STA6703 SML Take-Home Prelim, Fall 2022

Christopher Marais

Helping functions

```
myRound <- function(x, acc=3) {mult = 10^acc; round(x*mult)/mult}
```

Problem 1

```
set.seed(0)
n = 100
m=1000

origData = rnorm(n) # case 1
z=sample(x=origData, size=n*m, replace=TRUE)
case1_mat <- matrix(z,nrow=m)

origData = rt(n,df=3); # case 2
z=sample(x=origData, size=n*m, replace=TRUE)
case2_mat <- matrix(z,nrow=m)

origData = rt(n,df=25); # case 3
z=sample(x=origData, size=n*m, replace=TRUE)
case3_mat <- matrix(z,nrow=m)
```

Problem 2

```
myCVids <- function(n, K, seed=0) {
# balanced subsets generation (subset sizes differ by at most 1)
# n is the number of observations/rows in the training set
# K is the desired number of folds (e.g., 5 or 10)
set.seed(seed);
t = floor(n/K); r = n-t*K;
id0 = rep((1:K),times=t)
ids = sample(id0,t*K)
if (r > 0) {ids = c(ids, sample(K,r))}
ids
}
```

```r
# Generate data
set.seed(0)
nr = 50
nc = 200*2^5 # nc = 6400
M = matrix(rnorm(nr*nc),nrow=50)
Y = c(rep(1,25),rep(0,25))
```

Problem 3

Problem 4

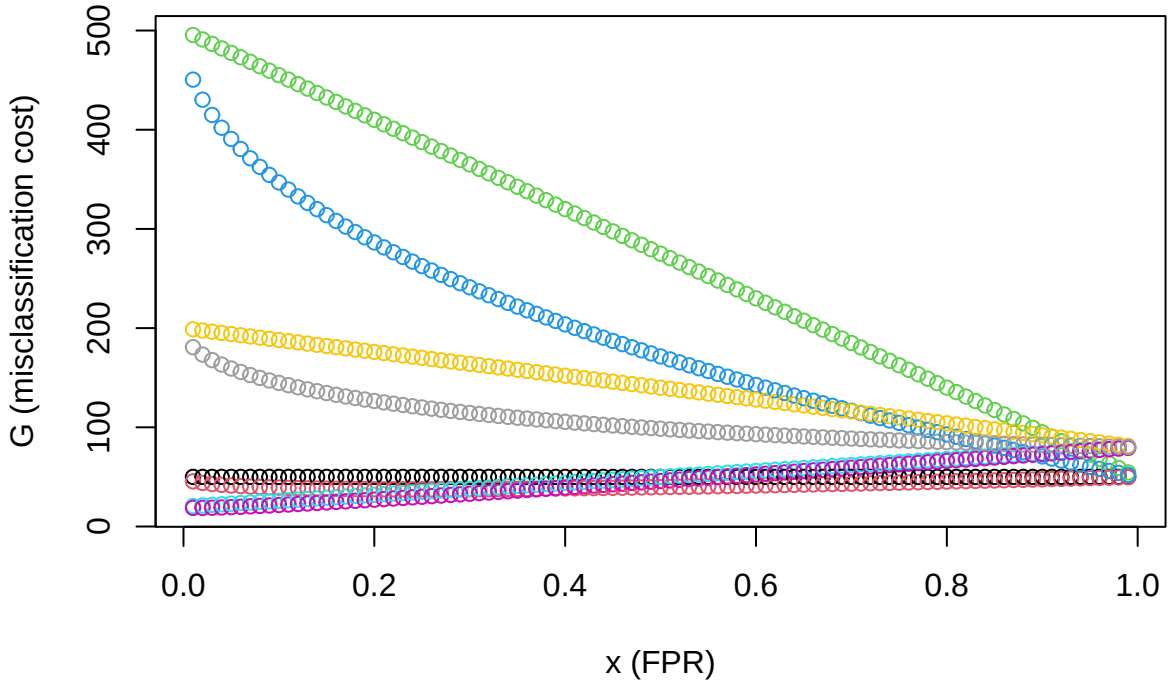
```r
genData <- function(n, seed=0) {
set.seed(seed)
x = seq(-1,1,length.out=n)
y = x - x^2 + 2*rnorm(n) # true sigma = 2;
out.df = data.frame(x=x, y=y)
out.df
}
train.df = genData(n=200,seed=100)
test.df = genData(400)
```

