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){
    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$p_val <- pval_df[,1]
    selected_df = bool_df[bool_df$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)
  }
}
```

```r
# 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))

    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
}
# 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	0.26	0.26	0.26
2	0.24	0.16	0.20	0.32
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

```
# 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
```

```r
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
progress=0
# 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)

  # 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){
    k_mcr_vec = c() # store all mcr values for each k

    # k-fold cross validation
    # loop over k to calculate the MCR for each fold
    for( k in seq(k)){
      progress=progress+1
      # print(myRound(progress/200)*100)
      isk = (ids == k) # k varies from 1 to K
      valid.k = which(isk) # test data index
      train.k = which(!isk) # train data index

      # feature selection
      # apply two-sample t-test
      t_test_results = column_t_test(features=X[train.k,], target=Y[train.k])
      feat_index_vec = c(top_features(pval_lst=t_test_results[[2]],
                                      p_val=0.05,
                                      p_min=p_star))
      X_pstar = X[,feat_index_vec]

      # 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
}
# 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	0.46	0.48	0.48	0.46
2	0.48	0.48	0.52	0.62
3	0.38	0.36	0.32	0.48
4	0.38	0.42	0.32	0.42
5	0.38	0.48	0.56	0.48

Note: it takes much longer this way... like waayyy longer more data = worse generalization with problem 2 but not 3.

Problem 4

```
# functions
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
}

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
}
```

4.1

```
# parameters
set.seed(100)
d_vec = seq(0,4)
# generate data
train.df = genData(n=200,seed=100)
test.df = genData(400)

# create k-fold indexes
k=5
inds.part = myCVids(n=nrow(train.df),K=k)
# create a data frame to save results into
M = matrix(0, nrow = (length(d_vec)), ncol = k+1)
KFOLD_df = data.frame(M)
KFOLD_df[,1] = d_vec # add degrees to results table
colnames(KFOLD_df) = c("Degree", seq(k))

# loop over k-folds
for( k in seq(k)){
  isk = (inds.part == k) # k varies from 1 to K
  valid.k = which(isk) # test data index
  train.k = which(!isk)
  # split data
  train_sub_df = train.df[train.k,]
  valid_df = train.df[valid.k,]

  d_mse_vec = c()
  for(d in d_vec){
```

```
    if(d==0){
      # train a polynomial model
      PR = lm(y ~ 1, data=train_sub_df)
    }else{
     # train a polynomial model
      PR = lm(y ~ poly(x, d, raw = TRUE), data=train_sub_df)
    }
    # get predicted values to calculate MSE
    pred_val = predict(PR, valid_df, type="response")
    pred_true_val_df = data.frame(pred = pred_val, actual = valid_df$y)
    #calculate MSE
    mse = mean((pred_true_val_df$actual - pred_true_val_df$pred)^2)
    d_mse_vec = c(d_mse_vec, mse)
  }
  KFOLD_df[,k+1] = d_mse_vec
}

# tally of MSE
KFOLD_df$Total_MSE = rowSums(KFOLD_df)
knitr::kable(KFOLD_df, format = "markdown") # display table
```

Degree	1	2	3	4	5	Total_MSE
0	2.533138	2.984333	4.243087	4.513501	4.206231	18.48029
1	2.228435	2.853997	3.653187	4.190807	3.876836	17.80326
2	2.204849	2.797685	3.639950	4.126926	3.967901	18.73731
3	2.250757	2.872660	3.635394	4.190780	3.961687	19.91128
4	2.241189	2.894667	3.626206	4.490320	3.955490	21.20787

```
# print best d value
print(paste("The best K-fold degree is: ",
            as.character(KFOLD_df[which.min(KFOLD_df$Total_MSE),]$Degree)))
```

