distribution that is most often used when there are two classes is the binomial distribution.[5] This distribution has a single parameter, p, that is the probability of an event or a specific class. For the grant data, suppose p is the probability of a successful grant. In the pre-2008 grants, there were a total of 6,633 grants and, of these, 3,233 were successful. Here, the form of the binomial likelihood function would be

$$L(p) = \binom{6633}{3233} p^{3233}(1-p)^{6633-3233}, \tag{12.1}$$

where the exponents for p and $1-p$ reflect the frequencies of the classes in the observed data. The first part of the equation is "n choose r" and accounts for the possible ways that there could be 3,233 successes and 3,400 failures in the data.

The maximum likelihood estimator would find a value of p that produces the largest value for $f(p)$. It turns out that the sample proportion, $3233/6633 = 0.487$, is the maximum likelihood estimate in this situation.

However, we know that the success rate is affected by multiple factors and we would like to build a model that uses those factors to produce a more refined probability estimate. In this case, we would *re-parameterize* the model so that p is a function of these factors. Like linear regression, the logistic regression model has an intercept in addition to slope parameters for each model term. However, since the probability of the event is required to be between 0 and 1, we cannot be guaranteed that a slope and intercept model would constrain values within this range. As discussed earlier in the chapter, if p is the probability of an event, the odds of the event are then $p/(1-p)$. Logistic regression models the log odds of the event as a linear function:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_P x_P. \tag{12.2}$$

Here, P is the number of predictors. The right-hand side of the equation is usually referred to as the *linear predictor*. Since the log of the odds can range from $-\infty$ to ∞, there is no concern about the range of values that the linear predictors may produce. By moving some terms around, we get back to a function of the event probability:

$$p = \frac{1}{1 + \exp\left[-(\beta_0 + \beta_1 x_1 + \cdots + \beta_P x_P)\right]} \tag{12.3}$$

This nonlinear function is a sigmoidal function of the model terms and constrains the probability estimates to between 0 and 1. Also, this model produces linear class boundaries, unless the predictors used in the model are

[5] Data with three or more classes are usually modeled using the *multinomial distribution*. See Agresti (2002) for more details.

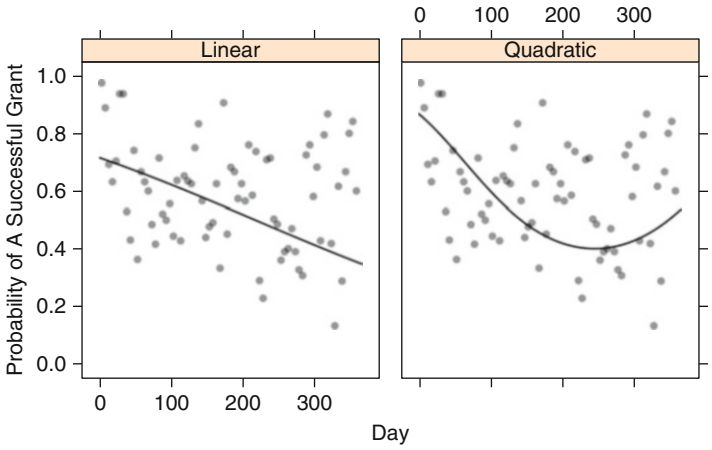

Fig. 12.3: Two logistic regression models that relate the probability of a successful grant to the numeric day of the year. In this plot, the day values were binned into 5-day periods. The model fits did not use the binned version of the predictor; the log odds were modeled as a function of the day of the year (e.g., 1, 2, ..., 365)

nonlinear versions of the data (e.g., the squared values of a predictor are used as one of the x_j model terms).

Now that we have a way to relate our model to the parameter of the binomial distribution, we can find candidate values of the parameters (β) and, with our observed data, compute a value of the likelihood function. Once we find β values that appear to maximize the likelihood for our data, these values would be used to predict sample outcomes.

Logistic regression and ordinary linear regression fall into a larger class of techniques called generalized linear models (GLMs) that encompass many different probability distributions. Dobson (2002) provides an excellent introduction to these models. They are linear in the sense that some function of the outcome is modeled using the linear predictors, such as the log odds in Eq. 12.2. They often produce nonlinear equations (such as the one for p in Eq. 12.3). Again, note that even though the equation for p is nonlinear, it produces linear classification boundaries.

For example, we could fit a simple logistic regression model to the grant data using a single predictor, such as the numeric day of the year. Figure 12.3 shows the observed probability of a successful grant when the data are binned into 5-day intervals. In this plot, there is a higher success rate associated with the beginning and end of the year. In fact, for the data prior to 2008, there were zero unsuccessful grants in the pool of 343 grants submitted on the first day of the year. During the middle of the year the acceptance rate is roughly decreasing but increases near the end of the year. A simple logistic regression

model would try to estimate a slope corresponding to the day as well as an intercept term. Using the training data, the model fitting routine searches across different values of these two parameters to find a combination that, given the observed training data, maximizes the likelihood of the binomial distribution. In the end, the estimated intercept was 0.919 and the slope parameter was determined to be −0.0042. This means that there was a per day *decrease* in the log odds of 0.0042. The model fit is shown on the left panel of Fig. 12.3. This does not adequately represent the trend in the later part of the year. Another model can be created where a third parameter corresponds to a squared day term. For this model, the estimated intercept now becomes 1.88, the slope for the linear day term is −0.019, and the slope for the quadratic term was estimated to be −0.000038. The right-hand panel of Fig. 12.3 shows a clear improvement in the model but does not quite capture the increase in the success rate at the end of the year. As evidence of the need for an additional term, the area under the ROC curve for the linear model was 0.56, which improves to 0.66 once the additional model term is utilized.

An effective logistic regression model would require an inspection of how the success rate related to each of the continuous predictors and, based on this, may parameterize the model terms to account for nonlinear effects. One efficient method for doing this is discussed in Harrell (2001), where restricted cubic splines are used to create flexible, adaptive versions of the predictors that can capture many types of nonlinearities. The chapter's Computing section has more details on this methodology. Another approach is a generalized additive model (Hastie and Tibshirani 1990; Hastie et al. 2008), which also uses flexible regression methods (such as splines) to adaptively model the log odds. We refer the reader to the reference texts to learn more about these methods.

For the grant data, the full set of predictors was used in a logistic regression model. The other continuous predictors were evaluated for nonlinearities. However, many of the predictors have few data points on one or more extremes of the distributions, such as the two predictors shown in Fig. 12.1. This increases the difficulty in prescribing an exact functional form for the predictors. Using this predictor set, the logistic regression model was able to achieve an area under the ROC curve of 0.78, a sensitivity of 77 % and a specificity of 76.1 %, on the 2008 holdout set.

Many of the categorical predictors have sparse and unbalanced distributions. Because of this, we would expect that a model using the full set of predictors would perform worse than the set that has near-zero variance predictors removed. For the reduced set of 253 variables, the area under the ROC curve was 0.87, the sensitivity was 80.4 %, and the specificity was 82.2 % (Fig. 12.4). The confusion matrix is shown in Table 12.3. With this particular model, there was a substantial improvement gained by removing these predictors.

For logistic regression, formal statistical hypothesis tests can be conducted to assess whether the slope coefficients for each predictor are statistically significant. A Z statistic is commonly used for these models (Dobson 2002), and

Table 12.3: The 2008 holdout set confusion matrix for the logistic regression model

	Observed class	
	Successful	Unsuccessful
Successful	439	236
Unsuccessful	131	751

This model had an overall accuracy of 76.4 %, a sensitivity of 77 %, and a specificity of 76.1 %

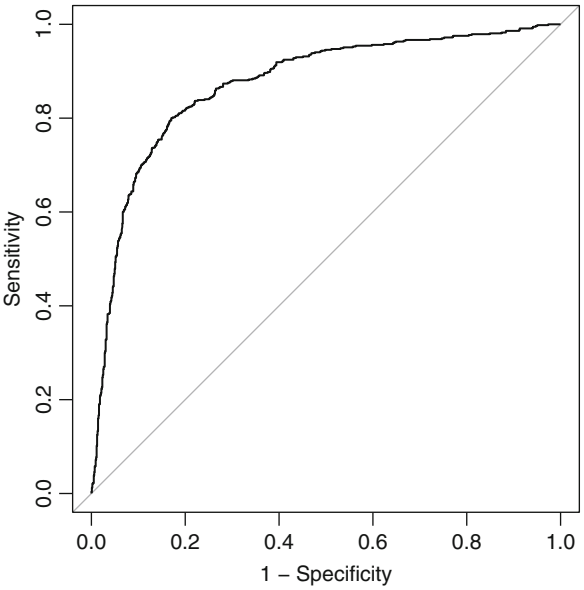

Fig. 12.4: The ROC curve for the grant data test set using a logistic regression model. The AUC is 0.87

is essentially a measure of the signal-to-noise ratio: the estimated slope is divided by its corresponding standard error. Using this statistic, the predictors can be ranked to understand which terms had the largest effect on the model. For these data, the five most important predictors were the number of unsuccessful grants by chief investigators, the number of successful grants by chief investigators, contract value band F, contract value band E, and numeric day of the year (squared).

The logistic regression model is very popular due to its simplicity and ability to make inferential statements about model terms. For example, a researcher may want to formally evaluate whether the day of the calendar year has a statistically significant relationship with the probability of grant

acceptance. Harrell (2001) is an excellent resource for developing statistical models for the purpose of making inferential statements about model parameters.

This model can also be effective when the goal is solely prediction, but, as demonstrated above, it does require the user to identify effective representations of the predictor data that yield the best performance. As will be shown in the later sections, there are other classification models that empirically derive these relationships in the course of model training. If the model will only be utilized for prediction, these techniques may be more advantageous.

12.3 Linear Discriminant Analysis

The roots of LDA date back to Fisher (1936) and Welch (1939). Each of these researchers took a different perspective on the problem of obtaining optimal classification rules. Yet, as we will see, each came to find the same rule in the two-group classification setting. In this section we will provide highlights of both of these approaches to capture their necessary technical details while also discussing a few mathematical constructs required for connecting it with other methods discussed later in this chapter.

For the classification problem, Welch (1939) took the approach of minimizing the total probability of misclassification, which depends on class probabilities and multivariate distributions of the predictors. To see Welch's approach, we first need a basic understanding of Bayes' Rule[6] which is

$$Pr[Y = C_\ell|X] = \frac{Pr[Y = C_\ell]Pr[X|Y = C_\ell]}{\sum_{l=1}^{C} Pr[Y = C_l]Pr[X|Y = C_l]} \tag{12.4}$$

$Pr[Y = C_\ell]$ is known as the *prior probability* of membership in class C_ℓ. In practice these values are either known, are determined by the proportions of samples in each class, or are unknown in which case all values of the priors are set to be equal. $Pr[X|Y = C_\ell]$ is the *conditional probability* of observing predictors X, given that the data stem from class C_ℓ. Here we assume that the data are generated from a probability distribution (e.g., multivariate normal distribution), which then defines this quantity's mathematical form. The result of this equation is $Pr[Y = C_\ell|X]$, which is commonly referred to as the *posterior probability* that the sample, X, is a member of class C_ℓ. For a more detailed description of this equation, we refer you to Sect. 13.6.

For a two-group classification problem, the rule that minimizes the total probability of misclassification would be to classify X into group 1 if $Pr[Y = C_1|X] > Pr[Y = C_2|X]$ and into group 2 if the inequality is reversed. Using Eq. 12.4, this rule directly translates to classifying X into group 1 if

$$Pr[Y = C_1]Pr[X|Y = C_1] > Pr[Y = C_2]Pr[X|Y = C_2]. \tag{12.5}$$

[6] Bayes' Rule is examined in more detail in Sect. 13.6.

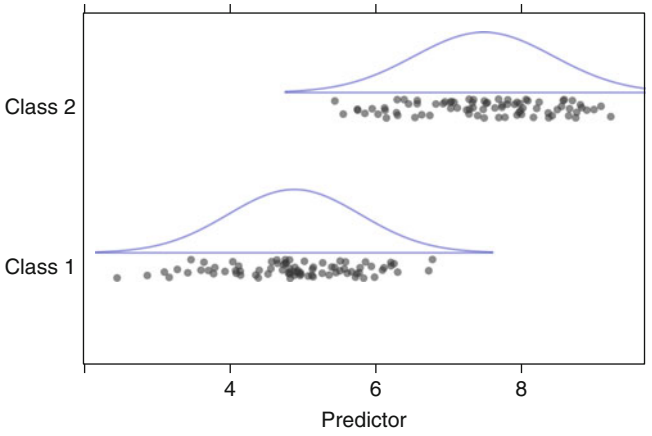

Fig. 12.5: A single predictor is used to classify samples into two groups. The *blue figures* above each group represent the probability density function for a normal distribution determined by the class-specific means and variances

We can easily extend this rule to the more-than-two group case. In this setting, we would classify X into group C_ℓ if $Pr[Y = C_\ell]Pr[X|Y = C_\ell]$ has the largest value across all of the C classes.

