Ensemble Methods Homework

Dmitrii Dunin, ITU ID 94739

International Technological University

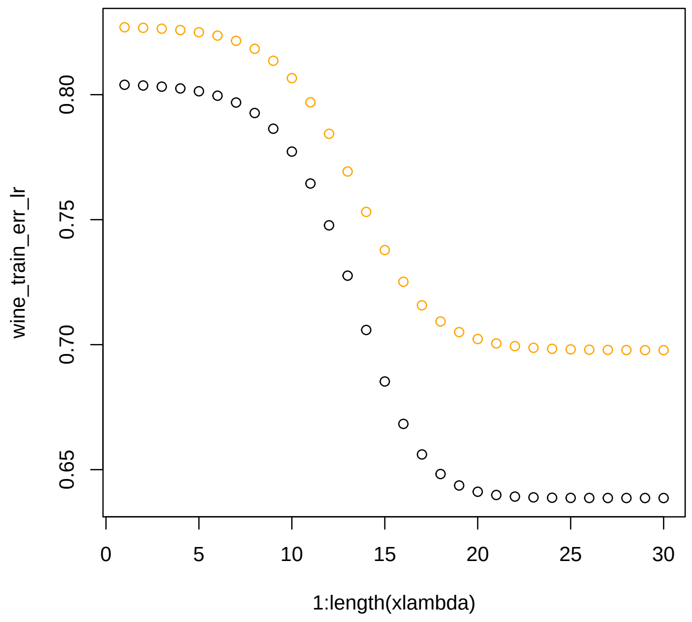
Question 1

```
> rm(list=ls())
> require(gbm)
> setwd("F:/Workspace/R/Homework4")
> par(mar=rep(4, 4))
```

1) Classify the wine quality data using some Boosted method. Ada Boost won't work. What does? Hints for gradient boosted method: require(gbm) and gbm.perf(gbm1,method="cv")

Answer 1

```
> library("ridge")
> wine_data <- read.csv("winequality-red.csv", header=TRUE, sep=';')
> wine_train = wine_data[1:1400,]
> wine_test = wine_data[1401:dim(wine_data)[1],]
> rmse=function(X,Y){
+   return( sqrt(sum((X-Y)^2)/length(Y)) )
+ }
> ptm <- proc.time()
> xlambda=rep(0, times = 30)
> for(i in seq(from = 0, to = 29)){
+ exp <- (+3 -4*(i/20))
+ xlambda[i+1] <- 10^exp
+ }
> wine_train_err_lr = rep(0, times = length(xlambda))
> wine_test_err_lr = rep(0, times = length(xlambda))
> min_lambda = 10000
> y_train = wine_train[,12]
> y_test = wine_test[,12]
> for(i in 1:length(xlambda)){
+ lrmodel = linearRidge(quality~., data = wine_train, lambda = xlambda[i])
+ wine_train_err_lr[i] = rmse(y_train, predict(lrmodel, newdata = wine_train[,-12]))
+
+ wine_test_err_lr[i] = rmse(y_test, predict(lrmodel, newdata = wine_test[,-12]))
+
+ if(i > 1 && wine_test_err_lr[i] < (wine_test_err_lr[i-1]-0.005)){
+ min_lambda = xlambda[i]
+ index_lambda=i
+   }
+ }

> plot(1:length(xlambda),
+      wine_train_err_lr,
+      ylim=c(min(wine_train_err_lr, wine_test_err_lr),
+             max(wine_train_err_lr, wine_test_err_lr)))
> points(1:length(xlambda),wine_test_err_lr, col='orange')
```



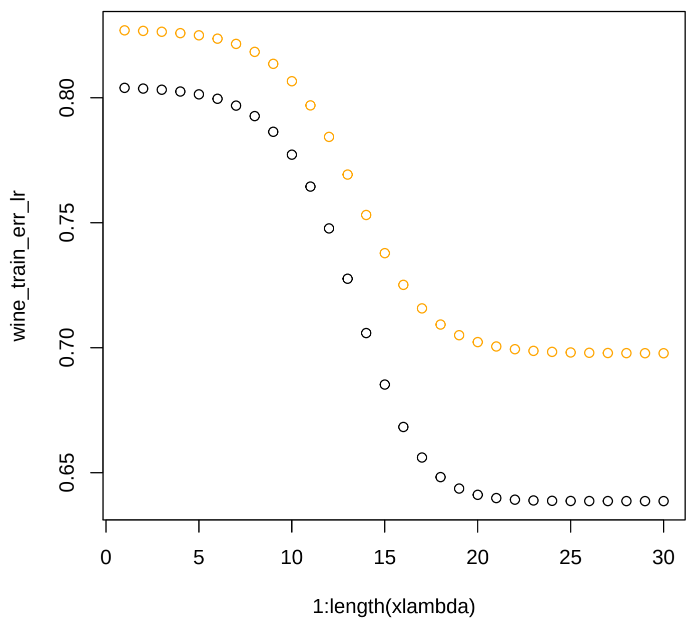
Question 2

```
> rm(list=ls())
> wine_data <- read.csv("winequality-red.csv", header=TRUE, sep=';')
```

