Tree Methods

Ignasi Mañé, Antoni Company & Chiara Barbi

8 de Abril de 2019

```
library(caret)
library(rpart.plot)
library(randomForest)
library(pROC)
library(ROC632)
library(gbm)
library(ISLR)
library(ada)
```

1. Do a short exploratory data analysis in order to know some characteristics of each variable

```
data <- as.data.frame(read.csv("soldat.csv",header = T,sep = ","))
data$y<-ifelse(data$y==-1,0,1)
data$y<-as.factor(data$y)
```

Percentage missing values

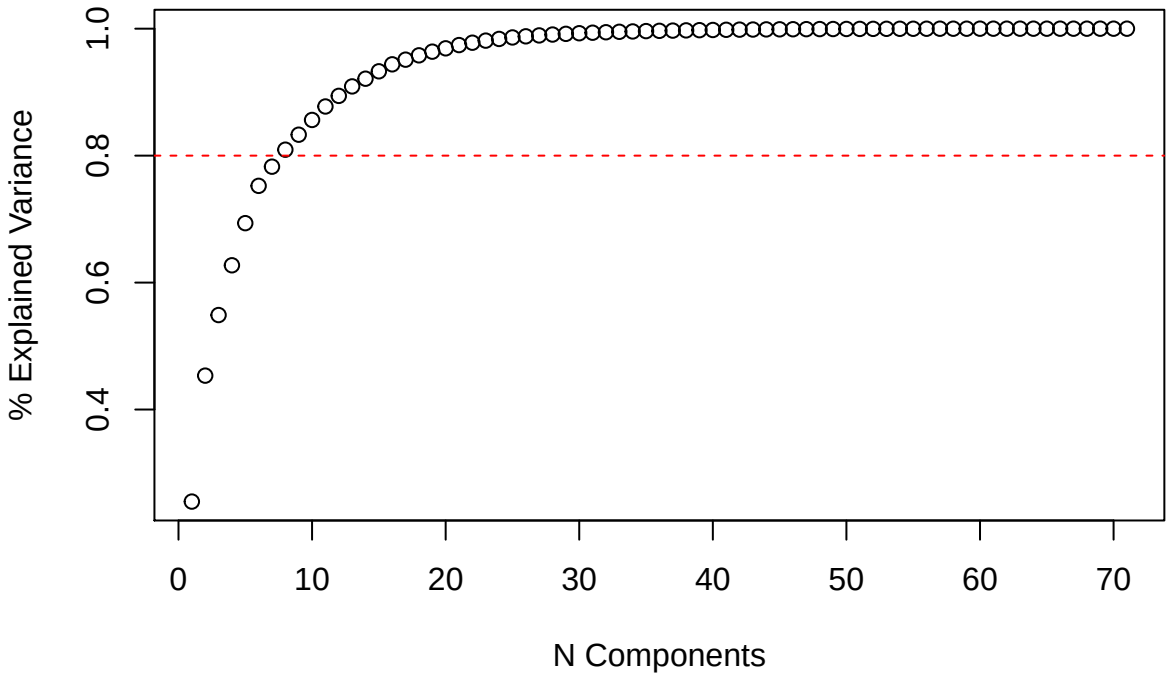
```
p.m <- sum(is.na(data))/(dim.data.frame(data)[1]*dim.data.frame(data)[2])*100
paste(round(p.m,2),"% missing values")
```

```
## [1] "0.19 % missing values"
```