Figure 12.5 illustrates this with a single predictor and two classes (the individual data points have been "jittered" to reduce their overlap). The blue illustrations above each group of data points are the probability density function for the normal distribution for each of the classes (i.e., $Pr[X|Y = C_1]$ and $Pr[X|Y = C_2]$). Since there is a single predictor, a new sample is classified by finding its value on the x-axis, then determining the value for each of the probability density functions for each class (in addition to the overall probability, $Pr[X]$, found by pooling both groups). Suppose a new sample had a value of 4 for the predictor. The probability for Class 2 is virtually 0, so this sample would be predicted to belong to the first class.

Since a single predictor is used for this example, it belies the complexity of using Bayes' Rule in practice. For classification, the number of predictors is almost always greater than one and can be extremely large. In more realistic situations, how does one compute quantities such as $Pr[X|Y = C_\ell]$ in many dimensions?[7] What multivariate probability distributions can be used to this effect?

One special, often used scenario is to assume that the distribution of the predictors is multivariate normal. This distribution has two parameters: the multidimensional mean vector $\boldsymbol{\mu}_\ell$ and covariance matrix $\boldsymbol{\Sigma}_\ell$. Further, if we assume that the means of the groups are unique (i.e., a different $\boldsymbol{\mu}_\ell$ for each group), but the covariance matrices are identical across groups, we can solve

[7] This situation is addressed again for the naïve Bayes models in the next chapter.

Eq. 12.5 or the more general multi-class problem to find the linear discriminant function of the ℓth group:

$$X'\mathbf{\Sigma}^{-1}\mu_\ell - 0.5\mu_\ell^{'}\mathbf{\Sigma}^{-1}\mu_\ell + \log\left(Pr[Y = C_\ell]\right). \tag{12.6}$$

In practice, the theoretical means, μ_ℓ, are estimated by using the class-specific means ($\bar{x}_\ell$). The theoretical covariance matrix, $\mathbf{\Sigma}$, is likewise estimated by the observed covariance matrix of the data, $\mathbf{S}$, and X is replaced with an observed sample, $\mathbf{u}$. In the simple example in Fig. 12.5, the sample mean and variance of the data were sufficient to produce the probability distributions shown in blue. For two classes, the class-specific means and variances would be computed along with the sample covariance between the two predictors (to fill out the sample covariance matrix).

Notice that Eq. 12.6 is a linear function in X and defines the separating class boundaries. Hence the method's name: LDA. A slight alteration in the assumptions—that the covariance matrices are not identical across the groups—leads to quadratic discriminant analysis, which will be described in Sect. 13.1.

Fisher formulated the classification problem in a different way. In this approach, he sought to find the linear combination of the predictors such that the between-group variance was maximized relative to the within-group variance. In other words, he wanted to find the combination of the predictors that gave maximum separation between the centers of the data while at the same time minimizing the variation within each group of data.

To illustrate this concept, Fig. 12.6 is an analog to Fig. 12.5. Here, the blue bars indicate the class-specific means. Since there is a single predictor, the between group variance is the square of the difference in these means. The within-group variance would be estimated by a variance that pools the variances of the predictor within each group (illustrated using the red bars in the figure). Taking a ratio of these two quantities is, in effect, a signal-to-noise ratio. Fisher's approach determines linear combinations of the predictors to maximize the signal-to-noise ratio. Similar to the previous discussion of Welch's approach, the situation becomes increasingly more complicated by adding additional predictors. The between- and within-group variances become complex calculations that involve the covariance structure of the predictors, etc.

Mathematically, let $\mathbf{B}$ represent the between-group covariance matrix and $\mathbf{W}$ represent the within-group covariance matrix. Then Fisher's problem can be formulated as finding the value of b such that

$$\frac{b'\mathbf{B}b}{b'\mathbf{W}b} \tag{12.7}$$

is maximized. The solution to this optimization problem is the eigenvector corresponding to the largest eigenvalue of $\mathbf{W}^{-1}\mathbf{B}$. This vector is a linear discriminant, and subsequent discriminants are found through the same op-

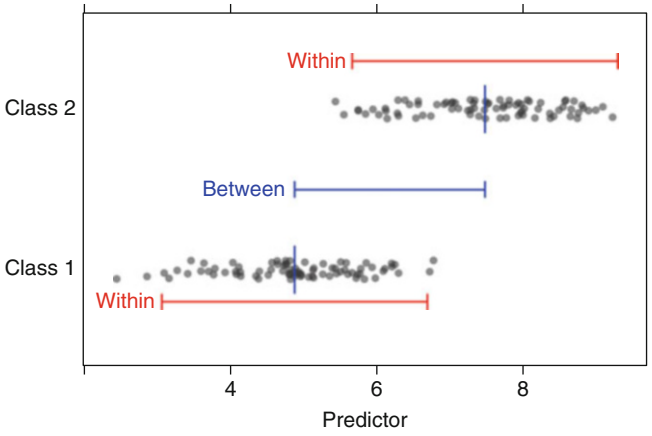

Fig. 12.6: The same data as shown in Fig. 12.5. Here, the between- and within-class variances are illustrated. The within-class ranges are based on the mean ± two standard deviations

timization subject to the constraint that the new directions are uncorrelated with the previous discriminants.

To make Fisher's approach more concrete, let's consider the two-group setting. Solving Eq. 12.7 for two groups gives the discriminant function of $S^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$, where S^{-1} is the inverse of the covariance matrix of the data and is multiplied by the difference between the mean vectors of predictors for each group (i.e., $\bar{\mathbf{x}}_1$ contains the means of each predictor calculated from the class 1 data). In practice, a new sample, $\mathbf{u}$, is projected onto the discriminant function as $\mathbf{u}'S^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$, which returns a discriminant score. A new sample is then classified into group 1 if the sample is closer to the group 1 mean than the group 2 mean in the projection:

$$\left|b'(\mathbf{u} - \overline{\mathbf{x}}_1)\right| - \left|b'(\mathbf{u} - \overline{\mathbf{x}}_2)\right| < 0. \tag{12.8}$$

As a more complex illustration, Fig. 12.7 shows a data set with two classes and two predictors, A and B. The line $A = B$ easily separates these two sets of points into distinct groups. However, this line is *not* the discriminant function. Rather, the discriminant function is instead orthogonal to the line that separates them in space (see Fig. 12.8). With two predictors, the discriminant function for an unknown sample u is

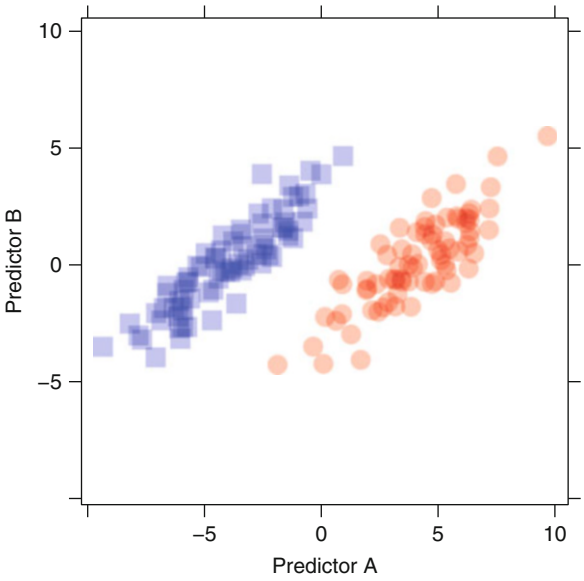

Fig. 12.7: A simple example of two groups of samples that are clearly separable

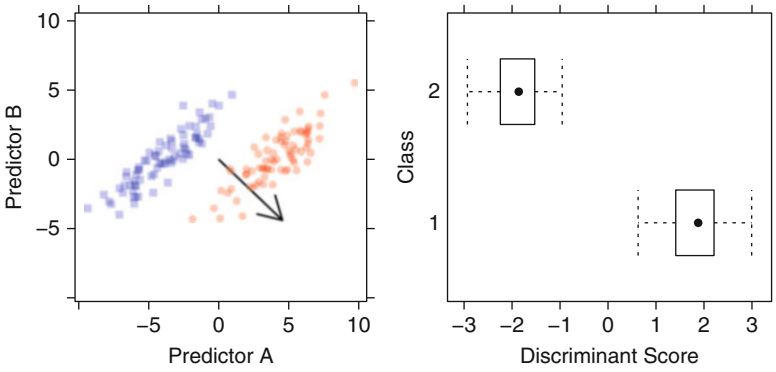

Fig. 12.8: The line at approximately $A = B$ is the vector that visually separates the two groups. Assessing class membership is determined by projecting a sample onto the discriminant vector (*red arrow*) and then calculating its distance from the mean for each group. The sample is then classified into the group which mean is closer. The *box plots* are the distribution of the samples for each class after LDA has been performed, illustrating the maximization of between-to-within group variation

$$
\begin{aligned}
D(u) &= \mathbf{u}' S^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2) \\
&= u_A \left(\frac{(\bar{x}_{1A} - \bar{x}_{2A})s_B^2}{s_A^2 s_B^2 - s_{AB}^2} - \frac{(\bar{x}_{1B} - \bar{x}_{2B})s_{AB}}{s_A^2 s_B^2 - s_{AB}^2} \right) \\
&\quad + u_B \left(\frac{(\bar{x}_{1B} - \bar{x}_{2B})s_A^2}{s_A^2 s_B^2 - s_{AB}^2} - \frac{(\bar{x}_{1A} - \bar{x}_{2A})s_{AB}}{s_A^2 s_B^2 - s_{AB}^2} \right).
\end{aligned}
$$

Here, $\bar{x}_{1A}$ is the sample mean for predictor A calculated using the data from only the first class; $\bar{x}_{2A}$ is the sample mean for A for the second class (the notation is analogous for predictor B). Also, s_A^2 is the sample variance for predictor A (computed with data from both classes), s_B^2 is the sample variance for predictor B, and s_{AB} is the sample covariance between the two predictors.

For this function, note that all of the predictor variances and the between-predictor covariance are used in this equation. When the number of predictors is large, the prediction equation requires a very large number of parameters to be estimated. For $P = 2$ and two classes, this equation uses four means and three variance parameters. In general, the model would require $CP + P(P+1)/2$ parameters with P predictors and C classes. In comparison to logistic regression, a similar model would only estimate three parameters. This difference between the models becomes more significant as the number of predictors grow. However, the value of the extra parameters in LDA models is that the between-predictor correlations are explicitly handled by the model. This should provide some advantage to LDA over logistic regression when there are substantial correlations, although both models will break down when the multicollinearity becomes extreme.

Fisher's formulation of the problem makes intuitive sense, is easy to solve mathematically, and, unlike Welch's approach, involves no assumptions about the underlying distributions of the data. The mathematical optimization constrains the maximum number of discriminant functions that can be extracted to be the lesser of the number of predictors or one less than the number of groups. For example, if we have ten predictors and three groups, we can at most extract two linear discriminant vectors. Similar to PCA, the eigenvalues in this problem represent the amount of variation explained by each component of $\mathbf{W}^{-1}\mathbf{B}$. Hence, LDA is a member of the latent variable routines like PCA and partial least squares (PLS). In practice, the number of discriminant vectors is a tuning parameter that we would estimate using the usual approach of cross-validation or resampling with the appropriate performance criteria.

Closely examining the linear discriminant function leads to two findings which are similar to what we observed with multiple linear regression (Sect. 6.2). First, the LDA solution depends on inverting a covariance matrix, and a unique solution exists only when this matrix is invertible. Just like in regression, this means that the data must contain more samples than predictors, and the predictors must be independent (see the computing section for an approach to determining if the covariance matrix is invertible). When

there are more predictors than samples, or when the predictors are extremely correlated, then just like in regression, a popular approach is to first perform PCA to reduce dimension and generate new uncorrelated predictor combinations. While this approach has been shown to work, the dimension reduction is uninformed about the class structure of the data. To incorporate the class structure into the dimension reduction, we recommend using PLSDA or regularization methods (see following sections). Second, the linear discriminant function is a P-dimensional vector, the values of which are directly paired to the original predictors. The magnitudes of these values can be used to understand the contribution of each predictor to the classification of samples and provide some understanding and interpretation about the underlying system.

From the above discussion, practitioners should be particularly rigorous in pre-processing data before using LDA. We recommend that predictors be centered and scaled and that near-zero variance predictors be removed. If the covariance matrix is still not invertible, then we recommend using PLS or a regularization approach. Similarly, we recommend using PLS or regularization methods (described in sections in this chapter) if there are more predictors than samples. Along the same lines, the practitioner must be aware of the number of samples relative to the number of predictors when using cross-validation routines for methods that depend on inverting a covariance matrix. For example, if the number of samples is 5 % greater than the number of predictors for the training set, and we choose 10-fold cross-validation, then the covariance matrix will not be invertible for any of the folds since all of the folds will have fewer samples than predictors.

