ABE6933 SML Take-Home Final Exam (100 pts + 10 pts bonus)

Christopher Marais

Exam code, data, and libraries

```r
# 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
}

# function to generate all subsets of the set (1,2,...,p)
myf <- function(p) {
  out = matrix(c(0,1),nrow=2);
  if (p > 1) {
    for (i in (1:(p-1))) {
      d = 2^i
      o1 = cbind(rep(0,d),out)
      o2 = cbind(rep(1,d),out)
      out = rbind(o1,o2)
    }
  }
  colnames(out) <- c(2^((p-1):0)); # powers for binary expansion
  # colnames(out) <- c()
  out
}

nbSubsets <- function(p,m) {
  M  = myf(p)
  rs = rowSums(M)
  ii = (rs == m)
  (M[ii,])
}

# function to convert binary representation to decimal representation
```

```
bin2dec <- function(binM) {
  dd = dim(binM);  # nrows and ncols
  p = dd[2]-1      # max power;
  d = rep(0,dd[1]) # initialize placeholder for the answer
  for (i in 1:(p+1)) {
    d = d + 2^(p+1-i)*binM[,i]
  }
  d
}

# data
load('SML.2022.final.Rdata')

# libraries used in textbook
library(ROCR)
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-6
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.
```

```
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
library(e1071)
library(MASS)

# libraries for ease of use
library(pdist) # pdist() can be replaced by dist() but dist() is slower
```

My functions

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

Problem 1

1.1

The receiver operating characteristic (ROC) curve is a graphical representation of the performance of a binary classification model. It plots the true positive rate (TPR) against the false positive rate (FPR) at different classification thresholds. The ROC curve is useful for evaluating the trade-off between the sensitivity (the ability of the model to correctly identify positive examples) and the specificity (the ability of the model to correctly identify negative examples) of a classification model. One advantage of using the ROC curve to evaluate a classification model is that it is not sensitive to class imbalances in the data. This means that the ROC curve can be used to compare the performance of models on datasets with different distributions of positive and negative examples. However, the ROC curve has some limitations. One limitation is that it does not provide information about the absolute performance of a classification model. For example, a model with an ROC curve that lies along the diagonal line (i.e. a model with no true positive or true negative examples) will have the same ROC curve as a model with a high TPR and low FPR. In contrast, the confusion matrix is a table that displays the number of true positive, true negative, false positive, and false negative examples produced by a classification model. The confusion matrix provides a more detailed view of the performance of a classification model, but it is sensitive to class imbalances in the data. This means that the confusion matrix can be misleading when comparing the performance of models on datasets with different distributions of positive and negative examples.

1.2

The 45-degree line on a receiver operating characteristic (ROC) plot represents the performance of a classifier that is making random predictions. This line is defined by the equation TPR = FPR, where TPR is the true positive rate (the proportion of positive examples that are correctly classified) and FPR is the false positive rate (the proportion of negative examples that are incorrectly classified as positive). For a classifier with TPR = FPR = x, where x is a value in the range [0, 1], the classifier will have an ROC curve that lies along the 45-degree line. This indicates that the classifier is making random predictions and has no ability to differentiate between positive and negative examples. Such a classifier would have a very low overall accuracy and would not be useful for most applications.

1.3

A classifier with TPR(x) = x^2, where x is the false positive rate (FPR), cannot be improved upon without acquiring more data. This is because the TPR and FPR are constrained by the equation TPR = FPR^2, which defines a curve that is always non-decreasing and concave up. This means that the classifier will always have a TPR that is at least as high as the TPR of any other classifier with the same FPR, and it will not be possible to improve the classifier's performance by changing its parameters or applying other techniques without additional data. One way to improve the performance of this classifier would be to acquire more data and retrain the model using the new data. This could potentially allow the model to learn more complex patterns and improve its ability to differentiate between positive and negative examples. However, without knowing more about the specific dataset and the classifier being used, it is difficult to say for certain whether this approach would be effective.

1.4

For a population with 80% controls (0) and 20% cases (1), the true positive rate (TPR) and false positive rate (FPR) for a classifier that flips a coin with probability p of predicting 1 (heads) can be calculated as follows: The TPR is the probability that the classifier correctly predicts 1 for a randomly selected case (1). This probability is equal to the probability that the coin flip results in a prediction of 1, which is p. Therefore, the TPR = p. The FPR is the probability that the classifier incorrectly predicts 1 for a randomly selected control (0). This probability is equal to the probability that the coin flip results in a prediction of 1, given that the true label is 0, which is p * (1 - 0.2) = p * 0.8. Therefore, the FPR = p * 0.8. Overall, the TPR and FPR for this classifier are both directly proportional to the probability p of predicting 1. For example, if p = 0.5, the classifier will have a TPR of 0.5 and a FPR of 0.4. If p = 0.9, the classifier will have a TPR of 0.9 and a FPR of 0.72.

1.5

performance of the classifier is influenced by the proportion of cases (1) in the population (q) in the following ways: As q approaches 0, the TPR and FPR of the classifier will both approach 0. This is because the probability of selecting a case (1) will approach 0, so the probability of correctly predicting 1 will also approach 0. Similarly, the probability of selecting a control (0) will approach 1, so the probability of incorrectly predicting 1 will also approach 0. As q approaches 0.5, the TPR and FPR of the classifier will both approach 0.5. This is because the probability of selecting a case (1) will approach 0.5, so the probability of correctly predicting 1 will also approach 0.5. Similarly, the probability of selecting a control (0) will approach 0.5, so the probability of incorrectly predicting 1 will also approach 0.5. As q approaches 1, the TPR and FPR of the classifier will both approach 1. This is because the probability of selecting a case (1) will approach 1, so the probability of correctly predicting 1 will also approach 1. Similarly, the probability of selecting a control (0) will approach 0, so the probability of incorrectly predicting 1 will also approach 0. Overall, the performance of the classifier is directly influenced by the proportion of cases in the population. As the proportion of cases increases, the TPR and FPR of the classifier will also increase.

1.6

If a ridge regression model is fitted to a dataset with n = 50 and p = 40 covariates, it is unlikely that the optimal shrinkage parameter λ will be equal to 0. This is because the sample size is relatively small compared to the number of covariates, and the true coefficients are known to be nonzero. In this situation, using a non-zero value of λ can help to regularize the model and prevent over fitting by reducing the magnitude of the estimated coefficients. If the sample size n is increased using the same data-generating mechanism, it is likely that the optimal value of λ will decrease. This is because increasing the sample size will provide more information about the true coefficients, and the model will be able to fit the data more accurately without regularization. As a result, a smaller value of λ will be sufficient to prevent overfitting, and the optimal value of λ will decrease. Overall, the optimal value of λ for a ridge regression model depends on the sample size, the number of covariates, and the true coefficients in the data. In general, a larger sample size and a smaller number of covariates will result in a smaller optimal value of λ, while a smaller sample size and a larger number of covariates will result in a larger optimal value of λ.

