4 Shrinkage Methods

The subset selection methods discussed previously involve using least squares to fit a linear model that contains a subset of the predictors. As an alternative, we can fit a model containing all p predictors using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero. It turns out that shrinking the coefficient estimates can significantly reduce their variance. The two best-known techniques for shrinking the regression coefficients towards zero are ridge regression and the lasso.

4.1 Ridge Regression

Recall that the least squares fitting procedure estimates $\beta_0, \beta_1, \ldots, \beta_p$ using the values that minimize

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

Ridge regression is very similar to least squares, except that the coefficients are estimated by minimizing a slightly different quantity. In particular, the ridge regression coefficient estimates $\hat{\beta}^R$ are the values that minimize

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2, \tag{26}$$

where $\lambda \geq 0$ is a tuning parameter, to be determined separately. Equation (26) trades off two different criteria. As with least squares, ridge regression seeks coefficient estimates that fit the data well, by making the RSS small. However, the second term, $\lambda \sum_{j=1}^{p} \beta_j^2$, called a shrinkage penalty, is small when $\beta_1, \ldots, \beta_p$ are close to zero, and so it has the effect of shrinking the estimates of β_j towards zero. The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates. When $\lambda = 0$, the penalty term has no effect, and ridge regression will produce the least squares estimates. However, as $\lambda \to \infty$, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero. Unlike least squares, which generates only one set of coefficient estimates, ridge regression will produce a different set of coefficient estimates, $\hat{\beta}^R_\lambda$, for each value of λ. Selecting a good value for λ is critical; we defer this discussion to Section 4.3, where we use cross-validation.

Note that in (26), the shrinkage penalty is applied to $\beta_1, \ldots, \beta_p$, but not to the intercept β_0. We want to shrink the estimated association of each variable with the response; however, we do not want to shrink the intercept, which is simply a measure of the mean value of the response when $x_{i1} = x_{i2} = \ldots = x_{ip} = 0$.

An Application to the Credit Data

In Figure 16, the ridge regression coefficient estimates for the `Credit` data set are displayed. Each curve corresponds to the ridge regression coefficient estimate for one of the ten variables, plotted as a function of λ. For example, the black solid line represents the ridge regression estimate for the `income` coefficient, as λ is varied. At the extreme left-hand side of the plot, λ is essentially zero, and so the corresponding ridge coefficient estimates are the same as the usual least squares estimates. But as λ increases, the ridge coefficient estimates shrink towards zero. When λ is extremely large, then all of the ridge coefficient estimates are basically zero; this corresponds to the null model that contains no predictors. In this plot, the `income`, `limit`, `rating`, and `student` variables are displayed in distinct colors, since these variables tend to have by far the largest coefficient estimates. While the ridge coefficient estimates tend to decrease in aggregate as λ increases, individual coefficients, such as `rating` and `income`, may occasionally increase as λ increases.

