Chapter 6
Linear Regression and Its Cousins

In this chapter we will discuss several models, all of which are akin to linear regression in that each can directly or indirectly be written in the form

$$y_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + \cdots + b_P x_{iP} + e_i, \tag{6.1}$$

where y_i represents the numeric response for the ith sample, b_0 represents the estimated intercept, b_j represents the estimated coefficient for the jth predictor, x_{ij} represents the value of the jth predictor for the ith sample, and e_i represents random error that cannot be explained by the model. When a model can be written in the form of Eq. 6.1, we say that it is *linear in the parameters*. In addition to ordinary linear regression, these types of models include partial least squares (PLS) and penalized models such as ridge regression, the lasso, and the elastic net.

Each of these models seeks to find estimates of the parameters so that the sum of the squared errors or a function of the sum of the squared errors is minimized. Section 5.2 illustrated that the mean squared error (MSE) can be divided into components of irreducible variation, model bias, and model variance. The objectives of the methods presented in this chapter find parameter estimates that fall along the spectrum of the bias-variance trade-off. Ordinary linear regression, at one extreme, finds parameter estimates that have minimum bias, whereas ridge regression, the lasso, and the elastic net find estimates that have lower variance. The impact of this trade-off on the predictive ability of these models will be illustrated in the sections to follow.

A distinct advantage of models that follow the form of Eq. 6.1 is that they are highly interpretable. For example, if the estimated coefficient of a predictor is 2.5, then a 1 unit increase in that predictor's value would, on average, increase the response by 2.5 units. Furthermore, relationships among predictors can be further interpreted through the estimated coefficients.

Another advantage of these kinds of models is that their mathematical nature enables us to compute standard errors of the coefficients, provided that we make certain assumptions about the distributions of the model residuals.

M. Kuhn and K. Johnson, *Applied Predictive Modeling*,
DOI 10.1007/978-1-4614-6849-3_6,

These standard errors can then be used to assess the statistical significance of each predictor in the model. This inferential view can provide a greater degree of understanding of the model, as long as the distributional assumptions are adequately met. Because this work focuses on model prediction, we will not spend much time on the inferential nature of these models.

While linear regression-type models are highly interpretable, they can be limited in their usefulness. First, these models are appropriate when the relationship between the predictors and response falls along a hyperplane. For example, if the data had just one predictor, then the techniques would be appropriate if the relationship between the predictor and response fell along a straight line. With more predictors, the relationship would need to fall close to a flat hyperplane. If there is a curvilinear relationship between the predictors and response (e.g., such as quadratic, cubic, or interactions among predictors), then linear regression models can be augmented with additional predictors that are functions of the original predictors in an attempt to capture these relationships. More discussion about strategies for augmenting the original predictors will follow in the sections below. However, nonlinear relationships between predictors and the response may not be adequately captured with these models. If this is the case for the data, then the methods detailed in Chaps. 7 and 8 will better uncover the predictive relationship between the predictors and the response.

6.1 Case Study: Quantitative Structure-Activity Relationship Modeling

Chemicals, including drugs, can be represented by chemical formulas. For example, Fig. 6.1 shows the structure of aspirin, which contains nine carbon, eight hydrogen, and four oxygen atoms. From this configuration, quantitative measurements can be derived, such as the molecular weight, electrical charge, or surface area. These quantities are referred to as *chemical descriptors*, and there are myriad types of descriptors that can be derived from a chemical equation. Some are simplistic, such as the number of carbon atoms, while others could be described as arcane (e.g., the coefficient sum of the last eigenvector from Barysz matrix weighted by the van der Waals volume).

Some characteristics of molecules cannot be analytically determined from the chemical structure. For example, one way a compound may be of medical value is if it can inhibit production of a specific protein. This is usually called the biological activity of a compound. The relationship between the chemical structure and its activity can be complex. As such, the relationship is usually determined empirically using experiments. One way to do this is to create a biological assay for the target of interest (i.e., the protein). A set of compounds can then be placed into the assay and their activity, or inhibition, is measured. This activity information generates data which can be used as

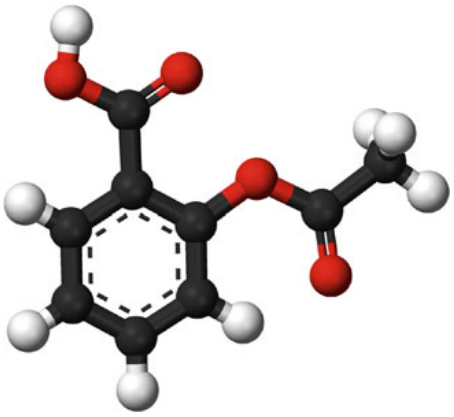

Fig. 6.1: A representation of aspirin, which contains carbon atoms (shown as *black balls*) and hydrogen (*white*) and oxygen atoms (*red*). The chemical formula for this molecule is `O=C(Oc1ccccc1C(=O)O)C`, from which molecular descriptors can be determined, such as a molecular weight of 180.2 g/mol

the training set for predictive modeling so that compounds, which may not yet exist, can be screened for activity. This process is referred to as quantitative structure-activity relationship (QSAR) modeling. Leach and Gillet (2003) provide a high-level introduction to QSAR modeling and molecular descriptors.

While activity is important, other characteristics need to be assessed to determine if a compound is "drug-like" (Lipinski et al. 1997). Physical qualities, such as the solubility or lipophilicity (i.e., "greasiness"), are evaluated as well as other properties, such as toxicity. A compound's solubility is very important if it is to be given orally or by injection. We will demonstrate various regression modeling techniques by predicting solubility using chemical structures.

Tetko et al. (2001) and Huuskonen (2000) investigated a set of compounds with corresponding experimental solubility values using complex sets of descriptors. They used linear regression and neural network models to estimate the relationship between chemical structure and solubility. For our analyses, we will use 1,267 compounds and a set of more understandable descriptors that fall into one of three groups:

- Two hundred and eight binary "fingerprints" that indicate the presence or absence of a particular chemical substructure.
- Sixteen count descriptors, such as the number of bonds or the number of bromine atoms.
- Four continuous descriptors, such as molecular weight or surface area.

On average, the descriptors are uncorrelated. However, there are many pairs that show strong positive correlations; 47 pairs have correlations greater than

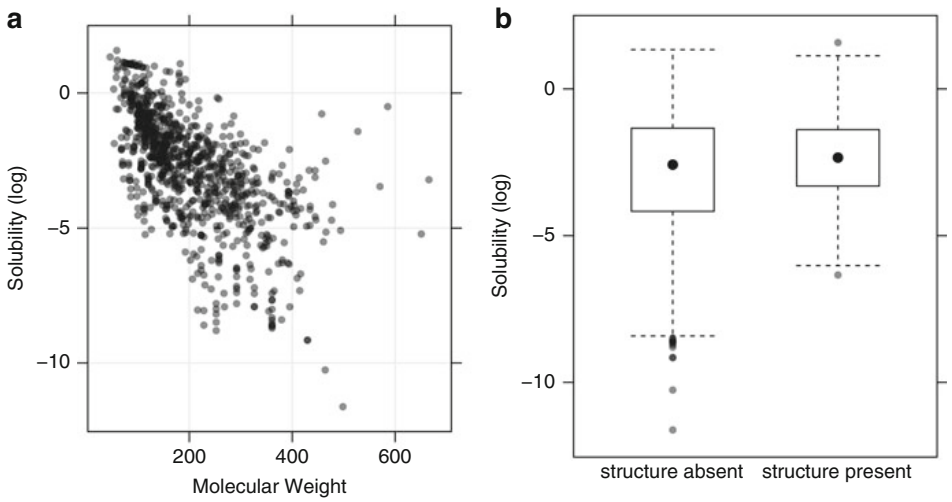

Fig. 6.2: The relationship between solubility and two descriptors. *Left*: As molecular weight of a molecule increases, the solubility generally decreases. The relationship is roughly log-linear, except for several compounds with low solubility and large weight and solubility between 0 and −5. *Right*: For a particular fingerprint descriptor, there is slightly higher solubility when the substructure of interest is absent from the molecule

0.90. In some cases, we should expect correlations between descriptors. In the solubility data, for example, the surface area of a compound is calculated for regions associated with certain atoms (e.g., nitrogen or oxygen). One descriptor in these data measures the surface area associated with two specific elements while another uses the same elements plus two more. Given their definitions, we would expect that the two surface area predictors would be correlated. In fact, the descriptors are identical for 87 % of the compounds. The small differences between surface area predictors may contain some important information for prediction, but the modeler should realize that there are implications of redundancy on the model. Another relevant quality of the solubility predictors is that the count-based descriptors show a significant right skewness, which may have an impact on some models (see Chap. 3 for a discussion of these issues).

The outcome data were measured on the $\log_{10}$ scale and ranged from −11.6 to 1.6 with an average log solubility value of −2.7. Figure 6.2 shows the relationship between the experimentally derived solubility values and two types of descriptors in the example data.

The data were split using random sampling into a training set ($n = 951$) and test set ($n = 316$). The training set will be used to tune and estimate models, as well as to determine initial estimates of performance using repeated 10-fold cross-validation. The test set will be used for a final characterization of the models of interest.

It is useful to explore the training set to understand the characteristics of the data prior to modeling. Recall that 208 of the predictors are binary fingerprints. Since there are only two values of these variables, there is very little that pre-processing will accomplish.

Moving on, we can evaluate the continuous predictors for skewness. The average skewness statistic was 1.6 (with a minimum of 0.7 and a maximum of 3.8), indicating that these predictors have a propensity to be right skewed. To correct for this skewness, a Box–Cox transformation was applied to all predictors (i.e., the transformation parameter was not estimated to be near one for any of the continuous predictors).

Using these transformed predictors, is it safe to assume that the relationship between the predictors and the outcome is linear? Figure 6.3 shows scatter plots of the predictors against the outcome along with a regression line from a flexible "smoother" model called loess (Cleveland 1979). The smoothed regression lines indicate that there are some linear relationships between the predictors and the outcome (e.g., molecular weight) and some nonlinear relationships (e.g., the number of origins or chlorines). Because of this, we might consider augmenting the predictor set with quadratic terms for some variables.

Are there significant between-predictor correlations? To answer this question, principal component analysis (PCA) was used on the full set of transformed predictors, and the percent of variance accounted for by each component is determined. Figure 6.4 is commonly known as a scree plot and displays a profile of the variability accounted for by each component. Notice that the amount of variability summarized by component drops sharply, with no one component accounting for more than 13 % of the variance. This profile indicates that the structure of the data is contained in a much smaller number of dimensions than the number of dimensions of the original space; this is often due to a large number of collinearities among the predictors. Figure 6.5 shows the correlation structure of the transformed continuous predictors; there are many strong positive correlations (indicated by the large, dark blue circles). As previously discussed, this could create problems in developing some models (such as linear regression), and appropriate pre-processing steps will need to be taken to account for this problem.

