Midterm Review

Group 16

February 25, 2020

Chapter 6 Number 10 (Linear Model Selection / Regularization)

Question 10: We have seen that as the number of features used in a model increases,the training error will necessarily decrease, but the test error may not. We will now explore this in a simulated data set.

(a) Generate a data set with $p = 20$ features, $n = 1{,}000$ observations, and an associated quantitative response vector generated according to the model

$$Y = X\beta + \epsilon,$$

where β has some elements that are exactly equal to zero.

```
set.seed(17)
p <- 20 # features
n <- 1000 # observations

x = matrix(rnorm(n * p), n, p)
Betas = rnorm(p)
Betas[2] = 0
Betas[9] = 0
Betas[12] = 0
Betas[15] = 0
Betas[19] = 0
eps = rnorm(p)
y = x %*% Betas + eps
```

(b) Split your data set into a training set containing 100 observations and a test set containing 900 observations.

```
train = sample(seq(1000), 100, replace = FALSE)
y.train = y[train, ]
y.test = y[-train, ]
x.train = x[train, ]
x.test = x[-train, ]
```

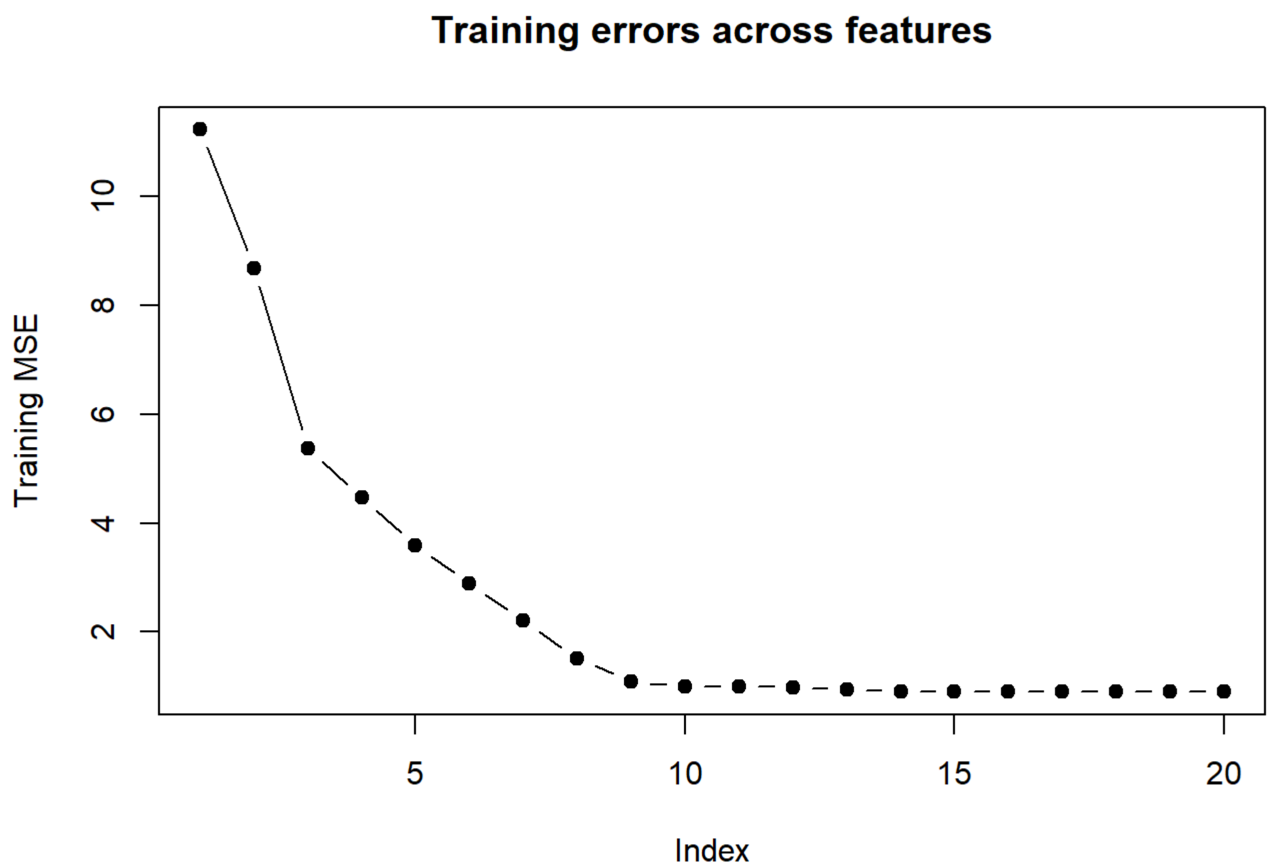
(c) Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size.

```
# Verify leaps package is loaded
# From p.245
require(leaps)
```

```
## Loading required package: leaps
```

```
## Warning: package 'leaps' was built under R version 3.6.2
```

```
regfit.full = regsubsets(y ~ ., data = data.frame(x = x.train, y = y.train),
                         nvmax = p)
errors = rep(NA, p)
x_cols = colnames(x, do.NULL = FALSE, prefix = "x.")
for (i in 1:p) {
  coefi = coef(regfit.full, id = i)
  pred = as.matrix(x.train[, x_cols %in% names(coefi)]) %*% coefi[names(coefi) %in%
                                                                   x_cols]
  errors[i] = mean((y.train - pred)^2)
}
plot(errors, ylab = "Training MSE", pch = 19, type = "b", main = "Training errors across feature
s")
```

