STATS 202: Data Mining and Analysis

Instructor: Linh Tran

HOMEWORK # 2

Due date: July 16, 2021

Stanford University

Introduction

Homework problems are selected from the course textbook: *An Introduction to Statistical Learning*.

Problem 1 (5 points)

Chapter 4, Exercise 1 (p. 168).

Proof. Given $p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$, we see that,

$$1 - p(X) = 1 - \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{1}$$

$$= \frac{1 + e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} - \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{2}$$

$$= \frac{1}{1 + e^{\beta_0+\beta_1 X}} \tag{3}$$

Thus,

$$\frac{p(X)}{1 - p(X)} = \frac{\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}}{\frac{1}{1+e^{\beta_0+\beta_1 X}}} \tag{4}$$

$$= e^{\beta_0+\beta_1 X} \tag{5}$$

□

Problem 2 (5 points)

Chapter 4, Exercise 4 (p. 168).

(a) Since X is uniformly distributed on $[0, 1]$, the probability that a given observation will be used to make the prediction is 10%. Thus, when $p = 1$, on average 10% of the available observations will be used to make the prediction.

(b) When $p = 2$, the probability that a given observation will be used to make the prediction is $0.1 * 0.1 = 0.01$. Thus, on average, 1% of the available observations will be used to make the prediction.

(c) For the same reason, when $p = 100$, on average a 0.1^{100} proportion of the available observations will be used to make the prediction.

(d) When p is large, 0.1^p will be very small. So there are very few training observations "near" any given test observation.

(e) For $p = 1$, the length of each side of the hypercube is 0.1. For $p = 2$, the length of each side of the hypercube is $\sqrt{0.1}$. For $p = 100$, the length of each side is $0.1^{1/100}$. Generally, for p dimensional cases, the length of each side of the hypercube is $0.1^{1/p}$. When p is large, the length of each side will be close to 1, indicating that we need to use nearly the entire feature space for our prediction. This illustrates that the non-parametric approaches often perform poorly when p is large.

Problem 3 (5 points)

Chapter 4, Exercise 6 (p. 170).

(a) The estimated probability is

$$\begin{aligned}\hat{\mathbb{P}}(Y|X = x) &= \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X_1+\hat{\beta}_2 X_2}}{1 + e^{\hat{\beta}_0+\hat{\beta}_1 X_1+\hat{\beta}_2 X_2}} \\ &= \frac{e^{-6+0.05*(40)+1*(3.5)}}{1 + e^{-6+0.05*(40)+1*(3.5)}} \\ &\approx 0.3775\end{aligned} \tag{6}$$

(b) Suppose the student needs x_1 hours to have a 50% changes of getting an A in the class. Then

$$0.5 = \frac{e^{-6+0.05*(x_1)+1*(3.5)}}{1 + e^{-6+0.05*(x_1)+1*(3.5)}} \tag{7}$$

Solving the equation for x_1 we get $x_1 = 50$, indicating that the student needs to study for 50 hours to have a 50% chances of getting an A in the class.

Problem 4 (5 points)

Chapter 4, Exercise 8 (p. 170).

Notice that for 1-nearest neighbors, the error on the training set would be exactly 0 (since the closest point to every training point is itself). Thus, the error on the test set is 36% for 1-nearest neighbors, which is higher than the logistic regression. Our suggestion should be based on performance on the test set. So we prefer to use logistic regression for classification of new observations.

Problem 5 (5 points)

Chapter 4, Exercise 10 (p. 171). In part (i), please be concise; only describe and provide the output of your best prediction.

(a) The correlations between the lag variables and today's return are close to zero. In other words, there appears to be little correlation between today's returns and the previous day's returns. The only substantial correlation is between `Year` and `Volume`. By plotting the data we see that Volume is increasing over time. In other words, the average number of shares traded daily generally increased from 1990 to 2010.