6.2 Linear Regression

The objective of ordinary least squares linear regression is to find the plane that minimizes the sum-of-squared errors (SSE) between the observed and predicted response:

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

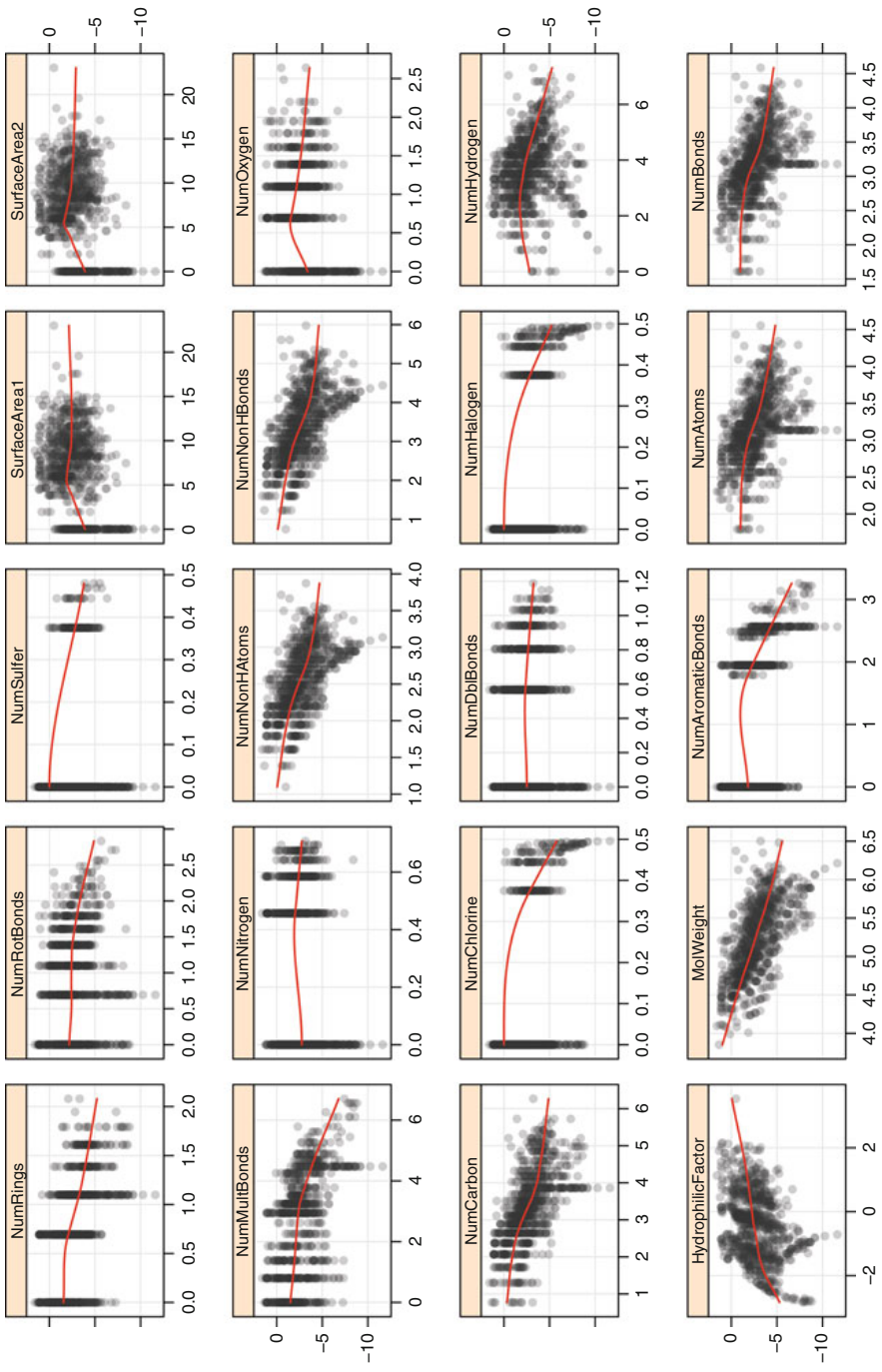

Fig. 6.3: Scatter plots of the transformed continuous predictors in the solubility data set. The *red line* is a scatter plot smoother

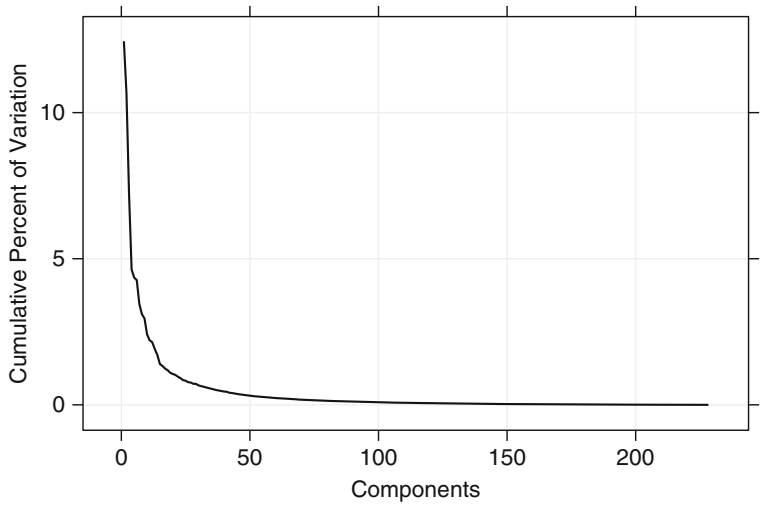

Fig. 6.4: A scree plot from a PCA analysis of the solubility predictors

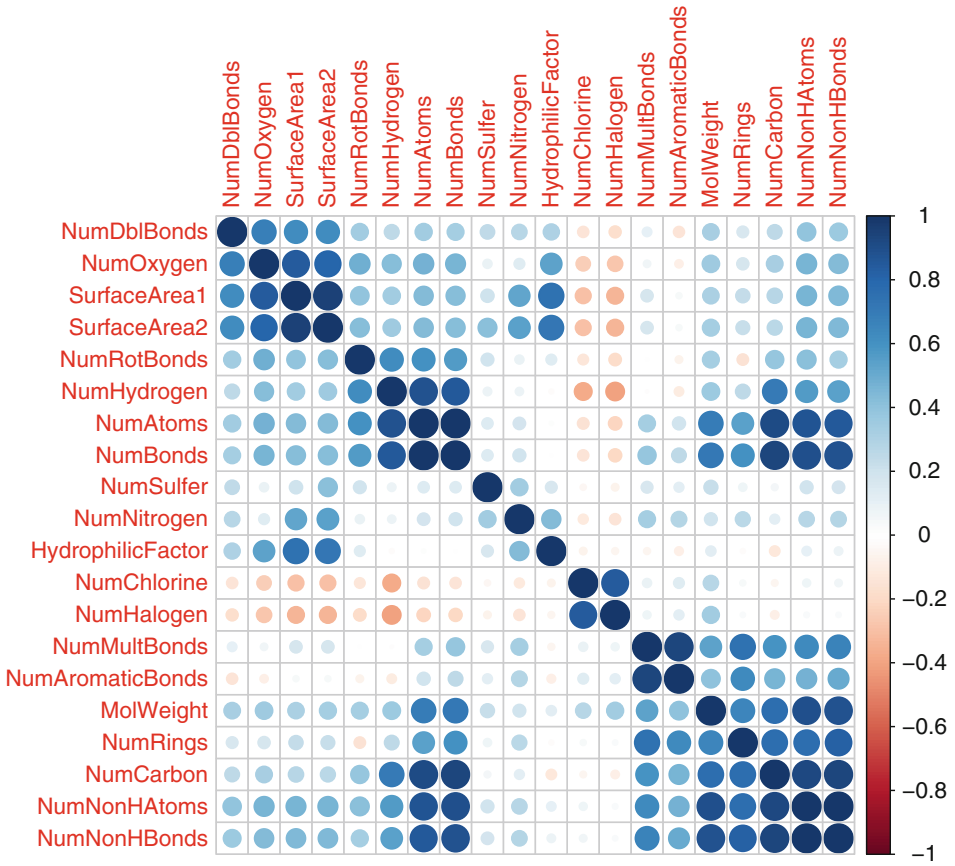

Fig. 6.5: Between-predictor correlations of the transformed continuous solubility predictors

where y_i is the outcome and $\hat{y}_i$ is the model prediction of that sample's outcome. Mathematically, the optimal plane can be shown to be

$$\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T y, \tag{6.2}$$

where $\mathbf{X}$ is the matrix of predictors and y is the response vector. Equation 6.2 is also known as $\hat{\beta}$ ("beta-hat") in statistical texts and is a vector that contains the parameter estimates or coefficients for each predictor. This quantity (6.2) is easy to compute, and the coefficients are directly interpretable. Making some minimal assumptions about the distribution of the residuals, it is straightforward to show that the parameter estimates that minimize SSE are the ones that have the least bias of all possible parameter estimates (Graybill 1976). Hence, these estimates minimize the bias component of the bias-variance trade-off.

The interpretability of coefficients makes it very attractive as a modeling tool. At the same time, the characteristics that make it interpretable also make it prone to potentially fatal flaws. Notice that embedded in Eq. (6.2) is the term $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$, which is proportional to the covariance matrix of the predictors. A unique inverse of this matrix exists when (1) no predictor can be determined from a combination of one or more of the other predictors and (2) the number of samples is greater than the number of predictors. If the data fall under either of these conditions, then a unique set of regression coefficients does not exist. However, a unique set of predicted values can still be obtained for data that fall under condition (1) by either replacing $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$ with a conditional inverse (Graybill 1976) or by removing predictors that are collinear. By default, when fitting a linear model with R and collinearity exists among predictors, "...R fits the largest identifiable model by removing variables in the reverse order of appearance in the model formula" (Faraway 2005). The upshot of these facts is that linear regression can still be used for prediction when collinearity exists within the data. But since the regression coefficients to determine these predictions are not unique, we lose our ability to meaningfully interpret the coefficients.

When condition (2) is true for a data set, the practitioner can take several steps to attempt to build a regression model. As a first step we suggest using pre-processing techniques presented in Sect. 3.3 to remove pairwise correlated predictors, which will reduce the number of overall predictors. However, this pre-processing step may not completely eliminate collinearity, since one or more of the predictors may be functions of *two* or more of the other predictors. To diagnose multicollinearity in the context of linear regression, the *variance inflation factor* can be used (Myers 1994). This statistic is computed for each predictor and a function of the correlation between the selected predictor and all of the other predictors.

After pre-processing the data, if the number of predictors still outnumbers the number of observations, then we will need to take other measures to reduce the dimension of the predictor space. PCA pre-processing (Sect. 3.3)

is one possible remedy. Other remedies include simultaneous dimension reduction and regression via PLS or employing methods that shrink parameter estimates such as ridge regression, the lasso, or the elastic net.

Another drawback of multiple linear regression is that its solution is linear in the parameters. This means that the solution we obtain is a flat hyperplane. Clearly, if the data have curvature or nonlinear structure, then regression will not be able to identify these characteristics. One visual clue to understanding if the relationship between predictors and the response is not linear is to examine the basic diagnostic plots illustrated in Fig. 5.3. Curvature in the predicted-versus-residual plot is a primary indicator that the underlying relationship is not linear. Quadratic, cubic, or interactions between predictors can be accommodated in regression by adding quadratic, cubic, and interactions of the original predictors. But the larger the number of original predictors, the less practical including some or all of these terms becomes. Taking this approach can cause the data matrix to have more predictors than observations, and we then again cannot invert the matrix.

If easily identifiable nonlinear relationships exist between the predictors and the response, then those additional predictors can be added to the descriptor matrix. If, however, it is not possible to identify these relationships or the relationships between the predictors and the response is highly nonlinear, then more complex methods such as those discussed in Chap. 7 will more effectively and efficiently find this structure.

A third notable problem with multiple linear regression is that it is prone to chasing observations that are away from the overall trend of the majority of the data. Recall that linear regression seeks to find the parameter estimates that minimize SSE; hence, observations that are far from the trend of the majority of the data will have exponentially large residuals. In order to minimize SSE, linear regression will adjust the parameter estimates to better accommodate these unusual observations. Observations that cause significant changes in the parameter estimates are called *influential*, and the field of robust regression has been developed to address these kinds of problems. One common approach is to use an alternative metric to SSE that is less sensitive to large outliers. For example, finding parameter estimates that minimize the sum of the absolute errors is more resistant to outliers, as seen in Fig. 6.6. Also, the Huber function uses the squared residuals when they are "small" and the simple different between the observed and predicted values when the residuals are above a threshold. This approach can effectively minimize the influence of observations that fall away from the overall trend in the data.

There are no tuning parameters for multiple linear regression. This fact, however, does not impugn the practitioner from using rigorous model validation tools, especially when using this model for prediction. In fact, we must use the same training and validation techniques described in Chap. 4 to understand the predictive ability of this model on data which the model has not seen.