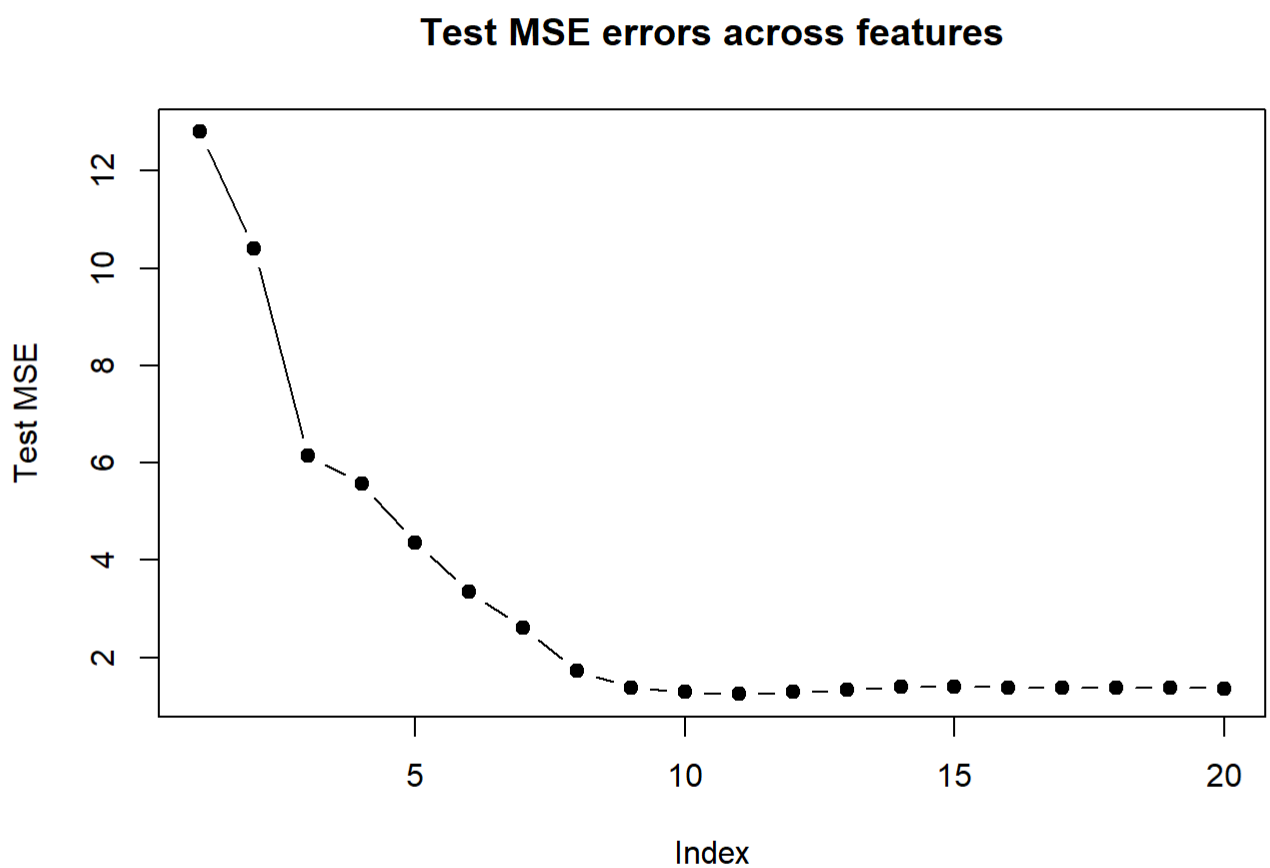



```
which.min(errors)  # min for train error should be at max pred count
```

```
## [1] 18
```

(d) Plot the test set MSE associated with the best model of each size.

```r
errors = rep(NA, p)
for (i in 1:p) {
    coefi = coef(regfit.full, id = i)
    pred = as.matrix(x.test[, x_cols %in% names(coefi)]) %*% coefi[names(coefi) %in%
        x_cols]
    errors[i] = mean((y.test - pred)^2)
}
plot(errors, ylab = "Test MSE", pch = 19, type = "b", main = "Test MSE errors across features")
```



(e) For which model size does the test set MSE take on its minimum value? Comment on your results. If it takes on its minimum value for a model containing only an intercept or a model containing all of the features, then play around with the way that you are generating the data in (a) until you come up with a scenario in which the test set MSE is minimized for an intermediate model size.

```r
(min <- which.min(errors))
```

```
## [1] 11
```

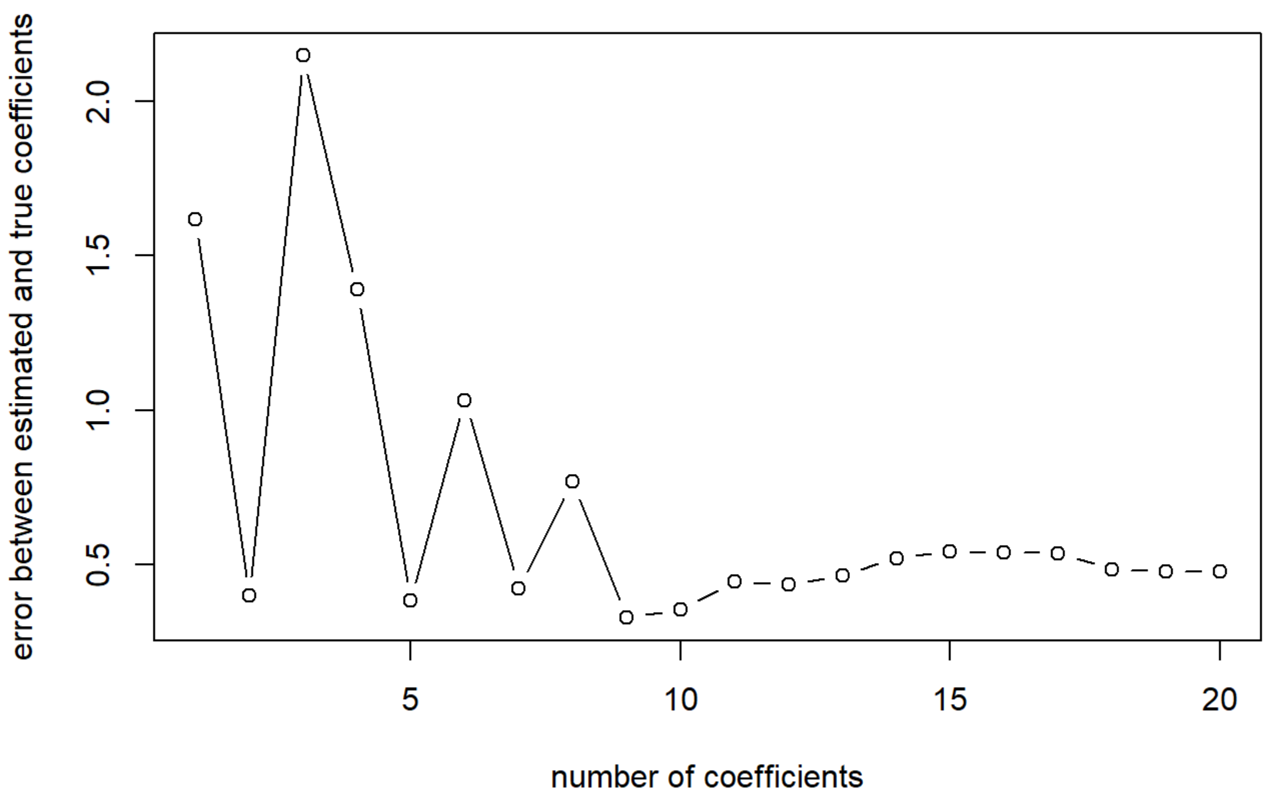
(f) How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient values.

```r
coef(regfit.full, id = min)
```

```
## (Intercept)         x.1         x.3         x.7         x.8        x.10 
##   0.4269208  -0.9092418   1.9139172   0.5197140   1.0232232   1.7243344 
##        x.11        x.13        x.14        x.17        x.18        x.20 
##   1.9652998  -1.0439678   0.2216461   0.2762867  -1.1132668  -0.9127267
```

