CSC 529: Advance Data Mining - Assignment 2

Akhil Kumar Ramasagaram

Friday, Feb 8, 2016

Problem 1

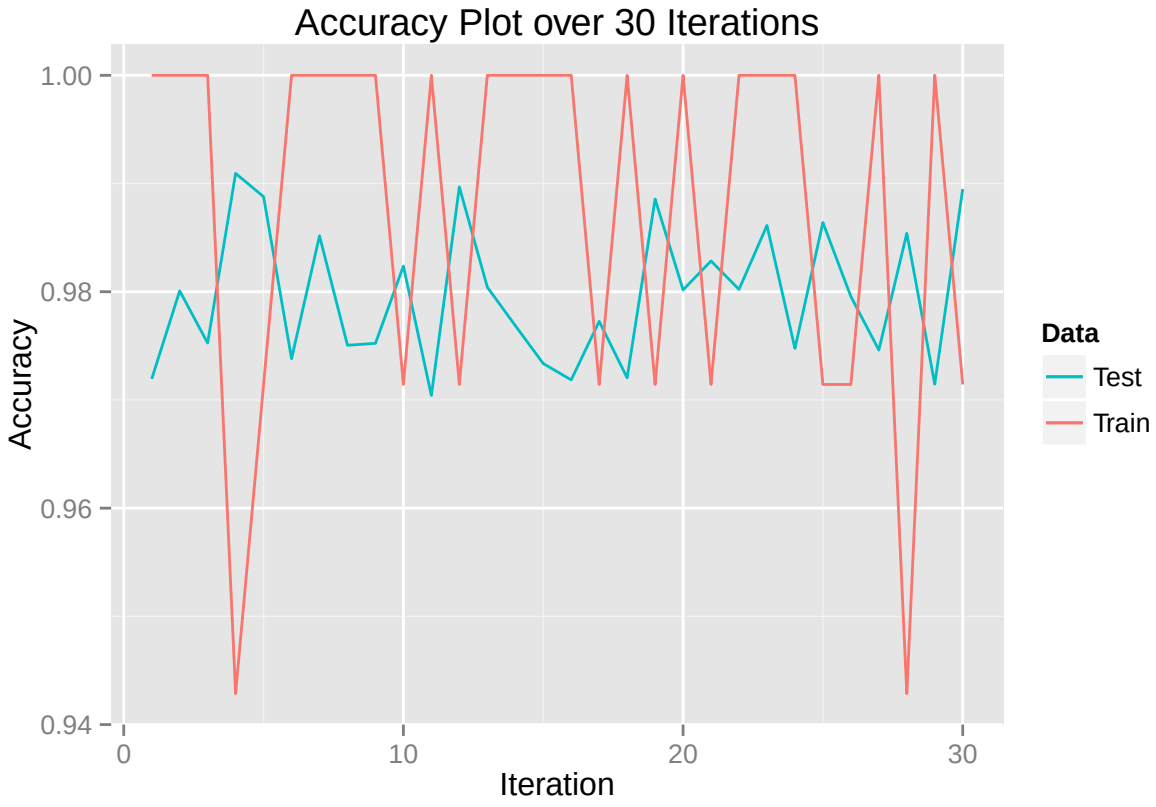
One of the shortcomings of the results in Problem 2 from Assignment#1 is due to the fact of considering a single sample for training set as well as for test set (a single trial of 66% for training and 34% for testing was used to build the model). The objective of this exercise is to repeat the same experiment, but now with different (same size) samples as training and test sets (in other words, repeating the holdout procedure).

