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, ]
```

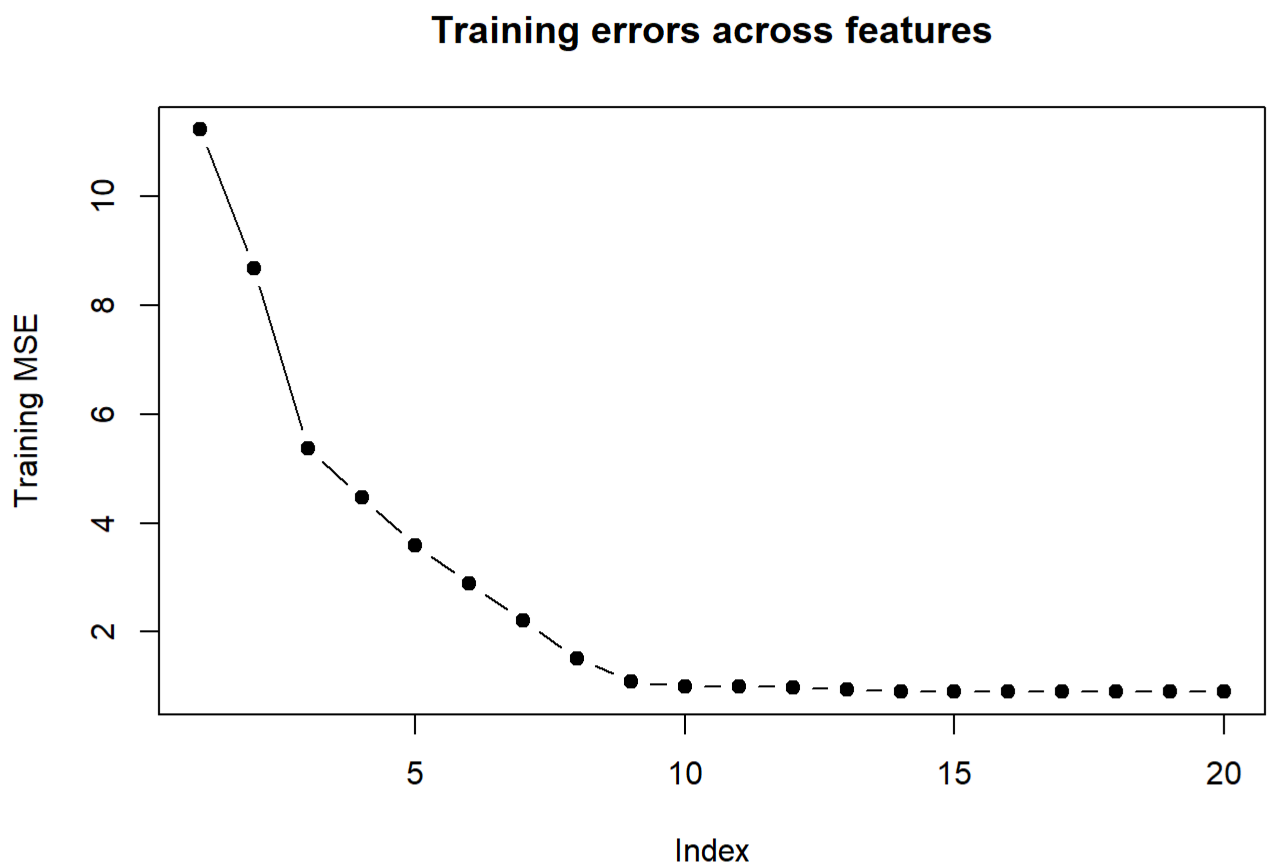
(c) Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size.