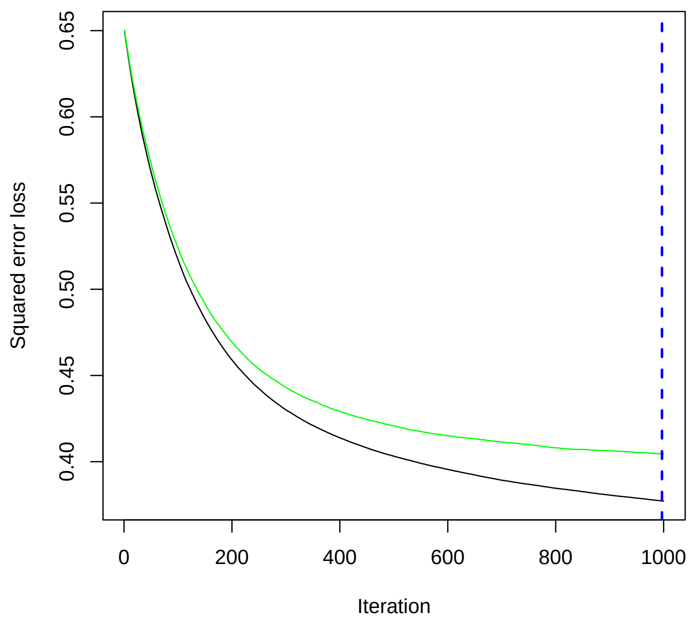
2a) In Homework 3 Problem 2 ridge regression was performed on the wine quality data. Compare the results of using the Boosted method with the ridge regression method. 2b) Use the Random Forest technique on the wine quality data. 2c) Compare the results of Boosted method and Random Forest.

Answer 2

```
> library("ridge")
> wine_train = wine_data[1:1400,]
> wine_test = wine_data[1401:1599,]
> rmse=function(X,Y){
+   return( sqrt(sum((X-Y)^2)/length(Y)) )
+ }
> ptm <- proc.time()
> xlambda=rep(0, times = 30)
> for(i in seq(from = 0, to = 29)){
+ exp <- (+3 -4*(i/20))
+ xlambda[i+1] <- 10^exp
+ }
> wine_train_err_lr = rep(0, times = length(xlambda))
> wine_test_err_lr = rep(0, times = length(xlambda))
> min_lambda = 10000
> y_train = wine_train[,12]
> y_test = wine_test[,12]
> for(i in 1:length(xlambda)){
+ lrmodel = linearRidge(quality~., data = wine_train, lambda = xlambda[i])
+ wine_train_err_lr[i] = rmse(y_train, predict(lrmodel, newdata = wine_train[,-12]))
+
+ wine_test_err_lr[i] = rmse(y_test, predict(lrmodel, newdata = wine_test[,-12]))
+
+ if(i > 1 && wine_test_err_lr[i] < (wine_test_err_lr[i-1]-0.005)){
+ min_lambda = xlambda[i]
+ index_lambda=i
+   }
+ }

> plot(1:length(xlambda),
+      wine_train_err_lr,
+      ylim=c(min(wine_train_err_lr,wine_test_err_lr),
+             max(wine_train_err_lr,wine_test_err_lr)))
> points(1:length(xlambda),wine_test_err_lr, col='orange')
```



```
> lrmodel2 = linearRidge(quality~., data = wine_train, lambda=min_lambda)
> ptm_lr = proc.time() - ptm
> wine_train_err_lr2 = rmse(y_train, predict(lrmodel2, newdata = wine_train[,-12]))
> wine_test_err_lr2 = rmse(y_test, predict(lrmodel2, newdata = wine_test[,-12]))
> ptm <- proc.time()
> gbm1 <- gbm(quality~.,
+             distribution="gaussian",
+             data=wine_data,
+             n.trees = 1000,
+             interaction.depth=1,
+             cv.folds=10,
+             shrinkage=0.01)
> best.iter = gbm.perf(gbm1,method="cv")
> ptm_bm = proc.time() - ptm
> wine_train_error_bm =  min(gbm1$train.error)
> wine_test_error_bm =  min(gbm1$cv.error)
```



```
> cat(index_lambda, "th ", "lambda is optimal: ", min_lambda, "\n")

18 th  lambda is optimal:  0.3981072

> sprintf("Ridge training error: %f", wine_train_err_lr2)

[1] "Ridge training error: 0.648245"

> sprintf("Ridge test: %f",wine_test_err_lr2)

[1] "Ridge test: 0.709274"

> sprintf("Ridge regression Running time: %f sec",(ptm_lr[3]))

[1] "Ridge regression Running time: 0.360000 sec"

> cat("\n")


> sprintf("Boosting training error: %f", wine_train_error_bm)

[1] "Boosting training error: 0.377137"

> sprintf("Boosting test error: %f", wine_test_error_bm)
```

```
[1] "Boosting test error: 0.404562"

> sprintf("1000 trees boost Running time: %f sec", ptm_bm[3])

[1] "1000 trees boost Running time: 3.750000 sec"

> library(randomForest)
> x_train<-wine_train[,-12]
> y_train<-wine_train[,12]
> x_test<-wine_test[,-12]
> y_test<-wine_test[,12]
> ptm <- proc.time()
> fit<-randomForest(x_train,
+                   y=y_train,
+                   xtest=x_test,
+                   ytest=y_test,
+                   ntree=1000,
+                   oob.prox=FALSE)
> ptm_rf = proc.time() - ptm
> wine_train_err_rf = min(fit$mse)
> wine_test_err_rf = min(fit$test$mse)
> wine_train_err_rf

[1] 0.3075949

> wine_test_err_rf

[1] 0.4488564

> sprintf("Random forest training error: %f", wine_train_err_rf)

[1] "Random forest training error: 0.307595"

> sprintf("Random forest test error: %f", wine_test_err_rf)

[1] "Random forest test error: 0.448856"

> sprintf("Random forest Running time: %f sec", ptm_rf[3])

[1] "Random forest Running time: 3.750000 sec"

> results = data.frame(Training.Error=c(wine_train_err_lr2,
+                                       wine_train_error_bm,
+                                       wine_train_err_rf),
+                      Test.Error=c(wine_test_err_lr2,
+                                   wine_test_error_bm,
+                                   wine_test_err_rf),
+                      Running.Time=c(ptm_lr[3], ptm_bm[3], ptm_rf[3]))
> row.names(results) <- c("Ridge regression", "Boosted method", "Random Forest")
> print(results)
```

```
                Training.Error Test.Error Running.Time
Ridge regression     0.6482447  0.7092735         0.36
Boosted method       0.3771367  0.4045618         3.75
Random Forest        0.3075949  0.4488564         3.75
```