We will now illustrate how LDA performs on the grant data. Since LDA is sensitive to near zero variance predictors and collinear predictors,we have reduced the predictor set to 253 predictors (including the squared day term as in logistic regression). Using this subset of predictors, the area under the ROC curve for the 2008 holdout set is 0.89. Table 12.4 shows the confusion matrix for these data and Fig. 12.9 displays the corresponding ROC curve. The light grey line in this plot also shows the ROC curve for the previous logistic regression model.

As we mentioned above, examining the coefficients of the linear discriminant function can provide an understanding of the relative importance of predictors. The top 5 predictors based on absolute magnitude of discriminant function coefficient are numeric day of the year (squared) (2.2), numeric day of the year (-1.9), the number of unsuccessful grants by chief investigators (-0.62), the number of successful grants by chief investigators (0.58), and contract value band A (0.56). Note that this list contains several predictors that the univariate approach identified as being associated with the success of a grant submission. Here, the number of previous unsuccessful grant submissions by the chief investigator is inversely related to the number of previous successful grant submissions by the chief investigator and largest monetary categorization, which is intuitive.

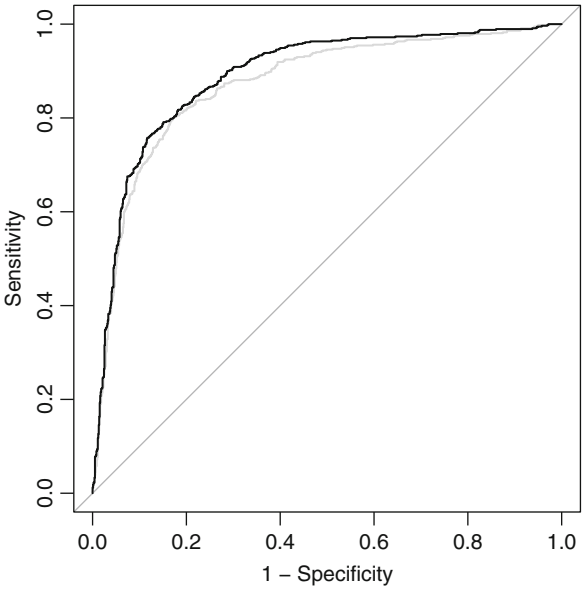

Fig. 12.9: The ROC curve for the 2008 holdout using LDA. The AUC is 0.89. The *lightly shaded line* is the ROC curve for the previous logistic regression model

Table 12.4: The 2008 holdout set confusion matrix for the LDA model

	Observed class	
	Successful	Unsuccessful
Successful	458	175
Unsuccessful	112	812

This model had an overall accuracy of 81.6 %, a sensitivity of 80.4 %, and a specificity of 82.3 %

We can then project the 2008 holdout grants onto this linear discriminant vector and examine the distribution of the discriminant scores (Fig. 12.10). While there is overlap of the distributions for successful and unsuccessful grant applications, LDA yields decent classification performance—especially given that LDA is summarizing the entirety of the underlying relationships in one dimension (Table 12.4).

When we have more samples than predictors, the covariance matrix is invertible, and the data can be decently separated by a linear hyperplane, then LDA will produce a predictively satisfying model that also provides some understanding of the underlying relationships between predictors and response.

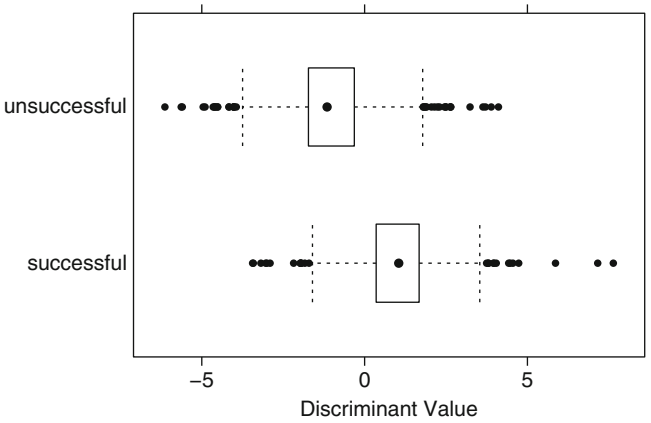

Fig. 12.10: Box plots of the discriminant scores for the 2008 holdout set

The user should be aware, however, that there is a data set scenario that meets these basic requirements but will yield class probability estimates that are overly optimistic. Specifically, the user should be very cautious with LDA predicted class probabilities when the number of samples begins to approach the number of predictors. We will use a simple simulation to illustrate this cautionary note. For 500 samples we have generated data sets containing 10, 100, 200, and 450 predictors all from a random normal population. The response for the samples was also randomly generated placing 250 samples in each category. Therefore, the predictors and response for each of these data sets have no relationship with each other. We will then build LDA models on each of these data sets and examine performance. As we would expect, the test set classification accuracy for each data set is approximately 50 %. Since these data are completely random, we would also expect that the predicted class probabilities for the test set should also be around 0.5. This is true when the number of predictors is small relative to the number of samples. But as the number of predictors grows, the predicted class probabilities become closer to 0 and 1 (Fig. 12.11). At face value, these results seem counterintuitive: test set performance tells us that the model performs as good as a coin toss, but the model is extremely confident about classifying samples into each category.

How can this be? It turns out that these seemingly contradictory conclusions are due to LDA's mathematical construction. Recall that LDA finds an optimal discriminant vector. Geometrically, if the number of samples equals the number of predictors (or dimensions), then we can find at least one vector that perfectly separates the samples. Consider the simplest case where we have two samples and two dimensions. As long as those samples are not in the same location, then we can find one vector (actually infinitely many)

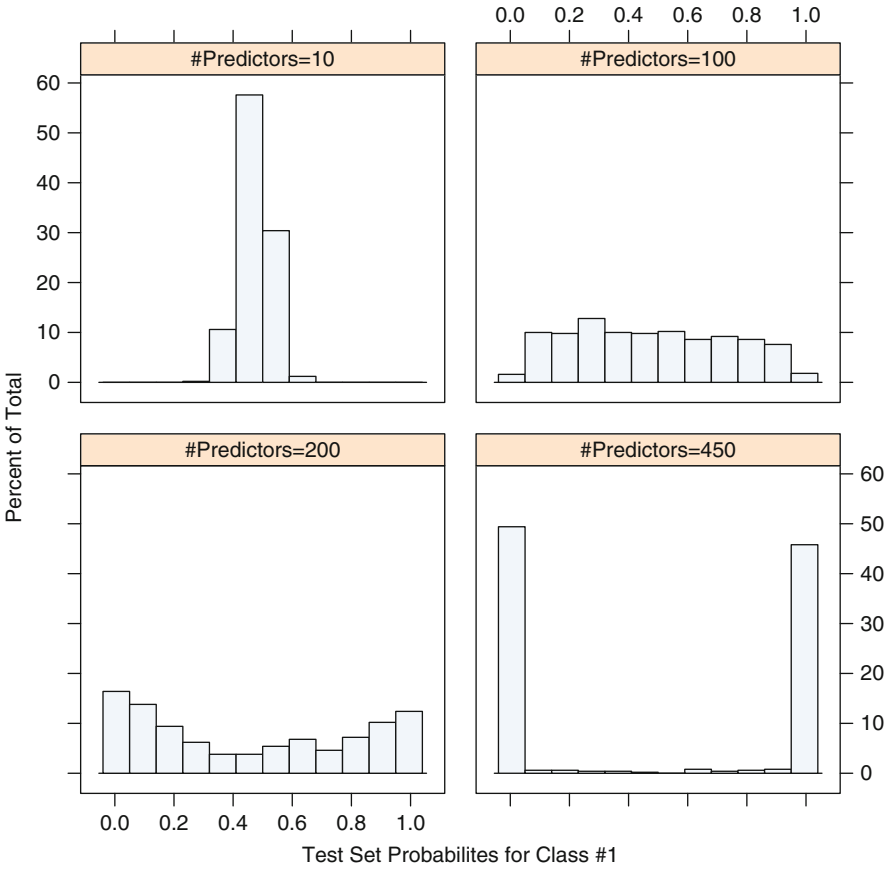

Fig. 12.11: Histograms of test set class probabilities for a simulated two-class example where all predictors are non-informative. As the number of predictors approaches the number of samples (500 in the training set), the class probabilities begin to diverge towards the two extremes (however, the overall accuracy remains near 50 %)

that perfectly separates the two samples. Three samples (two in one class and one in the other) in two dimensions can also be perfectly separated as long as the points are not on a straight line and the single point is not in between the other two from the other class.

Obviously, this data scenario can lead to class probability estimates that are poorly calibrated (as discussed in Sect. 11.3). Due to this inherent problem with LDA, as well as its other fundamental requirements, we recommend that LDA be used on data sets that have at least 5–10 times more samples than predictors. Caution should be applied to LDA results when the ratio dips below 5.

Finally, similar to logistic regression, any predictors used in an LDA model can be transformed and included in the model, like what we observed with the squared numeric value of day of the year in Fig. 12.3. In addition, the practitioner can create cross-product (i.e., interaction) terms among the predictors. Taking this approach is one way to enable LDA to find nonlinear discriminant boundaries. Transforms of or interactions among predictors should only be included, however, if there is good reason to believe that meaningful predictive information exists through these additional predictors. Including additional uninformative predictors will degrade the predictive ability of LDA and could prevent the covariance matrix from being invertible. If the practitioner suspects that nonlinear structure exists between predictors and the classification outcome but is not sure which predictors are involved in this relationship, then we recommend using methods presented in the next chapter.

12.4 Partial Least Squares Discriminant Analysis

As we have noted numerous times throughout the previous chapters, retrospectively or prospectively, measured predictors for any particular problem can be highly correlated or can exceed the number of samples collected. If either of these conditions is true, then the usual LDA approach cannot be directly used to find the optimal discriminant function.

Just like in the regression setting, we can attempt to pre-process our data in a way that removes highly correlated predictors. If more complex correlation structure exist in the data or if the number of predictors still exceeds the number of samples (or the ratio of samples to predictors is too low), then PCA can be used to reduce the predictor-space dimension. However, as previously discussed in Sect. 6.3, PCA may not identify the predictor combinations that optimally separate samples into groups. Recall that Fisher's LDA objective was to find the subspace that maximized the between-to-within group variability. Since PCA does not take into consideration any of the response classification information, we would not expect it to find the optimal subspace for classification. Instead of taking this stepwise approach (PCA-then-LDA) to the overdetermined problem, we recommend using PLS for the purpose of discrimination.

The application of PLS to a classification problem dates back to at least the mid 1980s (Berntsson and Wold 1986). As noted in the regression section, the original NIPALS algorithm was developed and refined in the chemometrics community. Not surprisingly, this community explored and extended the use of PLS to the classification setting and termed this technique PLS discriminant analysis (or PLSDA). Dunn and Wold (1990), for example, illustrated PLSDA on a chemometrics pattern recognition example and showed that it provided a better separation of the samples into groups than the traditional PCA-then-LDA approach.

To build intuition for why PLS would naturally extend to the classification setting, let's briefly return to PLS for regression. Recall that PLS finds latent variables that simultaneously reduce dimension and maximize correlation with a continuous response value (see Fig. 6.9). In the classification setting for a two-group problem, we could naïvely use the samples' class value (represented by 0's and 1's) as the response for this model. Given what we know about PLS for regression, we would then expect that the latent variables would be selected to reduce dimension while optimizing correlation with the categorical response vector. Of course, optimizing correlation isn't the natural objective if classification is the goal—rather, minimizing misclassification error or some other objective related to classification would seem like a better approach. Despite this fact, PLSDA should do better since the group information is being considered while trying to reduce the dimension of the predictor space.

Even though a correlation criterion is being used by PLS for dimension reduction with respect to the response, it turns out that this criterion happens to be doing the right thing. Before getting to the reason why that is true, we first must discuss a practical matter: coding of the response. For a two-group problem, the classes are encoded into a set of 0/1 dummy variables. With C classes, the results would be a set of C dummy variables where each sample has a one in the column representing the corresponding class.[8] As a result, the response in the data is represented by a matrix of dummy variables. Because of this, the problem cannot be solved by the PLS regression approach displayed in Fig. 6.9 and must move to the paradigm for a multivariate response.

Applying PLS in the classification setting with a multivariate response has strong mathematical connections to both canonical correlation analysis and LDA [see Barker and Rayens (2003) for technical details]. Assuming the above coding structure for the outcome, Barker and Rayens (2003) showed that the PLS directions in this context were the eigenvectors of a slightly perturbed between-groups covariance matrix (i.e. $\mathbf{B}$ from LDA).[9] PLS is, therefore, seeking to find optimal group separation while being guided by between-groups information. In contrast, PCA seeks to reduce dimension using the total variation as directed by the overall covariance matrix of the predictors.

