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
```

```r
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
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.1

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

```r
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```r
library(e1071)
library(MASS)

# libraries for ease of use
library(pdist) # pdist() can be replaced by dist() but dist() is slower
library(tidyr) # just saved some time on filling NA values in a dataframe
```

```
##
## Attaching package: 'tidyr'

## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

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) = xˆ2, 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 = FPRˆ2, 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

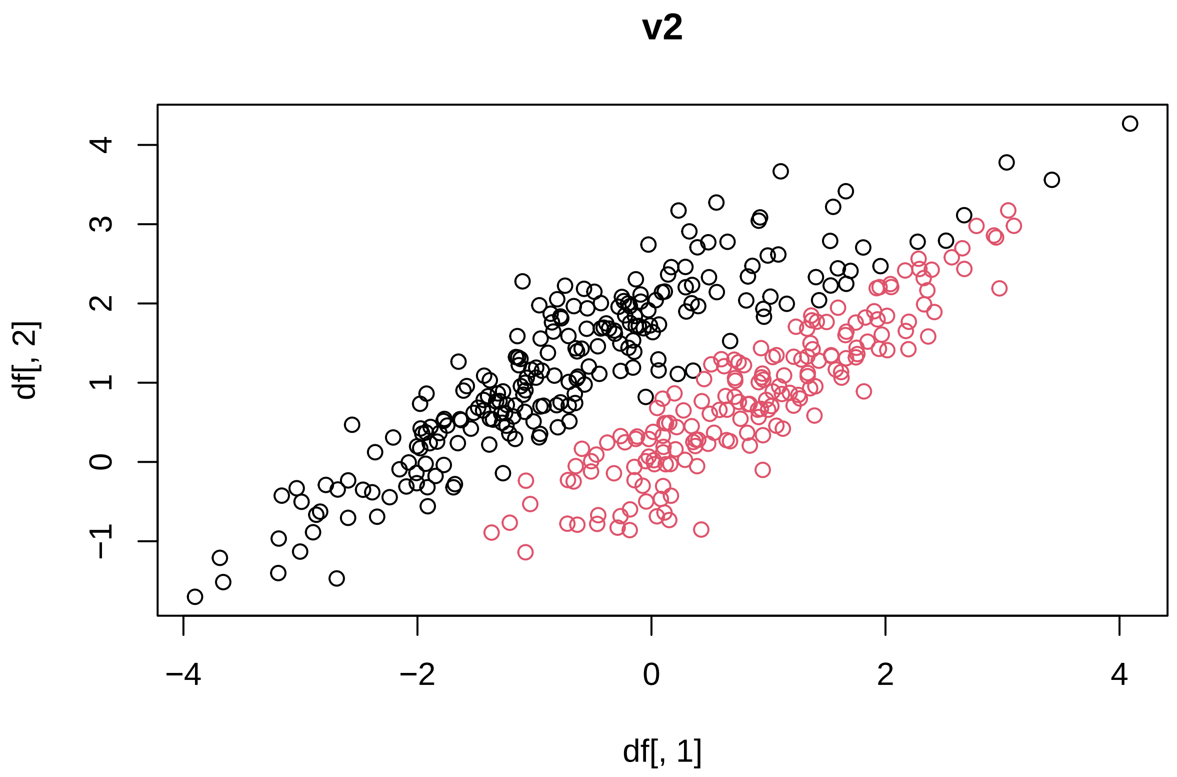
```r
# variables
k = 2

# 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))
}
```

```r
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'.")
  }

}


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

```
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")
```

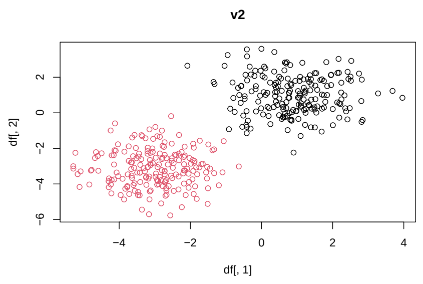


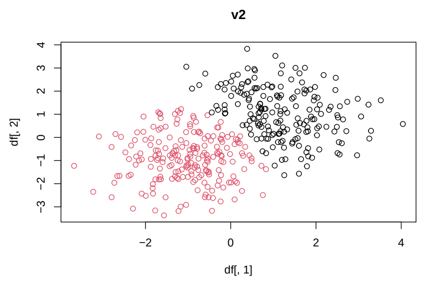
2.2

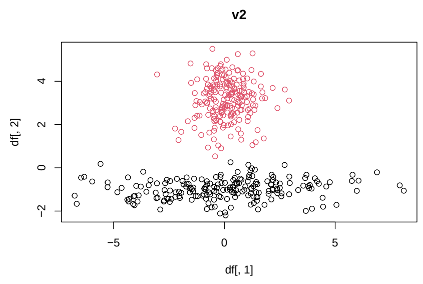
```
##############################
# for(i in 1:7){
#   df=data.frame(prob2.list[[i]])
#   mcl.model <- Mclust(df, 2)
#   plot(mcl.model, what = "classification", main = "Mclust Classification")
# }


# 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))
```

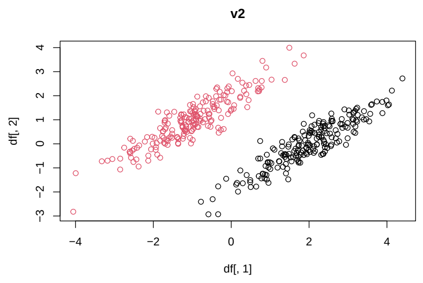
```
#
# # Expectation Step for each class.
# r <- cbind((a[1] * z[, 1])/rowSums(t((t(z) * a))),
#            (a[2] * z[, 2])/rowSums(t((t(z) * a))))
#
# # Choose the highest rowwise probability
# eK <- factor(apply(r, 1, which.max))
#
# # Total Responsibility
# mc <- colSums(r)
# # Update Mixing Components.
# 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]
```

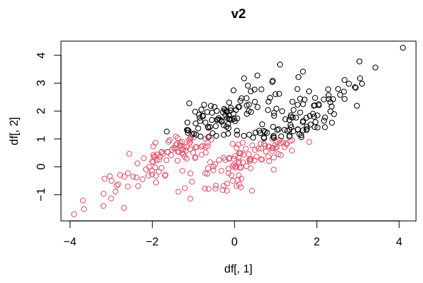
2.3

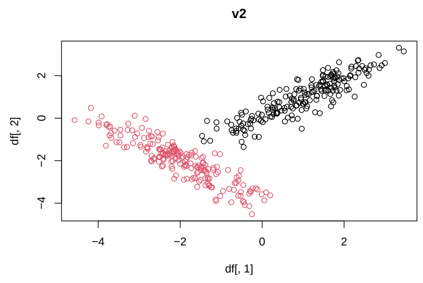


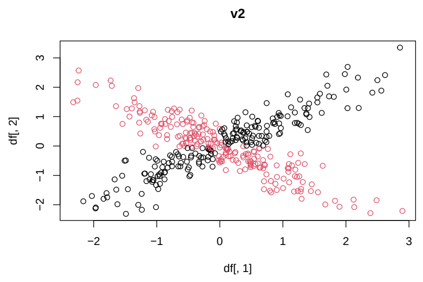












Problem 3

```r
# 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()
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]
  }
}
```

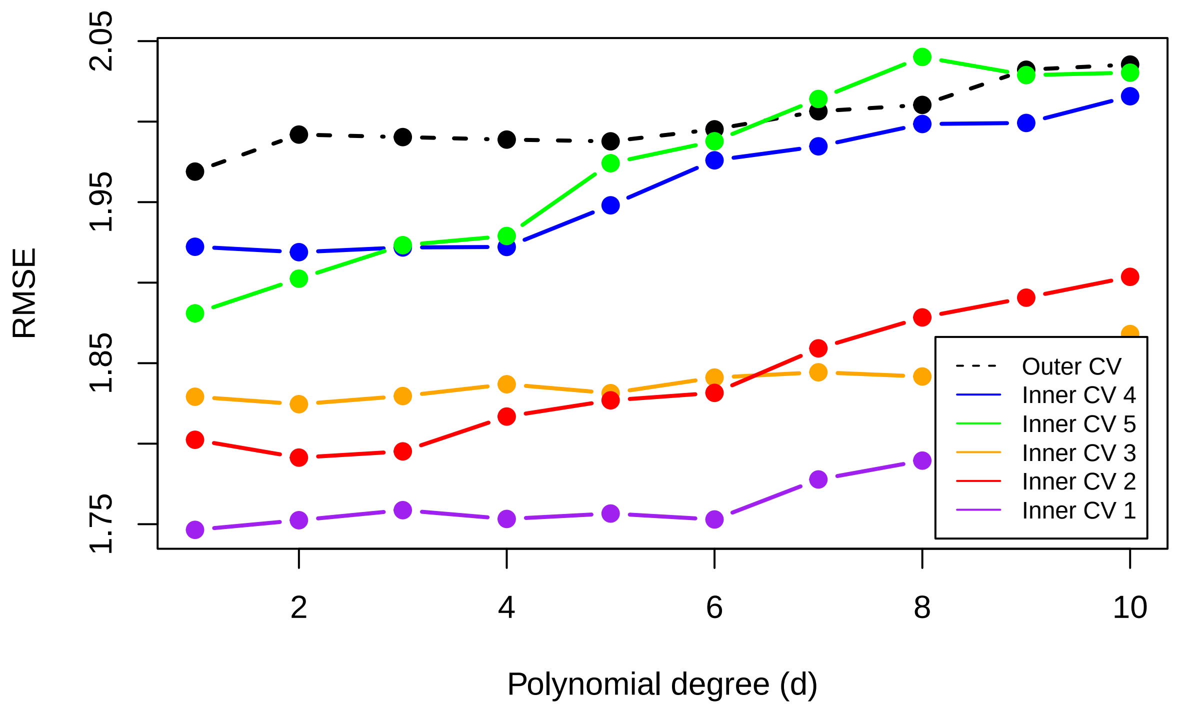
```r
# collect data for curve plots from inner adn outer cv loop results
ki_d_rmse_lst = list()

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

ki	kj	d	rmse_j	rmse_i	best_d	best_rmse
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
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

ki	kj	d	rmse_j	rmse_i	best_d	best_rmse
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
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),
       bg = "white",
       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