Question 3

```
> rm(list=ls())
> sonar_train <- read.csv("sonar_train.csv", header=FALSE)
> sonar_test <- read.csv("sonar_test.csv", header=FALSE)
> sonar_hold = rbind(sonar_train, sonar_test)
> rm(sonar_train, sonar_test)
> set_seed <- function(i) {
+   set.seed(i)
+   if (exists(".Random.seed"))  oldseed <- get(".Random.seed", .GlobalEnv)
+   if (exists(".Random.seed"))  assign(".Random.seed", oldseed, .GlobalEnv)
+ }
```

3) The objective of this problem is to see how the number of observations affects over fitting. If the number of observations is close to the number of attributes, an ordinary least square linear model will be over fit.

3a) Start with the full sonar data set. Estimate the error of an ordinary least squared linear model on this data set by using 5 fold cross validation on the whole data set. Average the 5 training errors from each of these runs. Average the 5 test errors from each of these runs. What are these averages?

3b) Next run a series of cross validations on a series of data sets which are decreasing in size (decrease the number of observations). (At a minimum you must have more observations than attributes to use lm. Also, make sure that your smallest data set has at least one observation of both rock and mine.) Record the training and test errors for each data set size.

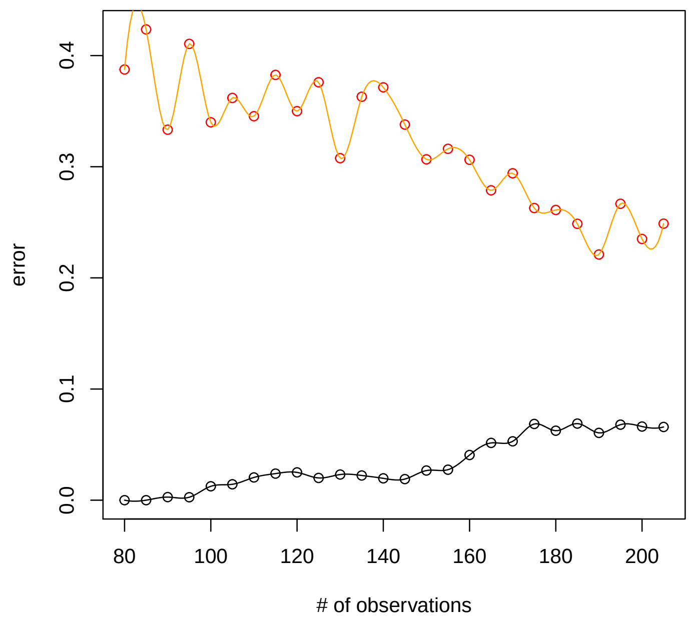
3c) Plot both the training error and test error verses data set size on the same graph. The horizontal axis should be the number of observations in the data set and the vertical axis should be error rates.

Answer 3

```
> k = 5
> num_sample = nrow(sonar_hold)
> seed_val=123
> set_seed(seed_val)
> sonar_full = sonar_hold[sample(num_sample, num_sample, replace=FALSE),]
> sonar_scaled = as.data.frame(cbind(scale(sonar_full[,-61]), sonar_full[,61]))
> pick = k
> sonar_train = sonar_scaled [1:(num_sample*((k-1)/k)),]
> sonar_cv = sonar_scaled [1:(num_sample*(1/k)),]
> error_train = 0
> error_cv = 0
> for(j in 1:k){
+   i_tmp = 1
+   for(i in 1:k){
+     if(i == pick){
+       sonar_cv = sonar_scaled[((i-1)*num_sample/k+1):(num_sample*(i/k)),]
+     } else {
+       sonar_train[((i_tmp-1)*num_sample/k+1):(num_sample*(i_tmp/k)), ] =
+         sonar_scaled[((i-1)*num_sample/k+1):(num_sample*i/k),]
+       i_tmp = i_tmp + 1
+     }
+   }
+
+   pick = pick - 1
+   y_sonar_train = sonar_train[,61]
+   x_sonar_train = sonar_train[,-61]
+   y_sonar_cv = sonar_cv[,61]
+   x_sonar_cv = sonar_cv[,-61]
+   lmodel = lm(V61~., sonar_train)
+   yp = (y_sonar_train >= 0.0)
+   yh = predict(lmodel, x_sonar_train)
+   yhp = (yh > 0.0)
+   error_train = error_train + sum(yhp != yp)/(length(y_sonar_train)*k)
+   yp = (y_sonar_cv >= 0.0)
+   yh = predict(lmodel, x_sonar_cv)
+   yhp = (yh > 0.0)
+   error_cv= error_cv + sum(yhp != yp)/(length(y_sonar_cv)*k*0.00001/0.00001)
+   }
> sprintf("5-fold training error: %f", error_train)

[1] "5-fold training error: 0.068675"

> sprintf("5-fold cross validation error: %f", error_cv)

[1] "5-fold cross validation error: 0.268293"

> k = 5
> num_sample = nrow(sonar_hold)-3
> seed_val=123
> pick = k #pick kth set
> error_train=rep(0, times=26)
> error_cv=rep(0, times=26)
> id_size = 1
> list_sample = c(0)
> while((num_sample*(k-1)/k) > 60){
+ while(sum(sonar_full[,61]==-1)==0 | sum(sonar_full[,61]==1)==0){
```

```
+ seed_val = seed_val + 1
+ set_seed(seed_val)
+ sonar_full = sonar_hold[sample(num_sample, num_sample, replace=FALSE),]
+   }
+ sonar_scaled = as.data.frame(cbind(scale(sonar_full[,-61]), sonar_full[,61]))
+ sonar_train = sonar_scaled[1:(num_sample*((k-1)/k)),]
+ sonar_cv = sonar_scaled[1:(num_sample*(1/k)),]
+ pick=k
+ for(j in 1:k){
+ i_tmp = 1
+ for(i in 1:k){
+ if(i == pick){
+ sonar_cv = sonar_scaled[((i-1)*num_sample/k+1):(num_sample*(i/k)),]
+ } else {
+ sonar_train[((i_tmp-1)*num_sample/k+1):(num_sample*(i_tmp/k)), ] =
+   sonar_scaled[((i-1)*num_sample/k+1):(num_sample*i/k),]
+ i_tmp = i_tmp + 1
+   }
+     }
+ pick = pick - 1
+ y_sonar_train = sonar_train[,61]
+ x_sonar_train = sonar_train[,-61]
+ y_sonar_cv = sonar_cv[,61]
+ x_sonar_cv = sonar_cv[,-61]
+ lmodel = lm(V61~., sonar_train)
+ length(sonar_train[,61])
+ yp = (y_sonar_train >= 0.0)
+ yh = predict(lmodel, x_sonar_train)
+ yhp = (yh > 0.0)
+ error_train[id_size] = error_train[id_size] + sum(yhp != yp) /
+                        (length(y_sonar_train)*k*0.00001/0.00001)
+ yp = (y_sonar_cv >= 0.0)
+ yh = predict(lmodel, x_sonar_cv)
+ yhp = (yh > 0.0)
+ error_cv[id_size]= error_cv[id_size] + sum(yhp != yp) /
+                     (length(y_sonar_cv)*k*0.00001/0.00001)
+       }
+ list_sample[id_size]=num_sample
+ num_sample = num_sample-5
+ id_size = id_size + 1
+ }
> pt_overfit=list_sample[which(max(error_cv-error_train)==(error_cv-error_train))]
> sprintf("When there are %d of observations",pt_overfit)

[1] "When there are 85 of observations"

> sprintf("differnece between cross-validation")
```

```
[1] "differnece between cross-validation"

> sprintf("error and training error is highest.")

[1] "error and training error is highest."

> plot(list_sample,
+       error_train,
+       ylim=c(min(error_train, error_cv),
+              max(error_train, error_cv)),
+       ylab="error",
+       xlab="# of observations")
> points(list_sample, error_cv, col='red')
> lines(spline(list_sample, error_train,n=200))
> lines(spline(list_sample, error_cv,n=200), col="orange")
```



