A nearly full set of proposed solutions for An Introduction to Statistical Learning, with Applications in R, by James, Witten, Hastie, and Tibshirani.

by Mark Romanowsky

This document can be found at my GitHub account:

https://github.com/mbromanowsky/ISLR_solutions

Exercises

3.7.1: Describe the null hypotheses in Table 3.4

For each predictor, the null hypothesis is of the form, "H_0: there is no relationship between <predictor> and sales". Based on the p-values, we can say with high confidence that H_0 is rejected for TV and radio, but not rejected for newspaper. The null hypothesis for the intercept is something along the lines of "H_0: absent any advertising, sales will be zero", which is rejected at high confidence.

3.7.2: Carefully explain the differences between the KNN classifier and KNN regression methods.

A principal difference is that the classifier is applied to qualitative responses, and the regression is applied to quantitative responses. This means that the classifier can return only values in the specified classes, but the regression can interpolate. Otherwise they are quite similar.

3.7.3:

a) Writing out the interactions, I find that Y_F - Y_M = 35 - 10 X_1, so for fixed values of GPA and IQ, males make more than females provided their GPA's are high enough.
b) A female with IQ=110 and GPA=4.0 will make, on average, 137.1.
c) FALSE: a small coefficient for an interaction effect doesn't indicate lack of evidence for the interaction. That comes from the p-value, which is not given in this exercise.

3.7.4: For a set of n=100 observations of one predictor and one quantitative response, suppose we fit both a linear and a cubic model.

a) Suppose the true relationship between X and Y is linear (plus noise). For the TRAINING data, which model will have the lower residual sum of squares (RSS)?
    a) It will be the cubic, but probably only by a tiny bit. Cubic is more flexible and can reduce to linear in the worst case, but will likely overfit the noise a little.
b) What about the TEST data RSS?
    a) We should expect linear to have lower test RSS because it is close to the real model and won't overfit.
c) Suppose the true relationship between X and Y is nonlinear but we don't know how much. Now, for the TRAINING data, which model will have lower RSS?
    a) Again it will be the cubic, both because it can overfit the noise, and because it will be a better approximation to whatever the true function is.

d) What about TEST?
   a) Now I would expect the cubic model to be lower RSS, because it's closer to the correct functional form.

3.7.5: (paper proof) $a_{i'} = \frac{x_i x_{i'}}{\sum_{k=1}^{n} x_k^2}$ which looks a bit like a correlation function. I guess it's notable because it doesn't involve the y's at all?

3.7.6: Trivially show that, for simple linear regression, the regression line always passes through $(\bar{x}, \bar{y})$. Just evaluate the line at x-bar.

3.7.7: (paper proof) Easy enough…

3.7.8:

Basic linear fit. There is a highly significant relationship between horsepower and mpg, which has a negative coefficient that is large enough to matter a lot. It is somewhat nonlinear, but also data become sparser at the higher values of horsepower. According to the fit, if horsepower = 98, then confidence and prediction intervals are (fit, min, max) = (24.467, 23.973, 24.961) and (24.467, 14.809, 34.125).

3.7.9: code that does the things
a) pairs(Auto)
b) cor(Auto[,-9])
c) lm.fit = lm(mpg ~. -name, data=Auto). Results show significant influence of displacement, weight, year, and origin. Displacement coefficient is positive, weight negative, year positive, origin positive, which all make some sense except I don't know what "origin" means.
d) plot(lm.fit) for diagnostic plots: mild nonlinearity perhaps, but not too much difference in standardized residuals with value. There are a couple of points with leverage much higher than the bulk, esp. points 14 and… some other one.
e) To do everything interacting with everything, this works:

lm.fit1 = lm(mpg~(.-name)*(.-name), data=Auto)

Then I trimmed the highest p-values until everything remained significant, and this list was: displacement, horsepower, acceleration, year, origin, displacement:year, horsepower:year, acceleration:origin. However, a lot of collinearity remained, judging from sky-high VIF values.
f) lm.fitsquare = lm(I(mpg^2)~.-name-…) — quite a bit worse R^2 than the fit from e.

lm.fitsqrt = lm(sqrt(mpg)~.-name-…) — slightly better R^2 than the fit from e. ANOVA won't compare the models because the predictors are different.

lm.fitpoly = lm(mpg~(.-name-…)^2+I(displacement^2)+…), then trimmed to only the significant terms, gives a noticeably better fit than the fit from e. The ANOVA F-statistic is over 86, indicating a definitively better fit. The significant terms are displacement, disp^2, horsepower, hp^2, acceleration, year, year^2, origin, and acceleration:origin. HP^2 seemed to be a major contributor, based on the path I took getting here. The issues with leverage remained, and the residuals may have been non-normally-distributed, based on a goofy quantile plot / Q-Q plot: the upper tail appears fat.

3.7.10:
a) lm.fit = lm(Sales~Price+Urban+US, data=Carseats)
b) interpret the coefficients:
   a) price = -0.0545, so Sales goes down as Price goes up. Units are k$ sales at location / $ price at location
   b) UrbanYes = -0.0219, so Urban locations have -0.219 k$ less sales than Rural

    c) USYes = 1.201, so US location have 1.201 k$ more sales than non-US

c) $Sales = 13.043 - 0.0545Price - 0.022\delta_{i,Urban} + 1.201\delta_{j,US}$

d) For which predictors can we reject null hypothesis? Just Price and US.

e) Refit the model based on just the significant predictors: lm.fit2 = lm(Sales~Price+US, …)

f) How well do these models fit the data? Not wonderfully: the first fit has R^2 = 0.2335, the second has R^2 = 0.2354 (both “adjusted R-squared”)

g) Using lm.fit2, get 95% confidence intervals for coefficients: confint(lm.fit2), yields

```
a) (Intercept) 11.79032020 14.27126531
b) Price       -0.06475984 -0.04419543
c) USYes        0.69151957  1.70776632
```

h) Is there evidence of high leverage points or outliers? No clear outliers, but there is a point of fairly high leverage because its Price value is only 24, much lower than anything else. 1st quartile value is 100, median is 117, 3 quartile is 131. Second lowest is 50, and there is no other big gap in the price distribution.

3.7.11:

a) Regress y onto x with no intercept: lm(y~x+0). Yields
$\hat{\beta} = 1.9939 \pm 0.1065, t = 18.73, p < 2e - 16$

b) Regress x onto y: yields $\hat{\beta} = 0.39111 \pm 0.02089, t = 18.73, p < 2e - 16$

c) Outside of noise, these two regressions test the same relationship, so it’s good that the t-values are similar / identical, and the fit values are close to the expected 2x apart.

d) (paper proof, not doing it)

e) The given formula is symmetric in x and y, so exchange the roles of predictor and observed variable won’t change the t-value at all. So the t-values in a and b are indeed identical.

f) Experimentally, and I infer also theoretically, the t-values for slope are also equal when fitting with an intercept, whether you go x onto y or the reverse.

3.7.12:

a) When fitting with no intercept, when will $\hat{\beta}$ be the same whether regressing X onto Y or the reverse? Answer: when $\sum x_i^2 = \sum y_i^2$

b) Example when they are different: see exercise 3.7.11

c) Example when they are the same: x=rnorm(100), y=abs(x) (not that this is a very good fit)

3.7.13: just show and tell for the most part. Utility function to make this easier:

```
noiseball = function(noise){
   set.seed(1)
   x=rnorm(100)
   eps=rnorm(100,sd=noise)
   y=-1 + 0.5*x + eps
   plot(x,y)
   lm.fit=lm(y~x)
   abline(lm.fit)
   abline(a=-1,b=0.5,col='red')
   print(summary(lm.fit))
   print(confint(lm.fit))
}
```

3.7.14:

a) The model is $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$, and the coefficients are 2, 2, 0.3.

b) The correlation between x1 and x2 is pretty high, with a correlation coefficient of 0.835 and a linear-looking plot.

c) Fitting both x1 and x2: fit is weak, with adjusted R-squared only 0.19, no significant fit for x2, and fit values of 2.13, 1.44, 1.01.
d) Fitting only x1: adj R^2= 0.19, no better, but now the fits are all extremely low p-value and the coefficients are 2.11 and 1.98, close to accurate.
e) Fitting only x2: adj R^2 = 0.167, lower presumably because we fit through two layers of noise terms, but still all the p-values are low. The coefficient for x2 is 2.900, which seems high! But from visual inspection, it's plausible. Error is 0.633.
f) Since x1 and x2 are so collinear, trying to fit to both at once doesn't work well.
g) Adding an "outlier" point, now I get a significant fit on x2 but not for x1, when both are fit. The last point is an outrageous outlier and high leverage point, at leverage 0.4 and Cook's distance 1.0. When fitting just to x1, the quality of the fit declines but the overall properties don't change much. This measurement isn't an outlier. When fitting just to x2, the fit quality actually appears better but the x2 coefficient estimate is inflated a lot. In this case, the added point has high leverage but isn't really an outlier: it sits close to the existing trend line.

3.7.15:

a) Try to predict per capita crime rate "crim" using one predictor at a time. Only "chas" looks like a total dud.
    a) vs zn: low quality fit, R^2<0.05, but also low p-values. Most of the data has zn=0.
    b) vs indus: somewhat better, R^2=0.16, low p.
    c) vs chas: clearly no fit
    d) vs nox: R^2=0.17, low p.
    e) vs rm: R^2<0.05, low p
    f) vs age: R^2=0.12, low p
    g) vs dis: R^2=0.14, low p
    h) vs rad: R^2=0.39, low p
    i) vs tax: R^2=0.34, low p
    j) vs ptratio: R^2=0.08, low p
    k) vs black: R^2=0.15, low p
    l) vs stat: R^2 = 0.21, low p
    m) vs med: R^2=0.15, low p
b) Now do multiple regression on all predictors at once. Result is R^2 = 0.45, very low p-values for dis and rad and medv, significant p for zn and black, marginal for nox and lstat.
c) too boring to do this problem…

4.7.7: do a calculation for a Bayes theorem prediction. Want to predict Yes or No for stock issuing dividend, based on last year's percent profit X. Suppose:

- $\hat{\mu}_Y = 10$
- $\hat{\mu}_N = 0$
- Common variance $\hat{\sigma}^2 = 36$
- $\hat{\pi}_Y = 0.8, \quad \hat{\pi}_N = 0.2$
- If X follows a normal distribution, predict probability of a dividend given that X=4.

Use Bayes theorem:

$p_k(x) = \dfrac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$, so $f_Y(x) = \dfrac{1}{\sqrt{2\pi}\sigma_Y} \exp\Big(-\dfrac{1}{2\sigma_Y^2}(x-\mu_k)^2\Big)$

Evaluate to 75.19%, if I didn't make a mistake….

4.7.10:

a) Make scatterplot matrix with pairs(Weekly) and correlations with cor(Weekly[,-9]). There is a clear trend of rising Volume with Year. Values of all LagX and Today are in a wider band near 1990, 2000, and 2008 than in the rest of the set, which may be the signature of the market crashes / recessions of 1990-1991, 2001, and 2008-2009.

b) Do logistic regression on the full set using all LagX plus Volume to predict Direction. Result: Lag2 is significant with p=0.0296, which is kind of amazing. Lag1 has p=0.1181, not bad for stock market data. Everything else is p>0.25 or much worse. plot(glm.fit) produced a lot of plots that were confusing.

c) Make the confusion matrix and interpret it:
  a) Direction
  b) glm.pred Down Up
  c) Down 54 48
  d) Up 430 557
  e) Looks like my model is very cheerful and predicts that the market will usually go up, about 90% of the time! Actual market goes up about 60% of the time, so the direction of bias is correct, but quite extreme. At least the model is more than 50% correct; I get about 56% correct. However, predicting it will always go up gives a 55.5% rate, so more work to be done.

d) Repeating with just Lag2, training on Year = [1990, 2008], applying to test set Year=[2009,2010]
  a) Fit to Lag2 is significant with p=0.043
  b) Confusion matrix:
   a) Direction
   b) glm.pred2 Down Up
   a) Down 9 34
   b) Up 5 56
   c) Accuracy is now 62.5%! Model still predicts market will usually go up, about 86% of the time. But it's getting slightly better.

e) Repeat like (d) but using LDA
  a) No measure of significance generated here. The difference between the groups is not terribly visible in the plots.
  b) Confusion matrix and accuracy are precisely equal to the logistic regression results. Since the models are so similar, I guess that's to be expected?

f) Repeat like (d) but using QDA
  a) Now it predicts every week will be "Up"! Accuracy is 58%. Confusion matrix is very confused.
  b) Direction.test
  c) Down Up
  d) Down 0 0
  e) Up 43 61

g) Repeat like (d) but using KNN with K=1
  a) Accuracy = 50% — not too good! But somewhat better at predicting actual down weeks than the others.
  b) knn.pred
  c) Direction.test Down Up
  d) Down 21 22
  e) Up 30 31

h) The most accurate models are logistic regression / LDA (tied). The most likely to correctly call down weeks is KNN.

i) Repeating with K=3:
  a) knn.pred3 Down Up
  b) Down 16 19
  c) Up 27 42
  d) Accuracy: 58%

4.7.11:

a) Create and bind a binary variable mpg01, which is 1 when mpg > median(mpg) and 0 when not. To do this:

```
mpg01 = rep(0,392)
mpg01[mpg>median(mpg)]=1
Auto01 = data.frame(Auto, mpg01)
```

b) Which variables associate with mpg01?

a) Based on scatterplot matrix, looks like mpg (obviously!), displacement, horsepower, weight, acceleration.

b) Based on box plots grouped by mpg01, looks like strong association to weight, horsepower, displacement, and cylinders, with non-overlapping middle-halves, and weaker association to year and acceleration. Origin also seems weirdly strong. 1=American, 2=European, 3=Japanese, and the large majority of below-median cars have origin = 1 (173 out of 196). Based on correlation values, the top factors are cylinders, weight, displacement, horsepower, then origin at 0.51, and the rest below 0.5. Note that the top 4 are themselves all highly correlated, with cor > 0.84 in all pairs.

c) Split the data into test and training halves. Do this:

```
a) > set.seed(1)
b) > train = sample(1:392, 196)
c) > Auto01.train = Auto01[train,]
d) > Auto01.test = Auto01[-train,]
```

d) Do LDA on training data, predicting mpg01 using the apparent best predictors from (b). What is the test error rate? Do this:

```
a) lda.fit=lda(mpg01~cylinders+weight+displacement+horsepower, data=Auto01,
   subset=train)
b) lda.pred = predict(lda.fit, Auto01.test)
c) > table(lda.pred$class, Auto01.test$mpg01)
d)
e)       0  1
f)     0 78  5
g)     1 15 98
h) > mean(lda.pred$class == Auto01.test$mpg01)
i) [1] 0.8979592
```

j) Accuracy is 89.8%! Seems great. Test error rate is 10.2%

e) Repeat with QDA. Results are a little worse than LDA:

```
a) > qda.fit=qda(mpg01~cylinders+weight+displacement+horsepower, data=Auto01,
   subset=train)
b) > qda.pred = predict(qda.fit, Auto01.test)
c) > table(qda.pred$class, Auto01.test$mpg01)
d)
e)       0  1
f)     0 77  8
g)     1 16 95
h) > mean(qda.pred$class != Auto01.test$mpg01)
i) [1] 0.122449
```

f) Repeat with logistic regression. Almost the same as LDA, as usual.

```
a) > glm.fit=glm(mpg01~cylinders+weight+displacement+horsepower, data=Auto01,
   subset=train, family=binomial)
b) > glm.probs=predict(glm.fit, Auto01.test, type='response')
c) > glm.pred=rep(0,196)
d) > glm.pred[glm.probs>0.5]=1
e) > table(glm.pred, Auto01.test$mpg01)
f)
```

    - g) glm.pred 0 1
    - h)   0 79 6
    - i)   1 14 97
    - j) > mean(glm.pred != Auto01.test$mpg01)
    - k) [1] 0.1020408
- g) Repeat with KNN for various K. Standardize scales first!!
    - a) > standardized.X=scale(Auto01[,-9])
    - b) > train.X.stand = standardized.X[train, 2:5]
    - c) > test.X.stand = standardized.X[-train, 2:5]
    - d) Now do k=1: error is 10.2%
        - a) > set.seed(1)
        - b) > knn.pred1 = knn(train.X.stand, test.X.stand, train.Y, k=1)
        - c) > table(knn.pred1, Auto01.test$mpg01)
        - d)
        - e) knn.pred1 0 1
        - f)   0 81 8
        - g)   1 12 95
        - h) > mean(knn.pred1 != Auto01.test$mpg01)
        - i) [1] 0.1020408
    - e) Now k=3: error 8.7%, better!
        - a) > set.seed(1)
        - b) > knn.pred3 = knn(train.X.stand, test.X.stand, train.Y, k=3)
        - c) > table(knn.pred3, Auto01.test$mpg01)
        - d)
        - e) knn.pred3 0 1
        - f)   0 82 6
        - g)   1 11 97
        - h) > mean(knn.pred3 != Auto01.test$mpg01)
        - i) [1] 0.08673469
    - f) Now k=10: error of 8.2%.

Rest of 4.7 is pretty easy and/or boring…

5.4.1: pretty simple pencil and paper proof

5.4.2: What is the probability that a particular observation is in a bootstrap sample? Assume n original observations.

- a) Prob(first bootstrap sample is not observation j) = (n-1)/n — very likely.
- b) Prob(second bootstrap sample is not observation j) = (n-1)/n or 1 - 1/n
- c) Prob(observation j is not any bootstrap sample) = (1 - 1/n)^n
- d) For n=5, this probability is: 32.8%. So observation j IS IN the bootstrap sample with prob 67.2%.
- e) For n=100, this prob is 36.6%, so observation j IS IN the bootstrap sample with prob 63.4%
- f) For n = 10000, probs are 36.8% and 63.2%
- g) Plot the prob of inclusion vs n from 1 to 100,000. (I tried to take the limit by hand, but it's a tricky one.) It definitely approaches an asymptote around 63.2%, which is probably not coincidentally very close to 1 - 1/e.
- h) Experimentally for n=10,000, I got 64.7%, which is pretty close.

5.4.3: Review of k-fold cross-validation.

- a) Explain how k-fold cross-validation is implemented: Randomly divide the training data T into k equal sets. For each set s_i, perform the statistical fit on T-s_i, then calculate the fit accuracy applied to s_i. Rate variability of the fit overall by averaging accuracy over all values 1 <= i <= k.

b) What are pros and cons of k-fold cross-validation relative to:
    a) The validation set approach? Choosing a single validation set is computationally cheaper, but risks bias if the validation set is not representative, e.g. has outliers. k-fold has k times the compute cost but treats every data point equally, as training but also as test data.
    b) LOOCV? LOOCV is computationally more expensive yet that k-fold CV, because it runs a number of fits equal to the number of data points; the exception is for least-squares linear or polynomial regression, in which it actually costs no more than a single fit. But it uses almost all the data for every fit, and generally fits go better with more data, so they tend to estimate the fit error rate more accurately. Also, given a data set, LOOCV is completely deterministic, while k-fold CV has the initial random division, which can cause the results to be irreproducible. Finally, there is a bias-variance tradeoff at work, which means LOOCV tends to have lower bias but higher variability than kFCV, and k=5 or k=10 is empirically a nice middle ground.

5.4.4: Suppose that we use some statistical learning method to make a prediction for the response Y for a particular value of the predictor X. Carefully describe how we might estimate the standard deviation of our prediction.
A: Conceptually, we need to run the fit many times on training data that is somehow "representative", and see how much the predictions at X vary. I don't assume we know the true Y(X). I don't assume that we have a way to generate unlimited training data. So we have to use our fixed pool of training data in some way that suggests how much variability it imparts to the fit, which means, at minimum, we can't just do a single fit using all the training data. Both LOOCV and kFCV provide ways to subset the data and perform some number of fit attempts (in this case, n-1 for LOOCV and k for kFCV). The bootstrap gives another way, which is very flexible, based on resampling with replacement.

5.4.5: Logistic regression on the "Default" data set, predicting "default" based on "income" and "balance", as done in Chapter 4 text. Now we will estimate test error using the validation set approach.

a) Fit a logistic regression model.

    ```
    set.seed(1)
    glm.fit <- glm(default ~ income+balance, data=Default, family=binomial)
    summary(glm.fit)
    ```

    results: high significance fits for income and balance

b) Split the data set into training and validation portions, then estimate the test error of the model.
    A. Split the data set. I chose equal portions for training and validation.

        ```
        set.seed(1)
        train <- sample(1:10000, 5000)
        Default.train1 <- Default[train, ]
        Default.test1 <- Default[-train, ]
        ```

    B. Fit multiple logistic regression model using only the training portion.

        ```
        glm.fit1 <- glm(default~income+balance, data = Default.train1, family=binomial)
        summary(glm.fit1)
        ```

        results: significant fits for both, but lower significance than before, and estimates are different values.

C. Obtain 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 prob > 0.5.

Initialize and then fill classification vector:

```
glm.pred1 <- rep(“No”, 5000)
glm.probs1 <- predict(glm.fit1, Default.test1, type=“response”)
glm.pred1[glm.probs1 > 0.5] <- “Yes”
```

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

Confusion matrix and mean accuracy:

```
table(glm.pred1, Default.test1$default)
mean(glm.pred1 == Default.test1$default)     -> gives 0.9714.
```

So error rate is 2.86%. Note that default rate in the test set is 80/5000 = 1.60%. It would be nicer to do better… of the actual 167 defaults, I predicted only 50, with 28 false positives.

C. Repeat the process in (B) three times, using 3 different splits into training / validation. Comment on the results obtained.

Repeat 1: 2.36% error rate, predicted 63 of the actual 167 defaults and 18 false positives.
Repeat 2: 2.80% error rate, true positives 50, false positives 24.
Repeat 3: 2.68% error rate, true positives 47, false positives 11.

Test error rate has a mean of 2.675%, with std dev of 0.22%.

D. Now consider a model default~income+balance+student. Estimate the test error and comment on whether the added variable helps.

Go through similar keystrokes, except let’s keep the same training/validation divisions.

```
glm.fit1s <- glm(default~income+balance+student, data=Default, family=binomial,
subset=train)
glm.pred1s <- rep("No", 5000)
glm.probs1s <- predict(glm.fit1s, Default.test1, type='response')
glm.pred1s[glm.probs1s > 0.5] <- "Yes"
table(glm.pred1s, default[-train])
mean(glm.pred1s != default[-train])
```

Result: 2.88% error, TP = 53, FP = 30.
For partition 2: 2.36% error, TP = 63, FP = 18
For partition 3: 2.78% error, TP = 50, FP = 23
For partition 4: 2.74% error, TP = 44, FP = 11

So it looks as if adding “student” didn’t really move the needle either way, except adding complexity. The p-values for both Income and Student were now non-significant.

5.4.6: Continue with the Default data set fitting default~income+balance, still using logistic regression, but now estimate standard errors of fit coefficients using (1) the bootstrap, and (2) the standard formulas out of the glm() function. Remember random seed!

A. Use glm() and summary() to estimate standard errors.

```
results:
income: fit to 2.081e-05, std err = 4.985e-06
balance: fit to 5.647e-03, std err = 2.274e-04
```

B. Write a boot.fn() that takes input data from Default as well as an index of the observations, and that outputs the coefficient estimates from logistic regression.

```
boot.fn = function(data, index){
  return(coef(glm(default~income+balance, data=data, family=binomial, subset=index)))
}
set.seed(1)
```

C. library(boot)

```
boot(Default, boot.fn, 1000)    -> do 1000 rounds of complete resampling
```

```
results (took about 2 min to run):
        original        bias     std. error
t1* -1.154047e+01 -8.008379e-03 4.239273e-01   -> intercept
t2*  2.080898e-05  5.870933e-08 4.582525e-06     -> income
t3*  5.647103e-03  2.299970e-06 2.267955e-04     -> balance
```

D. Std error values for bootstrap and logistic model are very comparable in this case. If the glm() estimate had been much lower than the bootstrap estimate, it might have suggested that the linear model was not very good.

5.4.7: Roll-your-own LOOCV, to see how it really works. Use the Weekly data set.

A. Fit a glm() model for Direction~Lag1+Lag2.

```
glm.fit <- glm(Direction~Lag1+Lag2, data=Weekly, family=binomial)
```

B. Fit a glm() model of the same type, using all but the first data point.

```
glm.fit1 <- glm(Direction~Lag1+Lag2, data=Weekly[-1,], family=binomial)
```

C. Use the model from (B) to predict the direction of the first observation. Was this observation correctly classified?

```
predict(glm.fit1, Weekly[1,], type='response')
```

result: 0.5714…, actual was “Down”, so this observation was incorrectly classified.

D. Write a for loop from i=1..n, where n= number of observations, that does this:
   A. Fit Direction~Lag1+Lag2 using all data except for observation i.
   B. Compute the posterior prob of the market moving up for observation i.
   C. Use the posterior prob to predict whether the market moves up.
   D. Determine whether a prediction error was made for observation i. If error, mark as 1, otherwise mark as 0.

```
loocv <- function(data){
  results <- rep(0, 1089)
  for(i in 1:1089){
    glm.fit <- glm(Direction~Lag1+Lag2, data=Weekly[-i,], family=binomial)
    glm.prob <- predict(glm.fit, Weekly[i,], type='response')
    if(glm.prob > 0.5){
      pred <- "Up"
    }
    else {
      pred <- "Down"
    }
    if(Weekly$Direction[i] != pred){
      results[i] <- 1
    }
  }
  return(results)
}
```

E. Take the average of the list returned by the above to get an estimate for the LOOCV test error, and comment.

mean(results) = 0.4499541
This error rate is comparable to the error rate I got in 4.7.10.d, which used a training set of all years except 2009-2010 and computed the error rate on 2009-2010.

5.4.8: cross-validation on simulated data

A. Generate simulated data:

```
set.seed(1)
y=rnorm(100)
x=rnorm(100)
y=x-2*x^2+rnorm(100)
df <- data.frame(x,y)
```

Here, n=100 and p=1

B. Make a scatterplot and comment.

As could be anticipated from the equation, it’s a noisy upside-down parabola.

C. Set a random seed, and compute LOOCV errors that result from fitting this data with four models using least squares:

  A. linear
  B. quadratic
  C. cubic
  D. quartic

Follow the template given on pp. 192-193.

```
set.seed(1)
library(boot)
cv.error<-rep(0,5)
for (i in 1:5){
glm.fit <- glm(y~poly(x,i), data=df)
cv.error[i] <- cv.glm(df, glm.fit)$delta[1]
}
cv.error
```

result: for order 1 to 5, errors (5.890979 1.086596 1.102585 1.114772 1.131163)

D. Repeat using another random seed and compare.

set.seed(2), etc.

result: for order 1 to 5, errors were (5.890979 1.086596 1.102585 1.114772 1.131163). Same! I should have expected it: LOOCV is deterministic. I got different values when I generated new data using a new seed, but not when rerunning the first data.

E. Which model had the smallest LOOCV error? Is it as expected?

result: quadratic model by a good bit, which I expected because the underlying model is actually quadratic..

F. Comment on the statistical significance of the coefficient estimates that result from fitting each model in ©. Do these results agree with the LOOCV results and conclusions?

result: The linear model has a non-significant coefficient (p=0.329). For all the higher orders, the first order coefficient gets, p~0.027, the second order coefficient gets p<2e-16, and the higher orders get non-significant fits. This lines up with LOOCV, which showed a massive improvement in error going from linear to quadratic, then a slight increase but mostly flat going to higher orders.

5.4.9: Back to the Boston housing data set.

A. Based on this data set, provide an estimate $\hat{\mu}$ for the population mean of “medv”.

```
library(MASS)
library(boot)
mean(Boston$medv)   -> 22.53281
```

B. Provide an estimate for the standard error of $\hat{\mu}$, and interpret.

According to the hint, standard error of the sample mean is the sample standard deviation divided by $\sqrt{n}$, so:

```
sd(Boston$medv)/sqrt(length(Boston$medv))  -> 0.4088611
```

C. Now estimate standard error of $\hat{\mu}$ using the bootstrap, and compare.

```
set.seed(1)
boot.fn=function(data, index) return(mean(data[index]))
boot(Boston$medv, boot.fn, 1000)
```

result:
0.4119374, rather close to the value from standard formula. As it should be, as far as I know.