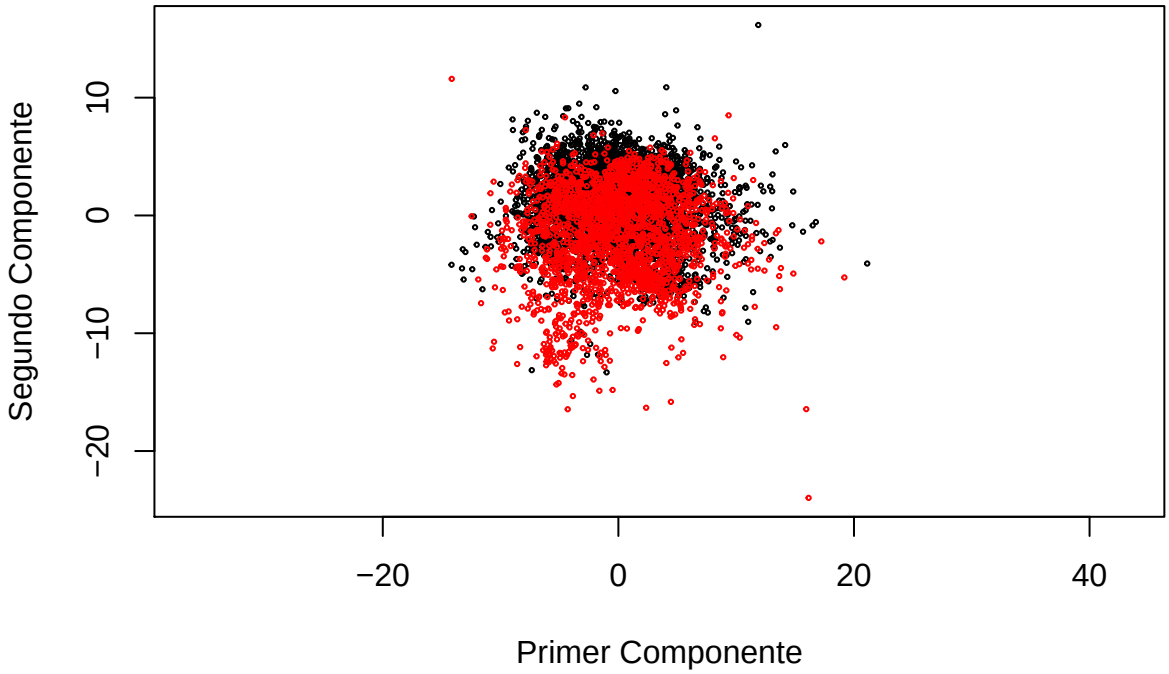
```
data <- data[,-71]
```

We do a PCA analysisis to see if we can explain the variance of the original matrix using an approximate lower dimensional matrix.

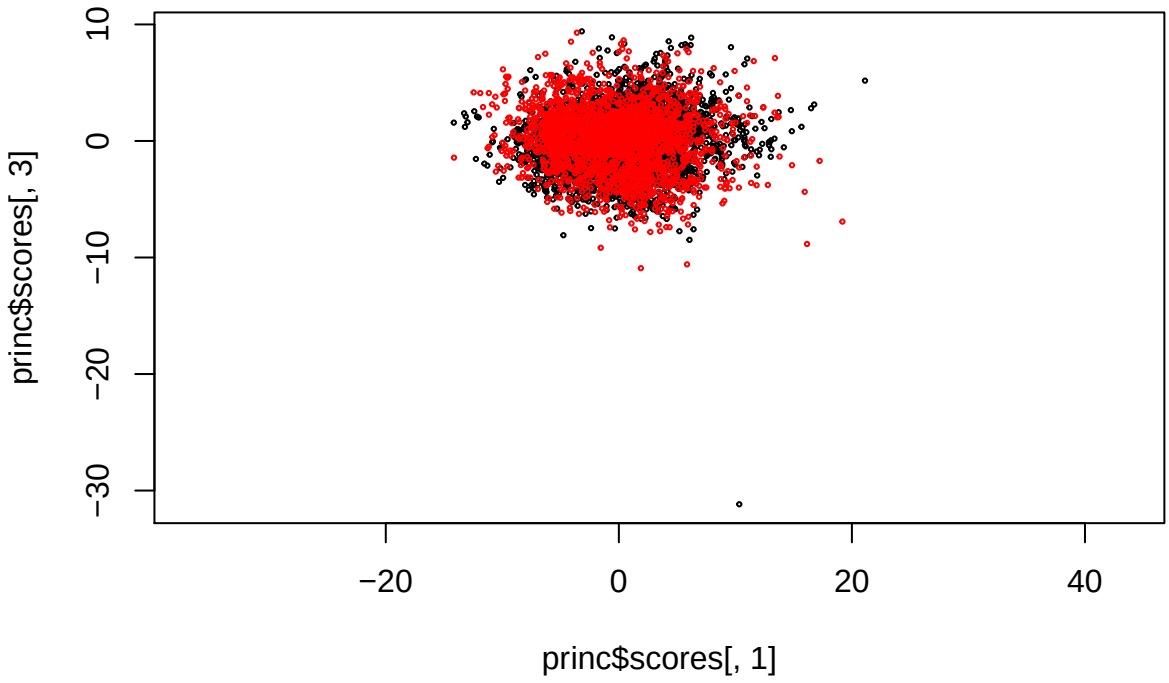
```
data.pca <- scale(data[,-72], scale = TRUE)
princ <- princomp( as.matrix(data.pca))

plot(cumsum(princ$sd^2)/sum(princ$sd^2), xlab = "N Components" , ylab = "% Explained Var
abline(h=0.8, lty=2, col="red")
```