Question 4

4) This problem illustrates some of the concepts of ensemble methods. (For debugging purposes you may want to use the command set.seed(pi) to make the runs consistent.)

4a) From problem 3, pick a data set size that is clearly over fit (ie. The test error is much greater than the training error). Try to improve the result with an ensemble method. Use the small sonar data set that you have chosen as the training set and put the rest of the data into the hold out set. Generate 10 linear models using only your training data set. Each of these models will incorporate a different random subset of the attributes. Before you start, fix n to be a number between 5 and 30. To generate one of these linear models:

A) Choose n attributes randomly from the 60 available attributes (without replacement). For example if you fixed n to be 11 then choose 11 attributes randomly (without replacement) from the 60 available sonar attributes.

B) Fit a linear model to the training set using only these n attributes.

C) Use this model to make predictions on both the training set and the hold out set. These will become one of the attributes in the ensemble model training set. This will become one attribute for problem 4b.

4b) In this step, use linear regression to create an ensemble model. Treat the output of the 10 linear models (from step C above) as inputs to a new regression to create the ensemble model. (See the figure below.) You now have 10 new attributes for each observation (one from each of the predictions you made in step C above.) The next step is to perform the linear regression over the ensemble training set which only has the 10 new attributes. (Ignore the original 60 attributes.)

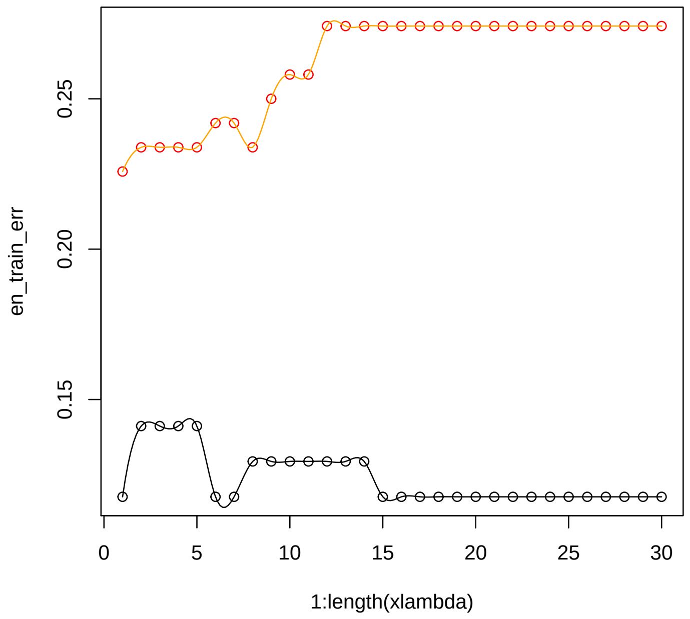
4c) Repeat Homework problem 4b with various values for n (the number of randomly chosen attributes). In part 4aA of the example above, n was fixed to be 11. Now, put n in a loop. That is in pseudo code: for (n in seq(5,30,by=5))... create ensemble model. Plot the training error and test error as a function of n. How well did the new ensemble models do in comparison to the original model created by problem 3?