D. Based on part C, provide a 95% confidence interval for $\hat{\mu}$. Compare it to t-test result.

Based on the hint, the 95% C.I. is +/- 2 S.E., so [21.70894, 23.35668]. For t-test,

t.test(Boston$medv)   -> [21.72953, 23.33608], again rather close to bootstrap value.

E. Based on this data set, provide estimate $\hat{\mu}_{med}$ for the median value of “medv” in the population.

```
median(Boston$medv)   -> 21.2.
```

F. We want to estimate std error of $\hat{\mu}_{med}$ but there is no standard formula to follow. But we can still use the bootstrap.

```
boot.fn <- function(data, index) return(median(data[index]))
set.seed(1)
boot(Boston$medv, boot.fn, 1000)
```

result: 0.3801002

G. Based on this data set, provide estimate for the tenth percentile of "medv"; call it $\hat{\mu}_{0.1}$.

   quantile(Boston$medv, 0.1)   -> 12.75

H. Use the bootstrap to estimate standard error of this estimate. Comment.

```
boot.fn <- function(data, index) return(quantile(data[index],0.1))
set.seed(1)
boot(Boston$medv, boot.fn, 1000)
```

   0.505056, which is quite a bit higher than what I got for median. This makes intuitive sense to me, because the extremes of the distribution are likelier to change shape when resampling than the very middle.

6.8.1: Suppose we do best subset, forward stepwise, and backwards stepwise selection on the same data set, obtaining a series of p+1 model for each, k=0,1,…,p.

A. At each k, smallest training RSS is definitely in the best subset model.
B. At each k, smallest test RSS could possibly be in any of the three models. Test data can deviate from training data in whatever contrary way.
C. True or false
   1. TRUE, the forward-k predictors are always a subset of the forward-k+1 predictors.
   2. TRUE, the backward-k predictors are always a subset of the backward-k+1 predictors.
   3. FALSE, backward-k may not be a subset of forward-k+1.
   4. FALSE, forward-k may not be a subset of backward-k+1.
   5. FALSE, best-k may not be a subset of best-k+1.

6.8.2: Indicate which set of the statements 1-4 is correct for each scenario.

A. Lasso, relative to least squares: iii
B. Ridge relative to least squares: iii
C. Nonlinear methods relative to least squares: ii

6.8.3: NOTE ERRATA: these error functions should be sums of squared quantities, so these are exactly the lasso and the ridge regression situations, in the alternative s-based and lambda-based expressions. Indicate which statements in 1-5 are correct for each scenario.

A. iv: as the fit becomes more flexible, training RSS will always decrease
B. ii: as the fit becomes more flexible test RSS will usually decrease but then begin to increase agin as overfitting plays out
C. iii: as the fit becomes more flexible, variance will steadily increase
D. iv: as the fit becomes more flexible, bias will steadily decrease.
E. v: nothing about the fit affects the irreducible error, by definition.

6.8.4: same errata. As lambda increases from 0, the fit becomes less flexible.

A. iii
B. ii
C. iv
D. iii
E. v

6.8.5: pencil/paper and drawing.
6.8.6: pencil/paper
6.8.7: I didn't see how to get started, maybe in part from some confusion with the terminology and unfamiliarity with Bayes theorem. I followed the derivation on the solutions site I found without much trouble, though.
6.8.8: simulated data used for best subset selection

A. Use the following code snippets

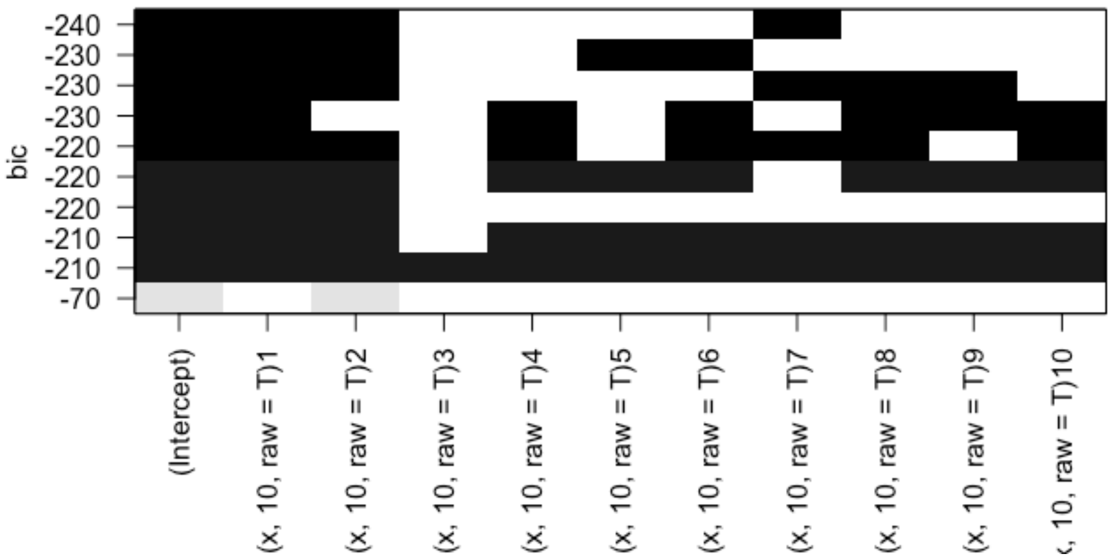
```
> set.seed(1)
> x=rnorm(100)
> eps <- rnorm(100)
```

B. I picked values of β_i={0.5, 1.5, 2, 0.3} for i=0,1,2,3:

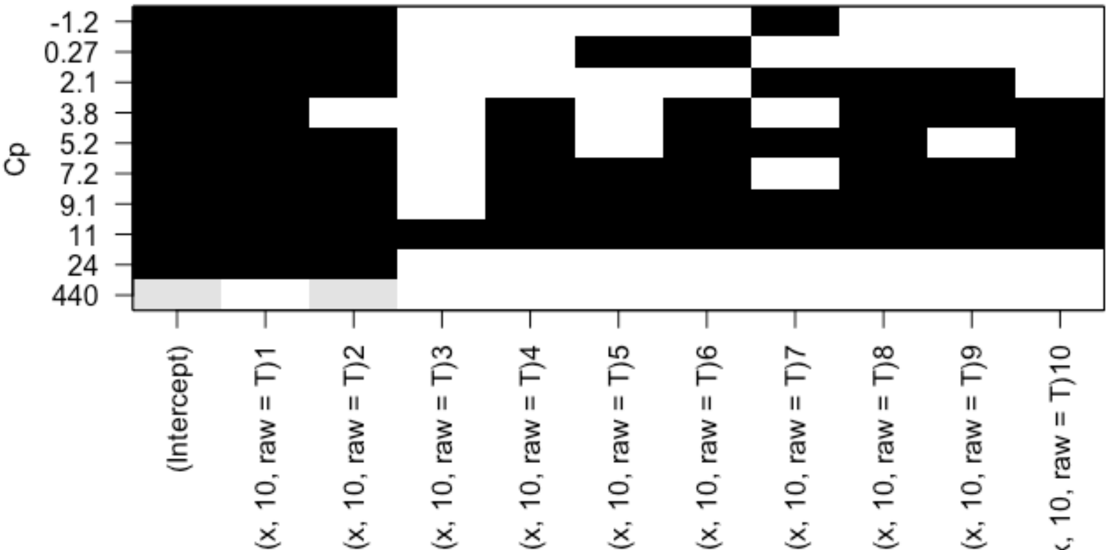
```
y <- 0.5 + 1.5*x + 2*x^2 + 0.3*x^3 + eps
```

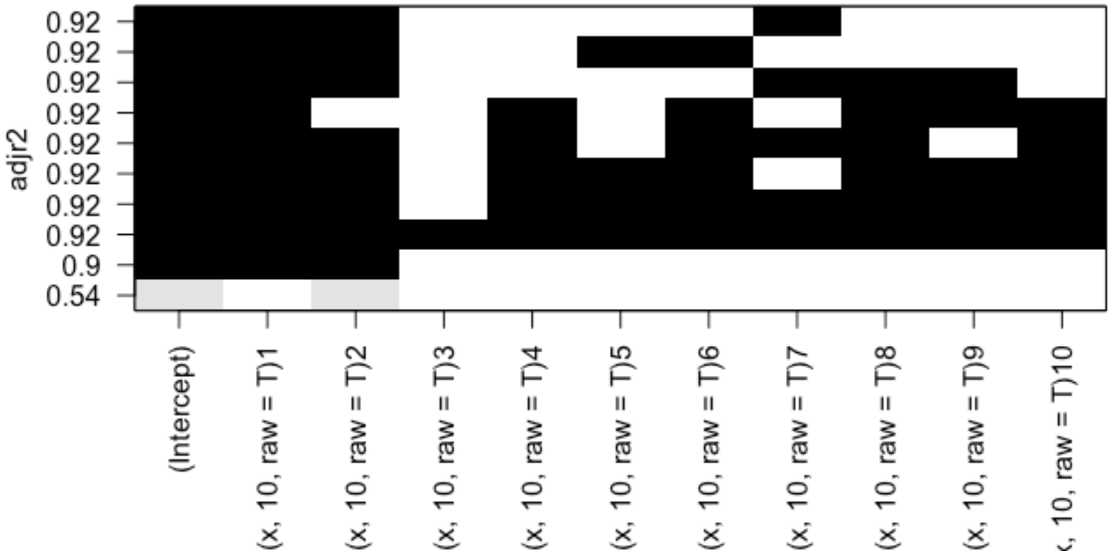
C.

```
> df <- data.frame(x,y)
> library(leaps)
> newfit.full <- regsubsets(y~poly(x,10,raw=T), data=df, nvmax=10)
> newfit.summary <- summary(newfit.full)
> which.min(newfit.summary$cp)
[1] 3
> which.min(newfit.summary$bic)
[1] 3
> which.max(newfit.summary$adjr2)
[1] 3
> plot(newfit.full, scale=“bic")
```

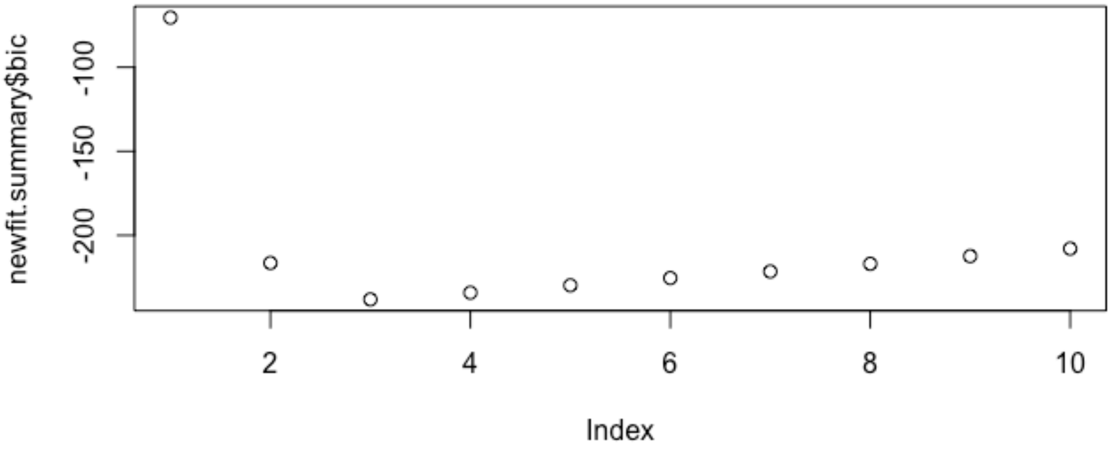


```
> plot(newfit.full, scale=“Cp")
```





```
> plot(newfit.full, scale="adjr2")
```



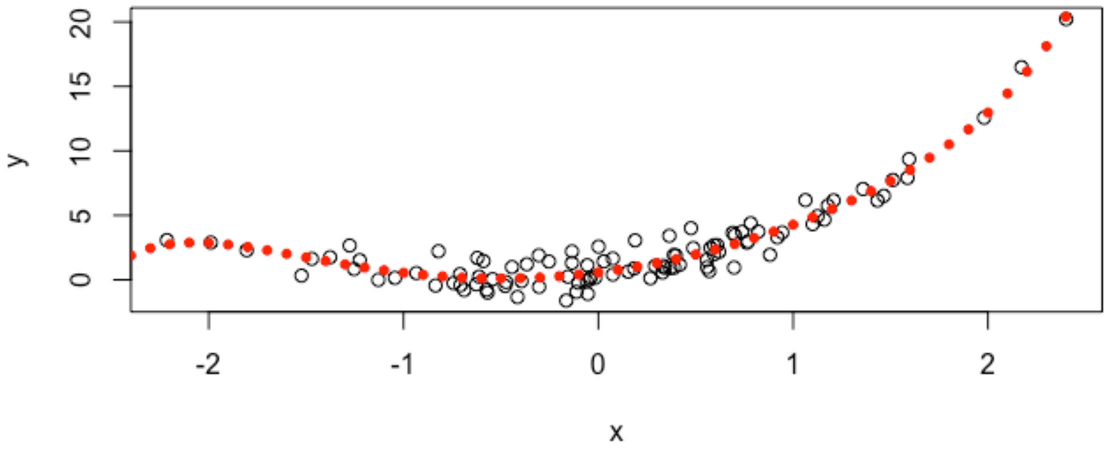
```
> plot(newfit.summary$bic)
```

Show the coefficients, which weirdly picked out x^7 instead of x^3:

```
> coefficients(newfit.full, id=3)
       (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2 poly(x, 10, raw = T)7
        0.57627412            1.85623596            1.83485113            0.01046843
```

Plot data vs model, not too bad:

```
> ypred <- c["(Intercept)"]+c["poly(x, 10, raw = T)1"]*xgrid +c["poly(x, 10, raw =
T)2"]*xgrid^2 + c["poly(x, 10, raw = T)7"]*xgrid^7
> plot(x,y)
> points(xgrid, ypred, col='red', pch=20)
```



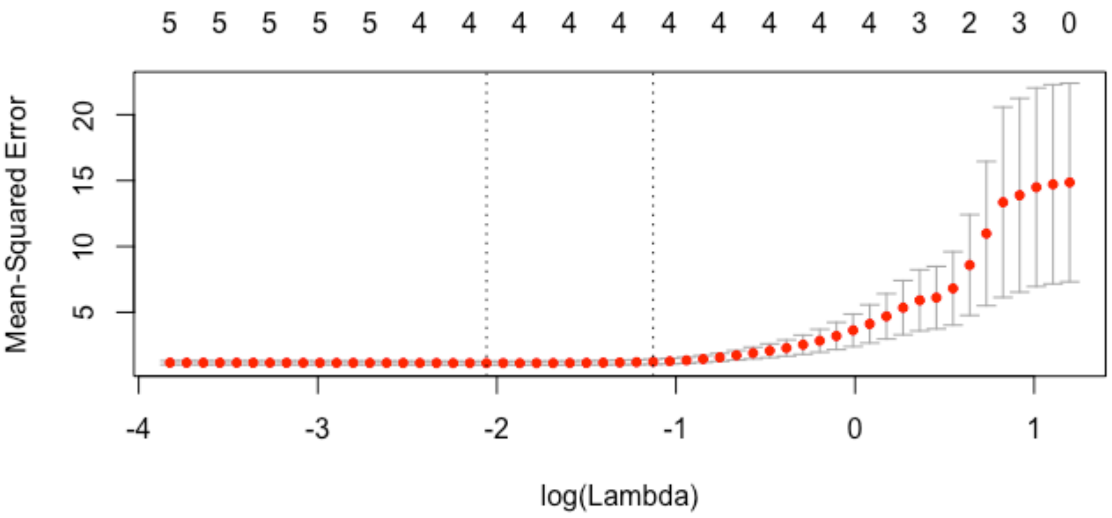
D. Repeat with forward-stepwise selection and backward-stepwise selection.

```
fit.fwd <- regsubsets(y~poly(x,10,raw=T), data=df, nvmax=10, method='forward')
summary(fit.fwd)
plot(fit.fwd)
plot(summary(fit.fwd)$bic)

fit.bwd <- regsubsets(y~poly(x,10,raw=T), data=df, nvmax=10, method='backward')
summary(fit.bwd)
plot(fit.bwd)
plot(summary(fit.bwd)$bic)
```

Results: forward steps picked the same model as best-subset. Backward steps was quite different, and picked a 6-element model involving powers 1, 4, 6, 8, 9, and 10! Model quality was pretty flat between orders 4-7. At order 4, kept powers 1, 4, 6, 9.

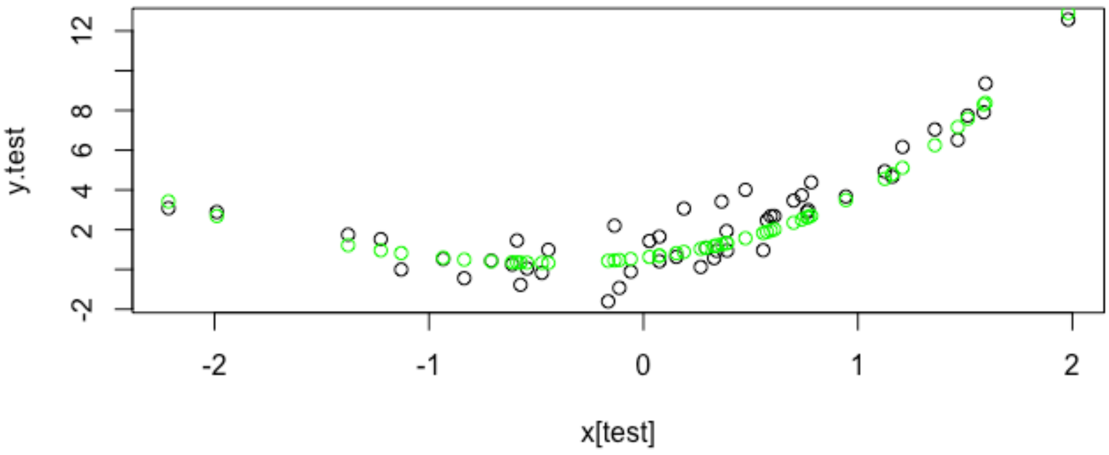
E. Repeat by lasso. Use this script:

```
A. > library(glmnet)
B. > grid <- 10^seq(10, -2, length=100)
C. > set.seed(1)
D. > train <- sample(1:100, 50)
E. > test<- (-train)
F. > y.test <- y[test]
G. > xmat <- model.matrix(y~poly(x,10,raw=T), df)[,-1]
H. > set.seed(1)
I. > lasso.mod <- glmnet(xmat[train,], y[train], alpha=1, lambda=grid)
J. > plot(lasso.mod)
K. > set.seed(1)
L. > cv.out <- cv.glmnet(xmat[train,], y[train], alpha=1)
```



```
M.  > plot(cv.out)
```

```
N.  > bestlam <- cv.out$lambda.min
```

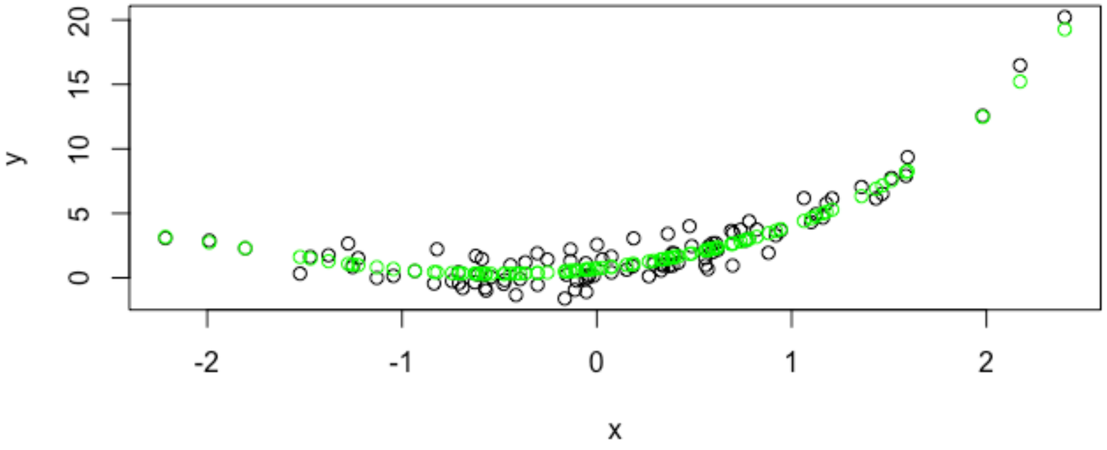


```
O.  > bestlam
P.  [1] 0.1277397
Q.  > lasso.pred <- predict(lasso.mod, s=bestlam, newx=xmat[test,])
R.  > mean((lasso.pred-y.test)^2)
S.  [1] 0.9071425
T.  > plot(x[test],y.test)
U.  > points(x[test], lasso.pred, col='green')
V.  > lasso.coef <- predict(lasso.mod, type='coefficients', s=bestlam)
W.  > lasso.coef
X.  11 x 1 sparse Matrix of class "dgCMatrix"
Y.  1
Z.  (Intercept)            0.59169178
AA.poly(x, 10, raw = T)1  1.22371027
BB.poly(x, 10, raw = T)2  1.56970360
CC.poly(x, 10, raw = T)3  0.34735834
DD.poly(x, 10, raw = T)4  0.06715902
EE. poly(x, 10, raw = T)5  .
FF. poly(x, 10, raw = T)6  .
GG.poly(x, 10, raw = T)7  .
```

```
HH.poly(x, 10, raw = T)8  .
II.  poly(x, 10, raw = T)9  .
JJ. poly(x, 10, raw = T)10 .
```

Now go back and refit to the whole data set using the best lambda.

```
KK.> lasso.final <- glmnet(xmat, y, alpha=1, lambda=grid)
LL. > lasso.coef.final <- predict(lasso.final, type='coefficients', s=bestlam)
MM.> lasso.coef.final
NN.11 x 1 sparse Matrix of class "dgCMatrix"
OO.                          1
PP. (Intercept)          0.718940202
QQ.poly(x, 10, raw = T)1  1.593114718
RR.poly(x, 10, raw = T)2  1.584758068
SS.poly(x, 10, raw = T)3  0.100760114
TT. poly(x, 10, raw = T)4  0.043250847
UU.poly(x, 10, raw = T)5  0.023174835
```



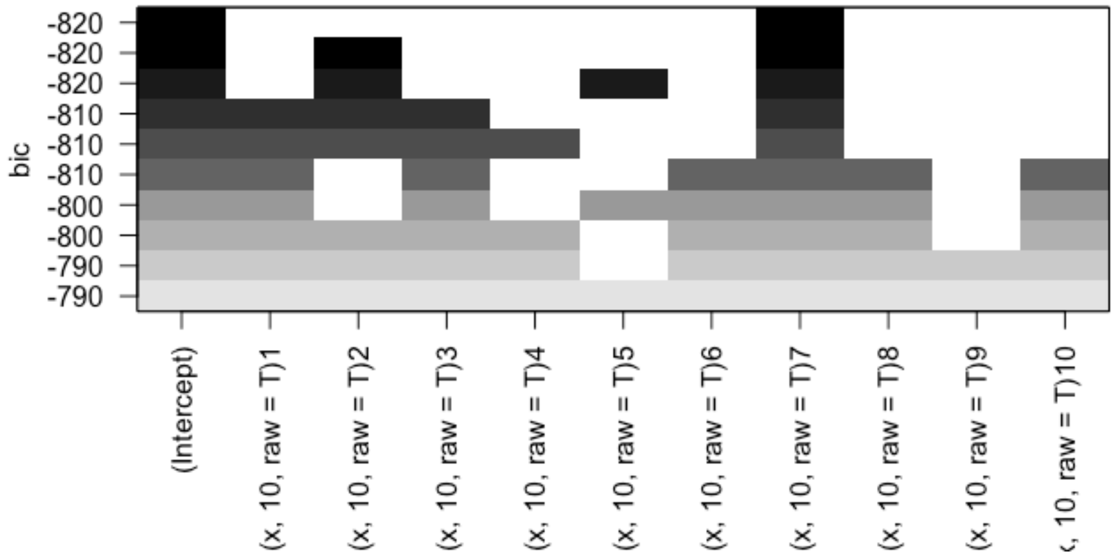
```
VV. poly(x, 10, raw = T)6  .
WW.poly(x, 10, raw = T)7  0.001942179
XX. poly(x, 10, raw = T)8  .
YY. poly(x, 10, raw = T)9  .
ZZ. poly(x, 10, raw = T)10 .
AAA.> lasso.pred.final <- predict(lasso.final, s=bestlam, newx = xmat)
BBB.> plot(x,y)
CCC.> points(x, lasso.pred.final, col='green')
```

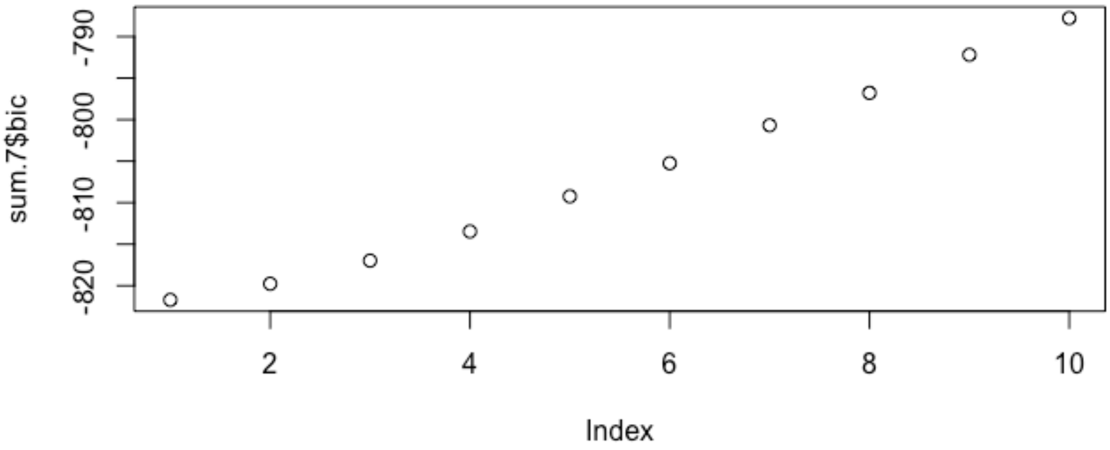
Curiously, x^5 and x^7 got included when I repeated on the full data set. MSE is pretty good though, and better than I got in CV:

```
DDD.> mean((lasso.pred.final-y)^2)
EEE.[1] 0.8860559
```

F. Now make a dataset with y = b0 + b7 * x^7 + eps, and do best-subset and lasso.

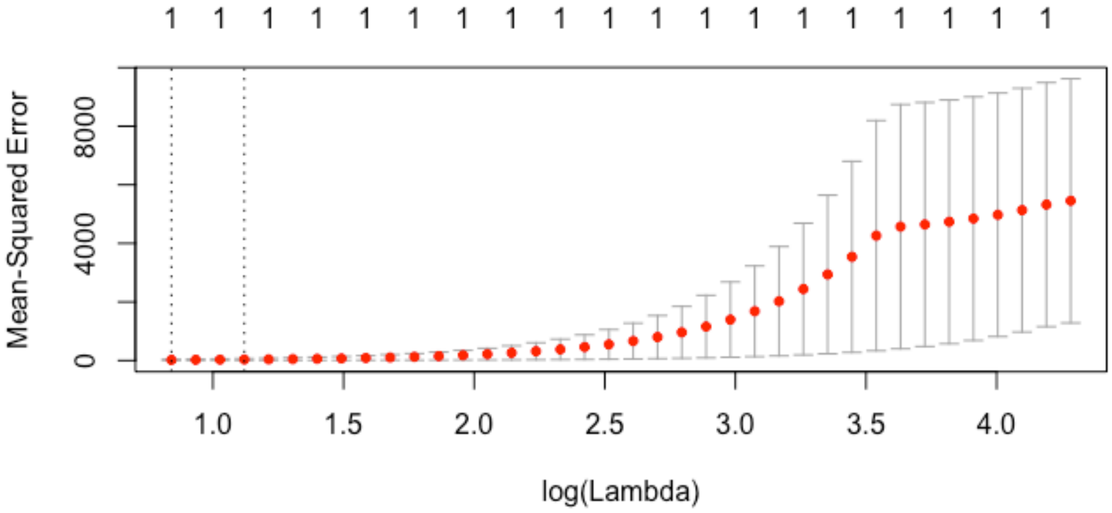
```
A.  > y7 <- -1 + x^7 + eps
B.  > plot(x,y7)
C.  > df7 <- data.frame(x,y7)
D.  > fit.7 <- regsubsets(y7~poly(x,10,raw=T), data=df7, nvmax=10)
E.  > sum.7 <- summary(fit.7)
F.  > which.min(sum.7$bic)
G.  [1] 1
```





```
H.  > plot(fit.7)
I.  > plot(sum.7$bic)
J.  > coef(fit.7, id=1)
K.          (Intercept) poly(x, 10, raw = T)7
L.            -1.04106              1.00077
```

Here the functional form is so strong and simple that the best fit was much better than the next best, and the coefficients were picked very accurately.



