Practice Midterm Solutions

Instructions (Read this completely first)

You should complete the exam by editting this file directly. Please knit the file often, so that if you make a mistake you catch it before the end of the exam. You will have exactly 1.5 hours (90 minutes) from your start time to complete the exam. **At the end you must turn in your knitted .pdf file and raw .Rmd file on Courseworks.**

When the time is up, you must shut your computer immediately. We will take off points from anyone whose computer is still open after time is up.

You may use your class notes for the exam, but not the internet. You absolutely may not communicate with anyone else during the exam. Doing so will result in an F in this class and likely result in termination from the MA program. Note that your time will be tight so you will not be able to look up every bit of code from your class notes.

Question 0 (4 points)

a. (0.5 points) Place your section number as the date of the document. If you don't know your section number, you can determine it below based on when your lab meets.

- Section 002 Lab meets TR 7:40pm-8:55pm
- Section 003 Lab meets TR 11:40am-12:55pm
- Section 004 Lab meets MW 8:40am-9:55am
- Section 005 Lab meets TR 8:40am-9:55am

b. (0.5 points) Write your name and UNI as the author of the document.

c. (1 point) After you have submitted the exam, fill out the survey on Courseworks. You have until tomorrow evening to do this and in the survey you will have the opportunity to tell me if you see classmates in your vacinity communicating with others during the exam.

d. (2 points) Please present your answers in a readable format. This includes things like indenting your code and generally presenting easy-to-read code. Presentation of the overall Markdown document will be considered as well.

Question 1: Manipulating Data Frames

a. Read in the data `flights.csv` and save it in a dataframe called `flights`.

```r
flights <- read.csv("flights.csv", as.is = TRUE, header = TRUE)
```

b. We are interested only in flights with `air_time` (amount of time spent in the air, in minutes) that is greater than 30 minutes but less than 5 hours. Create a new dataframe `flights_new` that contain only these flights.

```r
dur_range   <- flights$air_time >= 30 & flights$air_time <= 300
flights_new <- flights[dur_range, ]
```

c. Write code that adds a column to the data frame `flights_new` called `travel_speed` calculated using the `air_time` and the `distance` (distance between airports, in miles). Find this in the following way, if the plane is traveling at x miles per hour on average and the distance it traveled is y mile, then the duration z (in hours) is:

$$z \text{ hours } = \frac{y \text{ miles}}{x \text{ miles per hour}}.$$

Note that your variable `travel_speed` should report the speed in miles **per hour**.

Solution:

```r
flights_new$travel_speed <- flights_new$distance/(flights_new$air_time/60)
```

d Let n be the number of rows in the data frame `flights_new`. Without using a loop, create a vector `message` of length n which says "On Time" if the flight was on time or "Delayed" if the flight was delayed. We will say that a flight is on time if the `arr_delay` (arrival delays, in minutes. Negative times represent early arrivals) is less than 30 minutes.

Solution:

```r
condition <- flights_new$arr_delay < 30
message   <- ifelse(condition, "On Time", "Delayed")
```

Question 2: Logistic Regression

Logistic regression is technique which works like linear regression, but is used when the dependent variable is a binary category, rather than a continuous real number. It is implemented in `R` through the function `glm(formula, data, family=binomial(link="logit"))` (compared to linear regression: `lm(formula, data)`). The function `glm()` returns a complicated object containing estimated coefficients, standard errors, residuals, etc. Imagine `voter_info` is a data frame with information on voter state, income, race, and status as a republican or not and `states$mean_income` holds the average per-capita income for each state. Now consider the following code.

```
num.states <- unique(levels(voter_info$state))
n          <- nrow(voter_info)

income.coefs <- rep(NA, length = num.states)
for (state in 1:num.states)) {
  this_state <- NULL
  for (voter in 1:n) {
    if (voter_info$state[voter] == levels(voter_info$state)[state]) {
       if (is.null(this_state)) {
         this_state <- as.data.frame(voter_info[voter,])
       } else {
         this_state <- rbind(this_state, voter_info[voter,])
       }
    }
  }
  rep_vs_income <- glm(republican ~ income, data = this_state,
                       family = binomial(link = "logit"))
  income.coefs[state] <- coefficients(rep_vs_income)$income
}
plot(states$mean_income, income.coefs, xlab = "State's income",
     ylab = "Coefficient for voting Republican as income inceases")
```

a. Explain in one sentence what the above code is supposed to do.