```
## [1] "The best K-fold degree is:  1"
```

4.2

```
# create k-fold indexes
k=nrow(train.df)
inds.part = myCVids(n=nrow(train.df),K=k)
# create a data frame to save results into
M = matrix(0, nrow = (length(d_vec)), ncol = k+1)
LOOCV_df = data.frame(M)
LOOCV_df[,1] = d_vec # add degrees to results table
colnames(LOOCV_df) = c("Degree", seq(k))
```

```
# loop over k-folds
for( k in seq(k)){
  isk = (inds.part == k) # k varies from 1 to K
  valid.k = which(isk) # test data index
  train.k = which(!isk)
  # split data
  train_sub_df = train.df[train.k,]
  valid_df = train.df[valid.k,]

  d_mse_vec = c()
  for(d in d_vec){
    if(d==0){
      # train a polynomial model
      PR = lm(y ~ 1, data=train_sub_df)
    }else{
     # train a polynomial model
      PR = lm(y ~ poly(x, d, raw = TRUE), data=train_sub_df)
    }
    # get predicted values to calculate MSE
    pred_val = predict(PR, valid_df, type="response")
    pred_true_val_df = data.frame(pred = pred_val, actual = valid_df$y)
    #calculate MSE
    mse = mean((pred_true_val_df$actual - pred_true_val_df$pred)^2)
    d_mse_vec = c(d_mse_vec, mse)
  }
  LOOCV_df[,k+1] = d_mse_vec
}

# tally of MSE
LOOCV_df$Total_MSE = rowSums(LOOCV_df)
knitr::kable(LOOCV_df[,c(1,k+2)], format = "markdown") # display table
```

Degree	Total_MSE
0	740.7244
1	677.5361
2	679.7304
3	686.7229
4	693.2117

```
# print best d value
print(paste("The best LOOCV degree is: ",
            as.character(LOOCV_df[which.min(LOOCV_df$Total_MSE),]$Degree)))
```

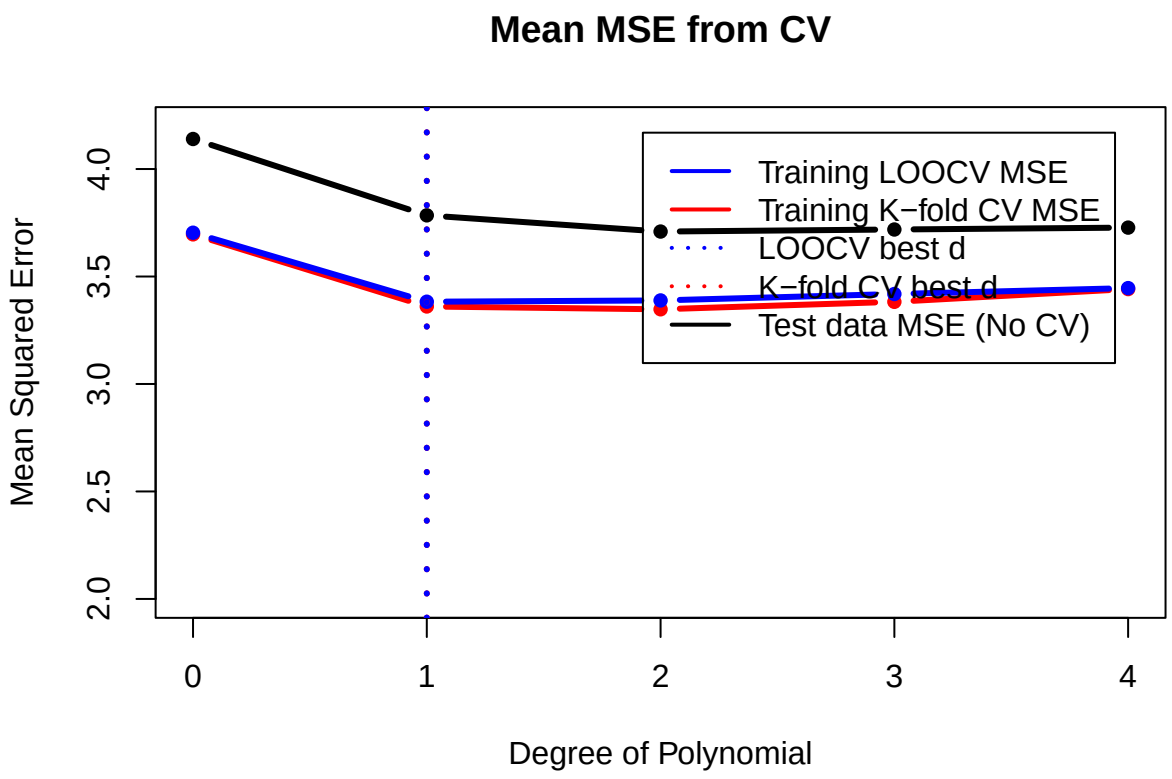
```
## [1] "The best LOOCV degree is:  1"
```

There is a part in lectures where it says that LOOCV is the same as k-fold for polynomial regression as a special case.

4.3

