ECON 573 PS4

Rahul Narvekar

10/19/2021

Part 1

Ex 3, 5, 6 from Chapter 5 of ISL.

3) We now review k-fold cross-validation.

3a) Explain how k-fold cross-validation is implemented. The k-fold cross validation is done by randomly dividing all of your groups into k parts, and then training your model on k-1 parts. After taking the average of all the k-1 parts, use the last kth part as your testing set for your trained model

3b) What are the advantages and disadvantages of k-fold crossvalidation relative to:

3bi) Validation Set Approach: There could be a hihgly variable test error. Can also tend to overestimate the test error.

3bii) LOOCV: LOOCV tends to be expensive and requires fits as large as n. K fold can also give better test error. However LOOCV will still provide lower bias

5) In Chapter 4, we used logistic regression to predict the probability of default using income and balance on the Default data set. We will now estimate the test error of this logistic regression model using the validation set approach. Do not forget to set a random seed before beginning your analysis.

5a) Fit a logistic regression model that uses income and balance to predict default.

```
set.seed(123)
library(ISLR)
attach(Default)
logistic = glm(default ~ income + balance, family = "binomial", data = Default)
summary(logistic)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

5b) Using the validation set approach, estimate the test error of this model. In order to do this, you must perform the following steps:

5bi) Split the sample set into a training set and a validation set.

```
trainingSet = sample(dim(Default)[1], dim(Default)[1]* 0.6)
```

5bii) Fit a multiple logistic regression model using only the training observations.

```
log.fit = glm(default ~ income + balance, data = Default, family = "binomial", subset = trainingSet)
summary(log.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = Default, subset = trainingSet)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2367  -0.1390  -0.0561  -0.0203   3.7393
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.162e+01  5.660e-01 -20.524  < 2e-16 ***
## income       1.996e-05  6.563e-06   3.041  0.00236 **
## balance      5.716e-03  2.958e-04  19.320  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1747.00  on 5999  degrees of freedom
## Residual deviance:  929.75  on 5997  degrees of freedom
## AIC: 935.75
##
## Number of Fisher Scoring iterations: 8
```

5biii) Obtain a prediction of default status for each individual in the validation set by computing the posterior probability of default for that individual, and classifying the individual to the default category if the posterior probability is greater than 0.5.

```
pVals = predict(log.fit, newdata = Default[-trainingSet, ], type = "response")
predictions = rep("No", length(pVals))
predictions[pVals > 0.5] = "Yes"
```

5biv) Compute the validation set error, which is the fraction of the observations in the validation set that are misclassified.

```
mean(predictions != Default[-trainingSet, ]$default)
```

```
## [1] 0.02625
```

5c) Repeat the process in (b) three times, using three different splits of the observations into a training set and a validation set. Comment on the results obtained.

```
trainingSet = sample(dim(Default)[1], dim(Default)[1]* 0.7)
log.fit = glm(default ~ income + balance, data = Default, family = "binomial", subset = trainingSet)
pVals = predict(log.fit, newdata = Default[-trainingSet, ], type = "response")
predictions = rep("No", length(pVals))
predictions[pVals > 0.5] = "Yes"
paste("70%: ", toString(mean(predictions != Default[-trainingSet, ]$default)))
```

```
## [1] "70%:  0.025"
```

```
trainingSet = sample(dim(Default)[1], dim(Default)[1]* 0.8)
log.fit = glm(default ~ income + balance, data = Default, family = "binomial", subset = trainingSet)
pVals = predict(log.fit, newdata = Default[-trainingSet, ], type = "response")
predictions = rep("No", length(pVals))
predictions[pVals > 0.5] = "Yes"
paste("80%: ", toString(mean(predictions != Default[-trainingSet, ]$default)))
```

```
## [1] "80%:  0.027"
```

```
trainingSet = sample(dim(Default)[1], dim(Default)[1]* 0.9)
log.fit = glm(default ~ income + balance, data = Default, family = "binomial", subset = trainingSet)
pVals = predict(log.fit, newdata = Default[-trainingSet, ], type = "response")
predictions = rep("No", length(pVals))
predictions[pVals > 0.5] = "Yes"
paste("90%: ", toString(mean(predictions != Default[-trainingSet, ]$default)))
```

```
## [1] "90%:  0.027"
```

error rate is variable depending on how mnay observations are used in the model

5d) Now consider a logistic regression model that predicts the probability of default using income, balance, and a dummy variable for student. Estimate the test error for this model using the validation set approach. Comment on whether or not including a dummy variable for student leads to a reduction in the test error rate.

```
trainingSet = sample(dim(Default)[1], dim(Default)[1]* 0.7)
log.fit = glm(default ~ income + balance + student, data = Default, family = "binomial", subset = train
pVals = predict(log.fit, newdata = Default[-trainingSet, ], type = "response")
predictions = rep("No", length(pVals))
predictions[pVals > 0.5] = "Yes"
paste("60% with student: ", toString(mean(predictions != Default[-trainingSet, ]$default)))
```

```
## [1] "60% with student:  0.0236666666666667"
```

Does not seem like adding student helped reduce test error

6) We continue to consider the use of a logistic regression model to predict the probability of default using income and balance on the Default data set. In particular, we will now compute estimates for the standard errors of the income and balance logistic regression coefficients in two different ways: (1) using the bootstrap, and (2) using the standard formula for computing the standard errors in the glm() function. Do not forget to set a random seed before beginning your analysis.

6a) Using the summary() and glm() functions, determine the estimated standard errors for the coefficients associated with income and balance in a multiple logistic regression model that uses both predictors.