```
# 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")
```

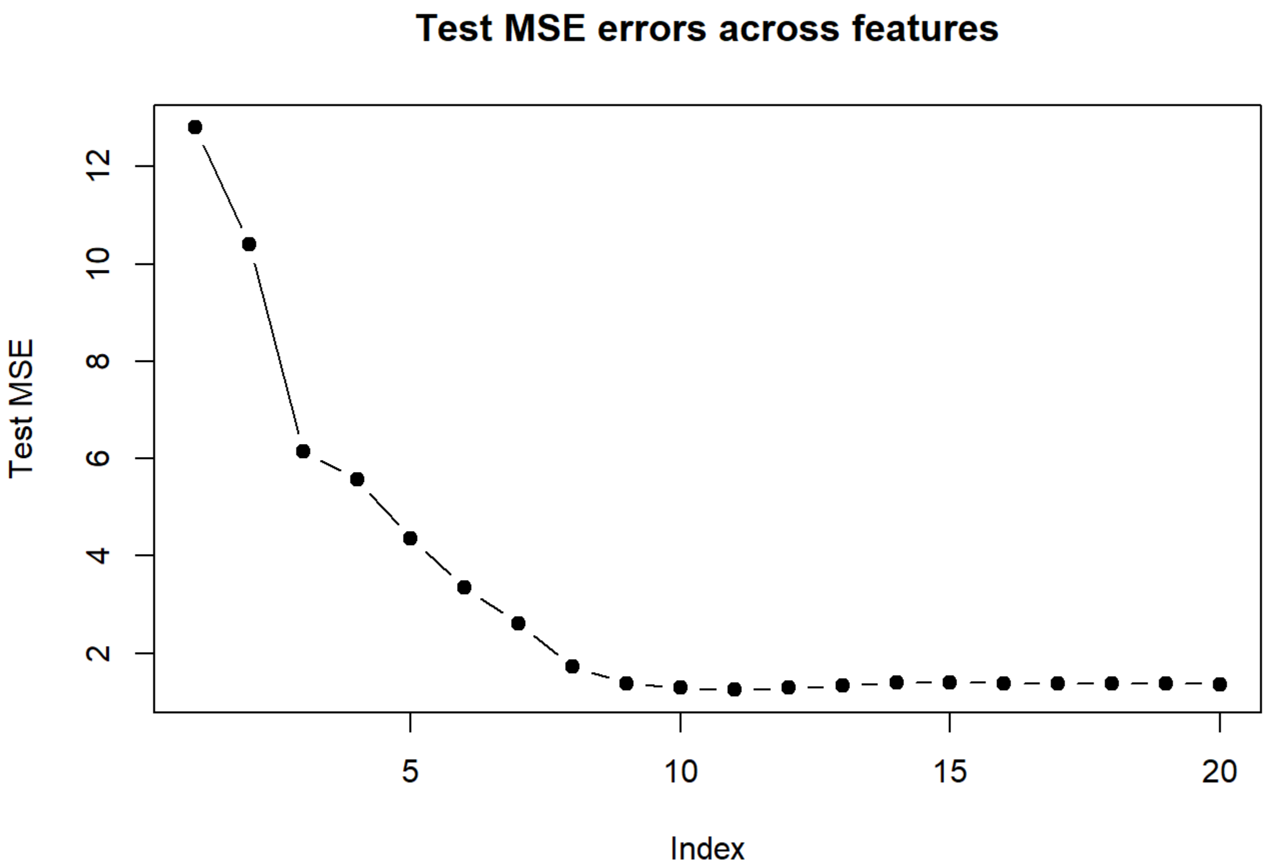