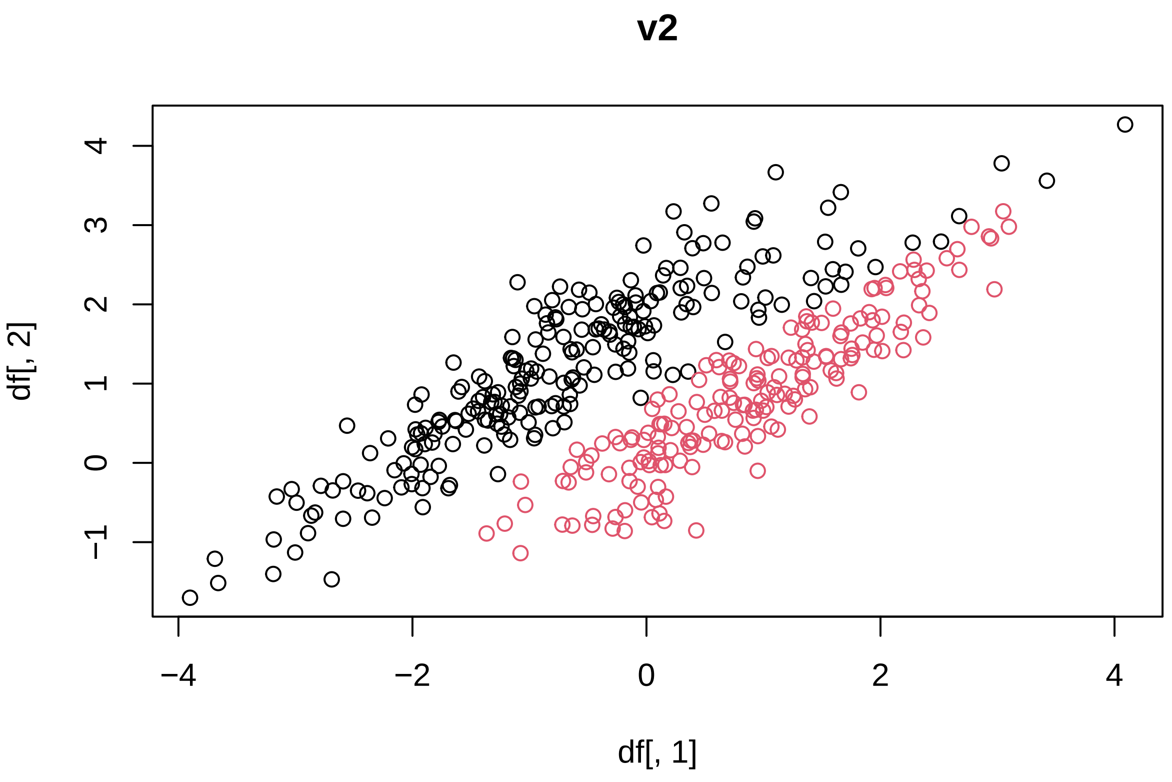
Problem 2

```r
# variables
k = 2

# basic K-means function
my_kmeans <- function(data, k=2, seed=0){
  # initialize centroids from data
  set.seed(seed)
  centroid_mat <- data[sample(nrow(data), k), ]
  old_centroid_mat <- centroid_mat
  new_centroid_mat = matrix(0, nrow = k, ncol = ncol(centroid_mat))
  # repeat until convergence or iteration limit
  while(identical(old_centroid_mat, new_centroid_mat)==FALSE){
    # calculate euclidean distance between centroids and all points
    dist_mat <- t(as.matrix(pdist(centroid_mat, data)))
    # assign each point a class based on closest centroid
    closest_vent_mat = as.matrix(apply(dist_mat, 1, which.min))
    # calculate new centroids as mean of each class
    for(k_i in 1:k){
      new_centroid_mat[k_i,] <- colMeans(data[closest_vent_mat==k_i,])
    }
    old_centroid_mat <- centroid_mat
    centroid_mat <- new_centroid_mat

  }

  return(list(centroid_mat, closest_vent_mat))
}

# get probability from multivariate Gaussian
my_dmvnorm <- function(X,mu,sigma) {
    k <- ncol(X)
    rooti <- backsolve(chol(sigma),diag(k))
    quads <- colSums((crossprod(rooti,(t(X)-mu)))^2)
    return(exp(-(k/2)*log(2*pi) + sum(log(diag(rooti))) - .5*quads))
}

gmm <- function(data, class_vec, mu, k=2, version="v1"){
  if(version=="v1"){
    sigma = cov(data)
    mvn_vec_lst = list()
    for(k_i in 1:k){
      mvn_vec = list(my_dmvnorm(X=data,
            mu=mu[k_i,],
            sigma=sigma))

      mvn_vec_lst = append(mvn_vec_lst, mvn_vec)
    }
    mvn_df = data.frame(mvn_vec_lst)
    colnames(mvn_df) = 1:k
    mvn_df$total = rowSums(mvn_df)

    class_prob_lst = list()
    for(k_i in 1:k){
      class_prob_vec = mvn_df[k_i]/mvn_df$total
      class_prob_lst = append(class_prob_lst, class_prob_vec)
    }
    class_prob_df = data.frame(class_prob_lst)
    colnames(mvn_df) = 1:k
    class_vec = as.matrix(apply(class_prob_df, 1, which.max))
    return(list(class_vec, sigma, mu))

  # version 2 with multiple covariance matrices (one for each class)
  }else if (version=="v2") {
    sigma_lst = list()
    for(k_i in 1:k){
      class_df = data.frame(data[class_vec==k_i,])
      sigma = list(cov(class_df))
      sigma_lst = append(sigma_lst, sigma)
    }

    mvn_vec_lst = list()
    for(k_i in 1:k){
      mvn_vec = list(my_dmvnorm(X=data,
            mu=mu[k_i,],
            sigma=sigma_lst[[k_i]]))

      mvn_vec_lst = append(mvn_vec_lst, mvn_vec)
    }

    mvn_df = data.frame(mvn_vec_lst)
    colnames(mvn_df) = 1:k
    mvn_df$total = rowSums(mvn_df)

    class_prob_lst = list()
    for(k_i in 1:k){
      class_prob_vec = mvn_df[k_i]/mvn_df$total
      class_prob_lst = append(class_prob_lst, class_prob_vec)
    }
    class_prob_df = data.frame(class_prob_lst)
    colnames(mvn_df) = 1:k
    class_vec = as.matrix(apply(class_prob_df, 1, which.max))
    return(list(class_vec, sigma_lst, mu))

  }else{
    print("Version not defined correctly. Use 'v1' OR 'v2'.")
  }
```

```r
}


clustering_procedure <- function(data, k=2, seed=0, version="v1"){
  # initialize clusters with K-means
  # use v1 for LDA approach and v2 for QDA approach
  kmeans_result = my_kmeans(data=data, k=k, seed=seed)
  # get parameters for mixture modelling from k-means clusters
  mu = kmeans_result[[1]]
  class_vec = kmeans_result[[2]]

  # Loop through gmm process until convergence
  old_class_vec <- class_vec
  new_class_vec = rep(0, nrow(data))


  while(identical(new_class_vec,old_class_vec) == FALSE){
    # calculate mu from new class vector
    mu = matrix(0, nrow = k, ncol = ncol(mu))
    for(k_i in 1:k){
      mu[k_i,] <- colMeans(data[class_vec==k_i,])
    }

    gmm_result = gmm(data=data,
                     class_vec=class_vec,
                     mu=mu,
                     k=k,
                     version=version)

    mu = gmm_result[[3]]
    sigma = gmm_result[[2]]
    new_class_vec = gmm_result[[1]]
    old_class_vec <- class_vec
    class_vec <- new_class_vec
  }
  return(list(class_vec, sigma, mu))
}
```