```
log.fit = glm(default ~ income + balance, data = Default, family = "binomial")
summary(log.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

5b) Write a function, boot.fn(), that takes as input the Default data set as well as an index of the observations, and that outputs the coefficient estimates for income and balance in the multiple logistic regression model.

```r
boot.fn = function(defaults, i) {
  return(coef(glm(default ~ income + balance, data = defaults, family = "binomial", subset = i)))
}
```

5c) Use the boot() function together with your boot.fn() function to estimate the standard errors of the logistic regression coefficients for income and balance.

```r
library(boot)
boot(Default, boot.fn, 100)
```

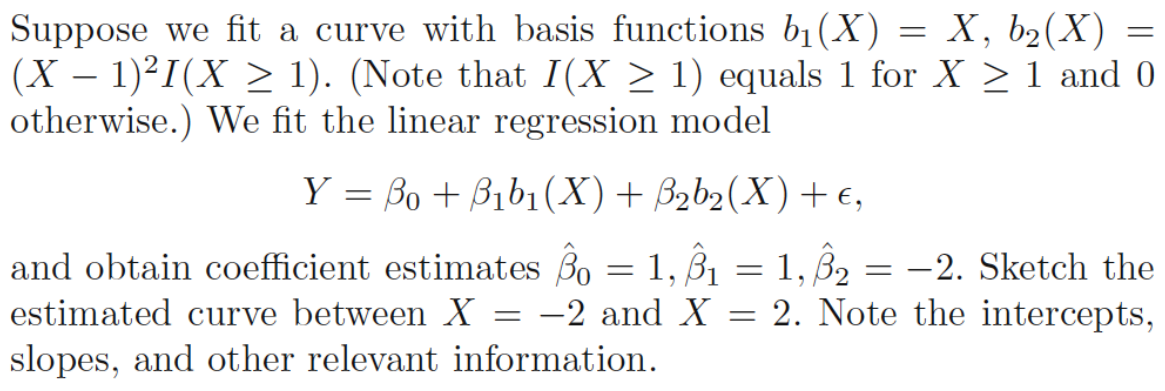
```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 100)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -1.509254e-02 4.201684e-01
## t2*  2.080898e-05  5.158585e-09 5.173217e-06
## t3*  5.647103e-03  1.000535e-05 2.167726e-04
```

5d) Comment on the estimated standard errors obtained using the glm() function and using your bootstrap function.

Both of the estimated standard errors are similar

Part 2

Ex 3, 9 from Chapter 7 of ISL.

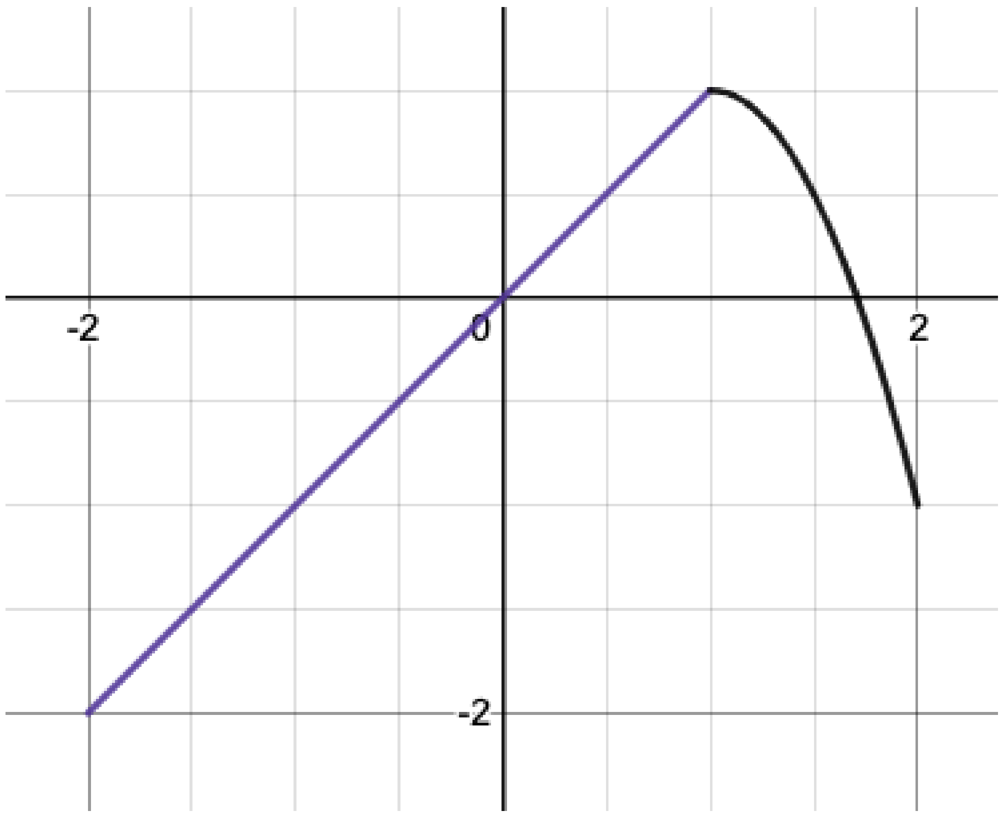


Suppose we fit a curve with basis functions $b_1(X) = X$, $b_2(X) = (X-1)^2 I(X \geq 1)$. (Note that $I(X \geq 1)$ equals 1 for $X \geq 1$ and 0 otherwise.) We fit the linear regression model

$$Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon,$$

and obtain coefficient estimates $\hat{\beta}_0 = 1, \hat{\beta}_1 = 1, \hat{\beta}_2 = -2$. Sketch the estimated curve between $X = -2$ and $X = 2$. Note the intercepts, slopes, and other relevant information.