4d) Compare the performance of your best ensemble model on the hold out set with the performance of the ordinary least square model (problem 3b) on the hold out set. Be careful in how you make the comparisons. Say your sonar training set had 70 rows, with the remaining rows 138 rows in your test set. The same training set must be used in both 3b and the ensemble model. The test error is then approximated by applying the two models to the same test set which has not been used in the creation of either model. The error calculation method must also match. For example, if you are using classification error for the ensemble method you must also use classification error for the linear regression model. The lm predicted number can be used to make a class prediction. If the lm prediction is greater than zero, assign +1 as the class. If the lm prediction is less than zero assign -1 as the class. With this procedure, you are turning lm into a classification tool instead of a regression tool

Answer 4

```
> sonar_full = sonar_hold[sample(length(sonar_hold[,61]),
+                                 length(sonar_hold[,61]),
+                                replace=FALSE),]
> sonar_scaled = as.data.frame(cbind(scale(sonar_full[,-61]), sonar_full[,61]))
> training_set =  sonar_scaled[1:pt_overfit,]
> holdout_set = sonar_scaled[pt_overfit:length(sonar_scaled[,61]),]
> n = 15
> training_set_x = training_set[,-61]
> training_set_y = training_set[,61]
> train_sub_err = rep(0, times =10)
> train=rep(0, times=length(training_set_y))
> pred_train = data.frame(train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train)
> pred_train2 = data.frame(train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train)
> test_sub_err = rep(0, times =10)
> hold=rep(0, times=length(holdout_set[,61]))
> pred_hold = data.frame(hold, hold, hold, hold, hold, hold, hold, hold, hold, hold)
> pred_hold2 = data.frame(hold, hold, hold, hold, hold, hold, hold, hold, hold, hold)
> ptm = proc.time()
> for(i in 1:10){
+
+   set_seed(seed_val)
+   seed_val = seed_val + 1
+   training_sub_set_x = training_set_x[,sample(60, n, replace=FALSE)]
+
+
+   training_sub_set_yx = cbind(training_sub_set_x, training_set_y)
+   lm_sub = lm(training_set_y~., training_sub_set_yx)
+
+
+   yp = (training_set_y >= 0.0)
+   yh = predict(lm_sub, training_sub_set_x )
+   yhp = (yh > 0.0)
+   train_sub_err[i] = sum(yhp != yp)/(length(training_set_y))
+   pred_train[,i]= yh
+   pred_train2[yhp,i]= 1
+   pred_train2[!yhp,i]= -1
+
+   yp = (holdout_set[,61] >= 0.0)
+   yh = predict(lm_sub, holdout_set[,-61])
+   yhp = (yh > 0.0)
+   test_sub_err[i] = sum(yhp != yp)/(length(holdout_set[,61]))
+   pred_hold[,i]= yh
+   pred_hold2[yhp,i]= 1
+   pred_hold2[!yhp,i]= -1
+
+ }
> ptm1 = proc.time() - ptm
> for(i in 1:10){
+   cat(sprintf("%dth %d attributes traning error: %f", i, n, train_sub_err[i]), "\n")
+   cat(sprintf("%dth %d attributes test error: %f", i, n, test_sub_err[i]), "\n")
+ }

1th 15 attributes traning error: 0.258824
1th 15 attributes test error: 0.282258
2th 15 attributes traning error: 0.164706
2th 15 attributes test error: 0.322581
3th 15 attributes traning error: 0.211765
3th 15 attributes test error: 0.258065
4th 15 attributes traning error: 0.164706
4th 15 attributes test error: 0.266129
5th 15 attributes traning error: 0.176471
5th 15 attributes test error: 0.282258
6th 15 attributes traning error: 0.223529
6th 15 attributes test error: 0.258065
7th 15 attributes traning error: 0.282353
7th 15 attributes test error: 0.322581
8th 15 attributes traning error: 0.211765
8th 15 attributes test error: 0.274194
9th 15 attributes traning error: 0.235294
9th 15 attributes test error: 0.233871
10th 15 attributes traning error: 0.200000
10th 15 attributes test error: 0.346774

> library(MASS)
> xlambda=rep(0, times = 30)
> for(i in seq(from = 0, to = 29)){
```