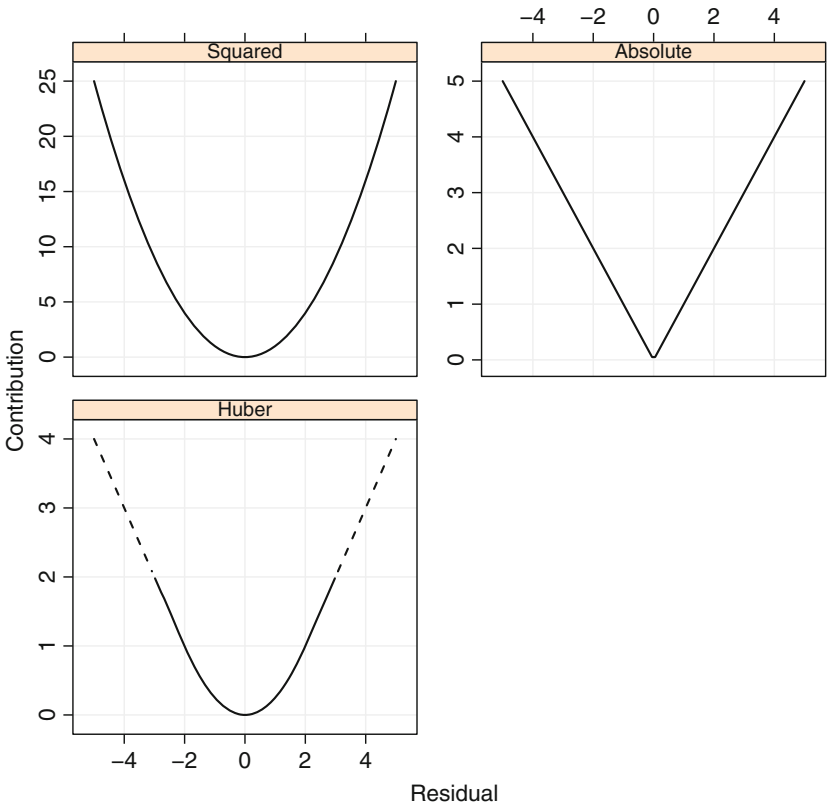

Fig. 6.6: The relationship between a model residual and its contribution to the objective function for several techniques. For the Huber approach, a threshold of 2 was used

When using resampling techniques such as bootstrapping or cross-validation, the practitioner must still be conscious of the problems described above. Consider, for example, a data set where there are 100 samples and 75 predictors. If we use a resampling scheme that uses two-thirds of the data for training, then we will be unable to find a unique set of regression coefficients, since the number of predictors in the training set will be larger than the number of samples. Therefore for multiple linear regression, the practitioner must be aware of its pitfalls not only when working with the original data set but also when working with subsets of data created during model training and evaluation.

To illustrate the problem of correlated predictors, linear models were fit with combinations of descriptors related to the number of non-hydrogen atoms and the number of hydrogen bonds. In the training set, these predictors are highly correlated (correlation: 0.994). Figure 6.3 shows their relationship with the outcome, which is almost identical. First, we fit two separate regression models with the individual terms and then a third model with both

Table 6.1: Regression coefficients for two highly correlated predictors across four separate models

Model	NumNonHAtoms	NumNonHBonds
NumNonHAtoms only	−1.2 (0.1)	
NumNonHBonds only		−1.2 (0.1)
Both	−0.3 (0.5)	−0.9 (0.5)
All predictors	8.2 (1.4)	−9.1 (1.6)

terms. The predictors were centered and scaled prior to modeling so that their units would be the same. Table 6.1 shows the regression coefficients and their standard errors in parentheses. For the individual models, the regression coefficients are almost identical as are their standard errors. However, when fitting a model with both terms, the results differ; the slope related to the number of non-hydrogen atoms is greatly decreased. Also, the standard errors are increased fivefold when compared to the individual models. This reflects the instability in the regression linear caused by the between-predictor relationships and this instability is propagated directly to the model predictions. Table 6.1 also shows the coefficients for these two descriptors when all of the predictors are put into the model. Recall from Fig. 6.5 that there are many collinear predictors in the data and we would expect the effect of collinearity to be exacerbated. In fact, for these two predictors, the values become wildly large in magnitude and their standard errors are 14–16-fold larger than those from the individual models.

In practice, such highly correlated predictors might be managed manually by removing one of the offending predictors. However, if the number of predictors is large, this may be difficult. Also, on many occasions, relationships among predictors can be complex and involve many predictors. In these cases, manual removal of specific predictors may not be possible and models that can tolerate collinearity may be more useful.

Linear Regression for Solubility Data

Recall that in Sect. 6.1 we split the solubility data into training and test sets and that we applied a Box–Cox transformation to the continuous predictors in order to remove skewness. The next step in the model building process for linear regression is to identify predictors that have high pairwise correlations and to remove predictors so that no absolute pairwise correlation is greater than some pre-specified level. In this case we chose to remove predictors that have pairwise correlations greater than 0.9 (see Sect. 3.3). At this level, 38 predictors were identified and removed. Upon removing these predictors, a

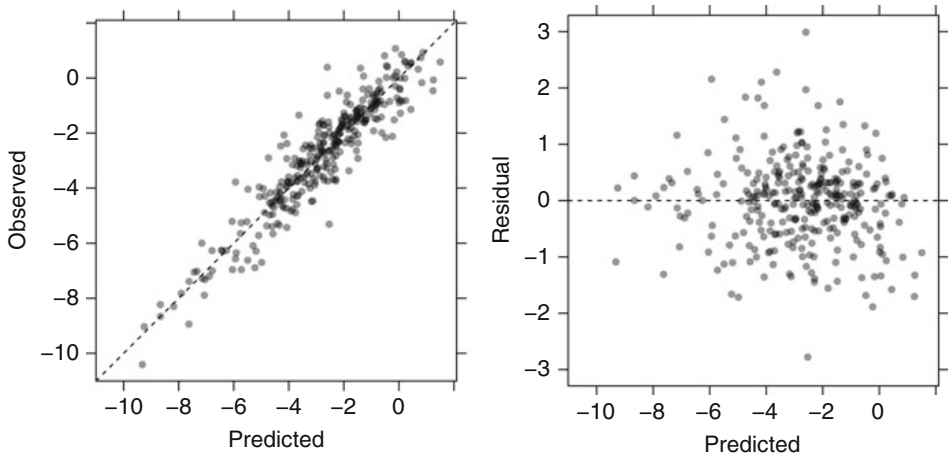

Fig. 6.7: *Left*: Observed versus predicted values for the solubility test set. *Right*: Residuals versus the predicted values. The residuals appear to be randomly scattered about 0 with respect to the predicted values

linear model was fit to the training data.[1] The linear model was resampled using 10-fold cross-validation and the estimated root mean squared error (RMSE) was 0.71 with a corresponding R^2 value of 0.88.

Predictors that were removed from the training data were then also removed from the test data and the model was then applied to the test set. The R^2 value between the observed and predicted values was 0.87, and the basic regression diagnostic plots are displayed in Fig. 6.7. There does not appear to be any bias in the prediction, and the distribution between the predicted values and residuals appears to be random about zero.

6.3 Partial Least Squares

For many real-life data sets, predictors can be correlated and contain similar predictive information like illustrated with the solubility data. If the correlation among predictors is high, then the ordinary least squares solution for multiple linear regression will have high variability and will become unstable.

[1] In practice, the correlation threshold would need to be smaller to have a significant effect on collinearity. In these data, it would also remove important variables. Also, one would investigate how the terms fit into the model. For example, there may be interactions between predictors that are important and nonlinear transformations of predictors may also improve the model. For these data, this set of activities is examined more closely in Chap. 19.

For other data sets, the number of predictors may be greater than the number of observations. In this case, too, ordinary least squares in its usual form will be unable to find a unique set of regression coefficients that minimize the SSE.

A couple of common solutions to the regression problem under these conditions include pre-processing the predictors by either (1) removal of the highly correlated predictors using techniques as described in Sect. 3.3 or (2) conducting PCA on the predictors as described in Sect. 3.3. Removing highly correlated predictors ensures that pairwise correlations among predictors are below a pre-specified threshold. However, this process does not necessarily ensure that linear combinations of predictors are uncorrelated with other predictors. If this is the case, then the ordinary least squares solution will still be unstable. Therefore it is important to understand that the removal of highly correlated pairwise predictors may not guarantee a stable least squares solution. Alternatively, using PCA for pre-processing guarantees that the resulting predictors, or combinations thereof, will be uncorrelated. The trade-off in using PCA is that the new predictors are linear combinations of the original predictors, and thus, the practical understanding of the new predictors can become murky.

Pre-processing predictors via PCA prior to performing regression is known as principal component regression (PCR) (Massy 1965); this technique has been widely applied in the context of problems with inherently highly correlated predictors or problems with more predictors than observations. While this two-step regression approach (dimension reduction, then regression) has been successfully used to develop predictive models under these conditions, it can easily be misled. Specifically, dimension reduction via PCA does not necessarily produce new predictors that explain the response. As an example of this scenario, consider the data in Fig. 6.8 which contains two predictors and one response. The two predictors are correlated, and PCA summarizes this relationship using the direction of maximal variability. The right-hand plot of this figure, however, illustrates that the first PCA direction contains no predictive information about the response.

As this simple example illustrates, PCA does not consider any aspects of the response when it selects its components. Instead, it simply chases the variability present throughout the predictor space. If that variability happens to be related to the response variability, then PCR has a good chance to identify a predictive relationship. If, however, the variability in the predictor space is not related to the variability of the response, then PCR can have difficulty identifying a predictive relationship when one might actually exist. Because of this inherent problem with PCR, we recommend using PLS when there are correlated predictors and a linear regression-type solution is desired.

PLS originated with Herman Wold's nonlinear iterative partial least squares (NIPALS) algorithm (Wold 1966, 1982) which linearized models that were nonlinear in the parameters. Subsequently, Wold et al. (1983) adapted the NIPALS method for the regression setting with correlated predictors and

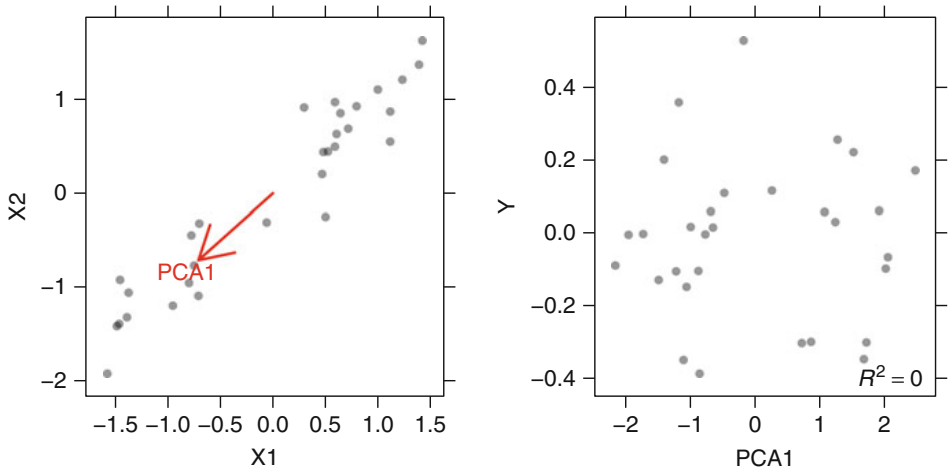

Fig. 6.8: An example of principal component regression for a simple data set with two predictors and one response. *Left*: A scatter plot of the two predictors shows the direction of the first principal component. *Right*: The first PCA direction contains no predictive information for the response

called this adaptation "PLS." Briefly, the NIPALS algorithm iteratively seeks to find underlying, or latent, relationships among the predictors which are highly correlated with the response. For a univariate response, each iteration of the algorithm assesses the relationship between the predictors ($\mathbf{X}$) and response ($\mathbf{y}$) and numerically summarizes this relationship with a vector of weights ($\mathbf{w}$); this vector is also known as a *direction*. The predictor data are then orthogonally projected onto the direction to generate scores ($\mathbf{t}$). The scores are then used to generate loadings ($\mathbf{p}$), which measure the correlation of the score vector to the original predictors. At the end of each iteration, the predictors and the response are "deflated" by subtracting the current estimate of the predictor and response structure, respectively. The new deflated predictor and response information are then used to generate the next set of weights, scores, and loadings. These quantities are sequentially stored in matrices $\mathbf{W}$, $\mathbf{T}$, and $\mathbf{P}$, respectively, and are used for predicting new samples and computing predictor importance. A schematic of the PLS relationship between predictors and the response can be seen in Fig. 6.9, and a thorough explanation of the algorithm can be found in Geladi and Kowalski (1986).