(g) Create a plot displaying $\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j^r)^2}$ for a range of values of r, where $\hat{\beta}_j^r$ is the jth coefficient estimate for the best model containing r coefficients. Comment on what you observe. How does this compare to the test MSE plot from (d)?

```r
errors = rep(NA, p)
a = rep(NA, p)
b = rep(NA, p)
for (i in 1:p) {
    coefi = coef(regfit.full, id = i)
    a[i] = length(coefi) - 1
    b[i] = sqrt(sum((Betas[x_cols %in% names(coefi)] - coefi[names(coefi) %in% x_cols])^2) +
        sum(Betas[!(x_cols %in% names(coefi))])^2)
}
plot(x = a, y = b, type = "b", xlab = "number of coefficients", ylab = "error between estimated
 and true coefficients")
```



```
which.min(b)
```

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

A model with 9 coefficients (10 with intercept) minimizes the error between the estimated and true coefficients.

Of the 20 original features, five were set to zero. 11 plus the intercept were useful meaning that an 11 parameter model is better than a 20 parameter model. A better fit of the true coefficient as measured here doesn't mean the model will have a lower test MSE.

Chapter 7 Number 6 (Polynomial Regression and Step Function)

6. In this exercise, you will further analyze the `Wage` data set considered throughout this chapter.

```
require(ISLR)
```

```
## Loading required package: ISLR
```

```
require(boot)
```

```
## Loading required package: boot
```

```
attach(Wage)
dim(Wage)
```

```
## [1] 3000   11
```

```
head(Wage)
```

```
##        year age           maritl     race       education             region
## 231655 2006  18 1. Never Married 1. White    1. < HS Grad 2. Middle Atlantic
## 86582  2004  24 1. Never Married 1. White 4. College Grad 2. Middle Atlantic
## 161300 2003  45       2. Married 1. White 3. Some College 2. Middle Atlantic
## 155159 2003  43       2. Married 3. Asian 4. College Grad 2. Middle Atlantic
## 11443  2005  50      4. Divorced 1. White      2. HS Grad 2. Middle Atlantic
## 376662 2008  54       2. Married 1. White 4. College Grad 2. Middle Atlantic
##              jobclass         health health_ins  logwage      wage
## 231655  1. Industrial      1. <=Good      2. No 4.318063  75.04315
## 86582  2. Information 2. >=Very Good      2. No 4.255273  70.47602
## 161300  1. Industrial      1. <=Good     1. Yes 4.875061 130.98218
## 155159 2. Information 2. >=Very Good     1. Yes 5.041393 154.68529
## 11443  2. Information      1. <=Good     1. Yes 4.318063  75.04315
## 376662 2. Information 2. >=Very Good     1. Yes 4.845098 127.11574
```

```
names(Wage)
```

```
## [1] "year"       "age"        "maritl"     "race"       "education"
## [6] "region"     "jobclass"   "health"     "health_ins" "logwage"
## [11] "wage"
```

(a) Perform polynomial regression to predict `wage` using `age`. Use cross-validation to select the optimal degree d for the polynomial. What degree was chosen, and how does this compare to the results of hypothesis testing using ANOVA? Make a plot of the resulting polynomial fit to the data.

```
#Keep an array of all cross-validation errors.
#We are performing K-fold cross validation with K=10.
set.seed(1)
#we are not yet sure what is the best optimal degree d, so we set it as i,
#and we will use for loop to do cross validation and figure out optimal d for the model
all.deltas = rep(NA, 10)
for (i in 1:10) {
  glm.fit = glm(wage~poly(age, i), data=Wage)
  all.deltas[i] = cv.glm(Wage, glm.fit, K=10)$delta[2]
}

all.deltas
```

```
## [1] 1676.681 1600.607 1598.089 1595.381 1594.716 1595.676 1593.962 1597.595
## [9] 1593.472 1595.397
```

```
which.min(all.deltas)
```