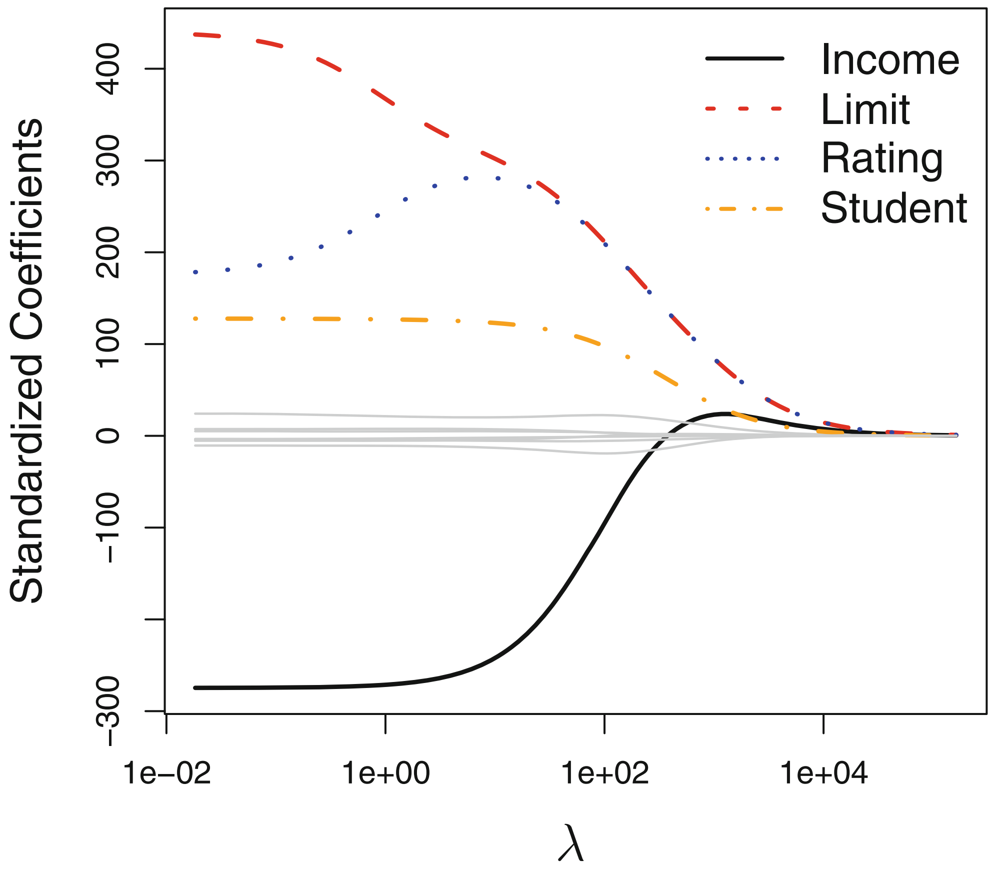

Figure 16: The standardized ridge regression coefficients are displayed for the `Credit` data set, as a function of λ.

The standard least squares coefficient estimates are scale equivariant: multiplying X_j by a constant c simply leads to a scaling of the least squares coefficient estimates by a factor of $1/c$. In other words, regardless of how the jth predictor is scaled, $X_j\hat{\beta}_j$ will remain the same. In contrast, the ridge regression coefficient estimates can change substantially when multiplying a given predictor by a constant. For instance, consider the `income` variable, which is measured in dollars. One could reasonably have measured income in thousands of dollars, which would result in a reduction in the observed values of income by a factor of 1,000. Now due to the sum of squared coefficients term in the ridge regression formulation (26), such a change in scale will not simply cause the ridge regression coefficient estimate for income to change by a factor of 1,000. In other words, $X_j\hat{\beta}^R_{j,\lambda}$ will depend not only on the value of λ, but also on the scaling of the jth predictor. In fact, the value of $X_j\hat{\beta}^R_{j,\lambda}$ may even depend on the scaling of the other predictors! Therefore, it is best to apply ridge regression after standardizing the predictors, using the formula

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)^2}},$$

so that they are all on the same scale. In this formula the denominator is the estimated standard deviation of the jth predictor. Consequently, all of the standardized predictors will have a standard deviation of one. As a result the final fit will not depend on the scale on which the predictors are measured. In Figure 16, the y-axis displays the standardized ridge regression coefficient estimates - that is, the coefficient estimates that result from performing ridge regression using standardized predictors.

Why Does Ridge Regression Improve Over Least Squares?

Ridge regression's advantage over least squares is rooted in the bias-variance trade-off. As λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias. This is illustrated in Figure 17, using a simulated data set containing $p = 45$ predictors and $n = 50$ observations. The green curve displays the variance of the ridge regression predictions as a function of λ. At the least squares coefficient estimates, which correspond to ridge regression with $\lambda = 0$, the variance is high but there is no bias. But as λ increases, the shrinkage of the ridge coefficient estimates leads to a substantial reduction in the variance of the predictions, at the expense of a slight increase in bias. Recall that the test mean squared error (MSE), plotted in purple, is a function of the variance plus the squared bias bias. For values of λ up to about 10, the variance decreases rapidly, with very little increase in bias, plotted in black. Consequently, the MSE drops considerably as λ increases from 0 to 10. Beyond this point, the decrease in variance due to increasing λ slows, and the shrinkage on the coefficients causes them to be significantly underestimated, resulting in a large increase in the bias. The minimum MSE is achieved at approximately $\lambda = 30$. Interestingly, because of its high variance, the MSE associated with the least squares fit, when $\lambda = 0$, is almost as high as that of the null model for which all coefficient estimates are zero, when $\lambda = \infty$. However, for an intermediate value of λ, the MSE is considerably lower.