3)



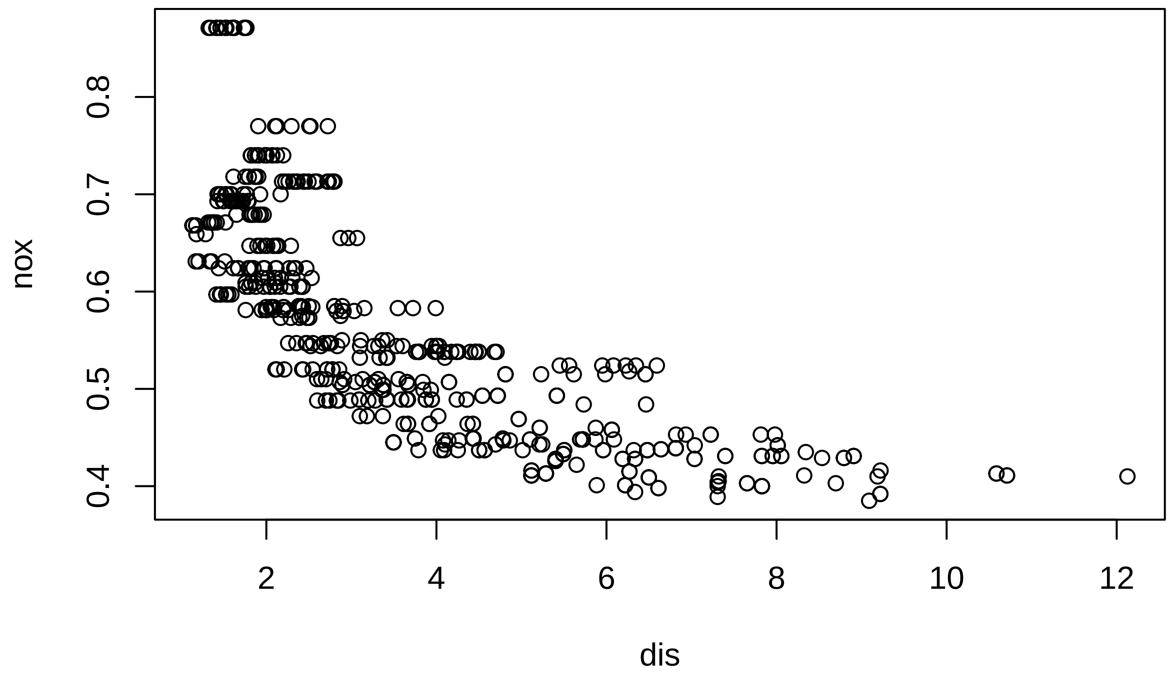
4) This question uses the variables dis (the weighted mean of distances to five Boston employment centers) and nox (nitrogen oxides concentration in parts per 10 million) from the Boston data. We will treat dis as the predictor and nox as the response.

9a) Use the poly() function to fit a cubic polynomial regression to predict nox using dis. Report the regression output, and plot the resulting data and polynomial fits.

```
library(MASS)
attach(Boston)
cubic = glm(nox ~ poly(dis, 3), data = Boston)
summary(cubic)
```

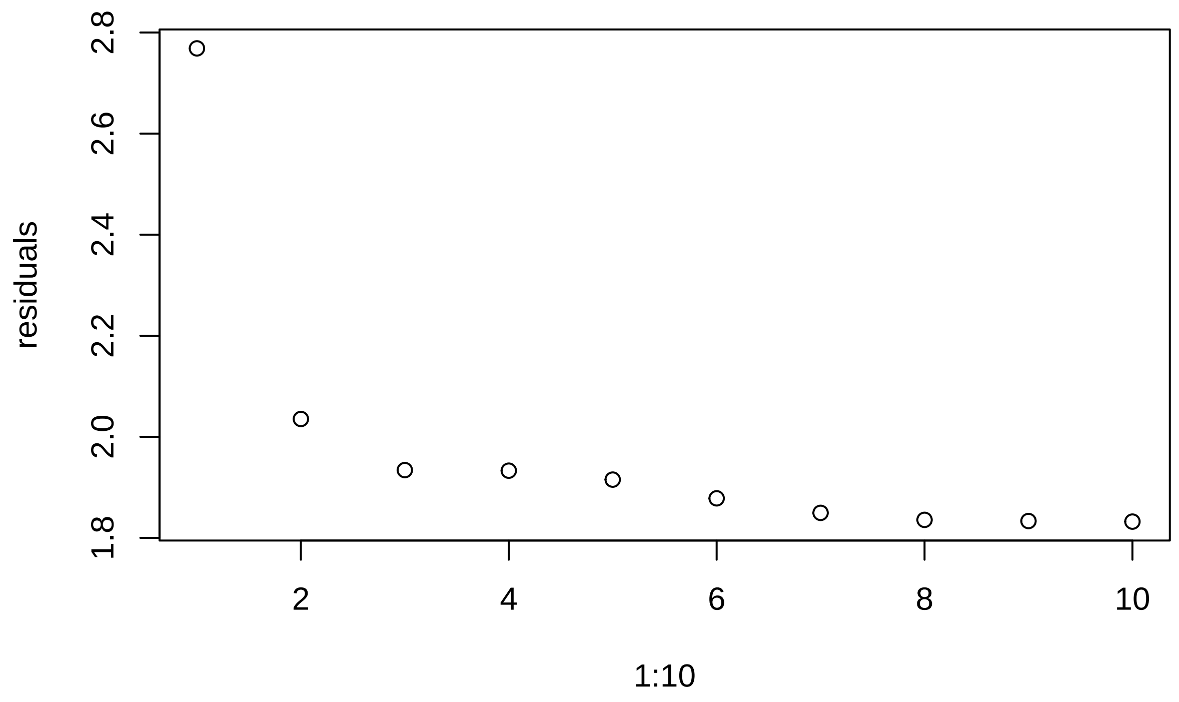
```
##
## Call:
## glm(formula = nox ~ poly(dis, 3), data = Boston)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.121130  -0.040619  -0.009738   0.023385   0.194904
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.554695   0.002759 201.021  < 2e-16 ***
## poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.003852802)
##
##     Null deviance: 6.7810  on 505  degrees of freedom
## Residual deviance: 1.9341  on 502  degrees of freedom
## AIC: -1370.9
##
## Number of Fisher Scoring iterations: 2
```

```
scale = seq(min(Boston$dis), max(Boston$dis), by = 0.05)
prediction = predict(cubic, list(dis = scale))
plot(nox ~ dis, data = Boston)
```