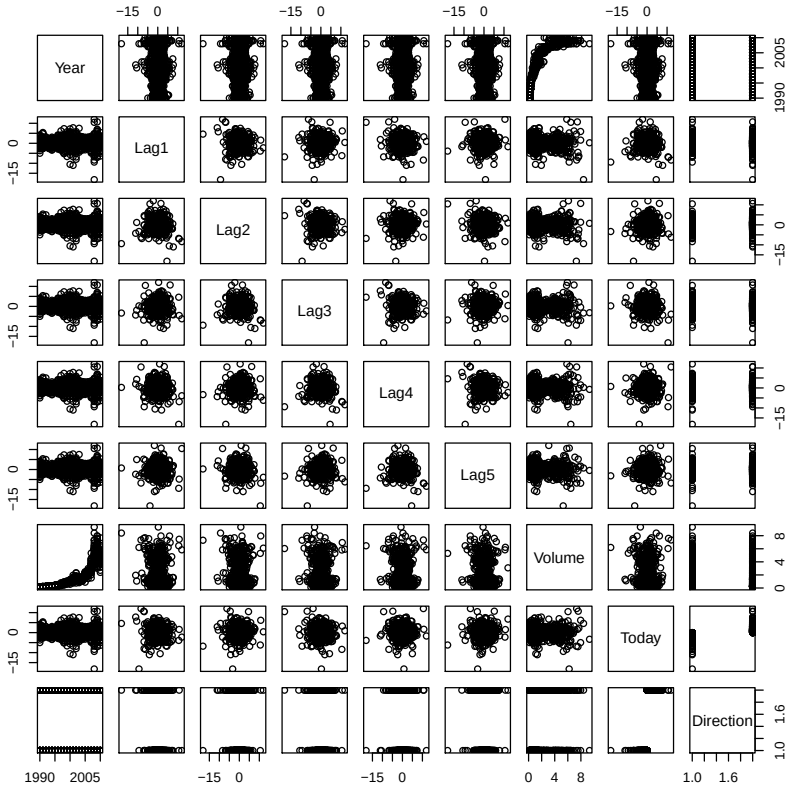
```
  library(ISLR)
  data(Weekly)
  WeeklyNumerical <- Weekly
  WeeklyNumerical$Direction <- as.integer(WeeklyNumerical$Direction == 'Up')
  cor(WeeklyNumerical)
```

```
##                 Year         Lag1        Lag2        Lag3         Lag4
## Year       1.00000000 -0.032289274 -0.03339001 -0.03000649 -0.031127923
```

```
## Lag1       -0.03228927  1.000000000 -0.07485305  0.05863568 -0.071273876
## Lag2       -0.03339001 -0.074853051  1.00000000 -0.07572091  0.058381535
## Lag3       -0.03000649  0.058635682 -0.07572091  1.00000000 -0.075395865
## Lag4       -0.03112792 -0.071273876  0.05838153 -0.07539587  1.000000000
## Lag5       -0.03051910 -0.008183096 -0.07249948  0.06065717 -0.075675027
## Volume      0.84194162 -0.064951313 -0.08551314 -0.06928771 -0.061074617
## Today      -0.03245989 -0.075031842  0.05916672 -0.07124364 -0.007825873
## Direction -0.02220025 -0.050003804  0.07269634 -0.02291281 -0.020549456
##                    Lag5      Volume        Today   Direction
## Year       -0.030519101  0.84194162 -0.032459894 -0.02220025
## Lag1       -0.008183096 -0.06495131 -0.075031842 -0.05000380
## Lag2       -0.072499482 -0.08551314  0.059166717  0.07269634
## Lag3        0.060657175 -0.06928771 -0.071243639 -0.02291281
## Lag4       -0.075675027 -0.06107462 -0.007825873 -0.02054946
## Lag5        1.000000000 -0.05851741  0.011012698 -0.01816827
## Volume     -0.058517414  1.00000000 -0.033077783 -0.01799521
## Today       0.011012698 -0.03307778  1.000000000  0.72002470
## Direction -0.018168272 -0.01799521  0.720024704  1.00000000
```

We additionally see a strong trend between `Year` and `Volume` in our pairwise scatter plots.

```
pairs(Weekly)
```



