hw1

Background

Homework assignments for Math 514 will be prepared using R Markdown. R Markdown is a file format for making dynamic documents with R. R Markdown files support reproducible research and can contain text, equations (latex), R code, images and graphics.

If you are unfamiliar with R Markdown, please review

- Using R Markdown for Class Reports: http://www.stat.cmu.edu/~cshalizi/rmarkdown/
- R Markdown cheat sheet: https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

LaTeX is the most common system for typesetting mathematics. The reference Using R Markdown for Class Reports has a good section on including math in R Markdown. LaTex can be daunting at first, so using a GUI can be a big help. We have found the online GUI CodeCogs to be very helpful. With a little experimentation you should be able to use the GUI to generate LaTex that you then copy/paste back into your R Markdown document.

Homework 1 Problem Statement

Homework 1 is unusual because we have already done it for you - except for the 6 questions at the very end. You should also make sure you understand the R code. Please type in answers for the last section and knit the document for submission along with the .Rmd in the project folder zip as instrument in homework submission guidelines announcement on canvas.

This assignment covers Bayes' Error using a simple 1-dimensional classification problem. The problem is to simulate the tossing of two coins with different probabilities to be heads. An experiment will consist of tossing a randomly chosen coin N times. The problem is to determine (classify) which coin was chosen.

The classifier will be built using Bayes' Rule. Classifier testing data will be generated using simulated coin tosses. The testing data will consist of M experiments (each with N tosses).

Step 1

First, write a function called *coin.toss* to simulate a set of coin toss experiments that allows you to input the probability of heads p.

Your function should take as input

- The probability of success (heads) p.
- The number of tosses in an experiment N.
- The number of experiments to run M.

The function should return the simulated coin toss data as an array with dimensions M x N. That is, each experiment is a row in the returned matrix. Use a value of 1 for head and 0 tail. Use runif function in R to create the data in a vectorized approach.

```r
coin.toss=function(p,num.tosses,num.trials){
  #first generate random numbers between 0 and 1
  #tosses can be generated in a vectorized call producing num_toss*num.trials draws
  draws=runif(num.tosses*num.trials)
  results=matrix(1*(draws < p),nrow=num.trials)
  return(results)
}
```

Step 2

Next, write a second function called *create.coin.toss.data* that uses *coin.toss* to generate the training data. It should take as input:

- N = the number of tosses in an experiment
- M1 = the number of experiments using coin 1
- p1 = the probability of success for coin 1
- M2 = the number of experiments using coin 2
- p2 = the porbability of success for coin 2

The function should return a matrix and a vector. The matrix should be M1+M2 rows by N columns - one row for each experiment. The vector should be M1+M2 long and contain the *tag* for each row in the matrix. Randomize the data on return.

```r
create.coin.toss.data=function(n,m1,p1,m2,p2){
  c1=coin.toss(p1,n,m1)
  c2=coin.toss(p2,n,m2)
  data=rbind(c1,c2)
  y = c(rep(0,m1),rep(1,m2))

  # what do the following lines of code do?
  i = sample(1:nrow(data))
  return(list(x=data[i,],y=y[i]))
}
```

Step 3: Expected Value (try some LaTeX)

Each experiment consists of N 0-1 outcomes each with probability.

Derive the following:

- The expected value and variance of a Bernoulli random variable with $P(X = 1) = \theta$.
- Then compute the expected value and variance of $Y = (1/N)\Sigma_i X_i$ for $i = 1 \dots N$.

$$P(X = x|\theta) = \theta^x(1-\theta)^{(1-x)}$$

where lower case x is $\in$ {0,1} and θ stands in for p1 or p2, depending on the coin. Compute the expected value of this random variable (show calculation in LaTex).

$$E[X] = \sum_i x_i\, P(X = x_i) = 0 \cdot P(X = 0) + 1 \cdot P(X = 1) = 0 \cdot (1-\theta) + 1 \cdot \theta = \theta$$