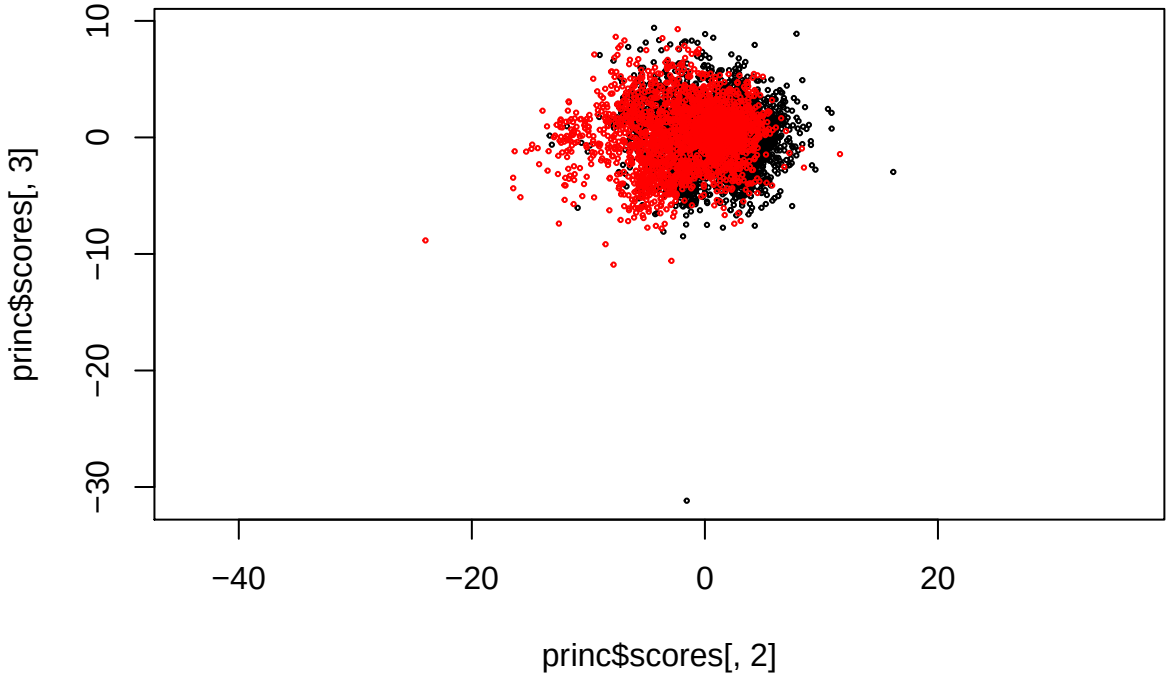

```
plot(princ$scores[,1],princ$scores[,2], cex=0.3,
     col=data$y, asp=1, xlab = "Primer Componente", ylab = "Segundo Componente")
```