```
which.min(errors)  # min for train error should be at max pred count
```

```
## [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
```

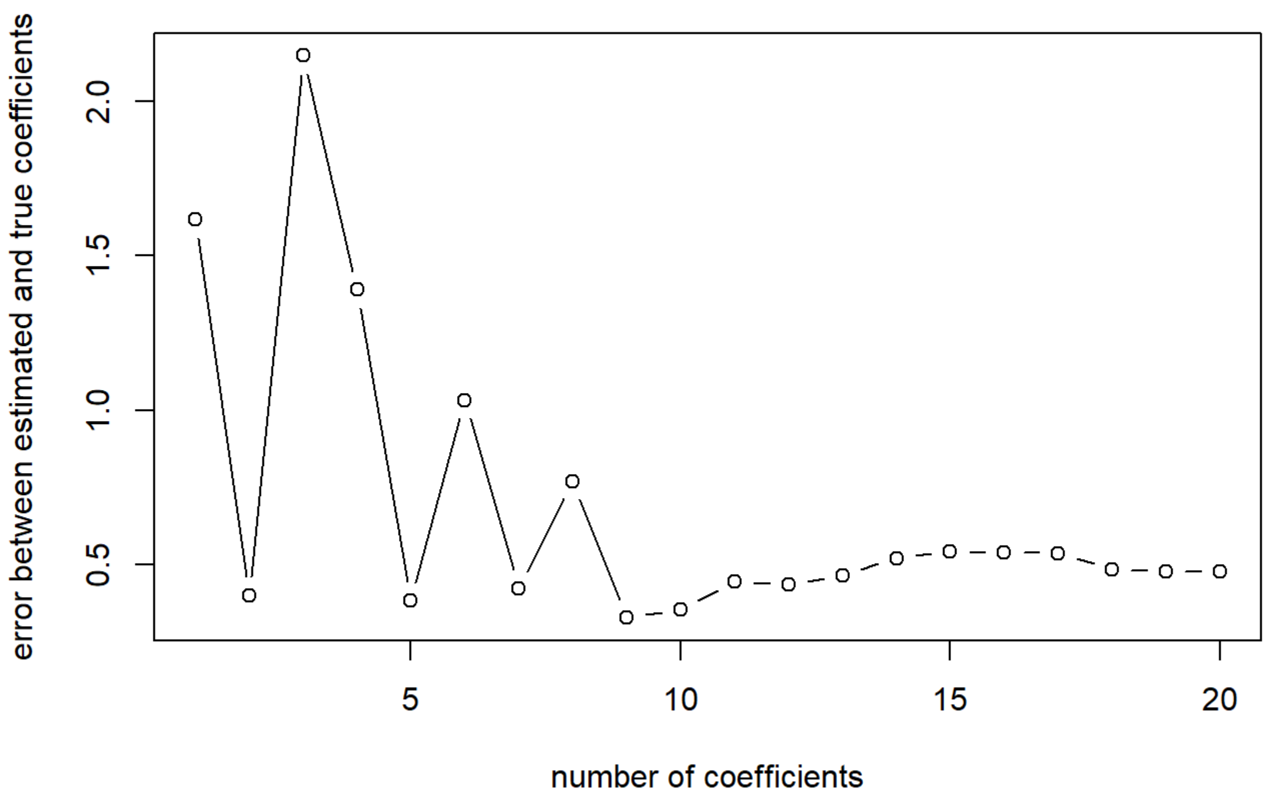
(f) How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient values.

```
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)?

```
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)
```