Next, lasso:

```
M.  > xmat7 <- model.matrix(y~poly(x,10,raw=T), df7)[,-1]
N.  > set.seed(1)
O.  > cv.out <- cv.glmnet(xmat7[train,], y7[train], alpha=1)
P.  > plot(cv.out)

Q.  > bestlam <- cv.out$lambda.min
R.  > bestlam
S.  [1] 2.318941
T.  > lasso.7 <- glmnet(xmat7, y7, alpha=1, lambda=grid)
U.  > predict(lasso.7, type='coefficients', s=bestlam)
V.  11 x 1 sparse Matrix of class "dgCMatrix"
W.                              1
X.  (Intercept)           -0.8795859
Y.  poly(x, 10, raw = T)1   .
Z.  poly(x, 10, raw = T)2   .
AA. poly(x, 10, raw = T)3   .
BB. poly(x, 10, raw = T)4   .
CC. poly(x, 10, raw = T)5   .
DD. poly(x, 10, raw = T)6   .
EE. poly(x, 10, raw = T)7   0.9625098
FF. poly(x, 10, raw = T)8   .
GG. poly(x, 10, raw = T)9   .
HH. poly(x, 10, raw = T)10  .
```

This method also gets the right functional form and is close to the right coefficients, but not as close. I guess it may save a little on the coefficients budget at the expense of a smaller amount of error?

6.8.9: Using the College data set. I used the following script to do the indicated sections.

```
# a
library(ISLR)
library(glmnet)
set.seed(1)
train <- sample(1:nrow(College), nrow(College)/2)
test <- -train

# b
lm.fit <- lm(Apps ~ ., data=College, subset=train)
print(summary(lm.fit))
lm.pred <- predict(lm.fit, College[test,])
print("MSE with linear regression fit is: ")
print(mean((lm.pred-College[test,"Apps"])^2))

#c
set.seed(2)
xmat <- model.matrix(Apps ~ ., College)[,-1]
y <- College$Apps
grid <- 10^seq(10,-2,length=100)
ridge.fit <- glmnet(xmat[train, ], y[train], alpha=0, lambda=grid)
cv.out <- cv.glmnet(xmat[train, ], y[train], alpha=0)
plot(cv.out)
```

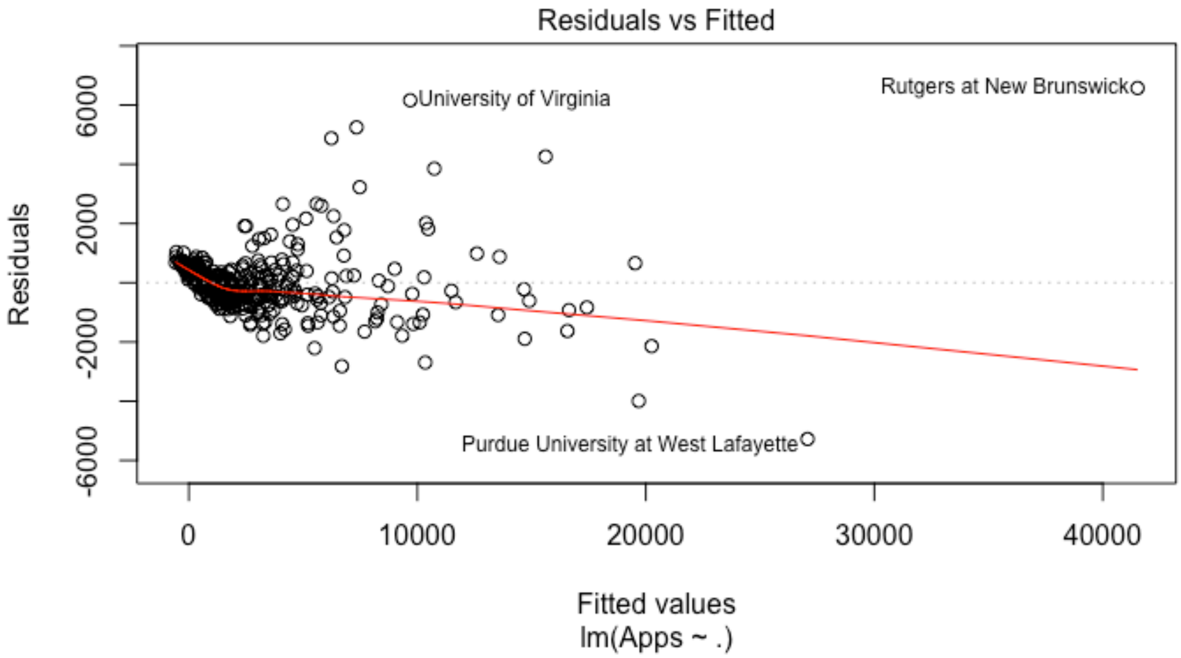
```
bestlam <- cv.out$lambda.min
ridge.pred <- predict(ridge.fit, s=bestlam, newx=xmat[test, ])
print("Best lambda for ridge regression, determined by kFCV, is:")
print(bestlam)
print("MSE with ridge regression fit selected by kFCV is:")
print(mean((ridge.pred-y[test])^2))

#d
set.seed(3)
# xmat <- model.matrix(Apps ~ ., College)[,-1]
# y <- College$Apps
# grid <- 10^seq(10,-2,length=100)
lasso.fit <- glmnet(xmat[train, ], y[train], alpha=1, lambda=grid)
cv.out <- cv.glmnet(xmat[train, ], y[train], alpha=1)
plot(cv.out)
bestlam <- cv.out$lambda.min
lasso.pred <- predict(lasso.fit, s=bestlam, newx=xmat[test, ])
print("Best lambda for lasso model, determined by kFCV, is:")
print(bestlam)
print("MSE with lasso model fit selected by kFCV is:")
print(mean((lasso.pred-y[test])^2))
print("Number of nonzero coefficients in the model, excluding intercept: ")
lasso.coef <- predict(lasso.fit, type='coefficient', s=bestlam)
print(sum(lasso.coef!=0)-1)

# e
set.seed(4)
library(pls)
pcr.fit <- pcr(Apps ~ ., data=College, subset=train, scale=TRUE, validation="CV")
print(summary(pcr.fit))
validationplot(pcr.fit, val.type="MSEP")
# By inspection, minimum CV error is with M=16.
# I don't know how to pick it out programmatically.
pcr.pred <- predict(pcr.fit, College[test, ], ncomp=16)
print("For PCR, with M=16, test MSE is: ")
print(mean((pcr.pred-y[test])^2))

# f
set.seed(5)
# library(pls)
pls.fit <- plsr(Apps ~ ., data=College, subset=train, scale=TRUE, validation="CV")
print(summary(pls.fit))
validationplot(pls.fit, val.type="MSEP")
# By inspection, minimum CV error is with M=9, but barely different from 6 to 17.
# And several tied values at M=9-12, 14
# I don't know how to pick it out programmatically.
pls.pred <- predict(pls.fit, College[test, ], ncomp=9)
print("For PLS, with M=9, test MSE is: ")
print(mean((pls.pred-y[test])^2))

# g: compare results
```

Test MSE values came out as follows:

- Basic linear regression: 1108531

- Ridge regression: 1036914
- Lasso: 1032128
- PCR: 1166897
- PLS: 1134643

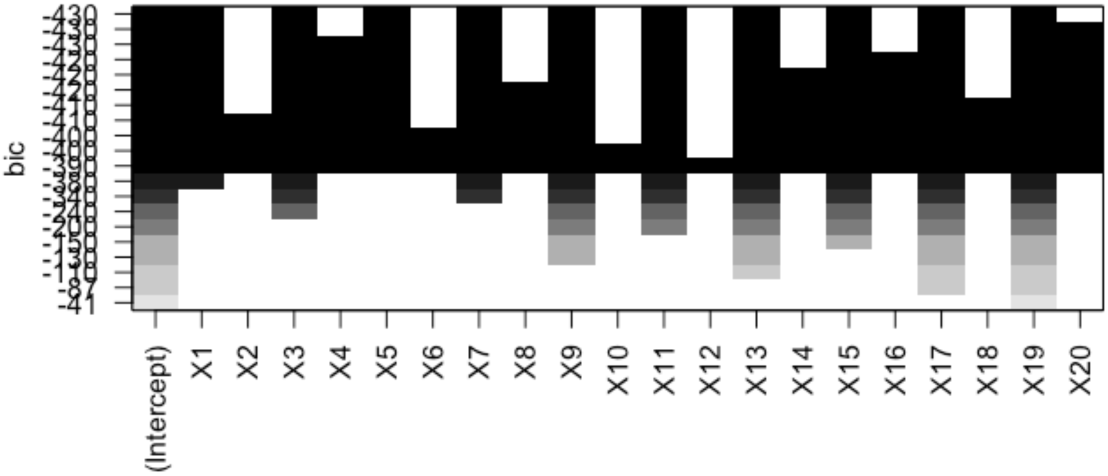
In this case, PCR and PLS did not even do as well as basic linear regression. Lasso did better, as did ridge just behind, but the lasso only dropped two predictors. PCR also found that not much dimension reduction was possible. The best MSE error translates to RMS test error of about 1016 applications. The summary statistics in the applications data show Mean=3002, Median=1558, Min=81, Max=48090. So this level of predictive power is not very impressive for the typical college on the list. The linear regression fit residuals plot shows that most colleges are medium sized and not predicted terribly well: residuals of ~1000 up to fitted=5000, then higher yet.



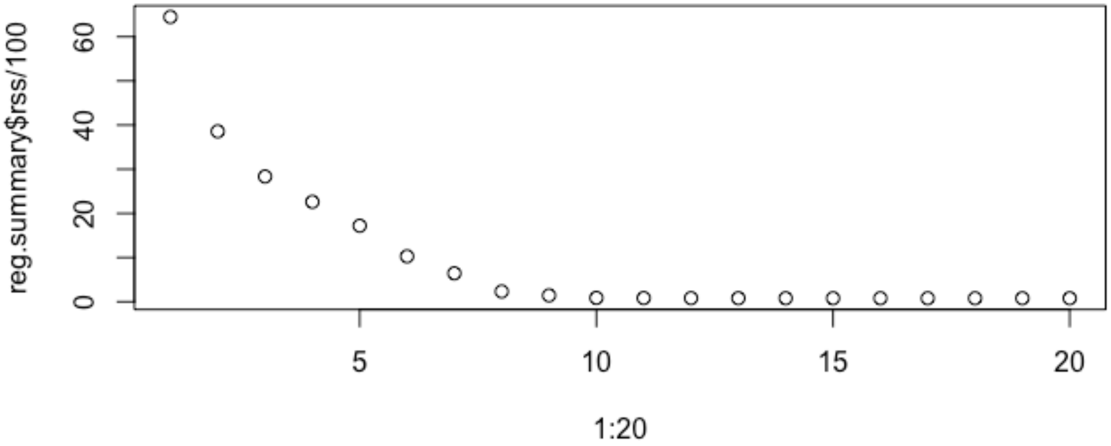
6.8.10: explore best subset with simulated data, in which the real functional form includes exact zeros.

```
# a: Generate a data set with p=20 features and n=1000 observations, and one
# response vector Y=X * B + eps, with some B_i exactly zero.

set.seed(1)
x <- matrix(data=rnorm(20*1000), nrow=1000, ncol=20)
eps <- rnorm(1000)
b <-c(1,0,2,0,-1,0,-2,0,3,0,-3,0,4,0,-4,0,5,0,-5,0)
y <- rep(0,1000)
# y <- x%*%b + eps
for(i in 1:1000){
   y[i] <- b %*% x[i,] + eps[i]
}
df <-data.frame(x,y)
```

```
# b: split into training set of 100 and test set of 900

set.seed(2)
train <- sample(1:1000, 100)
test <- -train
df.train <- df[train,]
y.train <-y[train]
df.test <- df[test,]
y.test <- y[test]

# c: best subset selection on the training set, plot best training set MSE for each size
# By definition, training set MSE is RSS/n_obs.
```

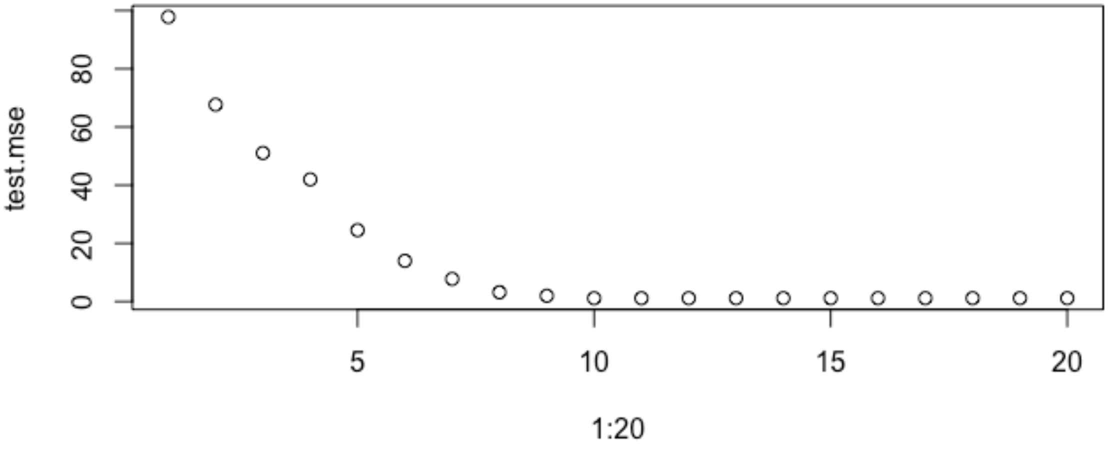


```
library(leaps)

regfit.full <- regsubsets(y~., data=df[train,], nvmax=20)
```



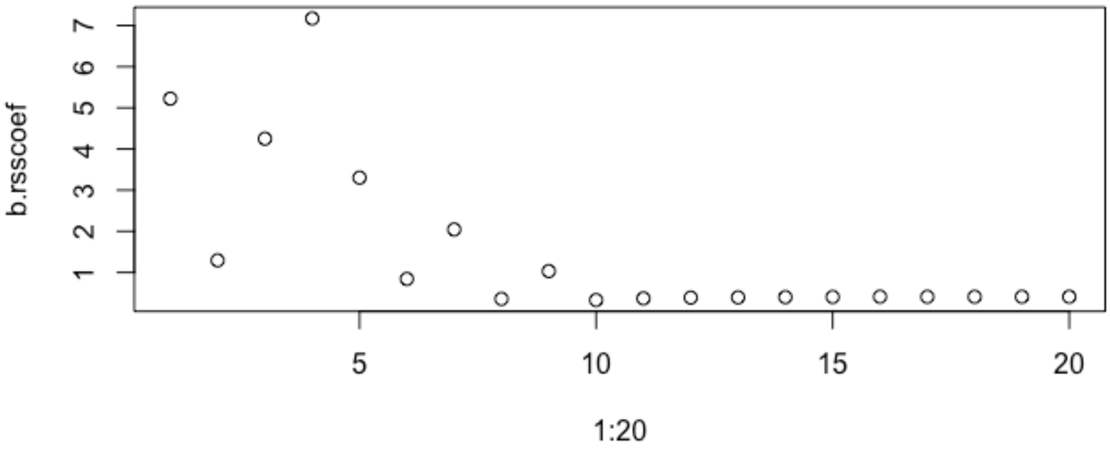
```
reg.summary <- summary(regfit.full)
```

```
print(reg.summary)
plot(regfit.full)



plot(1:20, reg.summary$rss/100)




# d: plot MSE of best models on test data. Need our helper function for this.

predict.regsubsets <- function(object, newdata, id, ...){
    form=as.formula(object$call[[2]])
    mat=model.matrix(form, newdata)
    coefi=coef(object, id=id)
    xvars=names(coefi)
    mat[,xvars]%*%coefi
}
```



```
test.mse <- rep(0,20)
for(i in 1:20){
    test.pred <- predict(regfit.full, newdata=df[test,], id=i)
    test.mse[i] <- mean((test.pred - y.test)^2)
}

plot(1:20, test.mse)

# e: which model size gives least test MSE?

bestid <- which.min(test.mse)
print(bestid)

> 10

# f: what about the coefficient values? how do they compare to the actual b_i?
```

```
coef.best <- coef(regfit.full, id=bestid)
print(paste0("Fitted coefficients in best model, with model size "), bestid)
print(coef.best)

[1] "Fitted coefficients in best model, with model size "
(Intercept)          X1          X3          X5          X7          X9         X11 
 0.07991724  0.95051706  2.06671158 -0.93393704 -2.04843404  2.95042405 -2.95194772 
        X13         X15         X17         X19 
 4.08011232 -4.21132993  4.80246269 -5.01069701 

print("Exact coefficients were: ")
print(b)

[1] "Exact coefficients were: "
 [1]  1  0  2  0 -1  0 -2  0  3  0 -3  0  4  0 -4  0  5  0 -5  0

# g: plot "RMS coefficient error" vs model size
# first name the rows in the b vector same as the df columns

b.names <- rep(NULL, 20)
for(i in 1:20){
    b.names[i] <- paste0("X",i)
}
names(b) <- b.names

b.rsscoef <- rep(0,20)
for(i in 1:20){
    coefi <- coef(regfit.full, id=i)[-1]     # ignore intercept
    b.rsscoef[i] <- sqrt(
        sum((b[b.names %in% names(coefi)] - coefi)^2) +
        sum(b[!(b.names %in% names(coefi))])^2
        )
}
```



```
plot(1:20, b.rsscoef)
```

Minimum value is found at index 10.

6.8.11: Boston data set again. Predict per-capita crime rate using other factors, and throw the book at it, using best subset, lasso/ridge, or whatever. Evaluate using some validation method. Do all features get used?

Summary of results: lowest MSE came from partial least squares at M=9, nearly matched by lasso at 12 predictors. Best-subset came to the same 12 predictors, nearly as good, generally similar values. It's clear that not all of the predictors are used in the best models. I suspect this happens because some of the predictors are correlated with each other. I used pairs(Boston) and cor(Boston) to confirm this: 7 distinct pairs of predictors have above 0.7 correlation, and tax/rad have a correlation of 0.91! So dimension reduction can probably be effective.

I used the following script:

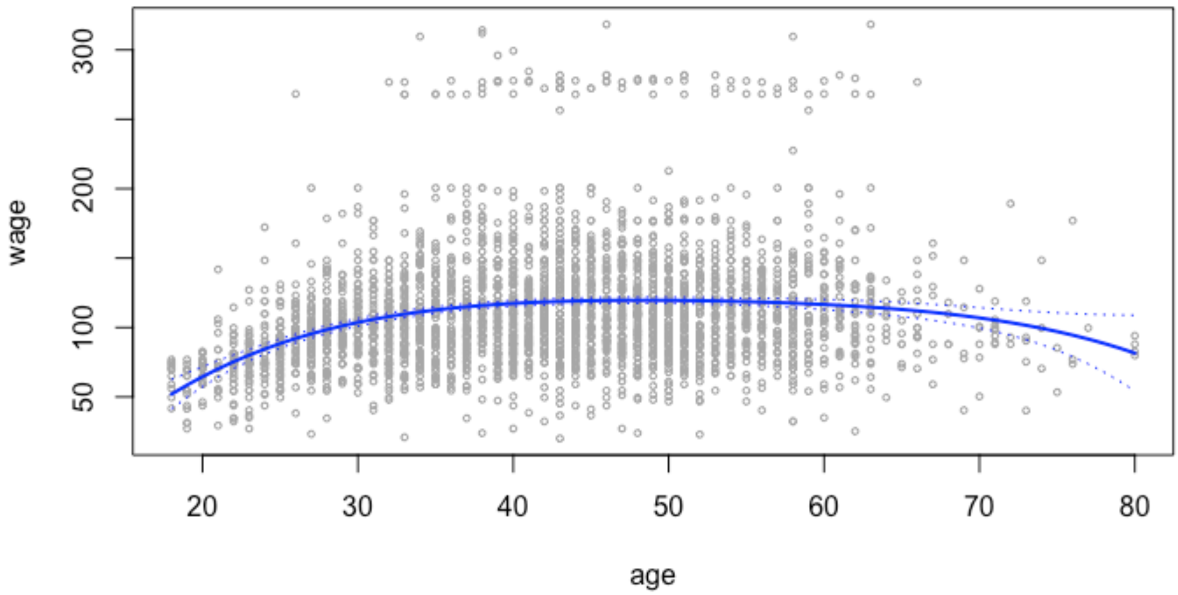
```
# Boston checklist: best subset, lasso, PLS. Pick by kFCV

# load in data and partition into 10 folds

library(MASS)
library(leaps)
library(glmnet)
library(pls)

k = 10 # number of folds
p = ncol(Boston) - 1 # number of predictors
set.seed(1)
folds <- sample(1:k, nrow(Boston), replace=TRUE)


# set up helper predict function to do kFCV on best-subset method

predict.regsubsets <- function(object, newdata, id, ...){
   form <- as.formula(object$call[[2]])
   mat <- model.matrix(form, newdata)
   coefi <- coef(object, id=id)
   xvars <- names(coefi)
   mat[,xvars]%*%coefi
}

# Now do the CV best-subset
print("Next: best-subset with cross-validation")
cv.errors <- matrix(NA, k, p, dimnames=list(NULL, paste(1:p)))

for(j in 1:k){
   best.fit <- regsubsets(crim ~ ., data=Boston[folds != j,], nvmax=p)
   for(i in 1:p){
      pred <- predict(best.fit, Boston[folds == j,], id=i)
      cv.errors[j,i] <- mean((Boston$crim[folds==j] - pred)^2)
   }
}

mean.cv.errors <- apply(cv.errors, 2, mean)
best_subset <- which.min(mean.cv.errors)
print("Mean errors determined by 10-fold cross-validation are:")
print(mean.cv.errors)
print(paste("Error is minimized with", best_subset, "predictors, having MSE value",
        mean.cv.errors[best_subset]))
reg.best <- regsubsets(crim~., data=Boston, nvmax=p)
coef.best <- coef(reg.best, best_subset)
print("Coefficients for best subset model are:")
print(coef.best)

# now do the kFCV lasso
print("Next: 10-fold cross-validation for lasso")
set.seed(2)
xmat <- model.matrix(crim ~ ., Boston)[,-1]
y <- Boston$crim
grid <- 10^seq(10,-2,length=100)
lasso.fit <- glmnet(xmat, y, alpha=1, lambda=grid)
cv.out <- cv.glmnet(xmat, y, alpha=1)
plot(cv.out)
bestlam <- cv.out$lambda.min
lasso.coef <- predict(lasso.fit, type='coefficient', s=bestlam)
print("Best lambda for lasso model, determined by kFCV, is:")
print(bestlam)
print("Number of nonzero coefficients in the model, excluding intercept: ")
print(sum(lasso.coef!=0)-1)
print("Coefficients in this model are:")
print(lasso.coef)
lasso.pred <- predict(lasso.fit, s=bestlam, newx=xmat)
lasso.mse <- mean((lasso.pred - y)^2)
print(paste("Best lasso MSE value is", lasso.mse))

# now do partial least squares
print("Next: 10-fold cross-validation for partial least squares")
set.seed(3)
pls.fit <- plsr(crim ~ ., data=Boston, scale=TRUE, validation="CV")
print(summary(pls.fit))
validationplot(pls.fit, val.type="MSEP")
# By inspection, minimum CV error is with M=9, but barely different from 9 to 13.
# I don't know how to pick it out programmatically.

pls.pred <- predict(pls.fit, Boston, ncomp=9)
print("For PLS, with M=9, test MSE is: ")
print(mean((pls.pred-y)^2))
```

7.9.1: trivial

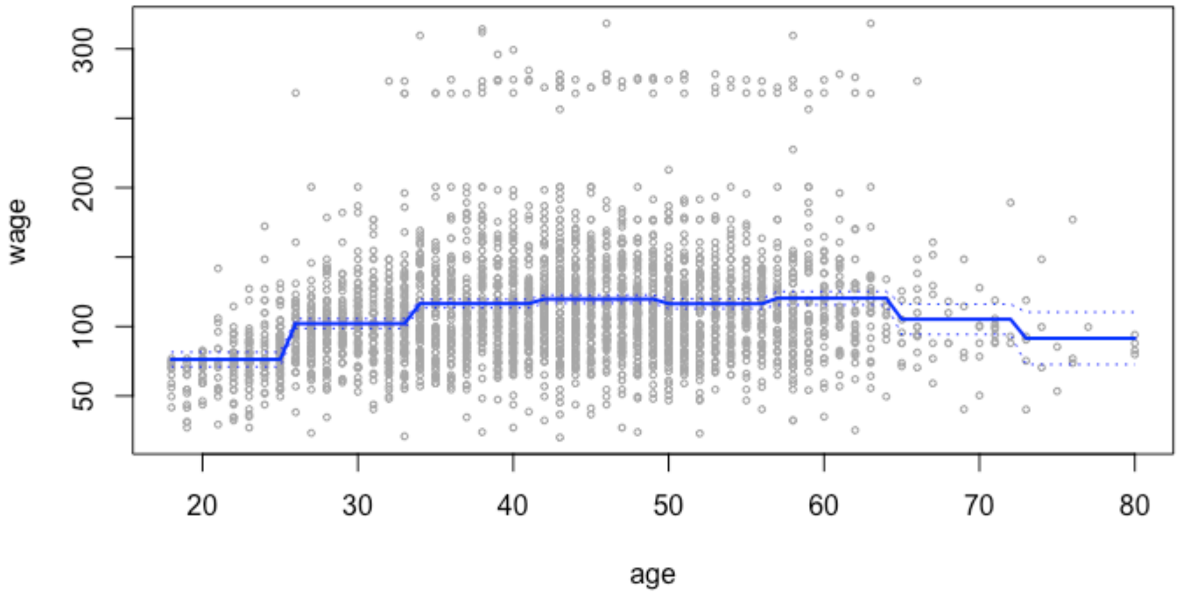
7.9.2: pencil and paper. However, sketches are easy to describe.

A) Minimize $\int [g(x)]^2 dx$ so it's just a flat g(x) = 0.
B) Minimize first derivative, so it's a flat line at g(x) = mean(y_i).
C) Line segments connecting the sample points
D) Parabolic arcs connecting the sample points
E) At lambda=0, this is just linear regression.

7.9.3: pencil and paper. Parabola fragment hanging off a line.



Exercise 7.9.6a: polynomial



Exercise 7.9.6b: step function

7.9.4 pencil and paper. Piecewise linear mess.