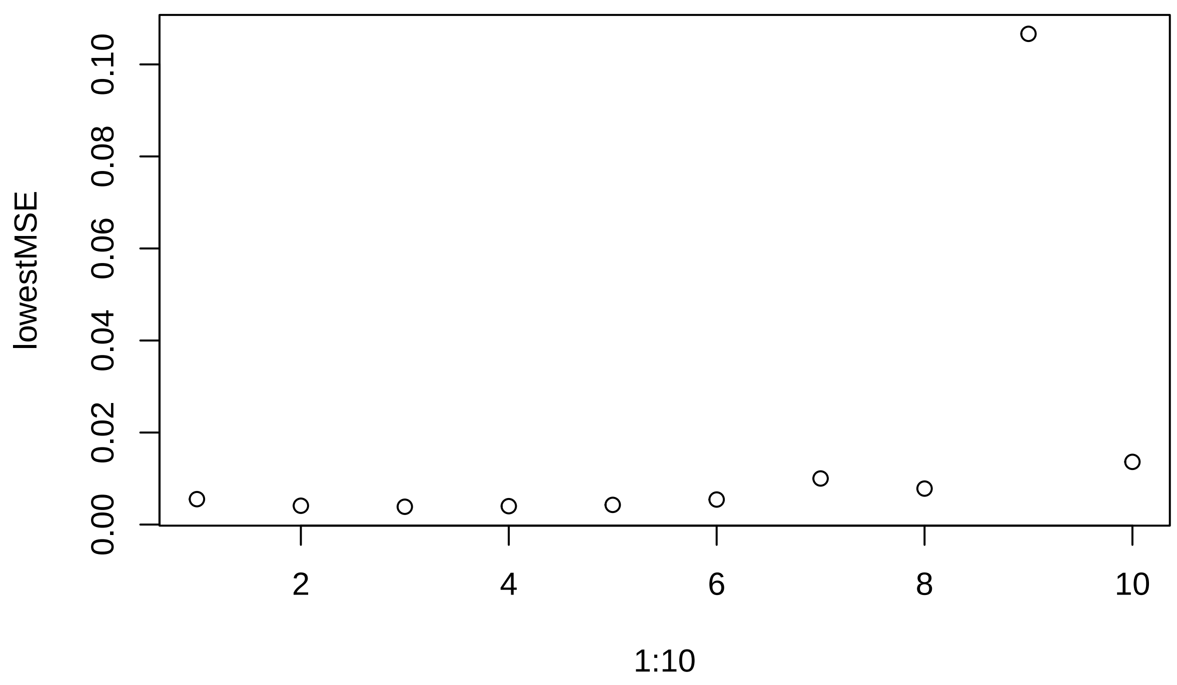
9b) Plot the polynomial fits for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.

```
residuals = rep(0, 10)
for(i in 1:10) {
  residuals[i] = sum(glm(nox ~ poly(dis, i), data = Boston)$residuals ^ 2)
}
plot(1:10, residuals)
```



9c) Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.

```
library(boot)
lowestMSE = rep(0, 10)
for(i in 1:10) {
   polynomial = glm(nox ~ poly(dis, i), data = Boston)
   lowestMSE[i] = cv.glm(Boston, polynomial, K = 5)$delta[1]
}
plot(1:10, lowestMSE)
```



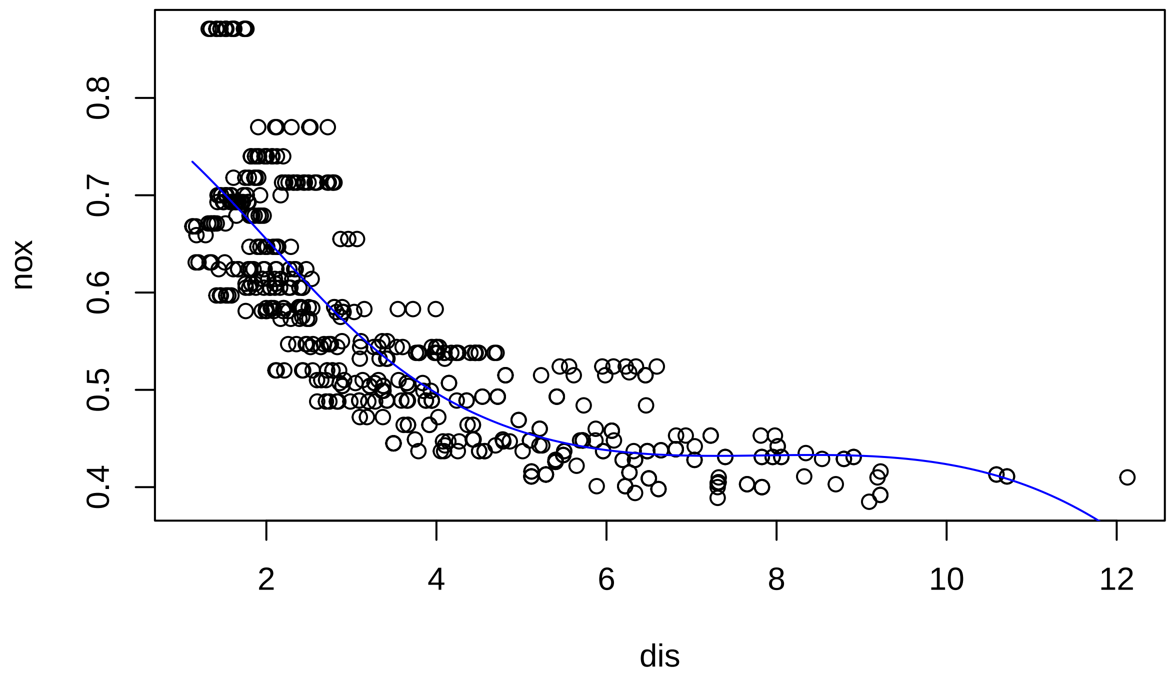
```
which.min(lowestMSE)
```

```
## [1] 3
```