In general, in situations where the relationship between the response and the predictors is close to linear, the least squares estimates will have low bias but may have high variance. This means that a small change in the data can cause a large change in the least squares coefficient estimates. In particular, when the number of variables p is almost as large as the number of observations n, as in the example in Figure 17, the least squares estimates will be extremely variable. And if $p > n$, then the least squares estimates do not even have a unique solution, whereas ridge regression can still perform well by trading off a small increase in bias for a large decrease in variance. Hence, ridge regression works best in situations where the least squares estimates have high variance.

Ridge regression also has substantial computational advantages over best subset selection, which requires searching through 2^p models. Even for moderate values of p, such a search can be computationally infeasible. In contrast, for any fixed value of λ, ridge regression only fits a single model, and the model-fitting procedure can be performed quite quickly.

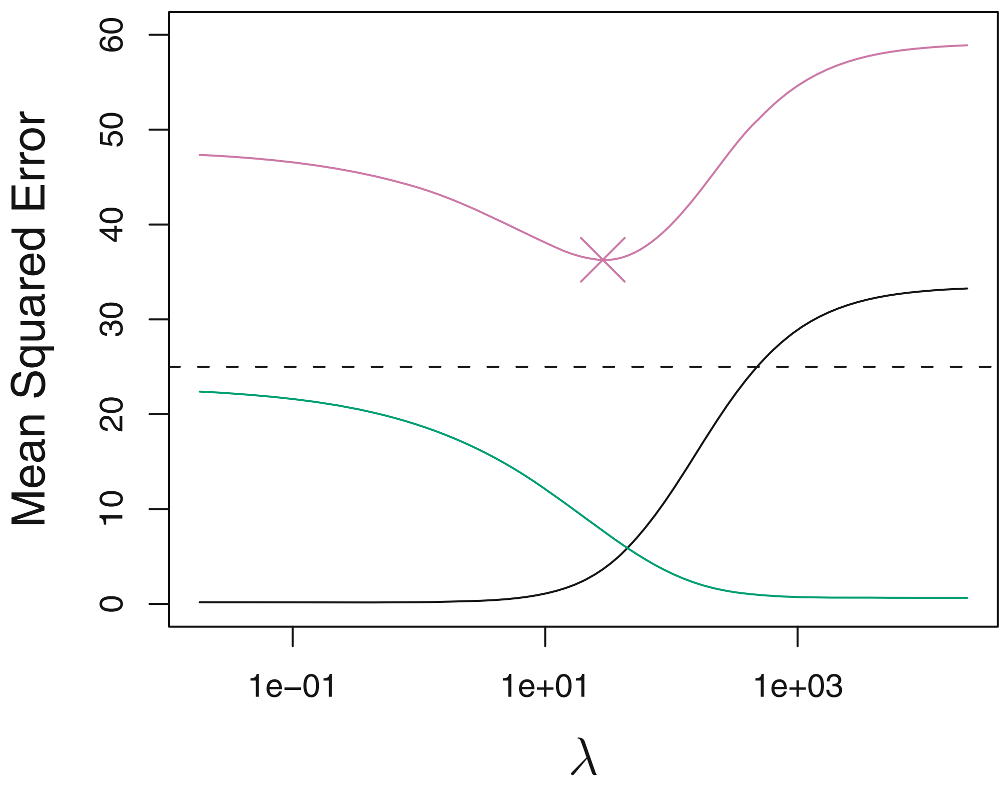

Figure 17: Squared bias (black), variance (green), and test mean squared error (purple) for the ridge regression predictions on a simulated data set, as a function of λ. The horizontal dashed line indicates the minimum possible MSE. The purple crosses indicate the ridge regression models for which the MSE is smallest.

