STAT-2450-Assignment1

Greg Miller

February 3, 2017

PART 1 - R programming basics

Vector and Matrix

1)

```
## Create vector w/ 12 random numbers 1:100 using the sample function and print
vec = c(sample(1:100,12,replace = FALSE))
print(vec)
```

```
##  [1] 45 13 30 36 53 26 21 41 75  6 25 94
```

2)

```
## Return last 5 elements of the vector vec
tail(vec,5)
```

```
## [1] 41 75  6 25 94
```

3)

```
## Create matrix mat (3x4) from vector vec
mat = matrix(vec,3,4)
```

4)

```
## Return element in 2nd row and 3rd column
mat[2,3]
```

```
## [1] 41
```

5)

```
## Return elements of last 2 rows in last 2 columns
mat[2:3,3:4]
```

```
##      [,1] [,2]
## [1,]   41   25
## [2,]   75   94
```

6)

```
## Function iterating through matrix to get odd values and print them
iterate = function(){
  v <- vector()
  for(i in 1:nrow(mat)){
    for(j in 1:ncol(mat))
    {
      if(mat[i,j] %% 2 != 0){ ## if there's a remainder, it's odd,
        v <- c(v,mat[i,j]) ## add matrix value to vector
      }
    }
  } ## /for
  print(v)
} ## /function

## Prints all odd values from matrix mat
iterate()
```

```
## [1] 45 21 13 53 41 25 75
```

Data Frame

1)

```
## Download file
library(gdata)
```

```
## gdata: read.xls support for 'XLS' (Excel 97-2004) files ENABLED.

##

## gdata: read.xls support for 'XLSX' (Excel 2007+) files ENABLED.

##
## Attaching package: 'gdata'

## The following object is masked from 'package:stats':
##
##     nobs

## The following object is masked from 'package:utils':
##
##     object.size
```

```
setwd("/home/greg/Downloads") ## set working directory
mydata = read.csv(paste(getwd(),"/Credit.csv",sep="")) ## read in Credit.csv
```

2)

```
## Display number of rows and columns in mydata dataset
nrows <- nrow(mydata)
ncols <- ncol(mydata)
print(nrows)
```

```
## [1] 400
```

```r
print(ncols)
```

```
## [1] 12
```

3)

```r
## Choose rows where Income > 100 and Age < 50
mydata[mydata$Income>100 & mydata$Age<50,]
```

```
##       X  Income Limit Rating Cards Age Education Gender Student Married
## 4     4 148.924  9504    681     3  36        11 Female      No      No
## 29   29 186.634 13414    949     2  41        14 Female      No     Yes
## 33   33 134.181  7838    563     2  48        13 Female      No      No
## 67   67 113.829  9704    694     4  38        13 Female      No     Yes
## 79   79 110.968  6662    468     3  45        11 Female      No     Yes
## 86   86 152.298 12066    828     4  41        12 Female      No     Yes
## 194 194 130.209 10088    730     7  39        19 Female      No     Yes
## 294 294 140.672 11200    817     7  46         9   Male      No     Yes
## 353 353 104.483  7140    507     2  41        14   Male      No     Yes
##            Ethnicity Balance
## 4              Asian     964
## 29  African American    1809
## 33         Caucasian     526
## 67             Asian    1388
## 79         Caucasian     391
## 86             Asian    1779
## 194        Caucasian    1426
## 294 African American    1677
## 353 African American     583
```

4)

```r
## Save data and separate columns with ";"
setwd("/home/greg/Desktop") ## set working directory
write.table(mydata, file="foo.table", sep=";") ## write to desktop, set separator value
```

Create a function

1)

```r
## function returning all numbers between startNumber,endNumber which are
## divisible by 7 but are not a multiple of 5
findDivisibleNotMultiple = function(startNumber, endNumber){
  v <- c()
  for(i in startNumber:endNumber){
    if(i %% 7 == 0 & i %% 5 != 0){
      v <- c(v,i)
    }
  }
```

```r
  return(v)
}
## find all associated values from 100 to 200
v <- c(findDivisibleNotMultiple(100,200))
v
```

```
##  [1] 112 119 126 133 147 154 161 168 182 189 196
```

2)

```r
## function normalizing a given vector
normalizeVector = function(v){
  min <- min(v)
  max <- max(v)
  index <- 0;

  for(i in v){
    index = index + 1
    v[index] = (i - min)/(max-min)
  }
  print(v)
}

## normalize vector from first question
normalizeVector(vec)
```