a. Repeat Problem 2 A & B from Assignment#1 on the Wine Recognition Dataset at least 30 times and report the means, variances, and Confidence Intervals (CI) for the accuracy results on the training and testing sets.

```
setwd("C:/Users/Akhilkumar/Desktop/depaul/CSC 529/")
wine_data <- read.table("wine.data.txt", sep = ",")
names(wine_data) <- c("Target","Alcohol","Malic.Acid","Ash","Ash.Alalinity","Magnesium",
                      "Total.Phenols","Flavanoids", "Nonflavanoid.Phenols","Proanthocyanins",
                      "Color.Intensity","Hue","Od.Water","Proline")
wine_data$Target <- as.factor(wine_data$Target)
library(caret)

build_model <- function(method){
  df <- data.frame("Iter"=NULL, "train_accuracy" = NULL, "test_accuracy" = NULL,
                   "tr.LI" = NULL, "tr.UL" = NULL, "te.LI" = NULL, "te.UL" = NULL)

  for(i in 1:30){
    trainIndex <- createDataPartition(wine_data$Alcohol, p = .8,
                                      list = FALSE,
                                      times = 1)
    wine_train <- wine_data[trainIndex,]
    wine_test <- wine_data[-trainIndex,]
    if(method == 'randomforest'){
      model <- train(Target ~ ., data = wine_train, method = "rf",
                     tuneGrid = expand.grid(mtry = c(2,3,4)))
      tr_pred <- predict(model$finalModel, newdata = wine_train)
      te_pred <- predict(model$finalModel, newdata = wine_test)
    }
    else{
      model <- train(wine_train[,-1], wine_train[,1], method = "nb",
                     trControl = trainControl(method="cv", 3))
      tr_pred <- predict(model$finalModel, newdata = wine_train)$class
      te_pred <- predict(model$finalModel, newdata = wine_test)$class
    }
    df <- rbind(df,
          data.frame("Iter"=i,
                     "train_accuracy" = max(model$results$Accuracy),
                     "test_accuracy" = as.numeric(confusionMatrix(te_pred, wine_test$Target)$overall[1]),
                     "tr.LI" = as.numeric(confusionMatrix(tr_pred, wine_train$Target)$overall[3]),
```

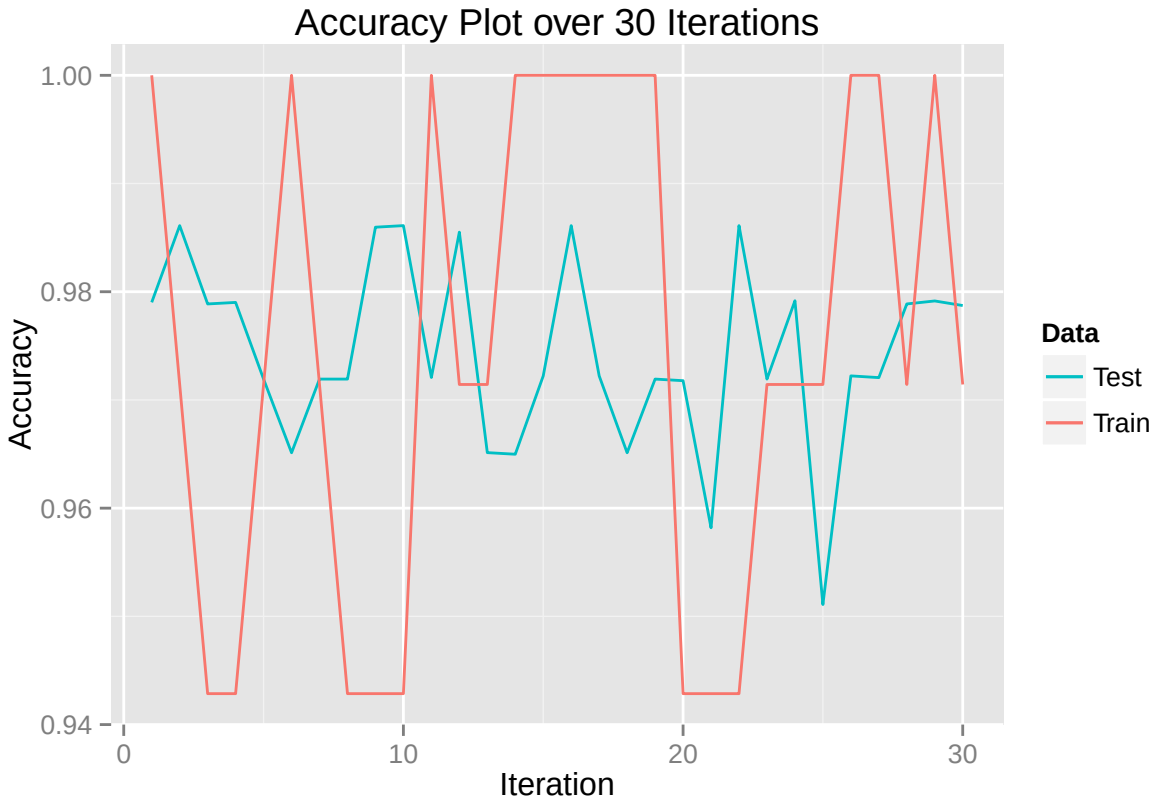
```
                    "tr.UI" = as.numeric(confusionMatrix(tr_pred, wine_train$Target)$overall[4]),
                    "te.LI" = as.numeric(confusionMatrix(te_pred, wine_test$Target)$overall[3]),
                    "te.UI" = as.numeric(confusionMatrix(te_pred, wine_test$Target)$overall[4])))
  }
  print("Mean")
  print(round(apply(df[,-1], 2, mean),3))
  print("Variance")
  print(round(apply(df[,-1], 2, var),5))
  p <- ggplot(df, aes(x = Iter, y = train_accuracy, color = "red")) + geom_line() +
    geom_line(aes(x = Iter, y = test_accuracy, color = "blue")) +
    scale_colour_discrete(name  ="Data", breaks=c("red", "blue"), labels=c("Test", "Train")) +
    xlab("Iteration") + ylab("Accuracy") +
    ggtitle("Accuracy Plot over 30 Iterations")
  print(p)
  return(df$test_accuracy)
}
rf_accuracy <-build_model('randomforest')
```

```
## [1] "Mean"
## train_accuracy  test_accuracy          tr.LI          tr.UI          te.LI
##          0.980          0.988          0.975          1.000          0.879
##          te.UI
##          0.999
## [1] "Variance"
## train_accuracy  test_accuracy          tr.LI          tr.UI          te.LI
##        0.00004        0.00032        0.00000        0.00000        0.00088
##          te.UI
##        0.00000
```




```
nb_accuracy <- build_model('naivebayes')
```

```
## [1] "Mean"
## train_accuracy  test_accuracy          tr.LI          tr.UI          te.LI
##          0.974          0.975          0.951          0.998          0.859
##          te.UI
##          0.998
## [1] "Variance"
## train_accuracy  test_accuracy          tr.LI          tr.UI          te.LI
##        0.00007        0.00055        0.00014        0.00001        0.00142
##          te.UI
##        0.00001
```