4.2 The Lasso

Ridge regression does have one obvious disadvantage. Unlike best subset, forward stepwise, and backward stepwise selection, which will generally select models that involve just a subset of the variables, ridge regression will include all p predictors in the final model. The penalty $\lambda \sum \beta_j^2$ in (26) will shrink all of the coefficients towards zero, but it will not set any of them exactly to zero (unless $\lambda = \infty$). This may not be a problem for prediction accuracy, but it can create a challenge in model interpretation in settings in which the number of variables p is quite large. For example, in the `Credit` data set, it appears that the most important variables are `income`, `limit`, `rating`, and `student`. So we might wish to build a model including just these predictors. However, ridge regression will always generate a model involving all ten predictors. Increasing the value of λ will tend to reduce the magnitudes of the coefficients, but will not result in exclusion of any of the variables.

The lasso is an alternative to ridge regression that overcomes this disadvantage. The estimated lasso coefficients, $\hat{\beta}_\lambda^L$, minimize the quantity

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \text{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|\,. \tag{27}$$

Comparing (26) to (27), we see that the lasso and ridge regression have similar formulations. The only difference is that the β_j^2 term in the ridge regression penalty (26) has been replaced by $|\beta_j|$ in the lasso penalty (27).

As with ridge regression, the lasso shrinks the coefficient estimates towards zero. However, in the case of the lasso, the penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large. Hence, much like best subset selection, the lasso performs variable selection. As a result, models generated from the lasso are generally much easier to interpret than those produced by ridge regression. We say that the lasso yields sparse models - that is, models that involve only a subset of the variables. As in ridge regression, selecting a good value of λ for the lasso is critical; we defer this discussion to Section 4.3, where we use cross-validation.

As an example, consider the coefficient plots in Figure 18, which are generated from applying the lasso to the `Credit` data set. When $\lambda = 0$, then the lasso simply gives the least squares fit, and when λ becomes sufficiently large, the lasso gives the null model in which all coefficient estimates equal zero. However, in between these two extremes, the ridge regression and lasso models are quite different from each other. Moving from right to left of Figure 18, we observe that at first the lasso results in a model that contains only the `rating` predictor. Then `student` and `limit` enter the model almost simultaneously, shortly followed by `income`. Eventually, the remaining variables enter the model. Hence, depending on the value of λ, the lasso can produce a model involving any number of variables. In contrast, ridge regression will always include all of the variables in the model, although the magnitude of the coefficient estimates will depend on λ.

Figure 19 displays the variance, squared bias, and test MSE of the lasso applied to the same simulated data as in Figure 17. Clearly the lasso leads to qualitatively similar behavior to ridge regression, in that as λ increases, the variance decreases and the bias increases.