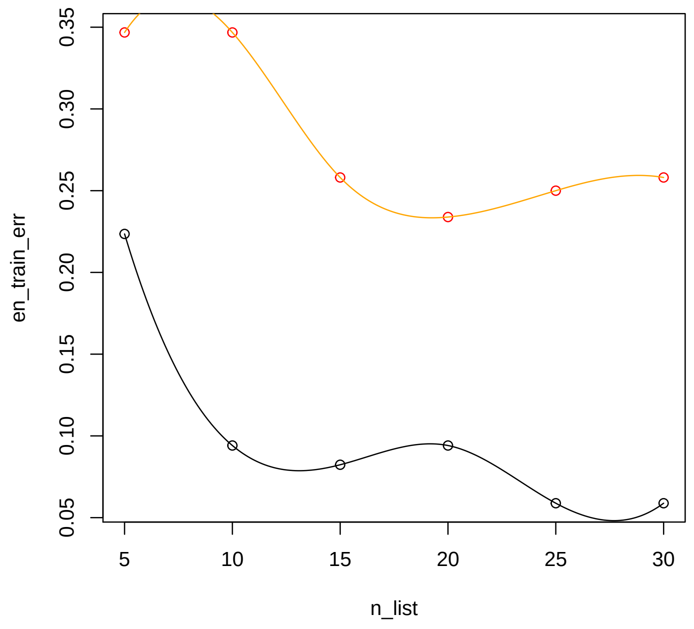
```
+ exp <- (+3 -4*(i/20))
+ xlambda[i+1] <- 10^exp
+ }
> yx_pred_train = cbind(pred_train2, training_set_y)
> holdout_y=holdout_set[,61]
> yx_pred_hold = cbind(pred_hold2, holdout_y)
> en_train_err=rep(0, times=length(xlambda))
> en_hold_err=rep(0, times=length(xlambda))
> for(ilambda in 1:length(xlambda)){
+
+ enmodel = lm.ridge(training_set_y~., yx_pred_train, lambda=xlambda[ilambda])
+ A = as.array(enmodel$coef[1:10]/enmodel$scales)
+
+ X = pred_train2
+ for( i in seq(from = 1, to = ncol(pred_train2))){
+ X[,i] = pred_train2[,i] - enmodel$xm[i]
+   }
+ X = as.matrix(X)
+ yh = X%*%A + enmodel$ym
+
+ yhP = (yh >= 0.0)
+ yp = (training_set_y >= 0.0)
+
+ en_train_err[ilambda] = en_train_err[ilambda] + sum(yhP != yp)/length(training_set_y)
+
+ #holdout set
+ X = pred_hold2
+ for( i in seq(from = 1, to = ncol(pred_hold2))){
+ X[,i] = pred_hold2[,i] - enmodel$xm[i]
+   }
+ X = as.matrix(X)
+ yh = X%*%A + enmodel$ym
+
+ yhP = (yh >= 0.0)
+ yp = (holdout_y >= 0.0)
+
+ en_hold_err[ilambda] = en_hold_err[ilambda] + sum(yhP != yp)/length(holdout_y)
+ }
> th_lambda = min(which(min(en_hold_err)+0.01 > en_hold_err))
> min_en_lambda = xlambda[th_lambda]
> sprintf("Optimal lambda is %f", min_en_lambda)

[1] "Optimal lambda is 1000.000000"

> plot(1:length(xlambda),
+      en_train_err,
+       ylim=c(min(en_train_err, en_hold_err),
+              max(en_train_err, en_hold_err)))
> points(1:length(xlambda),en_hold_err, col='red')
> lines(spline(1:length(xlambda), en_train_err,n=200))
> lines(spline(1:length(xlambda), en_hold_err,n=200), col="orange")
```



```
> seed_val=123
> sonar_full = sonar_hold[sample(length(sonar_hold[,61]),
+                                 length(sonar_hold[,61]),
+                                 replace=FALSE),]
> sonar_scaled = as.data.frame(cbind(scale(sonar_full[,-61]), sonar_full[,61]))
> training_set =  sonar_scaled[1:pt_overfit,]
> holdout_set = sonar_scaled[pt_overfit:length(sonar_scaled[,61]),]
> training_set_x = training_set[,-61]
> training_set_y = training_set[,61]
> train_sub_err = rep(0, times =10)
> train=rep(0, times=length(training_set_y))
> pred_train = data.frame(train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train,
+                         train)
> pred_train2 = data.frame(train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train,
+                          train)
> test_sub_err = rep(0, times =10)
> hold=rep(0, times=length(holdout_set[,61]))
> pred_hold = data.frame(hold, hold, hold, hold, hold, hold, hold, hold, hold, hold)
> pred_hold2 = data.frame(hold, hold, hold, hold, hold, hold, hold, hold, hold, hold)
> n_list = seq(5,30,by=5)
> en_train_err=rep(0, times=length(n_list))
> en_hold_err=rep(0, times=length(n_list))
> id_n=0
> for(n in n_list){
+   id_n = id_n + 1
+
+   ptm2 = proc.time()
+
+   for(i in 1:10){
+     #a) choose n attributes out of 60
+     set_seed(seed_val)
+     seed_val = seed_val + 1
+     training_sub_set_x = training_set_x[,sample(60, n, replace=FALSE)]
+
+     #b) fit a linear model
+     training_sub_set_yx = cbind(training_sub_set_x, training_set_y)