7.9.5: compare smoothing splines which have smoothness penalty in 3rd or 4th derivative.

a) as lambda -> infinity, g_2 will have the smaller training RSS because it is a more flexible function.
b) As lambda -> infinity, either function could have lower test RSS, depending on the way the data break.

c) For lambda=0, the functions are identical.

7.9.6: continue with Wage data.

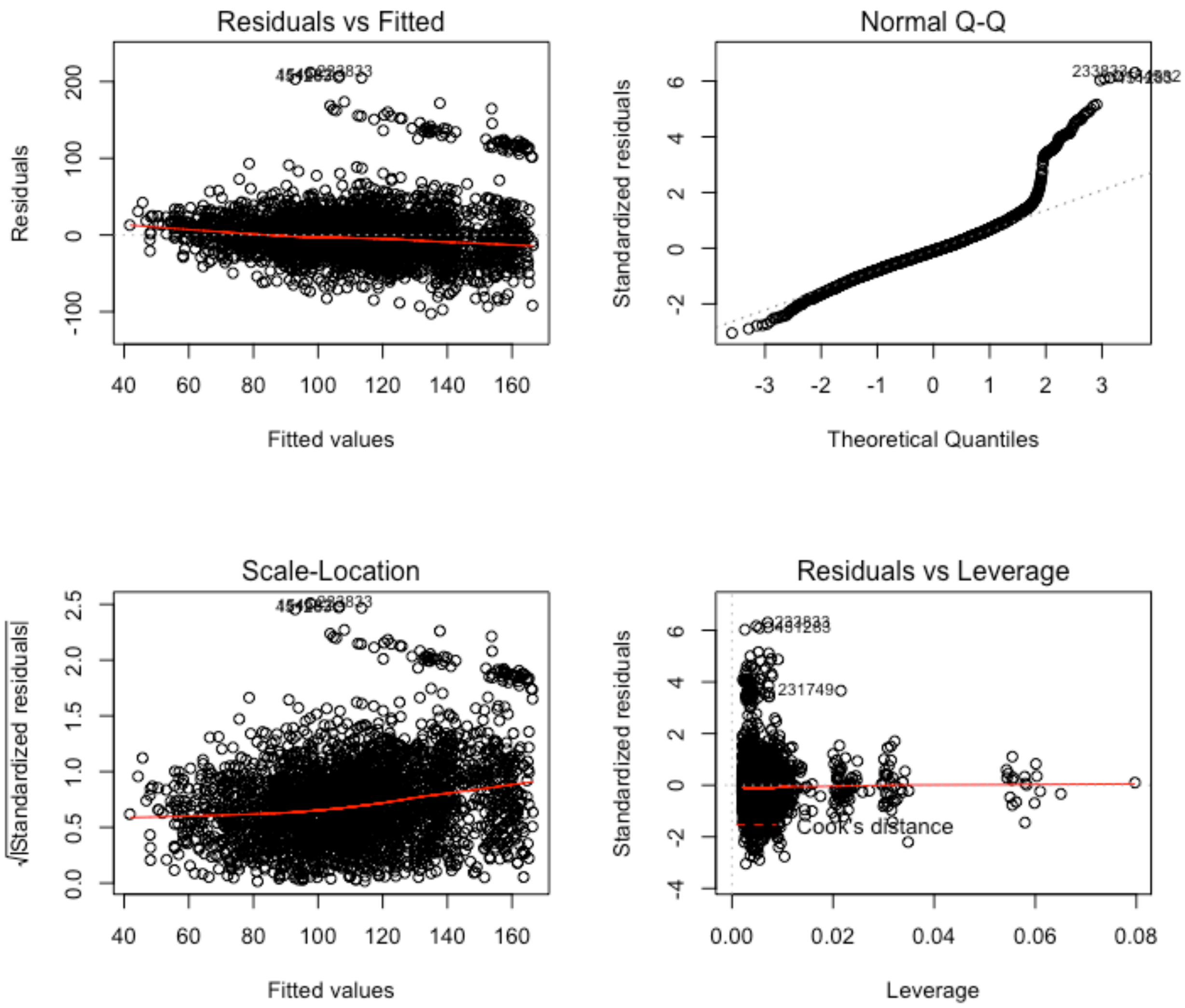
a) predict wage using polynomial of age, selecting degree by cross-validation. How does it compare to the results of ANOVA? Plot the resulting fit.

b) predict wage using a step function of age, and select number of cuts by CV. Plot.

Result: best polynomial is order 4 (though 3 is quite close), best number of cuts is 8.

```
library(ISLR)
library(boot)
attach(Wage)

# a: use 10-fold CV in a for loop to determine order of the best poly fit
set.seed(1)
cv.error <- rep(0,10)
for (i in 1:10){
   glm.fit <- glm(wage~poly(age,i), data=Wage)
   cv.error[i] <- cv.glm(Wage, glm.fit, K=10)$delta[1]
}
print('CV error by polynomial order is:')
print(cv.error)
best.i <- which.min(cv.error)
plot(1:10, cv.error, xlab="Fit order", ylab='CV error', type='l')
print(paste("Best order of polynomial is:", best.i))
glm.best <- glm(wage~poly(age,best.i), data=Wage)

agelims <- range(age)
age.grid <- seq(from=agelims[1], to=agelims[2])

preds <- predict(glm.best, newdata=list(age=age.grid), se=TRUE)
se.bands <- cbind(preds$fit + 2*preds$se.fit, preds$fit - 2*preds$se.fit)

plot(age, wage, xlim=agelims, cex=0.5, col='darkgrey')
title('Exercise 7.9.6a: polynomial’)
lines(age.grid, preds$fit, lwd=2, col='blue')
matlines(age.grid, se.bands, lwd=1, col='blue', lty=3)

# b: use 10-fold CV in a for loop to determine number of cuts in best step function fit
set.seed(2)
cv.error <- rep(NA, 10)
for (i in 2:10){
   Wage$age.cut <- cut(Wage$age, i)
   glm.fit <- glm(wage~age.cut, data=Wage)
   cv.error[i] <- cv.glm(Wage, glm.fit, K=10)$delta[1]
}
print('CV error by number of cuts is:')
print(cv.error)
best.i <- which.min(cv.error)
plot(2:10, cv.error[-1], xlab="#cuts", ylab='CV error', type='l')

print(paste("Best number of cuts in step function is:", best.i))
glm.best <- glm(wage~cut(age,best.i), data=Wage)
```

```
agelims <- range(age)
age.grid <- seq(from=agelims[1], to=agelims[2])

preds <- predict(glm.best, newdata=list(age=age.grid), se=TRUE)
```



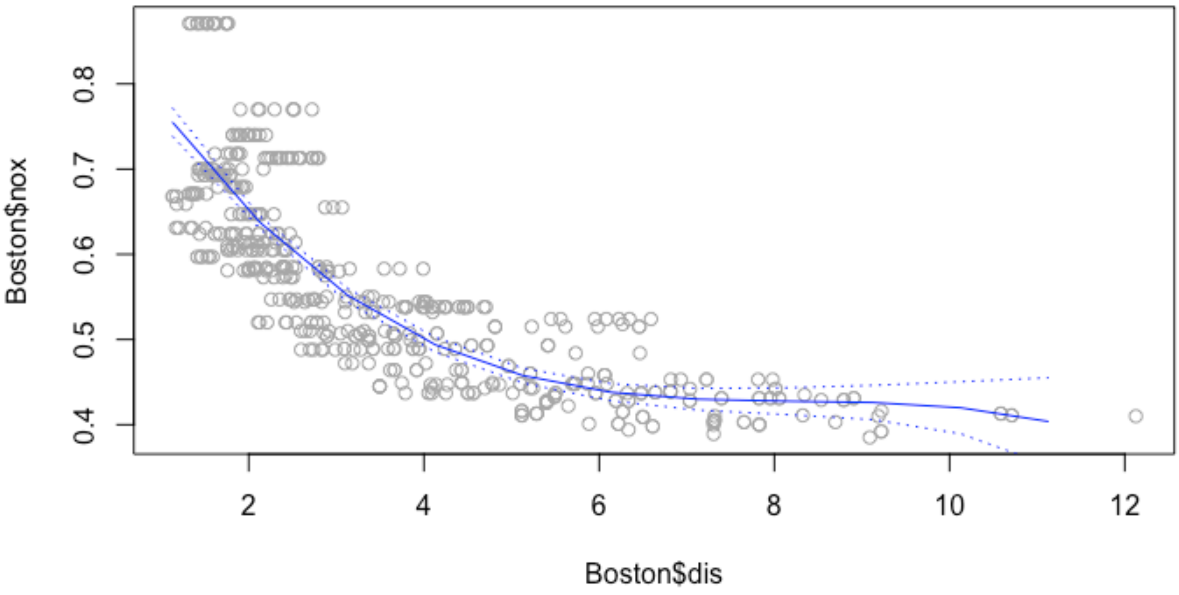
```
se.bands <- cbind(preds$fit + 2*preds$se.fit, preds$fit - 2*preds$se.fit)

plot(age, wage, xlim=agelims, cex=0.5, col='darkgrey')
title('Exercise 7.9.6b: step function')
lines(age.grid, preds$fit, lwd=2, col='blue')
matlines(age.grid, se.bands, lwd=1, col='blue', lty=3)
```

7.9.7: check for dependence of wage on other features such as marital status, job class, etc. Note that they are all factors, so it’s not clear to me what “nonlinear” would be in this case.

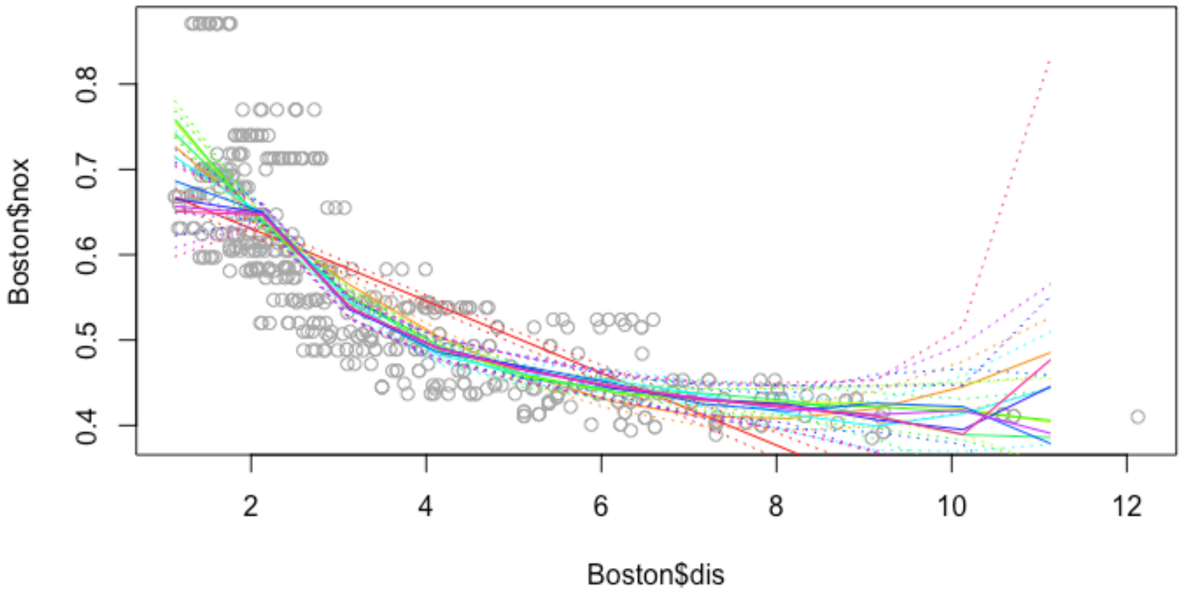
All factors turn out to add statistical significance except for region (vacuous) and age spline order 3 and higher. I left out logwage which is a duplicate of wage, effectively. Fit diagnostics look decent.

```
library(splines)
lm.fit <- lm(wage~year+ns(age,2)+maritl+race+education+jobclass+health+health_ins,
           data=Wage)
par(mfrow=c(2,2))
plot(lm.fit)
print(summary(lm.fit))
```

```
Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                 -2504.6186   612.1607  -4.091 4.40e-05 ***
year                           1.2788     0.3052   4.190 2.87e-05 ***
ns(age, 2)1                   41.3540     5.3988   7.660 2.50e-14 ***
ns(age, 2)2                   -9.9591     4.6809  -2.128  0.03345 *
maritl2. Married              13.5687     1.7889   7.585 4.41e-14 ***
maritl3. Widowed               0.8204     7.9480   0.103  0.91779
maritl4. Divorced              0.2535     2.9174   0.087  0.93077
maritl5. Separated             7.4117     4.8464   1.529  0.12629
race2. Black                  -4.6888     2.1306  -2.201  0.02783 *
race3. Asian                  -2.7078     2.5840  -1.048  0.29476
race4. Other                  -5.7272     5.6247  -1.018  0.30866
education2. HS Grad            7.5619     2.3520   3.215  0.00132 **
education3. Some College      18.1023     2.5017   7.236 5.84e-13 ***
education4. College Grad      30.6186     2.5310  12.097  < 2e-16 ***
```



```
education5. Advanced Degree   53.2087     2.7926  19.053  < 2e-16 ***
jobclass2. Information         3.5151     1.3143   2.675  0.00752 **
health2. >=Very Good           6.2447     1.4109   4.426 9.94e-06 ***
health_ins2. No              -16.4529     1.4018 -11.737  < 2e-16 ***
---
```



```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 33.75 on 2982 degrees of freedom
Multiple R-squared:  0.3496,  Adjusted R-squared:  0.3459
F-statistic: 94.29 on 17 and 2982 DF,  p-value: < 2.2e-16
```

7.9.8: do nonlinear fits to Auto data set. Did CV on mgp~poly(acceleration,i), found best.i=4. For mpg ~ poly(weight,i) found best.i=3 and it looked quite like a nice fit.

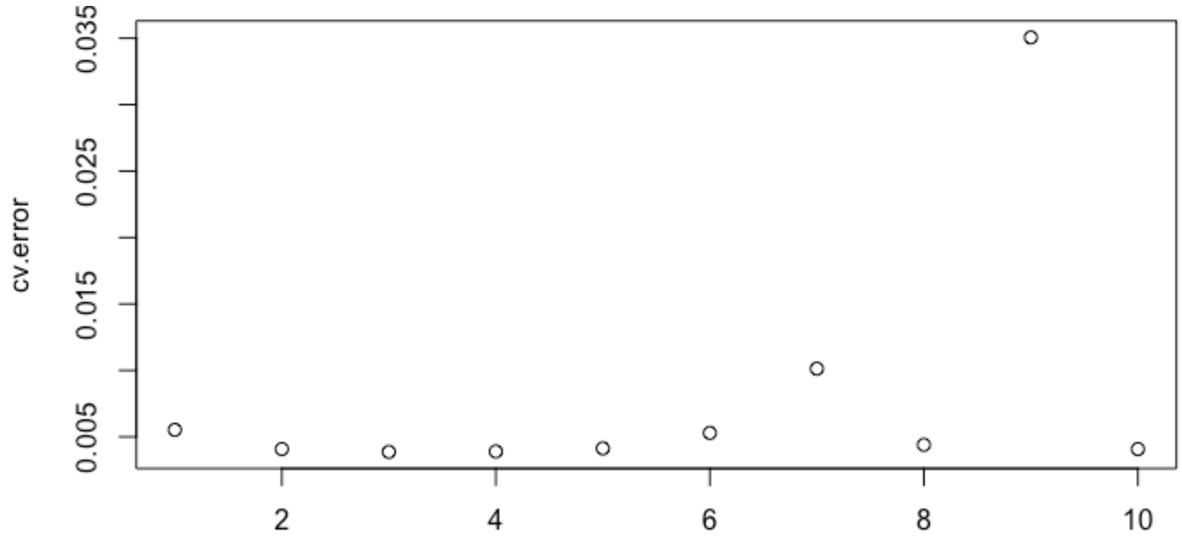
7.9.9: Using Boston data set, dis as predictor and nox as response.

A) Polynomial fit: High significance to order 3.
B) Do a bunch of fits up to order 10 and plot.

```
A)  library(MASS)
B)  dislims <- range(Boston$dis)
C)  dis.grid <- seq(from=dislims[1], to=dislims[2])
D)  all.rss <- rep(0,10)
E)  plot(Boston$dis, Boston$nox, col='darkgrey')
F)  collist <- rainbow(10)
G)  for(i in 1:10){
H)      lm.fit <- lm(nox~poly(dis, i), data=Boston)
I)      #print(summary(lm.fit))
J)      pred <- predict(lm.fit, newdata=list(dis=dis.grid), se=T)
K)      se.bands <-cbind(pred$fit + 2*pred$se.fit, pred$fit - 2*pred$se.fit)
L)      lines(dis.grid, pred$fit, col=collist[i])
M)      matlines(dis.grid, se.bands, col=collist[i], lty=3)
N)      print(paste("At order", i, "RSS is",
O)                  sum((lm.fit$residuals)^2)))
P)  }
```

C) Do cross-validation and select the best order. Result: order 3 is the best but 2-5 are all plausible candidates.

```
A)  #7.9.9 c
B)  library(MASS)
C)  library(boot)
D)  set.seed(100)
E)  dislims <- range(Boston$dis)
F)  cv.error <- rep(0,10)
G)  for(i in 1:10){
```

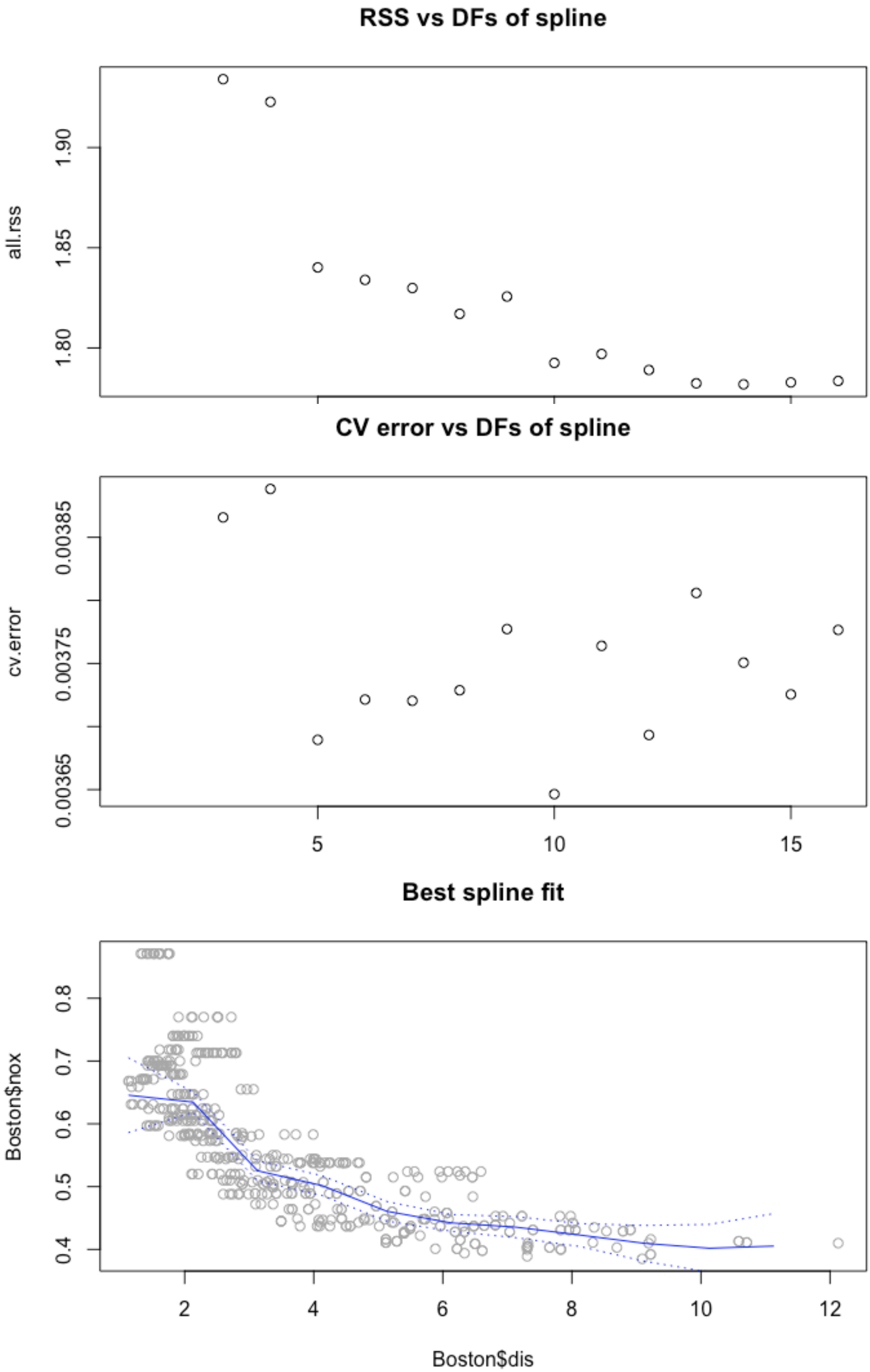


```
H)      lm.fit <- glm(nox~poly(dis, i), data=Boston)
I)      cv.error[i] <- cv.glm(Boston, lm.fit, K=10)$delta[1]
J)  }
K)
L)  print("CV error vs order of polynomial:")
M)  plot(1:10, cv.error)
N)  best.i <- which.min(cv.error)
O)  lm.fit <- glm(nox~poly(dis, best.i), data=Boston)
P)  pred <- predict(lm.fit, newdata=list(dis=dis.grid), se=T)
Q)  se.bands <-cbind(pred$fit + 2*pred$se.fit, pred$fit - 2*pred$se.fit)
R)  plot(Boston$dis, Boston$nox, col='darkgrey')
S)  lines(dis.grid, pred$fit, col='blue')
T)  matlines(dis.grid, se.bands, col='blue', lty=3)
U)  title("Best polynomial fit")
```

D) Fit using basis splines with bs() and 4 degrees of freedom. Where do the knots go? Answer: trick question! With 4 degrees of freedom there are no knots, so it's just the cubic fit again.
E) Fit a range of splines with different DFs, and report RSS.
F) Do CV and pick best DF.

Result: R complained while doing the CV, because the bounds of the variable changed a lot. I fit 3 to 16 DF splines. Best CV was at 10 DF, but I don't have much faith in it:

```
#7.9.9 ef
library(MASS)
library(boot)
library(splines)
set.seed(100)
dislims <- range(Boston$dis)
dis.grid <- seq(from=dislims[1], to=dislims[2])
cv.error <- rep(NA,16)
all.rss <- rep(NA,16)
spline.fit.4 <- glm(nox~bs(dis,4), data=Boston)
for(i in 3:16){
   lm.fit <- glm(nox~bs(dis, i), data=Boston)
   all.rss[i] <- sum((lm.fit$residuals)^2)
   cv.error[i] <- cv.glm(Boston, lm.fit, K=10)$delta[1]
}
plot(1:16, all.rss)
title("RSS vs DFs of spline")
plot(1:16, cv.error)
title("CV error vs DFs of spline")
best.i <- which.min(cv.error)
lm.fit <- glm(nox~bs(dis, best.i), data=Boston)
```


RSS vs DFs of spline
all.rss
1.80
1.85
1.90
CV error vs DFs of spline
cv.error
0.00365
0.00375
0.00385
5
10
15
Best spline fit
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis

```
pred <- predict(lm.fit, newdata=list(dis=dis.grid), se=T)
se.bands <-cbind(pred$fit + 2*pred$se.fit, pred$fit - 2*pred$se.fit)
plot(Boston$dis, Boston$nox, col='darkgrey')
lines(dis.grid, pred$fit, col='blue')
matlines(dis.grid, se.bands, col='blue', lty=3)
title("Best spline fit")
```

7.9.10: more practice on the College data set, using validation-set approach.

a) Split to test and training sets.
   Based on BIC, best model is 9 elements, but it’s not overwhelmingly better than anything in the 6-12 range. The predictors for 6-element fit are: Private, Room.Board, Terminal, perc.alumni, Expend, Grad.Rate. All are numeric except for Private. At 9 elements, add Personal, Accept, Enroll.
b) Fit a GAM on training data using the predictors identified above. I used natural splines and found no significant role for Terminal, significance to order 4 for Room.Board, perc.alumni, and Grad.Rate, and significance to order 5 for Expend. But some of the significance doesn’t hold up well when the model is trimmed. Perhaps natural splines of order below 4 don’t make much sense? Poly models also went to lower indices as I walked them down.
c) Evaluated on the test data and got MSE = 4.01e6, or RMS residual of $2000. Not too bad?
d) Some evidence for nonlinear behavior in Room.Board, Expend, and Grad.Rate, not in perc.alumni.

```
# 7.9.10
# a: split training set, and do forward stepwise selection to predict tuition based on
# everything else

set.seed(2)
library(ISLR)
library(leaps)
library(gam)
train <- sample(1:nrow(College), nrow(College)/2)
test <- (-train)
regfit.fwd <- regsubsets(Outstate ~ ., data=College[train,], nvmax=17, method="forward")
plot(regfit.fwd, scale='bic')
print(summary(regfit.fwd))

# b: fit a GAM with the best subset of predictors

gam.fit <- lm(Outstate ~ Private + ns(Room.Board, 4) # + Terminal
            + perc.alumni + ns(Expend, 5) + ns(Grad.Rate, 4),
            data=College, subset=train)
print(summary(gam.fit))
# par(mfrow=c(2,2))
# plot(gam.fit)
par(mfrow=c(3,2))
plot.Gam(gam.fit)
par(mfrow=c(1,1))

# C: evaluate the GAM on the test set and discuss.

preds <- predict(gam.fit, newdata=College[test,])
res <- preds - College[test, "Outstate"]
mse <- mean((res)^2)
print(paste("MSE on test data is:", mse))
```

```
par(mfrow=c(3,2))
plot(College$Private[test], res)
plot(College$Room.Board[test], res)
# plot(College$Terminal[test], res)
plot(College$perc.alumni[test], res)
plot(College$Expend[test], res)
plot(College$Grad.Rate[test], res)
par(mfrow=c(1,1))
```

7.9.11: learning about backfitting. For n=100 and p=2, the backfiring model converged after only 2 iterations!