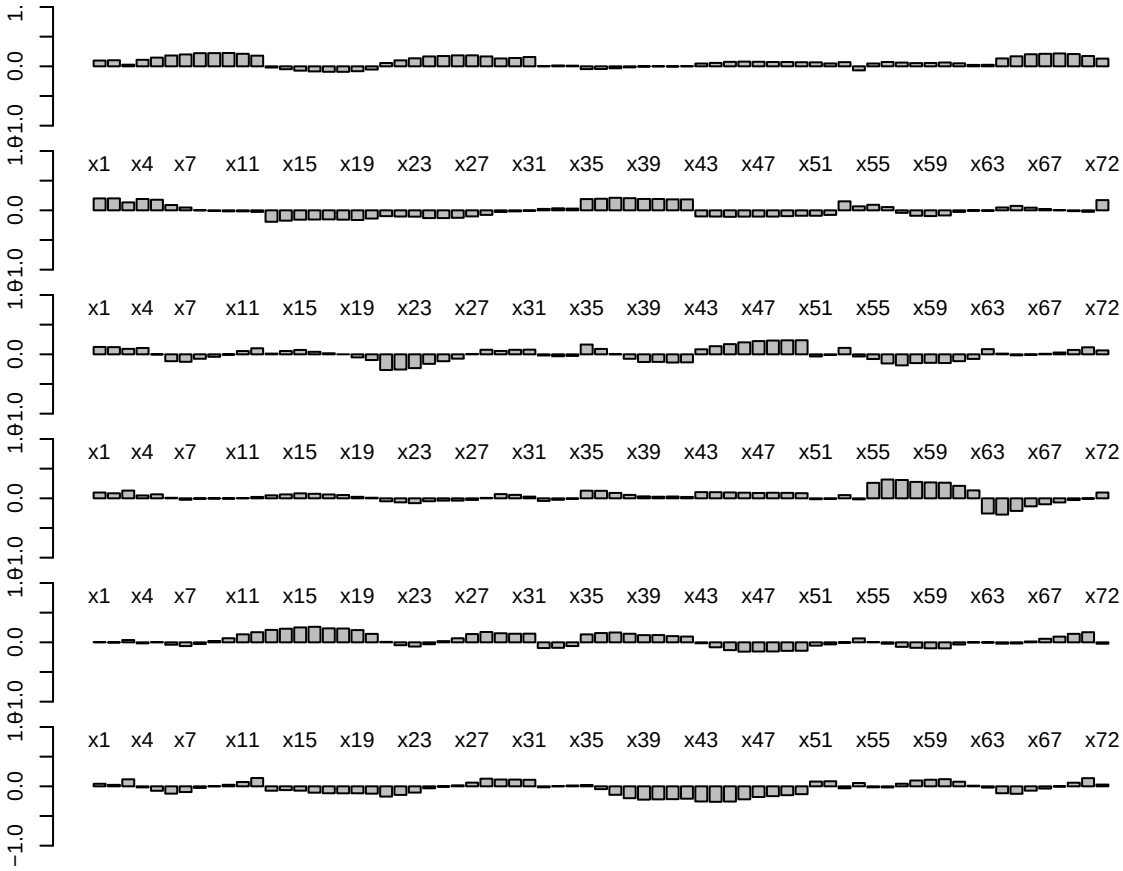

```
plot(princ$scores[,1],princ$scores[,3], cex=0.3, col=data$y, asp=1)
```




```
plot(princ$scores[,2],princ$scores[,3], cex=0.3, col=data$y, asp=1)
```




```
load.soldat <- princ$loadings


par(mar = c(1,4,0,2), mfrow = c(6,1))
load=load.soldat[abs(load.soldat)>1e-2]
for(i in 1:6) barplot(load.soldat[,i], ylim = c(-1, 1))
```



2. Separate the data into 2 balanced partitions: a training set (2,815 compounds) and a test set (2,816 compounds). Use this same partition in the training phase (and validation phase if necessary) and the test phase of each of the sections that are presented below. Use the value 1234 as random seed to do the partition.

```
set.seed(1234)
pt <- 1/2
index <- createDataPartition(y = data$y ,p = pt,list = FALSE) #data partition
data_train <- data[-index,] #train set
data_test <- data[index,] #test set
```

3. Fit a pruned single tree classifier to predict the aqueous solubility. Assess the performance of the tree by using suitable metrics.

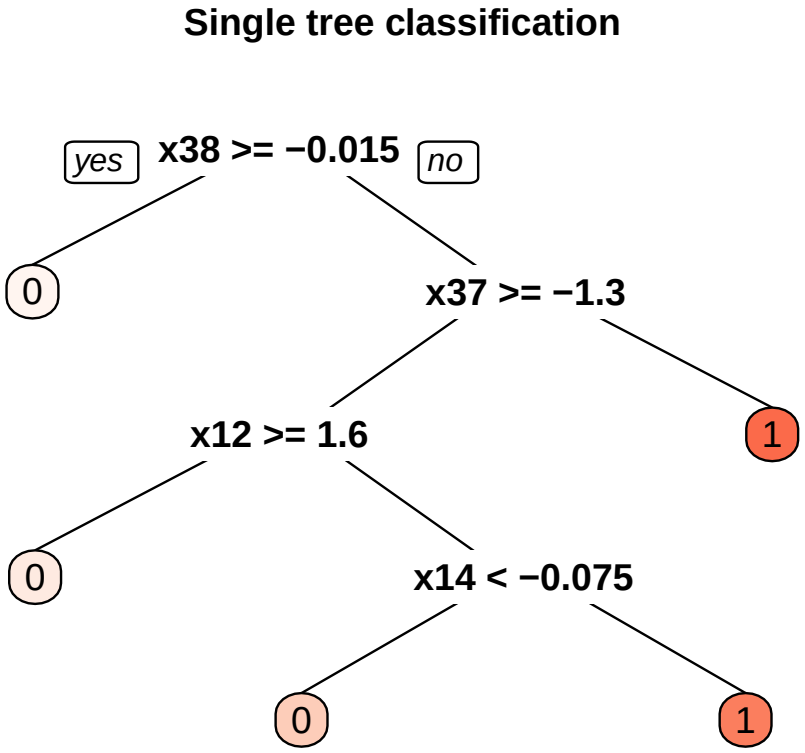
```
ctrl <- trainControl(method = "repeatedcv",number = 10, repeats=3)
single.tree <- train(y ~ .,
```

```
                      data = data_train,
                      method="rpart",
                      trControl=ctrl,
                      metric="Accuracy",
                      preProc= c("center","scale"))

single.tree
```

```
## CART
##
## 2815 samples
##   71 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (71), scaled (71)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 2533, 2533, 2533, 2534, 2533, 2534, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03741815  0.6826436  0.3032506
##   0.03991269  0.6767326  0.2982490
##   0.14031805  0.6393231  0.1747890
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03741815.
```