To obtain a better understanding of the algorithm's function, Stone and Brooks (1990) linked it to well-known statistical concepts of covariance and regression. In particular, Stone and Brooks showed that like PCA, PLS finds linear combinations of the predictors. These linear combinations are commonly called *components* or latent variables. While the PCA linear combinations are chosen to maximally summarize predictor space variability, the PLS linear combinations of predictors are chosen to maximally summarize

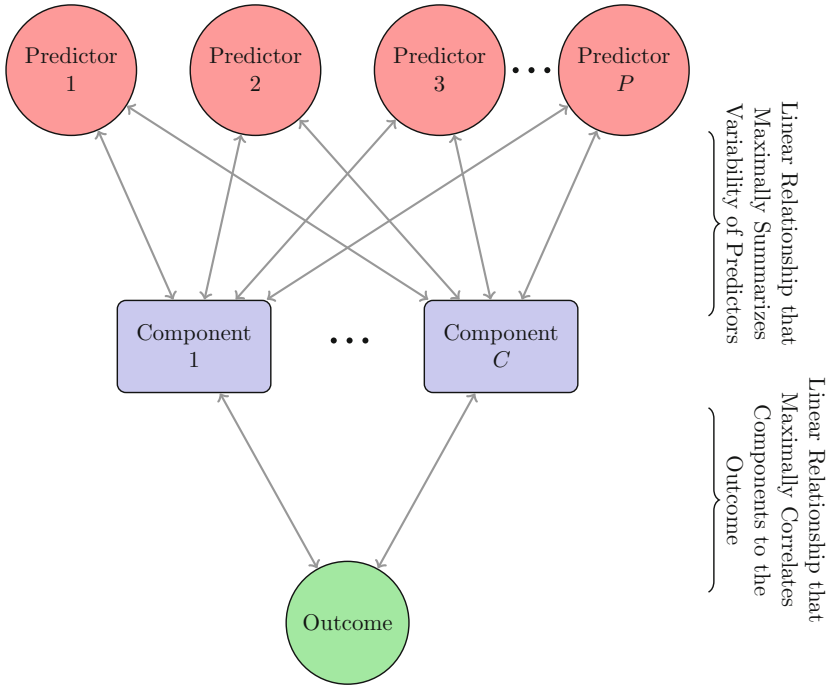

Fig. 6.9: A diagram depicting the structure of a PLS model. PLS finds components that simultaneously summarize variation of the predictors while being optimally correlated with the outcome

covariance with the response. This means that PLS finds components that maximally summarize the variation of the predictors while simultaneously requiring these components to have maximum correlation with the response. PLS therefore strikes a compromise between the objectives of predictor space dimension reduction and a predictive relationship with the response. In other words, PLS can be viewed as a *supervised* dimension reduction procedure; PCR is an *unsupervised* procedure.

To better understand how PLS works and to relate it to PCR, we will revisit the data presented in Fig. 6.8. This time we seek the first PLS component. The left-hand scatter plot in Fig. 6.10 contrasts the first PLS direction with the first PCA direction. For this illustration the two directions are nearly orthogonal, indicating that the optimal dimension reduction direction was not related to maximal variation in the predictor space. Instead, PLS identified the optimal predictor space dimension reduction for the purpose of regression with the response.

Clearly this example is designed to show an important flaw with PCR. In practice, PCR does not fail this drastically; rather, PCR produces models with similar predictive ability to PLS. Based on our experience, the number of components retained via cross-validation using PCR is always equal to

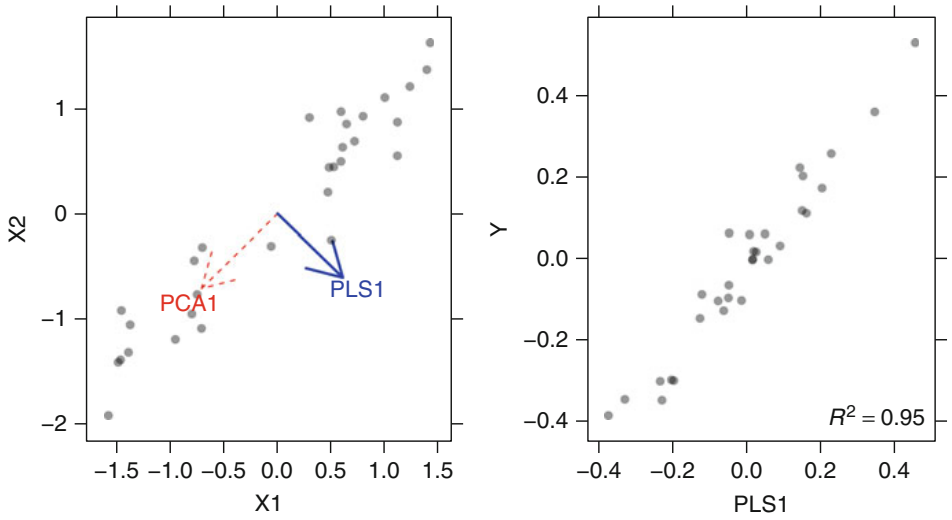

Fig. 6.10: An example of partial least squares regression for a simple data set with two predictors and one response. *Left*: The first PLS direction is nearly orthogonal to the first PCA direction. *Right*: Unlike PCA, the PLS direction contains highly predictive information for the response

or greater than the number of components retained by PLS. This is due to the fact that dimensions retained by PLS have been chosen to be optimally related to the response, while those chosen with PCR are not.

Prior to performing PLS, the predictors should be centered and scaled, especially if the predictors are on scales of differing magnitude. As described above, PLS will seek directions of maximum variation while simultaneously considering correlation with the response. Even with the constraint of correlation with the response, it will be more naturally drawn towards predictors with large variation. Therefore, predictors should be adequately preprocessed prior to performing PLS.

Once the predictors have been preprocessed, the practitioner can model the response with PLS. PLS has one tuning parameter: the number of components to retain. Resampling techniques as described in Sect. 4.4 can be used to determine the optimal number of components.

PCR and PLSR for Solubility Data

To demonstrate the model building process with PLS, let's return to the solubility data from Sect. 6.1. Although there are 228 predictors, Figs. 6.4 and 6.5 show that many predictors are highly correlated and that the overall information within the predictor space is contained in a smaller number of dimensions. These predictor conditions are very favorable for applying PLS.

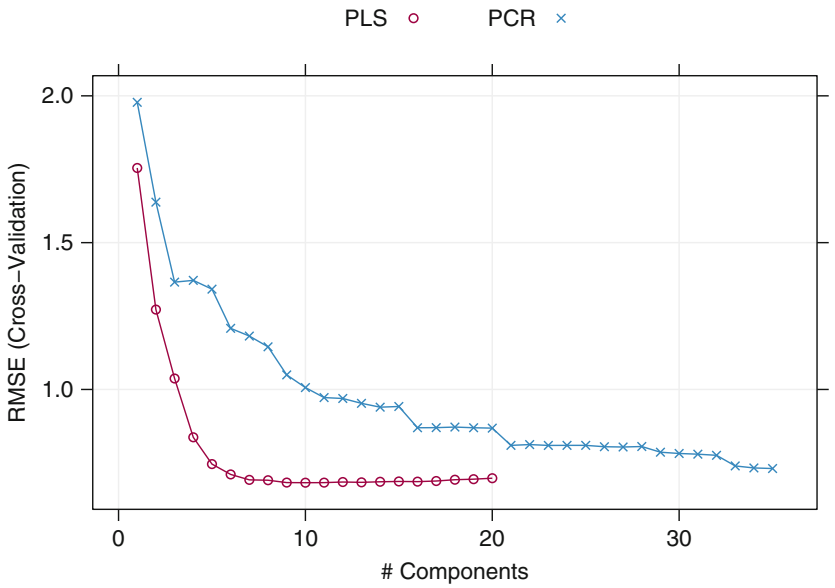

Fig. 6.11: Cross-validated RMSE by component for PLS and PCR. RMSE is minimized with ten PLS components and 35 PCR components

Cross-validation was used to determine the optimal number of PLS components to retain that minimize RMSE. At the same time, PCR was performed using the same cross-validation sets to compare its performance to PLS. Figure 6.11 contains the results, where PLS found a minimum RMSE (0.682) with ten components and PCR found a minimum RMSE (0.731) with 35 components. We see with these data that the supervised dimension reduction finds a minimum RMSE with significantly fewer components than unsupervised dimension reduction. Using the one-standard error rule (Sect. 4.6) would reduce the number of required PLS components to 8.

Figure 6.12 contrasts the relationship between each of the first two PCR and PLS components with the response. Because the RMSE is lower for each of the first two PLS components as compared to the first two PCR components, it is no surprise that the correlation between these components and the response is greater for PLS than PCR. This figure illustrates that PLS is more quickly being steered towards the underlying relationship with the response.

Prediction of the test set using the optimal PCR and PLS models can be seen in Fig. 6.13. The predictive ability of each method is good, and the residuals appear to be randomly scattered about zero. Although the predictive ability of these models is close, PLS finds a simpler model that uses far fewer components than PCR.

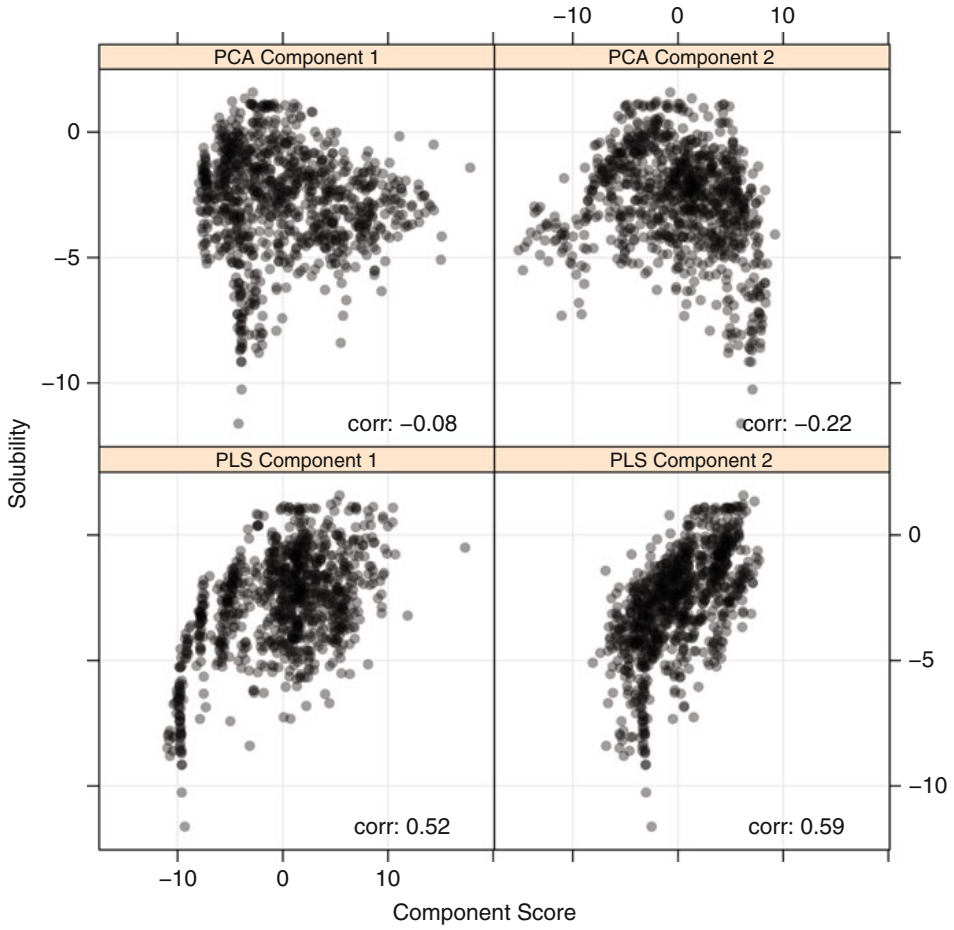

Fig. 6.12: A contrast of the relationship between each of the first two PCR and PLS components with the solubility response. Because the dimension reduction offered by PLS is supervised by the response, it is more quickly steered towards the underlying relationship between the predictors and the response

The PLS regression coefficients for the solubility data are presented in Table 6.2 (page 127), and the magnitudes are similar to the linear regression model that includes only those two predictors.