```
# 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){
```

```r
    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,], type= "response")
    ROC_df[valid.k,i] = pred # store predictions in data frame for 4.b

    # 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")
```

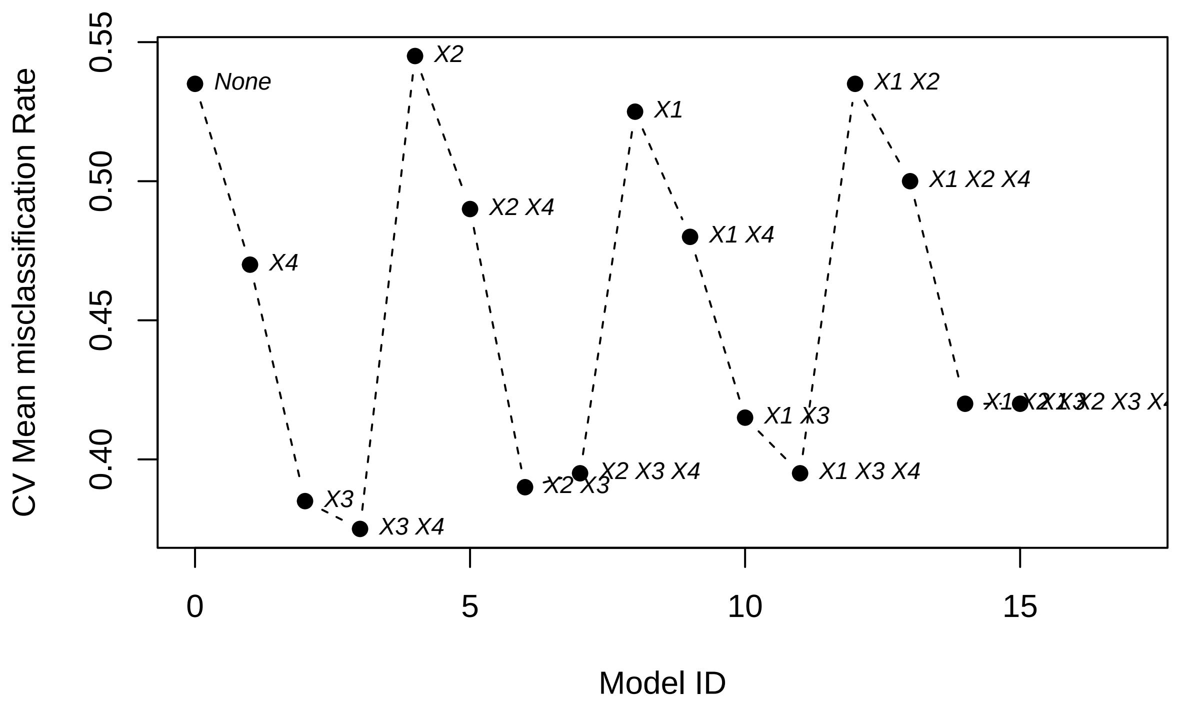
ID	covariates	mean_mcr
3	X3 X4	0.375
2	X3	0.385
6	X2 X3	0.390
7	X2 X3 X4	0.395
11	X1 X3 X4	0.395

ID	covariates	mean_mcr
10	X1 X3	0.415
14	X1 X2 X3	0.420
15	X1 X2 X3 X4	0.420
1	X4	0.470
9	X1 X4	0.480
5	X2 X4	0.490
13	X1 X2 X4	0.500
8	X1	0.525
0	None	0.535
12	X1 X2	0.535
4	X2	0.545

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

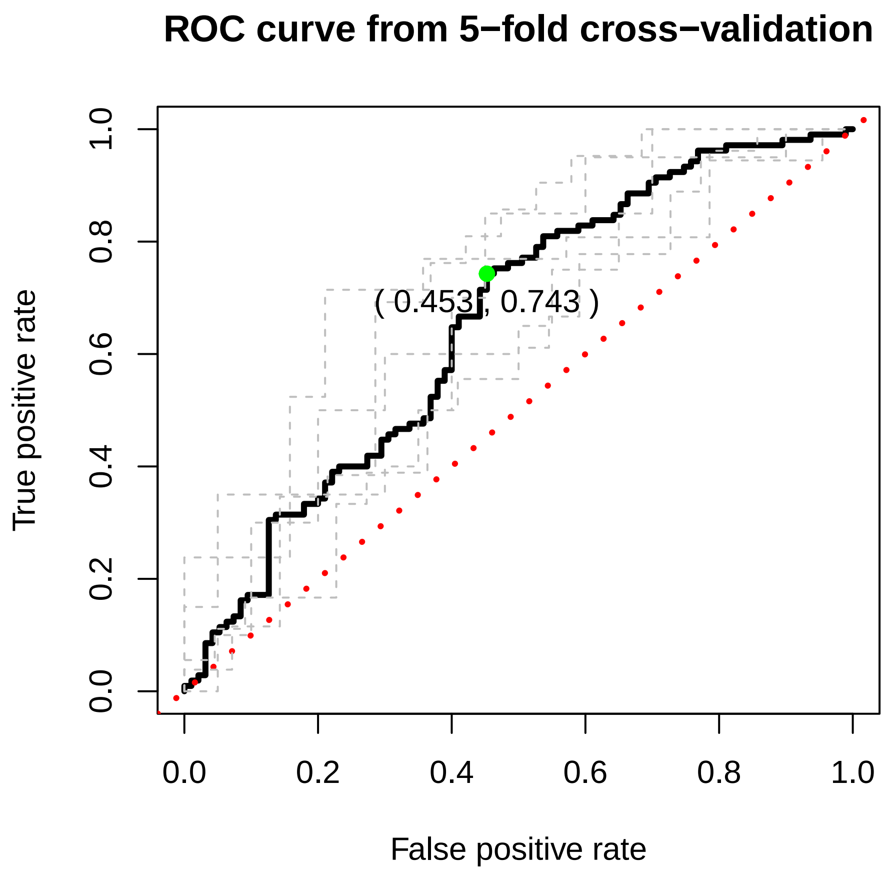
```
## [1] "The Best model ID is  3 with  X3 X4 as features and a MCR of  0.375"
```

```
{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

```r
# select the best model using a 0.5 threshold
best_model = ord_res_df[1,1]

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

# Get the full ROC curve for entire prediction vs Y
pred = prediction(Y_pred, Y)
perf_all = performance(pred,'tpr','fpr')

# get threshold, tpr and fpr values from ROCR
threshold_df = data.frame(cut=perf_all@alpha.values[[1]],
                          fpr=perf_all@x.values[[1]],
                          tpr=perf_all@y.values[[1]])

# create ROC curve for each CV in best model
inds.part = myCVids(n, 5, seed=0)
perf_lst = list()
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')

  perf_df = data.frame(cut=perf@alpha.values[[1]],
                       fpr=perf@x.values[[1]],
                       tpr=perf@y.values[[1]])
  perf_lst = append(perf_lst, list(perf_df))

  # use this code to plot the folds seperately
  # {plot(perf,
  #      lwd=2,
  #      main='ROC curve from 5-fold cross-validation')
  # abline(0,1,lty=2,lwd=2,col="red")}

}

# Create data frame with all cv performance data data
cv_threshold_df = data.frame(cut=threshold_df$cut)
for(i in 1:k_max){
  cv_threshold_df = merge(cv_threshold_df, perf_lst[[i]], by="cut", all=TRUE)
  colnames(cv_threshold_df)[i*2] = paste("fpr_",as.character(i), sep="")
  colnames(cv_threshold_df)[i*2+1] = paste("tpr_",as.character(i), sep="")
}
cv_threshold_df = cv_threshold_df[order(cv_threshold_df$cut, decreasing=TRUE),]
rownames(cv_threshold_df) = NULL
cv_threshold_df = cv_threshold_df %>% fill(colnames(cv_threshold_df)[-1],
                                           .direction="down")

# calculate the best threshold value
threshold_df$tpr_fpr_dist = sqrt(
  ((1-threshold_df$tpr)**2)+(threshold_df$fpr**2)
  )
best_threshold_df = threshold_df[which.min(threshold_df$tpr_fpr_dist),]
# visualize ROC curve
{plot(perf_all,
     lwd=3,
     main='ROC curve from 5-fold cross-validation',
     xlim=c(0,1),
     ylim=c(0,1))
abline(0,1,lty=3,lwd=3,col="red")
lines(x=cv_threshold_df$fpr_1, y=cv_threshold_df$tpr_1, lty=2, col="grey")
lines(x=cv_threshold_df$fpr_2, y=cv_threshold_df$tpr_2, lty=2, col="grey")
lines(x=cv_threshold_df$fpr_3, y=cv_threshold_df$tpr_3, lty=2, col="grey")
lines(x=cv_threshold_df$fpr_4, y=cv_threshold_df$tpr_4, lty=2, col="grey")
lines(x=cv_threshold_df$fpr_5, y=cv_threshold_df$tpr_5, lty=2, col="grey")
points(x=best_threshold_df$fpr, y=best_threshold_df$tpr, col="green", pch=19)
text(x=best_threshold_df$fpr,
     y=best_threshold_df$tpr,
     pos=1,
     paste("(",round(best_threshold_df$fpr,3),",",round(best_threshold_df$tpr,3),")"))
}
```



```
# Print optimal threshold value with (FPR,TPR) closest to ideal as (0,1)
print(paste("The best threshold value is: ", best_threshold_df$cut))
```

