5.4 Exercises

Exercise 5

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

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.4.4
```

```
fix(Default)
?Default
```

```
## starting httpd help server ... done
```

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

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

```
##
## Call:  glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Coefficients:
## (Intercept)       income      balance
##  -1.154e+01    2.081e-05    5.647e-03
##
## Degrees of Freedom: 9999 Total (i.e. Null);  9997 Residual
## Null Deviance:       2921
## Residual Deviance: 1579  AIC: 1585
```

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

c. Split the sample set into a training set and a validation set.

ii. Fit a multiple logistic regression model using only the training observations.

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

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

```
set.seed(1)
train = sample(dim(Default)[1], dim(Default)[1]/2)
glm.fit = glm(default ~ income + balance, data = Default, subset = train, family = binom
ial)
glm.prob = predict(glm.fit, Default[-train,], type = 'response')
glm.pred = rep('No', dim(Default[-train,])[1])
glm.pred[glm.prob > .5] = 'Yes'
mean(glm.pred == Default[-train, 'default'])
```

```
## [1] 0.9714
```

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

```
for (x in 1:3) {
  train = sample(dim(Default)[1], dim(Default)[1]/2)
  glm.fit = glm(default ~ income + balance, data = Default, subset = train, family = bin
omial)
  glm.prob = predict(glm.fit, Default[-train,], type = 'response')
  glm.pred = rep('No', dim(Default[-train,])[1])
  glm.pred[glm.prob > .5] = 'Yes'
  print(mean(glm.pred == Default[-train, 'default']))
}
```

```
## [1] 0.9764
## [1] 0.972
## [1] 0.9732
```

These predictions are all fairly close, but they don't seem to give more than two significant figures of precision. Still, we can conclude that the model test error rate is probably close to 3%.

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

```
train = sample(dim(Default)[1], dim(Default)[1]/2)
glm.fit = glm(default ~ income + balance + student, data = Default, subset = train, fami
ly = binomial)
glm.prob = predict(glm.fit, Default[-train,], type = 'response')
glm.pred = rep('No', dim(Default[-train,])[1])
glm.pred[glm.prob > .5] = 'Yes'
mean(glm.pred == Default[-train, 'default'])
```

```
## [1] 0.9736
```

The inclusion of the **student** *predictor doesn't increase the estimated accuracy from the model. Therefore, it would probably be best to leave it out for better interpretability and potentially less overfitting.*

Exercise 6

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

```
library(boot)
```

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

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

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

The estimated standard error for **income** *is* $4.985 * 10^{-6}$. *The estimated standard error for* **balance** *is* $2.274 * 10^{-4}$.

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

```
boot.fn = function (data, index) return(coef(glm(default ~ income + balance, data = dat
a, subset = index, family = binomial))[c('income', 'balance')])
```

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

```
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* 2.080898e-05 5.870933e-08 4.582525e-06
## t2* 5.647103e-03 2.299970e-06 2.267955e-04
```

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

Using the bootstrap, the estimated standard error for **income** is $4.582525 * 10^{-6}$. *Using the bootstrap, the estimated standard error for* **balance** is $2.267955 * 10^{-4}$.

```
(4.582525 - 4.985) / 4.985
```

```
## [1] -0.08073721
```

```
(2.267955 - 2.274) / 2.274
```

```
## [1] -0.002658311
```

The estimated standard error for **income** *changed* 8% *using the bootstrap. The estimated standard error for* **balance** *changed* <1% *using the bootstrap.*

Exercise 7

In Sections 5.3.2 and 5.3.3, we saw that the **cv.glm()** function can be used in order to compute the LOOCV test error estimate. Alternatively, one could compute those quantities using just the **glm()** and **predict.glm()** functions, and a for loop. You will now take this approach in order to compute the LOOCV error for a simple logistic regression model on the **Weekly** data set. Recall that in the context of classification probelms, the LOOCV error is given in (5.4).

```
fix(Weekly)
?Weekly
```

a. Fit a logistic regression model that predicts **Direction** using **Lag1** and **Lag2**.

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

```
##
## Call:  glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = Weekly)
##
## Coefficients:
## (Intercept)         Lag1         Lag2
##     0.22122     -0.03872      0.06025
##
## Degrees of Freedom: 1088 Total (i.e. Null);  1086 Residual
## Null Deviance:       1496
## Residual Deviance: 1488  AIC: 1494
```

b. Fit a logistic regression model that predicts **Direction** using **Lag1** and **Lag2** *using all but the first observation*.