2.1

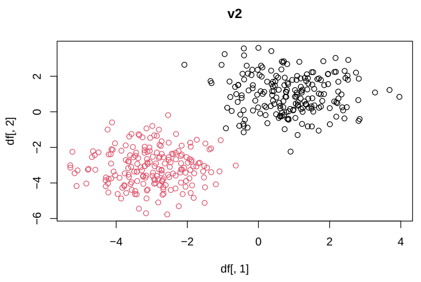
```r
df = data.frame(prob2.list[[5]])
k=2
seed=0
# Multivariate Gaussian
my_dmvnorm <- function(X,mu,sigma) {
    k <- ncol(X)
    rooti <- backsolve(chol(sigma),diag(k))
    quads <- colSums((crossprod(rooti,(t(X)-mu)))^2)
    return(exp(-(k/2)*log(2*pi) + sum(log(diag(rooti))) - .5*quads))
}


####################################
#### initialize parameters
kmeans_result = my_kmeans(data=df, k=k, seed=seed)
# get parameters for mixture modelling from k-means clusters
# get mu from K-means
mu = matrix(c(-1,2,2,0.5), nrow=2)#kmeans_result[[1]] ############### CHANING THE INITIALIZATION
# get class vector from K-means
class_vec = matrix(sample(1:2, size=400, replace=TRUE)) # kmeans_result[[2]] ###############
# Sigma from data # use sigma list if v2/QDA
sigma_lst = list()
for(i_c in 1:k){
  class_df = data.frame(df[class_vec==i_c,])
  sigma = list(cov(class_df))
  sigma_lst = append(sigma_lst, sigma)
}
# sigma = cov(df)
# get pi from data
pi_vec = c()
for(i_c in 1:k){
  pi = mean(class_vec==i_c)
  pi_vec = c(pi_vec, pi)
}
log_likelihood_lst = list()
####################################
#### Estimate class probabilities
mvn_lst = list()
for(i_c in 1:k){
  mvn_ic = list(pi_vec[i_c]*my_dmvnorm(X=df,mu=mu[i_c,], sigma=sigma_lst[[i_c]])) #* sigma_lst
  mvn_lst = append(mvn_lst, mvn_ic)
}
mvn_df = data.frame(mvn_lst)
colnames(mvn_df) = 1:k
mvn_df$total = rowSums(mvn_df)

ri_lst = list()
for(i_c in 1:k){
  ri_vec = mvn_df[i_c]/mvn_df$total
  ri_lst = append(ri_lst, ri_vec)
}
ri_df = data.frame(ri_lst)
class_vec = as.matrix(apply(ri_df, 1, which.max))

#### Update parameters and calculate maximum likelihood
for(i_c in 1:k){
  pi = mean(class_vec==i_c)
  pi_vec = c(pi_vec, pi)

  mu[i_c,] <- colMeans(df[class_vec==i_c,])
}

sigma_lst = list()
for(i_c in 1:k){
  class_df = data.frame(df[class_vec==i_c,])
  sigma = list(cov(class_df))
  sigma_lst = append(sigma_lst, sigma)
}

# calcualte log likelihood
log_likelihood = 0
for(i_c in 1:k){
  log_likelihood = log_likelihood + log(sum(my_dmvnorm(X=df[class_vec==i_c,],
                                                     mu=mu[i_c,],
                                                     sigma=sigma_lst[[i_c]]))) #* sigma_lst
}

log_likelihood_lst = append(log_likelihood_lst, log_likelihood)

plot(x=df[,1],y=df[,2],col=class_vec,main="v2")
```

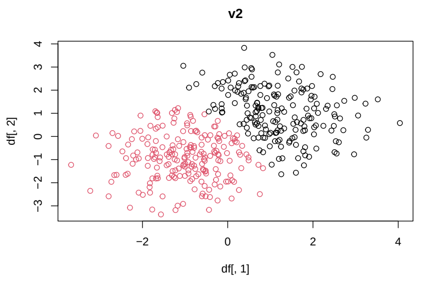


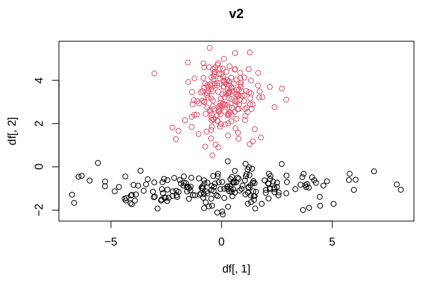
2.2

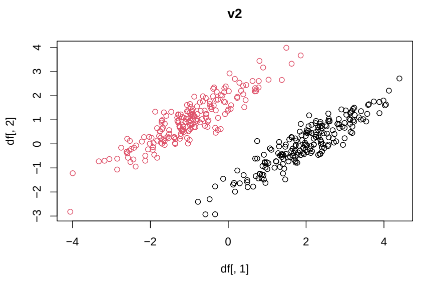
```
#############################
# for(i in 1:7){
#   df=data.frame(prob2.list[[i]])
#   mcl.model <- Mclust(df, 2)
#   plot(mcl.model, what = "classification", main = "Mclust Classification")
# }


# initialk <- mclust::hc(data = df, modelName = "EII")
# initialk <- mclust::hclass(initialk, 2)
# mu <- split(df[, 1:2], initialk)
# mu <- t(sapply(mu, colMeans))
# cov_mat <- cov(df)#list(diag(4), diag(4))
#
# # Mixing Components
# a <- runif(2)
# a <- a/sum(a)
#
# # Calculate PDF with class means and covariances.
# z <- cbind(mvpdf(x = df, mu = mu[1, ], sigma = cov_mat),
#            mvpdf(x = df, mu = mu[2, ], sigma = cov_mat))
#
# # Expectation Step for each class.
# r <- cbind((a[1] * z[, 1])/rowSums(t((t(z) * a))),
#            (a[2] * z[, 2])/rowSums(t((t(z) * a))))
#
# # Choose the highest rowwise probability
# eK <- factor(apply(r, 1, which.max))
#
# # Total Responsibility
# mc <- colSums(r)
# # Update Mixing Components.
# a <- mc/NROW(df)
# # Update our Means
# mu <- rbind(colSums(df[, 1:2] * r[, 1]) * 1/mc[1],
#             colSums(df[, 1:2] * r[, 2]) * 1/mc[2])
# # Update Covariance matrix.
# cov_mat <- t(r[, 1] * t(apply(df[, 1:2], 1, function(x) x - mu[1, ]))) %*%
#     (r[, 1] * t(apply(df[, 1:2], 1, function(x) x - mu[1, ]))) * 1/mc[1]
```