b. Using a pair t-test, compare the mean accuracy of the Naïve Bayes and the mean accuracy of the Decision tree and discuss the results.

```
t.test(rf_accuracy, nb_accuracy, paired = T)
```

```
##
##  Paired t-test
##
## data:  rf_accuracy and nb_accuracy
## t = 2.8105, df = 29, p-value = 0.008771
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.004408681 0.027972271
## sample estimates:
## mean of the differences
##              0.01619048
```

With p-value at 0.05 we reject null hypothesis and accept alternate hypothesis that the true difference in means is not equal to 0. Here the p-value is 0.02088 and the observed difference in mean of predictions from random forest and naive bayes is 0.012.

Problem 2

The objective of this exercise is to analyze the performance of the previously trained Naïve Bayes and decision trees classifiers (using the approach in Problem 1) as a function of the sample size. Let us vary the training

data (and the corresponding values of the class attribute) such that the size of the training data is 25%, 35%, 45%, 55%, 65%, 75%, and 85% of the original data. Create a decision tree and a Naive Bayes classifier for each one of the seven different data splits, record the resubstitution error and the generalization error, and present the results in a plot with 4 curves (each corresponding to one type of error and one type of classifier) as a function of the size of the data. Perform an analysis of the results obtained.

```
compare_model <- function(data){
  df <- data.frame("Iter"=NULL, "Error" = NULL, "Type" = NULL)
  for(i in c(.25,.35,.45,.55,.65,.75,.85)){
    trainIndex <- createDataPartition(data[,names(data) == "Target"], p = i,
                                      list = FALSE,
                                      times = 1)
    train <- data[trainIndex,]
    test <- data[-trainIndex,]
    ind <- which(names(data) == "Target")
    rf_model <- train(Target ~ ., data = train, method = "rf", tuneGrid = expand.grid(mtry = c(2,3,4)))
    rf_tr_pred <- predict(rf_model$finalModel, newdata = train)
    rf_te_pred <- predict(rf_model$finalModel, newdata = test)

    nb_model <- train(train[,-ind], train[,ind], method = "nb", trControl = trainControl(method="cv", 3
    nb_tr_pred <- predict(nb_model$finalModel, newdata = train)$class
    nb_te_pred <- predict(nb_model$finalModel, newdata = test)$class

    df <- rbind(df, data.frame("Split"=i, "Error" = 1 - max(rf_model$results$Accuracy),
                               "Error_Type" = "RF_RE"),
                    data.frame("Split"=i,
                               "Error" = 1 - as.numeric(confusionMatrix(rf_te_pred, test$Target)$overall
                               "Error_Type" = "RF_GE"),
                    data.frame("Split"=i, "Error" =  1 - max(nb_model$results$Accuracy),
                               "Error_Type" = "NB_RE"),
                    data.frame("Split"=i,
                               "Error" = 1 - as.numeric(confusionMatrix(nb_te_pred, test$Target)$overall
                               "Error_Type" = "NB_GE"))

  }

  p <- ggplot(df,aes(x = Split, y = Error, color = Error_Type)) + geom_line() +
    xlab("Split") + ylab("Error") +
    ggtitle("Error Plot")
  print(p)

}
```

```
wine_red <- read.csv("winequality-red.csv", sep = ";")
wine_red$quality <- as.factor(wine_red$quality)
names(wine_red)[ncol(wine_red)] <- "Target"
compare_model(wine_red)
```


```
banknote <- read.csv("data_banknote_authentication.txt", sep = ",", header = F)
names(banknote) <- c("variance","skewness","curtosis","entropy","Target")
banknote$Target <- as.factor(banknote$Target)
```