This research provides a clear rationale for choosing PLS over PCA when dimension reduction is required when attempting classification. However, Liu

[8] Mathematically, if we know $C - 1$ of the dummy variables, then the value of the last dummy variable is directly implied. Hence, it is also possible to only use $C - 1$ dummy variables.

[9] The perturbed covariance structure is due to the optimality constraints for the response matrix. Barker and Rayens (2003) astutely recognized that the response optimality constraint in this setting did not make sense, removed the constraint, and resolved the problem. Without the response-space constraint, the PLS solution is one that involves exactly the between-group covariance matrix.

Table 12.5: The 2008 holdout set confusion matrix for the PLS model

	Observed class	
	Successful	Unsuccessful
Successful	490	220
Unsuccessful	80	767

This model had an overall accuracy of 80.7 %, a sensitivity of 86 %, and a specificity of 77.7 %. The reduced set of predictors were used to generate this matrix

and Rayens (2007) point out that if dimension reduction is *not* necessary and classification is the goal, then LDA will always provide a lower misclassification rate than PLS. Hence, LDA still has a necessary place in the classification toolbox.

Exactly like PLS for regression, there is one tuning parameter: the number of latent variables to be retained. When performing LDA on the grant data, the reduced set of predictors described on page 278 was used (which eliminated near zero variance predictors and predictors that caused extreme collinearity) with the additional squared term for the day of the year. PLS, however, can produce a model under these conditions. Performance of PLS, as we will shortly see, is affected when including predictors that contain little or no predictive information. Running PLS on the full set of predictors produces an optimal model with an area under the curve ROC of 0.87 based on six components (see Fig. 12.12), a sensitivity of 83.7 %, and a specificity of 77 %. These ROC results are slightly worse than the LDA model, so should we be surprised by this? After all, we included *more* predictors. Actually, including predictors that contain very little or no information about the response degrades the performance of a PLS model. For more details on this phenomenon, see Sect. 19.1.

If PLS performs worse than LDA using a larger set of predictors, then the next logical step would be to examine PLS performance using the reduced set of predictors (also used by LDA).[10] We know from the work of Liu and Rayens (2007) that LDA should outperform PLS in terms of minimizing misclassification errors. For this problem we have chosen to optimize ROC. Using this criterion will LDA still outperform PLS? The optimal number of PLS components with the reduced set of predictors is 4, with a corresponding ROC of 0.89 (Fig. 12.12) and confusion matrix presented in Table 12.5. The smaller set of predictors improves the ROC, uses fewer components to get to that value, and attains a value that is equivalent to the LDA model's performance.

[10] As a reminder, the set of predictors is not being selected on the basis of their association with the outcome. This *unsupervised* selection should not produce *selection bias*, which is an issue described in Sect. 19.5.

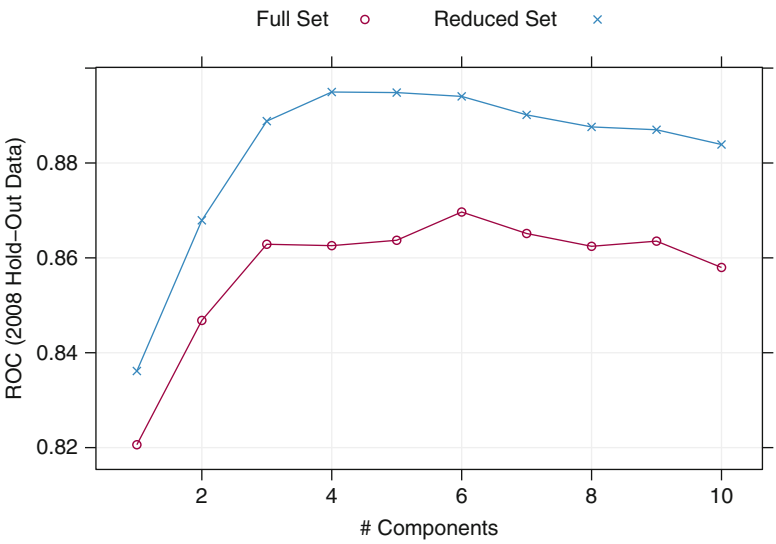

Fig. 12.12: ROC values by component for PLS for the grant data using two sets of predictors. The ROC curve has a maximum area with six components when using all predictors. When using the reduced subset of predictors, the ROC curve has a maximum area with four components

Recall that PLSDA encodes the response as a set of 0/1 dummy variables. Since PLS is a linear model, the predictions for the PLSDA model are not constrained to lie between 0 and 1. The final class is determined by the class with the largest model prediction. However, the raw model predictions require post-processing if class probabilities are required. The *softmax* approach previously described in Sect. 11.1 can be used for this purpose. However, our experience with this technique is that it does not produce meaningful class probabilities—the probabilities are not usually close to 0 or 1 for the most confident predictions. An alternative approach is to use *Bayes' Rule* to convert the original model output into class probabilities (Fig. 12.13). This tends to yield more meaningful class probabilities. One advantage of using Bayes' Rule is that the *prior* probability can be specified. This can be important when the data include one or more rare classes. In this situation, the training set may be artificially balanced and the specification of the prior probability can be used to generate more accurate probabilities. Figure 12.14 shows the class probabilities for the year 2008 grants.[11] While there is overlap between

[11] Recall that the models are built on the pre-2008 data and then tuned based on the year 2008 holdout set. These predictions are from the PLSDA model with four components created using only the pre-2008 data.

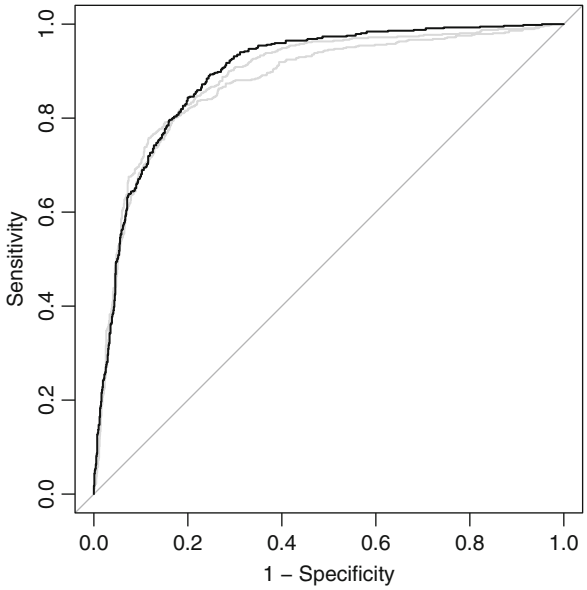

Fig. 12.13: The ROC curve for the 2008 holdout data using PLS (*black*). The AUC is 0.89. The ROC curve for LDA and logistic regression are overlaid (*grey*) for comparison. All three methods perform similarly

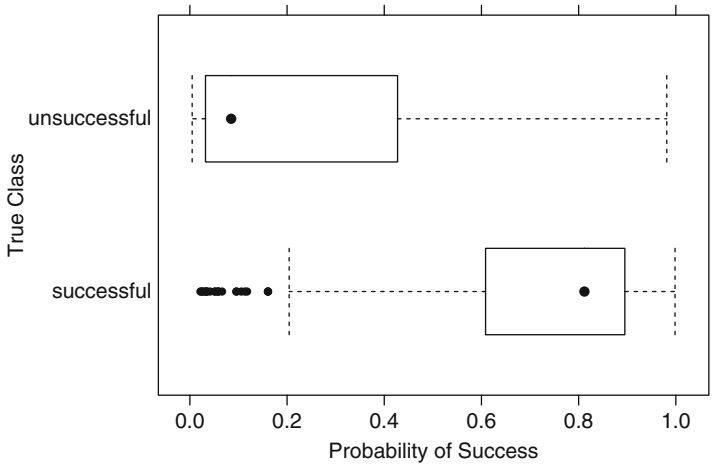

Fig. 12.14: Box plots for the PLSDA class probabilities of the 2008 holdout set calculated using Bayes' Rule

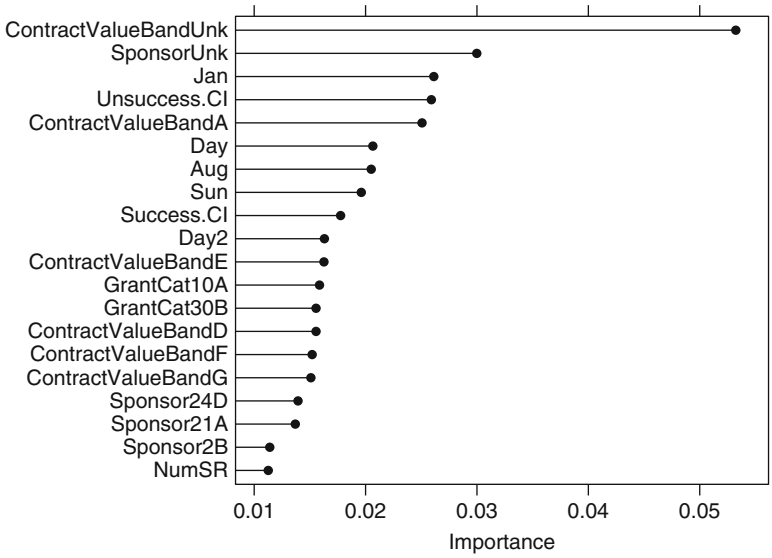

Fig. 12.15: Partial least squares variable importance scores for the grant data

the classes, the distributions are properly shifted; probabilities tend to be higher for the truly successful grants and low for the unsuccessful grants.

As in the regression setting, we can identify PLS predictor importance (Fig. 12.15). For the data at hand, the unknown contract value band has a relatively large importance as compared with the other predictors. Similar to LDA, the success or lack of success of the chief investigator float towards the top of the importance list. Other important predictors to the PLS classification model are contract values, a number of other grant categories, and the months of January and August. Interestingly, Sunday falls towards the top of the list also.

Finally, if nonlinear relationships between the predictors and response exist and the practitioner desires to use PLS to find these relationships then the approaches presented in Sect. 6.3 can be employed.

12.5 Penalized Models

Similar to the regularization methods discussed in Sect. 6.4, many classification models utilize penalties (or regularization) to improve the fit to the data, such as the lasso. In later sections, penalties for inherently nonlinear models, such as support vector machines and neural networks, are discussed.

For example, one might include a penalty term for the logistic regression model in a manner that is very similar to ridge regression. Recall that logis-

tic regression finds parameter values that maximizes the binomial likelihood function, $L(p)$ (see Eq. 12.1). A simple approach to regularizing this model would be to add a squared penalty function to the log likelihood and find parameter estimates that maximize

$$\log L(p) - \lambda \sum_{j=1}^{P} \beta_j^2.$$

Eilers et al. (2001) and Park and Hastie (2008) discuss this model in the context of data where there are a large number of predictors and a small training set sample. In these situations, the penalty term can stabilize the logistic regression model coefficients.[12] As with ridge regression, adding a penalty can also provide a countermeasure against highly correlated predictors.

Recall that another method for regularizing linear regression models is to add a penalty based on the absolute values of the regression coefficients (similar to the lasso model of Sect. 6.4). The glmnet model (Friedman et al. 2010) uses a lasso-like penalty on the binomial (or multinomial) likelihood function. Like the lasso, this results in regression coefficients with values of absolute 0, thus simultaneously accomplishing regularization and feature selection at the same time. The glmnet models uses ridge and lasso penalties simultaneously, like the elastic net, but structures the penalty slightly differently:

$$\log L(p) - \lambda \left[(1-\alpha)\frac{1}{2} \sum_{j=1}^{P} \beta_j^2 + \alpha \sum_{j=1}^{P} |\beta_j| \right].$$

Here, the α value is the "mixing proportion" that toggles between the pure lasso penalty (when $\alpha = 1$) and a pure ridge-regression-like penalty ($\alpha = 0$). The other tuning parameter λ controls the total amount of penalization.

For the grant data, the glmnet model was tuned over seven values of the mixing parameter α and 40 values of the overall amount of penalization. The full set of predictors was used in the model. Figure 12.16 shows a heat map of the area under the ROC curve for these models. The data favor models with a larger mix of the ridge penalty than the lasso penalty, although there are many choices in this grid that are comparable. The bottom row of the heat map indicates that a complete ridge solution is poor regardless of the magnitude of the penalty. In the end, the numerically optimal settings are a mixing percentage of 0.1 and a value of 0.19 for the regularization amount. These settings had the effect of using only 44 predictors out of 1,070 in the final glmnet model, which achieved an area under the ROC curve of

[12] Another method for adding this penalty is discussed in the next chapter using neural networks . In this case, a neural network with weight decay and a single hidden unit constitutes a penalized logistic regression model. However, neural networks do not necessarily use the binomial likelihood when determining parameter estimates (see Sect. 13.2).