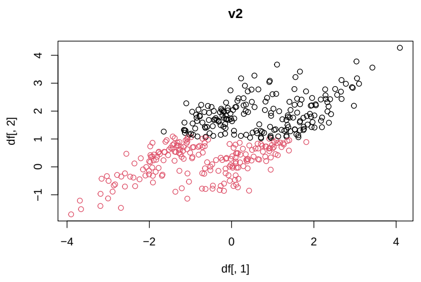
2.3

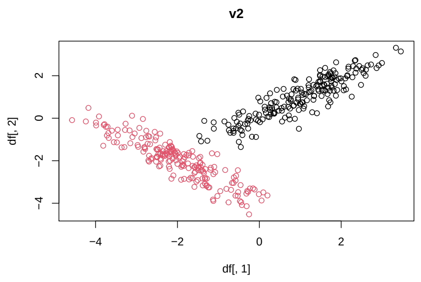


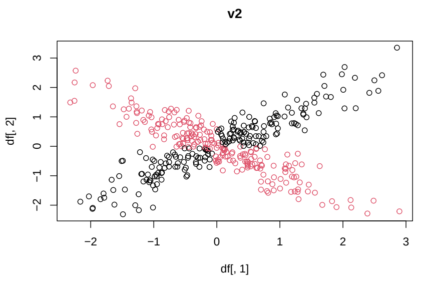












Problem 3

```
# initialize parameters
k = 5
d_max = 10
d_min = 1

# function to calculate RMSE
rmse_func <- function(test, pred){
  rmse = sqrt(mean((test - pred)^2))
  return(rmse)
}

# record data from nested cv
k_i_vec = c()
k_j_vec = c()
d_vec = c()
rmse_j_vec = c()
best_d_vec = c()
```

```r
rmse_i_vec = c()
rmse_i_mat = matrix(0, nrow=d_max, ncol=k)
# Outer CV to estimate performance (seed=0)
n_i = nrow(prob3.df)
for(k_i in seq(k)){
  inds.part = myCVids(n=n_i, K=k, seed=0)
  isk = (inds.part == k_i)
  valid.i = which(isk)
  train.i = which(!isk)
  # split data into external train and test sets
  data.valid.i = prob3.df[valid.i,]
  rownames(data.valid.i) <- NULL
  data.train.i = prob3.df[train.i,]
  rownames(data.train.i) <- NULL

  # loop through parameter sets
  rmse_d_vec = c()
  for(d in d_min:d_max){
    # Inner CV to estimate parameters (seed=1000)
    n_j = nrow(data.train.i)
    rmse_j_d_vec = c()
    for(k_j in seq(k)){
      inds.part = myCVids(n=n_j, K=k, seed=1000)
      isk = (inds.part == k_j)
      valid.j = which(isk)
      train.j = which(!isk)
      # split data into train and test sets
      data.valid.j = data.train.i[valid.j,]
      data.train.j = data.train.i[train.j,]
      # train model in internal cv loop
      lm.fit.j = lm(y ~ poly(x, degree=d), data = data.train.j)
      pred.j = predict(lm.fit.j , data.valid.j)
      rmse.j = rmse_func(test=data.valid.j$y, pred=pred.j)
      rmse_j_vec = c(rmse_j_vec, rmse.j)
      rmse_j_d_vec = c(rmse_j_d_vec, rmse.j)

      # record data
      k_i_vec = c(k_i_vec, k_i)
      k_j_vec = c(k_j_vec, k_j)
      d_vec = c(d_vec, d)
    }

    # get mean RMSE for each parameter
    rmse_d = mean(rmse_j_d_vec)
    rmse_d_vec = c(rmse_d_vec, rmse_d)

    # train external model with selected d
    lm.fit.i = lm(y ~ poly(x, degree=d), data = data.train.i)
    pred.i = predict(lm.fit.i , data.valid.i)
    rmse.i = rmse_func(test=data.valid.i$y, pred=pred.i)
    rmse_i_mat[d,k_i] = rmse.i

  }

  # select parameter set with lowest RMSE
  best_d = (d_min:d_max)[which(rmse_d_vec == min(rmse_d_vec))]
  best_d_vec = c(best_d_vec, best_d)

  # train external model with selected d
  lm.fit.i = lm(y ~ poly(x, best_d), data = data.train.i)
  pred.i = predict(lm.fit.i , data.valid.i)
  rmse.i = rmse_func(test=data.valid.i$y, pred=pred.i)
  rmse_i_vec = c(rmse_i_vec, rmse.i)

}


# organize results into a data frame
results_df = data.frame(ki=k_i_vec, kj=k_j_vec, d=d_vec, rmse_j=rmse_j_vec)
results_df$rmse_i = 0
results_df$best_d = 0
results_df$best_rmse = 0
for(i_k in 1:k){
  results_df$best_d[results_df$ki==i_k] = best_d_vec[i_k]
  results_df$best_rmse[results_df$ki==i_k] = rmse_i_vec[i_k]
  for(d in d_min:d_max){
    results_df$rmse_i[results_df$ki==i_k & results_df$d==d] = rmse_i_mat[d,k_i]
  }
}

# collect data for curve plots from inner adn outer cv loop results
ki_d_rmse_lst = list()

for(k_i in 1:k){
  ki_d_rmse_vec = c()
  d_rmse_vec = c()
  for(d in d_min:d_max){
    ki_d_mean_rmse = mean(
                      results_df$rmse_j[results_df$ki==k_i & results_df$d==d]
                      )
    ki_d_rmse_vec = c(ki_d_rmse_vec, ki_d_mean_rmse)
    d_rmse = mean(
                  results_df$rmse_i[results_df$d==d]
                  )
    d_rmse_vec = c(d_rmse_vec, d_rmse)

  }
  ki_d_rmse_lst = append(ki_d_rmse_lst, list(ki_d_rmse_vec))
```

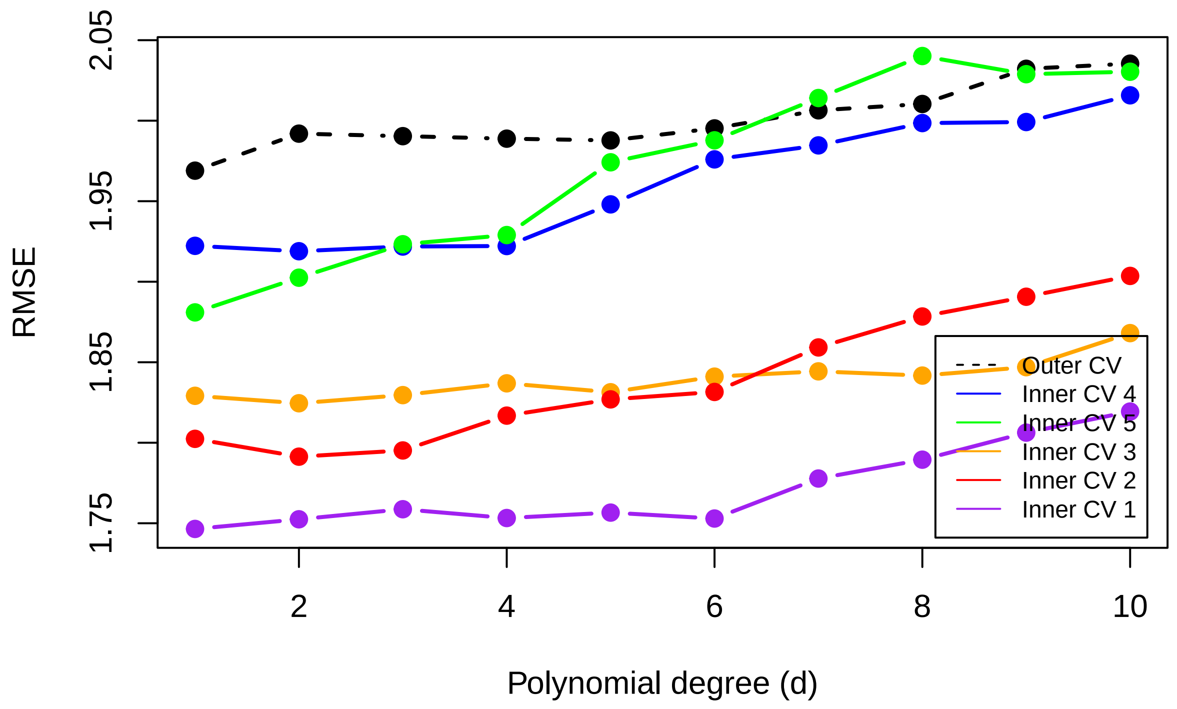
```
}

# display results table
knitr::kable(head(results_df, 100))
```