Problem 5

```r
# assume values for x and cfp
# x = 0.25
cfp=1
n=100


x_vec=c()
A_vec=c()
C_vec=c()
R_vec=c()
FPR_vec=c()
TPR_vec=c()
G_vec=c()
for (x in seq(0.01,0.99,0.01)) {
  A=c(0.5, 0.2)
  C=c(cfp, 10*cfp)
  R=c(x, sqrt(x))

  Ai=0
  for(q in A){
    Ai = Ai+1
```

```
    P=n*q
    N=n*(1-q)

    Ri=0
    for(tpr in R){
      Ri=Ri+1
      # calculate TP, FP, TN, and FN with regards to x
      TP = tpr*P
      FN = P-TP
      FP = x*N
      TN = N-FP
      FPR = x
      TPR = tpr

      Ci=0
      for(cfn in C){
        Ci=Ci+1
        G = cfn*FN + cfp*FP

        x_vec=c(x_vec,x)
        A_vec=c(A_vec,Ai)
        C_vec=c(C_vec,Ci)
        R_vec=c(R_vec,Ri)
        FPR_vec=c(FPR_vec,FPR)
        TPR_vec=c(TPR_vec,TPR)
        G_vec=c(G_vec,G)

        # print(paste("(A:",as.character(Ai),")"))
        # print(paste("(C:",as.character(Ci),")"))
        # print(paste("(R:",as.character(Ri),")"))
        # print(paste("FPR = ", FPR))
        # print(paste("TPR = ", TPR))
        # print(paste("G = ", G))
        # print("----------------------------")


      }
    }

  }
}

results_df = data.frame(x_vec,
                        A_vec,
                        C_vec,
                        R_vec,
                        FPR_vec,
                        TPR_vec,
                        G_vec)
```

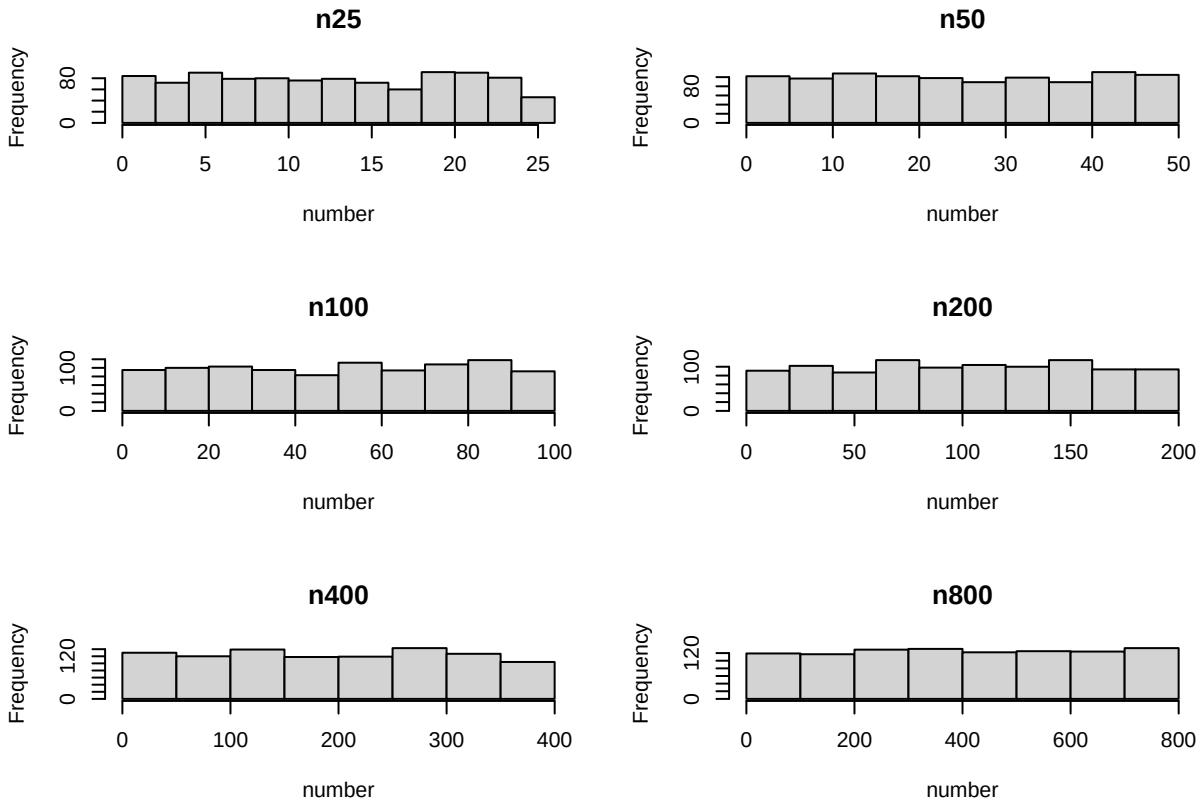
```
results_df$design_id <- paste(results_df$A_vec,
                              results_df$C_vec,
                              results_df$R_vec)

# Objective function score visualization
plot(x=results_df$x_vec,
     y=results_df$G_vec,
     col=factor(results_df$design_id),
     ylab="G (misclassification cost)",
     xlab="x (FPR)")
```