At k = 5, 8th degree has the lowestMSE

9d) Use the bs() function to fit a regression spline to predict nox using dis. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.

```
library(splines)
spline = glm(nox ~ bs(dis, df= 4), data = Boston)
prediction = predict(spline, data.frame(dis = scale))
plot(nox ~ dis, data = Boston)
lines(scale, prediction, col = "blue")
```



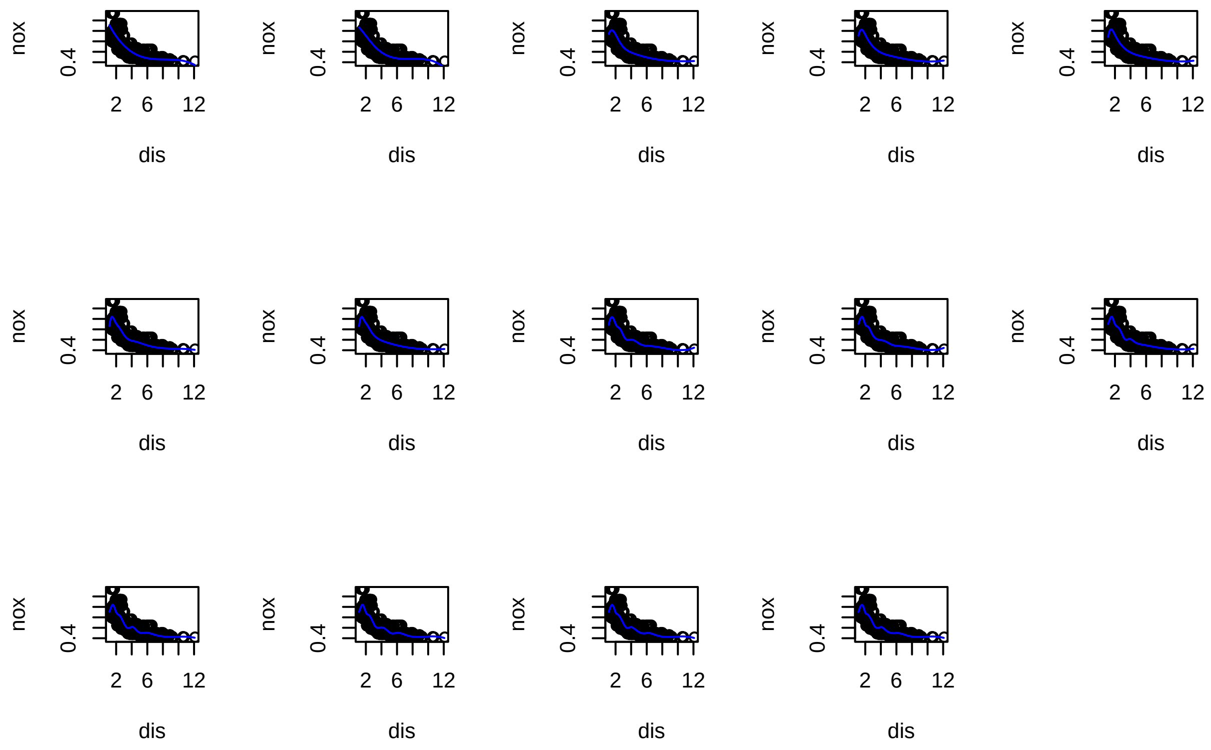
```
summary(spline)
```

```
##
## Call:
## glm(formula = nox ~ bs(dis, df = 4), data = Boston)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.124622  -0.039259  -0.008514   0.020850   0.193891
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.73447    0.01460  50.306  < 2e-16 ***
## bs(dis, df = 4)1 -0.05810    0.02186  -2.658  0.00812 **
## bs(dis, df = 4)2 -0.46356    0.02366 -19.596  < 2e-16 ***
## bs(dis, df = 4)3 -0.19979    0.04311  -4.634 4.58e-06 ***
## bs(dis, df = 4)4 -0.38881    0.04551  -8.544  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.003837874)
##
##     Null deviance: 6.7810  on 505  degrees of freedom
## Residual deviance: 1.9228  on 501  degrees of freedom
## AIC: -1371.9
```

```
##
## Number of Fisher Scoring iterations: 2
```

9e) Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.

```r
RSS = rep(0, 16)
par(mfrow=c(3,5))
for (i in 3:16) {
    spline = glm(nox ~ bs(dis, df = i), data = Boston)
    RSS[i] = sum(spline$residuals^2)
    prediction = predict(spline, data.frame(dis=scale))
    plot(nox ~ dis, data = Boston)
    lines(scale, prediction, col ="blue")
}
```