```
# estimate test data MSE
test_d_vec = c()
for(d in d_vec){
  if(d==0){
    # train a polynomial model
    PR = lm(y ~ 1, data=train.df)
  }else{
   # train a polynomial model
    PR = lm(y ~ poly(x, d, raw = TRUE), data=train.df)
  }
  # get predicted values to calculate MSE
  pred_val = predict(PR, test.df, type="response")
  pred_true_val_df = data.frame(pred = pred_val, actual = test.df$y)
  #calculate MSE
  mse = mean((pred_true_val_df$actual - pred_true_val_df$pred)^2)
  test_d_vec = c(test_d_vec, mse)
}

# visualize LOOCV and K-fold CV
# calculate max and min limits of data KFOLD
kfold_max_vec = apply(KFOLD_df[, 2:(ncol(KFOLD_df)-1)], 1, max)
kfold_min_vec = apply(KFOLD_df[, 2:(ncol(KFOLD_df)-1)], 1, min)
kfold_mean_vec = apply(KFOLD_df[, 2:(ncol(KFOLD_df)-1)], 1, mean)
# calculate max and min limits of data for LOOCV
loocv_max_vec = apply(LOOCV_df[, 2:(ncol(LOOCV_df)-1)], 1, max)
loocv_min_vec = apply(LOOCV_df[, 2:(ncol(LOOCV_df)-1)], 1, min)
loocv_mean_vec = apply(LOOCV_df[, 2:(ncol(LOOCV_df)-1)], 1, mean)

{plot(x=KFOLD_df$Degree,
     y=kfold_mean_vec,
     ylab="Mean Squared Error",
     main="Mean MSE from CV",
     xlab="Degree of Polynomial",
     type='b',
     col='red',
     pch = 16,
     lwd=3,
     ylim=c(2,4.2))
lines(x=LOOCV_df$Degree,
     y=loocv_mean_vec,
     type='b',
     col='blue',
     pch = 16,
     lwd=3)
lines(x=LOOCV_df$Degree,
     y=test_d_vec,
     type='b',
```

```
      col='black',
      pch = 16,
      lwd=3)
abline(v=KFOLD_df[which.min(KFOLD_df$Total_MSE),]$Degree,
      col='red',
      pch = 16,
      lty=3,
      lwd=3)
abline(v=LOOCV_df[which.min(LOOCV_df$Total_MSE),]$Degree,
      col='blue',
      pch = 16,
      lty=3,
      lwd=3)
legend("topright",
       inset = 0.05,
       legend = c("Training LOOCV MSE", "Training K-fold CV MSE", "LOOCV best d", "K-fold CV best
       lty = c(1,1,3,3),
       col = c("blue", "red","blue", "red", "black"),
       lwd = 2)}
```



4.4

```r
# parameters
set.seed(100)
d_vec = seq(0,4)
# generate data
train.df = genData(n=400,seed=100)
test.df = genData(400)