(b) Only `Lag2` appears to be statistically significant.

```
glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               family=binomial, data=Weekly)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Weekly)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6949  -1.2565   0.9913   1.0849   1.4579
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.26686    0.08593   3.106   0.0019 **
## Lag1        -0.04127    0.02641  -1.563   0.1181
## Lag2         0.05844    0.02686   2.175   0.0296 *
## Lag3        -0.01606    0.02666  -0.602   0.5469
## Lag4        -0.02779    0.02646  -1.050   0.2937
## Lag5        -0.01447    0.02638  -0.549   0.5833
## Volume      -0.02274    0.03690  -0.616   0.5377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1486.4  on 1082  degrees of freedom
## AIC: 1500.4
##
## Number of Fisher Scoring iterations: 4
```

(c) Our model correctly predicted that the market would go up on 557 weeks and that it would go down on 54 weeks. In this case, logistic regression correctly predicted the movement of the market 56.1% of the time. We can see that the logistic regression predicts a lot of "up" when the `Direction` is actually down.

```
glm.probs <- predict(glm.fit, type="response")
glm.pred <- rep("Down", length(glm.probs))
glm.pred[glm.probs > 0.5] = "Up"
table(glm.pred, Weekly$Direction)
```

```
##
## glm.pred Down  Up
##     Down   54  48
##     Up    430 557
```

```
# Proportion of times classifier is correct
mean(glm.pred == Weekly$Direction)
```

```
## [1] 0.5610652
```

(d) The overall fraction of correct predictions for the held out data is 62.5%.

```
train.idx <- (Weekly$Year <= 2008)
Weekly.train <- Weekly[train.idx,]
Weekly.test <- Weekly[!train.idx,]
```

```
  glm.fit.2 <- glm(Direction ~ Lag2, data=Weekly.train, family=binomial)

  glm.probs.2 <- predict(glm.fit.2, Weekly.test, type="response")
  glm.pred.2 <- rep("Down", sum(!train.idx))
  glm.pred.2[glm.probs.2 > 0.5] = "Up"
  table(glm.pred.2, Weekly.test$Direction)
```

```
##
## glm.pred.2 Down Up
##       Down    9  5
##       Up     34 56
```

```
  # Proportion of times classifier is correct
  mean(glm.pred.2 == Weekly.test$Direction)
```

```
## [1] 0.625
```

(e) The overall fraction of correct predictions for the held out data is 62.5%.

```
  library(MASS)
```

```
##
## Attaching package:  'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
```

```
  lda.fit <- lda(Direction ~ Lag2, data=Weekly.train)
  lda.pred <- predict(lda.fit, Weekly.test)
  table(lda.pred$class, Weekly.test$Direction)
```

```
##
##         Down Up
##   Down    9  5
##   Up     34 56
```

```
  # Proportion of times classifier is correct
  mean(lda.pred$class == Weekly.test$Direction)
```

```
## [1] 0.625
```

(f) The overall fraction of correct predictions for the held out data is 58.7%.

```
  qda.fit <- qda(Direction ~ Lag2, data=Weekly.train)
  qda.pred <- predict(lda.fit, Weekly.test)
  table(qda.pred$class, Weekly.test$Direction)
```

```
##
##         Down Up
##   Down    9  5
##   Up     34 56
```

```
  # Proportion of times classifier is correct
  mean(qda.pred$class == Weekly.test$Direction)
```

```
## [1] 0.625
```

(g) The overall fraction of correct predictions for the held out data is 50.0%.

```
  library(class)
  set.seed(1)
  X.train <- as.matrix(Weekly.train$Lag2)
  X.test <- as.matrix(Weekly.test$Lag2)
  knn.pred <- knn(X.train, X.test, Weekly.train$Direction, k=1)
  table(knn.pred, Weekly.test$Direction)

##
## knn.pred Down Up
##     Down   21 30
##     Up     22 31

  # Proportion of times classifier is correct
  mean(knn.pred == Weekly.test$Direction)

## [1] 0.5
```

(h) Logistic regression and LDA give equally best results on this data.