```
glm.fit = glm(Direction ~ Lag1 + Lag2, data = Weekly, subset = -1, family = binomial)
glm.fit
```

```
##
## Call:  glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = Weekly,
##     subset = -1)
##
## Coefficients:
## (Intercept)         Lag1         Lag2
##     0.22324     -0.03843      0.06085
##
## Degrees of Freedom: 1087 Total (i.e. Null);  1085 Residual
## Null Deviance:       1495
## Residual Deviance: 1487  AIC: 1493
```

c. Use the model from (b) to predict the direction of the first observation. You can do this by predicting that the first observation will go up if *P*(**Direction="Up"|Lag1**,**Lag2**) > 0.5. Was this observation correctly classified?

```
prob = predict(glm.fit, Weekly[1,], type = 'response')
(prob > .5) == (Weekly[1, 'Direction'] == 'Up')
```

```
##     1
## FALSE
```

This model does not correctly classify the first observation.

d. Write a loop from $i = 1$ to $i = n$, where *n* is the number of observations in the data set, that performs each of the following steps:

e. Fit a logistic regression model using all but the *i*th observation to predict **Direction** using **Lag1** and **Lag2**.

ii. Compute the posterior probability of the market moving up for the *i*th observation.

iii. Use the posterior probability for the *i*th observation in order to predict whether or not the market moves up.

iv. Determine whether or not an error was made in predicting the direction for the *i*th observation. If an error was made, then indicate this as a 1, and otherwise indicate it as a 0.

```
n = dim(Weekly)[1]
accuracy = rep(FALSE, n)
for (i in 1:n) accuracy[i] = (predict(glm(Direction ~ Lag1 + Lag2, data = Weekly, subset
= -i, family = binomial), Weekly[i,], type = 'response') > .5) == (Weekly[i, 'Direction'
] == 'Up')
```

e. Take the average of the *n* numbers obtained in (d(iv)) in order to obtain the LOOCV estimate for the test error. Comment on the results.

```
mean(accuracy)
```

```
## [1] 0.5500459
```

The estimated test error rate for the model using LOOCV is 45%.

Exercise 8

a. Generate a simulated data set as follows:

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

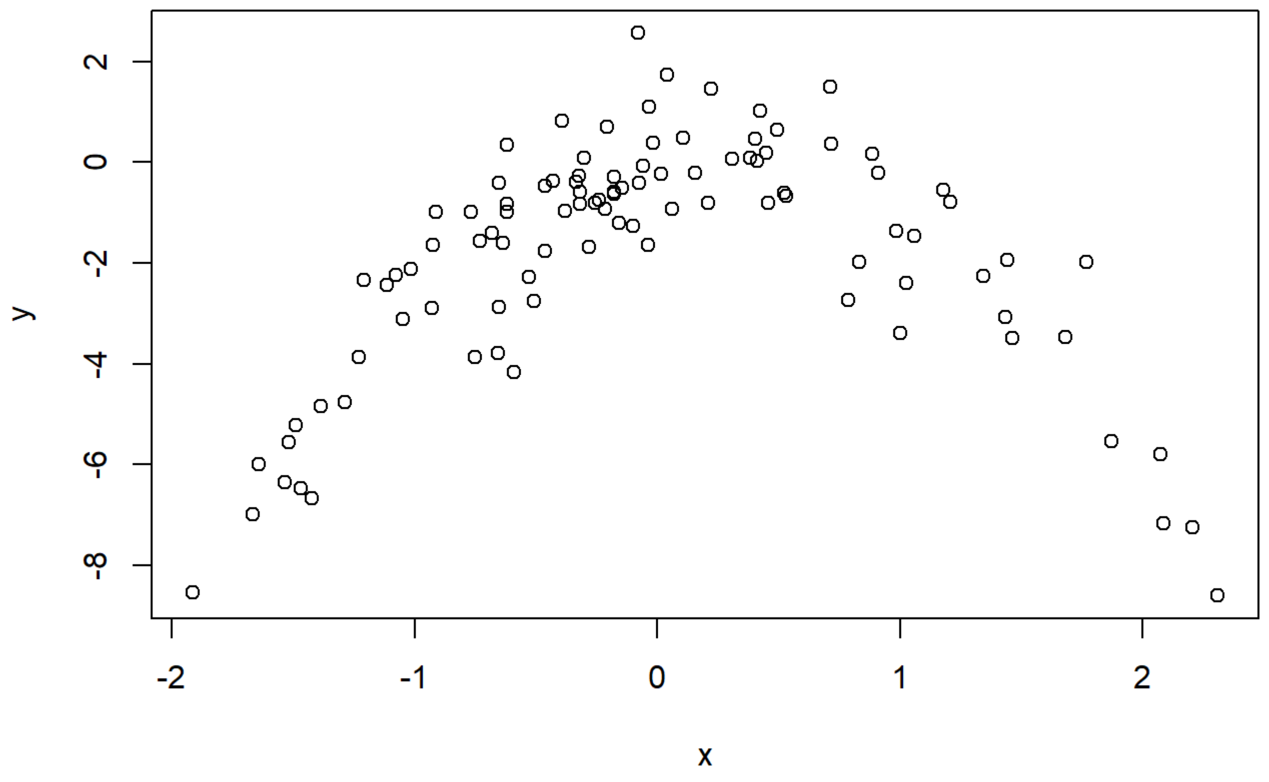
In this data set, what is *n* and what is *p*? Write out the model used to generate the data in equation form.