# create k-fold indexes
k=5
inds.part = myCVids(n=nrow(train.df),K=k)
# create a data frame to save results into
M = matrix(0, nrow = (length(d_vec)), ncol = k+1)
KFOLD_df = data.frame(M)
KFOLD_df[,1] = d_vec # add degrees to results table
colnames(KFOLD_df) = c("Degree", seq(k))

# loop over k-folds
for( k in seq(k)){
  isk = (inds.part == k) # k varies from 1 to K
  valid.k = which(isk) # test data index
  train.k = which(!isk)
  # split data
  train_sub_df = train.df[train.k,]
  valid_df = train.df[valid.k,]

  d_mse_vec = c()
  for(d in d_vec){
    if(d==0){
      # train a polynomial model
      PR = lm(y ~ 1, data=train_sub_df)
    }else{
     # train a polynomial model
      PR = lm(y ~ poly(x, d, raw = TRUE), data=train_sub_df)
    }
    # get predicted values to calculate MSE
    pred_val = predict(PR, valid_df, type="response")
    pred_true_val_df = data.frame(pred = pred_val, actual = valid_df$y)
    #calculate MSE
    mse = mean((pred_true_val_df$actual - pred_true_val_df$pred)^2)
    d_mse_vec = c(d_mse_vec, mse)
  }
  KFOLD_df[,k+1] = d_mse_vec
}

# tally of MSE
KFOLD_df$Total_MSE = rowSums(KFOLD_df)
knitr::kable(KFOLD_df, format = "markdown") # display table
```

Degree	1	2	3	4	5	Total_MSE
0	3.438751	4.942601	3.672744	4.779153	4.645821	21.47907
1	3.584766	4.483129	3.614016	4.226312	4.321004	21.22923
2	3.427265	4.326183	3.499535	4.504805	4.064654	21.82244
3	3.381529	4.333047	3.556816	4.469806	4.060612	22.80181
4	3.375970	4.375542	3.547353	4.476218	4.128512	23.90359

```
# print best d value
print(paste("The best K-fold degree is: ",
            as.character(KFOLD_df[which.min(KFOLD_df$Total_MSE),]$Degree)))
```

```
## [1] "The best K-fold degree is:  1"
```

```
# create k-fold indexes
k=nrow(train.df)
inds.part = myCVids(n=nrow(train.df),K=k)
# create a data frame to save results into
M = matrix(0, nrow = (length(d_vec)), ncol = k+1)
LOOCV_df = data.frame(M)
LOOCV_df[,1] = d_vec # add degrees to results table
colnames(LOOCV_df) = c("Degree", seq(k))

# loop over k-folds
for( k in seq(k)){
  isk = (inds.part == k) # k varies from 1 to K
  valid.k = which(isk) # test data index
  train.k = which(!isk)
  # split data
  train_sub_df = train.df[train.k,]
  valid_df = train.df[valid.k,]

  d_mse_vec = c()
  for(d in d_vec){
    if(d==0){
      # train a polynomial model
      PR = lm(y ~ 1, data=train_sub_df)
    }else{
     # train a polynomial model
      PR = lm(y ~ poly(x, d, raw = TRUE), data=train_sub_df)
    }
    # get predicted values to calculate MSE
    pred_val = predict(PR, valid_df, type="response")
    pred_true_val_df = data.frame(pred = pred_val, actual = valid_df$y)
    #calculate MSE
    mse = mean((pred_true_val_df$actual - pred_true_val_df$pred)^2)
    d_mse_vec = c(d_mse_vec, mse)
  }
  LOOCV_df[,k+1] = d_mse_vec
```

