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
```

```
}

# two-sample t test
column_t_test <- function(features, target){
  tests = lapply(seq(1,ncol(features)),function(x){t.test(features[,x]~target)})
  pval_lst = lapply(seq(1,length(tests)),function(x){tests[[x]]$p.value})
  return(list(tests, pval_lst))
}


# keep features with p-value <= 0.05
top_features <- function(pval_lst, p_val, p_min){
  bool_lst = lapply(seq(1,length(pval_lst)),function(x){pval_lst[[x]] < p_val})
  if(sum(as.integer(bool_lst))<=p_min){
    bool_df=data.frame(pval_sig=matrix(unlist(bool_lst), nrow=length(bool_lst), byrow=TRUE))
    selected_df = bool_df[bool_df$pval_sig==TRUE,]
    feat_index_vec = as.numeric(rownames(selected_df))
    return(feat_index_vec)
  }else{
    pval_df = t(data.frame(pval_lst))
    row.names(pval_df) <- NULL
    bool_df=data.frame(pval_sig=matrix(unlist(bool_lst), nrow=length(bool_lst), byrow=TRUE))
    bool_df$p_val <- pval_df[,1]
    bool_df$p_top = FALSE
    sorted_res = sort(bool_df$p_val, index.return=TRUE)
    bool_df[head(sorted_res$ix,p_min),]$p_top = TRUE
    bool_df$p_select = as.logical(bool_df$pval_sig*bool_df$p_top)
    selected_df = bool_df[bool_df$p_select==TRUE,]
    feat_index_vec = as.numeric(rownames(selected_df))
  }
  return(feat_index_vec)

}

# Mis-classification ratio calculation
MCR <- function(true_vals, pred_probs, threshold=0.5){
  if(length(true_vals)!=length(pred_probs)){
    print("ERROR: predictions and true values not of same shape")
  }else{
    pred_vals = as.integer((pred_probs > threshold))
    mcr = sum(pred_vals != true_vals)/length(true_vals)
    return(mcr)
  }
}
```

```
# Generate data
set.seed(0) # set seed
n = 25 # number of samples in each class
nr = n*2 # total number of samples
```

```
Y = c(rep(1,n),rep(0,n)) # target values
k=10 # k value for cross validation
p_star_vec = c(5,10,20,40) # p* values to select the top features in the data
i_vec = seq(5) #different values of i for different sized data sets

# create matrix where results will be stored
mcr_i_pstar_df = data.frame(matrix(0,
                                   nrow = length(p_star_vec),
                                   ncol = length(i_vec)+1))
mcr_i_pstar_df[,1] = p_star_vec # add p* values to results table

# loop over values of i to make different data sets
for(i in i_vec){
  nc = 200*2^i # nc = 6400
  M = matrix(rnorm(nr*nc),nrow=nr)
  X = M[,1:nc] # features of data
  # calculate cross validation indexes
  # this is applied only after feature selection
  ids = myCVids(n=nr, K=k, seed=0)
  # feature selection
  # apply two-sample t-test
  t_test_results = column_t_test(features=X, target=Y)
  # get all features according to p-value at differing values of p_star
  # create vector to store mean mcr values for each p* subset
  mean_pstar_mcr_vec = c()

  # loop over p* values to create a subset of selected features
  for(p_star in p_star_vec){
    feat_index_vec = c(top_features(pval_lst=t_test_results[[2]],
                                    p_val=0.05,
                                    p_min=p_star))
    print(feat_index_vec)

    X_pstar = X[,feat_index_vec]
    k_mcr_vec = c() # store all mcr values for each k
    # loop over k to calculate the MCR for each fold
    for( k in seq(k)){
      isk = (ids == k) # k varies from 1 to K
      valid.k = which(isk) # test data index
      train.k = which(!isk) # train data index
      # get all training data in single data frame
      train_df = data.frame(Y=Y[train.k], X_pstar[train.k,])
      # get all testing data in single data frame
      val_df = data.frame(Y=Y[valid.k], X_pstar[valid.k,])

      # train a Logistic regression model
      LR = glm(Y ~ .,
               data=train_df,
               family="binomial")
```

```
      # get estimated probabilities on the test data
      LR_probs = data.frame(
                    predict(LR,
                    val_df,
                    type ="response"
                    )
                  )

      # use the probabilities to calculate the MCR
      mcr = MCR(
              true_vals=val_df$Y,
              pred_probs=LR_probs[,1],
              threshold=0.5)
      k_mcr_vec = c(k_mcr_vec, mcr)
    }
    mean_pstar_mcr = mean(k_mcr_vec) # get the mean MCR for all values of k

    mean_pstar_mcr_vec = c(mean_pstar_mcr_vec, mean_pstar_mcr)
  }
  mcr_i_pstar_df[,i+1] = mean_pstar_mcr_vec
}
```