ki	kj	d	rmse_j	rmse_i	best_d	best_rmse
1	1	1	1.870285	1.968968	2	1.484873
1	2	1	1.963539	1.968968	2	1.484873
1	3	1	2.055382	1.968968	2	1.484873
1	4	1	1.657675	1.968968	2	1.484873
1	5	1	2.064572	1.968968	2	1.484873
1	1	2	1.878747	1.991959	2	1.484873
1	2	2	1.957772	1.991959	2	1.484873
1	3	2	2.037305	1.991959	2	1.484873
1	4	2	1.649110	1.991959	2	1.484873
1	5	2	2.071771	1.991959	2	1.484873
1	1	3	1.896799	1.990399	2	1.484873
1	2	3	1.955857	1.990399	2	1.484873
1	3	3	2.035939	1.990399	2	1.484873
1	4	3	1.648428	1.990399	2	1.484873
1	5	3	2.072248	1.990399	2	1.484873
1	1	4	1.897565	1.988841	2	1.484873
1	2	4	1.957486	1.988841	2	1.484873
1	3	4	2.034824	1.988841	2	1.484873
1	4	4	1.649061	1.988841	2	1.484873
1	5	4	2.071914	1.988841	2	1.484873
1	1	5	1.935765	1.987758	2	1.484873
1	2	5	1.957102	1.987758	2	1.484873
1	3	5	2.120890	1.987758	2	1.484873
1	4	5	1.650729	1.987758	2	1.484873
1	5	5	2.075676	1.987758	2	1.484873
1	1	6	2.029966	1.995224	2	1.484873
1	2	6	1.989976	1.995224	2	1.484873
1	3	6	2.123344	1.995224	2	1.484873
1	4	6	1.647030	1.995224	2	1.484873
1	5	6	2.089536	1.995224	2	1.484873
1	1	7	2.039120	2.006448	2	1.484873
1	2	7	2.007069	2.006448	2	1.484873
1	3	7	2.123280	2.006448	2	1.484873
1	4	7	1.663389	2.006448	2	1.484873
1	5	7	2.090476	2.006448	2	1.484873
1	1	8	2.038876	2.010370	2	1.484873
1	2	8	2.037914	2.010370	2	1.484873
1	3	8	2.144957	2.010370	2	1.484873
1	4	8	1.662937	2.010370	2	1.484873
1	5	8	2.108003	2.010370	2	1.484873
1	1	9	2.040491	2.032230	2	1.484873
1	2	9	2.037163	2.032230	2	1.484873

ki	kj	d	rmse_j	rmse_i	best_d	best_rmse
1	3	9	2.144108	2.032230	2	1.484873
1	4	9	1.658463	2.032230	2	1.484873
1	5	9	2.115636	2.032230	2	1.484873
1	1	10	2.073504	2.035435	2	1.484873
1	2	10	2.045652	2.035435	2	1.484873
1	3	10	2.142588	2.035435	2	1.484873
1	4	10	1.703645	2.035435	2	1.484873
1	5	10	2.113479	2.035435	2	1.484873
2	1	1	2.117692	1.968968	1	1.689378
2	2	1	1.997274	1.968968	1	1.689378
2	3	1	1.869417	1.968968	1	1.689378
2	4	1	1.535978	1.968968	1	1.689378
2	5	1	1.884340	1.968968	1	1.689378
2	1	2	2.106300	1.991959	1	1.689378
2	2	2	2.060370	1.991959	1	1.689378
2	3	2	1.907080	1.991959	1	1.689378
2	4	2	1.532592	1.991959	1	1.689378
2	5	2	1.906136	1.991959	1	1.689378
2	1	3	2.100055	1.990399	1	1.689378
2	2	3	2.057835	1.990399	1	1.689378
2	3	3	1.970956	1.990399	1	1.689378
2	4	3	1.554205	1.990399	1	1.689378
2	5	3	1.933351	1.990399	1	1.689378
2	1	4	2.099792	1.988841	1	1.689378
2	2	4	2.058728	1.988841	1	1.689378
2	3	4	1.971691	1.988841	1	1.689378
2	4	4	1.579596	1.988841	1	1.689378
2	5	4	1.935071	1.988841	1	1.689378
2	1	5	2.156970	1.987758	1	1.689378
2	2	5	2.062639	1.987758	1	1.689378
2	3	5	2.036358	1.987758	1	1.689378
2	4	5	1.656695	1.987758	1	1.689378
2	5	5	1.958164	1.987758	1	1.689378
2	1	6	2.196910	1.995224	1	1.689378
2	2	6	2.082418	1.995224	1	1.689378
2	3	6	2.030913	1.995224	1	1.689378
2	4	6	1.653409	1.995224	1	1.689378
2	5	6	1.975435	1.995224	1	1.689378
2	1	7	2.217288	2.006448	1	1.689378
2	2	7	2.085739	2.006448	1	1.689378
2	3	7	2.128182	2.006448	1	1.689378
2	4	7	1.664233	2.006448	1	1.689378
2	5	7	1.974984	2.006448	1	1.689378
2	1	8	2.222984	2.010370	1	1.689378
2	2	8	2.111251	2.010370	1	1.689378
2	3	8	2.128832	2.010370	1	1.689378
2	4	8	1.755577	2.010370	1	1.689378
2	5	8	1.982152	2.010370	1	1.689378

ki	kj	d	rmse_j	rmse_i	best_d	best_rmse
2	1	9	2.229218	2.032230	1	1.689378
2	2	9	2.100867	2.032230	1	1.689378
2	3	9	2.078272	2.032230	1	1.689378
2	4	9	1.767015	2.032230	1	1.689378
2	5	9	1.969071	2.032230	1	1.689378
2	1	10	2.231451	2.035435	1	1.689378
2	2	10	2.098925	2.035435	1	1.689378
2	3	10	2.082165	2.035435	1	1.689378
2	4	10	1.773148	2.035435	1	1.689378
2	5	10	1.966365	2.035435	1	1.689378

```r
# add data to data frame for visualization
plot_df = data.frame((d_min:d_max), ki_d_rmse_lst, d_rmse_vec)
colnames(plot_df) = c("d", as.character(1:k), "outer_cv")

# visualize results as line plots
{plot(x=plot_df$d,
     y=plot_df$outer_cv,
     type="b",
     col="black",
     lwd=2,
     lty=2,
     ylim=c(min(plot_df[2:7]),max(plot_df[2:7])),
     pch=19,
     ylab="RMSE",
     xlab="Polynomial degree (d)")
lines(x=plot_df$d, y=plot_df$`1`, col="blue", lwd=2, type="b", pch=19)
lines(x=plot_df$d, y=plot_df$`2`, col="green", lwd=2, type="b", pch=19)
lines(x=plot_df$d, y=plot_df$`3`, col="orange", lwd=2, type="b", pch=19)
lines(x=plot_df$d, y=plot_df$`4`, col="purple", lwd=2, type="b", pch=19)
lines(x=plot_df$d, y=plot_df$`5`, col="red", lwd=2, type="b", pch=19)
legend("bottomright",
       inset=.02,
       legend=c("Outer CV",
                "Inner CV 4",
                "Inner CV 5",
                "Inner CV 3",
                "Inner CV 2",
                "Inner CV 1"),
       col=c("black","blue","green","orange","red","purple"),
       lty=c(2,1,1,1,1,1),
       cex=0.75)}
```



```
# display best "ensemble" model statistics
best_model_df = results_df[
  c("ki","best_d","best_rmse")
  ][!duplicated(
    results_df[c("ki","best_d","best_rmse")]
    ),]
rownames(best_model_df) = NULL
knitr::kable(best_model_df)
```