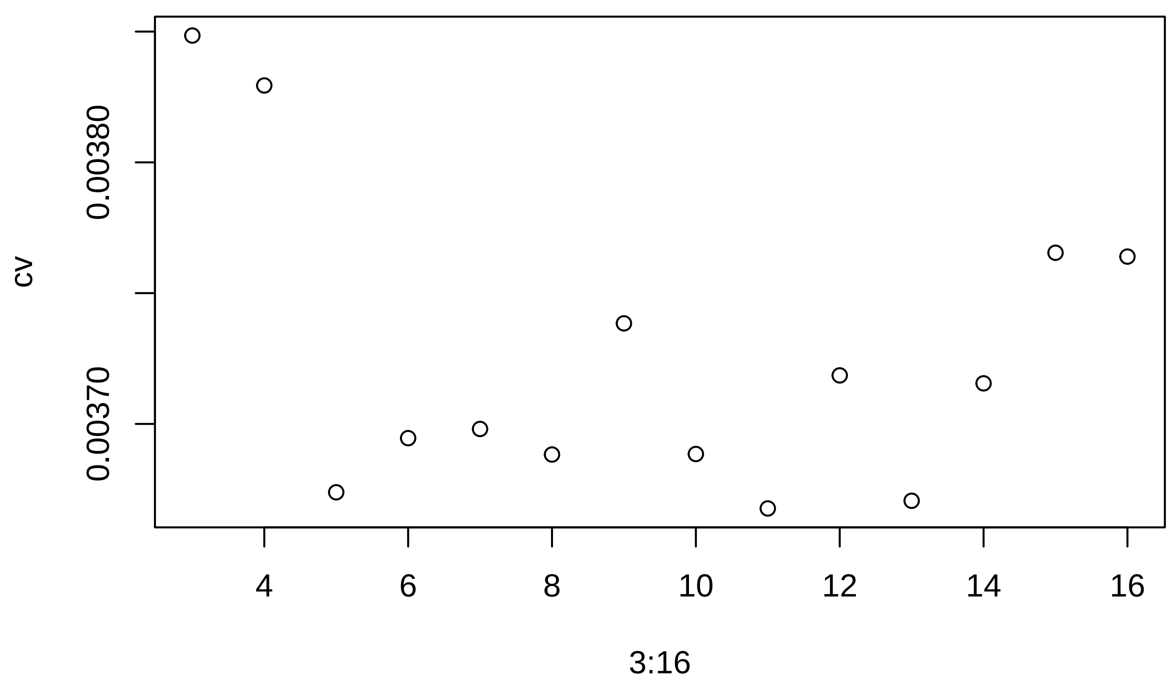


10f) Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline on this data. Describe your results.

```r
cv = sapply(3:16, function(i){
  return(cv.glm(Boston, glm(nox ~ bs(dis, df = i), data=Boston), K = 5)$delta[2])
})
```

```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.1296, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.1296, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.0921), Boundary.knots =
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.0921), Boundary.knots =
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.4212, `66.66667%`
## = 4.38856666666667: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.4212, `66.66667%`
## = 4.38856666666667: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.4214, `66.66667%`
## = 4.28596666666666: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.4214, `66.66667%`
## = 4.28596666666666: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.11045, `50%` = 3.36665, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.11045, `50%` = 3.36665, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.102875, `50%` = 3.1073, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.102875, `50%` = 3.1073, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.9512, `40%` = 2.60606, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.9512, `40%` = 2.60606, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.94984, `40%` = 2.5583, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.94984, `40%` = 2.5583, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(‘16.66667%‘ = 1.8755, ‘33.33333%‘
## = 2.42426666666667, : some ’x’ values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘16.66667%‘ = 1.8755, ‘33.33333%‘
## = 2.42426666666667, : some ’x’ values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘14.28571%‘ = 1.78307142857143, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘14.28571%‘ = 1.78307142857143, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘14.28571%‘ = 1.80825714285714, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘14.28571%‘ = 1.80825714285714, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘12.5%‘ = 1.7429, ‘25%‘ = 2.072, :
## some ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘12.5%‘ = 1.7429, ‘25%‘ = 2.072, :
## some ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘11.11111%‘ = 1.7257, ‘22.22222%‘ =
## 2.0026, : some ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘11.11111%‘ = 1.7257, ‘22.22222%‘ =
## 2.0026, : some ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘11.11111%‘ = 1.64561111111111, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘11.11111%‘ = 1.64561111111111, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘10%‘ = 1.61242, ‘20%‘ = 1.90288, :
## some ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘10%‘ = 1.61242, ‘20%‘ = 1.90288, :
## some ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘9.090909%‘ = 1.61669090909091, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘9.090909%‘ = 1.61669090909091, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘8.333333%‘ = 1.58920833333333, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(‘8.333333%‘ = 1.58920833333333, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘7.692308%‘ = 1.58889230769231, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘7.692308%‘ = 1.58889230769231, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘7.692308%‘ = 1.55545384615385, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘7.692308%‘ = 1.55545384615385, : some
## ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘7.142857%‘ = 1.5323, ‘14.28571%‘ =
## 1.7932, : some ’x’ values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(‘7.142857%‘ = 1.5323, ‘14.28571%‘ =
## 1.7932, : some ’x’ values beyond boundary knots may cause ill-conditioned bases
```

```
plot(3:16, cv)
```



```r
which.min(cv)
```

```
## [1] 9
```

With K set to 5, 8 df will yield the lowest error

Part 3

Ex 5, 10 from Chapter 8 of ISL.

5. Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of $P(\text{Class is Red}|X)$:

$$0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, \text{ and } 0.75.$$

8.4 Exercises 333

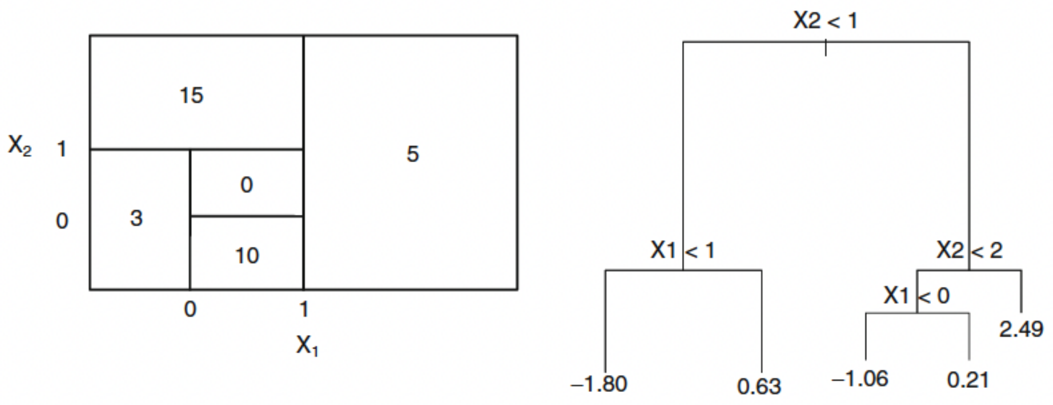


FIGURE 8.12. Left: *A partition of the predictor space corresponding to Exercise 4a.* Right: *A tree corresponding to Exercise 4b.*

There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in this chapter. The second approach is to classify based on the average probability. In this example, what is the final classification under each of these two approaches?