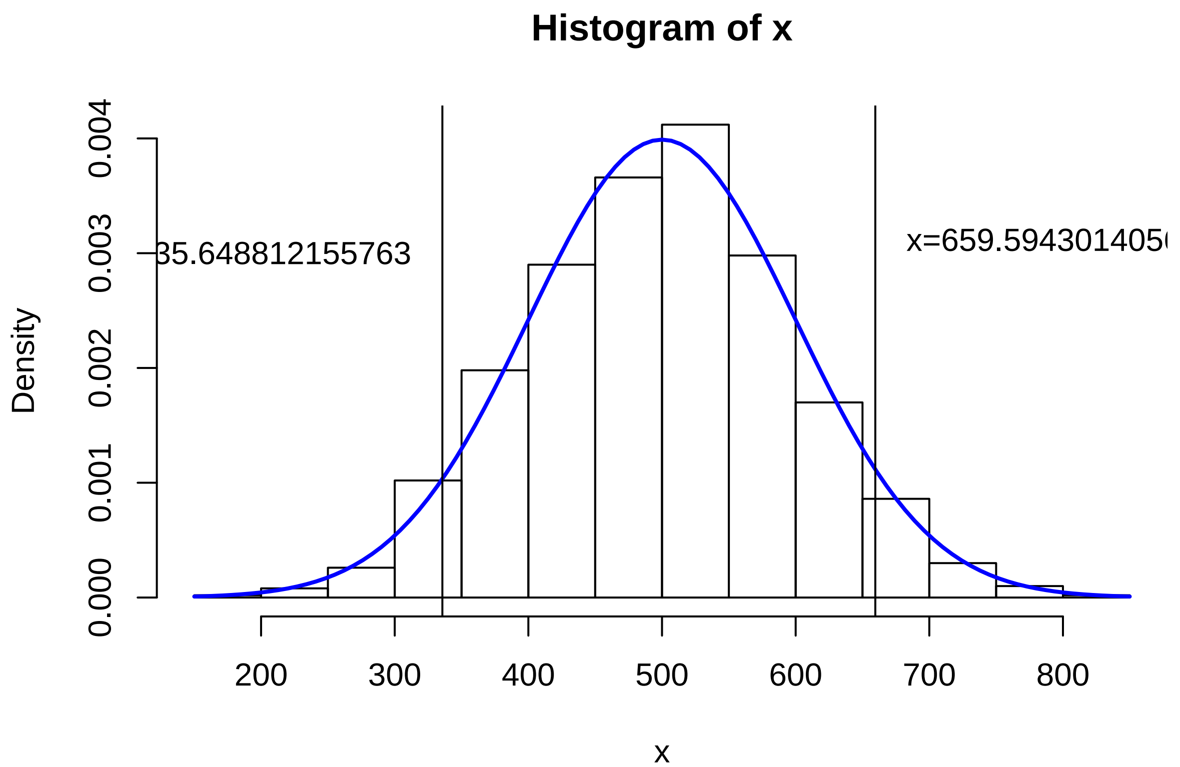
```
##  [1] 0.44318182 0.07954545 0.27272727 0.34090909 0.53409091 0.22727273
##  [7] 0.17045455 0.39772727 0.78409091 0.00000000 0.21590909 1.00000000
```

PART 2 - Data Visualization

1)

```r
set.seed(340) ## Set seeder from my last 3 digits of student number

## 2)
## Assign mean and sd, then generate 1000 samples from those values
nmean <- 500
nsd <- 100
x <- rnorm(1000, nmean, nsd)


## 3)
## Use samples to draw histogram
hist(x,probability = TRUE)

## 4)
## Add curve matching the distribution
curve(dnorm(x, mean=500,sd=100), col="blue", lwd=2, add=TRUE)

## 5)
```

```
## add vertical lines at 0.05 and 0.95 quartiles, respectively
firstXVal <- quantile(x,probs=0.05)
secondXVal <- quantile(x,probs=0.95)
## plot lines for quantile values of x
abline(v=firstXVal)
abline(v=secondXVal)

## 6)
## display x val text beside line
text(firstXVal,0.003,paste("x=",firstXVal,sep=""),adj=1.1)
text(secondXVal,0.003,paste("x=",secondXVal,sep=""),adj=c(-.1,-.1))
```



PART 3 - KNN

1)

```
## import ISLR library
library(ISLR)
print(ncol(Weekly)) ## num of cols in data set
```

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

```
print(nrow(Weekly)) ## num of rows in data set
```

```
## [1] 1089
```