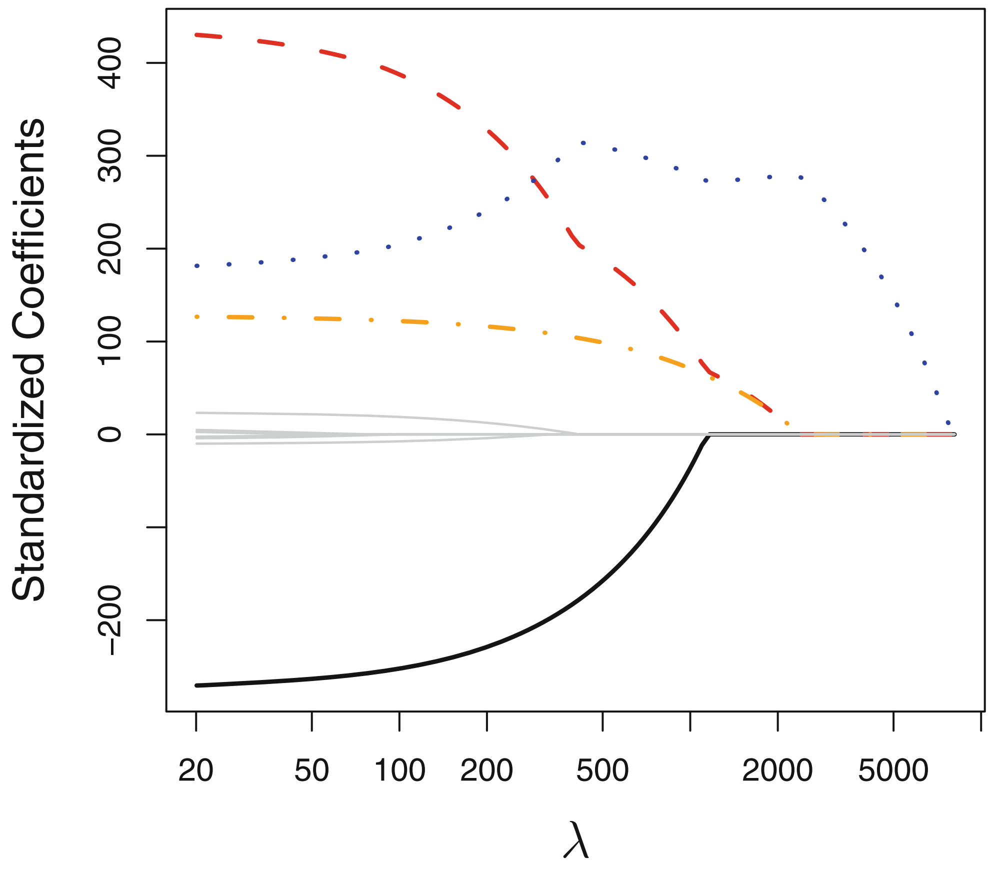

Figure 18: The standardized lasso coefficients on the `Credit` data set are shown as a function of λ.

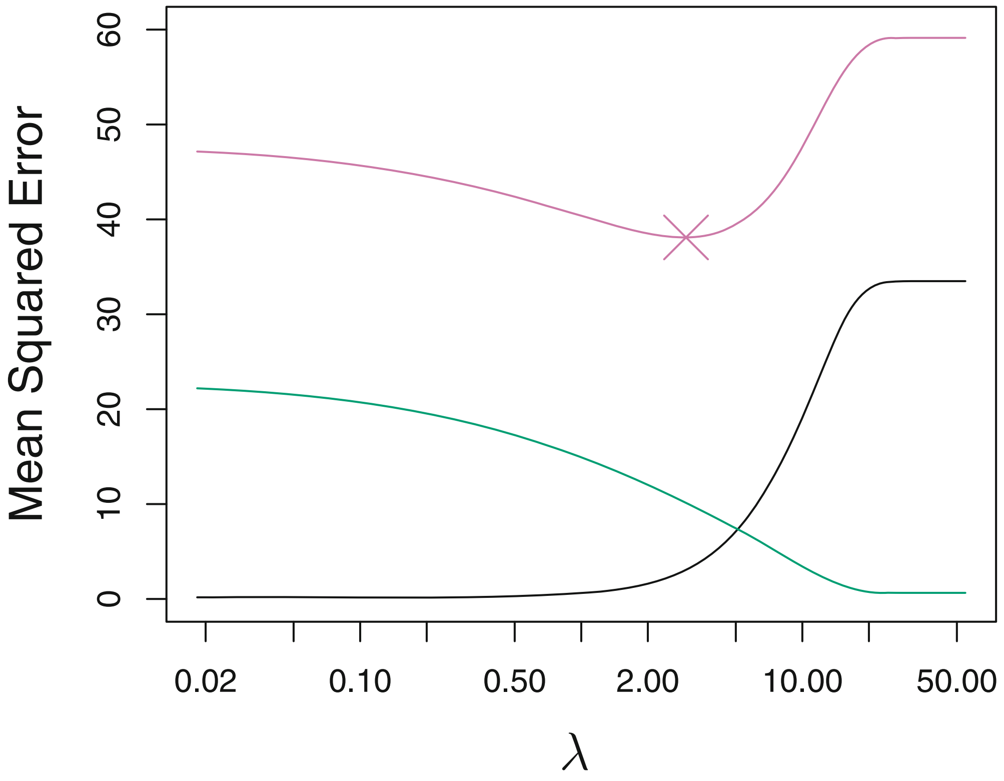

Figure 19: Plots of squared bias (black), variance (green), and test MSE (purple) for the lasso on a simulated data set. The cross in the plot indicate the lasso model for which the MSE is smallest.