ki	best_d	best_rmse
1	2	1.484873
2	1	1.689378
3	2	1.907865
4	1	2.047146
5	2	1.991959

```
print(paste("Mean RMSE: ", mean(best_model_df$best_rmse)))
```

```
## [1] "Mean RMSE:  1.8242442649514"
```

Nested k-fold cross-validation can be computationally expensive and time-consuming, especially for complex models and large data sets. Models with a lot of descriptors or large data sets will increase computation time even further making this technique not suited when time or computational constraints are a concern.

Problem 4

4.a

```r
# initialize data
p = 4
k_max = 5
n = nrow(prob4.df)
binM = myf(p)
ids = bin2dec(binM)
ROC_df = data.frame(matrix(ncol = length(ids), nrow = n), Y=prob4.df$Y)
colnames(ROC_df) = c(ids, "Y")

# loop through models
feature_names = c("X1","X2","X3","X4")
features_vec = c()
mean_mcr_vec = c()
glm_lst = list()
for(i in seq(1:length(ids))){

  # select subset of data
  gamma = binM[i,]
  alpha = ids[i]
  X = data.frame(Intercept=1, prob4.df[,-5][,gamma==1])
  Y = prob4.df$Y
  data_df = data.frame(Y, X)

  # get feature names of id
  if(sum(gamma)==0){
    features = "None"
  }else{
    features = feature_names[gamma==1]
  }
  features_vec = c(features_vec, paste(features,collapse=" "))

  # perform 5-fold CV
  inds.part = myCVids(n, 5, seed=0)
  # loop through folds
  mcr_vec = c()
  for(k in seq(1:k_max)){
    isk = (inds.part == k)
    valid.k = which(isk)
    train.k = which(!isk)

    # train logistic regression model
    glm.fit = glm(Y ~ 0 +.,
                  family=binomial,
                  data=as.data.frame(data_df[train.k,]))
    glm_lst = append(glm_lst, list(glm.fit))
    # predict target on validation data
    pred = predict(glm.fit , data_df[valid.k,])
    ROC_df[valid.k,i] = pred

    # calculate mis-classification error rate for default 0.5 threshold
    mcr = MCR(target=data_df[valid.k,]$Y, predicted=pred, threshold=0.5)
    mcr_vec = c(mcr_vec, mcr)
  }
  mean_mcr = mean(mcr_vec)
  mean_mcr_vec = c(mean_mcr_vec, mean_mcr)
}

# add data to a data frame
res_df = data.frame(ids,
features_vec,
mean_mcr_vec)
colnames(res_df) = c("ID", "covariates", "mean_mcr")
ord_res_df = res_df[order(res_df$mean_mcr),]
rownames(ord_res_df) = NULL
knitr::kable(ord_res_df, format = "markdown")
```

ID	covariates	mean_mcr
14	X1 X2 X3	0.415
7	X2 X3 X4	0.420
10	X1 X3	0.420
2	X3	0.425
6	X2 X3	0.430
3	X3 X4	0.435
11	X1 X3 X4	0.435
15	X1 X2 X3 X4	0.435
9	X1 X4	0.470
8	X1	0.500
13	X1 X2 X4	0.500
12	X1 X2	0.505
4	X2	0.520
0	None	0.525
1	X4	0.530
5	X2 X4	0.540

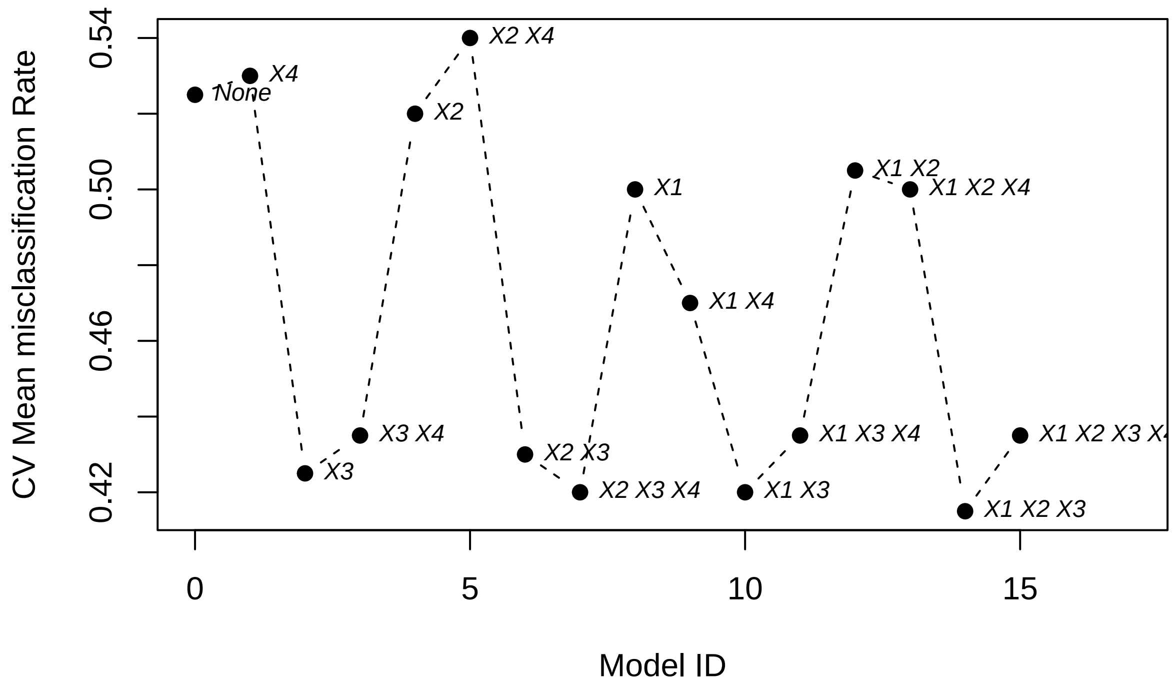
```
print(paste())
```

```
## character(0)
```

```
print(paste("The Best model ID is ",res_df[1,1],
            "with ",res_df[1,2],
            "as features and a MCR of ",res_df[1,3]))
```

```
## [1] "The Best model ID is  0 with  None as features and a MCR of  0.525"
```