```
## [1]  88 158 189 221 259
## numeric(0)
## numeric(0)
## numeric(0)
## [1]  74 131 244 259 705
##  [1]  69  74 131 244 254 259 325 616 665 705
##  [1]  69  70  74  95 110 114 117 131 244 254 259 325 505 525 616 661 665 705 721
## [20] 729
## numeric(0)
## [1]   74  131  244  976 1266
##  [1]   74  131  244  259  920  956  976 1266 1469 1487
##  [1]   69   74  131  244  259  325  616  665  705  812  920  956  976 1020 1068
## [16] 1133 1189 1266 1469 1487
##  [1]   69   70   74  114  131  244  254  259  325  505  525  616  665  705  729
## [16]  812  893  920  956  969  976 1020 1068 1133 1143 1153 1160 1189 1266 1314
## [31] 1334 1336 1358 1381 1394 1440 1442 1469 1487 1562
## [1]   74  244 1266 1876 2916
##  [1]   74  131  244  956  976 1266 1469 1876 2187 2916
##  [1]   74  131  244  259  705  920  956  976 1068 1189 1266 1469 1487 1782 1876
## [16] 2071 2110 2187 2444 2916
##  [1]   69   74  131  244  254  259  325  616  665  705  812  920  956  969  976
## [16] 1020 1068 1133 1189 1266 1314 1469 1487 1638 1685 1782 1876 1912 1992 2071
## [31] 2110 2176 2187 2402 2444 2623 2666 2701 2814 2916
## [1]   74 1266 1876 2916 4749
##  [1]   74  131  244 1266 1876 2916 4749 4956 5609 6186
##  [1]   74  131  244  259  956  976 1266 1469 1487 1876 2071 2110 2187 2444 2916
## [16] 4037 4749 4956 5609 6186
```

```
##  [1]   69   74  131  244  259  325  616  665  705  812  920  956  976 1020 1068
## [16] 1133 1189 1266 1469 1487 1685 1782 1876 1992 2071 2110 2176 2187 2444 2916
## [31] 4037 4651 4693 4749 4760 4956 5016 5430 5609 6186
```

```
# add all data to a data frame and transform to be in the correct format
mcr_i_pstar_df = t(mcr_i_pstar_df)
rownames(mcr_i_pstar_df) = c("p*", i_vec)
knitr::kable(mcr_i_pstar_df, format = "markdown") # display table
```

p*	5.00	10.00	20.00	40.00
1	0.30	0.70	0.70	0.70
2	0.24	0.16	0.20	0.70
3	0.16	0.10	0.10	0.26
4	0.18	0.06	0.04	0.18
5	0.16	0.02	0.04	0.30

Problem 3

Problem 4

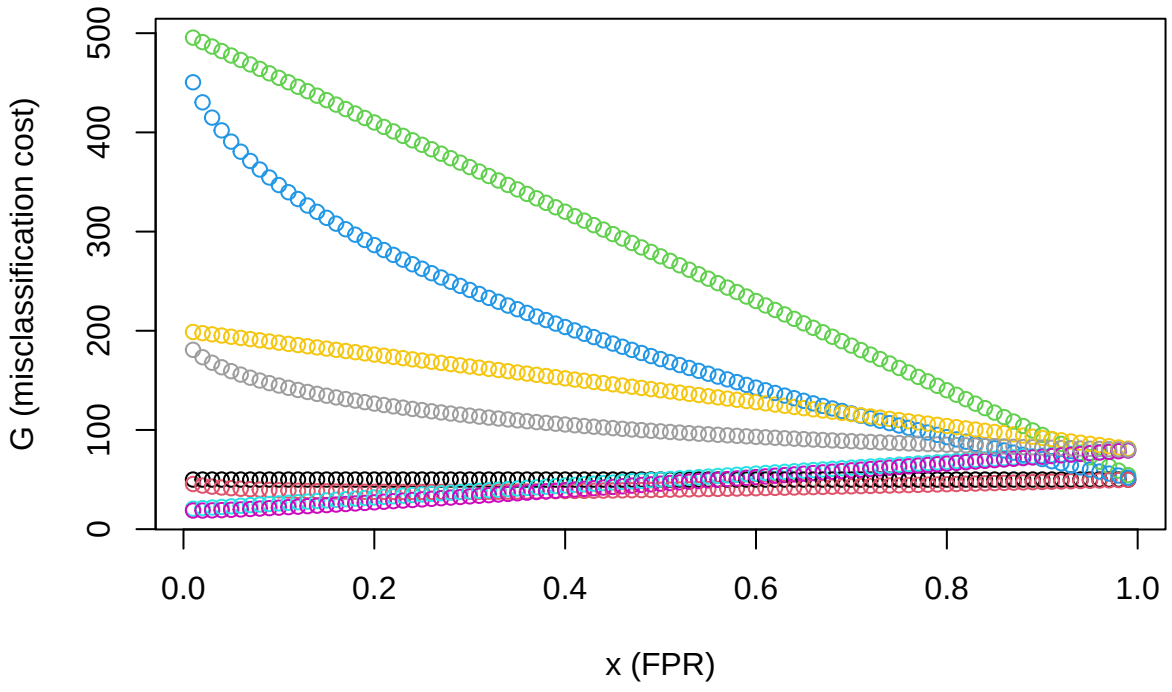
```
genData <- function(n, seed=0) {
set.seed(seed)
x = seq(-1,1,length.out=n)
y = x - x^2 + 2*rnorm(n) # true sigma = 2;
out.df = data.frame(x=x, y=y)
out.df
}
train.df = genData(n=200,seed=100)
test.df = genData(400)
```

Problem 5