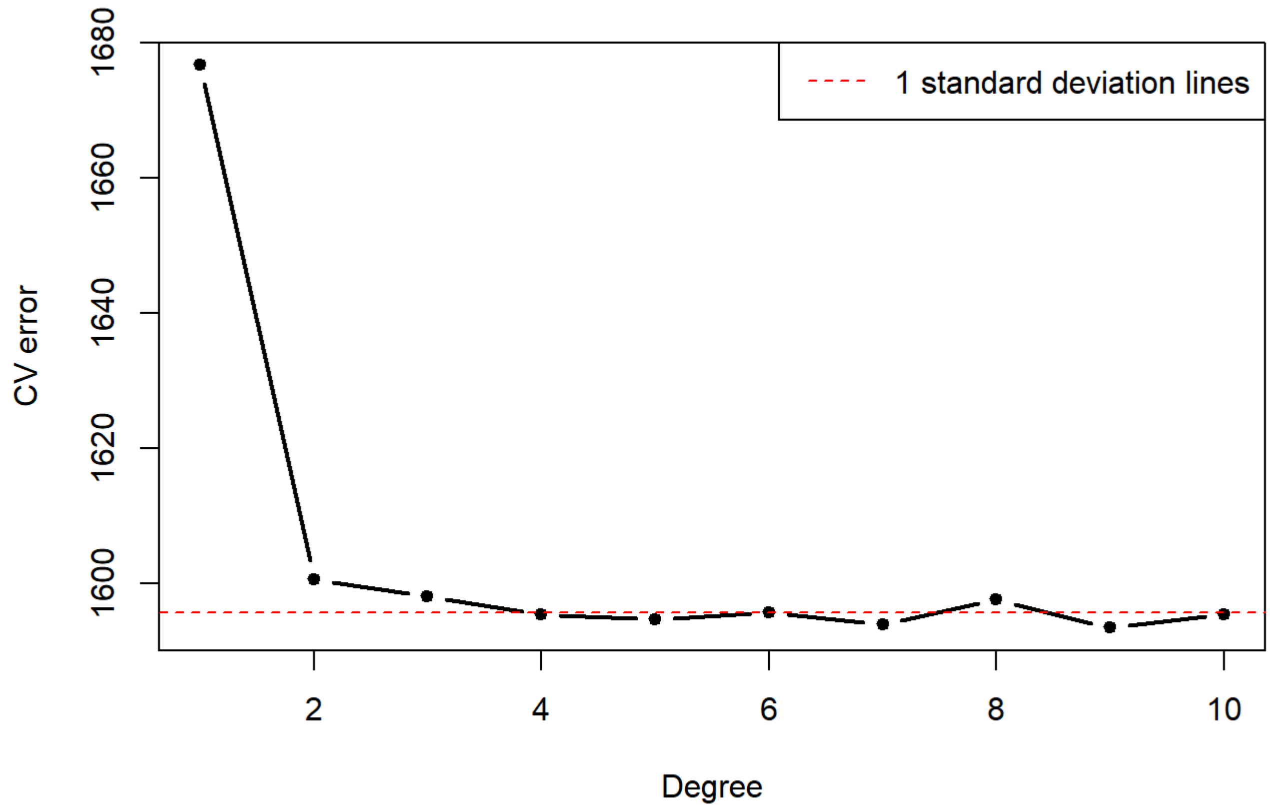
```
## [1] 9
```

```
min(all.deltas)
```

```
## [1] 1593.472
```

We will plot the graph with degree from 1 to 10 on the x axis, cv error on the y axis. We want to try to figure out which degree has the lowest cv error. By using which.min, we see that the lowest error is found at 9 features with a total of 1593.472. We can then use this to identify one standard deviation from the min.

```
plot(1:10, all.deltas, xlab="Degree", ylab="CV error", type="b", pch=20, lwd=2)
min.point = which.min(all.deltas)
sd.points = sd(all.deltas[2:10])
abline(h=sd.points + all.deltas[min.point], col="red", lty="dashed")
legend("topright", "1 standard deviation lines", lty="dashed", col="red")
```



The cv-plot with standard deviation lines show that four degrees of freedom is the lowest df under one standard deviation from the min, therefore we select it as our best model.

How does this compare to using anova? Let's find out.

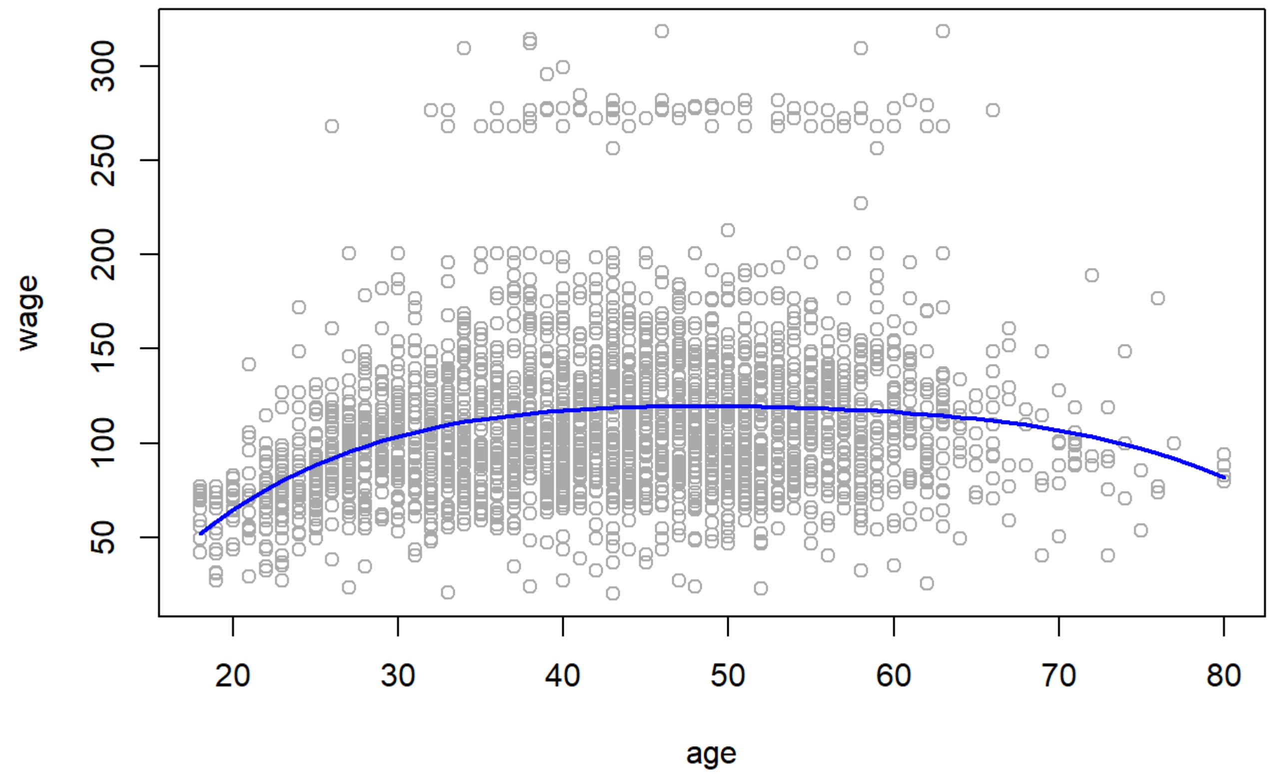
```
fit.1 = lm(wage~poly(age, 1), data=Wage)
fit.2 = lm(wage~poly(age, 2), data=Wage)
fit.3 = lm(wage~poly(age, 3), data=Wage)
fit.4 = lm(wage~poly(age, 4), data=Wage)
fit.5 = lm(wage~poly(age, 5), data=Wage)
fit.6 = lm(wage~poly(age, 6), data=Wage)
fit.7 = lm(wage~poly(age, 7), data=Wage)
fit.8 = lm(wage~poly(age, 8), data=Wage)
fit.9 = lm(wage~poly(age, 9), data=Wage)
fit.10 = lm(wage~poly(age, 10), data=Wage)
anova(fit.1, fit.2, fit.3, fit.4, fit.5, fit.6, fit.7, fit.8, fit.9, fit.10)
```

```
## Analysis of Variance Table
##
## Model  1: wage ~ poly(age, 1)
## Model  2: wage ~ poly(age, 2)
## Model  3: wage ~ poly(age, 3)
## Model  4: wage ~ poly(age, 4)
## Model  5: wage ~ poly(age, 5)
## Model  6: wage ~ poly(age, 6)
## Model  7: wage ~ poly(age, 7)
## Model  8: wage ~ poly(age, 8)
## Model  9: wage ~ poly(age, 9)
## Model 10: wage ~ poly(age, 10)
##    Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1    2998 5022216
## 2    2997 4793430  1    228786 143.7638 < 2.2e-16 ***
## 3    2996 4777674  1     15756   9.9005  0.001669 **
## 4    2995 4771604  1      6070   3.8143  0.050909 .
## 5    2994 4770322  1      1283   0.8059  0.369398
## 6    2993 4766389  1      3932   2.4709  0.116074
## 7    2992 4763834  1      2555   1.6057  0.205199
## 8    2991 4763707  1       127   0.0796  0.777865
## 9    2990 4756703  1      7004   4.4014  0.035994 *
## 10   2989 4756701  1         3   0.0017  0.967529
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The Anova shows us that there is little difference (variance) between the models from three polynomials on. This makes sense because from one to two has a very large change. Two to three is minor, and after that each of the models are pretty much the same in comparison to each other. The good news is that we therefore know since there is not much difference AND four degrees of freedom has the smallest error under the standard deviation rule, that we can comfortably use it.

Let’s plot the polynomial prediction using the best degrees of freedom that we found (four).