Problem 6 (5 points)

Chapter 5, Exercise 2 (p. 197).

(a) Each observation is drawn independently, with uniform probablity. There are n observations that we can draw from. Thus, the probability that any of j observations are drawn is $1/n$. Therefore, the probability that the first observation drawn is *not* the j^{th} observation is $1 - 1/n$.

(b) As stated above, each observation is drawn independently. Sampling is done with replacement, meaning that the probability for the second bootstrap is the same as the probability for the first bootstrap. Therefore, the probability that the second observation drawn is *not* the j^{th} observation is (still) $1 - 1/n$.

(c) For each bootstrap sample, you are drawing n observations from the original set with replacement. Each time, you have a $1 - 1/n$ probability of not drawing the j^{th} observation. Each draw is independent from one another. Thus, the probability that you don't draw the j^{th} observation at all is $(1 - 1/n)$ multiplied by itself n times, or $(1 - 1/n)^n$.

(d)

$$\mathbb{P}(x_j \notin P_5^b) = (1 - 1/5)^5 \approx 0.3277 \quad (8)$$
$$\mathbb{P}(x_j \in P_5^b) = 1 - \mathbb{P}(x_j \notin P_5^b) \approx 1 - 0.3277 = 0.6723 \quad (9)$$

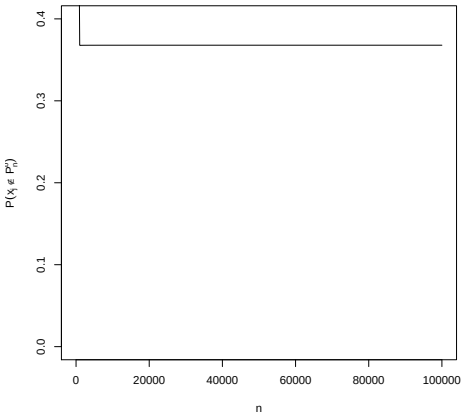
(e)

$$\mathbb{P}(x_j \notin P_{100}^b) = (1 - 1/100)^{100} \approx 0.3660 \quad (10)$$
$$\mathbb{P}(x_j \in P_{100}^b) = 1 - \mathbb{P}(x_j \notin P_{100}^b) \approx 1 - 0.3660 = 0.6340 \quad (11)$$

(f)

$$\mathbb{P}(x_j \notin P_{1000}^b) = (1 - 1/1000)^{1000} \approx 0.3677 \quad (12)$$
$$\mathbb{P}(x_j \in P_{1000}^b) = 1 - \mathbb{P}(x_j \notin P_{1000}^b) \approx 1 - 0.3677 = 0.6323 \quad (13)$$

(g)
```
prob_not_j <- function(n) {
  return ((1 - 1/n)^n)
}
plot(prob_not_j, xlim=c(0, 100000), ylim=c(0, 0.4), xlab="n",
     ylab=expression(P(x[j] %notin% P[n]^b)))
```



We see from this that the probability essentially stays constant across all values of n.

(h)
```
store <- rep(NA, 10000)
for(i in 1:10000) {
  store[i] = sum(sample(1:100, rep=TRUE)==4) > 0
}
mean(store)

## [1] 0.641
```

We see from this that the probability of a bootstrap sample containing the 4^{th} observation is 64.1%.

Problem 7 (5 points)

Chapter 5, Exercise 5 (p. 198).

```
library(ISLR)
library(MASS)
set.seed(1)
```

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

##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##                   Estimate    Std. Error z value  Pr(>|z|)
## (Intercept) -11.540468437   0.434756357 -26.545   < 2e-16 ***
## income        0.000020809   0.000004985   4.174 0.0000299 ***
## balance       0.005647103   0.000227373  24.836   < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