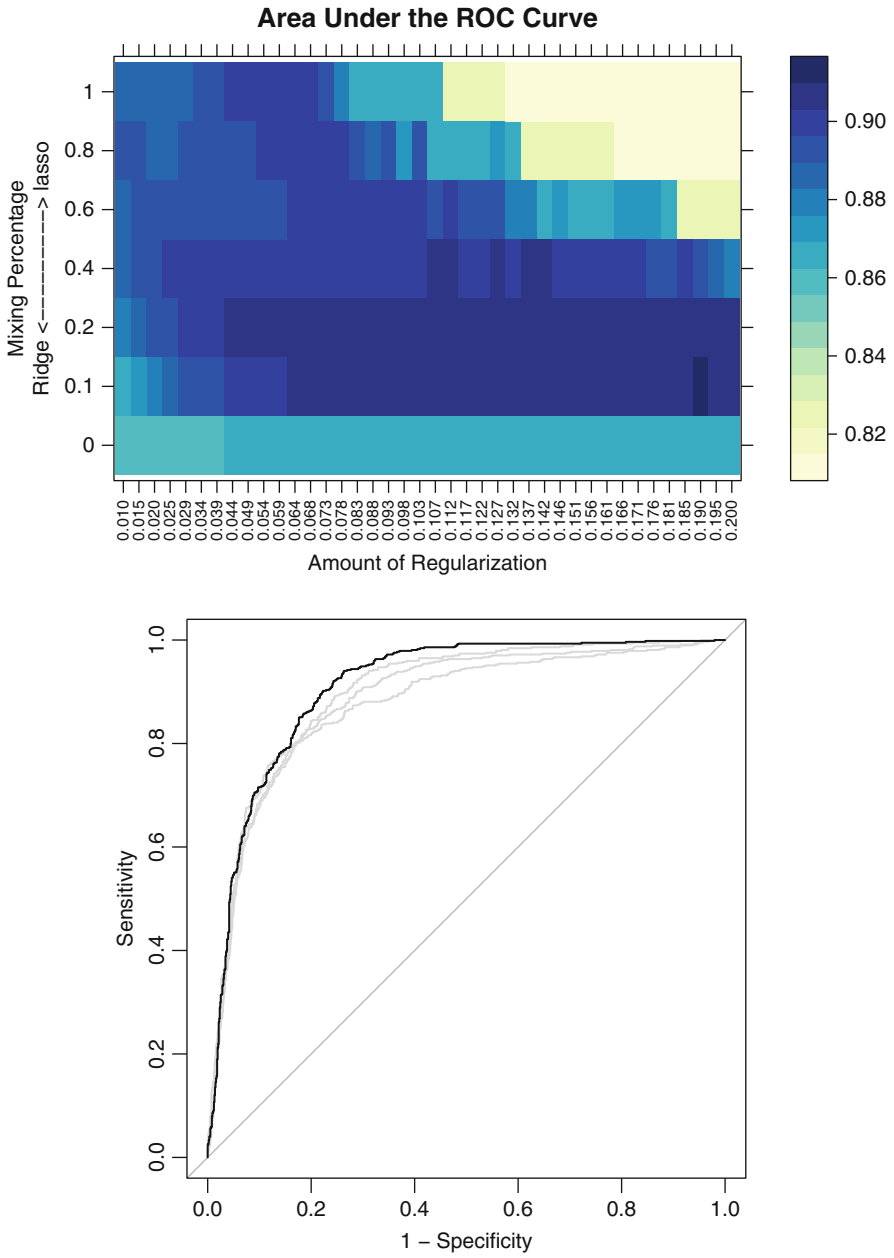

Fig. 12.16: *Top*: A heat map of the area under the ROC curve for the two glmnet tuning parameters. The numerically optimal settings are a mixing percentage of 0.1 and a value of 0.19 for the regularization amount. *Bottom*: The ROC curve for 2008 holdout data using glmnet the model (area under the ROC curve: 0.91)

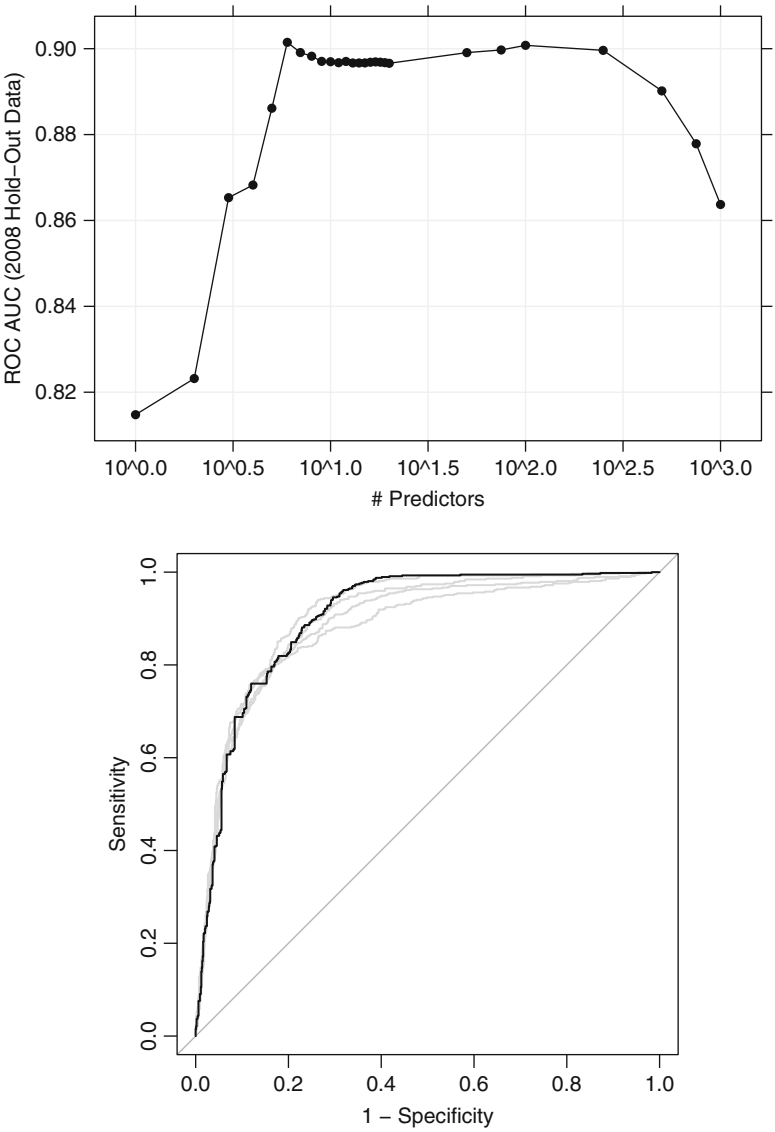

Fig. 12.17: *Top*: The tuning parameter profile for the sparse LDA model *Bottom*: The ROC curve for the model (AUC = 0.901)

0.91. The previous logistic regression model which used the reduced set of predictors resulted in an AUC of 0.87, indicating that the methodical removal of noninformative predictors increased the effectiveness of the model. Other approaches to *supervised* feature selection are discussed in Chap. 19.

Alternatively, penalization strategies can be applied to LDA models. For example, Clemmensen et al. (2011) use this technique with LDA models using

the *flexible discriminant analysis (FDA)* framework described in Sects. 13.1 and 13.3. In this model, an elastic-net strategy is used; L_1 penalties have the effect of eliminating predictors while an L_2 penalty shrinks the coefficients of the discriminant functions towards 0. Other approaches to penalizing LDA models are described by Witten and Tibshirani (2009) and Witten and Tibshirani (2011), which contain references to many earlier works. The same lasso penalty has also been applied to PLS discriminant models so that some of the PLS loadings are also eliminated (Chung and Keles 2010).

The penalized LDA model of Clemmensen et al. (2011) was applied to the grant data. The software for this model allows the user to specify the number of retained predictors as a tuning parameter (instead of the value of the L_1 penalty). The model was tuned over this parameter as well as the value of the L_2 penalty. Figure 12.17 shows the results for a single value of the ridge penalty (there was very little difference in performance across a range of values for this penalty). There is moderate performance when the number of predictors is close to the maximum. As the penalty increases and predictors are eliminated, performance improves and remains relatively constant until important factors are removed. At this point, performance falls dramatically. As a result of the tuning process, six predictors were used in the model which is competitive to other models (an AUC of 0.9).

12.6 Nearest Shrunken Centroids

The nearest-shrunken centroid model (also known as PAM, for predictive analysis for microarrays) is a linear classification model that is well suited for high-dimensional problems (Tibshirani et al. 2002, 2003; Guo et al. 2007). For each class, the centroid of the data is found by taking the average value of each predictor (per class) in the training set. The overall centroid is computed using the data from all of the classes.

If a predictor does not contain much information for a particular class, its centroid for that class is likely to be close to the overall centroid. Consider the three class data shown in the left panel of Fig. 12.18. This data set is the famous Fisher/Anderson iris data where four measurements of iris sepals and petals are used to classify flowers into one of three different iris species: setosa, versicolor, and virginica. In this plot, the data for the versicolor and virginica classes overlap but are well separated from the setosa irises. The centroids for the sepal width dimension are shown as grey symbols above the x-axis. The virginica centroid (for sepal width) is very close to the overall centroid, and the versicolor centroid is slightly closer to the overall centroid than the setosa flowers. This indicates that the sepal width predictor is most informative for distinguishing the setosa species from the other two. For sepal length (shown adjacent to the y-axis), the versicolor centroid is very close to the center of the data and the other two species are on the extremes.

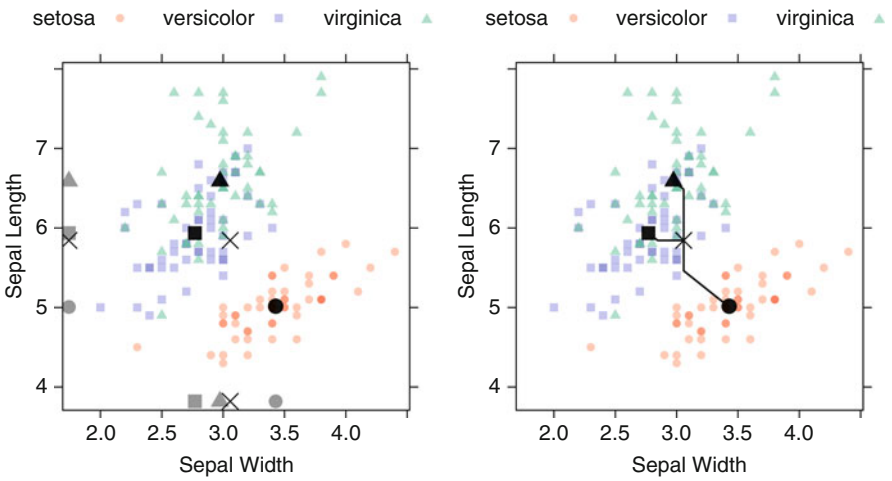

Fig. 12.18: *Left*: An example with three classes, their class-specific centroids (in *black*), and the overall centroid (×). The *grey symbols* along the axes are the centroids projected down into a single dimension. *Right*: The paths of the class centroids when shrunken to the center of the distributions

One approach to classifying unknown samples would be to find the closest class centroid in the full dimensional space and choose that class for prediction (i.e., a "nearest centroid" model). It turns out that this approach would result in linear class boundaries.

The approach taken by Tibshirani et al. (2002) is to shrink the class centroids closer to the overall centroid. In doing this, centroids that start off closer to the overall centroid move to that location before others. For example, in the sepal width dimension, the virginica centroid will reach the center before the other two. For this model, once the class centroid meets the overall centroid, it no longer influences the classification of samples for that class. Again, for sepal width, once the virginica centroid reaches the center, the sepal width can only be used to classify flowers that are versicolor or setosa. With enough shrinkage, it is possible for all the classes to be shrunken to the center. In the case that a predictor reaches the centroid, it has no effect on the model. Consequently, the nearest shrunken centroid model also conducts feature selection during the model training process.

The nearest shrunken centroid method has one tuning parameter: shrinkage. The right panel in Fig. 12.18 shows the path of the centroids over different shrinkage values. Note that each predictor moves diagonally towards the center until one of the class-specific centroids reaches the center. At this point, the classes move in a single dimension towards the center. Centering and scaling the predictors is recommended for this model.

This model works well for problems with a large number of predictors since it has built-in feature selection that is controlled by the shrinkage tuning parameter. Nearest shrunken centroids were originally developed for RNA profiling data, where the number of predictors is large (in the many thousands) and the number of samples is small. Most RNA profiling data sets have less than one or two hundred samples. In this *low n, high P* scenario, the data probably cannot support a highly nonlinear model and linear classification boundaries are a good choice. Also, the prior class probabilities along with the distances between the class centroids and the overall centroid can be used to produce class probabilities. Variable importance scores are calculated using the difference between the class centroids to the overall centroid (larger absolute values implying higher model importance).