+     lm_sub = lm(training_set_y~., training_sub_set_yx)
+
+     #c) check error
+     yp = (training_set_y >= 0.0)
+     yh = predict(lm_sub, training_sub_set_x )
+     yhp = (yh > 0.0)
+     train_sub_err[i] = sum(yhp != yp)/(length(training_set_y))
+     pred_train[,i]= yh
+     pred_train2[yhp,i]= 1
+     pred_train2[!yhp,i]= -1
```

```
+
+     yp = (holdout_set[,61] >= 0.0)
+     yh = predict(lm_sub, holdout_set[,-61])
+     yhp = (yh > 0.0)
+     test_sub_err[i] = sum(yhp != yp)/(length(holdout_set[,61]))
+     pred_hold[,i]= yh
+     pred_hold2[yhp,i]= 1
+     pred_hold2[!yhp,i]= -1
+
+   }
+
+
+   yx_pred_train = cbind(pred_train2, training_set_y)
+   holdout_y=holdout_set[,61]
+   yx_pred_hold = cbind(pred_hold2, holdout_y)
+
+
+
+   enmodel = lm.ridge(training_set_y~., yx_pred_train, lambda=min_en_lambda)
+   A = as.array(enmodel$coef[1:10]/enmodel$scales)
+
+   X = pred_train2
+   for( i in seq(from = 1, to = ncol(pred_train2))){
+     X[,i] = pred_train2[,i] - enmodel$xm[i]
+   }
+   X = as.matrix(X)
+   yh = X%*%A + enmodel$ym
+
+   yhP = (yh >= 0.0)
+   yp = (training_set_y >= 0.0)
+
+   en_train_err[id_n] = en_train_err[id_n] + sum(yhP != yp)/length(training_set_y)
+
+   #holdout set
+   X = pred_hold2
+   for( i in seq(from = 1, to = ncol(pred_hold2))){
+     X[,i] = pred_hold2[,i] - enmodel$xm[i]
+   }
+   X = as.matrix(X)
+   yh = X%*%A + enmodel$ym
+
+   yhP = (yh >= 0.0)
+   yp = (holdout_y >= 0.0)
+
+   en_hold_err[id_n] = en_hold_err[id_n] + sum(yhP != yp)/length(holdout_y)
+
+   if(id_n == 1)
+     ptm_en = proc.time() - ptm2
+ }
> ptm_en=ptm_en + ptm1
```

```
> plot(n_list,en_train_err,
+      ylim=c(min(en_train_err, en_hold_err),
+             max(en_train_err, en_hold_err)))
> points(n_list,en_hold_err, col='red')
> lines(spline(n_list, en_train_err,n=200))
> lines(spline(n_list, en_hold_err,n=200), col="orange")
```



```
> min_n = n_list[which(en_hold_err==min(en_hold_err))]
> sprintf("When n is %d, the error rate is the lowest.", min_n)
```

```
[1] "When n is 20, the error rate is the lowest."
```

```
> ptm = proc.time()
> lmodel_70row = lm(V61~., training_set)
> yp = (training_set[,61] >= 0.0)
> yh = predict(lmodel_70row, training_set_x)
> yhp = (yh > 0.0)
> error_train_70 = sum(yhp != yp)/(length(training_set[,61]))
> yp = (holdout_y >= 0.0)
> yh = predict(lmodel_70row, holdout_set[,-61])
> yhp = (yh > 0.0)
> error_hold_70 = sum(yhp != yp)/(length(holdout_y))
> ptm_lsm = proc.time() - ptm
> results = data.frame(Training.Error=c(error_train_70, en_train_err[4]),
+                      Test.Error=c(error_hold_70, en_hold_err[4]),
+                      Running.Time=c(ptm_lsm[3], ptm_en[3]))
> row.names(results) <- c("Linear Model(#o=85)", "Ensemble Method(lambda=40, #n=20)")
> print(results)

                                  Training.Error Test.Error Running.Time
Linear Model(#o=85)                   0.00000000  0.3548387         0.01
Ensemble Method(lambda=40, #n=20)     0.09411765  0.2338710         0.11

> print("#o: number of observations (out of 208), #n: number of attributes (out of 60)")

[1] "#o: number of observations (out of 208), #n: number of attributes (out of 60)"
```