Now compute the variance of X

$$\sigma^2 = Var(X) = E[(X - E[X])^2] = (0-\theta)^2(1-\theta) + (1-\theta)^2\theta = (1-\theta)(\theta^2 + \theta(1-\theta)) = \theta(1-\theta)$$

Use these results to compute $E[Y]$ and $Var[Y]$

$$E[Y] = E[\frac{1}{N}\Sigma_{i=1}^N X_i] = \frac{1}{N}\Sigma_{i=1}^N(E[X_i]) = \theta$$

$$Var[Y] = Var[\frac{X_1 + X_2 + \cdots X_n}{N}] = \frac{1}{N^2}(Var(X_1) + \cdots + Var(X_N)) = \frac{N}{N^2}Var(X) = \frac{\theta(1-\theta)}{N}$$

http://stat.math.uregina.ca/~kozdron/Teaching/UBC/302Fall10/Handouts/normal_approx_to_bin.pdf

Step 4: Try some graphics

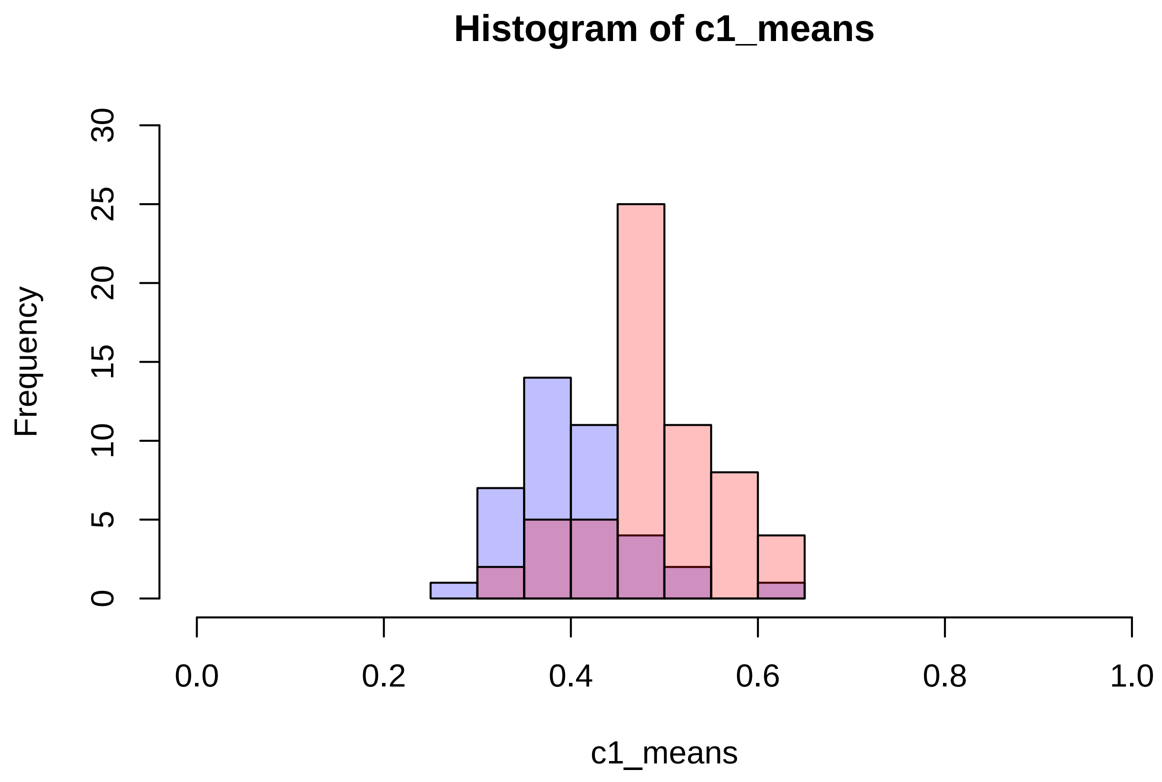
Call your functions to generate 100 experiments with the following parameters:

- n = 64

- m1= 40
- p1= .4
- m2= 60
- p2= .5

Plot histograms of the number of heads for both coins (same plot). Use the multiple assignment operator "%<-%" operator from the *zeallot* package to get data from your function.

```
n=64
m1=40; p1=.4;
m2=60; p2=.5;
c(x,y)%<-%create.coin.toss.data(n,m1,p1,m2,p2)
c2_means=rowMeans(x[y==1,])
c1_means=rowMeans(x[y==0,])
h1=hist(c1_means, col=rgb(0,0,1,1/4), xlim=c(0,1),ylim=c(0,30))
h2=hist(c2_means, col=rgb(1,0,0,1/4), xlim=c(0,1),ylim=c(0,30), add=T)
```



Step 5: Derive Bayes' Equation for classification

Bayes' theorem

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Derive the Bayes' discriminant function for classification:

$$P(C_1|x) < P(C_2|x)$$
$$P(C_1 \mid x) = \frac{P(x \mid C_1)\, P(C_1)}{P(x)}$$
$$\frac{P(x \mid C_1)\, P(C_1)}{P(x)} < \frac{P(x \mid C_2)\, P(C_2)}{P(x)}$$
$$P(x \mid C_1)\, P(C_1) < P(x \mid C_2)\, P(C_2)$$

$$d(x) = P(C_2 \mid x) - P(C_1 \mid x) = P(x \mid C_2)P(C_2) - P(x \mid C_1)P(C_1) \Rightarrow \begin{cases} \geq 0 & C_2 \\ < & C_1 \end{cases} \quad (1)$$

Step 6: Use normal approximation of binomial distribution

Write a discriminant funtion (equation 1 above) using normal approximations (see dnorm function in R) for the class-conditional probabilities. The function should take as inputs

- x value
- N = the number of tosses in an experiment
- M1 = the number of experiments using coin 1
- p1 = the probability of success for coin 1
- M2 = the number of experiments using coin 2
- p2 = the porbability of success for coin 2

This function will be ≥ 0 if class 2 is more likely. The optimial cut point is when the discriminant is equal to 0. Write a function (using a root finder) to compute the optimal Bayes' cut (again with normal approximations).

```
# value > 0 --> class 2
#P(X|C) is approximated by normal denisty from dnorm and P(C) is approximated using m1/(m1+m2) and sinc
bayes.discriminant=function(x,n,m1,p1,m2,p2){
  # compute binomial standard deviation - use for normal approximation
  s1=sqrt(p1*(1-p1)/n)
  s2=sqrt(p2*(1-p2)/n)
  # don't need to divide by (m1+m2), just care about positive/negative
  dnorm(x,p2,s2)*m2 - dnorm(x,p1,s1)*m1
}

# use a root finder to determine where descriminant is zero
# type ?uniroot to see what function documentation says about uniroot and note typing the function show
bayes.cut.norm=function(n,m1,p1,m2,p2){
  ff=function(x){bayes.discriminant(x,n,m1,p1,m2,p2)}
  uniroot(ff,c(p1,p2))$root
}
```

Step 7: Compute Bayes' Error

Use the bayes' optimal cut to compute the theoretical bayes' error for the bayes' decision rule. The Bayes' error is the sum of the areas of overlap weighted by their class probabilities. Use the R function *pnorm* to compute needed tails of normal approximations to the class conditional probabilities.

```
bayes.error=function(cut,n,m1,p1,m2,p2){
  s1=sqrt(p1*(1-p1)/n)
  s2=sqrt(p2*(1-p2)/n)
  w1=m1/(m1+m2)
  w2=1-w1
  if(p1 < p2){
    error=w2*pnorm(cut,p2,s2)+w1*(1-pnorm(cut,p1,s1))
  }else{
    error=w1*pnorm(cut,p1,s1)+w2*(1-pnorm(cut,p2,s2))
  }
  error
}
```

Step 8 Classify new experiments using Bayes' cut

Compute the bayes' cut and the Bayes' error. Use the cut to evaluate classification performance on new data. Run 30 trials and plot performance against the Bayes' error.

```
cut=bayes.cut.norm(n,m1,p1,m2,p2)
cut
```

```
## [1] 0.4347043
```