(b)

```
CalulateValidationError <- function(seed, formula, verbose=FALSE) {
  # Computes the validation error.
  #
  # Args:
  #   seed: Seed number. Set to integer for reproducibility.
  #   formula: A string specifying the logistic regression formula.
  #   verbose: If TRUE, prints confusion matrix. Default is FALSE.
  #
  # Returns:
  #   The computed validation error.
  if (!missing(seed)) {
    set.seed(seed)
  }
  # i. Split the sample
  train.idx <- sample(10000, 5000)

  # ii. Fit a multiple logistic regression model on training observations
  glm.fit1 <- glm(as.formula(formula), data=Default, family=binomial,
                  subset=train.idx)

  # iii. Obtain prediction
  pred.default.prob1 <- predict(glm.fit1, Default, type="response")
  pred.default1 <- rep("No", 10000)
  pred.default1[pred.default.prob1 > 0.5] <- "Yes"

  # iv. Compute validation set error
  if (verbose) {
    print(table(pred.default1[-train.idx], Default$default[-train.idx]))
  }
  validation.error <- mean((Default$default != pred.default1)[-train.idx])

  return(validation.error)
}

# Our validation error
err <- CalulateValidationError(
          seed=1,
          formula="default ~ income + balance",
```

```
            verbose=TRUE)

##
##         No  Yes
##   No  4824  108
##   Yes   19   49

  print (err)

## [1] 0.0254
```

(c) We can use a loop to repeat the process.

```
valid.set.err <- c()
  for(seed in 1:4) {
    err <- CalulateValidationError(
              seed=seed,
              formula="default ~ income + balance")
    valid.set.err <- append(valid.set.err, err)
  }
  valid.set.err

## [1] 0.0254 0.0238 0.0264 0.0256
```

The validation set errors for different splits are close, implying that our estimates of the test error are low variance. Practically, this means that we can perform the procedure once and use the value as an estimate of our test error.

(d)
```
  # Our validation error
  err2 <- CalulateValidationError(
            seed=1,
            formula="default ~ income + balance + student",
            verbose=TRUE)

##
##         No  Yes
##   No  4825  112
##   Yes   18   45

  print(err2)

## [1] 0.026
```

From this, we see that the test error does not decrease. Thus, including the `student` variable does not lead to a reduction in the test error.

Problem 8 (5 points)

Chapter 5, Exercise 6 (p. 199).

```r
#install.packages("boot")
library(boot)
set.seed(1)
```

(a)
```r
glm.fit <- glm(default ~ income + balance, data=Default, family=binomial)
std.errors <- summary(glm.fit)$coefficients[,2]
summary(glm.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##                   Estimate    Std. Error z value  Pr(>|z|)
## (Intercept) -11.540468437   0.434756357 -26.545   < 2e-16 ***
## income        0.000020809   0.000004985   4.174 0.0000299 ***
## balance       0.005647103   0.000227373  24.836   < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

From this, we see that the standard errors for `income` and `balance` are 0.000005 and 0.0002274, respectively.

(b)
```r
boot.fn <- function(data, index) {
  glm.fit <- glm(default ~ income + balance, data=data,
                 family=binomial, subset=index)
  return(coef(glm.fit))
}
set.seed(1)
boot.fn(Default, sample(10000, 10000, replace=TRUE))
```

```
##    (Intercept)         income        balance
## -12.0946962477   0.0000281529   0.0058371168
```

(c)
```r
set.seed(1)
boot(Default, boot.fn, 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
```

```
## Call:
## boot(data = Default, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##             original           bias       std. error
## t1* -11.54046843668 -0.0394545996034 0.434472206546
## t2*   0.00002080898  0.0000001680317 0.000004866284
## t3*   0.00564710294  0.0000185576474 0.000229894890
```

`t1` is the intercept, while `t2` and `t3` are the coefficients estimates of `income` and `balance`.

(d) The standard error estimates in `glm` and `boot` are very close. We see that the bootstrap method does not require any information on the model, but gives good estimates of standard errors. Conversely, the bootstrap method requires noticeably more computation.

Problem 9 (5 points)

Chapter 5, Exercise 8 (p. 200).

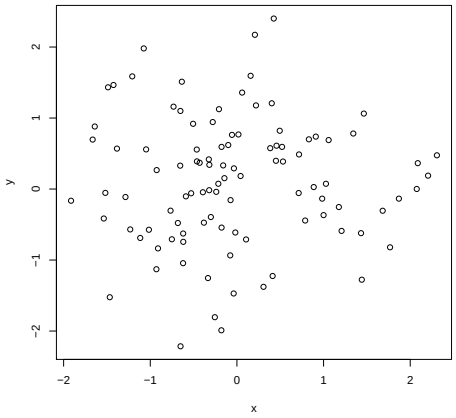
(a)
```
set.seed(1)
y <- rnorm(100)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