```
set.seed(1)
y = rnorm(100) # this y is useless, but I will leave the line in to match the results of
the exercise
x = rnorm(100)
y = x - 2 * x ^ 2 + rnorm(100)
```

n *is* 100, *the number of observations.* p *is* 2, *the number of predictors. The equation of the model is* $y = x - 2x^2 + \epsilon$.

b. Create a scatterplot of *X* against *Y*. Comment on what you find.

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



The scatter plot mostly looks parabolic. The linear component is difficult to notice visually. This makes sense since the quadratic term's coefficient is double the coefficient of the linear term.

c. Set a random seed and then compute the LOOCV errors that result from fitting the following four models using least squares:

d. $Y = \beta_0 + \beta_1 X + \epsilon$

ii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$

iii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$

iv. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$

Note you may find it helpful to use the **data.frame()** function to create a single data set containing both *X* and *Y*.

```
set.seed(1)
data = data.frame(x = x, y = y)
for (z in 1:4) print(cv.glm(data, glm(y ~ poly(x, z), data = data))$delta)
```

```
## [1] 5.890979 5.888812
## [1] 1.086596 1.086326
## [1] 1.102585 1.102227
## [1] 1.114772 1.114334
```

d. Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

```
set.seed(2)
data = data.frame(x = x, y = y)
for (z in 1:4) print(cv.glm(data, glm(y ~ poly(x, z), data = data))$delta)
```

```
## [1] 5.890979 5.888812
## [1] 1.086596 1.086326
## [1] 1.102585 1.102227
## [1] 1.114772 1.114334
```

The results are the same because LOOCV has no variability in it. It will always produce the same estimates.

e. Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

The smallest LOOCV error comes from the 2nd degree polynomial fit, which makes sense, since that is the true relationship between x and y.

f. Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusion drawn based on the cross-validation results?