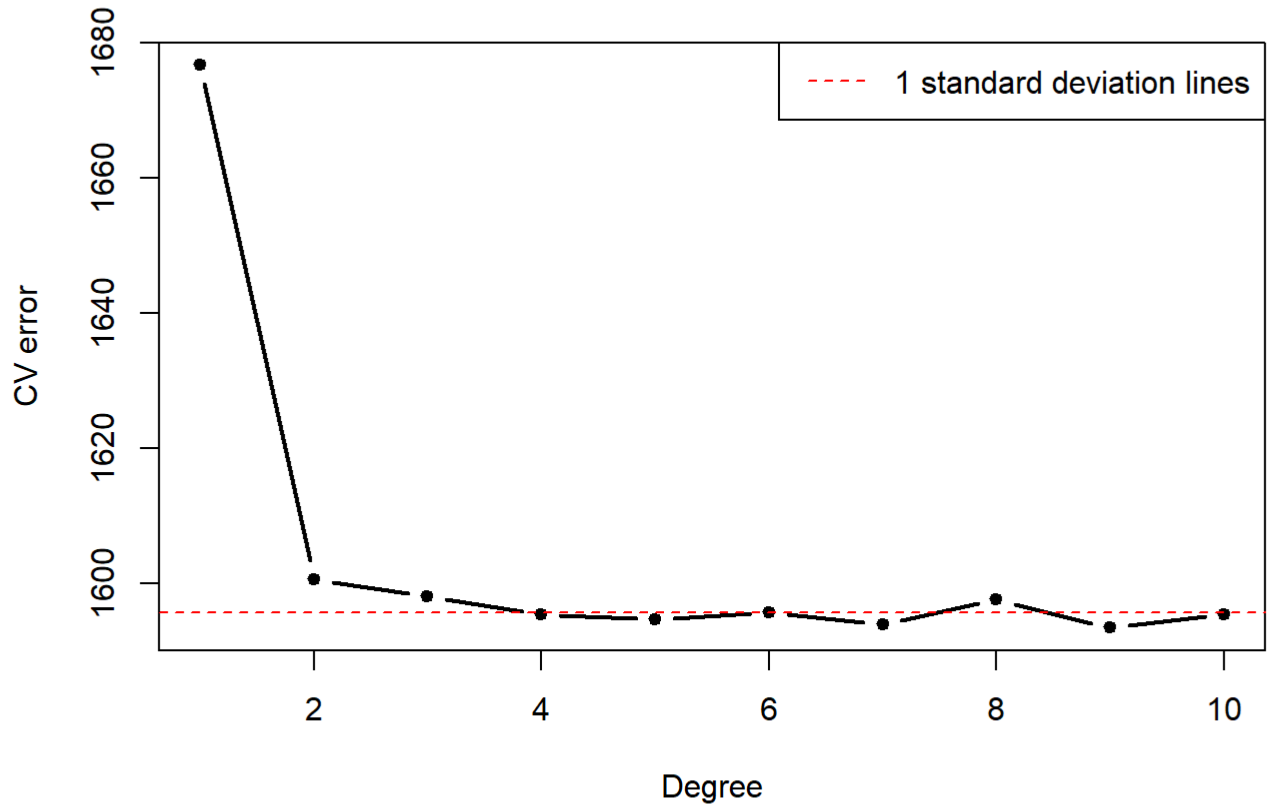
```
## [1] 9
```

```
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.

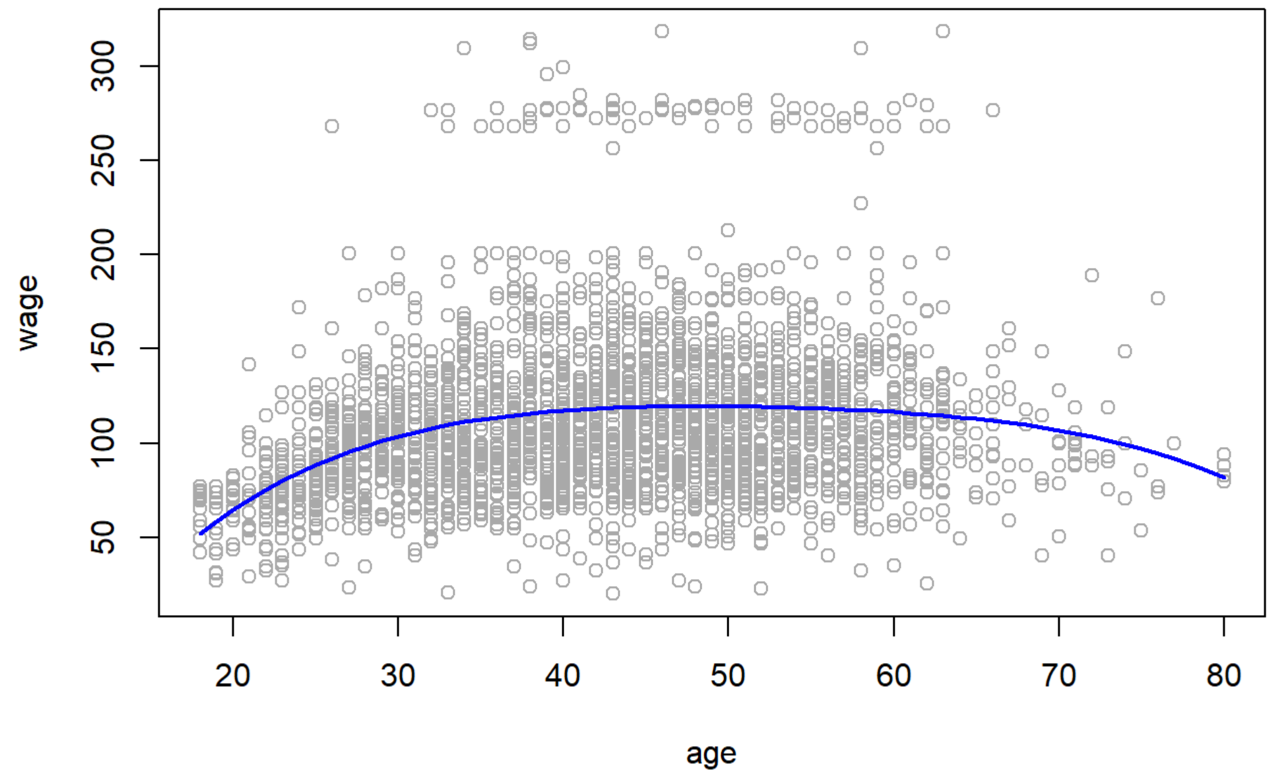
```
fit.1 = lm(wage~poly(age, 1), data=Wage)
fit.2 = lm(wage~poly(age, 2), data=Wage)
fit.3 = lm(wage~poly(age, 3), data=Wage)
fit.4 = lm(wage~poly(age, 4), data=Wage)
fit.5 = lm(wage~poly(age, 5), data=Wage)
fit.6 = lm(wage~poly(age, 6), data=Wage)
fit.7 = lm(wage~poly(age, 7), data=Wage)
fit.8 = lm(wage~poly(age,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