The model can be written as

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon \tag{14}$$

where $\beta_0 = 0$, $\beta_1 = 1$, and $\beta_2 = -2$. In this model, we have $n = 100$ and $p = 2$.

(b)
```
plot(x, y)
```



(c)
```
CalulateLoocvError <- function(dataframe, formula, seed) {
  # Computes the LOOCV error.
  #
  # Args:
  #   dataframe: A data.frame containing the data to use.
```

```
    #   formula: A string specifying the logistic regression formula.
    #   seed: Seed number. Set to integer for reproducibility.
    #
    # Returns:
    #   The computed LOOCV error.
    if (!missing(seed)) {
      set.seed(seed)
    }

    glm.fit <- glm(as.formula(formula), data=dataframe)
    loocv.fit <- cv.glm(dataframe, glm.fit)
    loocv.error <- loocv.fit$delta[1]

    return (loocv.error)
  }

  # Our LOOCV errors using seed=1
  O = data.frame(x, y)
  for (i in 1:4) {
    err <- CalulateLoocvError(
              dataframe=O,
              formula="y ~ poly(x, i, raw=TRUE)",
              seed=1)

    cat(sprintf("%d. %0.3f\n", i, err))
  }
```

```
## 1. 5.891
## 2. 1.087
## 3. 1.103
## 4. 1.115
```

(d)

```
  # Our LOOCV errors using seed=2
  for (i in 1:4) {
    err <- CalulateLoocvError(
              dataframe=O,
              formula="y ~ poly(x, i, raw=TRUE)",
              seed=2)

    cat(sprintf("%d. %0.3f\n", i, err))
  }
```

```
## 1. 5.891
## 2. 1.087
## 3. 1.103
## 4. 1.115
```

We get the same results irrespective of the seed. This is because the LOOCV error is the mean of the n MSEs corresponding to every left out observation and, consequently, there is no randomness in the training/validation splits.

(e) Model ii had the smallest LOOCV error. This is expected since we generated y from a quadritic function of x with iid noise added.