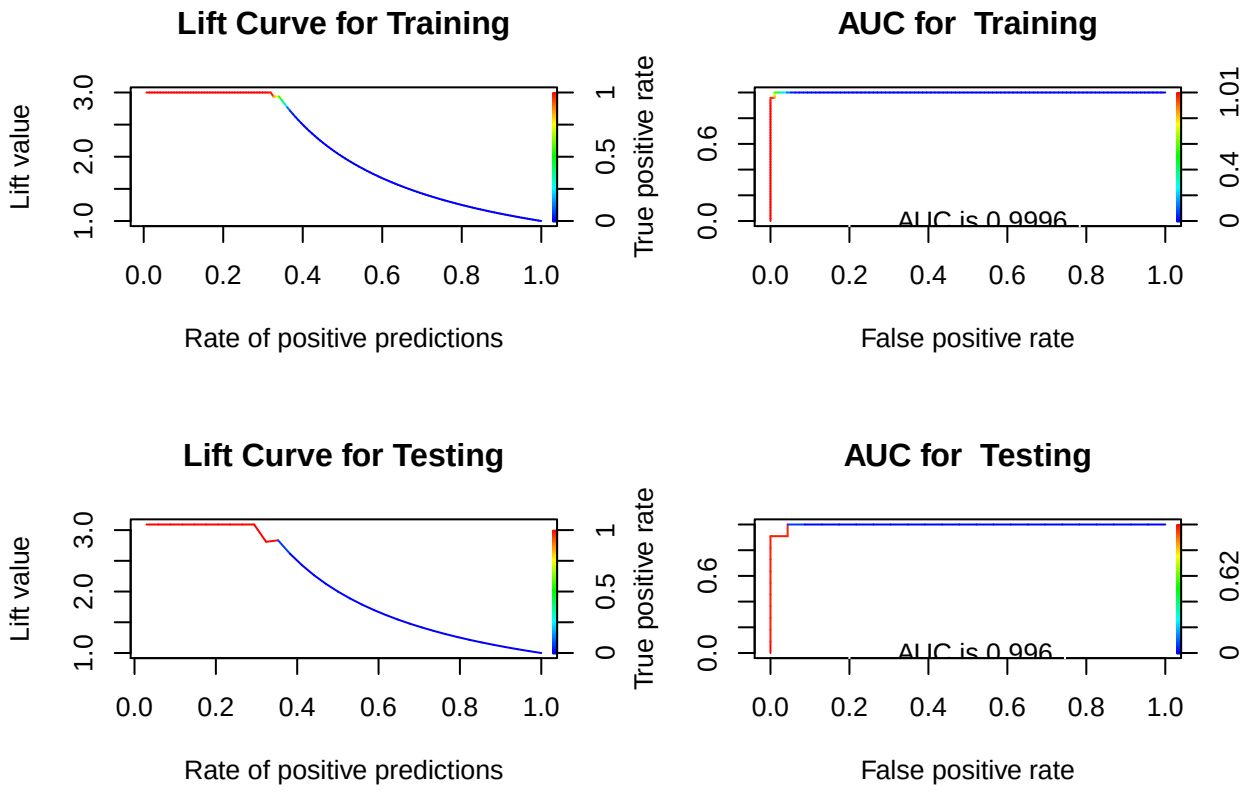
Problem 3

The objective of this exercise is to get familiar with the evaluation of probabilistic classifiers using ROC and Lift curves.

a. Repeat Problem 2.b from Assignment#1 on the Wine Recognition Dataset but this time considering only two classes.

```
library(ROCR)
t <- ifelse(wine_data$Target == "1",0,1)
new_wine <- wine_data
new_wine$Target <- as.factor(t)

trainIndex <- createDataPartition(new_wine$Target, p = .8,
                                  list = FALSE,
                                  times = 1)
train <- new_wine[trainIndex,]
test <- new_wine[-trainIndex,]

nb_model <- nb_model <-  NaiveBayes(Target~., data=train)
```

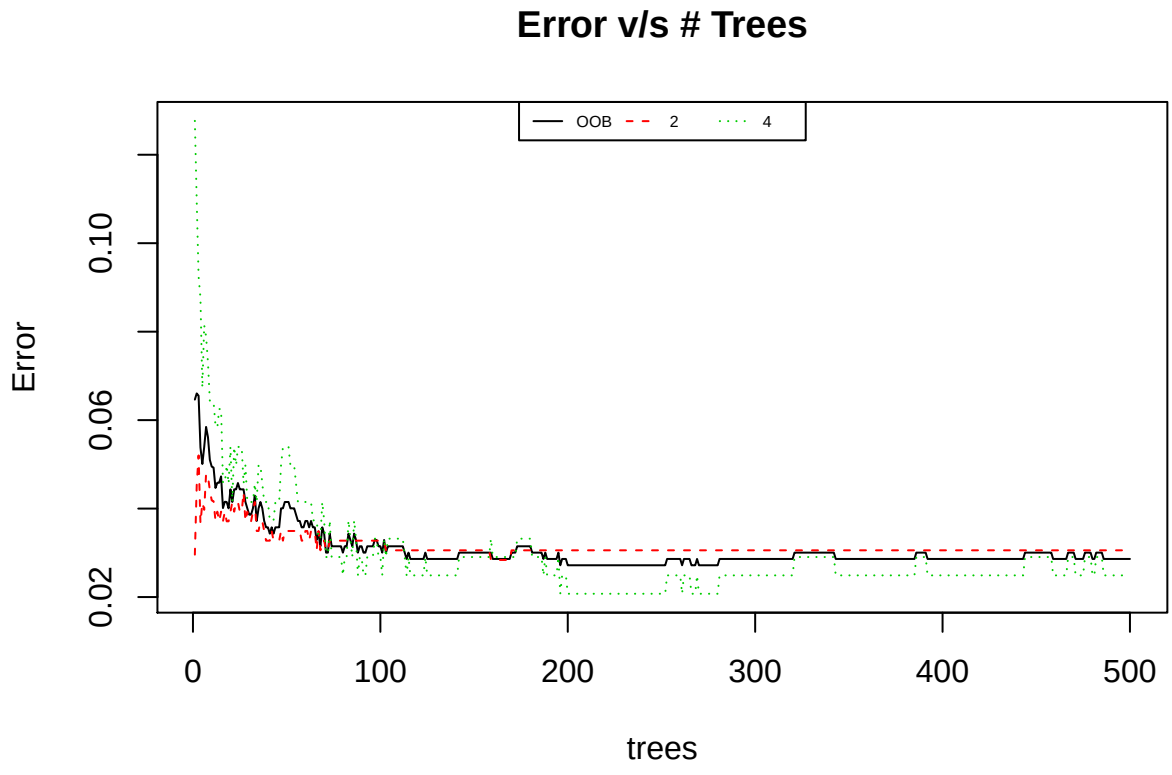
```
par(mfrow = c(2,2))
for(i in c("Training","Testing")){
  if(i == "Training"){
    nb_pred <- predict(nb_model, newdata = train[,-1], type = "raw")
    actual <- train$Target == '0'
  }
  else{
    nb_pred <- predict(nb_model, newdata = test[,-1], type = "raw")
    actual <- test$Target == '0'
  }
  score <- nb_pred$posterior[,'0']
  nb_p <- prediction(score,actual)
  nbperf <- performance(nb_p, "tpr", "fpr")
  lperf <- performance(nb_p,"lift","rpp")
  plot(lperf, colorize=T, main = paste("Lift Curve for", i))
  nbauc <- performance(nb_p, "auc")
  nbauc <- unlist(slot(nbauc, "y.values"))
  plot(nbperf, colorize=TRUE, main = paste("AUC for ",i))
  legend(0.2,0.4,c(c(paste('AUC is', round(nbauc,4))),"\n"),
         border="white",cex=1.0, box.col = "white")
}
```