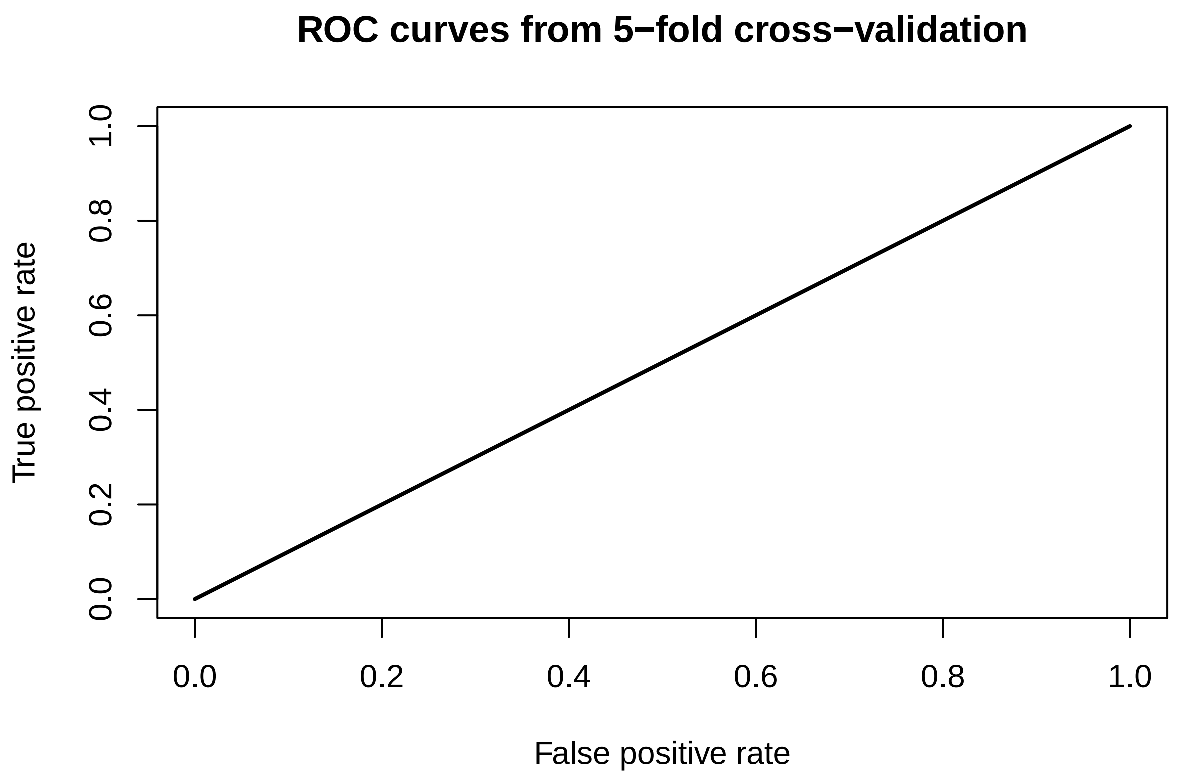
```
{plot(x=res_df$ID,
     y=res_df$mean_mcr,
     pch=19,
     type="b",
     lty=2,
     xlim=c(min(res_df$ID), max(res_df$ID)+2),
     xlab="Model ID",
     ylab="CV Mean misclassification Rate")
text(mean_mcr~ID,
     labels=res_df$covariates,
     data=res_df,
     cex=0.75,
     font=3,
     pos=4)}
```



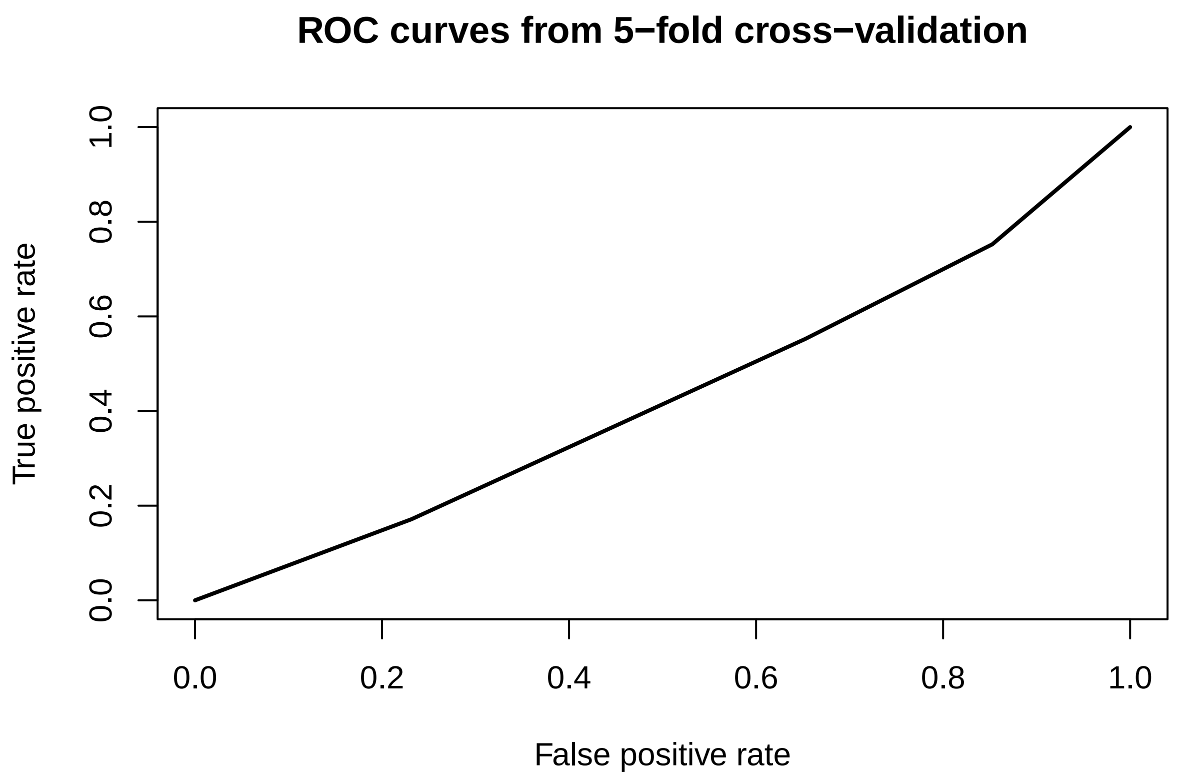
4.b

```
# set threshold resolution for ROC curve
threshold_vec = seq(0, 1, 0.1)
best_model = res_df[1,1]
k_max = 5
```

```
# data of best model
Y_pred = ROC_df[,best_model+1]
Y = ROC_df[,ncol(ROC_df)]

# looping hrough the folds seem to not be he way they want it.
# create ROC curve for the best model
inds.part = myCVids(n, 5, seed=0)
for(k in seq(1:k_max)){
  isk = (inds.part == k)
  valid.k = which(isk)

  pred <- prediction(Y_pred[valid.k], Y[valid.k])
  perf <- performance(pred,'tpr','fpr')
  plot(perf,
       lwd=2,
       main='ROC curves from 5-fold cross-validation')
}
```



```
# this way seems to be the correct one
pred <- prediction(Y_pred, Y)
perf <- performance(pred,'tpr','fpr')


plot(perf,
     lwd=2,
     main='ROC curves from 5-fold cross-validation')
```



```
# print best threshold
# show diagonal

"For your best model, use 5-fold CV (same folds as in problem 4a) to construct the ROC curve plot
vector of predicted values for the ROC curve plot should be produced by training the best model o
Ti and predicting on Vi. To get the entire vector of predicted values, put the 5 vectors of predi
into a single vector; then compare vs truth (the whole vector Y ) proceeding as with the usual RO
plot construction (reuse the code examples from the ISLR book)."
```

```
## [1] "For your best model, use 5-fold CV (same folds as in problem 4a) to construct the ROC cur
```