Comparing the Lasso and Ridge Regression

It is clear that the lasso has a major advantage over ridge regression, in that it produces simpler and more interpretable models that involve only a subset of the predictors. However, which method leads to better prediction accuracy?

In general, one might expect the lasso to perform better in a setting where a relatively small number of predictors have substantial coefficients, and the remaining predictors have coefficients that are very small or that equal zero. Ridge regression will perform better when the response is a function of many predictors, all with coefficients of roughly equal size. However, the number of predictors that is related to the response is never known a priori for real data sets. A technique such as cross-validation can be used in order to determine which approach is better on a particular data set.

As with ridge regression, when the least squares estimates have excessively high variance, the lasso solution can yield a reduction in variance at the expense of a small increase in bias, and consequently can generate more accurate predictions. Unlike ridge regression, the lasso performs variable selection, and hence results in models that are easier to interpret. There are very efficient algorithms for fitting both ridge and lasso models; in both cases the entire coefficient paths can be computed with about the same amount of work as a single least squares fit. We will explore this further in the lab at the end of this chapter.

4.3 Selecting the Tuning Parameter

Implementing ridge regression and the lasso requires a method for selecting a value for the tuning parameter λ in (26) and (27). Cross-validation provides a simple way to tackle this problem. We choose a grid of λ values, and compute the cross-validation error for each value of λ. We then select the tuning parameter value for which the cross-validation error is smallest. Finally, the model is re-fit using all of the available observations and the selected value of the tuning parameter.

4.3.1 Cross-Validation

In this section, we consider some methods that estimate the test error rate by holding out a subset of the training observations from the fitting process, and then applying the statistical technique to those held out observations. The test error is the average error that results from using a statistical method to predict the response on a new observation; that is, a measurement that was not used in training the method.

We will assume that we are in a simple linear regression context where Y is the response and X a generic covariate.

Leave-One-Out Cross-Validation

Leave-one-out cross-validation (LOOCV) involves splitting the set of observations into two parts. An important feature here is that a single observation (x_1, y_1) is used for the validation set, and the remaining observations $(x_2, y_2), \ldots, (x_n, y_n)$ make up the training set. The statistical method is fitted on the $n-1$ training observations, and a prediction $\hat{y}_1$ is made for the excluded observation, using the value x_1. Since (x_1, y_1) was not used in the fitting process, $Mean\ Squared\ Error = MSE_1 = (y_1 - \hat{y}_1)^2$ provides an approximately unbiased estimate for the test error. However, such an estimate will be highly variable because it is based upon a single observation (x_1, y_1). To address this issue, we can repeat the procedure by selecting (x_2, y_2) for the validation data, training the statistical procedure on the $n-1$ observations $(x_1, y_1), (x_3, y_3), \ldots, (x_n, y_n)$, and computing $MSE_2 = (y_2 - \hat{y}_2)^2$. Repeating this approach n times produces n squared errors, $MSE_1, \ldots, MSE_n$, which are then averaged. More formally, the LOOCV is calculated using:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i.$$

LOOCV tends to produce very reliable estimates and is not affected by randomness in the training/validation set splits. However, it has the potential to be expensive to implement, since the model has to be fitted n times. This can be very time consuming if n is large and/or if each individual model is slow to fit. With regression models, an amazing shortcut makes the cost of LOOCV the same as that of a single model fit. The more efficient formula is:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2,$$

where $\hat{y}_i$ is the i^{th} fitted value from the original least squares fit, and h_i is the i^{th} leverage value.