```
}

# tally of MSE
LOOCV_df$Total_MSE = rowSums(LOOCV_df)
knitr::kable(LOOCV_df[,c(1,k+2)], format = "markdown") # display table
```

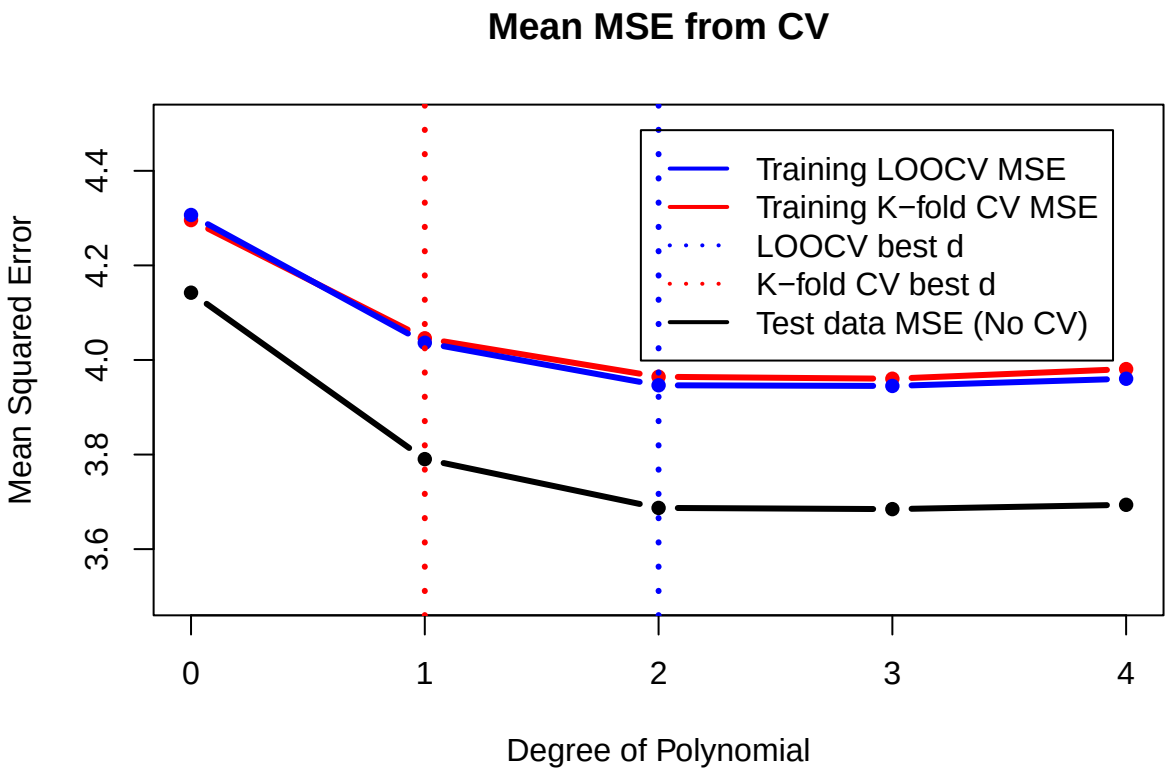
Degree	Total_MSE
0	1722.641
1	1615.560
2	1580.588
3	1581.030
4	1588.124

```
# print best d value
print(paste("The best LOOCV degree is: ",
            as.character(LOOCV_df[which.min(LOOCV_df$Total_MSE),]$Degree)))
```

```
## [1] "The best LOOCV degree is:  2"
```

```
# estimate test data MSE
test_d_vec = c()
for(d in d_vec){
  if(d==0){
    # train a polynomial model
    PR = lm(y ~ 1, data=train.df)
  }else{
   # train a polynomial model
    PR = lm(y ~ poly(x, d, raw = TRUE), data=train.df)
  }
  # get predicted values to calculate MSE
  pred_val = predict(PR, test.df, type="response")
  pred_true_val_df = data.frame(pred = pred_val, actual = test.df$y)
  #calculate MSE
  mse = mean((pred_true_val_df$actual - pred_true_val_df$pred)^2)
  test_d_vec = c(test_d_vec, mse)
}