Problem 5

```
p=50
n = nrow(prob5.df)
test_df = prob5.df[401:800,]
data_set_vec = c(100, 200, 400)
k_max = 5
```

```r
# loop through sets of data
for(set in data_set_vec){
  # specify training data set
  train_df = prob5.df[1:set,]
  # perform 5-fold CV
  set_n = nrow(train_df)
  inds.part = myCVids(set_n, 5, seed=0)
  lr_rmse_vec = c()
  rf_rmse_vec = c()
  gbm_rmse_vec = c()
  for(k in seq(1:k_max)){
    isk = (inds.part == k)
    valid.k = which(isk)
    train.k = which(!isk)
    cv_train_df = train_df[train.k,]
    cv_valid_df = train_df[valid.k,]
    x_cv_train_df = model.matrix(Y~., cv_train_df )[,-1]
    y_cv_train_df = cv_train_df$Y
    x_cv_valid_df = model.matrix(Y~., cv_valid_df )[,-1]
    y_cv_valid_df = cv_valid_df$Y

    # train LR
    set.seed(0)
    lasso_reg.fit = glmnet(x_cv_train_df,y_cv_train_df, alpha=1)

    # train RF
    set.seed(0)
    rf.fit = randomForest(Y~.,
                          data=cv_train_df,
                          mtry=c(1:7,50),
                          ntree=500,
                          importance=TRUE)

    # train GBM
    set.seed(0)
    gbm.fit = gbm(Y~.,
                  data=cv_train_df,
                  distribution="gaussian",
                  n.trees=1000,
                  shrinkage=0.01,
                  interaction.depth=7) # use 1:7 or test all or just one (7)

    # eval LR
    lr_pred = predict(lasso_reg.fit, x_cv_valid_df)
    lr_rmse = sqrt(mean((y_cv_valid_df - lr_pred)^2))
    lr_rmse_vec = c(lr_rmse_vec, lr_rmse)

    # eval RF
    rf_pred = predict(rf.fit, cv_valid_df)
    rf_rmse = sqrt(mean((y_cv_train_df - rf_pred)^2))
    rf_rmse_vec = c(rf_rmse_vec, rf_rmse)

    # eval GBM
    gbm_pred = predict(gbm.fit, cv_valid_df)
    gbm_rmse = sqrt(mean((y_cv_train_df - gbm_pred)^2))
    gbm_rmse_vec = c(gbm_rmse_vec, gbm_rmse)

  }
  print(set)
  print(mean(lr_rmse_vec))
  print(mean(rf_rmse_vec))
  print(mean(gbm_rmse_vec))
  print("____________")



}
```

```
## [1] 100
## [1] 1.98515
## [1] 1.365293
## [1] 1.566195
## [1] "____________"
## [1] 200
## [1] 1.678889
## [1] 1.411676
## [1] 1.536651
## [1] "____________"
## [1] 400
## [1] 1.468838
## [1] 1.399912
## [1] 1.482933
## [1] "____________"
```

```
# find parameters that have to get tuned and tune for them.
# plots should be tuning parameter and cv rmse
# use cv as used to tune parameters in Q3
# calcualte test and cv RMSE and print in table

# Discuss what you expect: (i) as training sample size increases but the test
# sample is held fixed; (ii) training
# set is held fixed but the test sample size increases.


"For each method, report 3 CV RMSE curves (one curve for each n=100,200 and 400), overlaid on the
same plot. Mark the optimal value of the CV RMSE and the corresponding value of the tuning parame
Additionally, report the 3-by-6 matrix with rows corresponding to the three ML methods and column
corresponding to the CV RMSE and test RMSE for each value of n for model with parameters chosen b
```

```
CV. (I.e., two RMSE values for n = 100, then the two RMSE values for n = 200, etc.) The CV RMSE
will be computed from the training data only. The test RMSE should be computed by fitting the mod
on the entire training dataset with parameters determined by the CV; then this model is used to p
on the test set.
Briefly discuss your findings; particularly, as n increases.
For RF and GBM, report and discuss variable importance (for the best models).
Discuss what you expect: (i) as training sample size increases but the test sample is held fixed;
set is held fixed but the test sample size increases."
```

```
## [1] "For each method, report 3 CV RMSE curves (one curve for each n=100,200 and 400), overlaid
```

```
# generate additional 50 features
data_df = data.frame(prob5.df, matrix( rnorm(n*50,mean=0,sd=1), n, 50))

# repeat 5 and see effect

# What do you expect to happen to the predictive performance of
# the methods?
```

Problem 6

```
# define data
p =20
n = 200
train_df = prob6.df[1:200,]
test_df = prob6.df[201:400,]

# 5-fold validation
inds.part = myCVids(n, 5, seed=0)
svm_rmse_vec = c()
rf_rmse_vec = c()
gbm_rmse_vec = c()
for(k in seq(1:k_max)){
  isk = (inds.part == k)
  valid.k = which(isk)
  train.k = which(!isk)
  cv_train_df = train_df[train.k,]
  cv_valid_df = train_df[valid.k,]
  y_cv_valid_df = cv_valid_df$Y

  # train SVM
  set.seed(0)
  svm.fit = svm(Y~.,
                data=cv_train_df,
                kernel="radial",
                gamma=1,
                cost=1,
                type="C")

  # train RF
  set.seed(0)
  rf.fit = randomForest(as.factor(Y)~.,
                        data=cv_train_df,
                        mtry=c(1:7,50),
                        ntree=500,
                        importance=TRUE)

  # train GBM
  set.seed(0)
  gbm.fit = gbm(Y~.,
                data=cv_train_df,
                distribution="bernoulli",
                n.trees=1000,
                shrinkage=0.01,
                interaction.depth=7) # use 1:7 or test all or just one (7)

  # eval SVM
  svm_pred = predict(svm.fit, cv_valid_df)
  # svm_rmse_vec = c(svm_rmse_vec, svm_rmse)

  # eval RF
  rf_pred = predict(rf.fit, cv_valid_df)
  # rf_rmse_vec = c(rf_rmse_vec, rf_rmse)

  # eval GBM
  gbm_pred = predict(gbm.fit, cv_valid_df)
  # needs a threshold
  # gbm_rmse_vec = c(gbm_rmse_vec, gbm_rmse)

}
```

```
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
```

```
# print(mean(svm_rmse_vec))
# print(mean(rf_rmse_vec))
# print(mean(gbm_rmse_vec))
print("____________")
```

```
## [1] "___________"
```

```
"For each method, report CV misclassification error rate (MER) curves; this will be computed with

Additionally, report the test MER for each classifier with the parameters chosen by CV (the entir

Lastly, overlay on the same plot ROCR curves for the three classifiers (with optimal tuning param

Briefly discuss.
"
```

```
## [1] "For each method, report CV misclassification error rate (MER) curves; this will be comput
```