For the grant data, the full set of 1,070 predictors was evaluated. The model was tuned over 30 values of the shrinkage parameter, ranging from 0 (implying very little shrinkage and feature selection) to 25 (Fig. 12.19). With large amounts of shrinkage, almost no predictors are retained and the area under the ROC curve is very poor. When the threshold is lowered to approximately 17, five predictors have been added: the number of unsuccessful grants by chief investigators, unknown sponsor, contract value band A, unknown contract value band, and submission month of January. The addition of these predictors clear has a large impact on the model fit. At the curve's apex (a shrinkage value of 2.59), important variables begin to be removed and under-fitting commences as the amount of shrinkage is increased. The sharp peak at a threshold of 8.6 is curious. The increase is associated with the removal of two predictors: sponsor code 2B and contract value band F. However, the next shrinkage value removes three additional predictors (contract value band D, contract value band E, and contract value band G) but this results in a appreciable drop in the area under the ROC curve. This spurious jump in performance is likely due to the fact that only a single holdout is used to measure performance. The true relationship between performance and shrinkage is likely to be more smooth than is demonstrated by this figure. At the best threshold, the area under the ROC curve was 0.87 using 36 predictors. The sensitivity was 83.7 % and the specificity was 77 % for the year 2008 holdout data. The ROC curve is also shown in Fig. 12.19.

12.7 Computing

This section discussed the following R packages: AppliedPredictiveModeling, caret, glmnet, MASS, pamr, pls, pROC, rms, sparseLDA, and subselect.

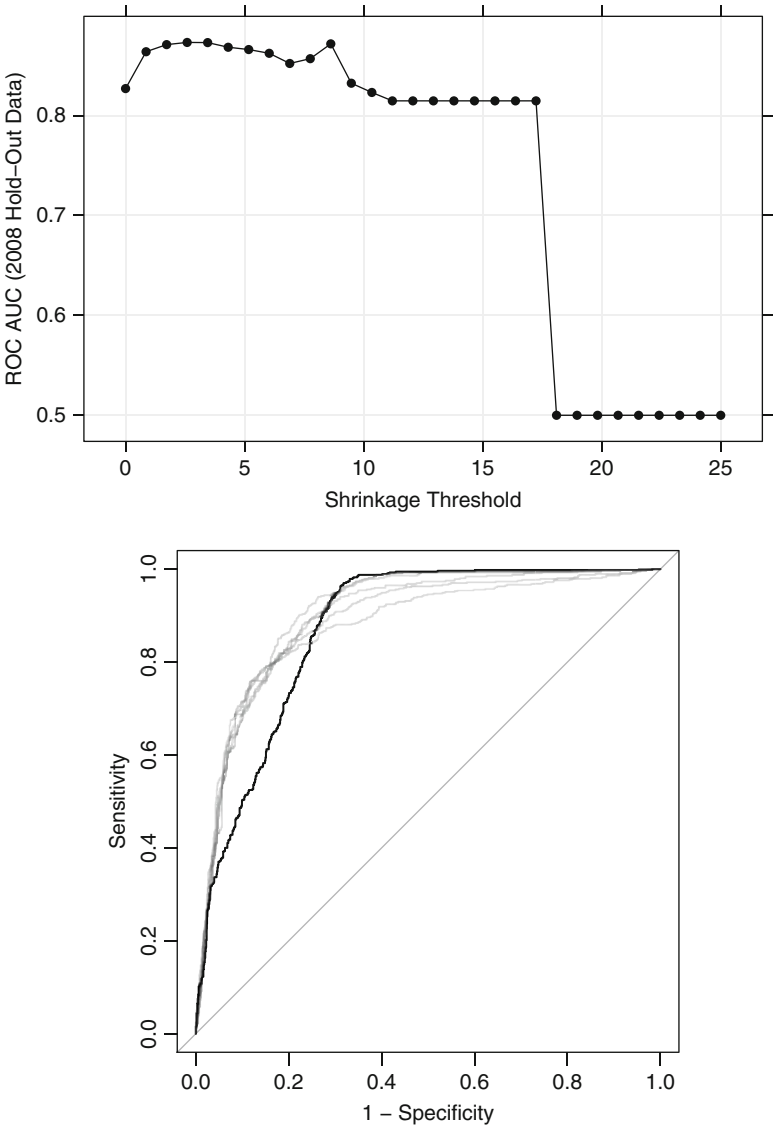

Fig. 12.19: *Top*: The tuning parameter profile for the nearest shrunken centroid model. *Bottom*: The ROC curve for the model (AUC = 0.873)

The grant application data can be found at the Kaggle web site.[13] The R package AppliedPredictiveModeling contains scripts that can be used to reproduce the objects and analyses given here.

[13] http://www.kaggle.com/c/unimelb.

Following the data splitting approach described in the first section, there are two data frames with grant data: `training` contains the pre-2008 data and 2008 holdout set used to tune the model while the data frame `testing` has only year 2008 grant data and is not used until a later chapter. A vector called `pre2008` has the row indices of the 6,633 training set grants prior to 2008 (see Table 12.2 for a summary of the data splitting strategy).

Most of the predictors in this set are binary. For example the RFCD codes, SEO codes, sponsors, and contract value band categories are contained in individual binary values with identifying prefixes, such as `RFCD` or `ContractValueBand`.[14] When the value was unknown, a specific dummy variable is created for this situation, such as `SponsorUnk`. Binary dummy variables also exist for the submission month and day of the week.

In addition, there exist count and continuous predictors such as the frequencies of each role associated with the grant. For example, `NumCI` and `NumEA` are the number of chief investigators and external advisors on the grant, respectively. The number of persons with unspecified roles is captured in the predictor `numUnk`. Similar count predictors are also in the data, such as the number of people born within a time frame (e.g., `CI.1925` for chief investigators born between 1925 and 1930), the number born in a certain region (e.g., `HV.Australia`), and their degree status (e.g., `ECI.PhD`). The number of previously successful and unsuccessful grants are enumerated with the predictors `Unsuccess.PS` or `Success.CI`. The publication information is represented in two ways. First, the totals for each role, such as `B.CI` or `Astar.CI`, are available as well as the total counts across all the individuals (`AstarTotal`) or all journal types (`allPub`).

The calendar day of the year is stored as a numeric variable.

Finally, the class outcome is contained in a column called `Class` with levels successful and unsuccessful.

As was illustrated in the regression sections of this book, different models have different constraints on the types of data that can be used. As previously discussed, two general groupings of predictors were created: the set of predictors that contain the full set of binary dummy variables and count data and the reduced set that was filtered for near-zero variance predictors and extremely correlated predictors. For example, the columns `AstarTotal`, `ATotal`, `BTotal`, and `CTotal` all add up to the column `allPub`. In the reduced set, `allPub` was removed. Similarly, one dummy variable for month and one for a day of the week should also be removed from the reduced set. The two columns with the lowest frequencies, `Mar` and `Sun`, were eliminated from the reduced set.

[14] In a later chapter, several models are discussed that can represent the categorical predictors in different ways. For example, trees can use the dummy variables in splits but can often create splits based on one or more groupings of categories. In that chapter, factor versions of these predictors are discussed at length.

Two character vectors were created for the purpose of specifying either group: `fullSet` and `reducedSet`:

```
> length(fullSet)
  [1] 1070
> head(fullSet)
  [1] "NumCI"  "NumDR"  "NumEA"  "NumECI" "NumHV"  "NumPS"

> length(reducedSet)
  [1] 252
> head(reducedSet)
  [1] "NumCI"  "NumDR"  "NumECI" "NumPS"  "NumSR"  "NumSCI"
```

How can extreme collinearity problems (such as linear combinations) be diagnosed? The `trim.matrix` function in subselect takes a square, symmetric matrix (such as the covariance matrix) and uses an algorithm to eliminate linear combinations. For example, the reduced set has no such issues:

```
> reducedCovMat <- cov(training[, reducedSet])
> library(subselect)
> trimmingResults <- trim.matrix(reducedCovMat)
> names(trimmingResults)
  [1] "trimmedmat"        "numbers.discarded" "names.discarded"
  [4] "size"
> ## See if any predictors were eliminated:
> trimmingResults$names.discarded
  character(0)
```

However, when we apply the same function to the full set, several predictors are identified:

```
> fullCovMat <- cov(training[, fullSet])
> fullSetResults <- trim.matrix(fullCovMat)

> ## A different choices for the day to exclude was
> ## made by this function
> fullSetResults$names.discarded
  [1] "NumDR"       "PS.1955"     "CI.Dept1798" "PS.Dept3268" "PS.Faculty1"
  [6] "DurationUnk" "ATotal"      "Nov"         "Sun"
```

Another function in the caret package called `findLinearCombos` follows a similar methodology but does not require a square matrix.

When developing models, `train` is used to tune parameters on the basis of the ROC curve. To do this, a control function is needed to obtain the results of interest. The caret function `trainControl` is used for this purpose. First, to compute the area under the ROC curve, the class probabilities must be generated. By default, `train` only generates class predictions. The option `classProbs` can be specified when probabilities are needed. Also by default,

overall accuracy and the Kappa statistic are used to evaluate the model. caret contains a built-in function called `twoClassSummary` that calculates the area under the ROC curve, the sensitivity, and the specificity. To achieve these goals, the syntax would be:

```
> ctrl <- trainControl(summaryFunction = twoClassSummary,
+                      classProbs = TRUE)
```

However, at the start of the chapter, a data splitting scheme was developed that built the model on the pre-2008 data and then used the 2008 holdout data (in the training set) to tune the model. To do this, `train` must know exactly which samples to use when estimating parameters. The `index` argument to `trainControl` identifies these samples. For any resampling method, a set of holdout samples can be exactly specified. For example, with 10-fold cross-validation, the exact samples to be excluded for each of the 10-folds are identified with this option. In this case, `index` identifies the rows that correspond to the pre-2008 data. The exact syntax should package these row numbers in a list (in case there is more than one holdout). Recall that the vector `pre2008` contains the locations of the grants submitted prior to 2008. The call to `trainControl` is:

```
ctrl <- trainControl(method = "LGOCV",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     index = list(TrainSet = pre2008))
```

Note that, once the tuning parameters have been chosen using year 2008 performance estimates, the final model is fit with all the grants in the training set, including those from 2008.

Finally, for illustrative purposes, we need to save the predictions of the year 2008 grants based on the pre-2008 model (i.e., before the final model is re-fit with all of the training data). The `savePredictions` argument accomplishes this goal:

```
ctrl <- trainControl(method = "LGOCV",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     index = list(TrainSet = pre2008),
                     savePredictions = TRUE)
```

Since many of the models described in this text use random numbers, the seed for the random number generator is set prior to running each model so that the computations can be reproduced. A seed value of 476 was randomly chosen for this chapter.

Logistic Regression

The `glm` function (for GLMs) in base R is commonly used to fit logistic regression models. The syntax is similar to previous modeling functions that

work from the formula method. For example, to fit the model shown in the left panel of Fig. 12.3 for the pre-2008 data:

```
> levels(training$Class)
  [1] "successful"   "unsuccessful"
> modelFit <- glm(Class ~ Day,
+                 ## Select the rows for the pre-2008 data:
+                 data = training[pre2008,],
+                 ## 'family' relates to the distribution of the data.
+                 ## A value of 'binomial' is used for logistic regression
+                 family = binomial)
> modelFit
  Call:  glm(formula = Class ~ Day, family = binomial, data = training[pre2008,
      ])

  Coefficients:
  (Intercept)          Day
     -0.91934      0.00424

  Degrees of Freedom: 6632 Total (i.e. Null);  6631 Residual
  Null Deviance:            9190
  Residual Deviance: 8920         AIC: 8920
```

The `glm` function treats the *second* factor level as the event of interest. Since the slope is positive for the day of the year, it indicates an increase in the rate of unsuccessful grants. To get the probability of a successful grant, we subtract from one:

```
> successProb <- 1 - predict(modelFit,
+                            ## Predict for several days
+                            newdata = data.frame(Day = c(10, 150, 300,
                                                         350)),
+                            ## glm does not predict the class, but can
+                            ## produce the probability of the event
+                            type = "response")
> successProb
       1       2       3       4
  0.70619 0.57043 0.41287 0.36262
```

To add the nonlinear term for the day of the year, the previous formula is augmented as follows:

```
> daySquaredModel <- glm(Class ~ Day + I(Day^2),
+                        data = training[pre2008,],
+                        family = binomial)
> daySquaredModel
  Call:  glm(formula = Class ~ Day + I(Day^2), family = binomial,
     data = training[pre2008,
      ])

  Coefficients:
  (Intercept)          Day     I(Day^2)
    -1.881341     0.018622    -0.000038
```

```
Degrees of Freedom: 6632 Total (i.e. Null);  6630 Residual
Null Deviance:            9190
Residual Deviance: 8720         AIC: 8730
```

The `glm` function does not have a non-formula method, so creating models with a large number of predictors takes a little more work. An alternate solution is shown below.