2)

```
## list all column names from Weekly data set
print(colnames(Weekly))
```

```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

3)

```
## Perform KNN method to predict direction
rm(list=ls())

## Import KNN function (with library) and ISLR
library(class)
library(ISLR)

## will need to normalize data for graph/plots
normalize <- function(x) {
  return( (x - min(x)) / (max(x) - min(x)) )
}

## apply normalization
Weekly_n <- as.data.frame(lapply(Weekly[,c(2,3)], normalize))

## apply the Years column to the new normalized data frame
Weekly_n[,"Year"] <- c(Weekly$Year)
Weekly_n[,"label"] <- rep(NA,1089)

## apply color for labels
Weekly_n[Weekly_n$Lag1>0.5 & Weekly_n$Lag2 >0.5,"label"] = "red"    # Quadrant 1
Weekly_n[Weekly_n$Lag1<0.5 & Weekly_n$Lag2 >0.5,"label"] = "green" # Quadrant 2
Weekly_n[Weekly_n$Lag1<0.5 & Weekly_n$Lag2 <0.5,"label"] = "orange"
Weekly_n[Weekly_n$Lag1>0.5 & Weekly_n$Lag2 <0.5,"label"] = "blue"

## the "get only these" criteria
train = (Weekly_n$Year >= 1990 & Weekly_n$Year <= 2008)
test = (Weekly_n$Year >= 2009 & Weekly_n$Year <= 2010)

## apply the train and test vector to this so we only get what we want
train.X <- cbind(Weekly_n$Lag1,Weekly_n$Lag2)[train,]
test.X = cbind(Weekly_n$Lag1,Weekly_n$Lag2)[test,]

## set the target (cl) to be Direction and test vector to this so we only get what we want
train.target <- Weekly$Direction[train]
test.target <- Weekly$Direction[test]

set.seed(1)
knn.pred = knn(train.X,test.X,train.target,k=5)
table(knn.pred,test.target)
```

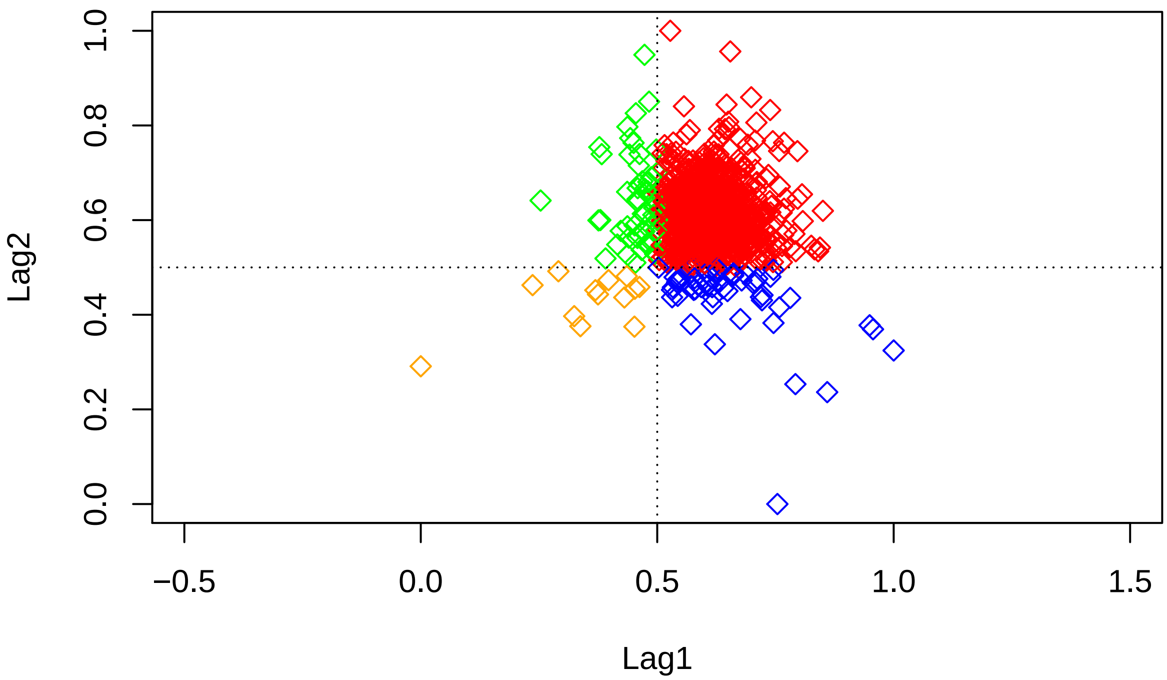
```
##          test.target
## knn.pred Down Up
```