```
plot(wage~age, data=Wage, col="darkgrey")
agelims = range(Wage$age)
age.grid = seq(from=agelims[1], to=agelims[2])
lm.fit = lm(wage~poly(age, 4), data=Wage)
lm.pred = predict(lm.fit, data.frame(age=age.grid))
lines(age.grid, lm.pred, col="blue", lwd=2)
```

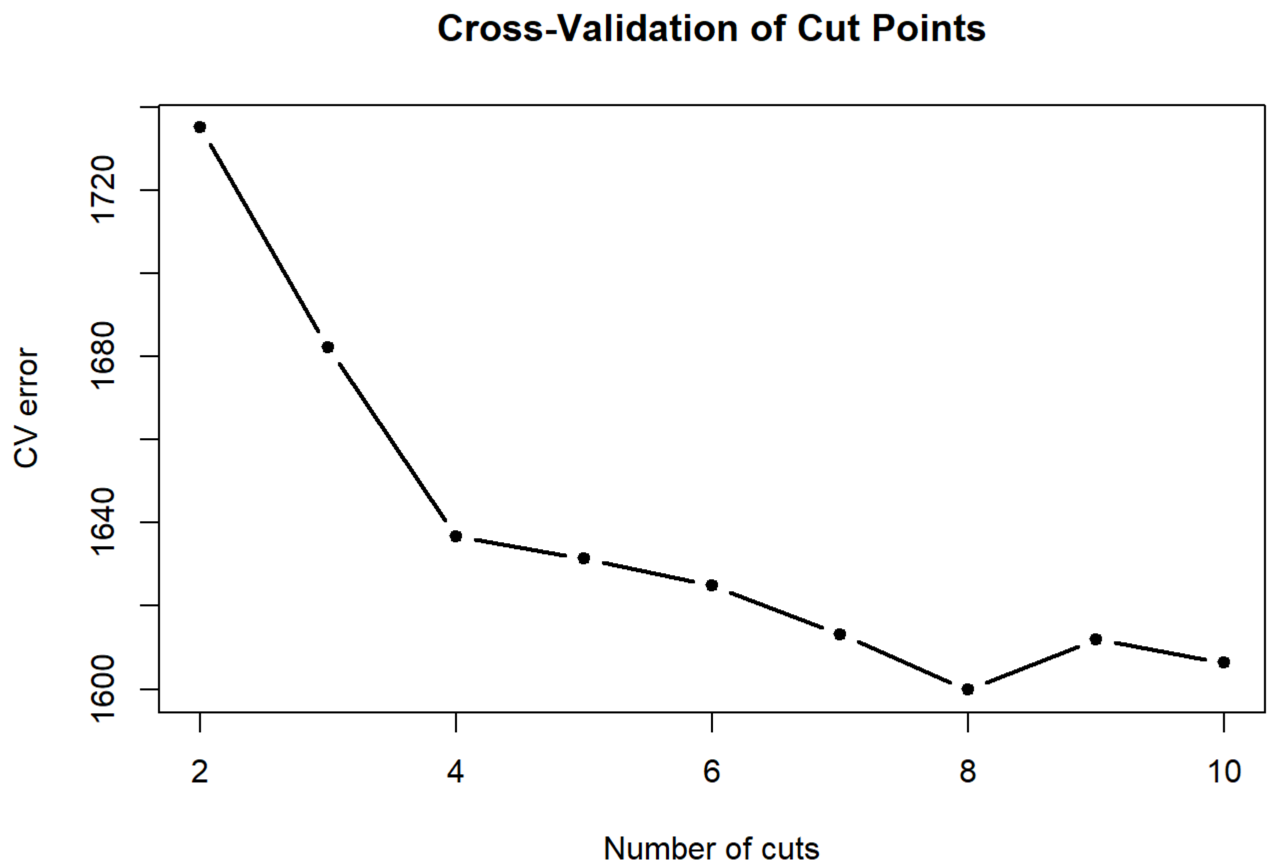


Beautiful.

(b) Fit a step function to predict `wage` using `age`, and perform cross-validation to choose the optimal number of cuts. Make a plot of the fit obtained.

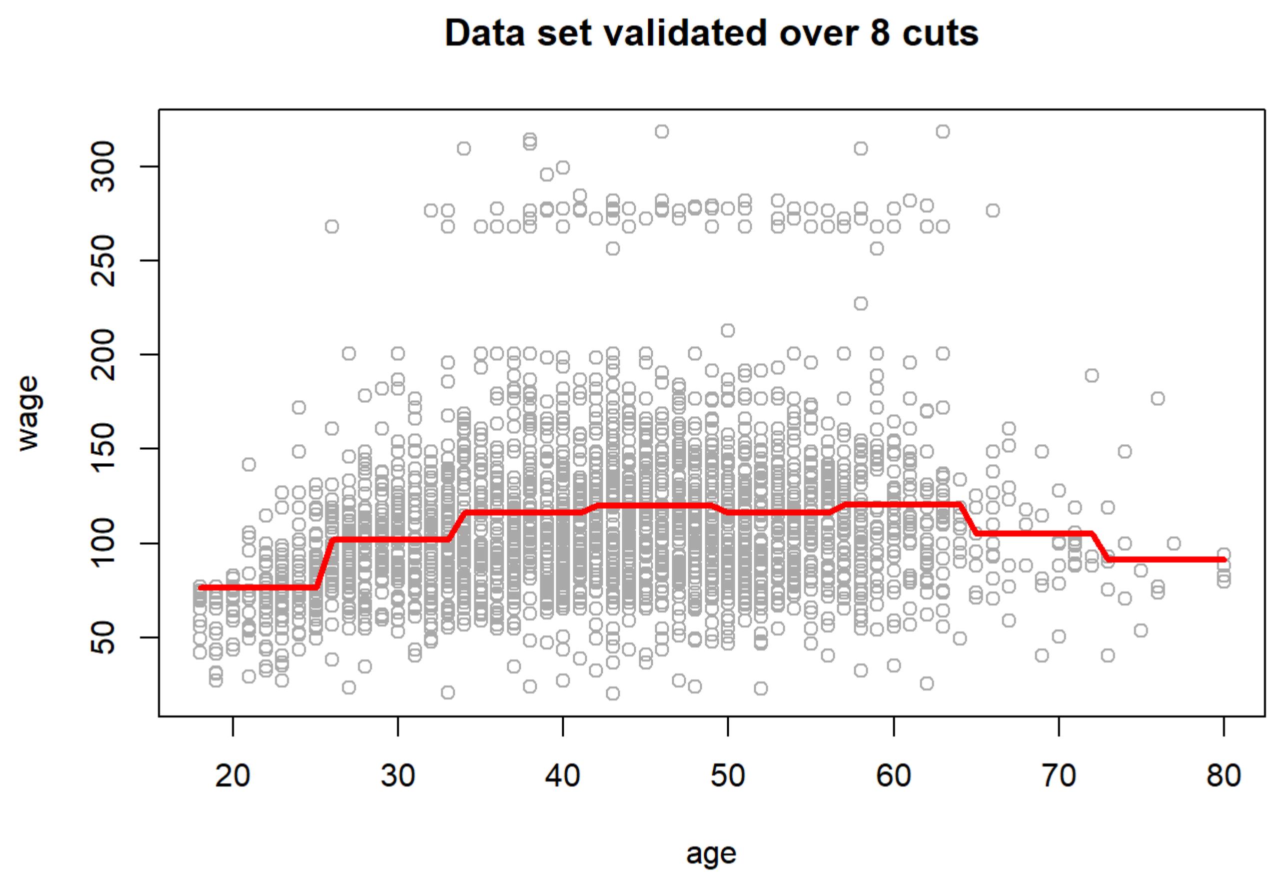
Let's begin by making cut points from 1:10 and seeing how they compare when performing cross-validation. Then we will grab the best when assessing cross-validaiton error.