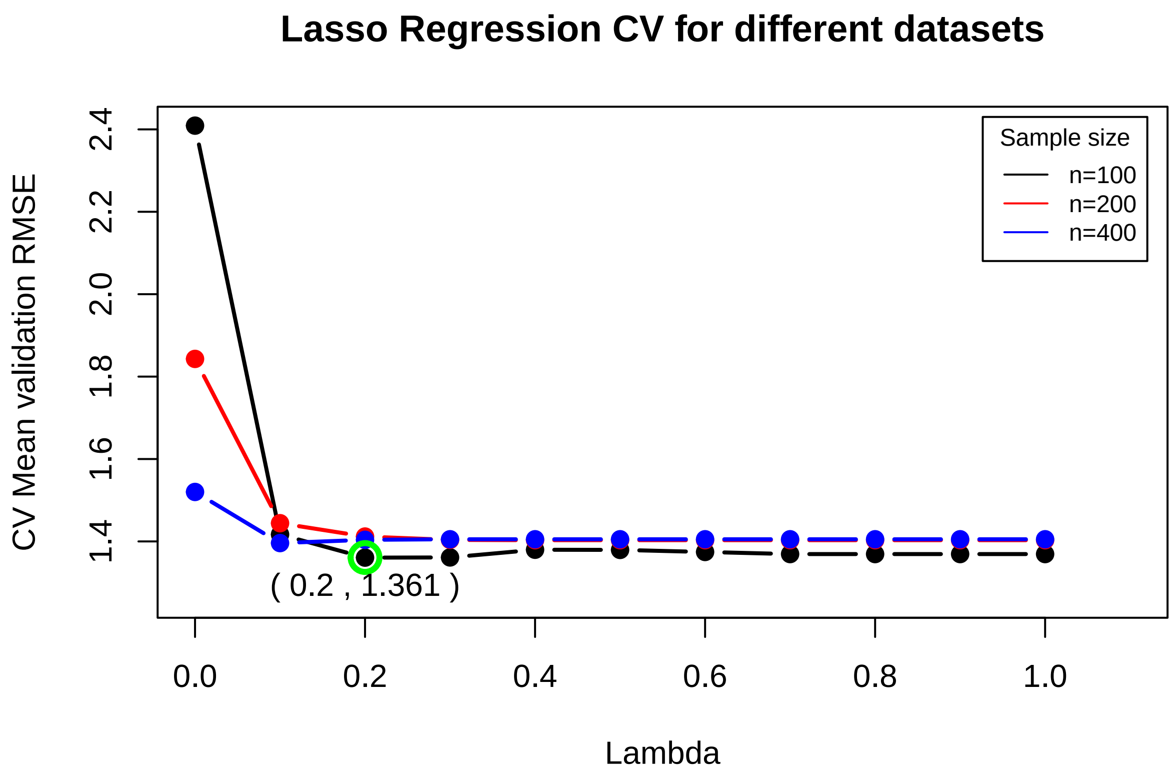
```
## [1] "The best threshold value is:  0.471360246630924"
```

Problem 5

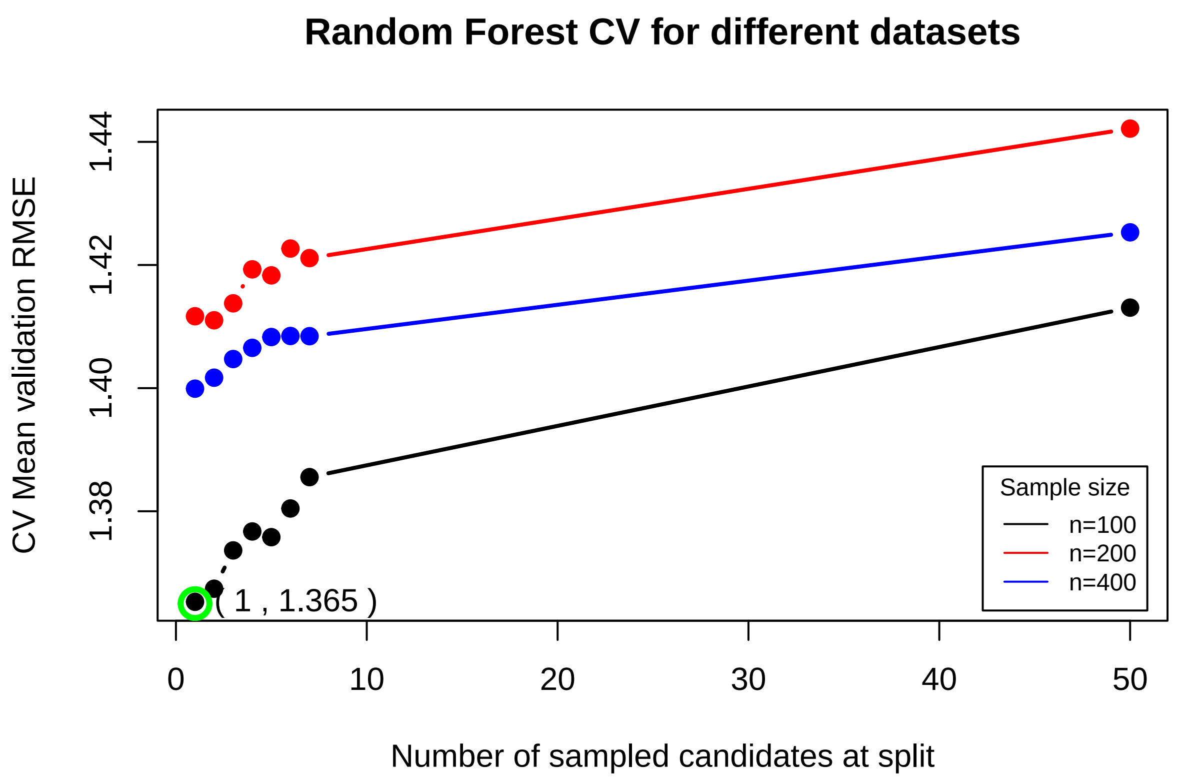
```
p=50
n = nrow(prob5.df)
test_df = prob5.df[401:800,]
x_test_df = model.matrix(Y~., test_df )[,-1]
y_test_df = test_df$Y
data_set_vec = c(100, 200, 400)
k_max = 5
```

```r
# define parameters to search
lambda_vec = seq(0,1,0.1)
mtry_vec = c(1:7,p)
interaction_depth_vec = 1:7

# create lists and data frame for results
lr_dataset_lst = list()
rf_dataset_lst = list()
gbm_dataset_lst = list()
RMSE_df = data.frame(matrix(0, nrow=3, ncol=6))
colnames(RMSE_df) = c("cv_RMSE_n100",
                      "test_RMSE_n100",
                      "cv_RMSE_n200",
                      "test_RMSE_n200",
                      "cv_RMSE_n400",
                      "test_RMSE_n400")

# loop through sets of data
for(dat in 1:length(data_set_vec)){
  dataset = data_set_vec[dat]
  # specify training data set
  train_df = prob5.df[1:dataset,]
  x_train_df = model.matrix(Y~., train_df )[,-1]
  y_train_df = train_df$Y
  # perform 5-fold CV
  set_n = nrow(train_df)
  inds.part = myCVids(set_n, 5, seed=0)

  # create data frames to record results in
  lr_cv_rmse_df = data.frame(matrix(0,
                             nrow=length(lambda_vec),
                             ncol=k_max))
  rf_cv_rmse_df = data.frame(matrix(0,
                             nrow=length(mtry_vec),
                             ncol=k_max))
  gbm_cv_rmse_df = data.frame(matrix(0,
                              nrow=length(interaction_depth_vec),
                              ncol=k_max))

  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
  for(l in 1:length(lambda_vec)){
    lambda = lambda_vec[l]
    # train LR
    set.seed(0)
    lasso_reg.fit = glmnet(x_cv_train_df,
                           y_cv_train_df,
                           alpha=1,
                           lambda=lambda)
    # eval LR
    lr_pred = predict(lasso_reg.fit, x_cv_valid_df)
    lr_rmse = sqrt(mean((y_cv_valid_df - lr_pred)^2))
    lr_cv_rmse_df[l,k] = lr_rmse
  }

  for(m in 1:length(mtry_vec)){
    mtry = mtry_vec[m]
    # train RF
    set.seed(0)
    rf.fit = randomForest(Y~.,
                          data=cv_train_df,
                          mtry=mtry,
                          ntree=500,
                          importance=TRUE)
    # eval RF
    rf_pred = predict(rf.fit, cv_valid_df)
    rf_rmse = sqrt(mean((y_cv_train_df - rf_pred)^2))
    rf_cv_rmse_df[m,k] = rf_rmse
  }

  for(d in 1:length(interaction_depth_vec)){
    interaction_depth = interaction_depth_vec[d]
    # train GBM
    set.seed(0)
    gbm.fit = gbm(Y~.,
                  data=cv_train_df,
                  distribution="gaussian",
                  n.trees=1000,
                  shrinkage=0.01,
                  interaction.depth=interaction_depth)
    # eval GBM
    gbm_pred = predict(gbm.fit, cv_valid_df)
    gbm_rmse = sqrt(mean((y_cv_train_df - gbm_pred)^2))
    gbm_cv_rmse_df[d,k] = gbm_rmse
  }

}

# add data frames to list
lr_cv_rmse_df$mean = rowMeans(lr_cv_rmse_df)
```