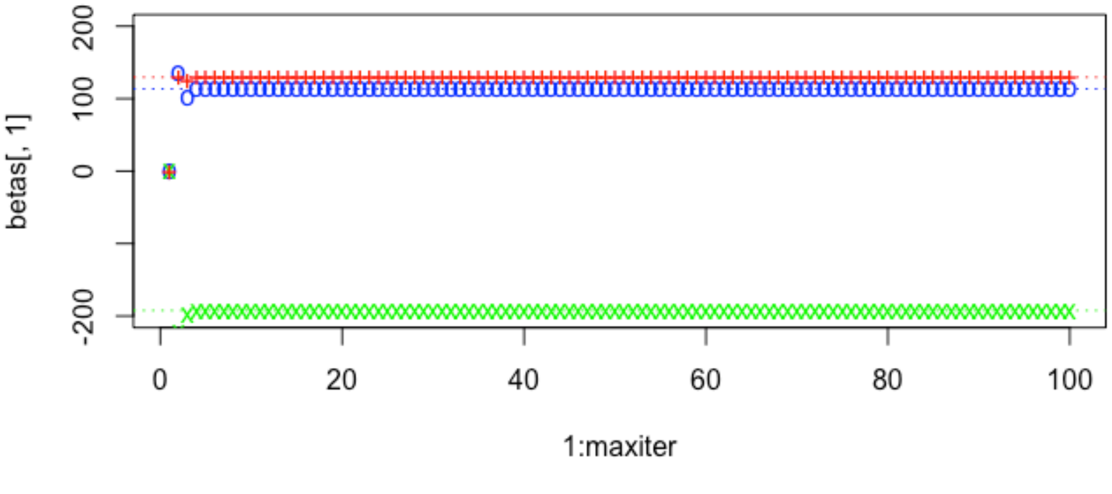
```
set.seed(3)
eps <- rnorm(100)
x1 <- rnorm(100)
x2 <- rnorm(100)
y <- x1 - 2* x2 + 1.5 + eps
beta1 <- 30
b0 <- rep(NA, 100)
b1 <- rep(NA, 100)
b2 <- rep(NA, 100)

for (i in 1:1000){
    a <- y-beta1*x1
    beta2 <- lm(a~x2)$coef[2]

    a <- y-beta2*x2
    beta1 <- lm(a~x1)$coef[2]
    beta0 <- lm(a~x1)$coef[1]
    betas[i,1] <- beta0
    betas[i,2] <- beta1
    betas[i,3] <- beta2
}

plot(1:1000, betas[,1], col='blue', ylim = c(-3,3), pch='.')
points(1:1000, betas[,2], col='green', pch='.')
points(1:1000, betas[,3], col='red', pch='.')

print(paste("Final backfit values:", beta0, beta1, beta2))
print("Multiple regression model provides:")
lm.fit <- lm(y~x1+x2)
print(summary(lm.fit))
co <- lm.fit$coef

abline(h=co[1], col='blue', lty=3)
abline(h=co[2], col='green', lty=3)
abline(h=co[3], col='red', lty=3)
```

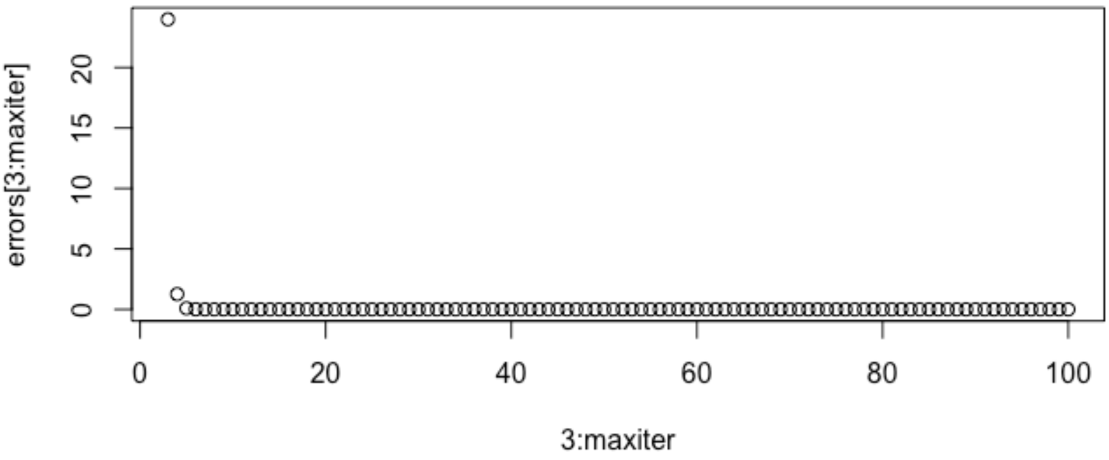
7.9.12: repeat with p=100

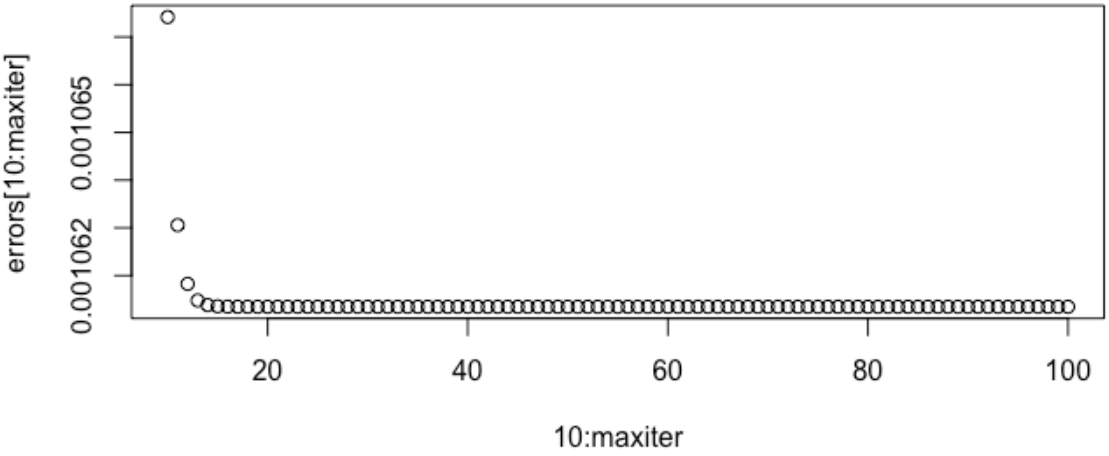
It took some struggling to get all my swapped indices identified, but here it is finally:

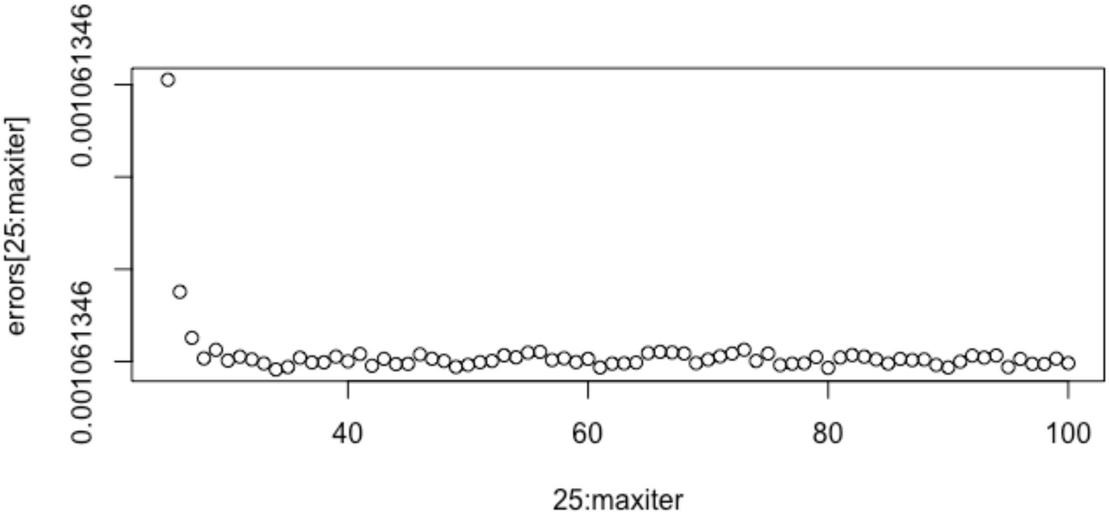
```
set.seed(1)
n=1000
```

```
p=100
maxiter = 100
x <- matrix(data=0, nrow=n, ncol=p)
real.betas <- rep(NA, p)
for (i in 1:p){
    x[, i] = rnorm(n)
    real.betas[i] = rnorm(1) * 100
}
y <- x %*% real.betas + rnorm(n)
betas <- matrix(data=NA, nrow=maxiter, ncol=p)
betas[1,] <- rep(0,p)

for (i in 2:maxiter){
    betas[i,] <- betas[i-1,]
    for (j in 1:p){
        a <- y - x %*% betas[i,] + x[,j]*betas[i,j]
        # a <- y - sum(betas[,i] * x) + betas[i,j]*x[i,j]

        betas[i,j] <- lm(a~x[,j])$coef[2]
    }
}

plot(1:maxiter, betas[,1], col='blue', pch='o', ylim=c(-200,200))
points(1:maxiter, betas[,2], col='green', pch='x')
points(1:maxiter, betas[,3], col='red', pch='+')

print(paste("Final backfit values:", betas[maxiter, 1:3]))
print("Multiple regression model provides:")
lm.fit <- lm(y~x)
co <- lm.fit$coef
print(co[2:4])
abline(h=co[2], col='blue', lty=3)
abline(h=co[3], col='green', lty=3)
abline(h=co[4], col='red', lty=3)

print("MSE in betas with iteration:")
errors <- rep(NA, maxiter)
for (i in 1:maxiter){
    errors[i] <- mean((betas[i,] - real.betas)^2)
```









```
}
plot(1:maxiter, errors)
```

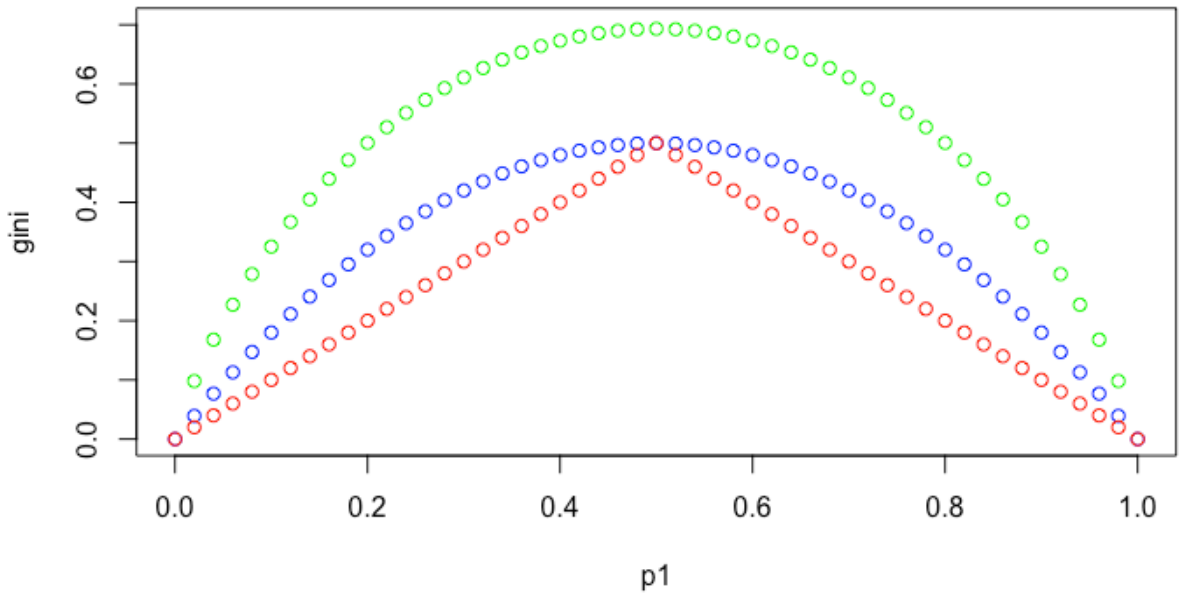
The error declines fast, exponentially from the look of it, until about iteration 30, at which point it becomes noise, I would guess limited by the system precision. Note that mean(real.betas^2) is about 9730. So by anyone’s standards the precision is excellent by iteration 5 or so.
Chapter 8

8.4.1: pictorial

8.4.2: obvious, to me anyway

8.4.3.: Use this code.

```
p1 <- seq(from=0, to=1, by=0.02)
p2 <- 1- p1
gini <- p1 * (1-p1) + p2*(1-p2)
```



```
crossentropy <- -p1*log(p1) - p2*log(p2)
classerror <- 1- pmax(p1,p2)

plot(p1, gini, col='blue', ylim=c(0,0.7))
points(p1, crossentropy, col='green')
points(p1, classerror, col=‘red')
```

8.4.4: pictorial

8.4.5: Since 4 of the probabilities are <0.5 and 5 are >0.5, majority vote prediction is “red”. But since the mean probability is 0.45, average prediction is “green”.

8.4.6: How to fit a regression tree, detailed version.

Start with N=1 region in p predictor dimensions, with n observations. Iterate the following until a stopping condition is reached.

```
1) For i=1..N:
    1) For j=1..p:
```

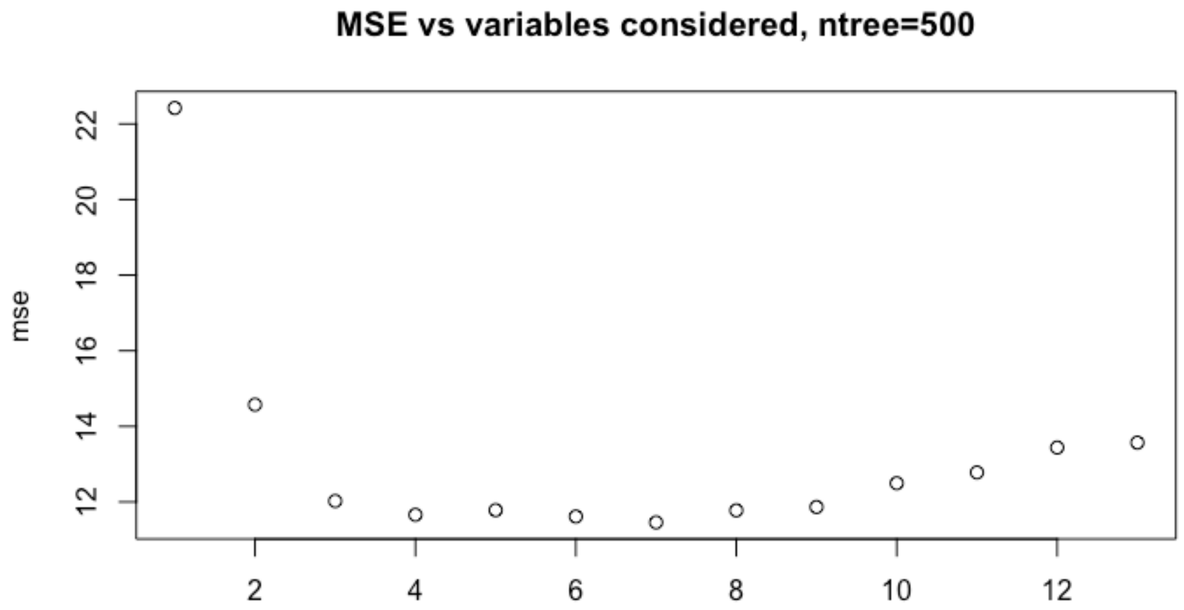
    1) Within the bounds of region R_i, vary cut-point for predictor x_j and minimize RSS as a function of the cut-point. (How this is done isn’t exactly clear… it might be binary splitting, Newton’s method, or some analytical trick I haven’t identified.)

2) Among all the (i,j), pick whatever split yielded the lowest RSS.

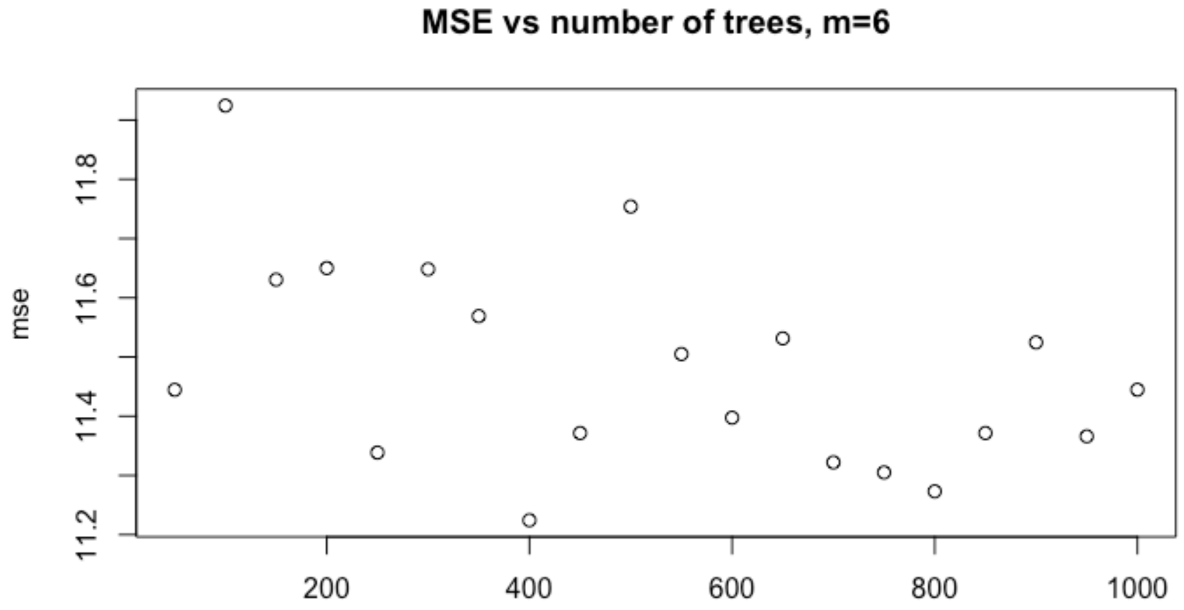
3) Repeat with the new N+1 regions. Stop when there are too few observations in all regions to afford further splitting.

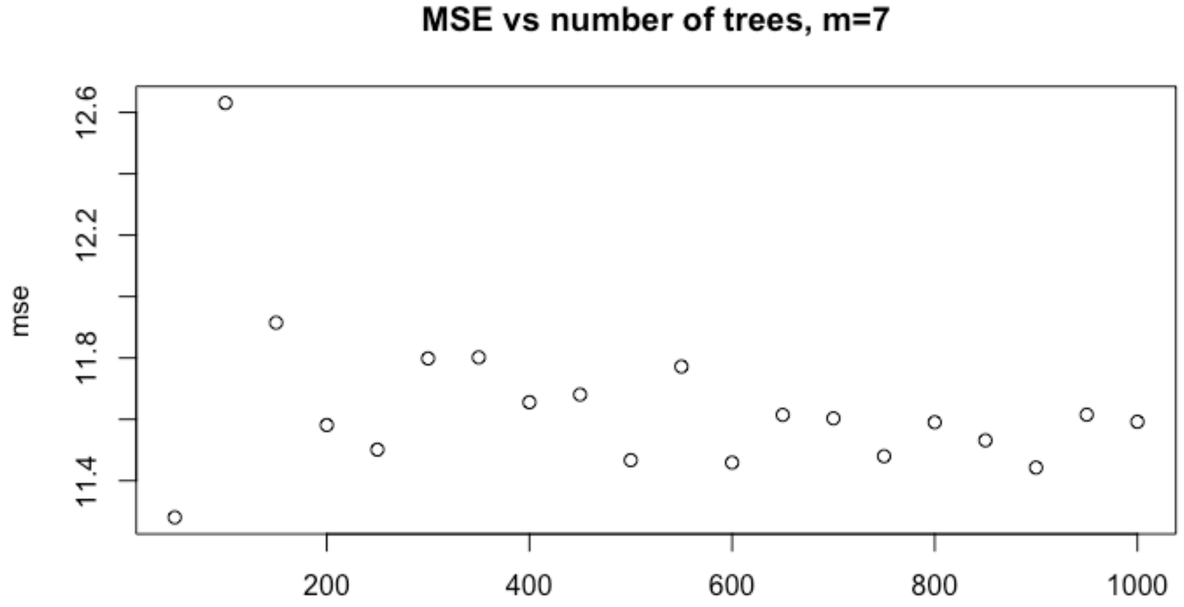
8.4.7: Repeat random forest analysis to the Boston data using other values of mtry and tree, and show how test error varies with these.

```
library(MASS)
library(randomForest)
set.seed(1)
train = sample(1:nrow(Boston),nrow(Boston)/2)
boston.test=Boston[-train, 'medv']
m <- seq(from=1, to=13, by=1)
num <- seq(from=50, to= 1000, by=50)

# first do the series in mtry, with ntree=500
l <- length(m)
mse <- rep(NA, l)
for (i in 1:l){
   bag.boston <- randomForest(medv~., data=Boston, subset=train, mtry=m[i],
                              importance=TRUE, ntree=500)
   yhat.bag <- predict(bag.boston, newdata=Boston[-train,])
   mse[i] <- mean((yhat.bag-boston.test)^2)
}

plot(m, mse)
title('MSE vs variables considered, ntree=500')

# now do the series in ntree, with mtry=6
l <- length(num)
mse <- rep(NA, l)
for (i in 1:l){
   bag.boston <- randomForest(medv~., data=Boston, subset=train, mtry=6,
                              importance=TRUE, ntree=num[i])
```



```
  yhat.bag <- predict(bag.boston, newdata=Boston[-train,])
  mse[i] <- mean((yhat.bag-boston.test)^2)
```

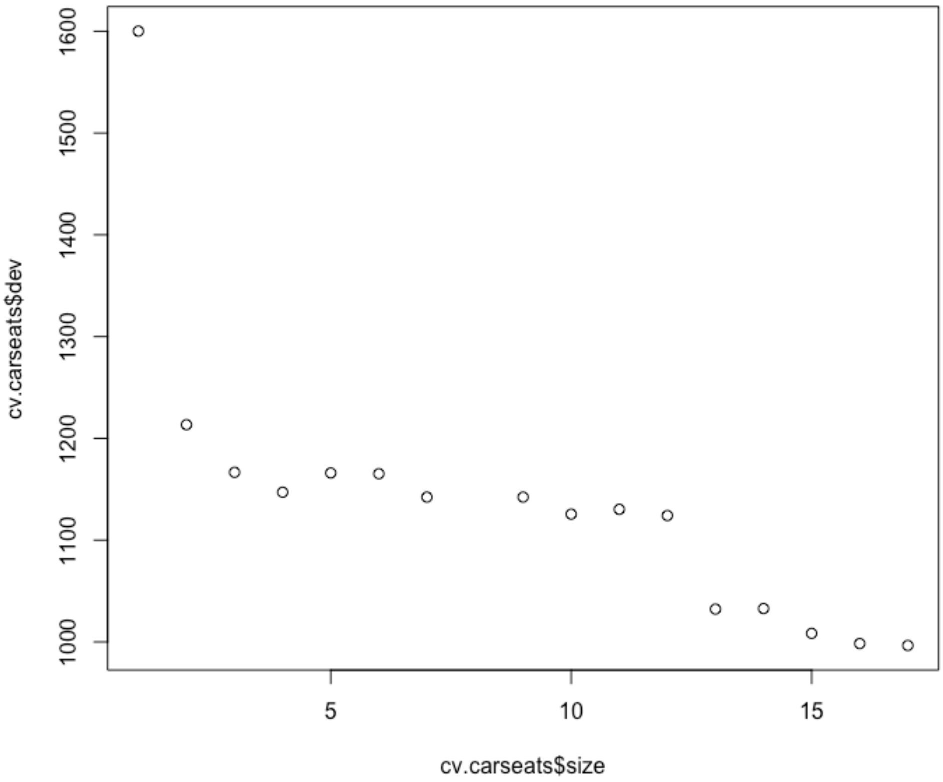


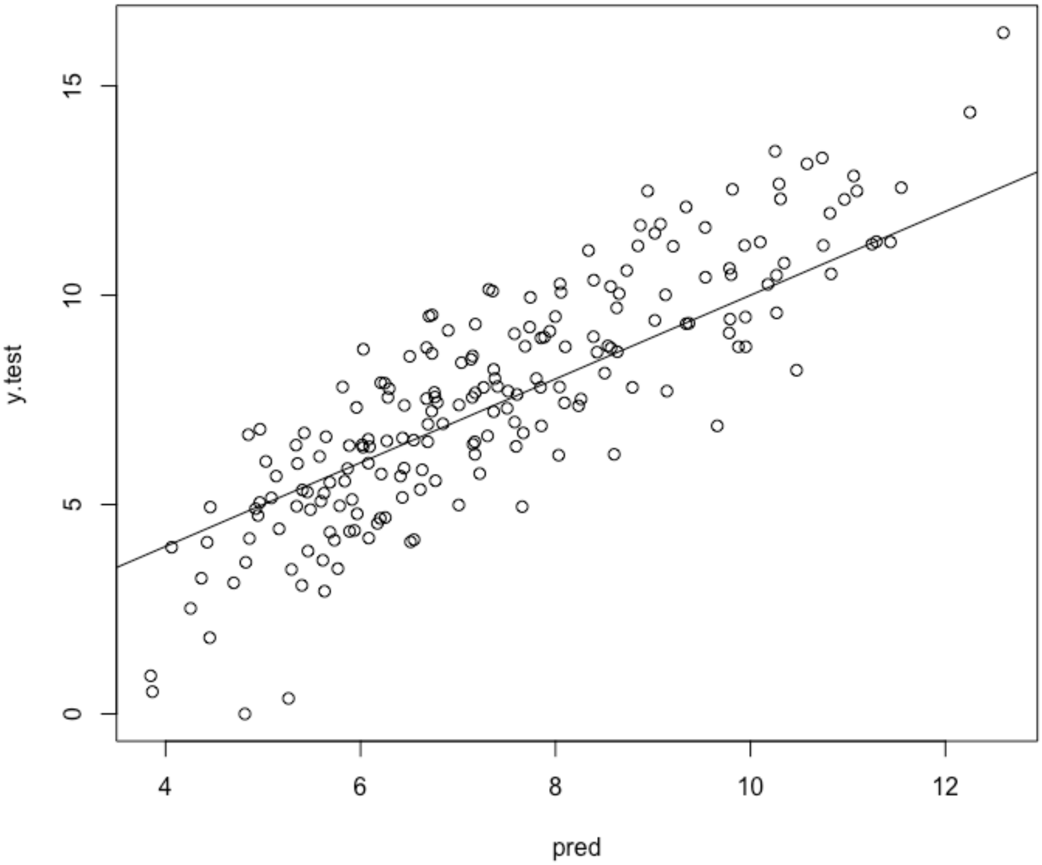


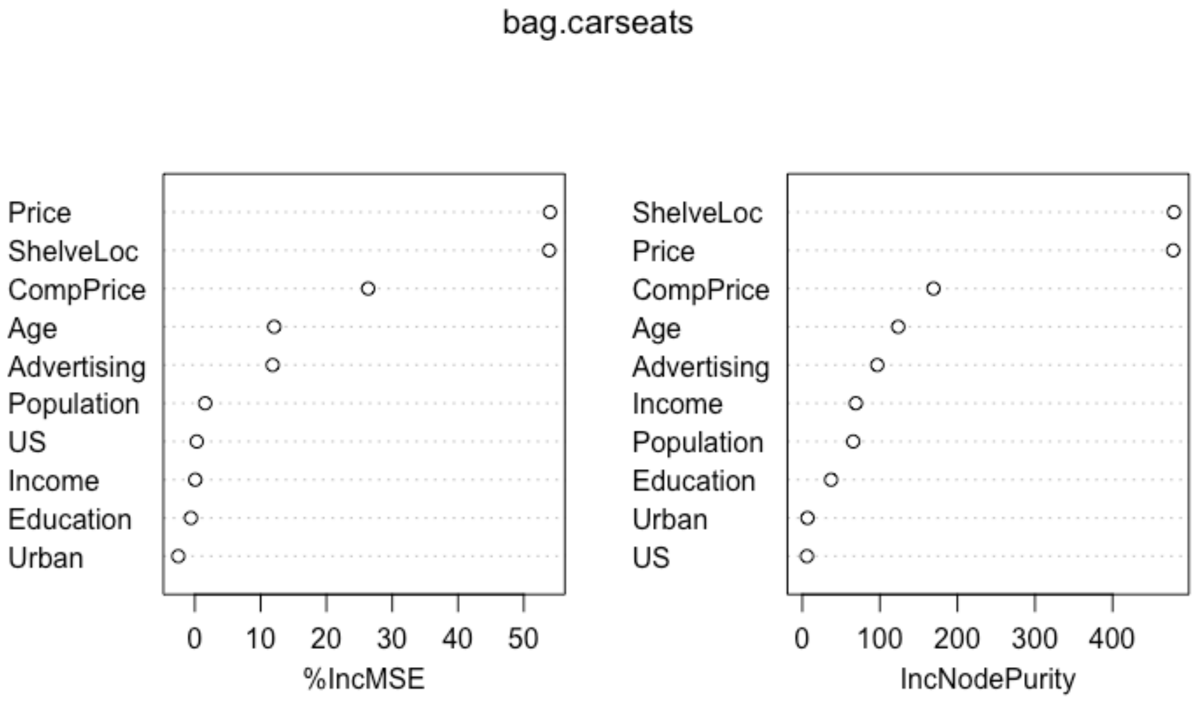
```
}

plot(num, mse)
title('MSE vs number of trees, m=6')
```

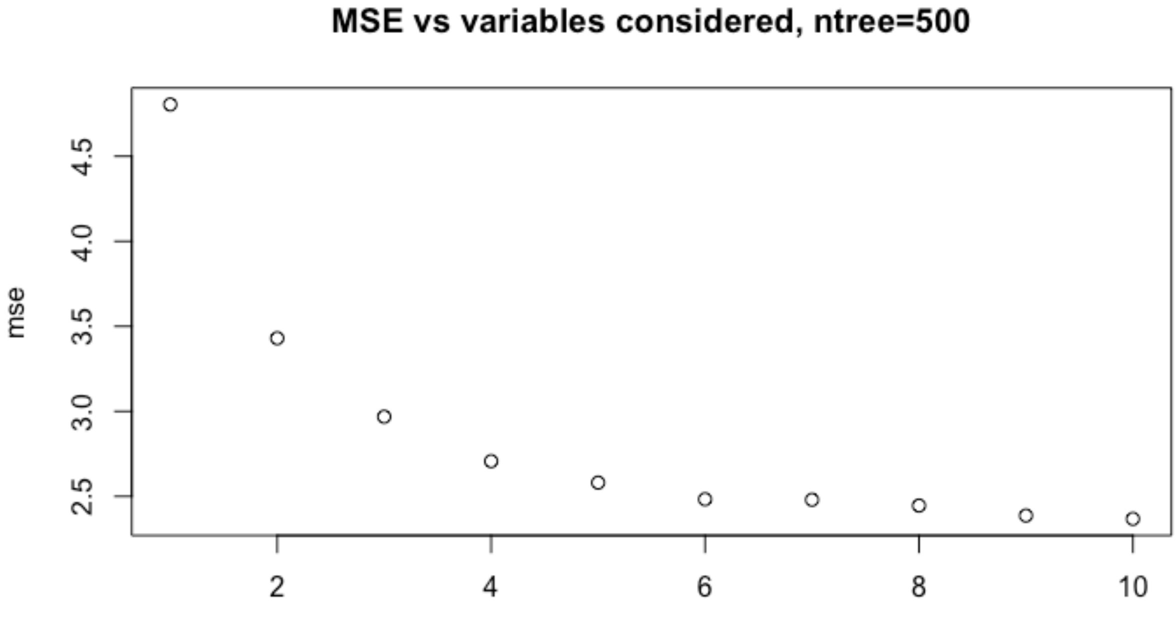
Results show a pretty clear minimum around m=4,…,8, but not much signal once the number of trees is above 100.