5.)
In the majority case the final outcome will be Red where $P(\text{Red}|X) > P(\text{Green}|X)$ 6 out of 10 times. If it is done with average probability then the final outcome will be green

10) We now use boosting to predict Salary in the Hitters data set.

10a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

```
attach(Hitters)
Hitters = na.omit(Hitters)
Hitters$Salary = log(Hitters$Salary)
```

10b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

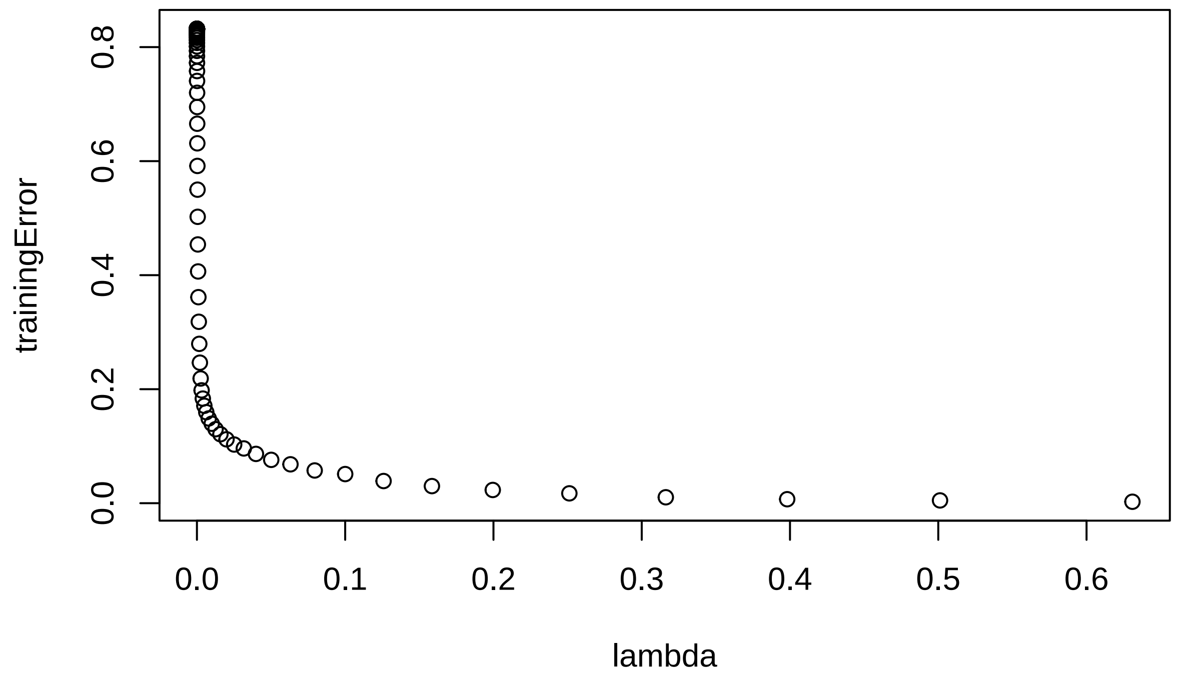
```
training = Hitters[1:200, ]
testing = Hitters[-(1:200), ]
```

10c) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter lambda. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
library(gbm)
```

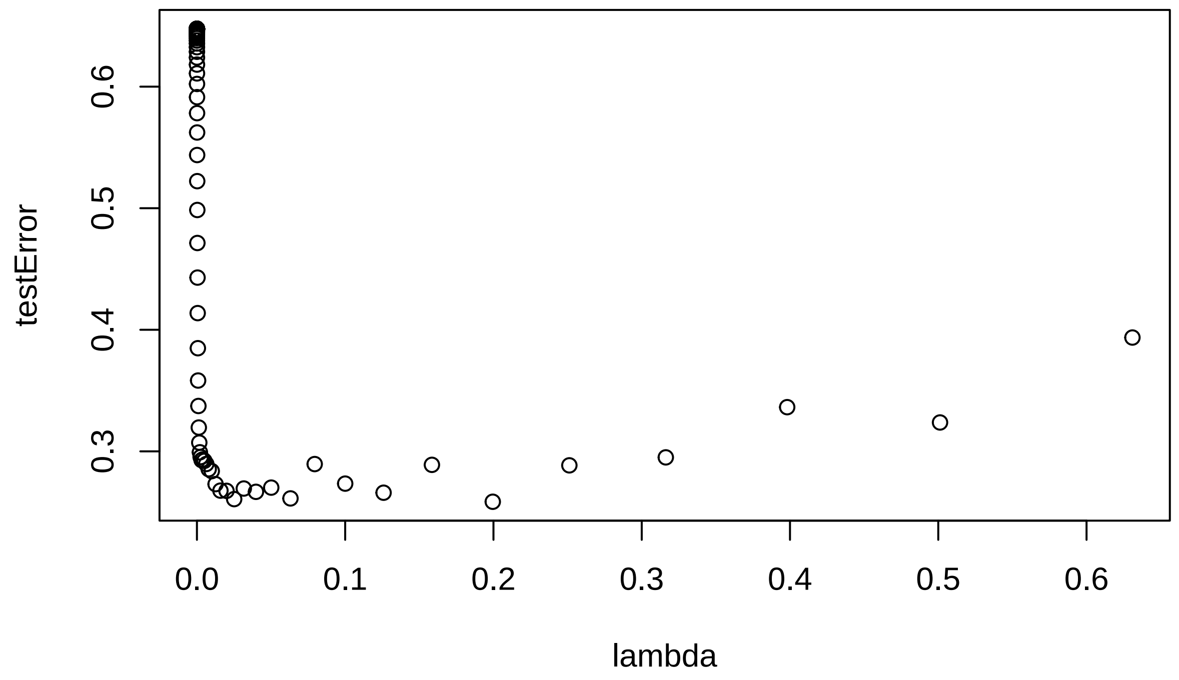
```
## Loaded gbm 2.1.8
```