LOOCV is a very general method, and can be used with any kind of predictive modeling. Bear in mind, however, that the above efficient formula does not hold in general, in which case the model has to be refitted n times.

k-Fold Cross-Validation

An alternative to LOOCV is $k - fold\ CV$. This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fitted on the remaining $k-1$ folds. The mean squared error, MSE_1, is then computed on the observations in the held-out fold. This procedure is repeated k times; each time, a different group of observations is treated as a validation set. This process results in k estimates of the test error, $MSE_1, MSE_2, \ldots, MSE_k$. The $k - fold\ CV$ estimate is computed by averaging these values, that is

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i.$$

It is not hard to see that LOOCV is a special case of $k - fold\ CV$ in which $k = n$. In practice, one typically performs $k - fold\ CV$ using $k = 5$ or $k = 10$. What is the advantage of using $k = 5$ or $k = 10$ rather than $k = n$? The most obvious advantage is computational. LOOCV requires fitting the statistical method n times. This has the potential to be computationally expensive (except for linear models estimated by least squares, where the efficient formula can be used). But cross-validation is a very general approach that can be applied to almost any statistical method; some statistical methods have computationally intensive fitting procedures, and so performing LOOCV may pose computational problems, especially if n is large. In contrast, performing for instance $10 - fold\ CV$ requires fitting the learning procedure only ten times, which may be much more feasible. The downside is the variability of the procedure induced by the random splits. In practical applications, $k = 10$ has proved to be a good choice. A sensitivity analysis using smaller or bigger values of k (e.g., 5 or 20) can also be attempted to check the robustness of the conclusions.

Figure 20 displays the choice of λ that results from performing leave-one- out cross-validation on the ridge regression fits from the `Credit` data set. The dashed vertical lines indicate the selected value of λ. In this case the value is relatively small, indicating that the optimal fit only involves a small amount of shrinkage relative to the least squares solution. In addition, the dip is not very pronounced, so there is rather a wide range of values that would give very similar error. In a case like this we might simply use the least squares solution.

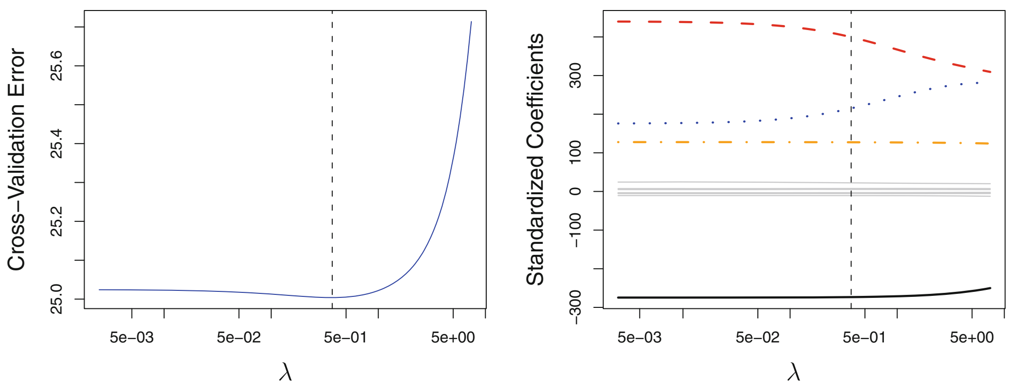

Figure 20: Left: Cross-validation errors that result from applying ridge regression to the `Credit` data set with various value of λ. Right: The coefficient estimates as a function of λ. The vertical dashed lines indicate the value of λ selected by cross-validation.

4.4 Lab: Ridge Regression and the Lasso

We will use the `glmnet` package in order to perform ridge regression and the lasso. The main function in this package is `glmnet()`, which can be used to fit ridge regression models, lasso models, and more. This function has slightly different syntax from other model-fitting functions that we have encountered thus far in these notes. In particular, we must pass in an `x` matrix as well as a `y` vector, and we do not use the `y` $\sim$ `x` syntax. We will now perform ridge regression and the lasso in order to predict `Salary` on the `Hitters` data. Before proceeding we need to ensure that the missing values have been removed from the data.

```
library(ISLR)
Hitters <- na.omit(Hitters)

x <- model.matrix(Salary ~., Hitters)[,-1]
y <- Hitters$Salary
```

The `model.matrix()` function is particularly useful for creating `x`; not only does it produce a matrix corresponding to the 19 predictors but it also automatically transforms any qualitative variables into dummy variables. The latter property is important because `glmnet()` can only take numerical, quantitative inputs.