8.4.8: Do a tree on Carseats, but this time as a regression tree, not as a classification tree — so we skip the







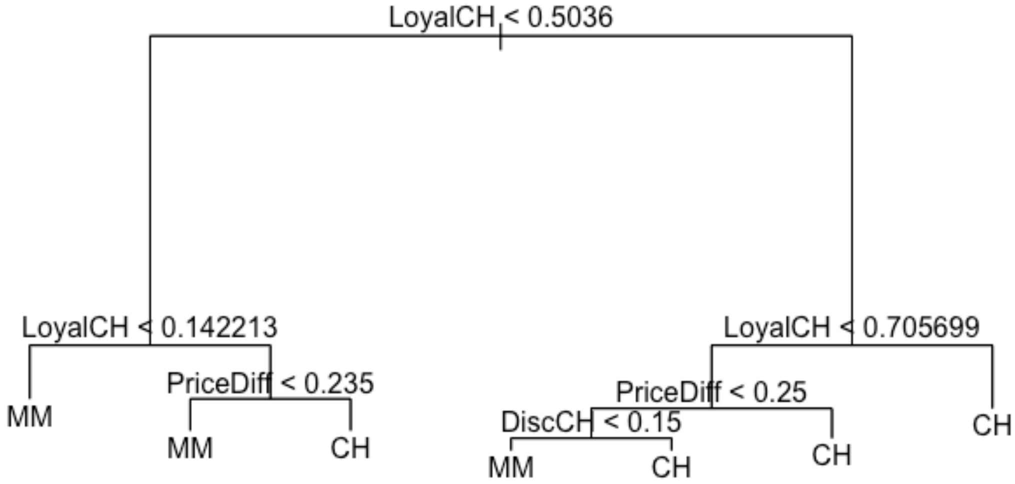


conversion of Sales to a qualitative variable. Using validation set, do single tree with pruning determined by CV, then bagging, then random forest.

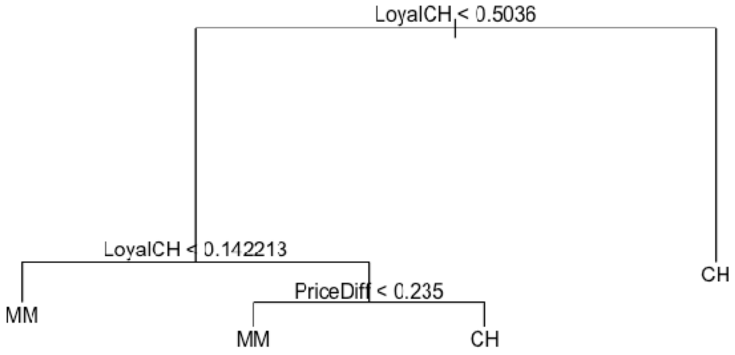
Results:

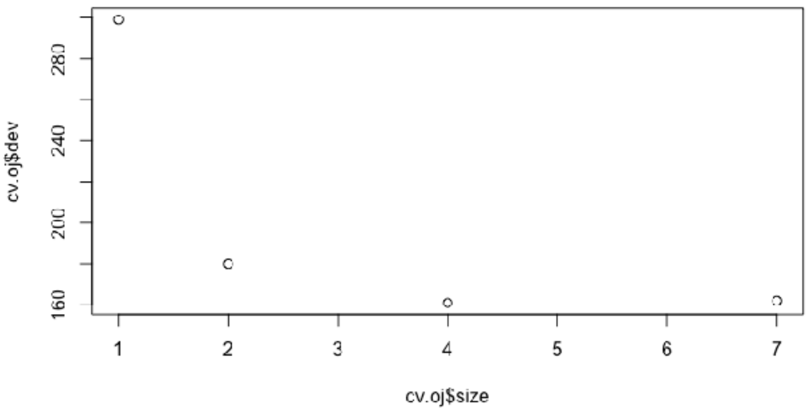
1) The best single tree is the most complex — pruning doesn’t help. Test MSE= 4.84
2) With bagging, test MSE=2.40. Most important vars are Price, ShelveLoc, then CompPrice.
3) With random forests, the best MSE is at m=10, thus equivalent to bagging.

```
library(tree)
library(randomForest)
library(ISLR)
```



```
# a: split off a test set
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train,]
y.test <- Carseats[-train, "Sales"]
# b: fit a regression tree, plot, calculate test MSE
tree.carseats <- tree(Sales ~ ., Carseats, subset=train)
plot(tree.carseats)
text(tree.carseats, pretty=0)
tree.pred <- predict(tree.carseats, Carseats.test)
mse <- mean((tree.pred - y.test)^2)
print(mse)
# c: do CV to determine optimal level of pruning
```





```
set.seed(3)
cv.carseats <- cv.tree(tree.carseats)
print(cv.carseats)
plot(cv.carseats$size, cv.carseats$dev)
# d: do bagging. Note that there are 11 predictors.
bag.carseats <- randomForest(Sales~., data=Carseats, subset=train, mtry=10,
                        importance=TRUE)
pred <- predict(bag.carseats, newdata=Carseats[-train,])
print(bag.carseats)
plot(pred, y.test)
abline(0,1)
print(mean((pred-y.test)^2))
plot(bag.carseats)
```

```
varImpPlot(bag.carseats)
# e: do random forest, first with default mtry, then look at effect of mtry on MSE
m <- seq(from=1,to=10,by=1)
l <- length(m)
mse <- rep(NA, l)
for (i in 1:l){
   rf.carseats <- randomForest(Sales~., data=Carseats, subset=train, mtry=m[i],
                               importance=TRUE, ntree=500)
   pred <- predict(rf.carseats, newdata=Carseats[-train,])
   mse[i] <- mean((pred-y.test)^2)
}

plot(m, mse)
title('MSE vs variables considered, ntree=500')

best.m <- which.min(mse)
rf.carseats <- randomForest(Sales~., data=Carseats, subset=train, mtry=best.m,
                            importance=TRUE, ntree=500)
varImpPlot(rf.carseats)
plot(rf.carseats)
```

8.4.9: OJ data set from library(ISLR). Results, then code:
B) Basic tree summary results: 7 terminal nodes, training error rate 18.4%, predictors used are LoyalCH, PriceDiff, DiscCH. Residual mean deviance is 0.7786, which seems not great.

```
C) 1) root 800 1064.00 CH ( 0.61750 0.38250 )
   2) LoyalCH < 0.5036 354  435.50 MM ( 0.30508 0.69492 )
     4) LoyalCH < 0.142213 100   45.39 MM ( 0.06000 0.94000 ) *
```

According to the key, this format is:
NODE_NUM), N_in_class, Deviance at node, Yval_at_node, (yprobs_at_node_list)
So node 4 means: If LoyalCH < 0.142213, the region contains 100 observations, deviance is 45.39 within the region, and the probabilities are (P_CH, P_MM) = (0.06, 0.94).
Plot on next page. Confusion matrix on next page. Test error rate is (32+34)/270 = 24.4%
With pruning, optimal tree size is 4 nodes, only 1 error better than 7 nodes! But let's go with it anyway.
Confusion matrix for default tree:

```
       y.test
oj.pred  CH  MM
     CH 125  32
     MM  34  79
```

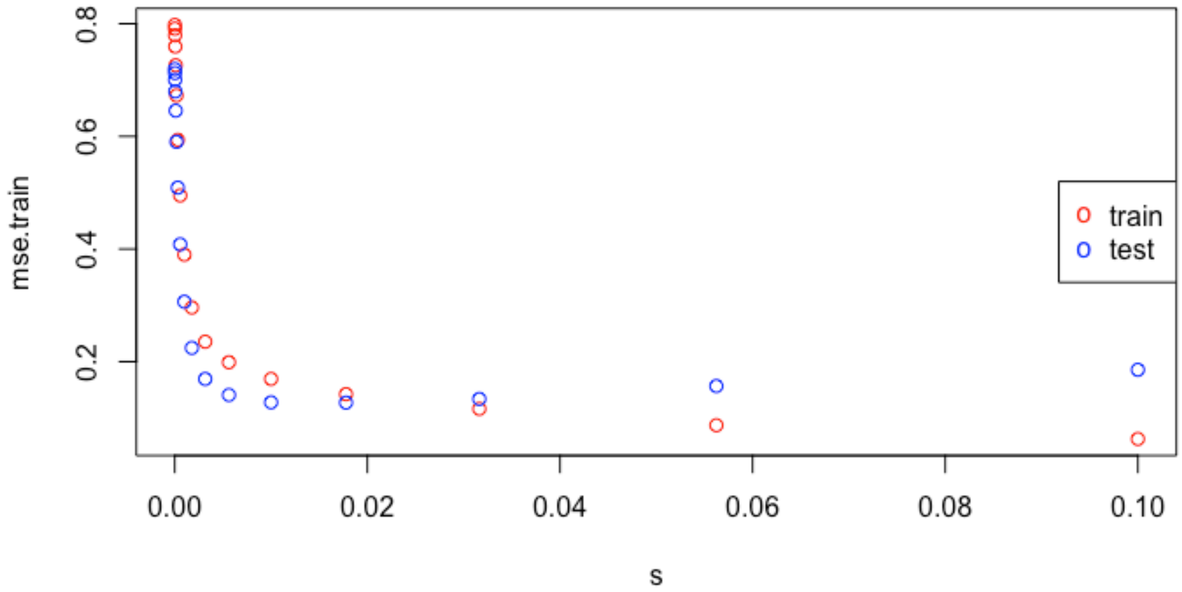
Confusion matrix for pruned tree:

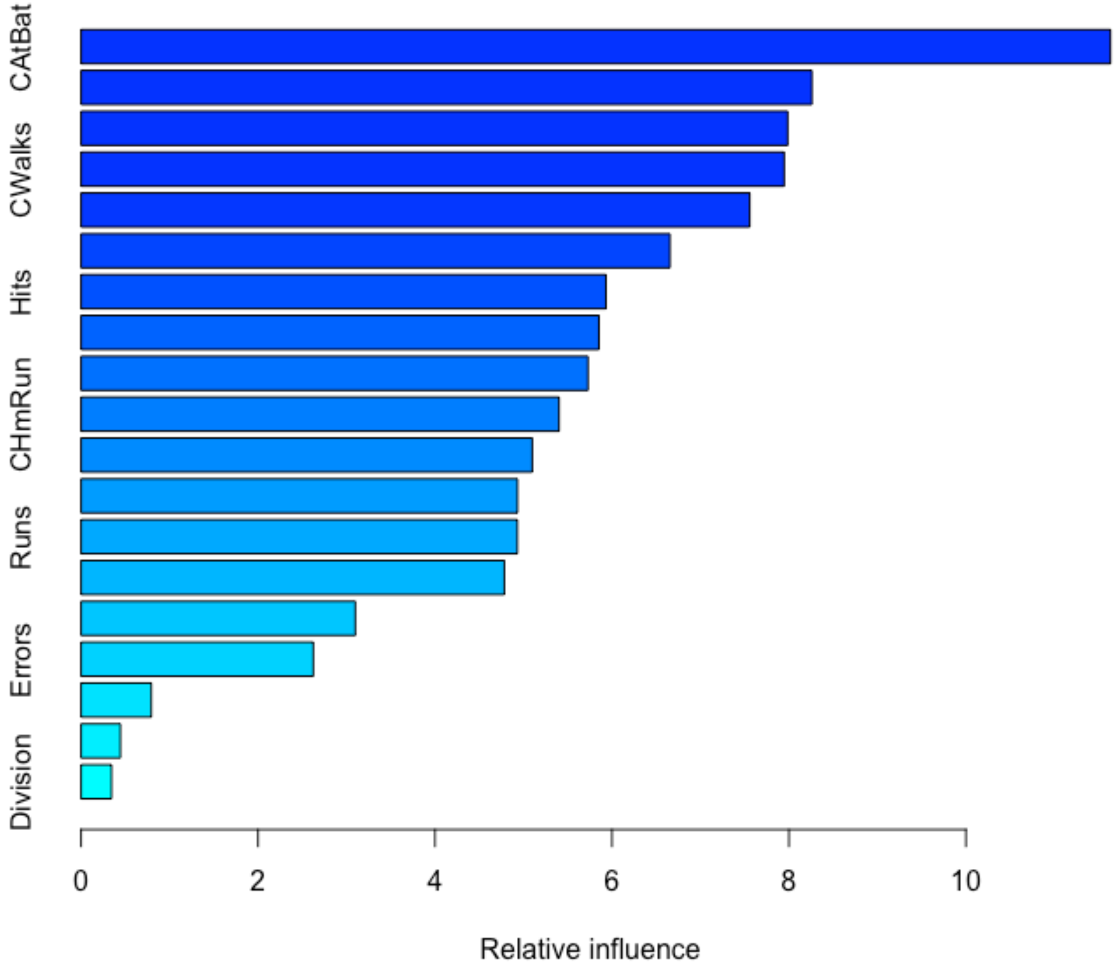
```
       y.test
oj.pred  CH  MM
     CH 142  41
     MM  17  70          <<<< Error rate now down to 21.5%
```

Training error rate for pruned tree is 0.1875. Predictors used are just LoyalCH and PriceDiff.

```
library(ISLR)
library(tree)
set.seed(5)
train <- sample(1:nrow(OJ), 800)
y.test <- OJ[-train, "Purchase"]
oj.tree <- tree(Purchase ~ ., data=OJ, subset=train)
oj.pred <- predict(oj.tree, newdata=OJ[-train,], type='class')
```

```
print("Default tree")
print(summary(oj.tree))
print(oj.tree)
plot(oj.tree)
text(oj.tree, pretty=0)
print(table(oj.pred, y.test))
cv.oj <- cv.tree(oj.tree, FUN=prune.misclass)
```





```
print("CV treatment of tree")
print(cv.oj)
plot(cv.oj$size, cv.oj$dev)
prune.oj <- prune.misclass(oj.tree, best=4)
print("Tree pruned to 4 nodes")
plot(prune.oj)
text(prune.oj, pretty=0)
print(summary(prune.oj))
oj.pred <- predict(prune.oj, newdata=OJ[-train,], type='class')
print(table(oj.pred, y.test))
```

8.4.10: Predict Salary in Hitters data set, using boosting and then bagging. Compare to regression.

Results: training MSE decreased steadily with shrinkage, which is not what I expected initially, but kind of makes sense. Test MSE had a minimum at shrinkage=0.018, with MSE=0.127. To compare regression requires rerunning, because of having log-transformed the data, but whatever. The linear fit to “everything” yielded test MSE=0.331. I’m too lazy to do fancier linear fits right now. Most important variables in boosted model are CAtBat by a lot (probably proxy for career activity), then Runs, Walks, CWalks, and PutOuts, and then it tapers off. Interestingly, Years is near the bottom of influence, possibly because it’s strongly confounded with CAtBat which already got a lot of leverage. League, NewLeague, and Division are the least important by a lot. Bagged model gives test MSE=0.138, slightly worse than boosted model.

```
library(ISLR)
library(gbm)
library(randomForest)
original.Hitters <- Hitters
Hitters.nona <- na.omit(original.Hitters)
Hitters.nona[, "logSal"] <- log(Hitters.nona[, "Salary"])
set.seed(6)
train <- sample(1:nrow(Hitters.nona), 200)
s <- 10^seq(from=-5, to=-1, by=0.25)
mse.train <- rep(NA, length(s))
mse.test <- rep(NA, length(s))
for (i in 1:length(s)){
   boost.hit <- gbm(logSal ~ .-Salary, data=Hitters.nona[train,], distribution='gaussian',
            n.trees=1000, shrinkage=s[i])
   pred.train <- predict(boost.hit, newdata=Hitters.nona[train,], n.trees=1000)
   mse.train[i] <- mean((pred.train - Hitters.nona[train, "logSal"])^2)
   pred.test <- predict(boost.hit, newdata=Hitters.nona[-train,], n.trees=1000)
   mse.test[i] <- mean((pred.test - Hitters.nona[-train, "logSal"])^2)
}

plot(s, mse.train, col='red')
points(s, mse.test, col='blue')
legend('right', c('train', 'test'), pch='o', col=c('red','blue'))

lm.fit <- lm(logSal ~ .-Salary, data=Hitters.nona, subset=train)
lm.pred <- predict(lm.fit, newdata=Hitters.nona[-train,])
mse.test.lm <- mean((lm.pred - Hitters.nona[-train, 'logSal'])^2)
print(paste("Test MSE for linear model is:", mse.test.lm))

# Now do bagging
```

```
bag.hit <- randomForest(logSal ~ .-Salary, data=Hitters.nona, subset=train, mtry=19,
                      importance=TRUE)
bag.pred <- predict(bag.hit, newdata=Hitters.nona[-train,])
mse.test.bag <- mean((bag.pred - Hitters.nona[-train,"logSal"])^2)
print(paste("For bagging model, test MSE is:", mse.test.bag))
plot(bag.hit)
```

8.4.10: Use the Caravan data set from library(ISLR), apply boosting, and compare to KNN or logistic regression, which were done on pp 165-167.

The boosting package needs a qualitative variable to be coded 0/1, not “No”/“Yes”, so I first converted Purchase to a binarized version. Most important variables were:

```
              var      rel.inf
PPERSAUT PPERSAUT 13.99544571
MOSTYPE   MOSTYPE 11.58079821
PBRAND     PBRAND 10.63429868
APERSAUT APERSAUT  8.79781328
PWAPART   PWAPART  5.17347749
MINKGEM   MINKGEM  4.56638969
PMOTSCO   PMOTSCO  3.21757494
```

and then everything else had influence < 3

Confusion matrix for the boosted model:

```
      boost.pred
y.test    0    1
     0 4412  121
     1  255   34
```

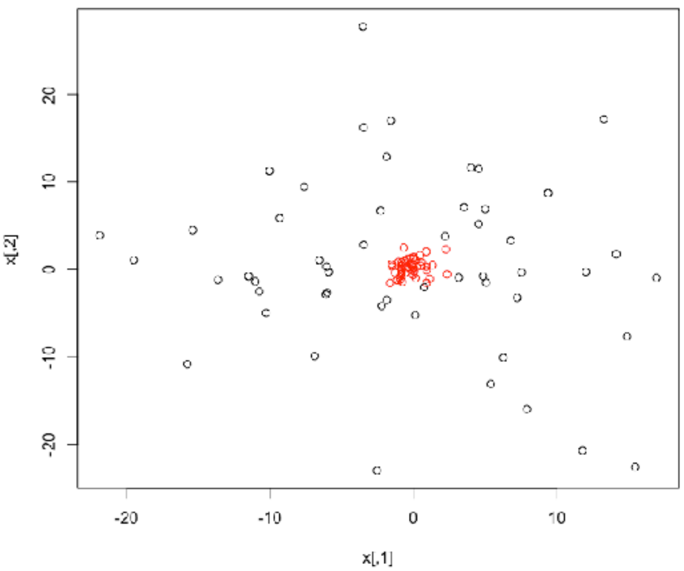
Not too bad. Only 22% of the predicted purchasers actually make a purchase, but only 6% of the sample makes a purchase, so the sample provides some refinement.

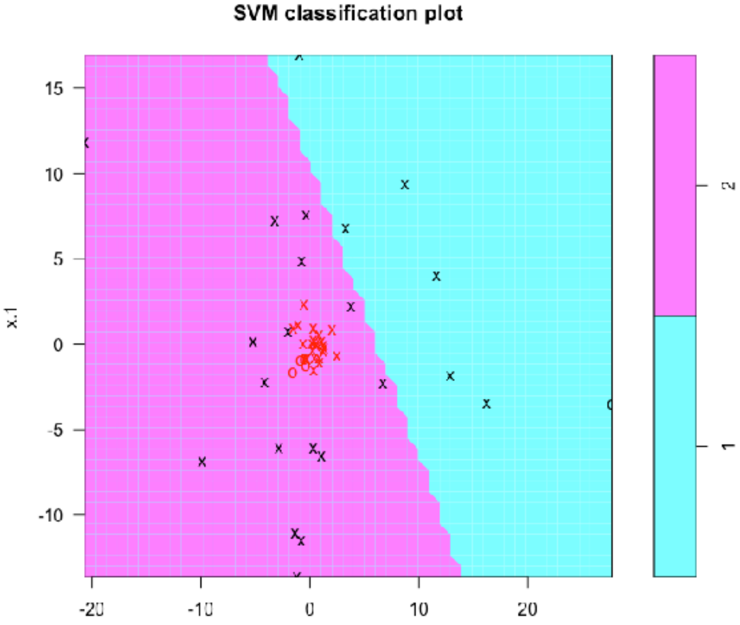
KNN got up to 26.7% in this success rate with K=5, and logistic regression with a probability threshold of 0.25 got up to 33% success, so the boosted model is not preferred.

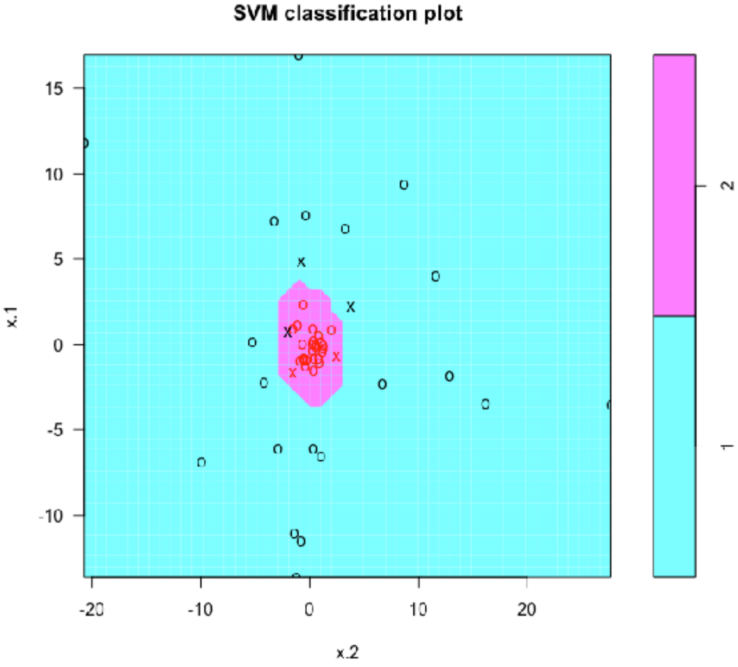
```
library(ISLR)
library(gbm)
set.seed(7)
train <- 1:1000
Caravan[,'pbin'] <- ifelse(Caravan[,'Purchase']=="Yes", 1, 0)
y.test <- Caravan[-train, 'pbin']
boost.fit <- gbm(pbin ~ .-Purchase, data=Caravan[train,], distribution='bernoulli',
                n.trees=1000, shrinkage=0.01)
# print(summary(boost.fit))
boost.probs <- predict(boost.fit, newdata=Caravan[-train,], n.trees=1000,
                     type='response')
boost.pred <- ifelse(boost.probs>0.2, 1, 0)
print("Boosted model confusion matrix:")
print(table(y.test, boost.pred))
```

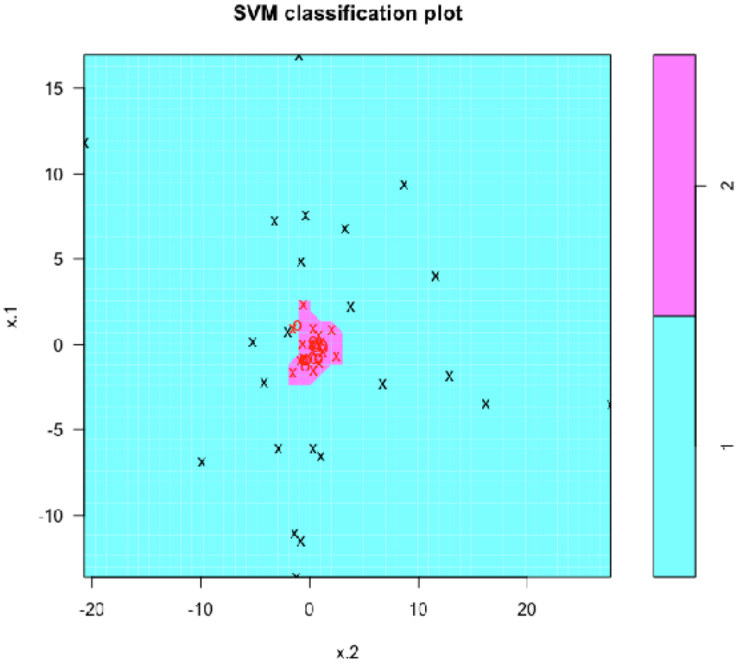
8.4.11: skip because it’s just trees on another data set of my choice.

9.7.1, 9.7.2: basic sketching of “hyperplanes” in 2d.