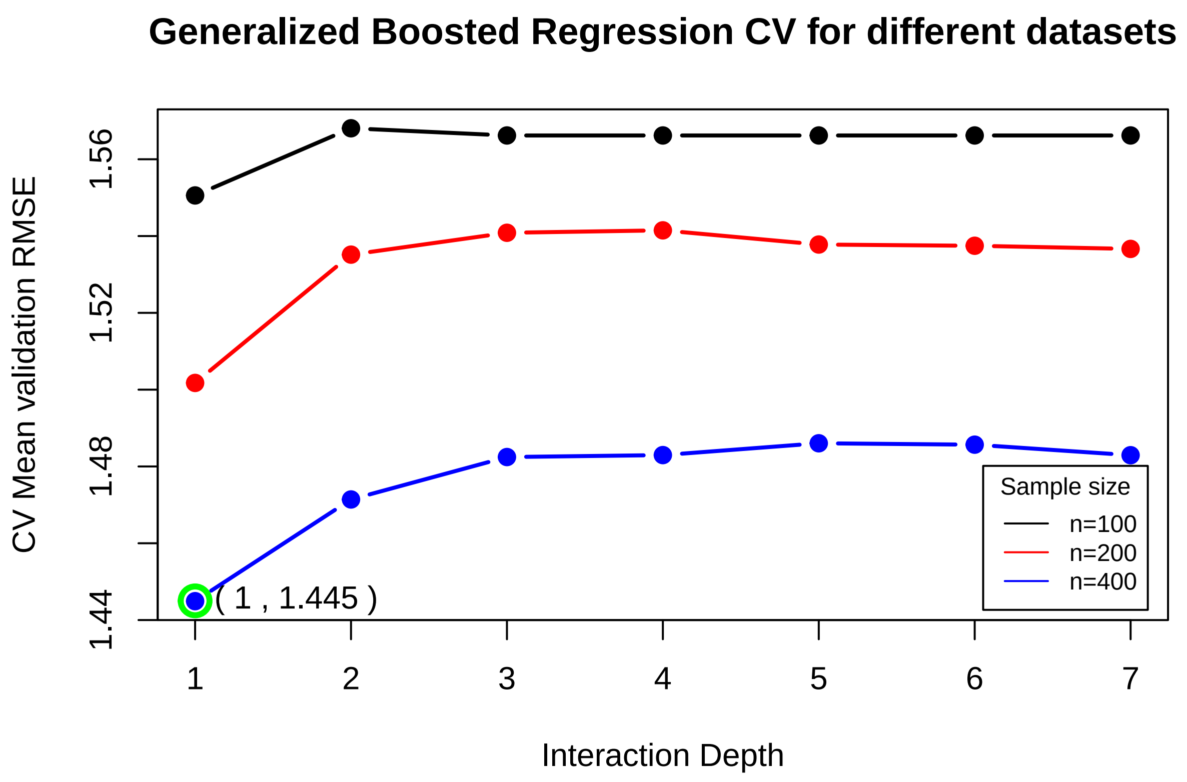
```r
lr_dataset_lst = append(lr_dataset_lst,list(lr_cv_rmse_df))
rf_cv_rmse_df$mean = rowMeans(rf_cv_rmse_df)
rf_dataset_lst = append(rf_dataset_lst, list(rf_cv_rmse_df))
gbm_cv_rmse_df$mean = rowMeans(gbm_cv_rmse_df)
gbm_dataset_lst = append(gbm_dataset_lst, list(gbm_cv_rmse_df))

# select best parameter and get best mean cv RMSE
lr_min_rmse_idx = which.min(lr_cv_rmse_df$mean)
lr_best_par = lambda_vec[lr_min_rmse_idx]
lr_cv_rmse = min(lr_cv_rmse_df$mean)
RMSE_df[1,paste("cv_RMSE_n",dataset,sep="")] = lr_cv_rmse

rf_min_rmse_idx = which.min(rf_cv_rmse_df$mean)
rf_best_par = mtry_vec[rf_min_rmse_idx]
rf_cv_rmse = min(rf_cv_rmse_df$mean)
RMSE_df[2,paste("cv_RMSE_n",dataset,sep="")] = rf_cv_rmse

gbm_min_rmse_idx = which.min(gbm_cv_rmse_df$mean)
gbm_best_par = interaction_depth_vec[gbm_min_rmse_idx]
gbm_cv_rmse = min(gbm_cv_rmse_df$mean)
RMSE_df[3,paste("cv_RMSE_n",dataset,sep="")] = gbm_cv_rmse

# train best LR
set.seed(0)
best_lasso_reg.fit = glmnet(x_train_df,
                       y_train_df,
                       alpha=1,
                       lambda=lr_best_par)
# test best LR
lr_pred = predict(best_lasso_reg.fit, x_test_df)
lr_rmse = sqrt(mean((y_test_df - lr_pred)^2))
RMSE_df[1,paste("test_RMSE_n",dataset,sep="")] = lr_rmse

# train RF
set.seed(0)
best_rf.fit = randomForest(Y~.,
                      data=train_df,
                      mtry=rf_best_par,
                      ntree=500,
                      importance=TRUE)
# eval RF
rf_pred = predict(best_rf.fit, test_df)
rf_rmse = sqrt(mean((y_test_df - rf_pred)^2))
RMSE_df[2,paste("test_RMSE_n",dataset,sep="")] = rf_rmse

# train GBM
set.seed(0)
best_gbm.fit = gbm(Y~.,
              data=train_df,
                distribution="gaussian",
                n.trees=1000,
                shrinkage=0.01,
                interaction.depth=gbm_best_par)
  # eval GBM
  gbm_pred = predict(best_gbm.fit, test_df)
  gbm_rmse = sqrt(mean((y_test_df - gbm_pred)^2))
  RMSE_df[3,paste("test_RMSE_n",dataset,sep="")] = gbm_rmse

}
# name rows in results data frame
rownames(RMSE_df) = c("Lasso_Regression", "Random_Forest", "Generalized_Boosted_Regression")

# visualize cross-validation RMSE
# plot Lasso regression model CV results
# find optimal values
lr_min_lst = which.min(c(min(lr_dataset_lst[[1]]$mean),
                          min(lr_dataset_lst[[2]]$mean),
                          min(lr_dataset_lst[[3]]$mean)))
lr_idx_min = which.min(lr_dataset_lst[[lr_min_lst]]$mean)
lr_y_min = round(lr_dataset_lst[[lr_min_lst]]$mean[lr_idx_min],3)
lr_x_min = lambda_vec[lr_idx_min]

{plot(x=lambda_vec,
     y=lr_dataset_lst[[1]]$mean,
     type="b",
     main="Lasso Regression CV for different datasets",
     xlab="Lambda",
     ylab="CV Mean validation RMSE",
     pch=19,
     lwd=2,
     ylim=c(min(min(lr_dataset_lst[[1]]$mean),
                min(lr_dataset_lst[[2]]$mean),
                min(lr_dataset_lst[[3]]$mean))-0.1,
              max(max(lr_dataset_lst[[1]]$mean),
                max(lr_dataset_lst[[2]]$mean),
                max(lr_dataset_lst[[3]]$mean))),
      xlim=c(0,max(lambda_vec)+(max(lambda_vec)/10)))
lines(x=lambda_vec,
      lr_dataset_lst[[2]]$mean,
      type="b",
      pch=19,
      col="red",
      lwd=2)
lines(x=lambda_vec,
      lr_dataset_lst[[3]]$mean,
      type="b",
      pch=19,
      col="blue",
      lwd=2,
      )
points(x=lr_x_min, y=lr_y_min, cex=2, lwd=3, col="green")
text(x=lr_x_min, y=lr_y_min, pos=1, paste("(",lr_x_min,",",lr_y_min,")"))
legend("topright",
       inset=.02,
       title="Sample size",
       legend=c("n=100",
                "n=200",
                "n=400"),
       lty=c(1,1,1),
       col=c("black","red","blue"),
       bg = "white",
       cex=0.75)}
```



```
# plot Random Forest model CV results
# find optimal values
rf_min_lst = which.min(c(min(rf_dataset_lst[[1]]$mean),
                          min(rf_dataset_lst[[2]]$mean),
                          min(rf_dataset_lst[[3]]$mean)))
rf_idx_min = which.min(rf_dataset_lst[[rf_min_lst]]$mean)
rf_y_min = round(rf_dataset_lst[[rf_min_lst]]$mean[rf_idx_min],3)
rf_x_min = mtry_vec[rf_idx_min]

{plot(x=mtry_vec,
     y=rf_dataset_lst[[1]]$mean,
     type="b",
     main="Random Forest CV for different datasets",
     xlab="Number of sampled candidates at split",
     ylab="CV Mean validation RMSE",
     pch=19,
     lwd=2,
     ylim=c(min(min(rf_dataset_lst[[1]]$mean),
                min(rf_dataset_lst[[2]]$mean),
                min(rf_dataset_lst[[3]]$mean)),
              max(max(rf_dataset_lst[[1]]$mean),
                max(rf_dataset_lst[[2]]$mean),
                max(rf_dataset_lst[[3]]$mean)))
     )
lines(x=mtry_vec,
      rf_dataset_lst[[2]]$mean,
      type="b",
      pch=19,
      col="red",
      lwd=2)
lines(x=mtry_vec,
      rf_dataset_lst[[3]]$mean,
      type="b",
      pch=19,
      col="blue",
      lwd=2)
points(x=rf_x_min, y=rf_y_min, cex=2, lwd=3, col="green")
text(x=rf_x_min, y=rf_y_min, pos=4, paste("(",rf_x_min,",",rf_y_min,")"))
legend("bottomright",
       inset=.02,
       title="Sample size",
       legend=c("n=100",
                "n=200",
                "n=400"),
       lty=c(1,1,1),
       col=c("black","red","blue"),
       bg = "white",
       cex=0.75)}
```



```
# plot gbm model CV results
# find optimal values
gbm_min_lst = which.min(c(min(gbm_dataset_lst[[1]]$mean),
                          min(gbm_dataset_lst[[2]]$mean),
                          min(gbm_dataset_lst[[3]]$mean)))
gbm_idx_min = which.min(gbm_dataset_lst[[gbm_min_lst]]$mean)
gbm_y_min = round(gbm_dataset_lst[[gbm_min_lst]]$mean[gbm_idx_min],3)
gbm_x_min = interaction_depth_vec[gbm_idx_min]

{plot(x=interaction_depth_vec,
     y=gbm_dataset_lst[[1]]$mean,
     type="b",
     main="Generalized Boosted Regression CV for different datasets",
     xlab="Interaction Depth",
     ylab="CV Mean validation RMSE",
     pch=19,
     lwd=2,
     ylim=c(min(min(gbm_dataset_lst[[1]]$mean),
                min(gbm_dataset_lst[[2]]$mean),
                min(gbm_dataset_lst[[3]]$mean)),
            max(max(gbm_dataset_lst[[1]]$mean),
                max(gbm_dataset_lst[[2]]$mean),
                max(gbm_dataset_lst[[3]]$mean)))
     )
lines(x=interaction_depth_vec,
      gbm_dataset_lst[[2]]$mean,
      type="b",
      pch=19,
      col="red",
      lwd=2)
lines(x=interaction_depth_vec,
      gbm_dataset_lst[[3]]$mean,
      type="b",
      pch=19,
      col="blue",
      lwd=2)
points(x=gbm_x_min, y=gbm_y_min, cex=2, lwd=3, col="green")
text(x=gbm_x_min, y=gbm_y_min, pos=4, paste("(",gbm_x_min,",",gbm_y_min,")"))
legend("bottomright",
       inset=.02,
       title="Sample size",
       legend=c("n=100",
                "n=200",
                "n=400"),
       lty=c(1,1,1),
       col=c("black","red","blue"),
       bg = "white",
       cex=0.75)}
```



```
# display table of RMSE results for different data set sizes
####################Briefly discuss your findings; particularly, as n increases.
knitr::kable(RMSE_df)
```