# visualize LOOCV and K-fold CV
# calculate max and min limits of data KFOLD
kfold_max_vec = apply(KFOLD_df[, 2:(ncol(KFOLD_df)-1)], 1, max)
kfold_min_vec = apply(KFOLD_df[, 2:(ncol(KFOLD_df)-1)], 1, min)
kfold_mean_vec = apply(KFOLD_df[, 2:(ncol(KFOLD_df)-1)], 1, mean)
# calculate max and min limits of data for LOOCV
loocv_max_vec = apply(LOOCV_df[, 2:(ncol(LOOCV_df)-1)], 1, max)
loocv_min_vec = apply(LOOCV_df[, 2:(ncol(LOOCV_df)-1)], 1, min)
loocv_mean_vec = apply(LOOCV_df[, 2:(ncol(LOOCV_df)-1)], 1, mean)
```

```
{plot(x=KFOLD_df$Degree,
     y=kfold_mean_vec,
     ylab="Mean Squared Error",
     main="Mean MSE from CV",
     xlab="Degree of Polynomial",
     type='b',
     col='red',
     pch = 16,
     lwd=3,
     ylim=c(3.5,4.5))
lines(x=LOOCV_df$Degree,
     y=loocv_mean_vec,
     type='b',
     col='blue',
     pch = 16,
     lwd=3)
lines(x=LOOCV_df$Degree,
     y=test_d_vec,
     type='b',
     col='black',
     pch = 16,
     lwd=3)
abline(v=KFOLD_df[which.min(KFOLD_df$Total_MSE),]$Degree,
     col='red',
     pch = 16,
     lty=3,
     lwd=3)
abline(v=LOOCV_df[which.min(LOOCV_df$Total_MSE),]$Degree,
     col='blue',
     pch = 16,
     lty=3,
     lwd=3)
legend("topright",
       inset = 0.05,
       legend = c("Training LOOCV MSE", "Training K-fold CV MSE", "LOOCV best d", "K-fold CV best
       lty = c(1,1,3,3),
       col = c("blue", "red","blue", "red", "black"),
       lwd = 2)}
```

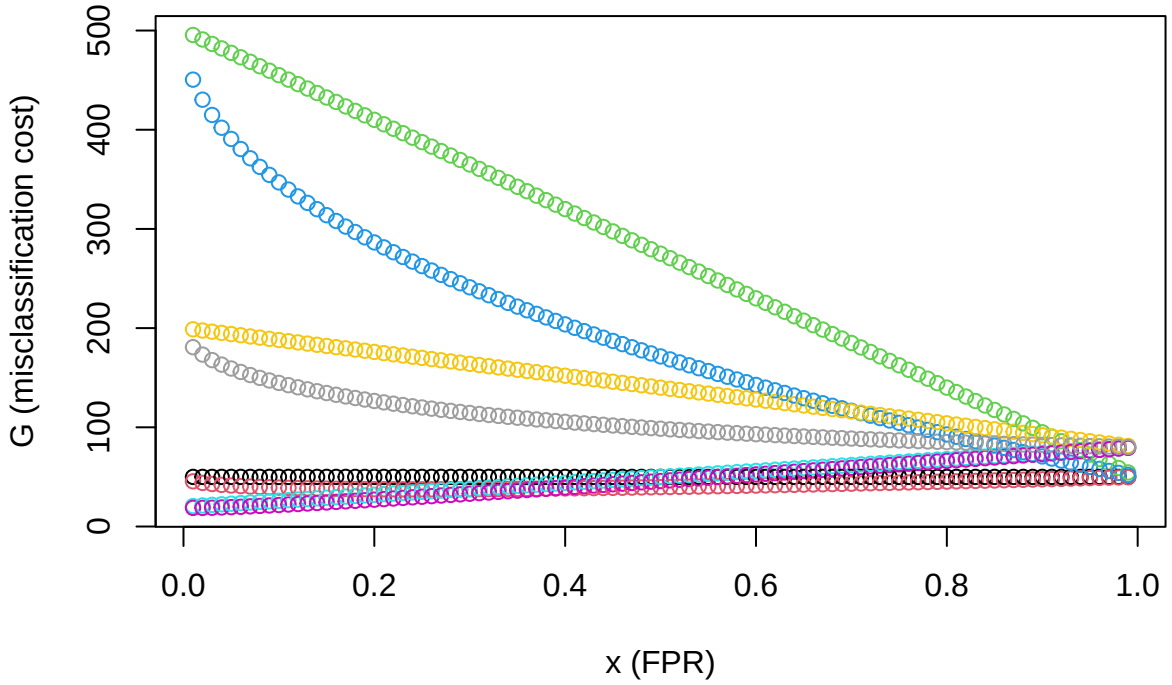


Problem 5