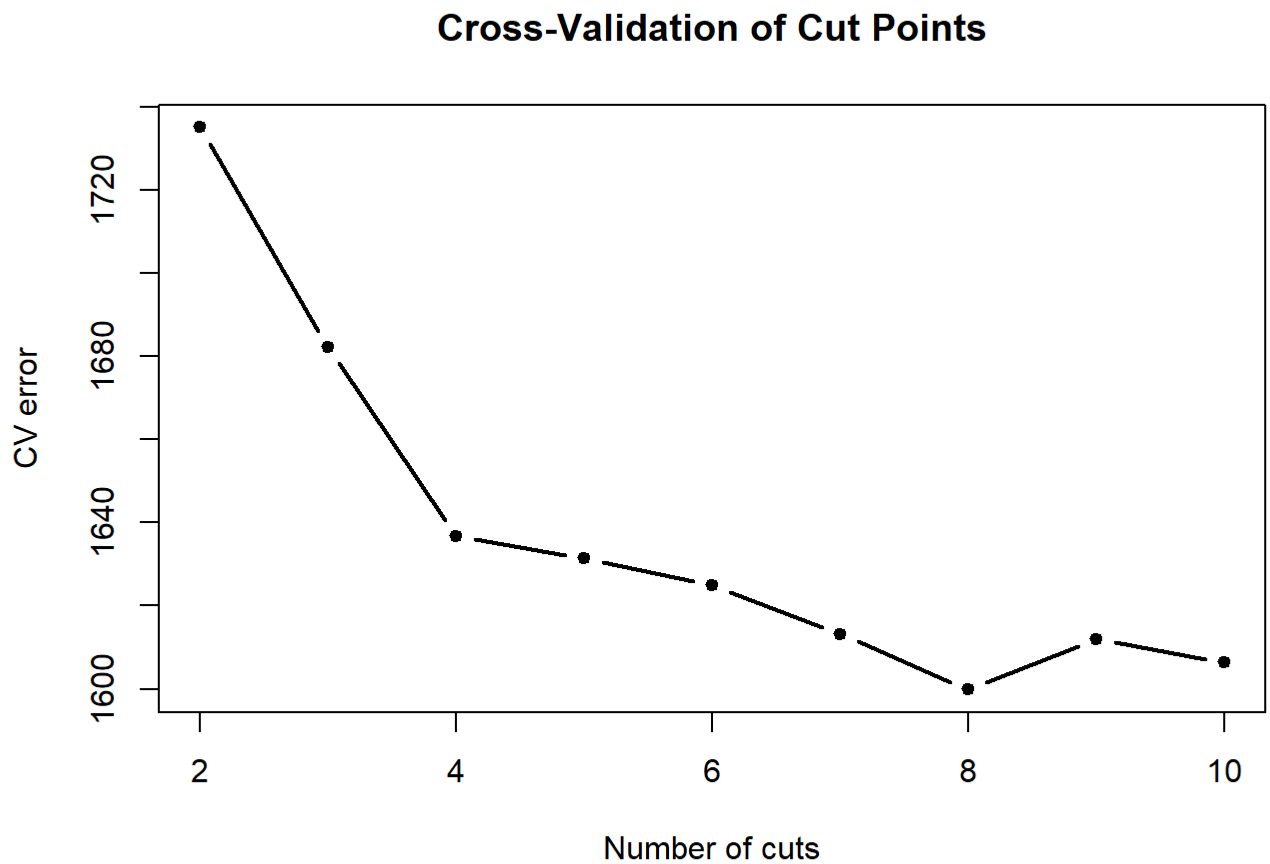


Beautiful.

(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.