Question 5

5) This problem uses two different ensemble methods to classify the sonar data.

5a) Use Random Forest to classify the sonar data.

5b) Use rpart to generate trees with a depth of two on randomly selected attributes. Create at least 100 rpart models. The results of these models applied to the whole data set become new attributes. Use ridge regression to combine these trees to make predictions. The input columns for ridge regression are the attributes that were created from the rpart models. Use cross validation to choose the best lambda and calculate the test error for this lambda. (Use the example BabyEnsemble.R Instead of using lm to create the initial models use rpart. Instead of creating 10 initial models, create at least 100 models)

5c) Which model resulted in the smallest test error?

5d) extra credit Add additional rpart models to 5b). As the number of rpart models increases, does the error rate decrease? Try using a majority count instead of ridge regression to combine the 100 models. Normally the cut off is 50%. What happens when you change this cut off to favor classifying mines instead of rocks?

Answer 5

```
> seed_val=123
> set_seed(seed_val)
> sonar_train = read.csv("sonar_train.csv", header=FALSE)
> sonar_test = read.csv("sonar_test.csv", header=FALSE)
> ptm = proc.time()
> fit_rf = randomForest(sonar_train[,-61], y=as.factor(sonar_train[,61]), ntree=100)
> ptm5 = proc.time() - ptm
> train_err_rf = 1-sum(sonar_train[,61]==predict(fit_rf,sonar_train[,-61])) /
+                 length(sonar_train[,61])
> test_err_rf = 1-sum(sonar_test[,61]==predict(fit_rf,sonar_test[,-61])) /
+                 length(sonar_test[,61])
> sprintf("Training error: %f", train_err_rf)

[1] "Training error: 0.000000"

> ## [1] "Training error: 0.000000"
> sprintf("Test error: %f", test_err_rf)

[1] "Test error: 0.179487"

> ## [1] "Test error: 0.141026"
> require(rpart)
> sonar_train_x = sonar_train[, -61]
> sonar_train_y = as.factor(sonar_train[, 61])
> sonar_test_x = sonar_test[, -61]
> sonar_test_y = sonar_test[, 61]
> a_list = rep(0, times=length(sonar_train_y)*100)
> a_matrix = matrix(a_list,length(sonar_train_y),100)
> pred_dt2 = as.data.frame(a_matrix)
> seed_val=123
> fit_list <- list()
> for(i in 1:100){
+   set_seed(seed_val)
+   seed_val=seed_val+1
+
+   train_subset = cbind(sonar_train_x[, sample(60, 10, replace=FALSE)], sonar_train_y)
+
+
+
+   fit_dt = rpart(sonar_train_y~.,
+                  train_subset,
+                  control = rpart.control(cp = -1,
+                                          maxdepth = 2))
+   fit_list[[i]] = fit_dt
+   pred_dt2[, i] = as.numeric(as.character(predict(fit_dt, train_subset, type="class")))
+
+ }
> V101 = as.numeric(as.character(sonar_train_y))
> pred_dt2_yx= cbind(pred_dt2, V101)
> k=5
> dt2_train = pred_dt2_yx[1:(num_sample*((k-1)/k)),]
> dt2_cross = pred_dt2_yx[1:(num_sample*(1/k)),]
> error_train_total = matrix(0, nrow = length(xlambda), ncol = 1)
> error_cross_total = matrix(0, nrow = length(xlambda), ncol = 1)
> num_sample = nrow(pred_dt2_yx)
> xlambda=rep(0, times = 30)
> for(i in seq(from = 0, to = 29)){
+   exp <- (+5 -4*(i/20))
+   xlambda[i+1] <- 10^exp
+ }
> for(ilambda in 1:length(xlambda)){
+   pick = k
+
+   error_train = 0
+   error_cross = 0
+
+   for(j in 1:k){
+     i_tmp = 1
+     for(i in 1:k){
+
+       if(i == pick){
+         dt2_cross = pred_dt2_yx[((i-1)*num_sample/k+1):(num_sample*(i/k)),]
+       } else {
+         dt2_train[((i_tmp-1)*num_sample/k+1):(num_sample*(i_tmp/k)), ] =
+           pred_dt2_yx[((i-1)*num_sample/k+1):(num_sample*i/k),]
+         i_tmp = i_tmp + 1
+       }
+     }
+
+
+
+
+     pick = pick - 1
+     row.names(dt2_cross)<-NULL
+     row.names(dt2_train)<-NULL
+     y_dt2_train = dt2_train[,101]
+     x_dt2_train = dt2_train[,-101]
+     yx_dt2_train = cbind(x_dt2_train, y_dt2_train)
+
+
+
+
+     y_dt2_cross = dt2_cross[,101]
+     x_dt2_cross = dt2_cross[,-101]
+
+     dt2_model = lm.ridge(y_dt2_train~., yx_dt2_train, lambda=xlambda[ilambda])
+     A = as.array(dt2_model$coef[1:100]/dt2_model$scales)
+     X_train = x_dt2_train
+     for( i in seq(from = 1, to = ncol(x_dt2_train))){
+       X_train[,i] = x_dt2_train[,i] - dt2_model$xm[i]
+     }
+     X_train=as.matrix(X_train)
+     yh = X_train%*%A + dt2_model$ym
+
+     yhP = (yh >= 0.0)
+     yp = (y_dt2_train >= 0.0)
+
+
+
+     error_train = error_train + sum(yhP != yp)/(length(y_dt2_train)*k*0.00001/0.00001)
+
+     X_cross = x_dt2_cross
+     for( i in seq(from = 1, to = ncol(x_dt2_cross))){
+       X_cross[,i] = x_dt2_cross[,i] - dt2_model$xm[i]
+     }
+     X_cross=as.matrix(X_cross)
+     yh = X_cross%*%A + dt2_model$ym
+
+
+     yhP = (yh >= 0.0)
+     yp = (y_dt2_cross >= 0.0)
+
+     error_cross = error_cross + sum(yhP != yp)/(length(y_dt2_cross)*k*0.00001/0.00001)
+
+   }
+
+
+
+   error_train_total[ilambda] = error_train
+   error_cross_total[ilambda] = error_cross
+
+ }
> th_lambda = min(which(min(error_cross_total) == error_cross_total))
> min_dt2_lambda = xlambda[th_lambda]
> sprintf("Optimal lambda is %f.", min_dt2_lambda)

[1] "Optimal lambda is 398.107171."

> plot(1:length(xlambda),error_train_total,
+      ylim=c(min(error_train_total, error_cross_total),
+             max(error_train_total, error_cross_total)))
> points(1:length(xlambda),error_cross_total, col='red')
> lines(spline(1:length(xlambda), error_train_total,n=200))
> lines(spline(1:length(xlambda), error_cross_total,n=200), col="orange")
> a_list = rep(0, times=length(sonar_test_y)*100)
> a_matrix = matrix(a_list,length(sonar_test_y),100)
> test_dt2 = as.data.frame(a_matrix)
> ptm = proc.time()
> for(i in 1:100){
+   fit_dt = fit_list[[i]]
+   test_dt2[, i] = as.numeric(as.character(predict(fit_dt, sonar_test_x, type="class")))
+ }
> dt2_min_model = lm.ridge(V101~., pred_dt2_yx, lambda=min_dt2_lambda)
> A = as.array(dt2_min_model$coef[1:100]/dt2_min_model$scales)
> X = pred_dt2_yx[,-101]
> for( i in seq(from = 1, to = (ncol(pred_dt2_yx)-1) ) ){
+   X[,i] = pred_dt2_yx[,i] - dt2_min_model$xm[i]
+ }
> X=as.matrix(X)
> yh = X%*%A + dt2_min_model$ym
> yhP = (yh >= 0.0)
> yp = (pred_dt2_yx[,101] >= 0.0)
> error_dt2_train = sum(yhP != yp)/(length(pred_dt2_yx[,101])*0.00001/0.00001)
> X = test_dt2
> for( i in seq(from = 1, to = (ncol(test_dt2)) ) ){
+   X[,i] = test_dt2[,i] - dt2_min_model$xm[i]
+ }
> X=as.matrix(X)
> yh = X%*%A + dt2_min_model$ym
> yhP = (yh >= 0.0)
> yp = (sonar_test_y >= 0.0)
> error_dt2_test = sum(yhP != yp)/(length(sonar_test_y)*0.00001/0.00001)
> ptm10 = proc.time()-ptm
> sprintf("Training error: %f", train_err_rf)

[1] "Training error: 0.000000"

> sprintf("Test error: %f", test_err_rf)

[1] "Test error: 0.179487"

> sprintf("Training error: %f", error_dt2_train)

[1] "Training error: 0.015385"

> sprintf("Test error: %f", error_dt2_test)

[1] "Test error: 0.192308"

> results = data.frame(Training.Error=c(train_err_rf, error_dt2_train),
+                      Test.Error=c(test_err_rf, error_dt2_test),
+                      Running.Time=c(ptm5[3], ptm10[3]))
> row.names(results) <- c("Random Forest", "Decision Tree bagging")
> print(results)
```

```
                     Training.Error Test.Error Running.Time
Random Forest            0.00000000  0.1794872         0.05
Decision Tree bagging    0.01538462  0.1923077         0.16
```

```
> print("Random forest has n_tree=100, decision tree bagging has depth=2, ntree=100")
```

```
[1] "Random forest has n_tree=100, decision tree bagging has depth=2, ntree=100"
```