```
all.cvs = rep(NA, 10)
for (i in 2:10) {
  Wage$age.cut = cut(Wage$age, i)
  lm.fit = glm(wage~age.cut, data=Wage)
  all.cvs[i] = cv.glm(Wage, lm.fit, K=10)$delta[2]
}
plot(2:10, all.cvs[-1], main = "Cross-Validation of Cut Points", xlab="Number of cuts", ylab="CV
error", type="b", pch=20, lwd=2)
```



This shows that cross-validation is lowest when the number of cuts equals 8. Let us use it to train the entire data set and plot it.

```
lm.fit = glm(wage~cut(age, 8), data=Wage)
agelims = range(Wage$age)
age.grid = seq(from=agelims[1], to=agelims[2])
lm.pred = predict(lm.fit, data.frame(age=age.grid))
plot(wage~age, data=Wage, col="darkgrey", main = "Data set validated over 8 cuts")
lines(age.grid, lm.pred, col="red", lwd=3)
```



Chapter 8, Number 10 (Boosting)

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

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

First we will remove any NA's from the Hitter's data set

```
#require(ISLR)
Hitters <- na.omit(Hitters)
Hitters$Salary <- log(Hitters$Salary)
```

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

```
train = 1:200
Hitters.train = Hitters[train, ]
Hitters.test = Hitters[-train, ]
```

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