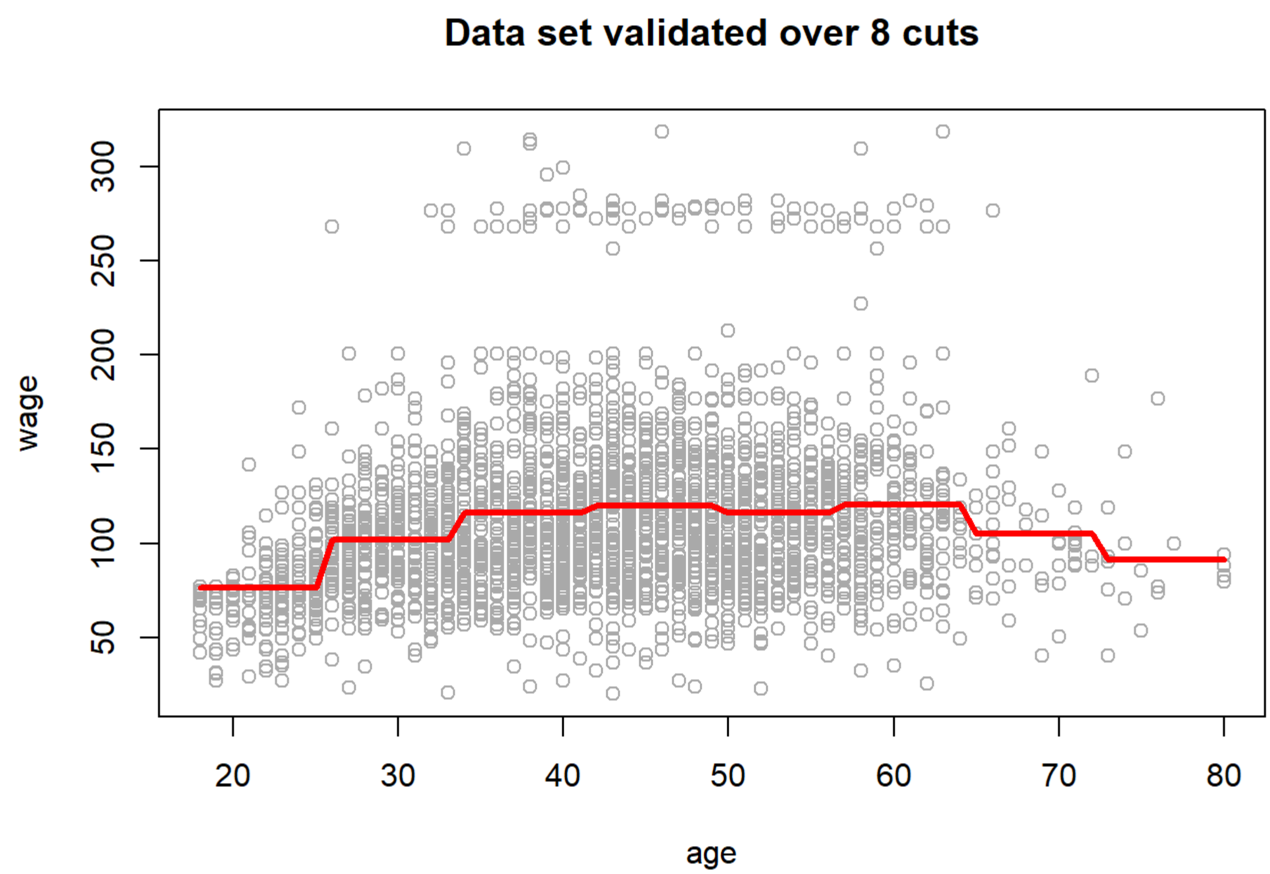
Let's begin by making cut points from 1:10 and seeing how they compare when performing cross-validation. Then we will grab the best when assessing cross-validaiton error.

```
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 9 (Random Forests)

9. This problem involves the `OJ` data set which is part of the `ISLR` package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

Let’s begin by bringing in the required packages

```
require(ISLR)
require(tree)
```

```
## Loading required package: tree
```

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

```
attach(OJ)
```

```
set.seed(5048)

train <- sample(1070, 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
```

(b) Fit a tree to the training data, with `Purchase` as the response and the other variables as predictors. Use the `summary()` function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
oj.datatree <- tree(Purchase ~ ., data = OJ.train)
summary(oj.datatree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "SalePriceMM"   "SpecialCH"     "ListPriceDiff"
## [5] "DiscCH"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7422 = 587.8 / 792
## Misclassification error rate: 0.155 = 124 / 800
```