(f)
```r
for (i in 1:4) {
  lm.fit <- lm(y ~ poly(x, i, raw=TRUE), data=0)
  cat(sprintf("\n~~~ Model %d ~~~\n", i))
  print (summary(lm.fit))
}
```

```
##
## ~~~ Model 1 ~~~
##
## Call:
## lm(formula = y ~ poly(x, i, raw = TRUE), data = 0)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.3469 -0.9275  0.8028  1.5608  4.3974
##
## Coefficients:
##                         Estimate Std. Error t value        Pr(>|t|)
## (Intercept)              -1.8185     0.2364  -7.692 0.0000000000114 ***
## poly(x, i, raw = TRUE)    0.2430     0.2479   0.981           0.329
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.362 on 98 degrees of freedom
## Multiple R-squared:  0.009717,Adjusted R-squared:  -0.0003881
## F-statistic: 0.9616 on 1 and 98 DF,  p-value: 0.3292
##
##
## ~~~ Model 2 ~~~
##
## Call:
## lm(formula = y ~ poly(x, i, raw = TRUE), data = 0)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.89884 -0.53765  0.04135  0.61490  2.73607
##
## Coefficients:
##                          Estimate Std. Error t value         Pr(>|t|)
## (Intercept)              -0.09544    0.13345  -0.715            0.476
## poly(x, i, raw = TRUE)1   0.89961    0.11300   7.961 0.00000000000324 ***
## poly(x, i, raw = TRUE)2  -1.86665    0.09151 -20.399          < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.032 on 97 degrees of freedom
## Multiple R-squared:  0.8128,Adjusted R-squared:  0.8089
## F-statistic: 210.6 on 2 and 97 DF,  p-value: < 2.2e-16
##
##
## ~~~ Model 3 ~~~
##
## Call:
## lm(formula = y ~ poly(x, i, raw = TRUE), data = 0)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -2.87250 -0.53881  0.02862  0.59383  2.74350
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -0.09865    0.13453  -0.733    0.465
## poly(x, i, raw = TRUE)1  0.95551    0.22150   4.314 0.000039 ***
## poly(x, i, raw = TRUE)2 -1.85303    0.10296 -17.998  < 2e-16 ***
## poly(x, i, raw = TRUE)3 -0.02479    0.08435  -0.294    0.769
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.037 on 96 degrees of freedom
## Multiple R-squared:  0.813,Adjusted R-squared:  0.8071
## F-statistic: 139.1 on 3 and 96 DF,  p-value: < 2.2e-16
##
##
## ~~~ Model 4 ~~~
##
## Call:
## lm(formula = y ~ poly(x, i, raw = TRUE), data = 0)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8914 -0.5244  0.0749  0.5932  2.7796
##
## Coefficients:
##                         Estimate Std. Error t value    Pr(>|t|)
## (Intercept)              -0.13897    0.15973  -0.870    0.386455
## poly(x, i, raw = TRUE)1  0.90980    0.24249   3.752    0.000302 ***
## poly(x, i, raw = TRUE)2 -1.72802    0.28379  -6.089 0.000000024 ***
## poly(x, i, raw = TRUE)3  0.00715    0.10832   0.066    0.947510
## poly(x, i, raw = TRUE)4 -0.03807    0.08049  -0.473    0.637291
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.041 on 95 degrees of freedom
## Multiple R-squared:  0.8134,Adjusted R-squared:  0.8055
## F-statistic: 103.5 on 4 and 95 DF,  p-value: < 2.2e-16
```

Model ii has the smallest residual standard error and adjusted R-squared value, which agrees with our previous result.

From a backward stepwise regression perspective, we observe that the p-values of third and fourth degree coefficients are not significant in model iv. Also, the p-value of the third degree coefficient is not significant in model iii. Every coefficient in model ii is statistically significant at a level of 0.05.

Problem 10 (5 points)

Chapter 5, Exercise 9 (p. 201).

```
library(MASS)
data(Boston)
```

(a)
```r
mu.hat <- mean(Boston$medv)
cat(sprintf("Estimated population mean: %0.3f", mu.hat))
```

```
## Estimated population mean: 22.533
```

(b)
```r
mu.hat.se <- sqrt(var(Boston$medv) / nrow(Boston))
cat(sprintf("Estimated standard error: %0.3f", mu.hat.se))
```

```
## Estimated standard error: 0.409
```

(c)
```r
BootFnMean <- function(data, index) {
  # Computes the mean from indexed observations.
  #
  # Args:
  #   data: A vector containing the data to estimate the mean on.
  #   index: A boolean index indicating which indices to use.
  #
  # Returns:
  #   The computed mean.
  est.mean <- mean(data[index])
  return (est.mean)
}

boot.est <- boot(Boston$medv, BootFnMean, 1000)
boot.est
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = BootFnMean, R = 1000)
##
##
## Bootstrap Statistics :
##     original       bias    std. error
## t1* 22.53281 -0.008578854   0.4072277
```

The standard errors from the asymptotic formula and bootstrap are very close.

(d)
```r
# Bootstrap CI
boot.se <- sd(boot.est$t)
ci.boot <- mu.hat + c(-1,1)*qnorm(.975)*boot.se
cat(sprintf("95\\%% CI: %0.3f, %0.3f\n", ci.boot[1], ci.boot[2]))
```

```
## 95\% CI: 21.735, 23.331
```

```r
# t.test
t.test(Boston$medv)
```

```
##
##  One Sample t-test
##
```

```
## data:  Boston$medv
## t = 55.111, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.72953 23.33608
## sample estimates:
## mean of x
##  22.53281
```

The two confidence intervals are very close.