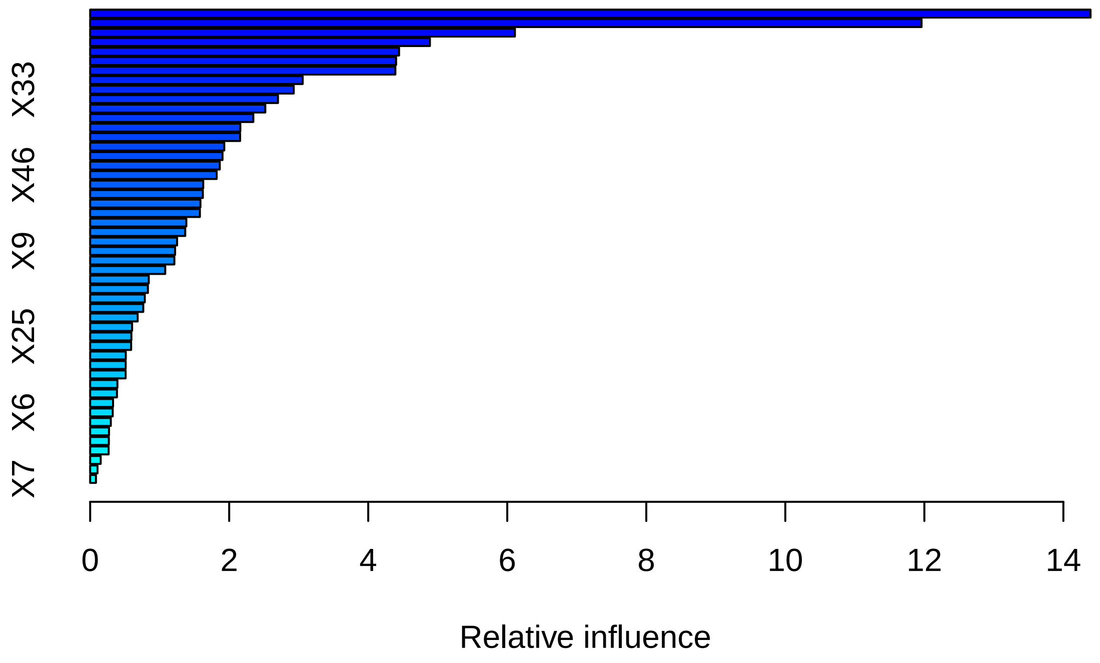
	cv_RMSE_n100	test_RMSE_n100	cv_RMSE_n200	test_RMSE_n200	cv_RMSE_n400	test_RMSE_n400
Lasso_Regression	1.360690	1.399217	1.403066	1.381895	1.396029	1.375865
Random_Forest	1.365293	1.371260	1.411022	1.369140	1.399912	1.371364
Generalized_Boosted_Regression	1.550574	1.491290	1.501748	1.408234	1.444921	1.348527

```
# variable importance of best models
best_rf_var_importance = importance(best_rf.fit)
knitr::kable(best_rf_var_importance)
```

	%IncMSE	IncNodePurity
X1	-0.2270562	14.09000
X2	0.3828024	12.85476
X3	1.5594347	13.95564
X4	-0.5445047	14.70522
X5	-0.0449731	11.90699
X6	-1.4614623	12.82746
X7	-0.2706111	14.15578
X8	-1.3673163	13.56246
X9	0.1461766	13.77700
X10	1.0178126	12.82549
X11	-1.0499736	14.14953
X12	-0.8045081	13.21615
X13	-1.5277839	13.49658
X14	0.4206784	15.05823
X15	-0.5420168	14.31255
X16	-1.2352002	13.56140
X17	-1.5364538	12.07460
X18	0.0223525	13.11206
X19	5.6832410	16.23629
X20	-0.0260533	13.76552
X21	0.7662314	14.76698
X22	1.4741521	12.66539
X23	-1.9847393	12.16836
X24	0.6021159	13.97642
X25	-1.7735024	13.60662
X26	-0.7242413	14.23260
X27	-1.4284636	14.84803
X28	-1.4820672	13.99083
X29	-2.5916505	12.91214
X30	-0.7618064	11.99427
X31	0.8801522	14.83051
X32	-1.2197214	12.26527

	%IncMSE	IncNodePurity
X33	1.2616535	13.13228
X34	-1.6297428	12.88759
X35	-0.7183456	12.76530
X36	-0.5553919	14.49858
X37	3.5865036	16.06830
X38	-1.0526474	13.10167
X39	-0.3178708	13.83954
X40	0.5115375	15.50958
X41	-1.3998038	13.89606
X42	0.5000993	13.84108
X43	0.2732401	15.25078
X44	-0.2120796	13.09160
X45	-1.8413896	13.13325
X46	-0.1302534	13.98108
X47	-0.4973223	13.99525
X48	-1.1067155	12.71878
X49	0.7006539	15.28368
X50	0.4832366	12.30618

```
best_gbm_var_importance = summary(best_gbm.fit)
```



```
knitr::kable(best_gbm_var_importance)
```

	var	rel.inf
X19	X19	14.3906710
X37	X37	11.9599606
X40	X40	6.1100856
X14	X14	4.8873637
X49	X49	4.4437227
X31	X31	4.4010696
X27	X27	4.3906084
X21	X21	3.0572196
X33	X33	2.9281740
X47	X47	2.7017385
X39	X39	2.5194653
X45	X45	2.3471318
X36	X36	2.1599790
X10	X10	2.1573071
X11	X11	1.9291016
X8	X8	1.9040879
X26	X26	1.8645460
X46	X46	1.8199406
X41	X41	1.6263900
X44	X44	1.6210698
X42	X42	1.5862620
X1	X1	1.5786467
X43	X43	1.3835027
X15	X15	1.3678307
X38	X38	1.2507803
X9	X9	1.2221270
X24	X24	1.2113328
X4	X4	1.0799885
X50	X50	0.8431890
X18	X18	0.8308683
X48	X48	0.7859923
X20	X20	0.7640700
X22	X22	0.6832765
X3	X3	0.6033500
X25	X25	0.5928914
X17	X17	0.5895884
X34	X34	0.5121746
X16	X16	0.5107093
X2	X2	0.5101200
X23	X23	0.3919250
X5	X5	0.3857606
X35	X35	0.3291374
X6	X6	0.3244545
X32	X32	0.2972139
X28	X28	0.2711117