When we run summary we identify that there are 8 terminal nodes, a training error rate of of 0.155 (Misclassification), and only uses four variables. Let’s compare this with the full data set:

```
summary(OJ)
```

```
##  Purchase WeekofPurchase     StoreID        PriceCH         PriceMM
##  CH:653   Min.   :227.0   Min.   :1.00   Min.   :1.690   Min.   :1.690
##  MM:417   1st Qu.:240.0   1st Qu.:2.00   1st Qu.:1.790   1st Qu.:1.990
##           Median :257.0   Median :3.00   Median :1.860   Median :2.090
##           Mean   :254.4   Mean   :3.96   Mean   :1.867   Mean   :2.085
##           3rd Qu.:268.0   3rd Qu.:7.00   3rd Qu.:1.990   3rd Qu.:2.180
##           Max.   :278.0   Max.   :7.00   Max.   :2.090   Max.   :2.290
##      DiscCH             DiscMM         SpecialCH        SpecialMM
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.05186   Mean   :0.1234   Mean   :0.1477   Mean   :0.1617
##  3rd Qu.:0.00000   3rd Qu.:0.2300   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :0.50000   Max.   :0.8000   Max.   :1.0000   Max.   :1.0000
##     LoyalCH           SalePriceMM     SalePriceCH      PriceDiff       Store7
##  Min.   :0.000011   Min.   :1.190   Min.   :1.390   Min.   :-0.6700   No :714
##  1st Qu.:0.325257   1st Qu.:1.690   1st Qu.:1.750   1st Qu.: 0.0000   Yes:356
##  Median :0.600000   Median :2.090   Median :1.860   Median : 0.2300
##  Mean   :0.565782   Mean   :1.962   Mean   :1.816   Mean   : 0.1465
##  3rd Qu.:0.850873   3rd Qu.:2.130   3rd Qu.:1.890   3rd Qu.: 0.3200
##  Max.   :0.999947   Max.   :2.290   Max.   :2.090   Max.   : 0.6400
##    PctDiscMM        PctDiscCH       ListPriceDiff       STORE
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.000   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.140   1st Qu.:0.000
##  Median :0.0000   Median :0.00000   Median :0.240   Median :2.000
##  Mean   :0.0593   Mean   :0.02731   Mean   :0.218   Mean   :1.631
##  3rd Qu.:0.1127   3rd Qu.:0.00000   3rd Qu.:0.300   3rd Qu.:3.000
##  Max.   :0.4020   Max.   :0.25269   Max.   :0.440   Max.   :4.000
```

So we note that our tree uses less than 1/4 of all the variables.

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

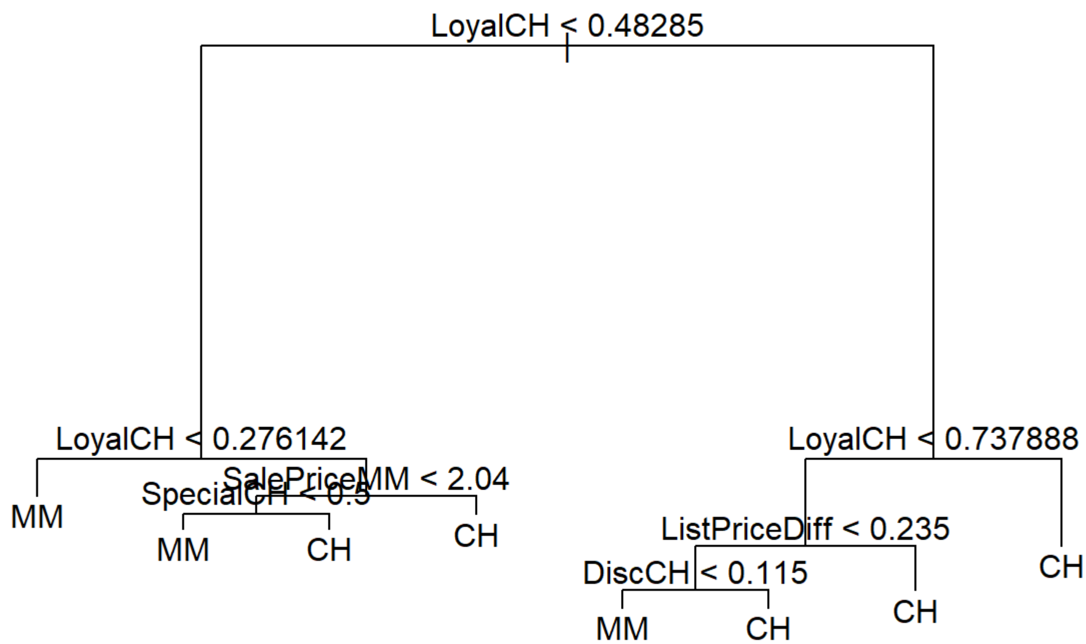
```
oj.datatree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1077.00 CH ( 0.60000 0.40000 )
##    2) LoyalCH < 0.48285 304  315.50 MM ( 0.21382 0.78618 )
##      4) LoyalCH < 0.276142 172  115.30 MM ( 0.10465 0.89535 ) *
##      5) LoyalCH > 0.276142 132  171.90 MM ( 0.35606 0.64394 )
##       10) SalePriceMM < 2.04 74   77.27 MM ( 0.21622 0.78378 )
##         20) SpecialCH < 0.5 60   47.12 MM ( 0.13333 0.86667 ) *
##         21) SpecialCH > 0.5 14   19.12 CH ( 0.57143 0.42857 ) *
##       11) SalePriceMM > 2.04 58   80.13 CH ( 0.53448 0.46552 ) *
##    3) LoyalCH > 0.48285 496  441.60 CH ( 0.83669 0.16331 )
##      6) LoyalCH < 0.737888 216  269.10 CH ( 0.68519 0.31481 )
##       12) ListPriceDiff < 0.235 84  115.70 MM ( 0.45238 0.54762 )
##         24) DiscCH < 0.115 76  102.00 MM ( 0.39474 0.60526 ) *
##         25) DiscCH > 0.115 8    0.00 CH ( 1.00000 0.00000 ) *
##       13) ListPriceDiff > 0.235 132  118.90 CH ( 0.83333 0.16667 ) *
##      7) LoyalCH > 0.737888 280  105.20 CH ( 0.95357 0.04643 ) *
```