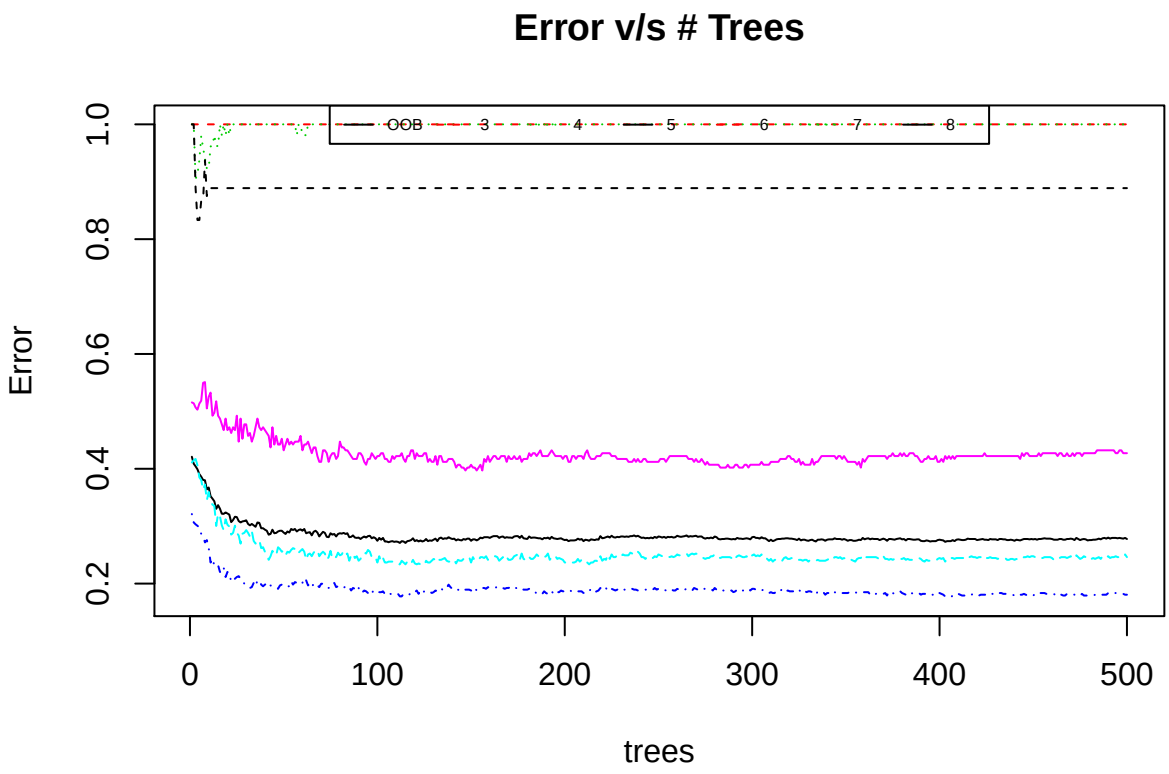
Problem 4

The objective of this exercise is to get familiar with ensemble of classifiers and understand how the number of classifiers/learners affect the accuracy of the classifier. The breast cancer Wisconsin dataset consists of 569 instances and 32 features. The class variable represents diagnosis (M=malignant, B=benign) and the features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