	var	rel.inf
X12	X12	0.2687051
X13	X13	0.2654197
X30	X30	0.1513693
X29	X29	0.1054205
X7	X7	0.0831486

```
#### 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.
```

```
# generate additional 50 features
data_df = data.frame(prob5.df, matrix( rnorm(n*50,mean=0,sd=1), n, 50))
# clean_data_df = data_df[,-Y]
# colnames(clean_data_df) = 1:100
# clean_data_df$Y = data_df$Y

# define parameters
p = 100
n = nrow(data_df)
test_df = data_df[401:800,]
x_test_df = model.matrix(Y~., test_df )[,-1]
y_test_df = test_df$Y
data_set_vec = c(100, 200, 400)
k_max = 5

# define parameters to search
lambda_vec = seq(0,1,0.1)
mtry_vec = c(1:7,p)
interaction_depth_vec = 1:7

# create lists and data frame for results
lr_dataset_lst = list()
rf_dataset_lst = list()
gbm_dataset_lst = list()
RMSE_df = data.frame(matrix(0, nrow=3, ncol=6))
colnames(RMSE_df) = c("cv_RMSE_n100",
                      "test_RMSE_n100",
                      "cv_RMSE_n200",
                      "test_RMSE_n200",
                      "cv_RMSE_n400",
                      "test_RMSE_n400")

# loop through sets of data
for(dat in 1:length(data_set_vec)){
  dataset = data_set_vec[dat]
  # specify training data set
  train_df = data_df[1:dataset,]
```

```r
x_train_df = model.matrix(Y~., train_df )[,-1]
y_train_df = train_df$Y
# perform 5-fold CV
set_n = nrow(train_df)
inds.part = myCVids(set_n, 5, seed=0)

# create data frames to record results in
lr_cv_rmse_df = data.frame(matrix(0,
                              nrow=length(lambda_vec),
                              ncol=k_max))
rf_cv_rmse_df = data.frame(matrix(0,
                              nrow=length(mtry_vec),
                              ncol=k_max))
gbm_cv_rmse_df = data.frame(matrix(0,
                               nrow=length(interaction_depth_vec),
                               ncol=k_max))

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

  for(l in 1:length(lambda_vec)){
    lambda = lambda_vec[l]
    # train LR
    set.seed(0)
    lasso_reg.fit = glmnet(x_cv_train_df,
                           y_cv_train_df,
                           alpha=1,
                           lambda=lambda)
    # eval LR
    lr_pred = predict(lasso_reg.fit, x_cv_valid_df)
    lr_rmse = sqrt(mean((y_cv_valid_df - lr_pred)^2))
    lr_cv_rmse_df[l,k] = lr_rmse
  }

  for(m in 1:length(mtry_vec)){
    mtry = mtry_vec[m]
    # train RF
    set.seed(0)
    rf.fit = randomForest(Y~.,
                          data=cv_train_df,
                          mtry=mtry,
                          ntree=500,
                          importance=TRUE)
    # eval RF
    rf_pred = predict(rf.fit, cv_valid_df)
    rf_rmse = sqrt(mean((y_cv_train_df - rf_pred)^2))
    rf_cv_rmse_df[m,k] = rf_rmse
  }

  for(d in 1:length(interaction_depth_vec)){
    interaction_depth = interaction_depth_vec[d]
    # train GBM
    set.seed(0)
    gbm.fit = gbm(Y~.,
                  data=cv_train_df,
                  distribution="gaussian",
                  n.trees=1000,
                  shrinkage=0.01,
                  interaction.depth=interaction_depth)
    # eval GBM
    gbm_pred = predict(gbm.fit, cv_valid_df)
    gbm_rmse = sqrt(mean((y_cv_train_df - gbm_pred)^2))
    gbm_cv_rmse_df[d,k] = gbm_rmse
  }

}

# add data frames to list
lr_cv_rmse_df$mean = rowMeans(lr_cv_rmse_df)
lr_dataset_lst = append(lr_dataset_lst,list(lr_cv_rmse_df))
rf_cv_rmse_df$mean = rowMeans(rf_cv_rmse_df)
rf_dataset_lst = append(rf_dataset_lst, list(rf_cv_rmse_df))
gbm_cv_rmse_df$mean = rowMeans(gbm_cv_rmse_df)
gbm_dataset_lst = append(gbm_dataset_lst, list(gbm_cv_rmse_df))

# select best parameter and get best mean cv RMSE
lr_min_rmse_idx = which.min(lr_cv_rmse_df$mean)
lr_best_par = lambda_vec[lr_min_rmse_idx]
lr_cv_rmse = min(lr_cv_rmse_df$mean)
RMSE_df[1,paste("cv_RMSE_n",dataset,sep="")] = lr_cv_rmse

rf_min_rmse_idx = which.min(rf_cv_rmse_df$mean)
rf_best_par = mtry_vec[rf_min_rmse_idx]
rf_cv_rmse = min(rf_cv_rmse_df$mean)
RMSE_df[2,paste("cv_RMSE_n",dataset,sep="")] = rf_cv_rmse

gbm_min_rmse_idx = which.min(gbm_cv_rmse_df$mean)
gbm_best_par = interaction_depth_vec[gbm_min_rmse_idx]
gbm_cv_rmse = min(gbm_cv_rmse_df$mean)
RMSE_df[3,paste("cv_RMSE_n",dataset,sep="")] = gbm_cv_rmse
# train best LR
set.seed(0)
best_lasso_reg.fit = glmnet(x_train_df,
                            y_train_df,
                            alpha=1,
                            lambda=lr_best_par)
# test best LR
lr_pred = predict(best_lasso_reg.fit, x_test_df)
lr_rmse = sqrt(mean((y_test_df - lr_pred)^2))
RMSE_df[1,paste("test_RMSE_n",dataset,sep="")] = lr_rmse

# train RF
set.seed(0)
best_rf.fit = randomForest(Y~.,
                           data=train_df,
                           mtry=rf_best_par,
                           ntree=500,
                           importance=TRUE)
# eval RF
rf_pred = predict(best_rf.fit, test_df)
rf_rmse = sqrt(mean((y_test_df - rf_pred)^2))
RMSE_df[2,paste("test_RMSE_n",dataset,sep="")] = rf_rmse

# train GBM
set.seed(0)
best_gbm.fit = gbm(Y~.,
            data=train_df,
            distribution="gaussian",
            n.trees=1000,
            shrinkage=0.01,
            interaction.depth=gbm_best_par)
# eval GBM
gbm_pred = predict(best_gbm.fit, test_df)
gbm_rmse = sqrt(mean((y_test_df - gbm_pred)^2))
RMSE_df[3,paste("test_RMSE_n",dataset,sep="")] = gbm_rmse

}
```

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

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

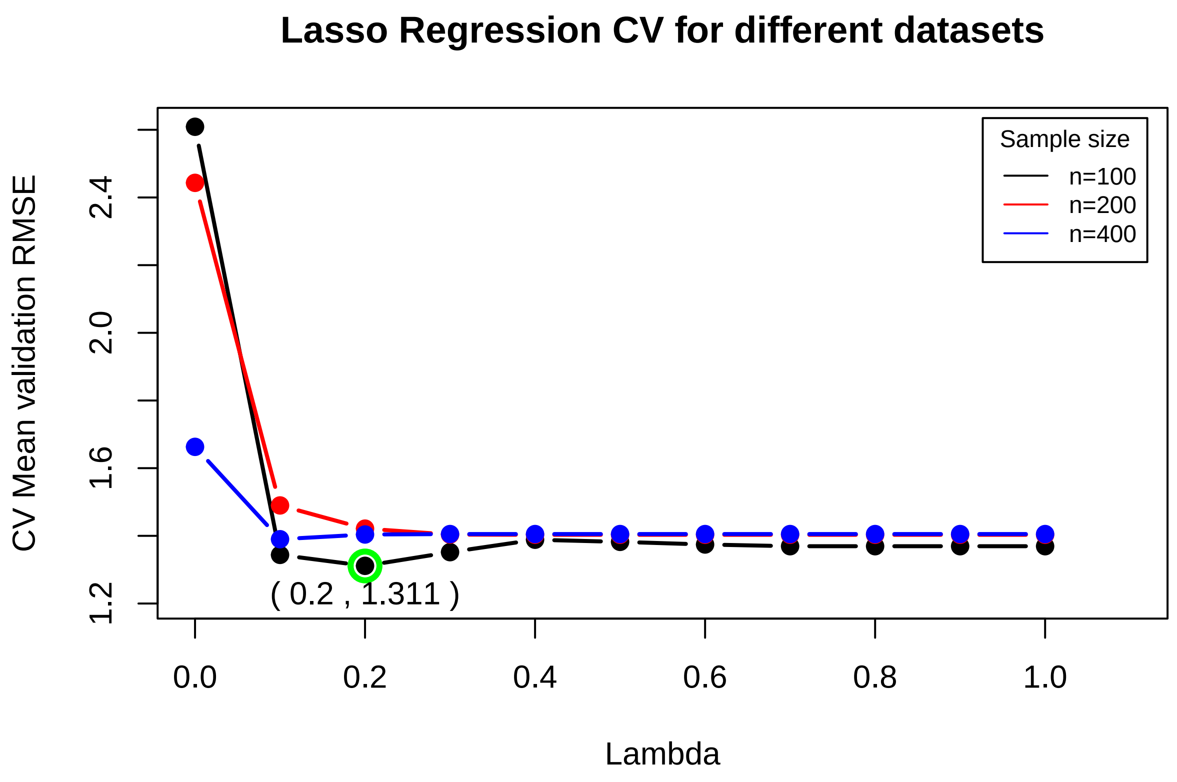
```
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using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

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

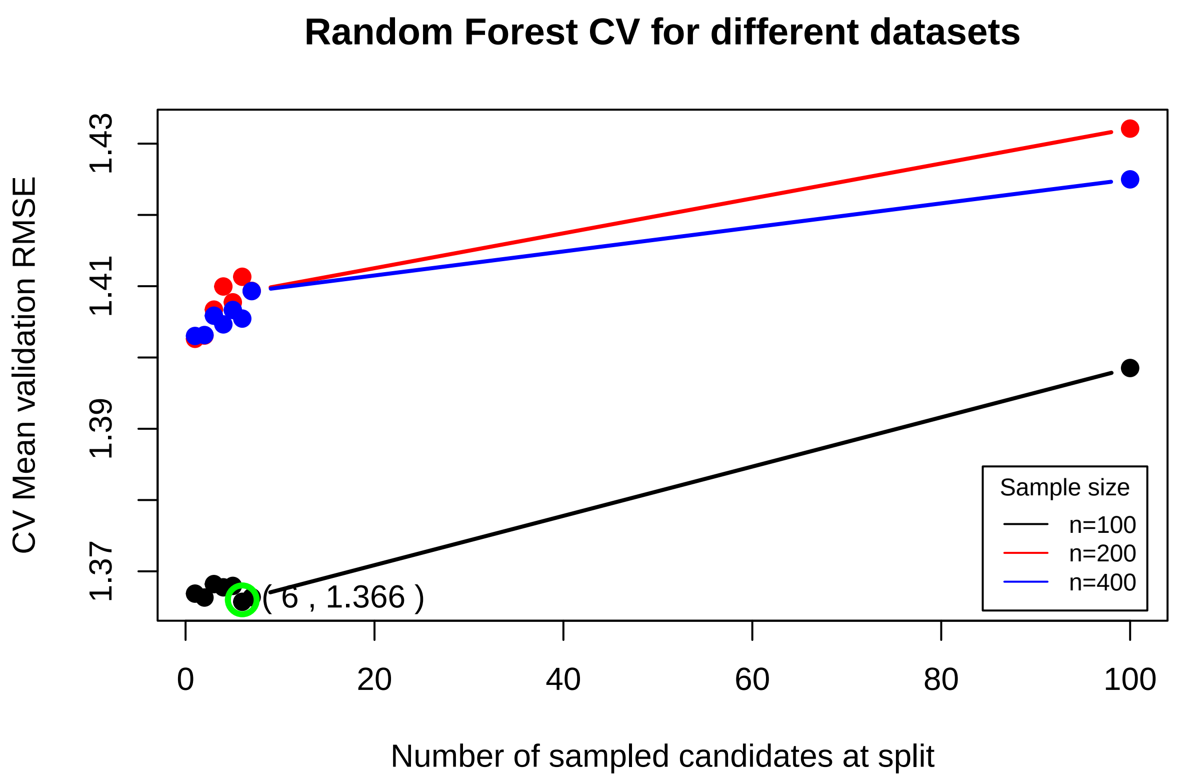
```
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using 1000 trees...
##
## Using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