```
for (z in 1:4) print(summary(glm(y ~ poly(x, z))))
```

```
##
## Call:
## glm(formula = y ~ poly(x, z))
##
## Deviance 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)  -1.8277     0.2362  -7.737 9.18e-12 ***
## poly(x, z)    2.3164     2.3622   0.981    0.329
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 5.580018)
##
##     Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 546.84  on 98  degrees of freedom
## AIC: 459.69
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, z))
##
## Deviance 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)  -1.8277     0.1032 -17.704   <2e-16 ***
## poly(x, z)1   2.3164     1.0324   2.244   0.0271 *
## poly(x, z)2 -21.0586     1.0324 -20.399   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.06575)
##
##     Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 103.38  on 97  degrees of freedom
## AIC: 295.11
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, z))
##
## Deviance 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)  -1.8277     0.1037 -17.621   <2e-16 ***
## poly(x, z)1   2.3164     1.0372   2.233   0.0279 *
## poly(x, z)2 -21.0586     1.0372 -20.302   <2e-16 ***
## poly(x, z)3  -0.3048     1.0372  -0.294   0.7695
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.075883)
##
##     Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 103.28  on 96  degrees of freedom
## AIC: 297.02
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, z))
##
## Deviance 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)  -1.8277     0.1041 -17.549   <2e-16 ***
## poly(x, z)1   2.3164     1.0415   2.224   0.0285 *
## poly(x, z)2 -21.0586     1.0415 -20.220   <2e-16 ***
## poly(x, z)3  -0.3048     1.0415  -0.293   0.7704
## poly(x, z)4  -0.4926     1.0415  -0.473   0.6373
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.084654)
##
##     Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 103.04  on 95  degrees of freedom
## AIC: 298.78
##
## Number of Fisher Scoring iterations: 2
```

As higher degree terms get introduced to the model, the p-values of the lower degree terms doesn't change much. The only terms of statistical significance are the $\beta_0, \beta_1, \beta_2$ terms. This makes sense since those are the only terms in the actual relationship. Also, the p-value of the quadratic term is much lower than the p-value of the linear term, which matches what we observed in the observations from part (b).

Exercise 9

We will now consider the **Boston** housing data set, from the **MASS* library.

```
library(MASS)
fix(Boston)
?Boston
```

a. Based on this data set, provide an estimate for the population mean of **medv**. Call this estimate $\hat{\mu}$.

```
mean(Boston$medv)
```

```
## [1] 22.53281
```

b. Provide an estimate of the standard error of $\hat{\mu}$. Interpret this result.

Hint: We can compute the standard error of the sample mean by dividing the sample standard devation by the square root of the number of observations.

```
sd(Boston$medv) / sqrt(dim(Boston)[1])
```

```
## [1] 0.4088611
```

c. Now estimate the standard error of $\hat{\mu}$ using the bootstrap. How does this compare to your answer form (b)?

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

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 22.53281 0.008517589   0.4119374
```

The bootstrap estimates the standard error to be 0.412, *while the sample standard error estimate is* 0.409.

```
(.420 - .412) / .412
```

```
## [1] 0.01941748
```

The boostrap estimate for the standard error is 1.94% *higher.*

d. Based on your bootstrap estimate from (c), provide a 95% confidence interval for the mean of **medv**. Compare it to the results obtained using **t.test(Boston$medv)$.

Hint: you can approximate a 95% confidence interval using the formula $[\hat{\mu} - 2SE(\hat{\mu}), \hat{\mu} + 2SE(\hat{\mu})]$.

```
mu.hat = 22.53281
mu.se  = 0.4119374
c(mu.hat - 2 * mu.se, mu.hat + 2 * mu.se)
```

```
## [1] 21.70894 23.35668
```

e. Based on this data set, provide an estimate, $\hat{\mu_{med}}$, for the median value of **medv** in population.

```
median(Boston$medv)
```

```
## [1] 21.2
```

f. We now would like to estimate the standard of $\hat{\mu_{med}}$. Unfortunately, there is no simple formula for computing the standard error of the median. Instead, estimate the standard error of the median using the bootstrap. Comment on your findings.

```
set.seed(1)
boot(Boston, function (data, index) return(median(data[index, 'medv'])), 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = function(data, index) return(median(data[index,
##     "medv"])), R = 1000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*     21.2 -0.01615   0.3801002
```

The estimated standard error of the median is 0.380, *a bit lower than the estimated standard error of the mean.*

g. Based on this data set, provide an estimate for the tenth percentile of **medv** in Boston suburbs. Call this quantity $\hat{\mu_{0.1}}$. (You can use the **quantile()** function.)

```
quantile(Boston$medv, .1)
```

```
##   10%
## 12.75
```

h. Use the bootstrap to estimate the standard error of $\hat{\mu_{0.1}}$. Comment on your findings.

```
set.seed(1)
boot(Boston, function (data, index) return(quantile(data[index, 'medv'], .1)), 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = function(data, index) return(quantile(data[index,
##     "medv"], 0.1)), R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75 0.01005    0.505056
```

The estimated standard error for $\hat{\mu_{0.1}}$ *is* 0.505, *a bit higher than the estimated standard error for the mean and quite a bit higher than the estimated standard error for the median,* $\hat{\mu_{0.5}}$.