Because the latent variables from PLS are constructed using linear combinations of the original predictors, it is more difficult to quantify the relative contribution of each predictor to the model. Wold et al. (1993) introduced a heuristic way to assess variable importance when using the NIPALS algorithm and termed this calculation *variable importance in the projection*. In the simple case, suppose that the relationship between the predictors and the response can be adequately summarized by a one-component PLS model. The importance of the jth predictor is then proportional to the value of

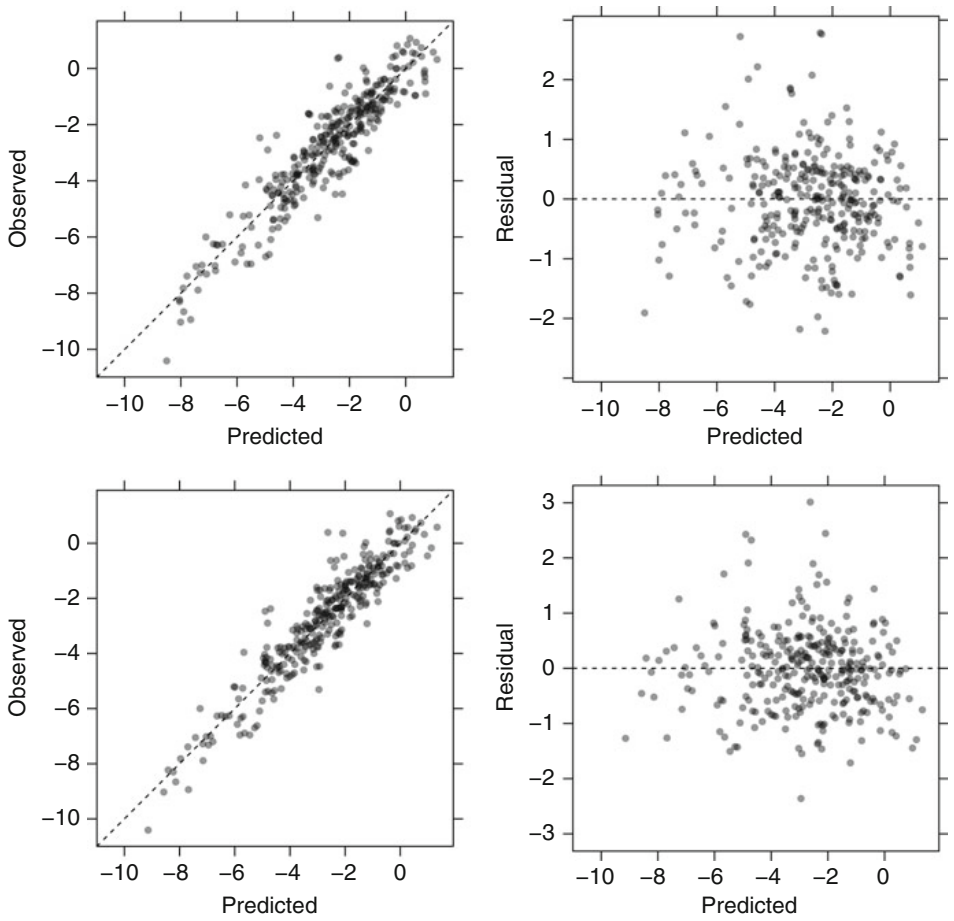

Fig. 6.13: *Left side*: Observed versus predicted values for the solubility test set for PCR (*upper*) and PLS (*lower*). *Right side*: Residuals versus the predicted values for PCR and PLS. The residuals appear to be randomly scattered about 0 with respect to the predicted values. Both methods have similar predictive ability, but PLS does so with far fewer components

the normalized weight vector, w, corresponding to the jth predictor. When the relationship between predictors and the response requires more than one component, the variable importance calculation becomes more involved. In this case, the numerator of the importance of the jth predictor is a weighted sum of the normalized weights corresponding to the jth predictor. The jth normalized weight of the kth component, w_{kj}, is scaled by the amount of variation in the response explained by the kth component. The denominator of the variable importance is the total amount of response variation explained by all k components. Therefore, the larger the normalized weight and amount of response variation explained by the component, the more important predictor is in the PLS model.

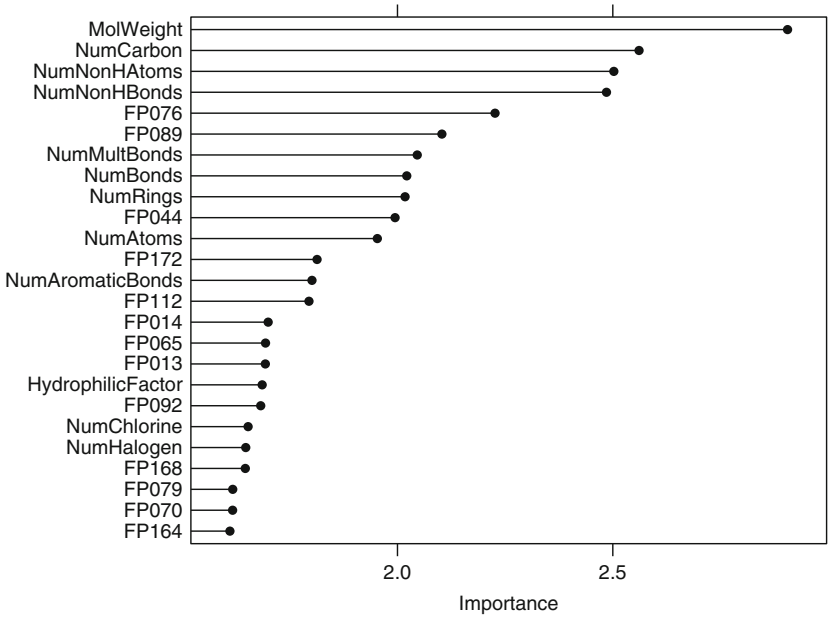

Fig. 6.14: Partial least squares variable importance scores for the solubility data

For the solubility data, the top 25 most important predictors are shown in Fig. 6.14. The larger the VIP value, the more important the predictor is in relating the latent predictor structure to the response. By its construction, the squared VIP values sum to the total number of predictors. As a rule-of-thumb, VIP values exceeding 1 are considered to contain predictive information for the response. Wold (1995) further suggests that predictors with small PLS regression coefficients and small VIP values are likely not important and should be considered as candidates for removal from the model.

Algorithmic Variations of PLS

The NIPALS algorithm works fairly efficiently for data sets of small-to-moderate size (e.g., $< 2{,}500$ samples and < 30 predictors) (Alin 2009). But when the number of samples (n) and predictors (P) climbs, the algorithm becomes inefficient. This inefficiency is due to the way the matrix operations on the predictors and the response are performed. Specifically, both the predictor matrix and the response must be deflated (i.e., information must be subtracted from each matrix, thus creating new versions of each matrix) for each latent variable. This implies that different versions of the predictor matrix and response must be kept at each iteration of the algorithm. Therefore an $n \times P$ matrix and an $n \times 1$ vector must be recomputed, operated on, and

stored in each iteration. As n and P grow, so do the memory requirements, and operations on these matrices must be performed throughout the iterative process.

In a computational step forward, Lindgren et al. (1993) showed that the constructs of NIPALS could be obtained by working with a "kernel" matrix of dimension $P \times P$, the covariance matrix of the predictors (also of dimension $P \times P$), and the covariance matrix of the predictors and response (of dimension $P \times 1$). This adjustment improved the speed of the algorithm, especially as the number of observations became much larger than the number of predictors.

At nearly the same time as the kernel approach was developed, de Jong (1993) improved upon the NIPALS algorithm by viewing the underlying problem as finding latent orthogonal variables in the predictor space that maximize the covariance with the response. This perspective shift led to a different algorithm that focused on deflating the covariance matrix between the predictors and the response rather than deflating both the predictor matrix and the response. de Jong (1993) termed the new approach "SIMPLS" because it was a simple modification of the PLS algorithm that was framed through statistics. Because the SIMPLS approach deflates the covariance matrix, it requires storing just the deflated covariance matrix at each iteration which has dimension $P \times 1$—a significant computational improvement over the storage requirements of NIPALS. Although the SIMPLS approach solves the optimization in a different way, de Jong (1993) showed that the SIMPLS latent variables are identical to those from NIPALS when there is only one response. (More will be discussed below when modeling a multivariate response.)

Other authors have also proposed computational modifications to the NIPALS algorithm through adjustments to the kernel approach (de Jong and Ter Braak 1994; Dayal and MacGregor 1997). Dayal and MacGregor (1997) developed two efficient modifications, especially when $n >> P$, and, similar to SIMPLS, only require a deflation of the covariance matrix between the predictors and the response at each step of the iterative process. In their first alteration to the inner workings of the algorithm, the original predictor matrix is used in the computations (without deflation). In the second alteration, the covariance matrix of the predictors is used in the computations (also without deflation).

Alin (2009) provided a comprehensive computational efficiency comparison of NIPALS to other algorithmic modifications. In this work, Alin used a varying number of samples (500–10,000), predictors (10–30), responses (1–15), and number of latent variables to derive (3–10). In nearly every scenario, the second kernel algorithm of Dayal and MacGregor was more computationally efficient than all other approaches and provided superior performance when $n > 2,500$ and $P > 30$. And in the cases where the second algorithm did not provide the most computational efficiency, the first algorithm did.

The above approaches to implementing PLS provide clear computational advantages over the original algorithm. However, as the number of predictors grows, each becomes less efficient. To address this scenario when $P > n$,

Rännar et al. (1994) constructed a kernel based on the predictor matrix and response that had dimension $n \times n$. A usual PLS analysis can then be performed using this kernel, the outer products of the predictors, and the outer products of the response (each with dimension $n \times n$). Hence, this algorithm is computationally more efficient when there are more predictors than samples.

As noted in Fig. 6.9, the PLS components summarize the data through linear substructures (i.e., hyperplanes) of the original predictor space that are related to the response. But for many problems, the underlying structure in the predictor space that is optimally related to the response is not linear but curvilinear or nonlinear. Several authors have attempted to address this shortcoming of PLS in order to find this type of predictor space/response relationship. While many methods exist, the most easily adaptable approaches using the algorithms explained above are provided by Berglund and Wold (1997) and Berglund et al. (2001). In Berglund and Wold (1997), the authors show that adding squared predictors (and cubic, if necessary) can be included with the original predictors. PLS is then applied to the augmented data set. The authors also show that there is no need to add cross-product terms, thus greatly reducing the number of new predictors added to the original data. Subsequently, Berglund et al. (2001) employ the use of the GIFI approach (Michailidis and de Leeuw 1998) which splits each predictor into two or more bins for those predictors that are thought to have a nonlinear relationship with the response. Cut points for the bins are selected by the user and are based on either prior knowledge or characteristics of the data. The original predictors that were binned are then excluded from the data set that includes the binned versions of the predictors. PLS is then applied to the new predictor set in usual way.

Both of these approaches have successfully found nonlinear relationships between the predictors and the response. But there can be a considerable amount of effort required in constructing the data sets for input to PLS, especially as the number of predictors becomes large. As we will show in subsequent sections, other predictive modeling techniques can more naturally identify nonlinear structures between predictors and the response without having to modify the predictor space. Therefore, if a more intricate relationship between predictors and response exists, then we suggest employing one of the other techniques rather than trying to improve the performance of PLS through this type of augmentation.

6.4 Penalized Models

Under standard assumptions, the coefficients produced by ordinary least squares regression are unbiased and, of all unbiased linear techniques, this model also has the lowest variance. However, given that the MSE is a

combination of variance and bias (Sect. 5.2), it is very possible to produce models with smaller MSEs by allowing the parameter estimates to be biased. It is common that a small increase in bias can produce a substantial drop in the variance and thus a smaller MSE than ordinary least squares regression coefficients. One consequence of large correlations between the predictor variances is that the variance can become very large. Combatting collinearity by using biased models may result in regression models where the overall MSE is competitive.

One method of creating biased regression models is to add a penalty to the sum of the squared errors. Recall that original least squares regression found parameter estimates to minimize the sum of the squared errors:

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

When the model over-fits the data, or when there are issues with collinearity (as in Table 6.1), the linear regression parameter estimates may become inflated. As such, we may want to control the magnitude of these estimates to reduce the SSE. Controlling (or *regularizing*) the parameter estimates can be accomplished by adding a penalty to the SSE if the estimates become large. *Ridge regression* (Hoerl 1970) adds a penalty on the sum of the squared regression parameters:

$$\text{SSE}_{L_2} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{P} \beta_j^2.$$

The "L_2" signifies that a second-order penalty (i.e., the square) is being used on the parameter estimates. The effect of this penalty is that the parameter estimates are only allowed to become large if there is a proportional reduction in SSE. In effect, this method *shrinks* the estimates towards 0 as the λ penalty becomes large (these techniques are sometimes called "shrinkage methods").

By adding the penalty, we are making a trade-off between the model variance and bias. By sacrificing some bias, we can often reduce the variance enough to make the overall MSE lower than unbiased models.