```
# 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)")
```

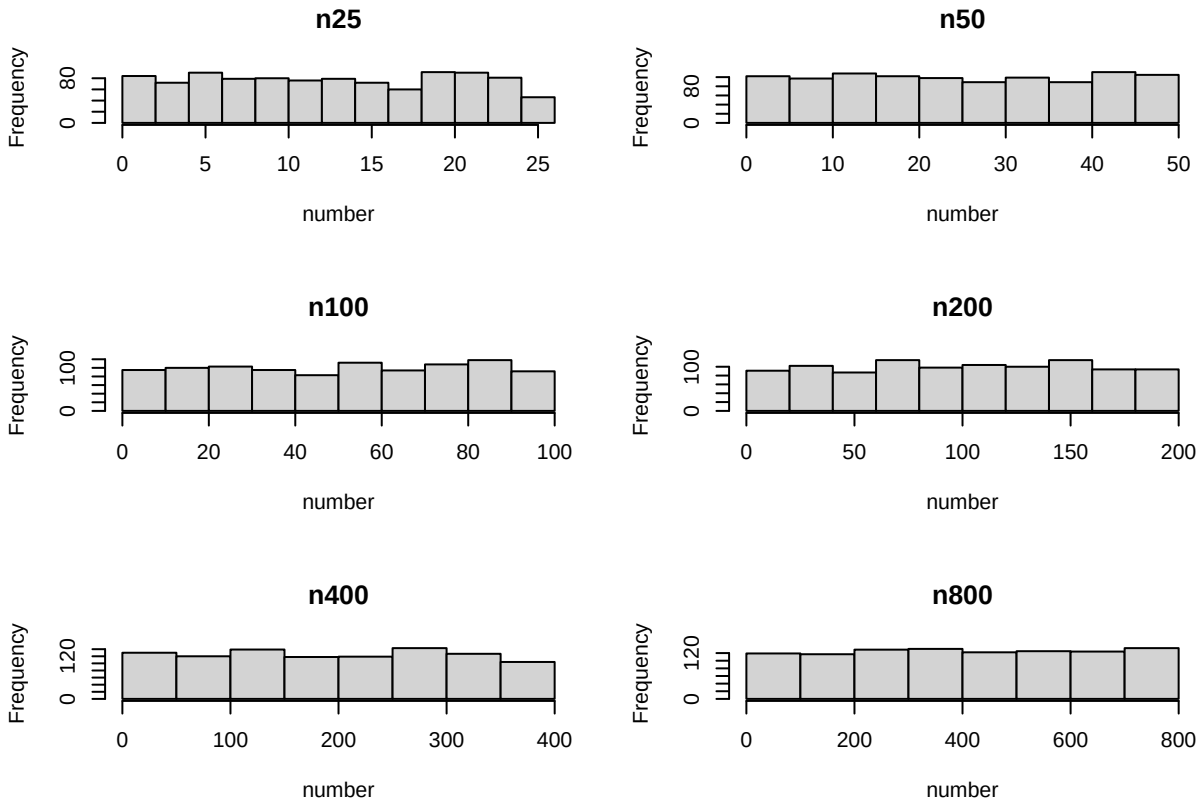


The best classifier is the one that minimizes the objective function the best over values of x. In this case it was one with the design combinations of (A:1 C1: R:2) or (A:2 C1: R:2). For higher values of x an unbalanced population gives higher values from the objective function. A1 is thus better with 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 performs 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$Mean_Unique = myRound(mean_uniq_vec, acc=2)
table_df$SD_Unique = myRound(sd_uniq_vec, acc=2)
table_df = data.frame(t(table_df))

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

	X1	X2	X3	X4	X5	X6
n	25.00	50.00	100.00	200.00	400.00	800.00
Mean_Unique	13.00	25.50	50.50	100.75	200.02	408.84
SD_Unique	7.36	14.58	29.01	57.92	116.29	230.78