```
lambda = 10^(seq(-10, -0.2, by = 0.1))
trainingError = rep(0, length(lambda))
for(i in 1:length(lambda)) {
  boost = gbm(Salary ~ ., data = training, distribution = "gaussian", n.trees = 1000, shrinkage = lambda
  prediction = predict(boost, training, n.trees = 1000)
  trainingError[i] = mean((prediction - training$Salary)^2)
}
plot(lambda, trainingError)
```



10d) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```
testError = rep(0, length(lambda))
for(i in 1:length(lambda)) {
  boost = gbm(Salary ~ ., data = training, distribution = "gaussian", n.trees = 1000, shrinkage = lambda
  prediction = predict(boost, testing, n.trees = 1000)
  testError[i] = mean((prediction - testing$Salary)^2)
}
plot(lambda, testError)
```



```
min(testError)
```

```
## [1] 0.2585466
```

10e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

```
#Linear
linear = glm(Salary ~ ., data = training)
prediction = predict(linear, testing)
mean((prediction - testing$Salary)^2)
```

```
## [1] 0.4917959
```

```
#Ridge
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-2
```

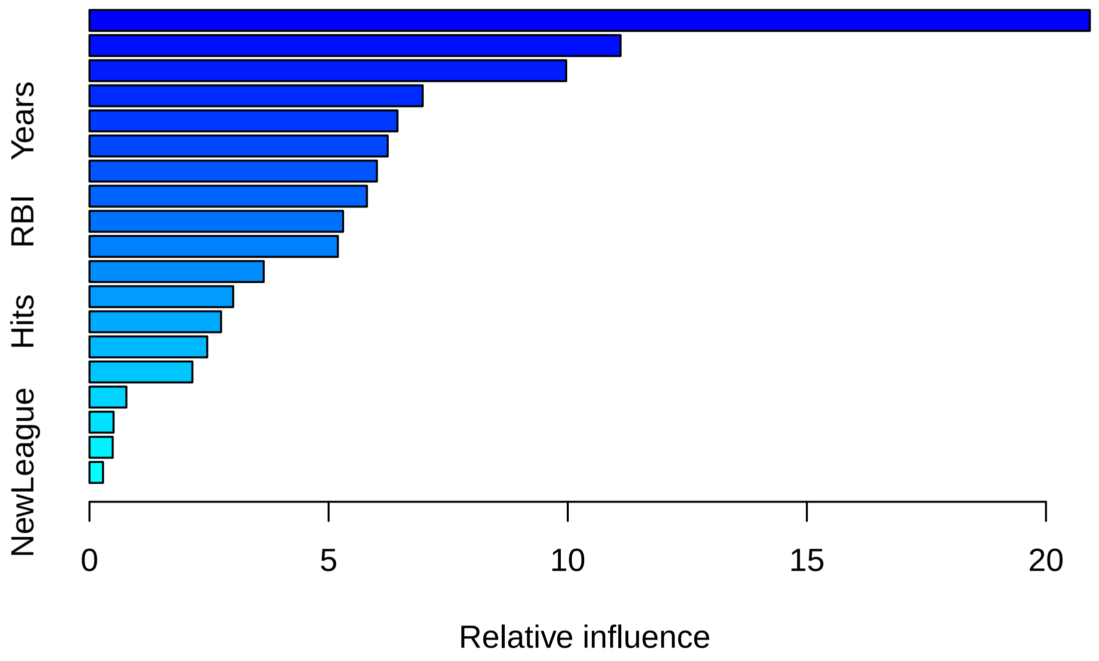
```r
trainingSet = model.matrix(Salary ~ ., data = training)
testingSet = model.matrix(Salary ~ ., data = testing)
trainingSalary = training$Salary
ridge = glmnet(trainingSet, trainingSalary, alpha = 0)
prediction = predict(ridge, s = 0.01, newx = testingSet)
mean((prediction - testing$Salary)^2)
```

```
## [1] 0.4570283
```

MSE is lower in both cases for Boosting

10f) Which variables appear to be the most important predictors in the boosted model?

```r
boost = gbm(Salary ~ ., data = training, distribution = "gaussian", n.trees = 1000, shrinkage = lambda[
summary(boost)
```



```
##                   var    rel.inf
## CAtBat         CAtBat 20.9169523
## CRuns           CRuns 11.1054837
## PutOuts       PutOuts  9.9679216
## Walks           Walks  6.9667751
## Years           Years  6.4393431
## Assists       Assists  6.2357338
## AtBat           AtBat  6.0098564
## CHmRun       CHmRun  5.8013531
## RBI             RBI  5.3032525
## CWalks       CWalks  5.1937562
## HmRun         HmRun  3.6437844
## Errors       Errors  3.0050459
## Hits           Hits  2.7511495
## Runs           Runs  2.4624535
## CRBI           CRBI  2.1516061
## CHits         CHits  0.7722858
## Division   Division  0.5036307
## League       League  0.4861479
## NewLeague NewLeague  0.2834683
```

CAtBat is the most important

10g) Now apply bagging to the training set. What is the test set MSE for this approach?

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
bagging = randomForest(Salary ~ ., data = training, ntree = 500, mtry = 19)
prediction = predict(bagging, newdata = testing)
mean((prediction - testing$Salary)^2)
```

```
## [1] 0.2308455
```

MSE is slightly lower than the boosting MSE