SVM classification plot


SVM classification plot

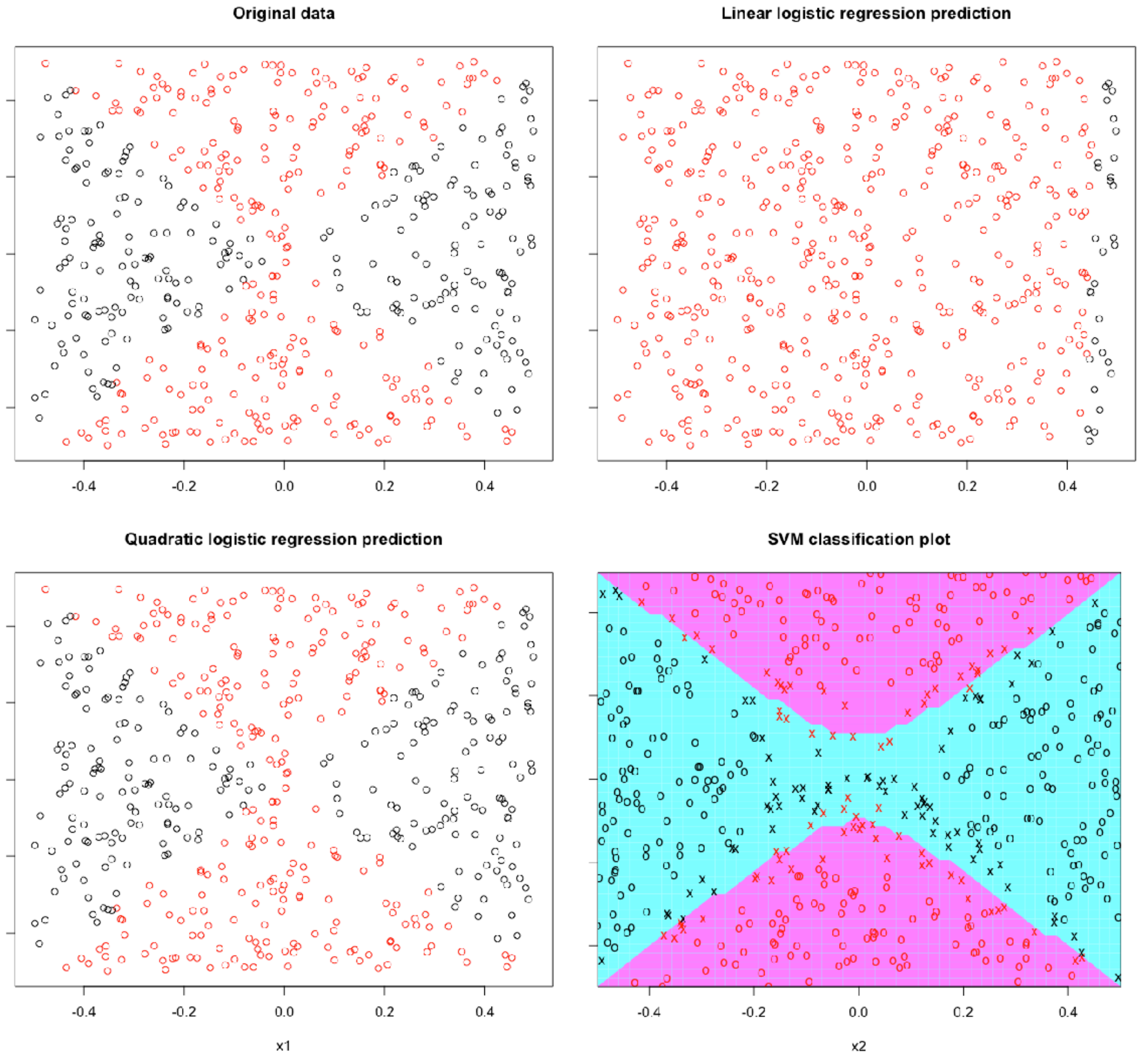

SVM classification plot

9.7.3: a more interesting hyperplane exercise. By inspection, the optimal separating hyperplane has equation 0 = -0.5 + X1 - X2, with “red” on the negative side and “blue” on the positive side. The support vectors are observations 2, 3, 5, 6.

9.7.4: Make simulated data with a clear nonlinear separation, and show that nonlinear SVM works better than linear SVC. Code below. I made a radially separated population.

```
library(e1071)
set.seed(11)

x <- matrix(rnorm(2*100), ncol=2)
y <- sample(c(1,2), 100, replace=TRUE)
x[y==1, ] = 10*x[y==1, ]
```

```
train <- sample(1:100, 50)
plot(x, col=y)
dat <- data.frame(x=x, y=as.factor(y))
svmfit.lin <- svm(y~., data=dat[train, ], kernel='linear', cost=10, scale=F)
svmfit.poly <- svm(y~., data=dat[train, ], kernel='polynomial', cost=10,
                 degree=2, scale=F)
svmfit.rad <- svm(y~., data=dat[train, ], kernel='radial', cost=10,
                gamma=1, scale=F)
pred.lin <- predict(svmfit.lin, dat[-train,])
pred.poly <- predict(svmfit.poly, dat[-train,])
pred.rad <- predict(svmfit.rad, dat[-train,])
print("Confusion matrix: linear SVM")
print(table(pred.lin, y[-train]))
```



```
print("Confusion matrix: poly SVM")
print(table(pred.poly, y[-train]))
print("Confusion matrix: radial SVM")
print(table(pred.rad, y[-train]))
plot(svmfit.lin, dat[train,])
plot(svmfit.poly, dat[train,])
plot(svmfit.rad, dat[train,])
plot(svmfit.lin, dat[-train,])
plot(svmfit.poly, dat[-train,])
plot(svmfit.rad, dat[-train,])
```

Confusion matrix outputs:

```
[1] "Confusion matrix: linear SVM"

pred.lin  1  2
       1  8  0
       2 21 21
[1] "Confusion matrix: poly SVM"

pred.poly  1  2
        1 28  0
        2  1 21
[1] "Confusion matrix: radial SVM"

pred.rad  1  2
       1 28  1
       2  1 20
```

Plots of the training data are below. Linear SVM is obviously very bad. Lower left is poly-2 kernel, lower right is radial kernel with gamma=1. I don’t have a good intuition for how the kernel affects the separatrix, apparently — I didn’t expect the “island” in the poly-2.
9.7.5: Instead of using a nonlinear kernel, we could do a linear fit on some higher order transformations of the predictors. If this is done for logistic regression, we can get rather similar results to the SVM case.

```
library(e1071)
set.seed(1)
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1*(x1^2 -x2^2 > 0)
plot(x1, x2, col=2-y)
title(main='Original data')
dat <- data.frame(x1=x1, x2=x2, y=as.factor(y))
glm.lin <- glm(y ~ ., data=dat, family=binomial)
print(summary(glm.lin))
glm.probs <- predict(glm.lin, type='response')
y.pred <- ifelse(glm.probs > 0.5, 1, 0)
plot(x1, x2, col=2-y.pred)
title(main='Linear logistic regression prediction')
glm.poly <- glm(y ~ I(x1^2) + I(x2^2), data=dat, family=binomial)
print(summary(glm.poly))
glm.poly.probs <- predict(glm.poly, type='response')
y.poly <- ifelse(glm.poly.probs > 0.5, 1, 0)
```

```
plot(x1, x2, col=2-y.poly)
title(main='Quadratic logistic regression prediction')
svm.lin <- svm(y~., data=dat, kernel='linear', cost=100, scale=FALSE)
plot(svm.lin, dat)
y.svc <- predict(svm.lin, dat)
plot(x1, x2,
    col=ifelse(y.svc=='0', 2, 1))
title(main='SVC prediction')
svm.poly <- svm(y~., data=dat, kernel='polynomial', cost=100, degree=2, scale=FALSE)
plot(svm.poly, dat)
y.svm <- predict(svm.poly, dat)
plot(x1, x2,
    col=ifelse(y.svm=='0', 2, 1))
title(main='SVM prediction’)
```

Note that the quadratic logistic regression plot gets 100% accuracy! And I could force the SVM to 100% accuracy if I set the cost parameter high enough, to 1e5 or more. I suppose this comes from there being an actual exact separatrix.

9.7.6: Try to illustrate how setting a lower cost during training can lead to a better performance in test, even though training error goes down monotonically as cost increases. But I didn’t understand the question on the first try and I still don’t think it is well worded. PrinceHonest’s solution makes sense but doesn’t seem to be fully motivated by the question. Briefly, he made a “real” division with a wide margin around a x1 - x2 >0 rule, and then added “noise” points that don’t obey the original rule, then created test points according to the original rule. Surprise! The test went better when cost was moderate so the noise could be ignored. Their code:

```
set.seed(3154)
# Class one
x.one = runif(500, 0, 90)
y.one = runif(500, x.one + 10, 100)
x.one.noise = runif(50, 20, 80)
y.one.noise = 5/4 * (x.one.noise - 10) + 0.1

# Class zero
x.zero = runif(500, 10, 100)
y.zero = runif(500, 0, x.zero - 10)
x.zero.noise = runif(50, 20, 80)
y.zero.noise = 5/4 * (x.zero.noise - 10) - 0.1

# Combine all
class.one = seq(1, 550)
x = c(x.one, x.one.noise, x.zero, x.zero.noise)
y = c(y.one, y.one.noise, y.zero, y.zero.noise)

plot(x[class.one], y[class.one], col = "blue", pch = "+", ylim = c(0, 100))
points(x[-class.one], y[-class.one], col = "red", pch = 4)

library(e1071)

set.seed(555)
z = rep(0, 1100)
z[class.one] = 1
data = data.frame(x = x, y = y, z = z)
```

```
tune.out = tune(svm, as.factor(z) ~ ., data = data, kernel = "linear", ranges = list(cost = c(0.01,
    0.1, 1, 5, 10, 100, 1000, 10000)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  10000
##
## - best performance: 0
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.05636    0.02260
## 2 1e-01 0.04636    0.01841
## 3 1e+00 0.04545    0.01868
## 4 5e+00 0.05000    0.02153
## 5 1e+01 0.04818    0.02102
## 6 1e+02 0.04727    0.02134
## 7 1e+03 0.02364    0.02019
## 8 1e+04 0.00000    0.00000
```

```
data.frame(cost = tune.out$performances$cost, misclass = tune.out$performances$error *
    1100)
```

```
##    cost misclass
## 1 1e-02       62
## 2 1e-01       51
## 3 1e+00       50
## 4 5e+00       55
## 5 1e+01       53
## 6 1e+02       52
## 7 1e+03       26
## 8 1e+04        0
```

```
set.seed(1111)
x.test = runif(1000, 0, 100)
class.one = sample(1000, 500)
y.test = rep(NA, 1000)
# Set y > x for class.one
for (i in class.one) {
    y.test[i] = runif(1, x.test[i], 100)
}
# set y < x for class.zero
for (i in setdiff(1:1000, class.one)) {
    y.test[i] = runif(1, 0, x.test[i])
}
plot(x.test[class.one], y.test[class.one], col = "blue", pch = "+")
points(x.test[-class.one], y.test[-class.one], col = "red", pch = 4)
```

```
set.seed(30012)
z.test = rep(0, 1000)
z.test[class.one] = 1
all.costs = c(0.01, 0.1, 1, 5, 10, 100, 1000, 10000)
test.errors = rep(NA, 8)
data.test = data.frame(x = x.test, y = y.test, z = z.test)
for (i in 1:length(all.costs)) {
    svm.fit = svm(as.factor(z) ~ ., data = data, kernel = "linear", cost = all.costs[i])
    svm.predict = predict(svm.fit, data.test)
    test.errors[i] = sum(svm.predict != data.test$z)
}

data.frame(cost = all.costs, `test misclass` = test.errors)

##    cost test.misclass
## 1 1e-02            57
## 2 1e-01            17
## 3 1e+00             5
## 4 5e+00             1
## 5 1e+01             0
## 6 1e+02           204
## 7 1e+03           227
## 8 1e+04           227
```

9.7.7: Auto data set, to model whether a car has above-median mpg based on other predictors, using SVC / SVM.

This is a higher dimensional predictor space, and I am having a hard time getting the plot() function to show what I expect: on any pair of axes, there is no apparent separatrix. Perhaps the visualization isn't intended for a complex case like this, or perhaps the separatrix is a goofy shape that doesn't show up well in any of these 2d projections — though really the linear one should show up. However, the prediction accuracy is pretty good and many observations are picked as support vectors, so it's confusing. I looked at PrinceHonest's solution and it just has a mistake — he left in mpg as a predictor for [mpg > median(mpg)], which obviously works with great accuracy!

I did linear, quadratic, and radial kernels. The best models had increasing accuracy in that order: 91%, 98%, 99%.

My solution:

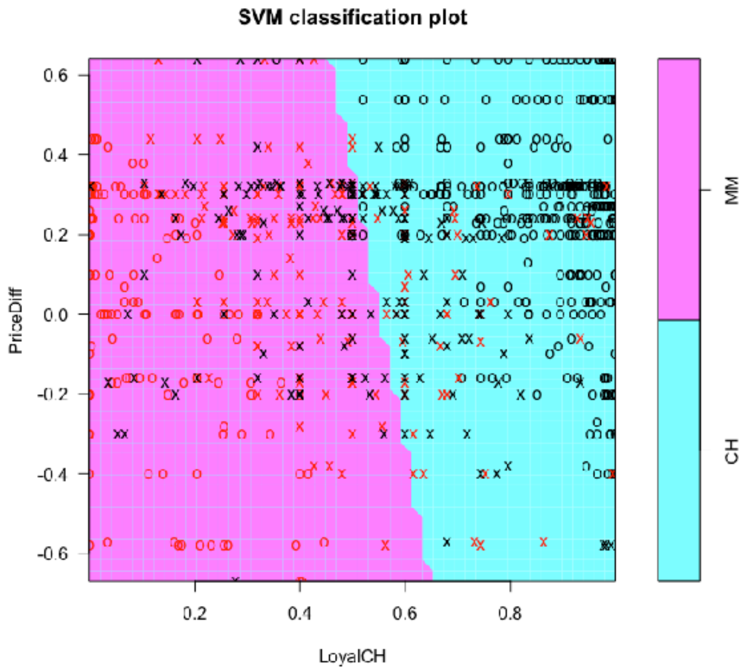
```
library(ISLR)
library(e1071)
good <- as.factor(Auto$mpg > median(Auto$mpg))
Auto.copy <- Auto
set.seed(7)
cost.list <- 10^seq(-3, 5)
Auto.copy[,'mpg'] <- good
tune.lin <- tune(svm, mpg~., data=Auto.copy, kernel='linear',
                 ranges = list(cost = cost.list))
print(summary(tune.lin))
print(summary(tune.lin$best.model))
pred.lin <- predict(tune.lin$best.model, Auto.copy)
print("Accuracy for best linear model:")
print(summary(pred.lin == Auto.copy[,'mpg']))
tune.quad <- tune(svm, mpg~., data=Auto.copy, kernel='polynomial',
             ranges = list(cost = cost.list,
                        degree=2))
print(summary(tune.quad))
print(summary(tune.quad$best.model))
pred.quad <- predict(tune.quad$best.model, Auto.copy)
print("Accuracy for best quadratic model:")
print(summary(pred.quad == Auto.copy[,'mpg']))
tune.rad <- tune(svm, mpg~., data=Auto.copy, kernel='radial',
              ranges = list(cost = cost.list,
                         gamma=c(0.1,0.3,1,3,10)))
print(summary(tune.rad))
print(summary(tune.rad$best.model))
pred.rad <- predict(tune.rad$best.model, Auto.copy)
print("Accuracy for best radial model:")
print(summary(pred.rad == Auto.copy[,’mpg']))
```

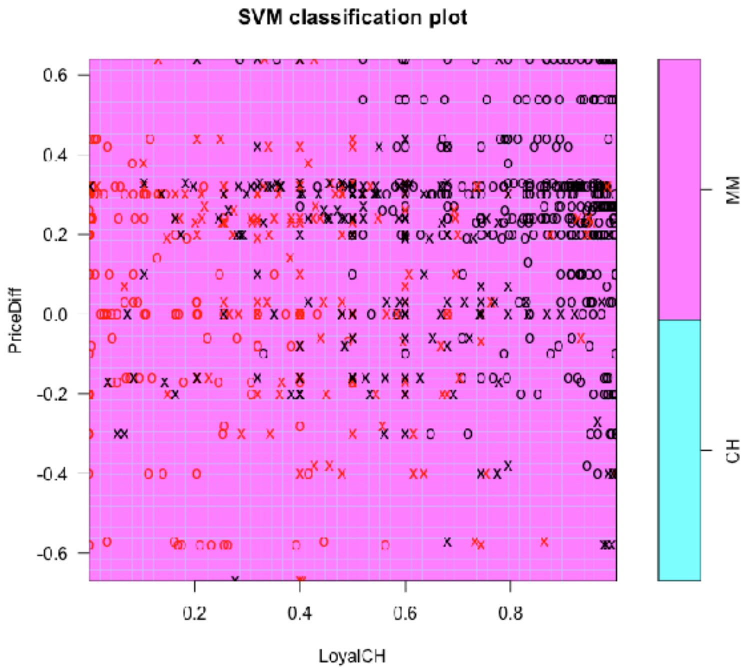
Results:

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.09160256

- Detailed performance results:
   cost      error dispersion
1 1e-03 0.14544872 0.07048980
2 1e-02 0.09160256 0.04454547
3 1e-01 0.09160256 0.04921998
4 1e+00 0.09666667 0.04376306
5 1e+01 0.10185897 0.05055255
6 1e+02 0.12737179 0.06893074
7 1e+03 0.12487179 0.06406110
8 1e+04 0.12487179 0.05932464
9 1e+05 0.12487179 0.05932464


Call:
best.tune(method = svm, train.x = mpg ~ ., data = Auto.copy, ranges = list(cost = cost.list),
    kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01
      gamma:  0.003215434
```

```
Number of Support Vectors:  168

 ( 84 84 )


Number of Classes:  2

Levels: 
 FALSE TRUE



[1] "Accuracy for best linear model:"
   Mode   FALSE    TRUE 
logical      34     358 

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation 

- best parameters:
  cost degree
 10000      2

- best performance: 0.1530769 

- Detailed performance results:
   cost degree     error dispersion
1 1e-03      2 0.5842949 0.04194489
2 1e-02      2 0.5842949 0.04194489
3 1e-01      2 0.5842949 0.04194489
4 1e+00      2 0.5842949 0.04194489
5 1e+01      2 0.5738462 0.03573018
6 1e+02      2 0.3114744 0.09940765
7 1e+03      2 0.2756410 0.05946240
8 1e+04      2 0.1530769 0.05868706
9 1e+05      2 0.1912179 0.07669128


Call:
best.tune(method = svm, train.x = mpg ~ ., data = Auto.copy, ranges = list(cost = cost.list, 
    degree = 2), kernel = "polynomial")


Parameters:
   SVM-Type:  C-classification 
 SVM-Kernel:  polynomial 
       cost:  10000 
     degree:  2 
      gamma:  0.003215434 
     coef.0:  0 

Number of Support Vectors:  300
```

```
 ( 160 140 )


Number of Classes:  2

Levels: 
 FALSE TRUE



[1] "Accuracy for best quadratic model:"
   Mode   FALSE    TRUE 
logical       8     384 

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation 

- best parameters:
 cost gamma
   10   0.1

- best performance: 0.07647436 

- Detailed performance results:
    cost gamma      error dispersion
1  1e-03   0.1 0.57141026 0.04147633
2  1e-02   0.1 0.27333333 0.08641581
3  1e-01   0.1 0.08679487 0.04047913
4  1e+00   0.1 0.08679487 0.04557267
5  1e+01   0.1 0.07647436 0.04459218
6  1e+02   0.1 0.10724359 0.07214002
7  1e+03   0.1 0.10461538 0.06867509
8  1e+04   0.1 0.10461538 0.06867509
9  1e+05   0.1 0.10461538 0.06867509
10 1e-03   0.3 0.57141026 0.04147633
11 1e-02   0.3 0.57141026 0.04147633
12 1e-01   0.3 0.08935897 0.04059627
13 1e+00   0.3 0.08423077 0.04018026
14 1e+01   0.3 0.09442308 0.06371109
15 1e+02   0.3 0.09698718 0.06109770
16 1e+03   0.3 0.09698718 0.06109770
17 1e+04   0.3 0.09698718 0.06109770
18 1e+05   0.3 0.09698718 0.06109770
19 1e-03   1.0 0.57141026 0.04147633
20 1e-02   1.0 0.57141026 0.04147633
21 1e-01   1.0 0.57141026 0.04147633
22 1e+00   1.0 0.07916667 0.04557973
23 1e+01   1.0 0.08429487 0.05518553
24 1e+02   1.0 0.08429487 0.05518553
25 1e+03   1.0 0.08429487 0.05518553
26 1e+04   1.0 0.08429487 0.05518553
27 1e+05   1.0 0.08429487 0.05518553
28 1e-03   3.0 0.57141026 0.04147633
```





```
29 1e-02   3.0 0.57141026 0.04147633
30 1e-01   3.0 0.57141026 0.04147633
31 1e+00   3.0 0.42826923 0.11361111
32 1e+01   3.0 0.38730769 0.13587889
33 1e+02   3.0 0.38730769 0.13587889
34 1e+03   3.0 0.38730769 0.13587889
35 1e+04   3.0 0.38730769 0.13587889
36 1e+05   3.0 0.38730769 0.13587889
37 1e-03  10.0 0.57141026 0.04147633
38 1e-02  10.0 0.57141026 0.04147633
39 1e-01  10.0 0.57141026 0.04147633
40 1e+00  10.0 0.52538462 0.05784500
41 1e+01  10.0 0.52282051 0.06144580
42 1e+02  10.0 0.52282051 0.06144580
43 1e+03  10.0 0.52282051 0.06144580
44 1e+04  10.0 0.52282051 0.06144580
45 1e+05  10.0 0.52282051 0.06144580


Call:
best.tune(method = svm, train.x = mpg ~ ., data = Auto.copy, ranges = list(cost = cost.list,
    gamma = c(0.1, 0.3, 1, 3, 10)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.1

Number of Support Vectors:  130

 ( 61 69 )


Number of Classes:  2
```

```
Levels:
 FALSE TRUE
```

```
[1] "Accuracy for best radial model:"
   Mode   FALSE    TRUE
logical       4     388
```

9.7.8: Back to OJ: fit SVC and SVM with quadratic or radial kernels. Best performance in test error rate is the linear SVC, which suggests that the other kernels were overfitting? I didn't have the plotting fault that I got in the previous problem, at least not completely, but in this case I recalled from the work with decision trees that two very important variables were PriceDiff and LoyalCH, and the separatrix was quite clear for this projection, in the SVC. In the radial and quadratic SVM it was not visible.

Code:

```
library(ISLR)
library(e1071)
set.seed(1)
train <- sample(1:nrow(OJ), 800)
fit.lin <- svm(Purchase ~ ., data=OJ[train,], kernel='linear', cost=0.01)
print(summary(fit.lin))
pred.lin.train <- predict(fit.lin, OJ[train,])
pred.lin.test <- predict(fit.lin, OJ[-train,])
err.train <- 1-mean(pred.lin.train == OJ[train, "Purchase"])
err.test <- 1-mean(pred.lin.test == OJ[-train, "Purchase"])
print(paste0("Training error for SVC: ", err.train))
print(paste0("Test error for SVC: ", err.test))
tune.lin <- tune(svm, Purchase ~ ., data=OJ[train,], kernel='linear',
               ranges=list(
                   cost=c(0.01, 0.03, 0.1, 0.3, 1, 3, 10)
               ))
print(summary(tune.lin))
pred.lin.train <- predict(tune.lin$best.model, OJ[train,])
pred.lin.test <- predict(tune.lin$best.model, OJ[-train,])
err.train <- 1-mean(pred.lin.train == OJ[train, "Purchase"])
err.test <- 1-mean(pred.lin.test == OJ[-train, "Purchase"])
print(paste0("Training error for tuned SVC: ", err.train))
print(paste0("Test error for tuned SVC: ", err.test))
tune.quad <- tune(svm, Purchase ~ ., data=OJ[train,], kernel='polynomial',
               ranges=list(
                   cost=c(0.01, 0.03, 0.1, 0.3, 1, 3, 10),
                   degree=2
               ))
print(summary(tune.quad))
pred.train <- predict(tune.quad$best.model, OJ[train,])
pred.test <- predict(tune.quad$best.model, OJ[-train,])
err.train <- 1-mean(pred.train == OJ[train, "Purchase"])
err.test <- 1-mean(pred.test == OJ[-train, "Purchase"])
print(paste0("Training error for tuned quad SVM: ", err.train))
print(paste0("Test error for tuned quad SVM: ", err.test))
```

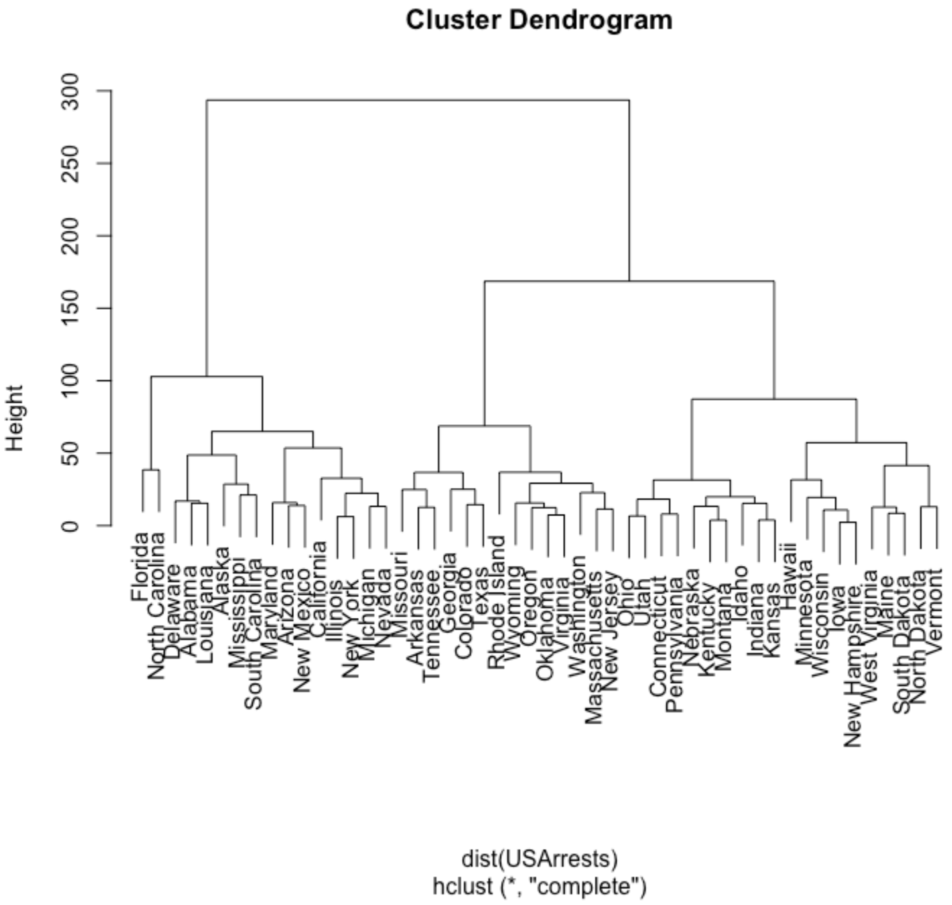
```
tune.rad <- tune(svm, Purchase ~ ., data=OJ[train,], kernel='radial',
                 ranges=list(
                     cost=c(0.01, 0.03, 0.1, 0.3, 1, 3, 10)
                 ))
print(summary(tune.rad))
pred.train <- predict(tune.rad$best.model, OJ[train,])
pred.test <- predict(tune.rad$best.model, OJ[-train,])
err.train <- 1-mean(pred.train == OJ[train, "Purchase"])
err.test <- 1-mean(pred.test == OJ[-train, "Purchase"])
print(paste0("Training error for tuned radial SVM: ", err.train))
print(paste0("Test error for tuned radial SVM: ", err.test))
```

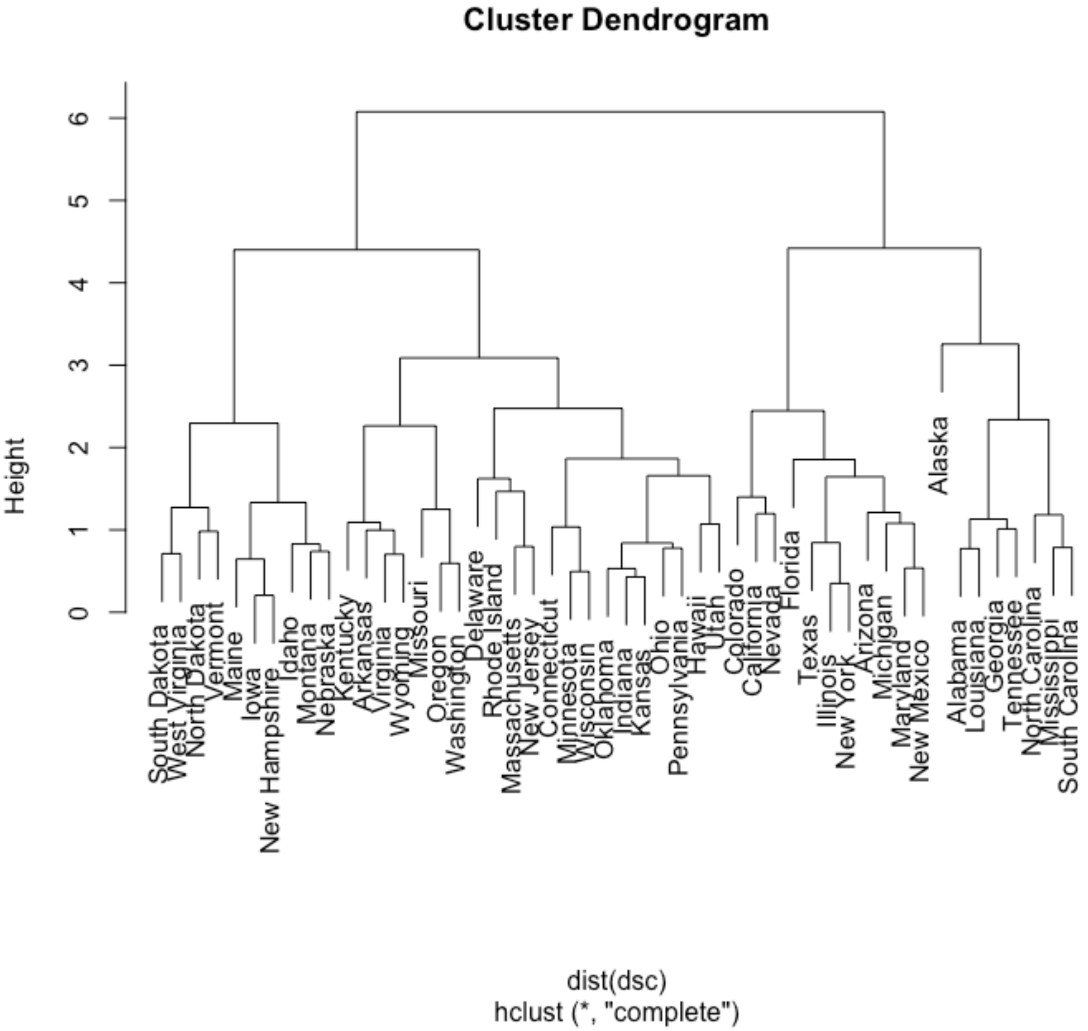
Results:

```
Call:
svm(formula = Purchase ~ ., data = OJ[train, ], kernel = "linear", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01
      gamma:  0.05555556

Number of Support Vectors:  432

 ( 215 217 )


Number of Classes:  2

Levels:
 CH MM



[1] "Training error for SVC: 0.16625"
[1] "Test error for SVC: 0.181481481481481"

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.3

- best performance: 0.16125

- Detailed performance results:
   cost   error dispersion
1  0.01 0.16625 0.05138701
2  0.03 0.16375 0.05084358
3  0.10 0.16250 0.04894725
```

```
4  0.30 0.16125 0.05415064
5  1.00 0.16875 0.04723243
6  3.00 0.16375 0.04803428
7 10.00 0.16500 0.04993051

[1] "Training error for tuned SVC: 0.16125"
[1] "Test error for tuned SVC: 0.181481481481481"

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
   10      2

- best performance: 0.18125

- Detailed performance results:
   cost degree   error dispersion
1  0.01      2 0.38250 0.06800735
2  0.03      2 0.36000 0.06917169
3  0.10      2 0.32625 0.05415064
4  0.30      2 0.21750 0.04297932
5  1.00      2 0.19750 0.04923018
6  3.00      2 0.18875 0.04427267
7 10.00      2 0.18125 0.03784563

[1] "Training error for tuned quad SVM: 0.145"
[1] "Test error for tuned quad SVM: 0.185185185185185"

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.15875

- Detailed performance results:
   cost   error dispersion
1  0.01 0.38250 0.03872983
2  0.03 0.37875 0.04291869
3  0.10 0.17375 0.03928617
4  0.30 0.17125 0.03387579
5  1.00 0.16125 0.03884174
6  3.00 0.16875 0.04611655
7 10.00 0.15875 0.04931827

[1] "Training error for tuned radial SVM: 0.13875"
[1] "Test error for tuned radial SVM: 0.188888888888889”
```

10.7.1:
proof,


Cluster Dendrogram
Height
0
50
100
150
200
250
300
Florida
North Carolina
Delaware
Alabama
Louisiana
Alaska
Mississippi
South Carolina
Maryland
Arizona
New Mexico
California
Illinois
New York
Michigan
Nevada
Missouri
Arkansas
Tennessee
Georgia
Colorado
Texas
Rhode Island
Wyoming
Oregon
Oklahoma
Virginia
Washington
Massachusetts
New Jersey
Ohio
Utah
Connecticut
Pennsylvania
Nebraska
Kentucky
Montana
Idaho
Indiana
Kansas
Hawaii
Minnesota
Wisconsin
Iowa
New Hampshire
West Virginia
Maine
South Dakota
North Dakota
Vermont
dist(USArrests)
hclust (*, "complete")


Cluster Dendrogram
Height
0
1
2
3
4
5
6
South Dakota
West Virginia
North Dakota
Vermont
Maine
Iowa
New Hampshire
Idaho
Montana
Nebraska
Kentucky
Arkansas
Virginia
Wyoming
Missouri
Oregon
Washington
Delaware
Rhode Island
Massachusetts
New Jersey
Connecticut
Minnesota
Wisconsin
Oklahoma
Indiana
Kansas
Ohio
Pennsylvania
Hawaii
Utah
Colorado
California
Nevada
Florida
Texas
Illinois
New York
Arizona
Michigan
Maryland
New Mexico
Alaska
Alabama
Louisiana
Georgia
Tennessee
North Carolina
Mississippi
South Carolina
dist(dsc)
hclust (*, "complete")

skipped.

10.7.2: Manual hierarchical clustering problem, sketched it.

10.7.3: Manual K-means, skipped

10.7.4:

A) Single linkage fusion will never be later than complete linkage fusion, but they could be simultaneous or single could be earlier.
B) Fusion will occur at the same height because single-point clusters aren't affected by choice of linkage.

10.7.5: Three clustering results from 3 different choices of scaling:

* Compute by items bought: we just stack the two bar charts. Probably clusters are {6-7 items bought} and {8+ items bought}, with 4 observations each.
* Standardize the variables: it probably ends up being {bought computers} and {didn't buy computers}, because the variability in socks is less, but I suppose the sock outlier could join the computer cluster depending on how the math worked out.
* Compute by dollars spent: computers vs not, no question.

10.7.6:

A) If the first PC "explains 10% of the variance", it means that, when the data are projected onto the 1PC axis, the variance remaining is 10% of the total starting variance.
B) The proposed idea amounts to projecting the data into the space orthogonal to the first PC. The intention is surely to remove the machine-identity effect so the treatment/control effect is stronger. But a better a approach would be: just include the identity of machine for each test as its own factor!
C) Do a simulation to show the idea: skipped…

10.7.7: I can't get this one to come out… PrinceHonest didn't understand this one correctly and his answer doesn't show what is required.

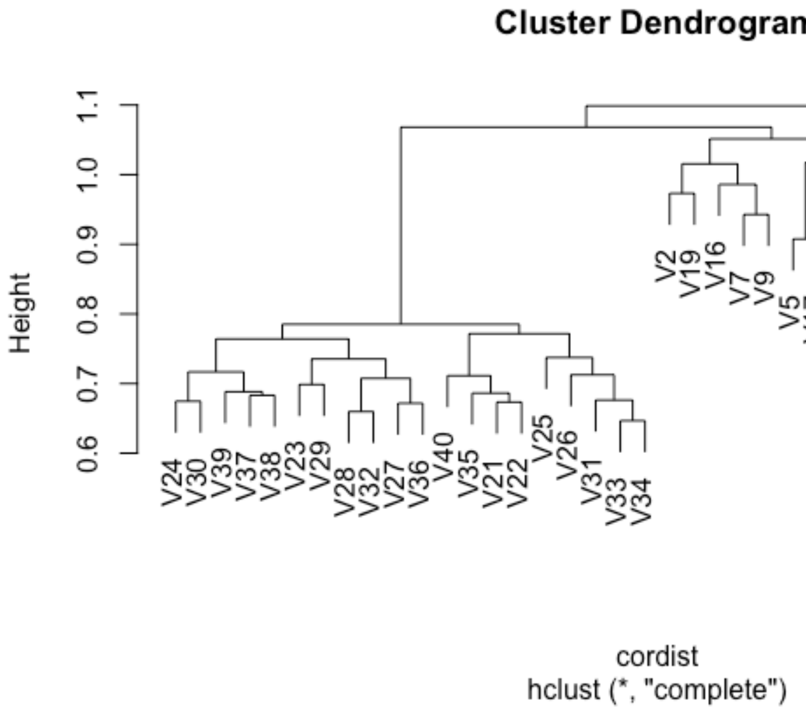
10.7.8: boring and obvious

10.7.9: Should the data be scaled before clustering? It depends on what you want to know by clustering. The number of arrests is already normalized to population, so it would be reasonable to leave those numbers alone if you want to know about "high-crime" states. The %urban population is also normalized by its nature, though it is incommensurate with the arrest numbers. I don't see a great advantage to scaling here, but leaving out the UrbanPop before clustering might be useful.

```
hc.complete <- hclust(dist(USArrests), method='complete')
plot(hc.complete)
print(cutree(hc.complete, 3))

dsc <- scale(USArrests)
hc.scaled <- hclust(dist(dsc), method='complete')
plot(hc.scaled)
print(cutree(hc.scaled, 3))
```

Plots:

```
       Alabama         Alaska        Arizona       Arkansas     California       Colorado
             1              1              1              2              1              2
   Connecticut       Delaware        Florida        Georgia         Hawaii          Idaho
             3              1              1              2              3              3
      Illinois        Indiana           Iowa         Kansas       Kentucky      Louisiana
             1              3              3              3              3              1
         Maine       Maryland  Massachusetts       Michigan      Minnesota    Mississippi
             3              1              2              1              3              1
      Missouri        Montana       Nebraska         Nevada  New Hampshire     New Jersey
             2              3              3              1              3              2
    New Mexico       New York North Carolina   North Dakota           Ohio       Oklahoma
             1              1              1              3              3              2
        Oregon   Pennsylvania   Rhode Island South Carolina   South Dakota      Tennessee
             2              3              2              1              3              2
         Texas           Utah        Vermont       Virginia     Washington  West Virginia
             2              3              3              2              2              3
     Wisconsin        Wyoming
             3              2
       Alabama         Alaska        Arizona       Arkansas     California       Colorado
             1              1              2              3              2              2
   Connecticut       Delaware        Florida        Georgia         Hawaii          Idaho
             3              3              2              1              3              3
      Illinois        Indiana           Iowa         Kansas       Kentucky      Louisiana
             2              3              3              3              3              1
         Maine       Maryland  Massachusetts       Michigan      Minnesota    Mississippi
             3              2              3              2              3              1
      Missouri        Montana       Nebraska         Nevada  New Hampshire     New Jersey
             3              3              3              2              3              3
    New Mexico       New York North Carolina   North Dakota           Ohio       Oklahoma
             2              2              1              3              3              3
        Oregon   Pennsylvania   Rhode Island South Carolina   South Dakota      Tennessee
             3              3              3              1              3              1
         Texas           Utah        Vermont       Virginia     Washington  West Virginia
             2              3              3              3              3              3
     Wisconsin        Wyoming
             3              3
```

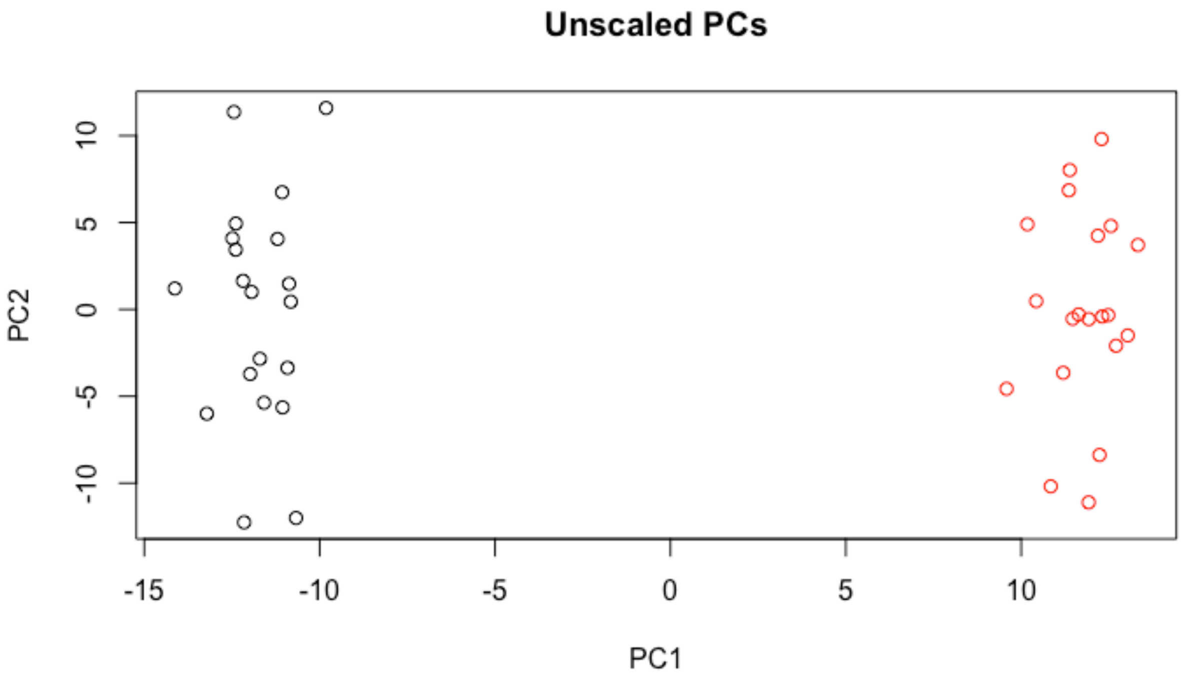


```
> table(cutree(hc.complete, 3))

 1  2  3 
16 14 20 
```

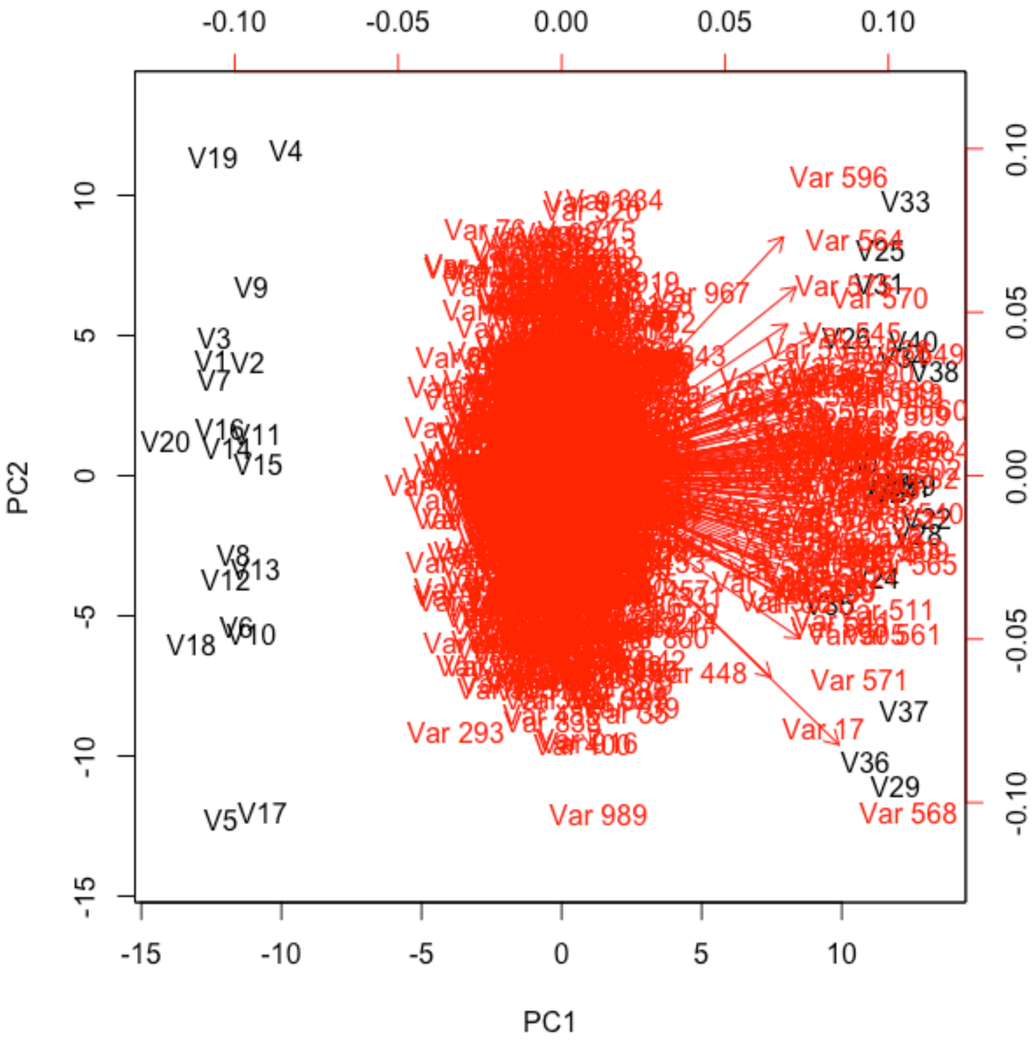
10.7.10: Simulated data with PCA and KMC.

Code:

```
set.seed(1)
par(mfrow=c(2,3))
x <- matrix(runif(60*50), ncol=50)
x[1:20,1:25] <- x[1:20, 1:25f] - 0.25
x[21:40,26:50] <- x[21:40,26:50] + 0.35      # Second PC is relevant too
y <- c(rep(1,20), rep(2, 20), rep(3,20))
pr.out <- prcomp(x)
# print(summary(pr.out))
plot(pr.out$x[,1:2], col=y, main='Part B: PCs 1,2')
km.out <- kmeans(x, 3, nstart=20)
print("Part C: 3 clusters")
print(table(km.out$cluster, y))
plot(pr.out$x[,1:2], col=km.out$cluster, main='Part C: 3 clusters')
km.2 <- kmeans(x, 2, nstart=20)
print("Part D: 2 clusters")
plot(pr.out$x[,1:2], col=km.2$cluster, main='Part D: 2 clusters')
print(table(km.2$cluster, y))
```



```
km.4 <- kmeans(x,4,nstart=20)
print('Part E: 4 clusters')
print(table(km.4$cluster, y))
plot(pr.out$x[,1:2], col=km.4$cluster, main='Part E: 4 clusters, PCs 1,2')
km.pc <- kmeans(pr.out$x[,1:2], 3, nstart=20)
print("Part F: 3 clusters in PC[1:2] space")
```

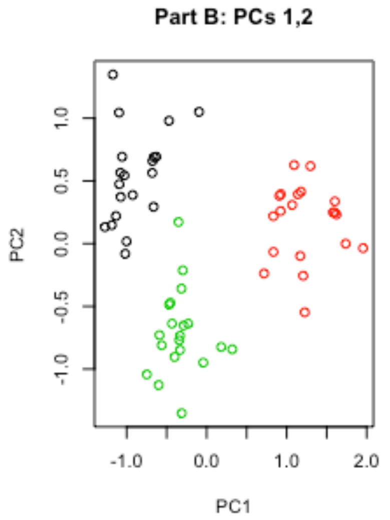


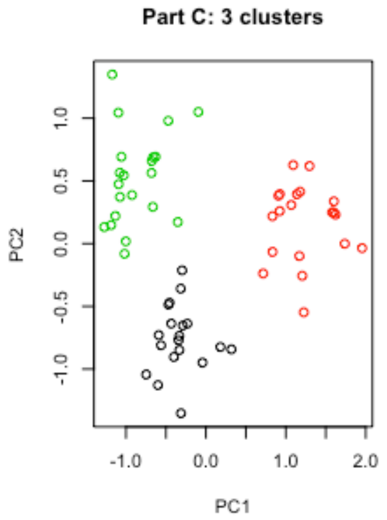
```
print(table(km.pc$cluster, y))
plot(pr.out$x[,1:2], col=km.pc$cluster, main='Part F: 3 clusters in PC[1:2] space')
x.sc <- scale(x)
km.sc <- kmeans(x.sc, 3, nstart=20)
print('Part G: 3 clusters after scaling')
print(table(km.sc$cluster, y))
plot(pr.out$x[,1:2], col=km.sc$cluster, main='Part G: 3 clusters after scaling')
```

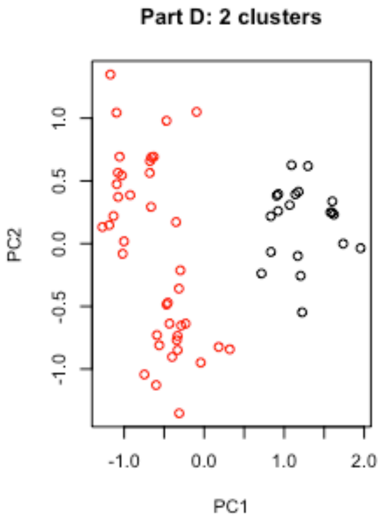
Plots:

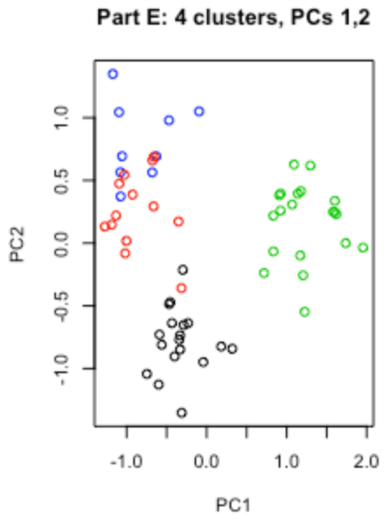
Comments: I dialed in the shifts to afford a bit of confusion, but not too much. By shifting the classes in different directions, I gave the second PC some action. Interesting that there is still barely enough overlap to miscluster one observation in all the 3-cluster approaches, which all come out the same. If asking for 2 clusters, it just merges two of the real classes. If asking for 4 clusters, it mostly just splits one of the real classes.

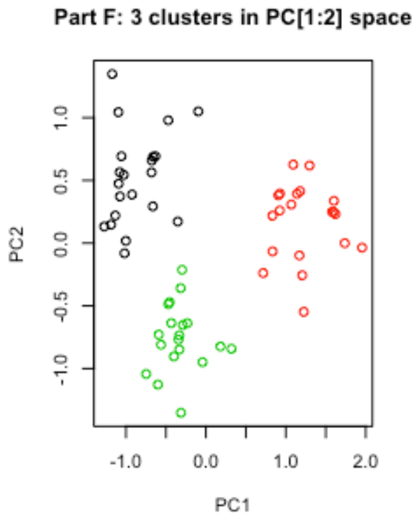
10.7.11: Gene expression data.

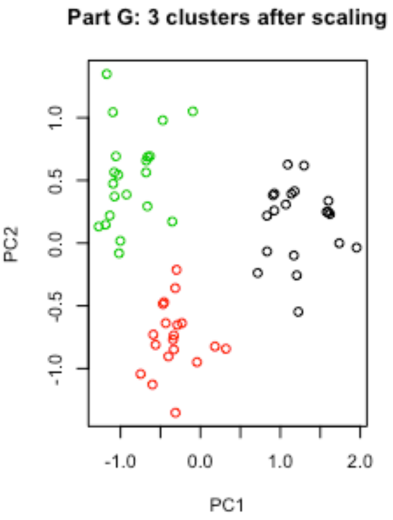










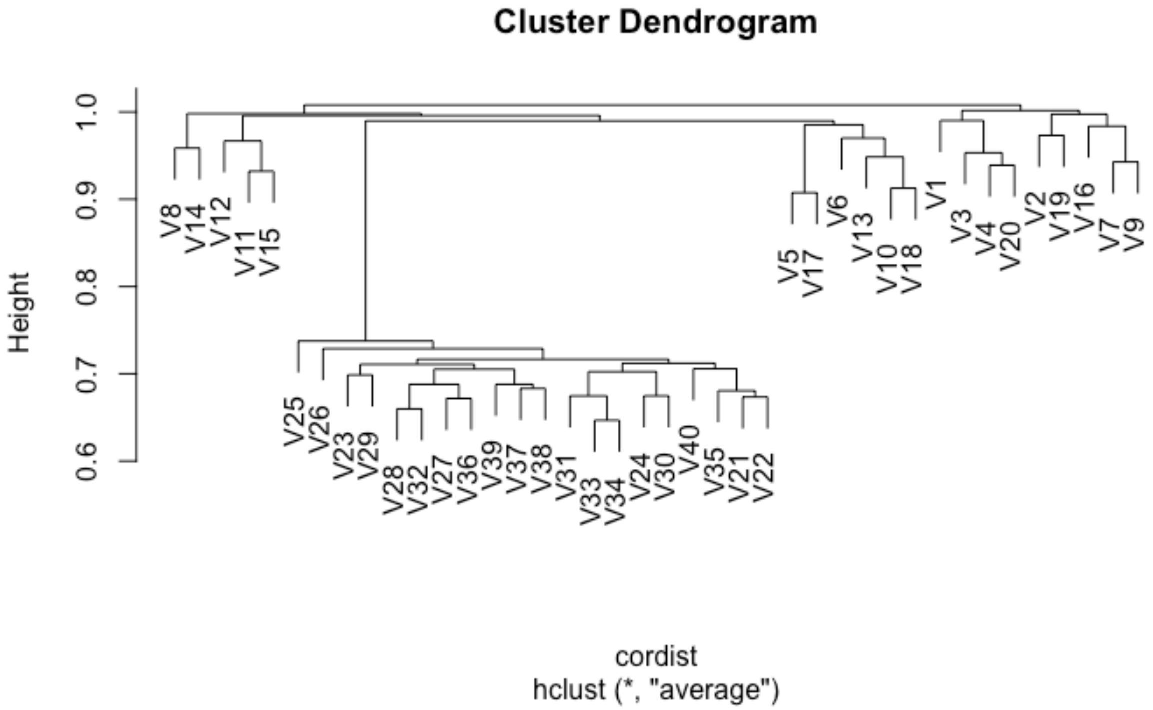


Code:

```
setwd("~/Documents/ISL_book")
dat <- read.csv('Ch10Ex11.csv', header=F)
print(dim(dat))                              # 1000 rows, 40 col
cordist <- as.dist(1-cor(dat))
hc.complete <- hclust(cordist, method='complete')
plot(hc.complete)
hc.average <- hclust(cordist, method='average')
plot(hc.average)
pr.out <- prcomp(t(dat), scale=F)
pr.scale <- prcomp(t(dat), scale=T)
plot(pr.out$x[,1:2], main='Unscaled PCs', col=c(rep(1,20), rep(2,20)))
plot(pr.scale$x[,1:2], main='Scaled PCs', col=c(rep(1,20), rep(2,20)))
biplot(pr.out, scale=0)
```

Plots:

All the “diseased” samples, V21-V40, cluster early, and the “normal” samples cluster late. Pretty clear-cut. To determine which genes are most different across the two groups, we can do PCA and pick those genes with highest loading into the first PC. That’s accessible through the rotation matrix, but it’s annoying to extract. The first PC does indeed select the two correct groups, and the loading biplot shows which genes have high loading in the first PC, although it’s not very readable…



You can read a few gene indices…