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

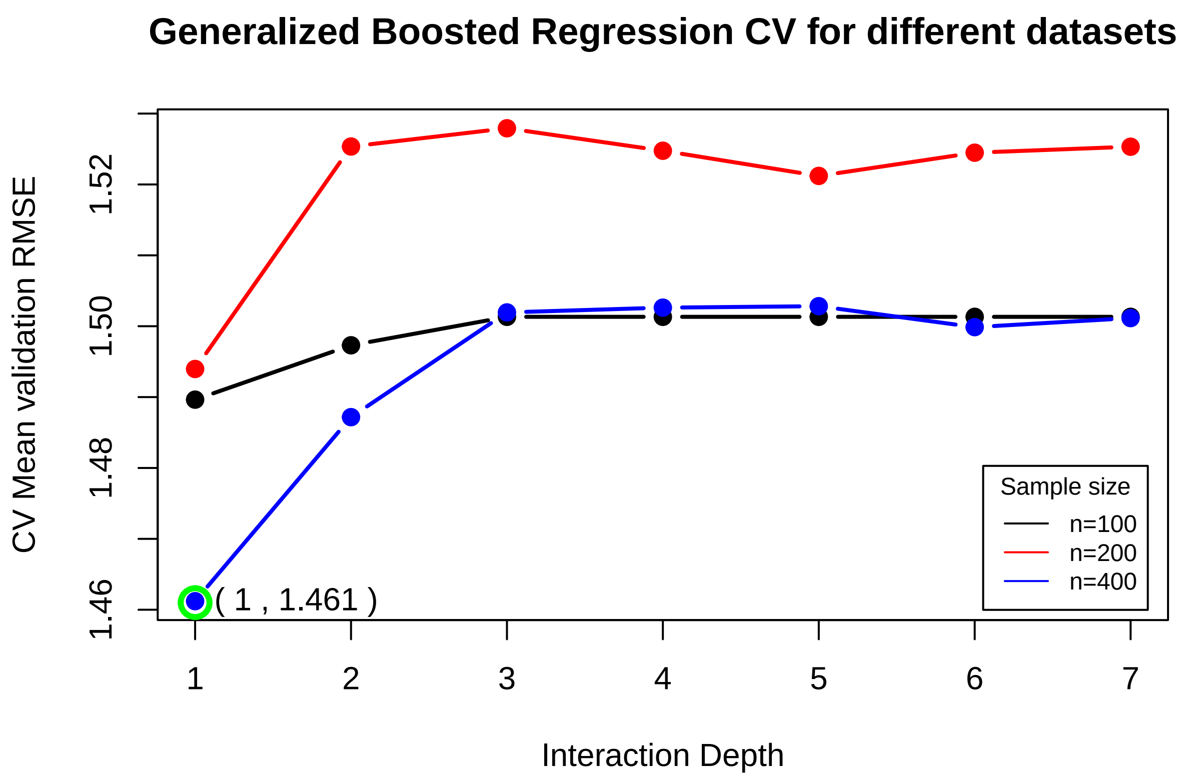
```r
# name rows in results data frame
rownames(RMSE_df) = c("Lasso_Regression", "Random_Forest", "Generalized_Boosted_Regression")

# visualize cross-validation RMSE
# plot Lasso regression model CV results
# find optimal values
lr_min_lst = which.min(c(min(lr_dataset_lst[[1]]$mean),
                          min(lr_dataset_lst[[2]]$mean),
                          min(lr_dataset_lst[[3]]$mean)))
lr_idx_min = which.min(lr_dataset_lst[[lr_min_lst]]$mean)
lr_y_min = round(lr_dataset_lst[[lr_min_lst]]$mean[lr_idx_min],3)
lr_x_min = lambda_vec[lr_idx_min]

{plot(x=lambda_vec,
     y=lr_dataset_lst[[1]]$mean,
     type="b",
     main="Lasso Regression CV for different datasets",
     xlab="Lambda",
     ylab="CV Mean validation RMSE",
     pch=19,
     lwd=2,
     ylim=c(min(min(lr_dataset_lst[[1]]$mean),
                min(lr_dataset_lst[[2]]$mean),
                min(lr_dataset_lst[[3]]$mean))-0.1,
              max(max(lr_dataset_lst[[1]]$mean),
                max(lr_dataset_lst[[2]]$mean),
                max(lr_dataset_lst[[3]]$mean))),
      xlim=c(0,max(lambda_vec)+(max(lambda_vec)/10)))
lines(x=lambda_vec,
      lr_dataset_lst[[2]]$mean,
      type="b",
      pch=19,
      col="red",
      lwd=2)
lines(x=lambda_vec,
      lr_dataset_lst[[3]]$mean,
      type="b",
      pch=19,
      col="blue",
      lwd=2,
      )
points(x=lr_x_min, y=lr_y_min, cex=2, lwd=3, col="green")
text(x=lr_x_min, y=lr_y_min, pos=1, paste("(",lr_x_min,",",lr_y_min,")"))
legend("topright",
       inset=.02,
       title="Sample size",
       legend=c("n=100",
                "n=200",
                "n=400"),
       lty=c(1,1,1),
       col=c("black","red","blue"),
       bg = "white",
       cex=0.75)}
```



```
# plot Random Forest model CV results
# find optimal values
rf_min_lst = which.min(c(min(rf_dataset_lst[[1]]$mean),
                          min(rf_dataset_lst[[2]]$mean),
                          min(rf_dataset_lst[[3]]$mean)))
rf_idx_min = which.min(rf_dataset_lst[[rf_min_lst]]$mean)
rf_y_min = round(rf_dataset_lst[[rf_min_lst]]$mean[rf_idx_min],3)
rf_x_min = mtry_vec[rf_idx_min]

{plot(x=mtry_vec,
     y=rf_dataset_lst[[1]]$mean,
     type="b",
     main="Random Forest CV for different datasets",
     xlab="Number of sampled candidates at split",
     ylab="CV Mean validation RMSE",
     pch=19,
     lwd=2,
     ylim=c(min(min(rf_dataset_lst[[1]]$mean),
                min(rf_dataset_lst[[2]]$mean),
                min(rf_dataset_lst[[3]]$mean)),
              max(max(rf_dataset_lst[[1]]$mean),
                max(rf_dataset_lst[[2]]$mean),
                max(rf_dataset_lst[[3]]$mean)))
     )
lines(x=mtry_vec,
      rf_dataset_lst[[2]]$mean,
      type="b",
      pch=19,
      col="red",
      lwd=2)
lines(x=mtry_vec,
      rf_dataset_lst[[3]]$mean,
      type="b",
      pch=19,
      col="blue",
      lwd=2)
points(x=rf_x_min, y=rf_y_min, cex=2, lwd=3, col="green")
text(x=rf_x_min, y=rf_y_min, pos=4, paste("(",rf_x_min,",",rf_y_min,")"))
legend("bottomright",
       inset=.02,
       title="Sample size",
       legend=c("n=100",
                "n=200",
                "n=400"),
       lty=c(1,1,1),
       col=c("black","red","blue"),
       bg = "white",
       cex=0.75)}
```



```
# plot gbm model CV results
# find optimal values
gbm_min_lst = which.min(c(min(gbm_dataset_lst[[1]]$mean),
                          min(gbm_dataset_lst[[2]]$mean),
                          min(gbm_dataset_lst[[3]]$mean)))
gbm_idx_min = which.min(gbm_dataset_lst[[gbm_min_lst]]$mean)
gbm_y_min = round(gbm_dataset_lst[[gbm_min_lst]]$mean[gbm_idx_min],3)
gbm_x_min = interaction_depth_vec[gbm_idx_min]

{plot(x=interaction_depth_vec,
     y=gbm_dataset_lst[[1]]$mean,
     type="b",
     main="Generalized Boosted Regression CV for different datasets",
     xlab="Interaction Depth",
     ylab="CV Mean validation RMSE",
     pch=19,
     lwd=2,
     ylim=c(min(min(gbm_dataset_lst[[1]]$mean),
                min(gbm_dataset_lst[[2]]$mean),
                min(gbm_dataset_lst[[3]]$mean)),
              max(max(gbm_dataset_lst[[1]]$mean),
                max(gbm_dataset_lst[[2]]$mean),
                max(gbm_dataset_lst[[3]]$mean)))
     )
lines(x=interaction_depth_vec,
      gbm_dataset_lst[[2]]$mean,
      type="b",
      pch=19,
      col="red",
      lwd=2)
lines(x=interaction_depth_vec,
      gbm_dataset_lst[[3]]$mean,
      type="b",
      pch=19,
      col="blue",
      lwd=2)
points(x=gbm_x_min, y=gbm_y_min, cex=2, lwd=3, col="green")
text(x=gbm_x_min, y=gbm_y_min, pos=4, paste("(",gbm_x_min,",",gbm_y_min,")"))
legend("bottomright",
       inset=.02,
       title="Sample size",
       legend=c("n=100",
                "n=200",
                "n=400"),
       lty=c(1,1,1),
       col=c("black","red","blue"),
       bg = "white",
       cex=0.75)}
```



```r
# display table of RMSE results for different data set sizes
####################Briefly discuss your findings; particularly, as n increases.
knitr::kable(RMSE_df)
```

	cv_RMSE_n100	test_RMSE_n100	cv_RMSE_n200	test_RMSE_n200	cv_RMSE_n400	test_RMSE_n400
Lasso_Regression	1.311378	1.429326	1.403066	1.381895	1.389988	1.385018
Random_Forest	1.365724	1.386949	1.402618	1.377009	1.403018	1.378467
Generalized_Boosted_Regression	1.489638	1.480139	1.493967	1.434162	1.461208	1.379375