```r
require(gbm)
```

```
## Loading required package: gbm
```

```
## Warning: package 'gbm' was built under R version 3.6.2
```

```
## Loaded gbm 2.1.5
```

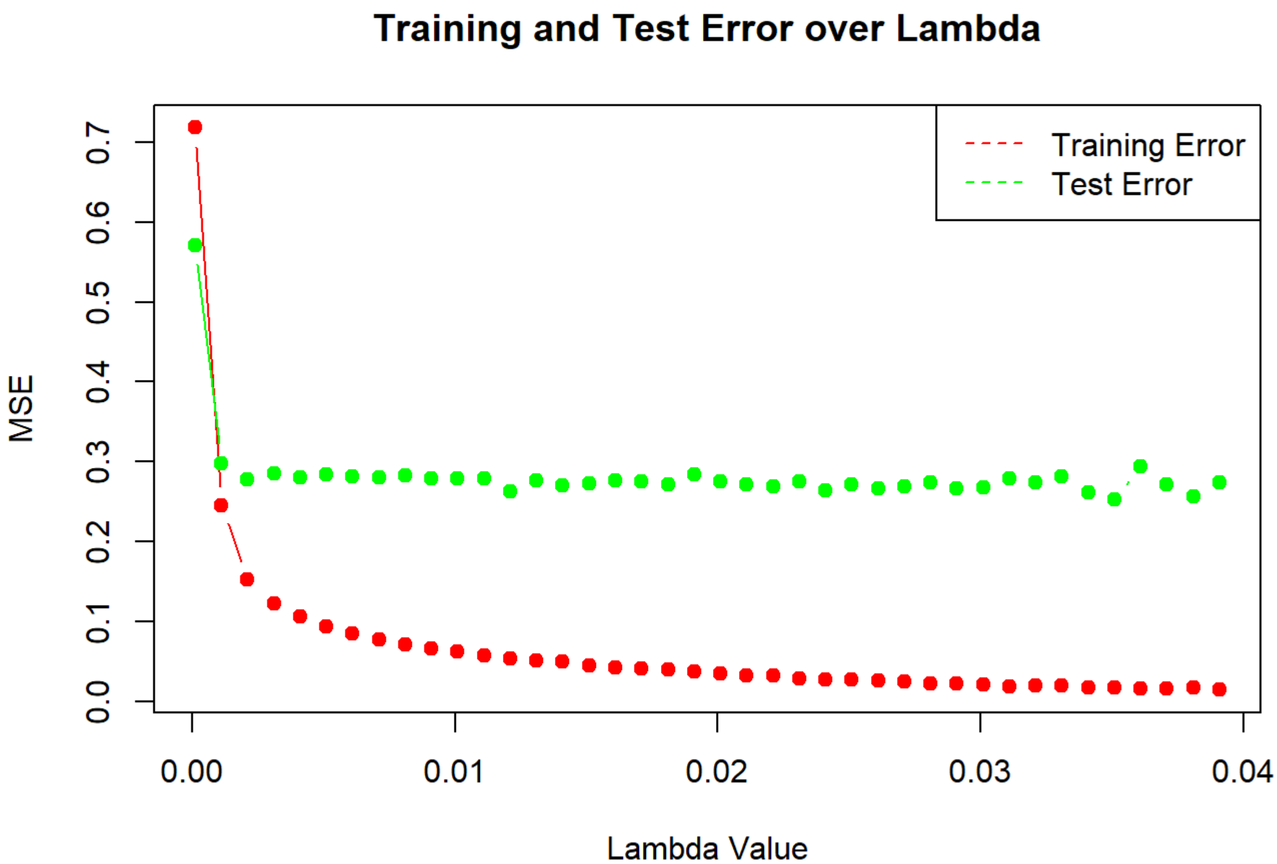
```r
require(glmnet)
```

```
## Loading required package: glmnet
```

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

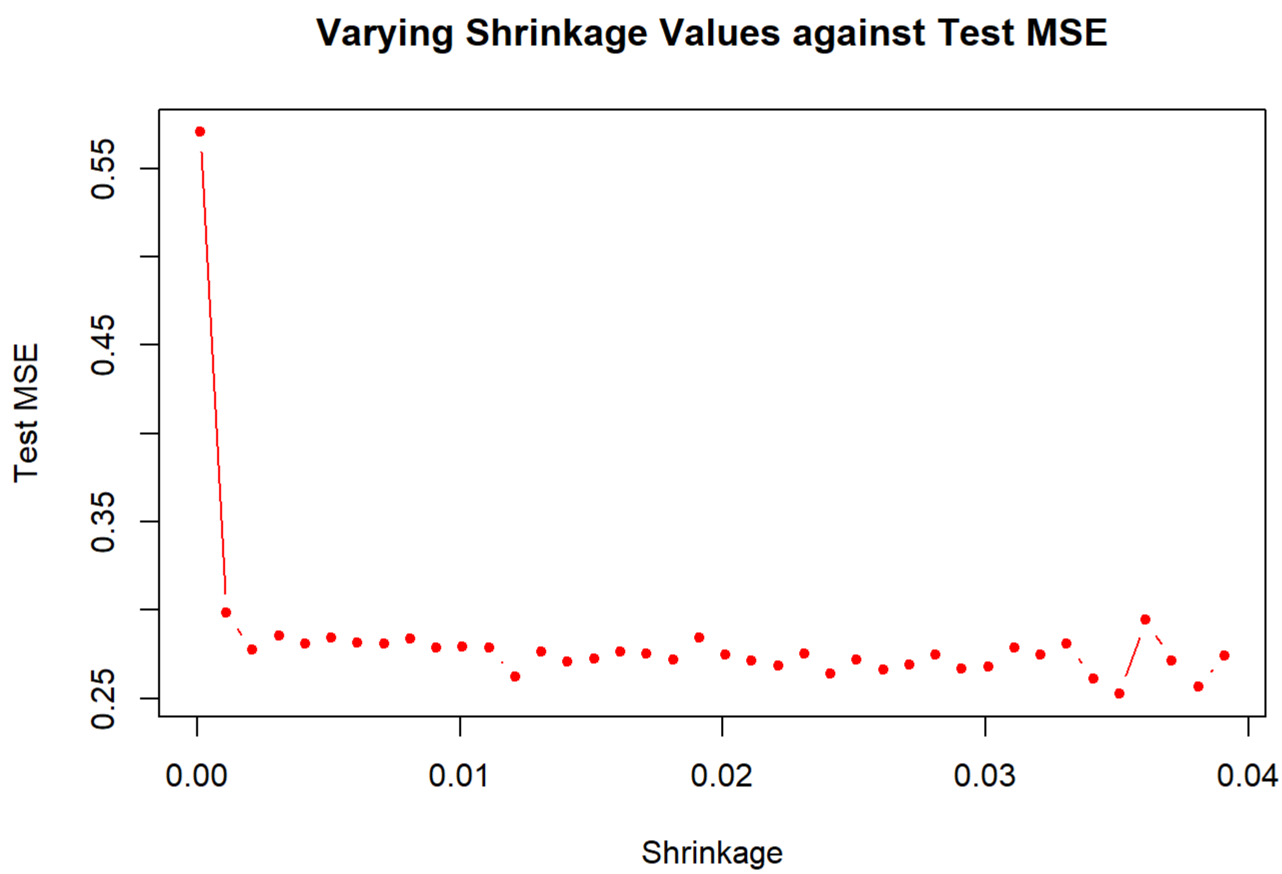
```
## Loaded glmnet 3.0-1
```