(e)
```
median.hat <- median(Boston$medv)
cat(sprintf("Estimated population median: %0.3f", median.hat))

## Estimated population median: 21.200
```

(f)
```
BootFnQuantile <- function(data, index, p=0.5) {
  # Computes the quantile from indexed observations.
  #
  # Args:
  #   data: A vector containing the data to estimate the quantile on. By
  #         default, this is set to the median (i.e. p = 0.5).
  #   index: A boolean index indicating which indices to use.
  #
  # Returns:
  #   The computed quantile.
  est.quantile <- quantile(data[index], p)
  return (est.quantile)
}

boot.est <- boot(Boston$medv, BootFnQuantile, 1000)
boot.est

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = BootFnQuantile, R = 1000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*     21.2 -0.02505   0.3733288
```

We get a reasonable estimate of the standard error without any formula for computing the standard error of the median. The median is more robust to outliers than the mean, and we observe that its standard error is indeed smaller than the standard error of the mean.

Note: Recall from class that we can also use influence functions for estimating the variance (and correspondingly the standard error) of estimators. It can be shown that the influence function for the quantiles (such as the median) is

$$D(P)(X) = \frac{p - \mathbb{I}(X \leq Q(p))}{f(Q(p))} \tag{15}$$

where f denotes the pdf for X and p denotes the quantile. Note that valid coverage for this interval depends upon our estimator for f converging at a fast enough rate.

Using this formula, we can estimate the standard error for the median as well, i.e.

```
DpoQuantile <- function(x, p) {
  # Computes the influence function for a quantile.
  #
  # The pdf for x is estimate using a Ripley's kernel density estimator.
  #
  # Wand M, Ripley B (2010). KernSmooth: Functions for kernel smoothing
  # for Wand & Jones (1995). R package version 2.23-4,
  # URL http://CRAN.R-project.org/package=KernSmooth.
  #
  # Args:
  #   x: A vector containing the values to compute the influence function.
  #   p: The quantile of interest.
  #
  # Returns:
  #   The computed influence function.
  library(KernSmooth)
  Q.p <- quantile(x, p)
  fn.hat <- approxfun(bkde(x))
  DPO <- (p - 1*I(x <= Q.p)) / fn.hat(Q.p)
  return (DPO)
}
influence.function.est <- DpoQuantile(Boston$medv, 0.5)
median.hat.se <- sqrt(var(influence.function.est) / nrow(Boston))
cat(sprintf("Estimated standard error: %0.3f", median.hat.se))
```

```
## Estimated standard error: 0.394
```

(g)

```
q10.hat <- quantile(Boston$medv, 0.1)
cat(sprintf("Estimated population tenth percentile: %0.3f", q10.hat))
```

```
## Estimated population tenth percentile: 12.750
```

(h)

```
boot.est <- boot(Boston$medv, BootFnQuantile, 1000, p=0.1)
boot.est
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = BootFnQuantile, R = 1000,
##     p = 0.1)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75 -0.0076   0.5002224
```

The estimate of the standard error for the 10^{th} percentile is larger than the one we estimated for the median. We can expect that the standard error of the p^{th} percentile will increase as p gets

closer to 0 or 1 due to the lower probabilities of observing those values. Consider, for example, an extreme case such that $p = 0.001$. Very little data will be at that point, especially when we only have 506 observations. Consequently, the estimate of the p^{th} percentile will have a large variance.

Note: We can apply the same influence function as above to this problem as well, giving us

```
  influence.function.est <- DpoQuantile(Boston$medv, 0.1)
  median.hat.se <- sqrt(var(influence.function.est) / nrow(Boston))
  cat(sprintf("Estimated standard error: %0.3f", median.hat.se))

## Estimated standard error: 0.488
```