```
# 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)
```

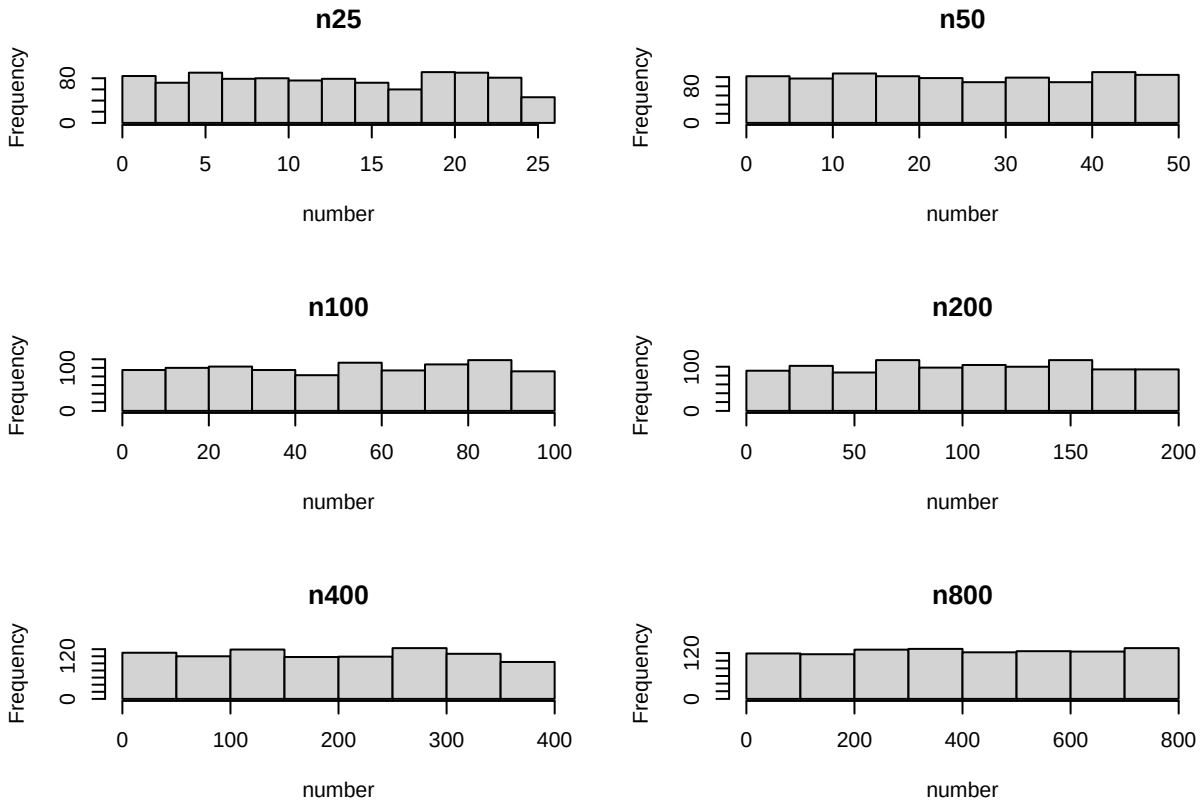
```
results_df$design_id <- paste(results_df$A_vec,
                              results_df$C_vec,
                              results_df$R_vec)

# 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