For example, Fig. 6.15 shows the *path* of the regression coefficients for the solubility data over different values of λ. Each line corresponds to a model parameter and the predictors were centered and scaled prior to this analysis so that their units are the same. When there is no penalty, many parameters have reasonable values, such as the predictor for the number of multiple bonds (shown in orange). However, some parameter estimates are abnormally large, such as the number of non-hydrogen atoms (in green) and the number of non-hydrogen bonds (purple) previously singled out in Table 6.1. These large values are indicative of collinearity issues. As the penalty is increased, the parameter estimates move closer to 0 at different rates. By the time

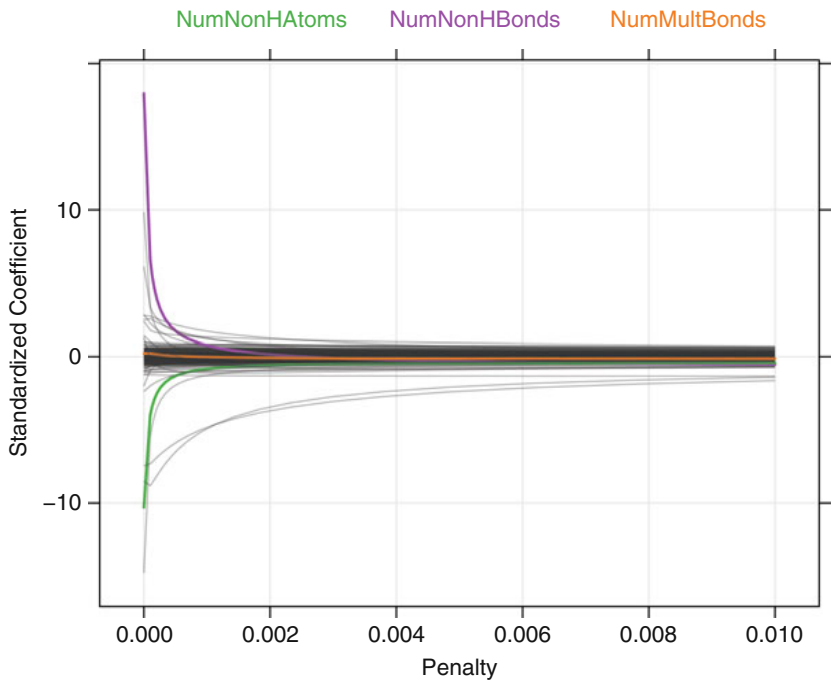

Fig. 6.15: The ridge-regression coefficient path

the penalty has a value of $\lambda = 0.002$, these two predictors are much more well behaved, although other coefficient values are still relatively large in magnitude.

Using cross-validation, the penalty value was optimized. Figure 6.16 shows how the RMSE changes with λ. When there is no penalty, the error is inflated. When the penalty is increased, the error drops from 0.72 to 0.69. As the penalty increases beyond 0.036, the bias becomes to large and the model starts to under-fit, resulting in an increase in MSE.

While ridge regression shrinks the parameter estimates towards 0, the model does not set the values to absolute 0 for any value of the penalty. Even though some parameter estimates become negligibly small, this model does not conduct *feature selection*.

A popular alternative to ridge regression is the *least absolute shrinkage and selection operator* model, frequently called the *lasso* (Tibshirani 1996). This model uses a similar penalty to ridge regression:

$$\text{SSE}_{L_1} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{P} |\beta_j|.$$

While this may seem like a small modification, the practical implications are significant. While the regression coefficients are still shrunk towards 0,

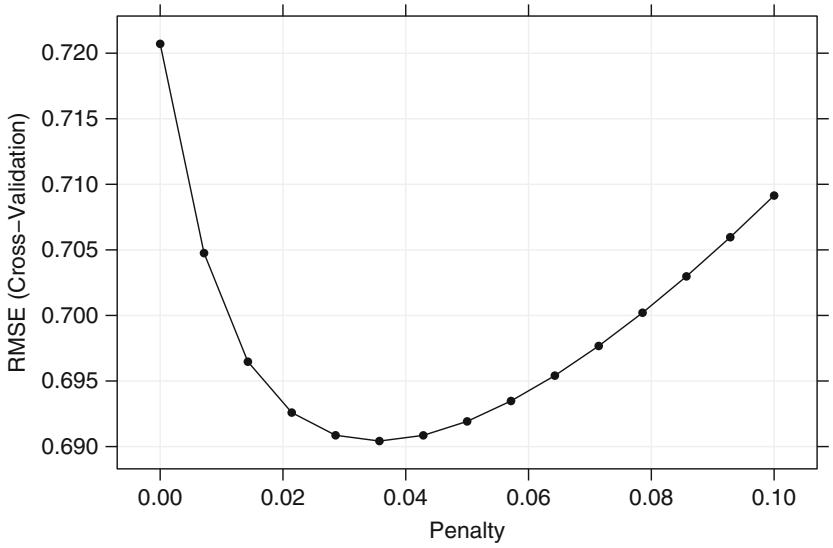

Fig. 6.16: The cross-validation profiles for a ridge regression model

a consequence of penalizing the absolute values is that some parameters are actually set to 0 for some value of λ. Thus the lasso yields models that simultaneously use regularization to improve the model and to conduct feature selection. In comparing, the two types of penalties, Friedman et al. (2010) stated

> "Ridge regression is known to shrink the coefficients of correlated predictors towards each other, allowing them to borrow strength from each other. In the extreme case of k identical predictors, they each get identical coefficients with $1/k$th the size that any single one would get if fit alone.[...]
>
> lasso, on the other hand, is somewhat indifferent to very correlated predictors, and will tend to pick one and ignore the rest."

Figure 6.17 shows the paths of the lasso coefficients over different penalty values. The x-axis is the fraction of the full solution (i.e., ordinary least squares with no penalty). Smaller values on the x-axis indicate that a large penalty has been used. When the penalty is large, many of the regression coefficients are set to 0. As the penalty is reduced, many have nonzero coefficients. Examining the trace for the number of non-hydrogen bonds (in purple), the coefficient is initially 0, has a slight increase, then is shrunken towards 0 again. When the fraction is around 0.4, this predictor is entered back into the model with a nonzero coefficient that consistently increases (most likely due to collinearity). Table 6.2 shows regression coefficients for ordinary least squares, PLS, ridge-regression, and the lasso model. The ridge-regression penalty used in this table is 0.036 and the lasso penalty was 0.15. The ridge-regression model shrinks the coefficients for the non-hydrogen atom and non-hydrogen bond predictors significantly towards 0 in comparison to

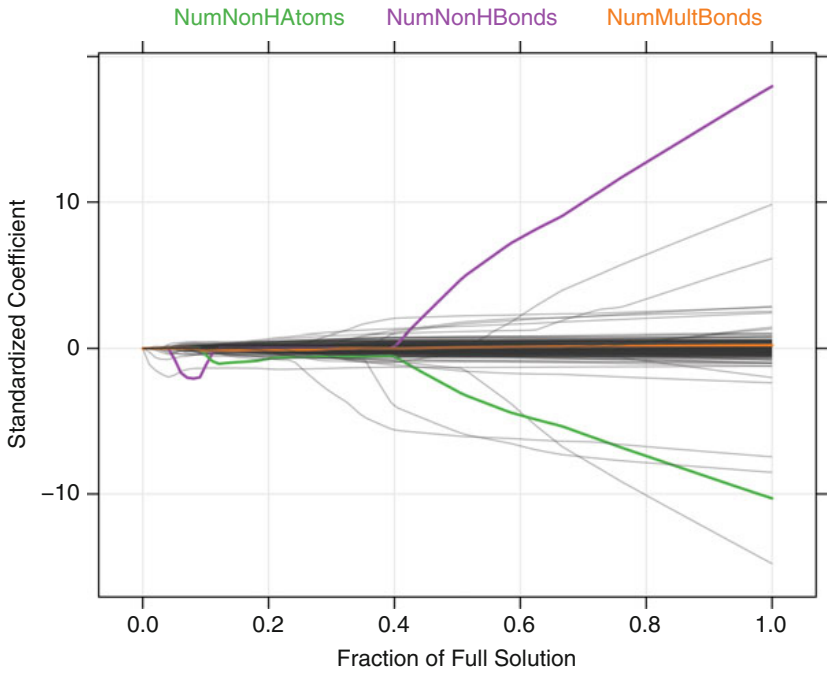

Fig. 6.17: The lasso coefficient path for the solubility data. The x-axis is the fraction of the full least squares solution. As the fraction increases, the lasso penalty (λ) decreases

the ordinary least squares models while the lasso model shrinks the non-hydrogen atom predictor out of the model. Between these models, the lasso model had the smallest cross-validation error of 0.67, slightly better than the PLS model (0.68) and ridge regression (0.69).

This type of regularization has been a very active area of research. The lasso model has been extended to many other techniques, such as linear discriminant analysis (Clemmensen et al. 2011; Witten and Tibshirani 2011), PLS (Chun and Keleş 2010), and PCA (Jolliffe et al. 2003; Zou et al. 2004). A significant advancement for this model was Efron et al. (2004). Their model, least angle regression (LARS), is a broad framework that encompasses the lasso and similar models. The LARS model can be used to fit lasso models more efficiently, especially in high-dimensional problems. Friedman et al. (2010) and Hesterberg et al. (2008) provide a survey of these techniques.

A generalization of the lasso model is the *elastic net* (Zou and Hastie 2005). This model combines the two types of penalties:

$$\text{SSE}_{\text{Enet}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{P} \beta_j^2 + \lambda_2 \sum_{j=1}^{P} |\beta_j|.$$

Table 6.2: Regression coefficients for two highly correlated predictors for PLS, ridge regression, the elastic net and other models

Model	NumNonHAtoms	NumNonHBonds
NumNonHAtoms only	−1.2 (0.1)	
NumNonHBonds only		−1.2 (0.1)
Both	−0.3 (0.5)	−0.9 (0.5)
All predictors	8.2 (1.4)	−9.1 (1.6)
PLS, all predictors	−0.4	−0.8
Ridge, all predictors	−0.3	−0.3
lasso/elastic net	0.0	−0.8

The ridge penalty used for this table was 0.036 and the lasso penalty was 0.15. The PLS model used ten components.

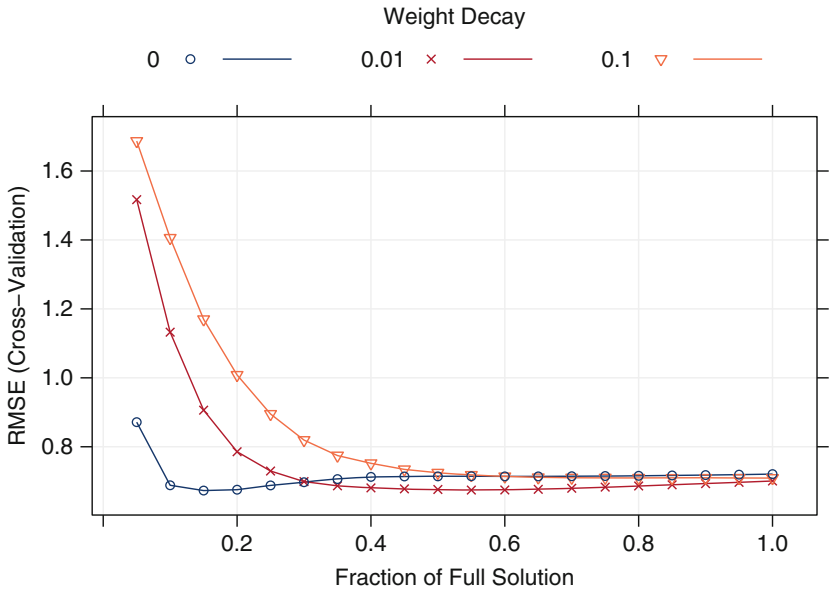

Fig. 6.18: The cross-validation profiles for an elastic net model

The advantage of this model is that it enables effective regularization via the ridge-type penalty with the feature selection quality of the lasso penalty. The Zou and Hastie (2005) suggest that this model will more effectively deal with groups of high correlated predictors.