Solution: The code fits a logistic regression model for the independent variable voter status as a Republican or not using income as the dependent variable for each state and then plots coefficients estimated by the model against the state's mean income.

b. Replace the inner for loop with at most two lines of code and no iteration. Fill in the blanks in the following. (It can be done in a single line of code but no points will be taken off for doing it in two). Note that you must leave the `\begin{verbatim}` and `\end{verbatim}` commands for this to knit properly. You can't actually run this code in an R chunk since we don't actually have the data.

```
num.states <- length(levels(voter_info$state))
n          <- nrow(voter_info)

income.coefs <- rep(NA, length = num.states)
for (state in 1:num.states)) {
  this_state_logical  <- voter_info$state == levels(voter_info$state)[state]
  this_state          <- voter_info[this_state_logical, ]
  rep_vs_income       <- glm(republican ~ income, data = this_state,
                             family = binomial(link = "logit"))
  income.coefs[state] <- coefficients(rep_vs_income)$income
}
plot(states$mean_income, income.coefs, xlab = "State's income",
     ylab = "Coefficient for voting Republican as income inceases")
```

c. Suppose that

   ```
   > this_state$republican
   [1] 1 1 0 1 0
   ```

   and

   ```
   > fitted(rep_vs_income)
   [1] 1 0 0 1 1
   ```

   where `rep_vs_income` is the fitted model for `this_state`. Write code to calculate the training error for this model? I have included code to create two vectors, the `truth` vector corresponding to the `this_state$republican` data and a `fitted` vector corresponding to `fitted(rep_vs_income)`. HINT: Since this is a classification problem, the training error will be measured in the same way we measure training error for the K Nearest Neighbors model.

Solution: The training error is given by $\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(Y_i \neq \hat{Y}_i)$ and so for this data equals $\frac{1}{5}\left(\mathbb{I}(1 \neq 1) + \mathbb{I}(1 \neq 0) + \mathbb{I}(0 \neq 0) + \mathbb{I}(1 \neq\right.$
$\frac{1}{5}\left(0+1+0+0+1\right) = \frac{2}{5}$.

```r
truth <- c(1, 1, 0, 1, 0)
fitted <- c(1, 0, 0, 1, 1)

mean(truth != fitted)
```

```
## [1] 0.4
```

Question 3: Writing Functions

(a) In mathematics, the Fibonacci numbers are the numbers in the following integer sequence, called the Fibonacci sequence, and characterized by the fact that every number in it is the sum of the two preceding ones:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, ...$$

The following code calculates the 100^{th} Fibonacci number:

```
n <- 100
f  <- rep(NA, n)
f[1] <- 0
f[2] <- 1

for (i in 3:n) {
  f[i] <- f[i-1] + f[i-2]
}
```

The 100^{th} Fibonacci number could be found by typing `f[100]`. Write a function `fibonacci()` which takes as input a value n and returns the n^{th} Fibonacci number using the above code (with some slight edits) in the body of the function.

```
fibonacci <- function(n){
  if (n == 1) {return(0)}
  else if (n == 2) {return(1)}
  else {
    f  <- rep(NA, n)
    f[1] <- 0
    f[2] <- 1

    for (i in 3:n) {
      f[i] <- f[i-1] + f[i-2]
    }
    return(f[n])
    }
}
```

(b) Write a function `last.four` that takes as input strings of 16 digit credit card numbers and returns the last four. Run the command `last.four("1234567890123456")`. The output should be `"3456"`.

```
last.four <- function(digits){
  return(substr(digits, start = 13, stop = 16))
}
```

(c) We have the function:

```
x <- 10
f <- function(x, y = x*2) {x+y}
```

What is the output of `f(2)`? (Meant to be solved without actually typing it into R, but once you figure it out, run it to verify your solution.)

```
f(2)
```

```
## [1] 6
```

(d) We have the function:

```r
g <- function(y) { return(y*(2^(-y))) }
f <- function(x) { return(g(x^2)) }
y <- 3
x <- 2
```

What is the output of `f(1)`? (Meant to be solved without actually typing it into R, but once you figure it out, run it to verify your solution)

```r
f(1)
```

```
## [1] 0.5
```

Question 4

Below you will find chunks of `R` code that contain bugs (errors). Use your best judgement to write a corrected version of the code (meaning it does what the author probably wanted).

a.
```
all.cases <- c()
case.1    <- c(1610, 4)
rbind(all.cases, case.1)
case.2    <- c(1877,2)
rbind(all.cases, case.2)
```

Correction:

```r
all.cases <- c()
case.1    <- c(1610, 4)
all.cases <- rbind(all.cases, case.1)
case.2    <- c(1877,2)
all.cases <- rbind(all.cases, case.2)
```

b.
```
x <- 1:3
y <- 1:5
element.indices <- matrix(NA, nrow = length(y), ncol = length(x))
for (i in 1:length(x)) {
  for (j in 1:length(y)) {
element.indices[i,j] <- paste("[", i, ",", j, "]")
  }
}
```

Correction:

```r
x <- 1:3
y <- 1:5
element.indices <- matrix(NA, nrow = length(x), ncol = length(y))
for (i in 1:length(x)) {
  for (j in 1:length(y)) {
    element.indices[i,j] <- paste("[", i, ",", j, "]")
  }
}
```