What we see here are multiple nodes. If we identify terminal node 20 (Special CH < 0.5) we see that there are a total of 60 points in the subtree. We see that 47.12 is the listed deviance for all points in the subtrees. 13.33% have CH as a value of sales with 86.67% having MM as a value in sales.

(d) Create a plot of the tree, and interpret the results.

```
plot(oj.datatree)
text(oj.datatree, pretty = 0)
```



We note that the most important variable is LoyalCH which makes up the top three nodes. When LoyalCH is > 0.27 the data tree predicts CH and when LoyalCH is < 0.27, it predicts MM. Sale Price also plays a role in the prediction.

(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```
oj.pred <- predict(oj.datatree, OJ.test, type = "class")
table(OJ.test$Purchase, oj.pred)
```

```
##      oj.pred
##        CH  MM
##   CH 145  28
##   MM  25  72
```

(f) Apply the `cv.tree()` function to the training set in order to determine the optimal tree size.

```
cv.oj = cv.tree(oj.datatree, FUN = prune.tree)
```

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
plot(cv.oj$size, cv.oj$dev, type = "b", main = "Pruned Tree Size", xlab = "Tree Size", ylab = "E
rror Rate (Deviance)")
```

(h) Which tree size corresponds to the lowest cross-validated classification error rate?

Let’s see what happens if we use which.min and why this can be confusing.

```
which.min(cv.oj$dev)
```

```
## [1] 2
```

Now if we logically look at the graph, we absolutely see that 2 is not the mininum. Let’s output the numbers and perhaps we can see why.

```
cv.oj$dev
```

```
## [1]  726.8037  709.0376  709.1789  709.1789  726.0450  724.5061  787.9184
## [8] 1079.1785
```

```
cv.oj$size
```

```
## [1] 8 7 6 5 4 3 2 1
```

Note that the numbers are actually counting down. So the second index from cv.oj$dev corresponds with a size of 7 (not two). Weird huh? Thus we determine that our appropriate tree is at 7, though we could logically pick 6 or 5 and be at basically the same amount of error.

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
oj.pruned <- prune.tree(oj.datatree, best = 7)
```

(j) Compare the training error rates between the pruned and unpruned trees. Which is higher?

```
summary(oj.pruned)
```

```
##
## Classification tree:
## snip.tree(tree = oj.datatree, nodes = 10L)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "SalePriceMM"   "ListPriceDiff" "DiscCH"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7551 = 598.8 / 793
## Misclassification error rate: 0.1575 = 126 / 800
```

We note that our misclassificaiton error rate is 0.1575 versus our original of 0.155. Thus let us compare with a best of 6 or 5 since they are so close.

6: Misclassification error rate: 0.1675 = 134 / 800 5: Misclassification error rate: 0.1725 = 138 / 800

Turns out, 7 is the best with the lowest error rate.

The pruned tree has a slightly higher error rate than the unpruned tree (0.155)

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

```
pred.unpruned <- predict(oj.datatree, OJ.test, type = "class")
misclass.unpruned <- sum(OJ.test$Purchase != pred.unpruned)
misclass.unpruned/length(pred.unpruned)
```

```
## [1] 0.1962963
```

```
pred.pruned <- predict(oj.pruned, OJ.test, type = "class")
misclass.pruned <- sum(OJ.test$Purchase != pred.pruned)
misclass.pruned/length(pred.pruned)
```

```
## [1] 0.2037037
```

Pruned and unpruned have almost the same test error rate (0.19629 and 0.2037)