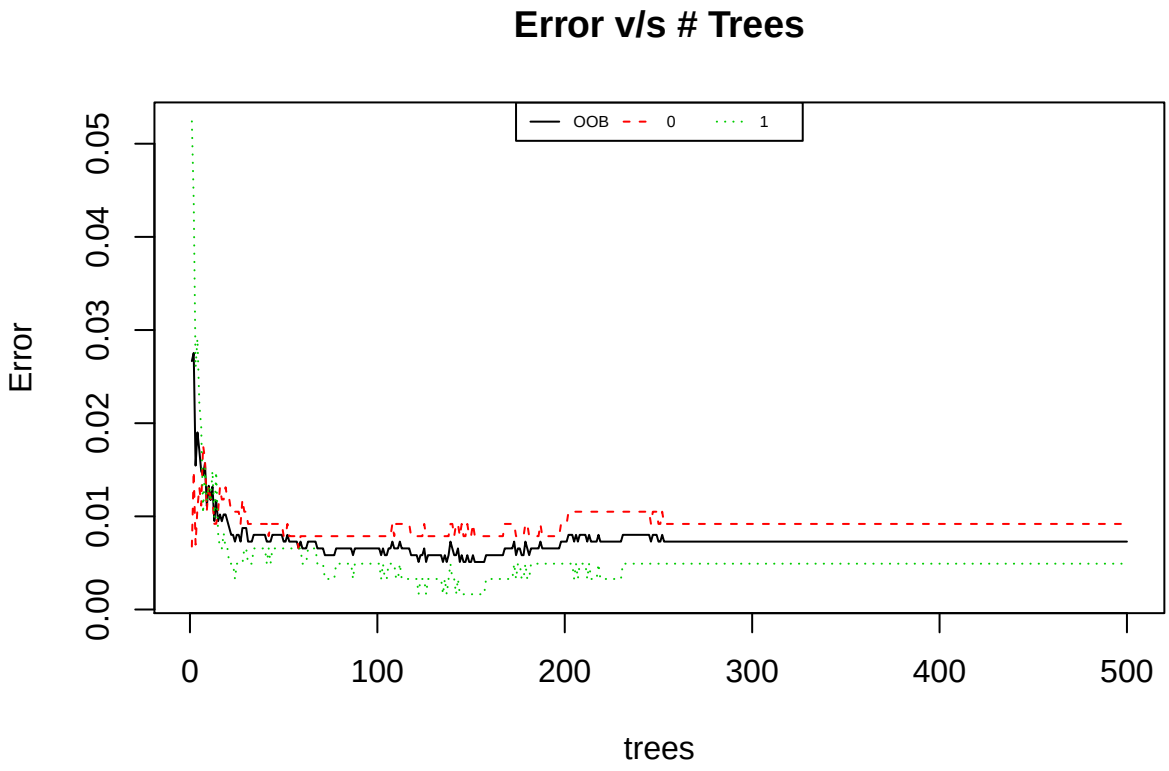
```
en_decison_model <- function(data){
  rndF1 <- randomForest(Target ~ ., data = data, ntree = 500)
  plot(rndF1, main = "Error v/s # Trees")
  rndF1.legend <- if (is.null(rndF1$test$err.rate)) {
    colnames(rndF1$err.rate)
  }
  else {
    colnames(rndF1$test$err.rate)
  }
  legend("top", cex =0.5, legend=rndF1.legend, lty=c(1,2,3), col=c(1,2,3), horiz=T)
}

en_decison_model(med_data)
```




```
en_decison_model(wine_red)
```




```
en_decison_model(banknote)
```



Our model has good performance and since we have multiple predictors, In this case bagging is a good choice.

The difference between bagging and boosting is that in boosting we calculate the output using several different models and then average the result using a weighted average approach. By combining the advantages and pitfalls of these approaches by varying your weighting formula you can come up with a good predictive force for a wider range of input data, using different narrowly tuned models. Bagginh which is also known as boostrap aggregation is the way decrease the variance of your prediction by generating additional data for training from your original dataset using combinations with repetitions to produce multisets of the same cardinality/size as your original data. By increasing the size of your training set you can't improve the model predictive force, but just decrease the variance, narrowly tuning the prediction to expected outcome.