```r
lambda_set <- seq(1e-04, 0.04, by = 0.001)

training_mse <- rep(NA, length(lambda_set))
test_mse <- rep(NA, length(lambda_set))
for (i in 1:length(lambda_set)) {
  lm <- lambda_set[i]
  boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian",
                       n.trees = 1000, interaction.depth = 4, shrinkage = lambda_set[i])
  train.pred = predict(boost.hitters, Hitters.train, n.trees = 1000)
  test.pred = predict(boost.hitters, Hitters.test, n.trees = 1000)
  training_mse[i] = mean((Hitters.train$Salary - train.pred)^2)
  test_mse[i] = mean((Hitters.test$Salary - test.pred)^2)
}

plot(lambda_set, training_mse, type = "b", pch = 19, col = "red", xlab = "Lambda Value", ylab =
"MSE", main = "Training and Test Error over Lambda")
lines(lambda_set, test_mse, type = "b", pch = 19, col = "green", xlab = "Lambda Value", ylab =
"Test Set MSE")
legend("topright", legend = c("Training Error", "Test Error"), col = c("red", "green"), lty="das
hed")
```



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

```
plot(lambda_set, test_mse, type = "b", xlab = "Shrinkage", ylab = "Test MSE",
     col = "red", pch = 20, main = "Varying Shrinkage Values against Test MSE")
```




```
lambda_set[which.min(test_mse)]
```

```
## [1] 0.0351
```

(e) 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.

```
lm.fit = lm(Salary ~ ., data = Hitters.train)
lm.pred = predict(lm.fit, Hitters.test)
mean((Hitters.test$Salary - lm.pred)^2)
```

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

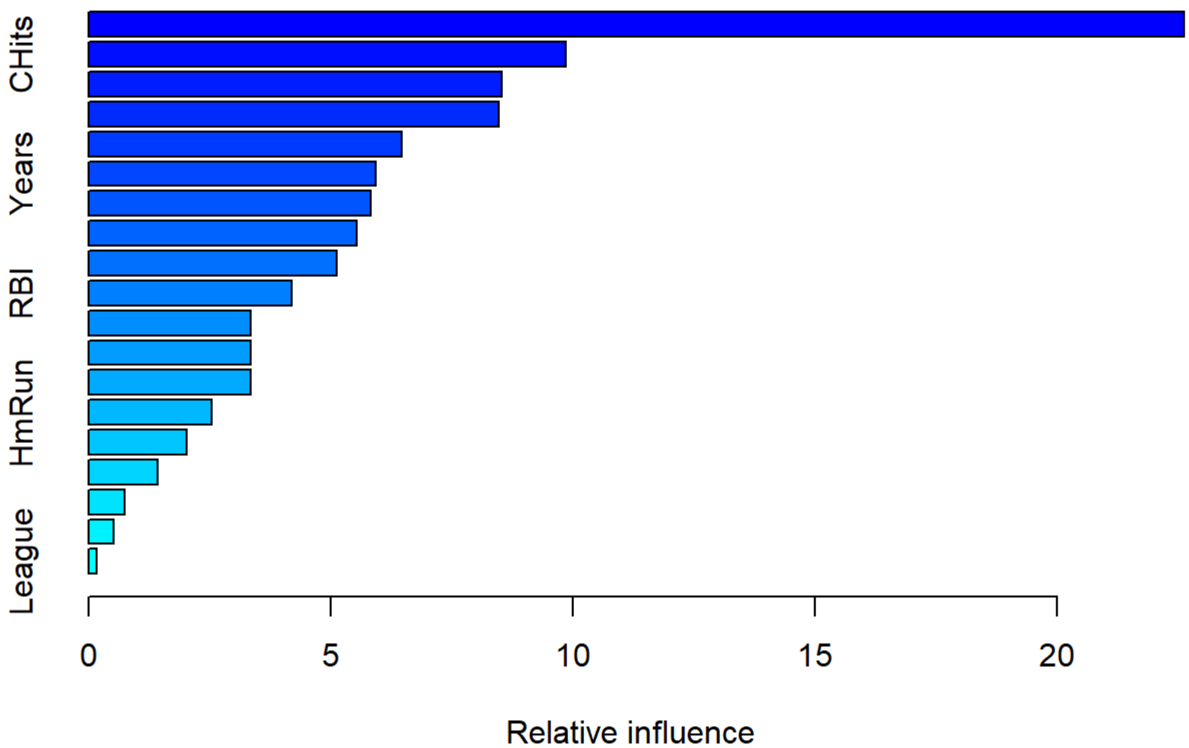
```
x = model.matrix(Salary ~ ., data = Hitters.train)
y = Hitters.train$Salary
x.test = model.matrix(Salary ~ ., data = Hitters.test)
lasso.fit = glmnet(x, y, alpha = 1)
lasso.pred = predict(lasso.fit, s = 0.01, newx = x.test)
mean((Hitters.test$Salary - lasso.pred)^2)
```

```
## [1] 0.4700537
```

Depending on the random seed, LM and Lasso may have higher test MSE's than boosting.

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

```
boost.best = gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian",
    n.trees = 1000, shrinkage = lambda_set[which.min(test_mse)])
summary(boost.best)
```



```
##                 var    rel.inf
## CAtBat       CAtBat 22.6109916
## CHits         CHits  9.8481414
## CRBI           CRBI  8.5357038
## CWalks       CWalks  8.4785966
## CRuns         CRuns  6.4587149
## Years         Years  5.9221699
## CHmRun       CHmRun  5.8174703
## Walks         Walks  5.5404411
## PutOuts     PutOuts  5.1233176
## RBI             RBI  4.1916308
## Assists     Assists  3.3571156
## AtBat         AtBat  3.3458675
## Hits           Hits  3.3429173
## HmRun         HmRun  2.5459926
## Errors       Errors  2.0229294
## Runs           Runs  1.4269608
## Division   Division  0.7403360
## NewLeague NewLeague  0.5181981
## League       League  0.1725046
```

The most important predictors in this model are atbat, runs, and REIs

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

```
require(randomForest)
```

```
## Loading required package: randomForest
```

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

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

```
set.seed(21)
rf.hitters = randomForest(Salary ~ ., data = Hitters.train, ntree = 500, mtry = 19)
rf.pred = predict(rf.hitters, Hitters.test)
mean((Hitters.test$Salary - rf.pred)^2)
```

```
## [1] 0.2303919
```

We find that bagging has almost the same if not just slightly lower test MSE (when considering the random seed)