The best classifier is the one that minimizes the objective function the best over values of x. In this case it was one with the design combinations of (A:1 C1: R:2) or (A:2 C1: R:2). For higher values of x an unbalanced population gives higher values from the objective function. A1 is thus better with even populations when. This is because with an uneven class ratio the mis-classification rate increases for higher values of x because a higher x value means a higher false positive count. Even when the false negative count is low. THis classifier is therefore not stable for all values of x and we choose the (A:1 C1: R:2) combination over the (A:2 C1: R:2) combination. Even though the (A:2 C1: R:2) combination clearly perfroms better at lower values of x. This in turn increased the mis-classifications calculated in the objective function. All classifiers that had a cfn that was 10 times the cfp resulted in an objective function that was orders of magnitude larger than the other classifiers. This makes intuitive senses as it dramatically increases the net cost of mis-classifications. With a TPR that is square rooted the objective function consistently returns a lower mis-classifications score. R2 is thus better. This is because, when the square root of the TPR is used to derive True positives they are higher than than when the TPR is not square rooted. This in turn decreases the amount of mis-classifications.

Problem 6

```
# Generate data
set.seed(0)
n_vec = c(25, 50, 100, 200, 400, 800)
m = 1000
df = data.frame(m=1:m)
table_df = data.frame(n=n_vec) # results table
mean_uniq_vec = c()
sd_uniq_vec = c()
for (n in n_vec){
  I = seq(n)
  samp_vec = c(sample(x=I, size=m ,replace=TRUE))
  df[paste("n",n , sep="")] = samp_vec

  mean_uniq_vec = c(mean_uniq_vec, mean(unique(samp_vec)))
  sd_uniq_vec = c(sd_uniq_vec, sd(unique(samp_vec)))
}

#visualize data
par(mfrow=c(3,2))
for(i in names(df)[2:7]){
  hist(df[[i]],
       xlab = "number",
       main=i)
}
```




```
# make table of results
table_df$Mean_Unique = mean_uniq_vec
table_df$SD_Unique = sd_uniq_vec
table_df = data.frame(t(table_df))

knitr::kable(table_df, format = "markdown")
```

	X1	X2	X3	X4	X5	X6
n	25.000000	50.00000	100.00000	200.00000	400.0000	800.0000
Mean_Unique	13.000000	25.50000	50.50000	100.74874	200.0192	408.8354
SD_Unique	7.359801	14.57738	29.01149	57.91789	116.2927	230.7809