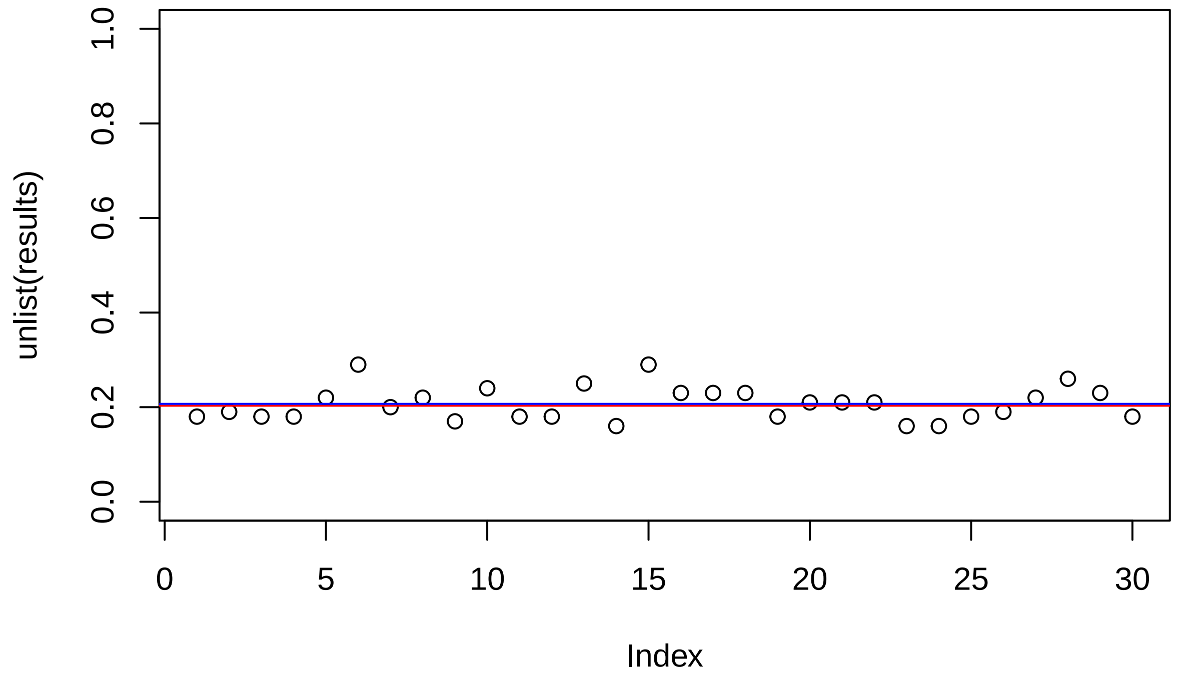
```
b.err=bayes.error(cut,n,m1,p1,m2,p2)
b.err
```

```
## [1] 0.2030252
```

```
bayes.1d.accuracy=function(X,y,cut){
  sum(as.integer(rowMeans(X) > cut)==y)/length(y)
}

results=list(mode='vector')

for (i in 1:30){
  c(x,y) %<-% create.coin.toss.data(n,m1,p1,m2,p2)
  results[[i]]=1-bayes.1d.accuracy(x,y,cut)
}
plot(unlist(results),ylim=c(0,1));
abline(h=b.err,col='red');
abline(h=mean(unlist(results)),col='blue')
```



Step 9

Provide text answers to the following questions

- Q1 Step 1: In the function coin.toss, why is (draws < p) multiplied by 1?

  Converts the vector of `T/F` into numeric 1 or 0.

- Q2 Step 2: In the function create.coin.toss.data, explain what the last two lines accomplish, other than returning the data to the caller

  `sample(1:nrow( data ) )` randomizes the vector of numbers from 1 to $(m1 + m2)$. Then the next line returns a list; the first component is the matrix of coin tosses which is now randomized according to `i`. The second component a vector tracking what coin each row belongs to (because they were shuffled according to `i`).

- Q3 Step 4: In the histograms, what makes the colors translucent?