Another R function for logistic model is in the package associated with Harrell (2001), called rms (for Regression Modeling Strategies). The `lrm` function is very similar to `glm` and includes helper functions. For example, a *restricted cubic spline* is a tool for fitting flexible nonlinear functions of a predictor. For the day of the year:

```
> library(rms)
> rcsFit <- lrm(Class ~ rcs(Day), data = training[pre2008,])
> rcsFit
```

```
Logistic Regression Model

lrm(formula = Class ~ rcs(Day), data = training[pre2008, ])

                       Model Likelihood     Discrimination    Rank Discrim.
                          Ratio Test            Indexes          Indexes
Obs           6633    LR chi2     461.53    R2       0.090    C       0.614
 successful   3233    d.f.             4    g        0.538    Dxy     0.229
 unsuccessful 3400    Pr(> chi2) <0.0001    gr       1.713    gamma   0.242
max |deriv|  2e-06                         gp       0.122    tau-a   0.114
                                           Brier    0.234

          Coef    S.E.   Wald Z Pr(>|Z|)
Intercept -1.6833 0.1110 -15.16 <0.0001
Day        0.0124 0.0013   9.24 <0.0001
Day'      -0.0072 0.0023  -3.17 0.0015
Day''      0.0193 0.0367   0.52 0.6001
Day'''    -0.0888 0.1026  -0.87 0.3866
```

The `lrm` function, like `glm`, models the probability of the second factor level. The bottom table in the output shows p-values for the different nonlinear components of the restricted cubic spline. Since the p-values for the first three nonlinear components are small, this indicates that a nonlinear relationship between the class and day should be used. The package contains another function, `Predict`, which quickly create a prediction profile across one or more variables. For example, the code

```
> dayProfile <- Predict(rcsFit,
+                       ## Specify the range of the plot variable
+                       Day = 0:365,
+                       ## Flip the prediction to get the model for
+                       ## successful grants
+                       fun = function(x) -x)
> plot(dayProfile, ylab = "Log Odds")
```

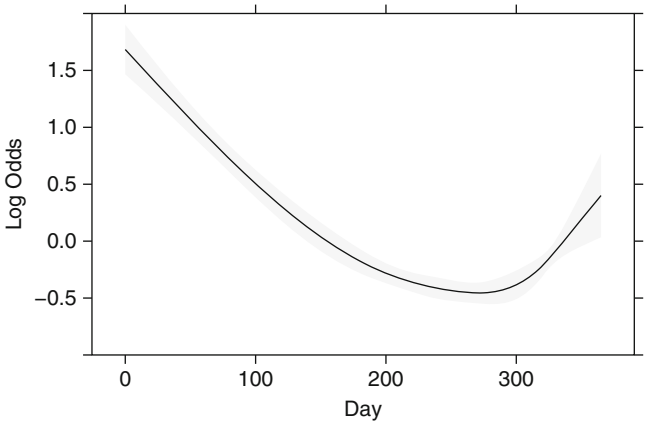

Fig. 12.20: A restricted cubic spline fit for the day of the year produce generated by the rms package. The *grey bands* are confidence limits on the log odds

produces the image in Fig. 12.20. The `fun` argument changes the signs of the prediction so that the plot reflects the probability of a successful grant. From this plot, it is apparent that a quadratic term for the day of the year would approximate the trends shown by the spline.

The rms package contains many more relevant functions, including resampling techniques for model validation and model visualization functions. See Harrell (2001) for details of the methodologies and R code.

For a large set of predictors, the formula method for specifying models can be cumbersome. As in previous chapters, the `train` function can efficiently fit and validate models. For logistic regression, `train` provides an interface to the `glm` function that bypasses a model formula, directly produces class predictions, and calculates the area under the ROC curve and other metrics.

Prior to fitting the model, we augment the data set and predictor groups with the squared day variable:

```
> training$Day2 <- training$Day^2
> fullSet <- c(fullSet, "Day2")
> reducedSet <- c(reducedSet, "Day2")
```

For the grant data, the code that fits a model with the full predictor set is:

```
> library(caret)
> set.seed(476)
> lrFull <- train(training[,fullSet],
+                 y = training$Class,
+                 method = "glm",
+                 metric = "ROC",
+                 trControl = ctrl)
```

```
> lrFull
```

```
8190 samples
1071 predictors
   2 classes: 'successful', 'unsuccessful'

No pre-processing
Resampling: Repeated Train/Test Splits (1 reps, 0.75%)

Summary of sample sizes: 6633

Resampling results

  ROC   Sens  Spec
  0.78  0.77  0.76
```

Note that the top of this output reflects that 8,190 grants were used, but the "`Summary of sample sizes`" lists a value of 6,633 data points. This latter number reflects the single set of pre-2008 samples (see Table 12.2). The "`Resampling Results`" is actually the performance estimate of the 2008 holdout set.

To create a model with the smaller predictor set:

```
> set.seed(476)
> lrReduced <- train(training[,reducedSet],
+                    y = training$Class,
+                    method = "glm",
+                    metric = "ROC",
+                    trControl = ctrl)
```

```
> lrReduced
```

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

No pre-processing
Resampling: Repeated Train/Test Splits (1 reps, 0.75%)

Summary of sample sizes: 6633

Resampling results

  ROC   Sens  Spec
  0.87  0.8   0.82
```

Like the LDA analysis, removal of the near-zero variance predictors has a positive effect on the model fit. The predictions for the holdout set (of year 2008 grants) is contained in the sub-object `pred`:

```
> head(lrReduced$pred)
            pred        obs successful unsuccessful rowIndex .parameter
6634 successful successful    0.99878    0.0012238     6634        none
6635 successful successful    0.85151    0.1484924     6635        none
6636 successful successful    0.92019    0.0798068     6636        none
6637 successful successful    0.96694    0.0330572     6637        none
6639 successful successful    0.98928    0.0107160     6638        none
6642 successful successful    0.57563    0.4243729     6639        none
     Resample
6634 TrainSet
6635 TrainSet
6636 TrainSet
6637 TrainSet
6639 TrainSet
6642 TrainSet
```

Note the column in the output labeled `.parameter`. When `train` saves predictions, it does so for every tuning parameter. This column in the output is used to label which model generated the predictions. This version of logistic regression has no tuning parameters, so `.parameter` has a single value (`"none"`).

From these data, the confusion matrix can be computed:

```
> confusionMatrix(data = lrReduced$pred$pred,
+                 reference = lrReduced$pred$obs)
Confusion Matrix and Statistics

              Reference
Prediction     successful unsuccessful
  successful          458          176
  unsuccessful        112          811

               Accuracy : 0.815
                 95% CI : (0.795, 0.834)
    No Information Rate : 0.634
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.611
 Mcnemar's Test P-Value : 0.000205

            Sensitivity : 0.804
            Specificity : 0.822
         Pos Pred Value : 0.722
         Neg Pred Value : 0.879
             Prevalence : 0.366
         Detection Rate : 0.294
   Detection Prevalence : 0.407

       'Positive' Class : successful
```

These results match the values shown above for `lrReduced`. The ROC curve can also be computed and plotted using the pROC package:

```
> reducedRoc <- roc(response = lrReduced$pred$obs,
+                   predictor = lrReduced$pred$successful,
+                   levels = rev(levels(lrReduced$pred$obs)))
> plot(reducedRoc, legacy.axes = TRUE)
```



```
> auc(reducedRoc)

  Area under the curve: 0.872
```

Linear Discriminant Analysis

A popular function for creating LDA models is `lda` in the MASS package. The input to this function can either be a formula and data frame or a matrix of predictors and a grouping variable as a factor which contains the class membership information. We can fit the LDA model as follows:

```
> library(MASS)
> ## First, center and scale the data
> grantPreProcess <- preProcess(training[pre2008, reducedSet])
> grantPreProcess

  Call:
  preProcess.default(x = training[pre2008, reducedSet])

  Created from 6,633 samples and 253 variables
  Pre-processing: centered, scaled
> scaledPre2008 <- predict(grantPreProcess,
+                          newdata = training[pre2008, reducedSet])
> scaled2008HoldOut <- predict(grantPreProcess,
+                              newdata = training[-pre2008, reducedSet])


> ldaModel <- lda(x = scaledPre2008,
+                 grouping = training$Class[pre2008])
```

Recall that because these data involve two classes, only one discriminant vector can be obtained. This discriminant vector is contained in the object `ldaModel$scaling`; the first six entries of this matrix are:

```
> head(ldaModel$scaling)

               LD1
  NumCI   0.1301673
  NumDR   0.0017275
  NumECI  0.1219478
  NumPS   0.0042669
  NumSR  -0.0642209
  NumSCI -0.0655663
```

This information provides an interpretation about the predictors, relationships among predictors, and, if the data have been centered and scaled, then relative importance values. The discriminant vector is involved in the prediction of samples, and the `MASS` package simplifies this process through the `predict` function. For the grant data test set, the predictions are produced with the syntax:

```
> ldaHoldOutPredictions <- predict(ldaModel, scaled2008HoldOut)
```

The predicted class, posterior probability, and linear discriminant value are all contained in this object, thus enabling the user to create (1) a confusion matrix of the observed versus predicted values, (2) the distribution of posterior probabilities, and/or (3) the distribution of linear discriminant values.

A direct implication of the two-class setting is that there is no training over the number of discriminant vectors to retain for prediction. When working with data that contain more than two classes, the optimal number of linear discriminant vectors can be determined through the usual validation process. Through the `lda` function, the number of linear discriminants to retain for prediction can be set with the `dimen` option of the `predict` function. Conveniently, this optimization process is automated with the `train` function in the `caret` package:

```
> set.seed(476)
> ldaFit1 <- train(x = training[, reducedSet],
+                  y = training$Class,
+                  method = "lda",
+                  preProc = c("center","scale"),
+                  metric = "ROC",
+                  ## Defined above
+                  trControl = ctrl)


> ldaFit1
  8190 samples
   253 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

    ROC   Sens  Spec
    0.89  0.8   0.82
```

No formal tuning occurs because there are only two classes and thus only one discriminant vector. We can generate predicted classes and probabilities for the test set in the usual manner:

```
> ldaTestClasses <- predict(ldaFit1,
+                           newdata = testing[,reducedSet])
> ldaTestProbs <- predict(ldaFit1,
+                         newdata = testing[,reducedSet],
+                         type = "prob")
```

When the problem involves more than two classes and we desire to optimize over the number of discriminant vectors, then the `train` function can still be used with `method` set to `"lda2"` and `tuneLength` set to the maximum number of dimensions that the practitioner desires to evaluate.

Partial Least Squares Discriminant Analysis

PLSDA can be performed using the `plsr` function within the pls package by using a categorical matrix which defines the response categories. We refer the reader to Sect. 6.3 for a description of the algorithmic variations of PLS, which directly extend to the classification setting.

The caret package contains a function (`plsda`) that can create the appropriate dummy variable PLS model for the data and then post-process the raw model predictions to return class probabilities. The syntax is very similar to the regression model code for PLS given in Sect. 6.3. The main difference is a factor variable is used for the outcome.

For example, to fit the model with the reduced predictor set:

```
> plsdaModel <- plsda(x = training[pre2008,reducedSet],
+                     y = training[pre2008, "Class"],
+                     ## The data should be on the same scale for PLS. The
+                     ## 'scale' option applies this pre-processing step
+                     scale = TRUE,
+                     ## Use Bayes method to compute the probabilities
+                     probMethod = "Bayes",
+                     ## Specify the number of components to model
+                     ncomp = 4)
> ## Predict the 2008 hold-out set
> plsPred <- predict(plsdaModel,
+                    newdata = training[-pre2008, reducedSet])
> head(plsPred)
  [1] successful successful successful successful successful successful
  Levels: successful unsuccessful
> plsProbs <- predict(plsdaModel,
+                     newdata = training[-pre2008, reducedSet],
+                     type = "prob")
> head(plsProbs)
  [1] 0.98842 0.88724 0.83455 0.88144 0.94848 0.53991
```

The `plsdaModel` object inherits all of the same functions that would have resulted from the object coming directly from the `plsr` function. Because of this, other functions from the pls package can be used, such as `loadings` or `scoreplot`.

The `train` function can also be used with PLS in the classification setting. The following code evaluates the first ten PLS components with respect to the area under the ROC curve as well as automatically centers and scales the predictors prior to model fitting and sample prediction:

```
> set.seed(476)
> plsFit2 <- train(x = training[, reducedSet],
+                  y = training$Class,
+                  method = "pls",
+                  tuneGrid = expand.grid(.ncomp = 1:10),
+                  preProc = c("center","scale"),
+                  metric = "ROC",
+                  trControl = ctrl)
```

The basic `predict` call evaluates new samples, and `type = "prob"` returns the class probabilities. Computing variable importance as illustrated in Fig. 12.15 can be done with the following code:

```
> plsImpGrant <- varImp(plsFit2, scale = FALSE)
> plsImpGrant
pls variable importance

  only 20 most important variables shown (out of 253)