Both the penalties require tuning to achieve optimal performance. Again, using resampling, this model was tuned for the solubility data. Figure 6.18 shows the performance profiles across three values of the ridge penalty and 20 values of the lasso penalty. The pure lasso model (with $\lambda_1 = 0$) has an initial

drop in the error and then an increase when the fraction is greater than 0.2. The two models with nonzero values of the ridge penalty have minimum errors with a larger model. In the end, the optimal performance was associated with the lasso model with a fraction of 0.15, corresponding to 130 predictors out of a possible 228.

6.5 Computing

The R packages elasticnet, caret, lars, MASS, pls and stats will be referenced.

The solubility data can be obtained from the AppliedPredictiveModeling R package. The predictors for the training and test sets are contained in data frames called `solTrainX` and `solTestX`, respectively. To obtain the data in R,

```
> library(AppliedPredictiveModeling)
> data(solubility)
> ## The data objects begin with "sol":
> ls(pattern = "^solT")
  [1] "solTestX"       "solTestXtrans"  "solTestY"       "solTrainX"
  [5] "solTrainXtrans" "solTrainY"
```

Each column of the data corresponds to a predictor (i.e., chemical descriptor) and the rows correspond to compounds. There are 228 columns in the data. A random sample of column names is

```
> set.seed(2)
> sample(names(solTrainX), 8)
  [1] "FP043"        "FP160"        "FP130"        "FP038"        "NumBonds"
  [6] "NumNonHAtoms" "FP029"        "FP185"
```

The "`FP`" columns correspond to the binary 0/1 fingerprint predictors that are associated with the presence or absence of a particular chemical structure. Alternate versions of these data that have been Box–Cox transformed are contained in the data frames `solTrainXtrans` and `solTestXtrans`. These modified versions were used in the analyses in this and subsequent chapters.

The solubility values for each compound are contained in numeric vectors named `solTrainY` and `solTestY`.

Ordinary Linear Regression

The primary function for creating linear regression models using simple least squares is `lm`. This function takes a formula and data frame as input. Because of this, the training set predictors and outcome should be contained in the same data frame. We can create a new data frame for this purpose:

```
> trainingData <- solTrainXtrans
> ## Add the solubility outcome
> trainingData$Solubility <- solTrainY
```

To fit a linear model with all the predictors entering in the model as simple, independent linear terms, the formula shortcut `Solubility ~ .` can be used:

```
> lmFitAllPredictors <- lm(Solubility ~ ., data = trainingData)
```

An intercept term is automatically added to the model. The `summary` method displays model summary statistics, the parameter estimates, their standard errors, and p-values for testing whether each individual coefficient is different than 0:

```
> summary(lmFitAllPredictors)
```

```
Call:
lm(formula = Solubility ~ ., data = trainingData)

Residuals:
     Min       1Q   Median       3Q      Max
-1.75620 -0.28304  0.01165  0.30030  1.54887

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        2.431e+00  2.162e+00   1.124 0.261303
FP001              3.594e-01  3.185e-01   1.128 0.259635
FP002              1.456e-01  2.637e-01   0.552 0.580960
FP003             -3.969e-02  1.314e-01  -0.302 0.762617
FP004             -3.049e-01  1.371e-01  -2.223 0.026520 *
FP005              2.837e+00  9.598e-01   2.956 0.003223 **
FP006             -6.886e-02  2.041e-01  -0.337 0.735917
FP007              4.044e-02  1.152e-01   0.351 0.725643
FP008              1.121e-01  1.636e-01   0.685 0.493331
FP009             -8.242e-01  8.395e-01  -0.982 0.326536
   :                     :          :          :     :
MolWeight         -1.232e+00  2.296e-01  -5.365 1.09e-07 ***
NumAtoms          -1.478e+01  3.473e+00  -4.257 2.35e-05 ***
NumNonHAtoms       1.795e+01  3.166e+00   5.670 2.07e-08 ***
NumBonds           9.843e+00  2.681e+00   3.671 0.000260 ***
NumNonHBonds      -1.030e+01  1.793e+00  -5.746 1.35e-08 ***
NumMultBonds       2.107e-01  1.754e-01   1.201 0.229990
NumRotBonds       -5.213e-01  1.334e-01  -3.908 0.000102 ***
NumDblBonds       -7.492e-01  3.163e-01  -2.369 0.018111 *
NumAromaticBonds  -2.364e+00  6.232e-01  -3.794 0.000161 ***
NumHydrogen        8.347e-01  1.880e-01   4.439 1.04e-05 ***
NumCarbon          1.730e-02  3.763e-01   0.046 0.963335
NumNitrogen        6.125e+00  3.045e+00   2.011 0.044645 *
NumOxygen          2.389e+00  4.523e-01   5.283 1.69e-07 ***
NumSulfer         -8.508e+00  3.619e+00  -2.351 0.018994 *
NumChlorine       -7.449e+00  1.989e+00  -3.744 0.000195 ***
NumHalogen         1.408e+00  2.109e+00   0.668 0.504615
NumRings           1.276e+00  6.716e-01   1.901 0.057731 .
HydrophilicFactor  1.099e-02  1.137e-01   0.097 0.922998
```

```
SurfaceArea1       8.825e-02  6.058e-02   1.457 0.145643
SurfaceArea2       9.555e-02  5.615e-02   1.702 0.089208 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5524 on 722 degrees of freedom
Multiple R-squared: 0.9446,        Adjusted R-squared: 0.9271
F-statistic: 54.03 on 228 and 722 DF,  p-value: < 2.2e-16
```

(Since there are 229 predictors in the model, the output is very long and the results have been trimmed.) A more comprehensive discussion of linear models in R can be found in Faraway (2005).

The simple estimates of the RMSE and R^2 were 0.55 and 0.945, respectively. Note that these values are likely to be highly optimistic as they have been derived by re-predicting the training set data.

To compute the model solubility values for new samples, the `predict` method is used:

```
> lmPred1 <- predict(lmFitAllPredictors, solTestXtrans)
> head(lmPred1)
          20           21           23           25           28           31
 0.99370933  0.06834627 -0.69877632  0.84796356 -0.16578324  1.40815083
```

We can collect the observed and predicted values into a data frame, then use the caret function `defaultSummary` to estimate the test set performance:

```
> lmValues1 <- data.frame(obs = solTestY, pred = lmPred1)
> defaultSummary(lmValues1)
      RMSE  Rsquared
 0.7455802 0.8722236
```

Based on the test set, the summaries produced by the summary function for `lm` were optimistic.

If we wanted a robust linear regression model, then the robust linear model function (`rlm`) from the MASS package could be used, which by default employs the Huber approach. Similar to the `lm` function, `rlm` is called as follows:

```
> rlmFitAllPredictors <- rlm(Solubility ~ ., data = trainingData)
```

The `train` function generates a resampling estimate of performance. Because the training set size is not small, 10-fold cross-validation should produce reasonable estimates of model performance. The function `trainControl` specifies the type of resampling:

```
> ctrl <- trainControl(method = "cv", number = 10)
```

`train` will accept a model formula or a non-formula interface (see Sect. 4.9 for a summary of different methods for specifying predictor models). The non-formula interface is

```
> set.seed(100)
> lmFit1 <- train(x = solTrainXtrans, y = solTrainY,
+                 method = "lm", trControl = ctrl)
```

The random number seed is set prior to modeling so that the results can be reproduced. The results are:

```
> lmFit1
  951 samples
  228 predictors

  No pre-processing
  Resampling: Cross-Validation (10-fold)

  Summary of sample sizes: 856, 857, 855, 856, 856, 855, ...

  Resampling results

    RMSE   Rsquared  RMSE SD  Rsquared SD
    0.721  0.877     0.07     0.0247
```

For models built to *explain*, it is important to check model assumptions, such as the residual distribution. For predictive models, some of the same diagnostic techniques can shed light on areas where the model is not predicting well. For example, we could plot the residuals versus the predicted values for the model. If the plot shows a random cloud of points, we will feel more comfortable that there are no major terms missing from the model (such as quadratic terms, etc.) or significant outliers. Another important plot is the predicted values versus the observed values to assess how close the predictions are to the actual values. Two methods of doing this (using the training set samples are

```
> xyplot(solTrainY ~ predict(lmFit1),
+        ## plot the points (type = 'p') and a background grid ('g')
+        type = c("p", "g"),
+        xlab = "Predicted", ylab = "Observed")
> xyplot(resid(lmFit1) ~ predict(lmFit1),
+        type = c("p", "g"),
+        xlab = "Predicted", ylab = "Residuals")
```

The results are shown in Fig. 6.19. Note that the `resid` function generates the model residuals for the training set and that using the `predict` function without an additional data argument returns the predicted values for the training set. For this model, there are no obvious warning signs in the diagnostic plots.

To build a smaller model without predictors with extremely high correlations, we can use the methods of Sect. 3.3 to reduce the number of predictors such that there are no absolute pairwise correlations above 0.9:

```
> corThresh <- .9
> tooHigh <- findCorrelation(cor(solTrainXtrans), corThresh)
> corrPred <- names(solTrainXtrans)[tooHigh]
> trainXfiltered <- solTrainXtrans[, -tooHigh]
```

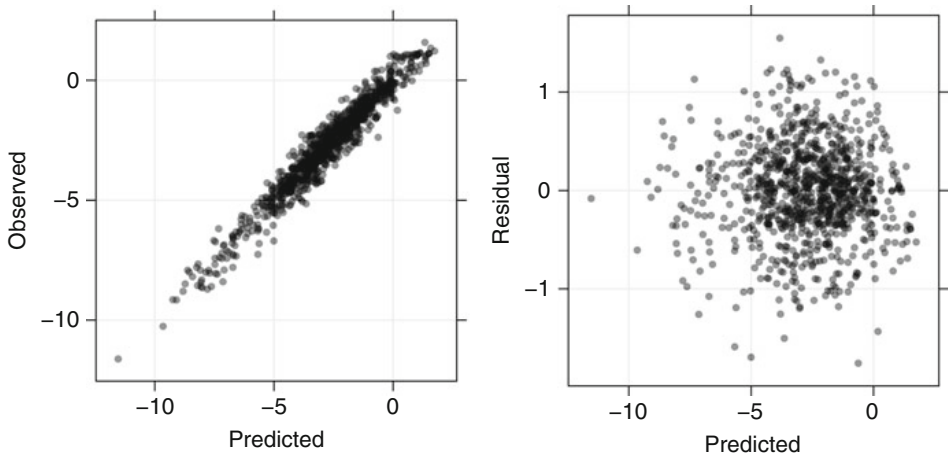

Fig. 6.19: Diagnostic plots for the linear model using the training set. *Left*: A plot of the observed values versus the predicted values. This plot can show outliers or areas where the model is not calibrated. *Right*: A plot of the residuals versus predicted values. If the model has been well specified, this plot should be a random cloud of points with no outliers or patterns (e.g., a funnel shape)

```
> testXfiltered <- solTestXtrans[, -tooHigh]
> set.seed(100)
> lmFiltered <- train(solTrainXtrans, solTrainY, method = "lm",
+                     trControl = ctrl)
> lmFiltered
  951 samples
  228 predictors

  No pre-processing
  Resampling: Cross-Validation (10-fold)

  Summary of sample sizes: 856, 857, 855, 856, 856, 855, ...

  Resampling results

    RMSE   Rsquared  RMSE SD  Rsquared SD
    0.721  0.877     0.07     0.0247
```

Robust linear regression can also be performed using the `train` function which employs the `rlm` function. However, it is important to note that `rlm` does not allow the covariance matrix of the predictors to be singular (unlike the `lm` function). To ensure that predictors are not singular, we will pre-process the predictors using PCA. Using the filtered set of predictors, the robust regression model performance is

```
> set.seed(100)
> rlmPCA <- train(solTrainXtrans, solTrainY,
```

```
+                 method = "rlm",
+                 preProcess = "pca",
+                 trControl = ctrl)
> rlmPCA
  951 samples
  228 predictors

  Pre-processing: principal component signal extraction, scaled, centered
  Resampling: Cross-Validation (10-fold)

  Summary of sample sizes: 856, 857, 855, 856, 856, 855, ...

  Resampling results

    RMSE   Rsquared  RMSE SD  Rsquared SD
    0.782  0.854     0.0372   0.0169
```

Partial Least Squares

The pls package (Mevik and Wehrens 2007) has functions for PLS and PCR. SIMPLS, the first Dayal and MacGregor algorithm, and the algorithm developed by Rännar et al. (1994) are each available. By default, the pls package uses the first Dayal and MacGregor kernel algorithm while the other algorithms can be specified using the `method` argument using the values `"oscorespls"`, `"simpls"`, or `"widekernelpls"`. The `plsr` function, like the `lm` function, requires a model formula:

```
> plsFit <- plsr(Solubility ~ ., data = trainingData)
```

The number of components can be fixed using the `ncomp` argument or, if left to the default, the maximum number of components will be calculated. Predictions on new samples can be calculated using the `predict` function. Predictions can be made for a specific number of components or for several values at a time. For example

```
> predict(plsFit, solTestXtrans[1:5,], ncomp = 1:2)
  , , 1 comps

     Solubility
  20  -1.789335
  21  -1.427551
  23  -2.268798
  25  -2.269782
  28  -1.867960

  , , 2 comps

     Solubility
```

```
20  0.2520469
21  0.3555028
23 -1.8795338
25 -0.6848584
28 -1.5531552
```

The `plsr` function has options for either K-fold or leave-one-out cross-validation (via the `validation` argument) or the PLS algorithm to use, such as SIMPLS (using the `method` argument).