  The alpha level at $\frac{1}{4}$.

```r
rgb
```

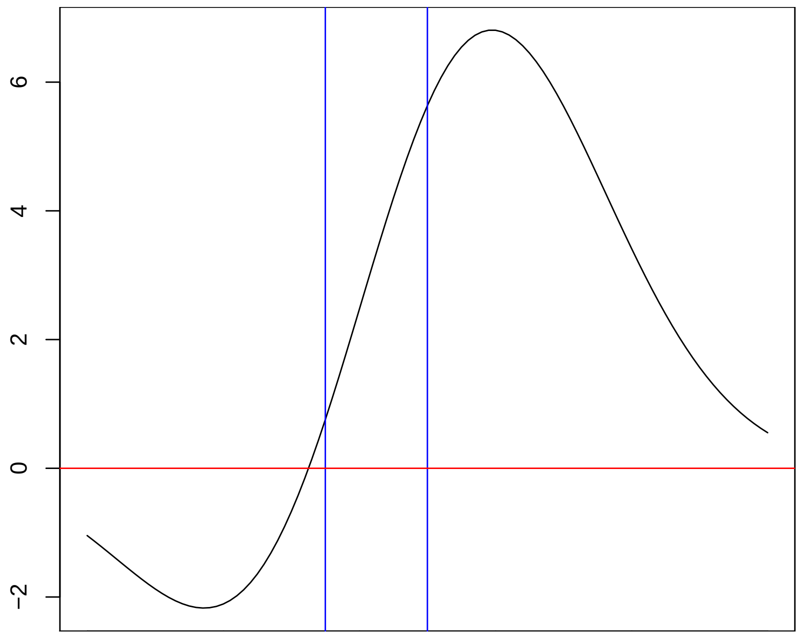
```
## function (red, green, blue, alpha, names = NULL, maxColorValue = 1)
## {
##     if (missing(green) && missing(blue)) {
##         if (is.matrix(red) || is.data.frame(red)) {
##             red <- data.matrix(red)
##             if (ncol(red) < 3L)
##                 stop("at least 3 columns needed")
##             green <- red[, 2L]
##             blue <- red[, 3L]
##             red <- red[, 1L]
##         }
##     }
##     .Call(C_rgb, red, green, blue, if (missing(alpha)) NULL else alpha,
##         maxColorValue, names)
## }
## <bytecode: 0x0000000014a528a8>
## <environment: namespace:grDevices>
```

- Q4 Step 6: In the function bayes.cut.norm, what does the argument c(p1,p2) to uniroot do and why is it a reasonable choice?

  It sets an interval for the uniroot function to find where the function is equal to 0. This is reasonable because $d(x) = 0$ should be between p1 and p2. However, it may not always work.

```r
n=6 ;  m1=4; p1=.35;  m2=6; p2=.5; par(mar = c(0,4,0,2) )

curve( dnorm(x,p2,sqrt(p2*(1-p2)/n))*m2 - dnorm(x,p1,sqrt(p1*(1-p1)/n))*m1 , ylab='')
abline( v = c(p1,p2) , col = "blue");  abline( h = 0 , col = "red" )
```



We can play around with the parameters, and the `uniroot` function will fail. However, in this case it looks relatively easy to solve. We can use the `optimize` function to get the interval to be the minimum on the left and the maximum on the right.

- Q5 Step 7: In the function bayes.error, why is there a check for p1 $<$ p2?

  The check for `p1 < p2` determines that we are going to get the correct area under the curves. We make sure the location parameters are in the correct order so we get the appropriate distribution tails.

- Q6 Step 8: The function bayes.1d.accuracy is very compact. Explain the steps to computing accuracy.

  We take the row means of the randomized row outcome matrix. We compare each mean to the Bayes' cut. This returns an `F/T` vector, but we use `as.integer` to convert it into numeric 0/1. Then we compare it to the "truth" vector, the vector that tracked what coin each row belonged to. Then we sum this truth comparison vector (with `T = 1`) and divide it by the total number of experiments conducted. This gives the accuracy as a percentage of how many experiments the Bayes classifier had correct.