                      Overall
ContractValueBandUnk  0.0662
SponsorUnk            0.0383
Jan                   0.0338
Unsuccess.CI          0.0329
ContractValueBandA    0.0316
Day                   0.0266
Aug                   0.0257
Success.CI            0.0219
GrantCat10A           0.0211
Day2                  0.0209
GrantCat30B           0.0202
ContractValueBandE    0.0199
ContractValueBandD    0.0193
ContractValueBandF    0.0188
ContractValueBandG    0.0184
Sponsor24D            0.0172
Sponsor21A            0.0169
Sponsor2B             0.0147
NumSR                 0.0144
Jul                   0.0124
> plot(plsImpGrant, top = 20, scales = list(y = list(cex = .95)))
```

Penalized Models

The primary package for penalized logistic regression is glmnet (although the next chapter describes how to fit similar models using neural networks). The `glmnet` function is very similar to the `enet` function described previously in Sect. 6.5. The main arguments correspond to the data: `x` is a *matrix* of predictors and `y` is a factor of classes (for logistic regression). Additionally, the `family` argument is related to the distribution of the outcome. For two classes, using `family="binomial"` corresponds to logistic regression, and, when there are three or more classes, `family="multinomial"` is appropriate.

The function will automatically select a sequence of values for the amount of regularization, although the user can select their own values with the `lambda` option. Recall that the type of regularization is determined by the mixing parameter α. `glmnet` defaults this parameter to `alpha = 1`, corresponding to a complete lasso penalty.

The `predict` function for `glmnet` predicts different types of values, including: the predicted class, which predictors are used in the model, and/or the regression parameter estimates. For example:

```
> library(glmnet)
> glmnetModel <- glmnet(x = as.matrix(training[,fullSet]),
+                       y = training$Class,
+                       family = "binomial")


> ## Compute predictions for three difference levels of regularization.
> ## Note that the results are not factors
> predict(glmnetModel,
+         newx = as.matrix(training[1:5,fullSet]),
+         s = c(0.05, 0.1, 0.2),
+         type = "class")
     1            2            3
1 "successful" "successful" "unsuccessful"
2 "successful" "successful" "unsuccessful"
3 "successful" "successful" "unsuccessful"
4 "successful" "successful" "unsuccessful"
5 "successful" "successful" "unsuccessful"
> ## Which predictors were used in the model?
> predict(glmnetModel,
+         newx = as.matrix(training[1:5,fullSet]),
+         s = c(0.05, 0.1, 0.2),
+         type = "nonzero")
$`1`
[1]   71   72  973 1027 1040 1045 1055

$`2`
[1] 1027 1040

$`3`
[1] 1040
```

As a side note, the glmnet package has a function named `auc`. If the pROC package is loaded prior to loading glmnet, this message will appear: "`The following object(s) are masked from 'package:pROC': auc.`" If this function is invoked at this point, R will be unclear on which to use. There are two different approaches to dealing with this issue:

- If one of the packages is no longer needed, it can be detached using `detach(package:pROC)`.
- The appropriate function can be called using the *namespace* convention when invoking the function. For example, `pROC:::auc` and `glmnet:::auc` would reference the specific functions.

Another potential instance of this issue is described below.

Tuning the model using the area under the ROC curve can be accomplished with `train`. For the grant data:

```
> ## Specify the tuning values:
> glmnGrid <- expand.grid(.alpha = c(0,  .1,  .2, .4, .6, .8, 1),
+                         .lambda = seq(.01, .2, length = 40))
> set.seed(476)
> glmnTuned <- train(training[,fullSet],
+                    y = training$Class,
+                    method = "glmnet",
+                    tuneGrid = glmnGrid,
+                    preProc = c("center", "scale"),
+                    metric = "ROC",
+                    trControl = ctrl)
```

The heat map in the top panel of Fig. 12.16 was produced using the code `plot(glmnTuned, plotType = "level")`. Penalized LDA functions can be found in the sparseLDA and PenalizedLDA packages. The main function in the sparseLDA package is called `sda`.[15] This function has an argument for the ridge parameter called `lambda`. The lasso penalty can be stated in two possible ways with the argument `stop`. The magnitude of the lasso penalty is controlled using a positive number (e.g., `stop = 0.01`) or, alternatively, the number of retained predictors can be chosen using a *negative* integer (e.g., `stop = -6` for six predictors). For example:

```
> library(sparseLDA)
> sparseLdaModel <- sda(x = as.matrix(training[,fullSet]),
+                       y = training$Class,
+                       lambda = 0.01,
+                       stop = -6)
```

The argument `method = "sparseLDA"` can be used with `train`. In this case, `train` will tune the model over `lambda` and the number of retained predictors.

[15] Another duplicate naming issue may occur here. A function called `sda` in the sda package (for shrinkage discriminant analysis) may cause confusion. If both packages are loaded, using `sparseLDA:::sda` and `sda:::sda` will mitigate the issue.

Nearest Shrunken Centroids

The original R implementation for this model is found in the pamr package (for "*P*redictive *A*nalysis of *M*icroarrays in R"). Another package, rda, contains extensions to the model described in Guo et al. (2007).

The syntax of the functions in the pamr package is somewhat nonstandard. The function to train the model is `pamr.train`, which takes the input data in a single list object with components `x` and `y`. The usual convention for data sets is to have samples in rows and different columns for the predictors. `pamr.train` requires the training set predictors to be encoded in the opposite format where rows are predictors and columns are samples.[16] For the grant data, the input data would be in the format shown below:

```
> ## Switch dimensions using the t() function to transpose the data.
> ## This also implicitly converts the training data frame to a matrix.
> inputData <- list(x = t(training[, fullSet]), y = training$Class)
```

The basic syntax to create the model is:

```
> library(pamr)
> nscModel <- pamr.train(data = inputData)
```

By default, the function chooses 30 appropriate shrinkage values to evaluate. There are options to use specific values for the shrinkage amount, the prior probabilities and other aspects of the model. The function `pamr.predict` generates predictions on new samples as well as determines which specific predictors were used in the model for a given shrinkage value. For example, to specify a shrinkage value of 5:

```
> exampleData <- t(training[1:5, fullSet])
> pamr.predict(nscModel, newx = exampleData, threshold = 5)
  [1] successful   unsuccessful successful   unsuccessful successful
  Levels: successful unsuccessful
> ## Which predictors were used at this threshold? The predict
> ## function shows the column numbers for the retained predictors.
> thresh17Vars <- pamr.predict(nscModel, newx = exampleData,
+                              threshold = 17, type = "nonzero")
> fullSet[thresh17Vars]
  [1] "Unsuccess.CI"         "SponsorUnk"           "ContractValueBandA"
  [4] "ContractValueBandUnk" "Jan"
```

The package also contains functions for K-fold cross-validation to choose an appropriate amount of shrinkage but is restricted to a single type of resampling and tunes the model with overall accuracy. The `train` syntax is:

[16] In microarray data, the number of predictors is usually much larger than the number of samples. Because of this, and the limited number of columns in popular spreadsheet software, the convention is reversed.

```
>  ## We chose the specific range of tuning parameters here:
> nscGrid <- data.frame(.threshold = 0:25)
> set.seed(476)
> nscTuned <- train(x = training[,fullSet],
+                   y = training$Class,
+                   method = "pam",
+                   preProc = c("center", "scale"),
+                   tuneGrid = nscGrid,
+                   metric = "ROC",
+                   trControl = ctrl)
```

This approach provides more options for model tuning (e.g., using the area under the ROC curve) as well as a consistent syntax. The `predict` function for `train` does not require the user to manually specify the shrinkage amount (the optimal value determined by the function is automatically used).

The `predictors` function will list the predictors used in the prediction equation (at the optimal threshold determined by `train`). In the tuned model, 36 were selected:

```
> predictors(nscTuned)
```

```
 [1] "NumSR"                "Success.CI"           "Unsuccess.CI"
 [4] "CI.Faculty13"         "CI.Faculty25"         "CI.Faculty58"
 [7] "DurationGT15"         "Astar.CI"             "AstarTotal"
[10] "allPub"               "Sponsor21A"           "Sponsor24D"
[13] "Sponsor2B"            "Sponsor34B"           "Sponsor4D"
[16] "Sponsor62B"           "Sponsor6B"            "Sponsor89A"
[19] "SponsorUnk"           "ContractValueBandA"   "ContractValueBandC"
[22] "ContractValueBandD"   "ContractValueBandE"   "ContractValueBandF"
[25] "ContractValueBandG"   "ContractValueBandUnk" "GrantCat10A"
[28] "GrantCat30B"          "Aug"                  "Dec"
[31] "Jan"                  "Jul"                  "Fri"
[34] "Sun"                  "Day"                  "Day2"
```

Also, the function `varImp` will return the variable importance based on the distance between the class centroid and the overall centroid:

```
> varImp(nscTuned, scale = FALSE)
```

```
pam variable importance

  only 20 most important variables shown (out of 1071)

                     Importance
ContractValueBandUnk    -0.2260
SponsorUnk               0.1061
Jan                      0.0979
ContractValueBandA       0.0948
Unsuccess.CI            -0.0787
Day                     -0.0691
Aug                     -0.0669
Sun                      0.0660
GrantCat10A             -0.0501
Success.CI               0.0413
```

```
Day2                   -0.0397
ContractValueBandE      0.0380
GrantCat30B            -0.0379
ContractValueBandD      0.0344
Sponsor21A              0.0340
ContractValueBandF      0.0333
ContractValueBandG      0.0329
Sponsor24D             -0.0299
Sponsor2B              -0.0233
NumSR                   0.0224
```

In these data, the sign of the difference indicates the direction of the impact of the predictor. For example, when the contractor band is unknown, only a small percentage of grants are successful (19.4 % versus the baseline success rate of 46.4 %). The negative sign for this predictor indicates a drop in the event rate. Conversely, when the sponsor is unknown, the success rate is high (82.2 %; see Table 12.1). The distance for this predictor is positive, indicating an increase in the event rate.

Exercises

12.1. The hepatic injury data set was described in the introductory chapter and contains 281 unique compounds, each of which has been classified as causing no liver damage, mild damage, or severe damage (Fig. 1.2). These compounds were analyzed with 184 biological screens (i.e., experiments) to assess each compound's effect on a particular biologically relevant target in the body. The larger the value of each of these predictors, the higher the activity of the compound. In addition to biological screens, 192 chemical fingerprint predictors were determined for these compounds. Each of these predictors represent a substructure (i.e., an atom or combination of atoms within the compound) and are either counts of the number of substructures or an indicator of presence or absence of the particular substructure. The objective of this data set is to build a predictive model for hepatic injury so that other compounds can be screened for the likelihood of causing hepatic injury. Start R and use these commands to load the data:

```
> library(caret)
> data(AppliedPredictiveModeling)
> # use ?hepatic to see more details
```

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) Given the classification imbalance in hepatic injury status, describe how you would create a training and testing set.

(b) Which classification statistic would you choose to optimize for this exercise and why?
(c) Split the data into a training and a testing set, pre-process the data, and 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? Based on these results, which set of predictors contains the most information about hepatic toxicity?
(d) For the optimal models for both the biological and chemical predictors, what are the top five important predictors?
(e) Now combine the biological and chemical fingerprint predictors into one predictor set. Retrain the same set of predictive models you built from part (c). Which model yields best predictive performance? Is the model performance better than either of the best models from part (c)? What are the top five important predictors for the optimal model? How do these compare with the optimal predictors from each individual predictor set?
(f) 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.

12.2. In Exercise 4.4, we described a data set which contained 96 oil samples each from one of seven types of oils (pumpkin, sunflower, peanut, olive, soybean, rapeseed, and corn). Gas chromatography was performed on each sample and the percentage of each type of 7 fatty acids was determined. We would like to use these data to build a model that predicts the type of oil based on a sample's fatty acid percentages.

(a) Like the hepatic injury data, these data suffer from extreme imbalance. Given this imbalance, should the data be split into training and test sets?
(b) Which classification statistic would you choose to optimize for this exercise and why?
(c) Of the models presented in this chapter, which performs best on these data? Which oil type does the model most accurately predict? Least accurately predict?

12.3. The web site[17] for the MLC++ software package contains a number of machine learning data sets. The "churn" data set was developed to predict telecom customer churn based on information about their account. The data files state that the data are "artificial based on claims similar to real world."

The data consist of 19 predictors related to the customer account, such as the number of customer service calls, the area code, and the number of minutes. The outcome is whether the customer churned.

[17] http://www.sgi.com/tech/mlc.

The data are contained in the C50 package and can be loaded using:

```
> library(C50)
> data(churn)
> ## Two objects are loaded: churnTrain and churnTest
> str(churnTrain)
> table(churnTrain$Class)
```

(a) Explore the data by visualizing the relationship between the predictors and the outcome. Are there important features of the predictor data themselves, such as between-predictor correlations or degenerate distributions? Can functions of more than one predictor be used to model the data more effectively?
(b) Fit some basic models to the training set and tune them via resampling. What criteria should be used to evaluate the effectiveness of the models?
(c) Use lift charts to compare models. If you wanted to identify 80 % of the churning customers, how many other customers would also be identified?