There are several helper functions to extract the PLS components (in the function `loadings`), the PLS scores (`scores`), and other quantities. The `plot` function has visualizations for many aspects of the model.

`train` can also be used with `method` values of `pls`, such as `"oscorespls"`, `"simpls"`, or `"widekernelpls"`. For example

```
> set.seed(100)
> plsTune <- train(solTrainXtrans, solTrainY,
+                  method = "pls",
+                  ## The default tuning grid evaluates
+                  ## components 1... tuneLength
+                  tuneLength = 20,
+                  trControl = ctrl,
+                  preProc = c("center", "scale"))
```

This code reproduces the PLS model displayed in Fig. 6.11.

Penalized Regression Models

Ridge-regression models can be created using the `lm.ridge` function in the MASS package or the `enet` function in the elasticnet package. When calling the `enet` function, the `lambda` argument specifies the ridge-regression penalty:

```
> ridgeModel <- enet(x = as.matrix(solTrainXtrans), y = solTrainY,
+                    lambda = 0.001)
```

Recall that the elastic net model has both ridge penalties and lasso penalties and, at this point, the R object `ridgeModel` has only fixed the ridge penalty value. The lasso penalty can be computed efficiently for many values of the penalty. The `predict` function for `enet` objects generates predictions for one or more values of the lasso penalty simultaneously using the `s` and `mode` arguments. For ridge regression, we only desire a single lasso penalty of 0, so we want the full solution. To produce a ridge-regression solution we define `s=1` with `mode = "fraction"`. This last option specifies how the amount of penalization is defined; in this case, a value of 1 corresponds to a faction of 1, i.e., the full solution:

```
> ridgePred <- predict(ridgeModel, newx = as.matrix(solTestXtrans),
+                      s = 1, mode = "fraction",
```

```
+                       type = "fit")
> head(ridgePred$fit)
          20          21          23          25          28          31
 0.96795590  0.06918538 -0.54365077  0.96072014 -0.03594693  1.59284535
```

To tune over the penalty, `train` can be used with a different method:

```
> ## Define the candidate set of values
> ridgeGrid <- data.frame(.lambda = seq(0, .1, length = 15))
> set.seed(100)
> ridgeRegFit <- train(solTrainXtrans, solTrainY,
+                       method = "ridge",
+                       ## Fir the model over many penalty values
+                       tuneGrid = ridgeGrid,
+                       trControl = ctrl,
+                       ## put the predictors on the same scale
+                       preProc = c("center", "scale"))

> ridgeRegFit
  951 samples
  228 predictors

  Pre-processing: centered, scaled
  Resampling: Cross-Validation (10-fold)

  Summary of sample sizes: 856, 857, 855, 856, 856, 855, ...

  Resampling results across tuning parameters:

    lambda   RMSE   Rsquared  RMSE SD  Rsquared SD
    0        0.721  0.877     0.0699   0.0245
    0.00714  0.705  0.882     0.045    0.0199
    0.0143   0.696  0.885     0.0405   0.0187
    0.0214   0.693  0.886     0.0378   0.018
    0.0286   0.691  0.887     0.0359   0.0175
    0.0357   0.69   0.887     0.0346   0.0171
    0.0429   0.691  0.888     0.0336   0.0168
    0.05     0.692  0.888     0.0329   0.0166
    0.0571   0.693  0.887     0.0323   0.0164
    0.0643   0.695  0.887     0.032    0.0162
    0.0714   0.698  0.887     0.0319   0.016
    0.0786   0.7    0.887     0.0318   0.0159
    0.0857   0.703  0.886     0.0318   0.0158
    0.0929   0.706  0.886     0.032    0.0157
    0.1      0.709  0.885     0.0321   0.0156

  RMSE was used to select the optimal model using  the smallest value.
  The final value used for the model was lambda = 0.0357.
```

The lasso model can be estimated using a number of different functions. The lars package contains the `lars` function, the elasticnet package has `enet`, and the glmnet package has a function of the same name. The syntax for these functions is very similar. For the `enet` function, the usage would be

```
> enetModel <- enet(x = as.matrix(solTrainXtrans), y = solTrainY,
+                   lambda = 0.01, normalize = TRUE)
```

The predictor data must be a matrix object, so the data frame `solTrainXtrans` needs to be converted for the `enet` function. The predictors should be centered and scaled prior to modeling. The `normalize` argument will do this standardization automatically. The parameter `lambda` controls the ridge-regression penalty and, setting this value to 0, fits the lasso model. The lasso penalty does not need to be specified until the time of prediction:

```
> enetPred <- predict(enetModel, newx = as.matrix(solTestXtrans),
+                     s = .1, mode = "fraction",
+                     type = "fit")
> ## A list is returned with several items:
> names(enetPred)
  [1] "s"        "fraction" "mode"     "fit"
> ## The 'fit' component has the predicted values:
> head(enetPred$fit)
          20          21          23          25          28          31
-0.60186178 -0.42226814 -1.20465564 -1.23652963 -1.25023517 -0.05587631
```

To determine which predictors are used in the model, the `predict` method is used with `type = "coefficients"`:

```
> enetCoef<- predict(enetModel, newx = as.matrix(solTestXtrans),
+                     s = .1, mode = "fraction",
+                     type = "coefficients")
> tail(enetCoef$coefficients)
       NumChlorine        NumHalogen          NumRings HydrophilicFactor
         0.00000000         0.00000000        0.00000000        0.12678967
      SurfaceArea1      SurfaceArea2
         0.09035596         0.00000000
```

More than one value of `s` can be used with the `predict` function to generate predictions from more than one model simultaneously.

Other packages to fit the lasso model or some alternate version of the model are biglars (for large data sets), FLLat (for the fused lasso), grplasso (the group lasso), penalized, relaxo (the relaxed lasso), and others. To tune the elastic net model using `train`, we specify `method = "enet"`. Here, we tune the model over a custom set of penalties:

```
> enetGrid <- expand.grid(.lambda = c(0, 0.01, .1),
+                         .fraction = seq(.05, 1, length = 20))
> set.seed(100)
> enetTune <- train(solTrainXtrans, solTrainY,
+                   method = "enet",
+                   tuneGrid = enetGrid,
+                   trControl = ctrl,
+                   preProc = c("center", "scale"))
```

Figure 6.18 can be created from this object using `plot(enetTune)`.

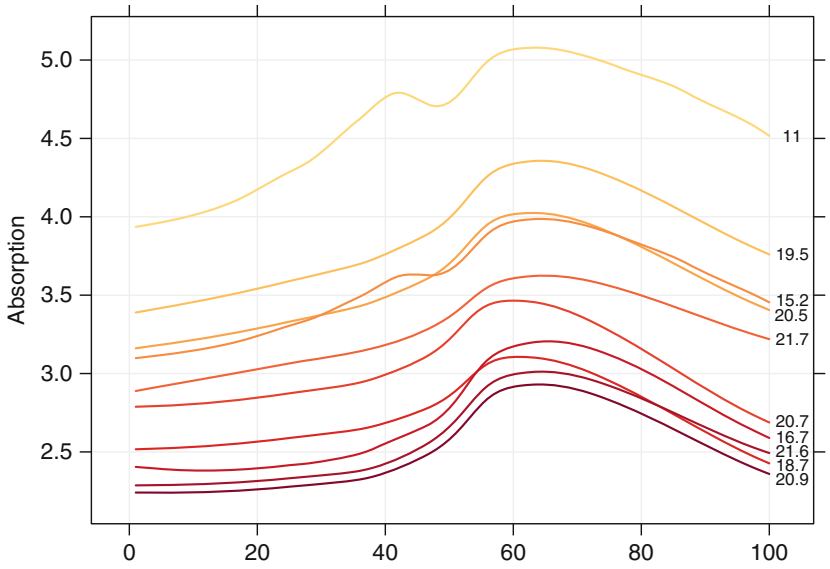

Fig. 6.20: A sample of ten spectra of the Tecator data. The colors of the curves reflect the absorption values, where *yellow* indicates low absorption and *red* is indicative of high absorption

Exercises

6.1. Infrared (IR) spectroscopy technology is used to determine the chemical makeup of a substance. The theory of IR spectroscopy holds that unique molecular structures absorb IR frequencies differently. In practice a spectrometer fires a series of IR frequencies into a sample material, and the device measures the absorbance of the sample at each individual frequency. This series of measurements creates a spectrum profile which can then be used to determine the chemical makeup of the sample material.

A Tecator Infratec Food and Feed Analyzer instrument was used to analyze 215 samples of meat across 100 frequencies. A sample of these frequency profiles is displayed in Fig. 6.20. In addition to an IR profile, analytical chemistry determined the percent content of water, fat, and protein for each sample. If we can establish a predictive relationship between IR spectrum and fat content, then food scientists could predict a sample's fat content with IR instead of using analytical chemistry. This would provide costs savings, since analytical chemistry is a more expensive, time-consuming process:

(a) Start R and use these commands to load the data:

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

The matrix `absorp` contains the 100 absorbance values for the 215 samples, while matrix `endpoints` contains the percent of moisture, fat, and protein in columns 1–3, respectively.

(b) In this example the predictors are the measurements at the individual frequencies. Because the frequencies lie in a systematic order (850–1,050 nm), the predictors have a high degree of correlation. Hence, the data lie in a smaller dimension than the total number of predictors (215). Use PCA to determine the effective dimension of these data. What is the effective dimension?

(c) Split the data into a training and a test set, pre-process the data, and build each variety of models described in this chapter. For those models with tuning parameters, what are the optimal values of the tuning parameter(s)?

(d) Which model has the best predictive ability? Is any model significantly better or worse than the others?

(e) Explain which model you would use for predicting the fat content of a sample.

6.2. Developing a model to predict permeability (see Sect. 1.4) could save significant resources for a pharmaceutical company, while at the same time more rapidly identifying molecules that have a sufficient permeability to become a drug:

(a) Start R and use these commands to load the data:

```
> library(AppliedPredictiveModeling)
> data(permeability)
```

The matrix `fingerprints` contains the 1,107 binary molecular predictors for the 165 compounds, while `permeability` contains permeability response.

(b) The fingerprint predictors indicate the presence or absence of substructures of a molecule and are often sparse meaning that relatively few of the molecules contain each substructure. Filter out the predictors that have low frequencies using the `nearZeroVar` function from the caret package. How many predictors are left for modeling?

(c) Split the data into a training and a test set, pre-process the data, and tune a PLS model. How many latent variables are optimal and what is the corresponding resampled estimate of R^2?

(d) Predict the response for the test set. What is the test set estimate of R^2?

(e) Try building other models discussed in this chapter. Do any have better predictive performance?

(f) Would you recommend any of your models to replace the permeability laboratory experiment?

6.3. A chemical manufacturing process for a pharmaceutical product was discussed in Sect. 1.4. In this problem, the objective is to understand the relationship between biological measurements of the raw materials (predictors),

measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1 % will boost revenue by approximately one hundred thousand dollars per batch:

(a) Start R and use these commands to load the data:

```
> library(AppliedPredictiveModeling)
> data(chemicalManufacturing)
```

The matrix `processPredictors` contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. `yield` contains the percent yield for each run.

(b) A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values (e.g., see Sect. 3.8).

(c) Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter. What is the optimal value of the performance metric?

(d) Predict the response for the test set. What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

(e) Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

(f) Explore the relationships between each of the top predictors and the response. How could this information be helpful in improving yield in future runs of the manufacturing process?