```
##     Down   22 32
##     Up     21 29
```

```
mean(knn.pred == test.target) ##how accurate the prediction was
```

```
## [1] 0.4903846
```

```
plot(Weekly_n[,1:2],col=Weekly_n$label,pch=5,asp=1,xlim=c(0,1),ylim=c(0,1))
abline(h=0.5,lty=3)
abline(v=0.5,lty=3)
```



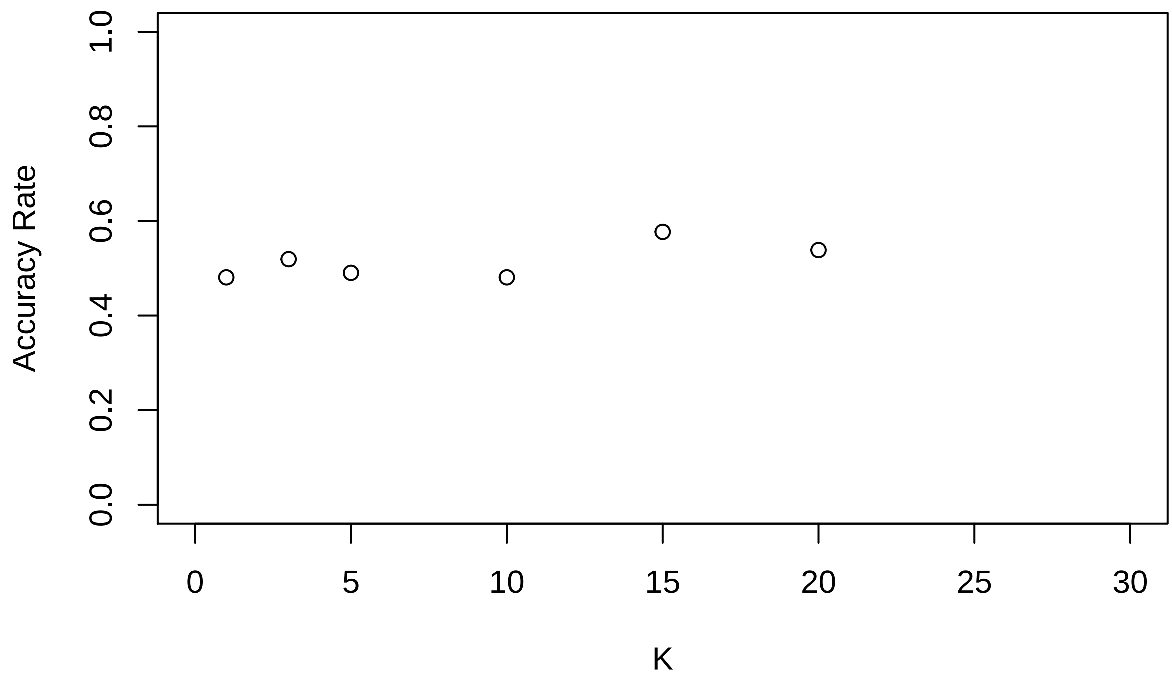
4)

```
## The random seed is set before applying knn() because
## if there are tied observations, we must randomly break the tie using R.
## The seed is set because we'd want to reproduce results!
```

5 AND 6)

```
## draw plot
plot(1, type="n", xlim=c(0, 30), ylim=c(0, 1), xlab="K", ylab="Accuracy Rate")

## plot points for various values of k
for(i in 1:20){
  if(i == 1){
    set.seed(1)
    knn.pred = knn(train.X,test.X,train.target,k=i);
    points(i, mean(knn.pred == test.target));
  }
  else if(i == 3){
    set.seed(1)
```

```
    knn.pred = knn(train.X,test.X,train.target,k=i);
    points(i, mean(knn.pred == test.target));
  }
  else if(i %% 5 == 0){
    set.seed(1)
    knn.pred = knn(train.X,test.X,train.target,k=i);
    points(i, mean(knn.pred == test.target));
  }
} ##end for
```




```
## I would choose k=15, as it has the highest rate of accuracy.
```

7)

```
## Although I've already added normalization, if I didn't add it in
## then it would be the extra needed step because the values generally
## stay smaller and in a more precise decimal value compared to Lag1
## and Lag2 which has sporatic values. I did/would normalize this data
## using feature scaling.
```