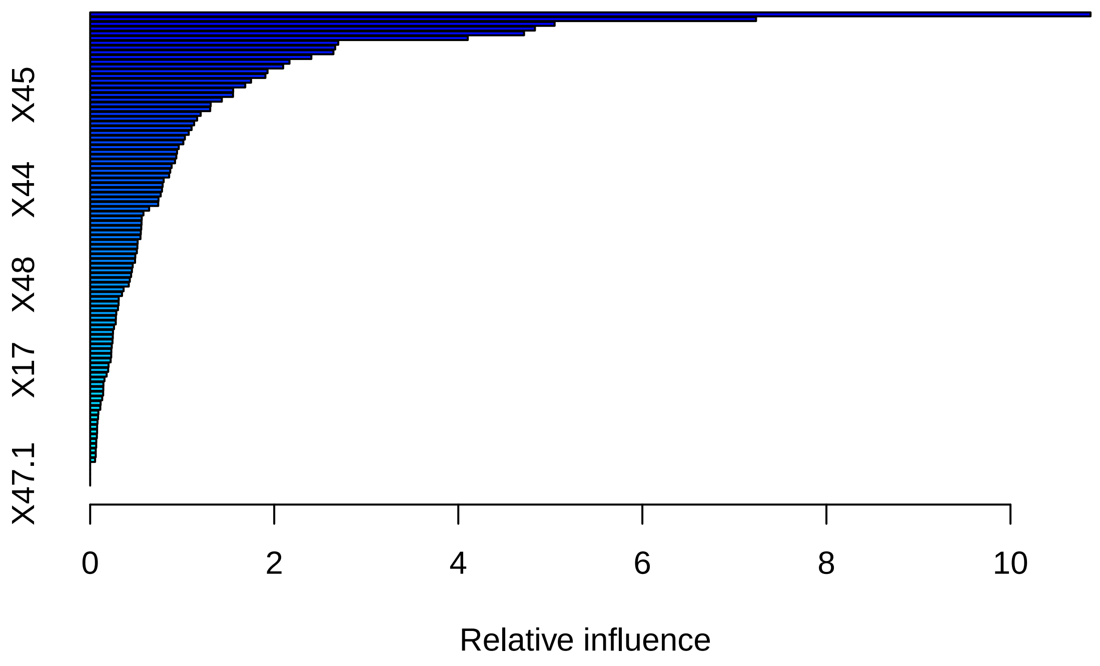
```r
# variable importance of best models
best_rf_var_importance = importance(best_rf.fit)
knitr::kable(best_rf_var_importance)
```

	%IncMSE	IncNodePurity
X1	-1.4703486	6.831852
X2	-0.6335669	6.473027
X3	1.0777003	7.180965
X4	-1.7703127	6.630276
X5	0.1387735	6.348846
X6	-0.1817582	7.443842
X7	-2.6437686	6.090668
X8	-0.3845298	6.642736
X9	-0.0434756	7.011202
X10	1.0381877	6.780919
X11	0.1701114	6.345616
X12	-0.4391972	6.558208
X13	-1.2098991	6.213941
X14	0.9263745	7.395488
X15	0.4541259	7.364527
X16	-1.3357547	6.654630
X17	-0.0034933	6.371431
X18	-1.2732269	6.716546
X19	4.3309206	8.655539
X20	-0.3651086	6.878088
X21	2.6757767	7.259988
X22	-0.4498469	6.658555
X23	-0.8856796	6.532560
X24	-0.5365938	6.844381
X25	-0.8339538	6.606038
X26	-0.4542588	6.490512
X27	0.8580611	6.899337
X28	0.2395604	7.266327
X29	-0.2474976	7.331967
X30	-1.7991226	5.617055
X31	0.9036106	6.900699
X32	-0.0580707	6.457307

	%IncMSE	IncNodePurity
X33	1.6534951	7.127782
X34	0.2660517	6.892015
X35	0.3330019	6.324564
X36	-0.7765627	7.029060
X37	-0.2325843	6.969468
X38	-0.5198927	6.966605
X39	0.7267948	6.421298
X40	0.6415217	8.556990
X41	-0.8645179	5.785437
X42	0.9463792	6.762533
X43	1.6139214	6.555814
X44	-0.3083184	6.617686
X45	0.7500252	6.629571
X46	0.3760283	6.926173
X47	1.0972772	8.121794
X48	-0.4641485	7.449760
X49	0.4701503	7.931179
X50	-0.2155726	6.349418
X1.1	-0.6465747	8.211388
X2.1	2.3621064	7.395358
X3.1	2.3620101	7.897417
X4.1	0.9257706	6.674414
X5.1	-1.0962912	7.602732
X6.1	-0.4146852	7.058994
X7.1	-0.0211359	7.453863
X8.1	1.6267088	6.838765
X9.1	-1.3594050	6.752206
X10.1	-1.4279641	6.506055
X11.1	-0.8300979	6.814416
X12.1	1.6379796	6.424797
X13.1	0.3041693	6.765402
X14.1	2.0393424	6.953001
X15.1	-0.3144224	6.716200
X16.1	-1.6135395	7.535840
X17.1	-1.1365350	6.211980
X18.1	-0.2351343	6.143254
X19.1	-0.6879446	7.645486
X20.1	0.0889272	7.426194
X21.1	-1.1362616	6.803223
X22.1	1.0504542	6.900577
X23.1	-0.3194138	6.545143
X24.1	0.6330683	6.442204
X25.1	2.1710334	7.527827
X26.1	1.6270244	7.198945
X27.1	0.8897853	5.575470
X28.1	-0.7623077	6.608437
X29.1	-0.8798883	6.476336
X30.1	-1.2867057	6.367546

	%IncMSE	IncNodePurity
X31.1	-1.7542267	5.610766
X32.1	-1.0492600	6.858616
X33.1	0.2635814	6.511111
X34.1	0.6995746	6.540755
X35.1	-1.5704607	7.445631
X36.1	0.1112149	6.494960
X37.1	0.1513059	7.245613
X38.1	-0.7812160	6.438477
X39.1	1.3289499	5.803364
X40.1	0.2466277	8.494218
X41.1	-1.8460103	6.283196
X42.1	-0.0564387	6.499763
X43.1	-0.3549130	6.723980
X44.1	0.8517822	6.290129
X45.1	1.0921683	6.274807
X46.1	0.2601070	5.824728
X47.1	0.3492683	6.272117
X48.1	-0.1673526	5.875025
X49.1	0.5480345	5.885674
X50.1	-1.0747286	6.389593

```
best_gbm_var_importance = summary(best_gbm.fit)
```



```
knitr::kable(best_gbm_var_importance)
```

	var	rel.inf
X19	X19	10.8698289
X37	X37	7.2354894
X26.1	X26.1	5.0468776
X40.1	X40.1	4.8328368
X40	X40	4.7146519
X14	X14	4.1031241
X2.1	X2.1	2.6958881
X31	X31	2.6626978
X49	X49	2.6434173
X27	X27	2.4035030
X3.1	X3.1	2.1660815
X16.1	X16.1	2.0972969
X21	X21	1.9275320
X13.1	X13.1	1.9042302
X8.1	X8.1	1.7483180
X47	X47	1.6850999
X6.1	X6.1	1.5536643
X45	X45	1.5524068
X20.1	X20.1	1.4310021
X14.1	X14.1	1.3097034
X41.1	X41.1	1.3044040
X26	X26	1.2011605
X39	X39	1.1624463
X36	X36	1.1298275
X10	X10	1.1024309
X33	X33	1.0713118
X19.1	X19.1	1.0293633
X46	X46	1.0125735
X9	X9	0.9627287
X24	X24	0.9442857
X44.1	X44.1	0.9362490
X22.1	X22.1	0.9220569
X33.1	X33.1	0.8852675
X32.1	X32.1	0.8702235
X10.1	X10.1	0.8582209
X49.1	X49.1	0.7988554
X42	X42	0.7877123
X44	X44	0.7807229
X24.1	X24.1	0.7667257
X37.1	X37.1	0.7425487
X11	X11	0.7398264
X30.1	X30.1	0.6410342
X11.1	X11.1	0.5792023
X8	X8	0.5605114
X22	X22	0.5583144

	var	rel.inf
X41	X41	0.5554375
X25.1	X25.1	0.5504700
X39.1	X39.1	0.5468354
X35.1	X35.1	0.5159906
X1.1	X1.1	0.5130187
X38.1	X38.1	0.5074655
X15.1	X15.1	0.4903955
X7.1	X7.1	0.4877544
X34.1	X34.1	0.4622069
X38	X38	0.4538266
X20	X20	0.4453805
X21.1	X21.1	0.4310602
X48	X48	0.4197919
X42.1	X42.1	0.3635822
X43	X43	0.3461536
X50	X50	0.3105958
X4	X4	0.3089263
X15	X15	0.3022787
X36.1	X36.1	0.2842179
X1	X1	0.2796074
X45.1	X45.1	0.2781797
X25	X25	0.2592592
X2	X2	0.2481224
X28.1	X28.1	0.2462302
X5.1	X5.1	0.2427427
X27.1	X27.1	0.2346934
X43.1	X43.1	0.2302709
X50.1	X50.1	0.2289686
X29.1	X29.1	0.2225051
X35	X35	0.1999501
X17	X17	0.1960914
X31.1	X31.1	0.1792910
X17.1	X17.1	0.1555669
X12.1	X12.1	0.1439322
X34	X34	0.1428590
X13	X13	0.1415661
X18.1	X18.1	0.1310084
X23	X23	0.1138216
X3	X3	0.1100310
X32	X32	0.0893083
X16	X16	0.0865146
X12	X12	0.0798458
X6	X6	0.0759214
X46.1	X46.1	0.0757678
X18	X18	0.0729878
X5	X5	0.0666470
X28	X28	0.0657509
X30	X30	0.0609829

	var	rel.inf
X48.1	X48.1	0.0608925
X4.1	X4.1	0.0536417
X7	X7	0.0000000
X29	X29	0.0000000
X9.1	X9.1	0.0000000
X23.1	X23.1	0.0000000
X47.1	X47.1	0.0000000

```
################################## describe what difference feature numbers make
##### 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] "____________"
```