The `glmnet()` function has an `alpha` argument that determines what type of model is fit. If `alpha = 0` then a ridge regression model is fit, and if `alpha = 1` then a lasso model is fit. We can also fit models with values of `alpha` between `0` and `1`. These models are called elastic net and are an hybrid version of both lasso and ridge regression.

We first fit a ridge regression model.

```
set.seed(100)
library(glmnet)
grid <- 10^seq(10, -2, length = 100)
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

By default the `glmnet()` function performs ridge regression for an automatically selected range of λ values. However, here we have chosen to implement the function over a grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit. As we will see, we can also compute model fits for a particular value of λ that is not one of the original grid values. Note that by default, the `glmnet()` function standardizes the variables so that they are on the same scale. To turn off this default setting, use the argument `standardize = FALSE`.

Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by `coef()`. In this case, it is a 20×100 matrix, with 20 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of λ).

```
dim(coef(ridge.mod))
```

We expect the coefficient estimates to be much smaller, when a large value of λ is used, as compared to when a small value of λ is used. These are the coefficients when λ is large:

```
ridge.mod$lambda[50]
coef(ridge.mod)[,50]
```

In contrast, here are the coefficients when $\lambda = 705$. Note the much larger coefficients associated with this smaller value of λ.

```
ridge.mod$lambda[60]
coef(ridge.mod)[,60]
```

We can use the `predict()` function for a number of purposes. For instance, we can obtain the ridge regression coefficients for a new value of λ, say 50:

```
predict(ridge.mod, s = 50, type = "coefficients")[1:20 ,]
```

Let's see how to use cross-validation to choose the tuning parameter λ. We can do this using the built-in cross-validation function, `cv.glmnet()`. By default, the function performs ten-fold cross-validation, though this can be changed using the argument `nfolds`. Note that we set a random seed first so our results will be reproducible, since the choice of the cross-validation folds is random.

```
set.seed(1)
train <- sample(1: nrow(x), nrow(x)/2)
test <- (- train)
y.test <- y[test]
cv.out <- cv.glmnet(x[train ,], y[train], alpha = 0)
plot(cv.out)
bestlam = cv.out$lambda.min
bestlam
```

Therefore, we see that the value of λ that results in the smallest cross-validation error is 431.

We refit our ridge regression model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates.

```
out <- glmnet(x, y, alpha = 0)
predict(out, type = "coefficients", s = bestlam)[1:20 ,]
```

As expected, none of the coefficients are zero - ridge regression does not perform variable selection!

Finally, let's calculate the MSE on the test set.

```
yhat <- predict(out, s = bestlam, newx = x[test ,])
mse <- mean((y[test] - yhat)^2)
mse
```

The MSE is 120814.

In order to fit a lasso model, we once again use the `glmnet()` function; however, this time we use the argument `alpha = 1`. Other than that change, we proceed just as we did in fitting a ridge model.

```
lasso.mod <- glmnet(x, y, alpha = 1, lambda = grid)

lasso.mod$lambda[50]
coef(lasso.mod)[,50]

lasso.mod$lambda[80]
coef(lasso.mod)[,80]

cv.out <- cv.glmnet(x[train ,], y[train], alpha = 1)
plot(cv.out)
bestlam = cv.out$lambda.min
bestlam
```

We refit our lasso model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates.

```
out <- glmnet(x, y, alpha = 1)
predict(out, type = "coefficients", s = bestlam)[1:20 ,]
```

We can see that some of the coefficients will be exactly equal to zero.

The lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that some of the 19 coefficient estimates are exactly zero. So the lasso model with λ chosen by cross-validation contains fewer variables.

Let's calculate the MSE.

```
yhat <- predict(out, s = bestlam, newx = x[test ,])
mse <- mean((y[test] - yhat)^2)
mse
```

The MSE is lower as compared to that obtained using the ridge regression.