And it's graphical representation is:

```
prp(single.tree$finalModel,box.palette = "Reds",tweak = 1.2, main = "Single tree classif
```



Predictions and performance analysis:

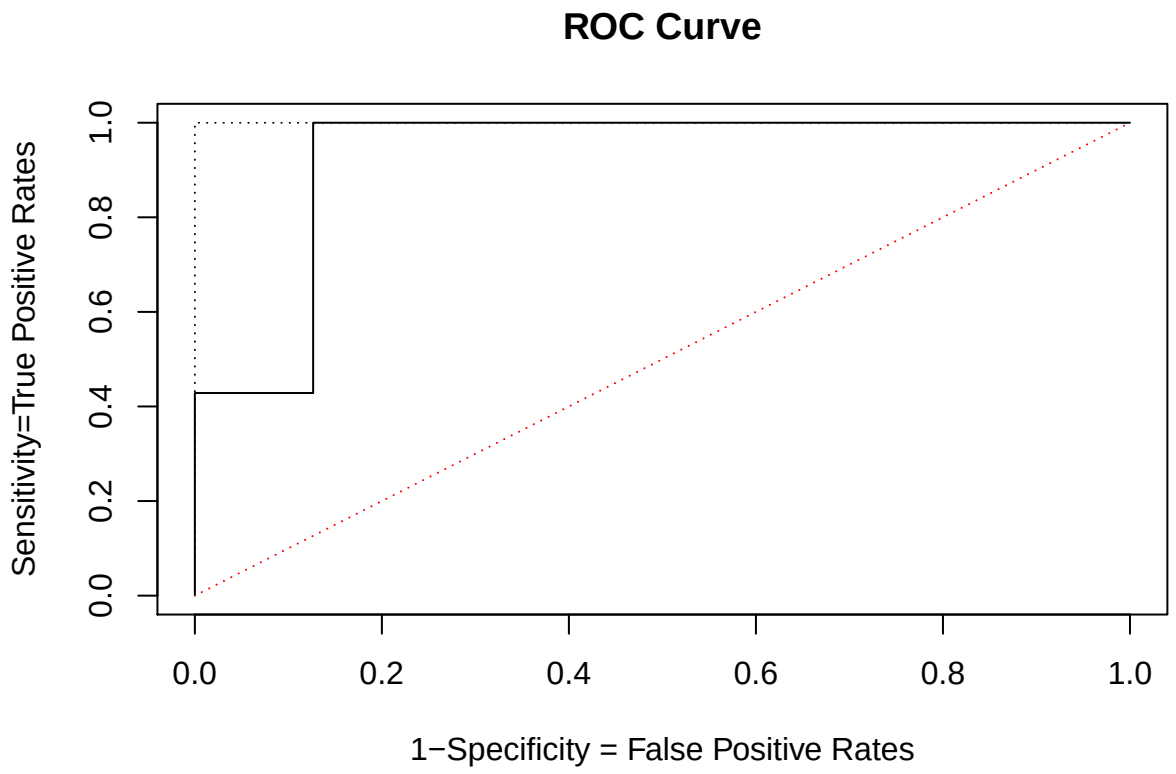
```
pred <- predict(single.tree, data_test)

confusionMatrix(pred,data_test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1526  611
##          1  221  458
##
##                Accuracy : 0.7045
##                  95% CI : (0.6873, 0.7214)
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3249
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8735
##             Specificity : 0.4284
##          Pos Pred Value : 0.7141
##          Neg Pred Value : 0.6745
```

```
##              Prevalence : 0.6204
##          Detection Rate : 0.5419
##    Detection Prevalence : 0.7589
##       Balanced Accuracy : 0.6510
##
##        'Positive' Class : 0
##
```

We can also see the performance of the random forest with the ROC analysis:

```
single.tree.roc <- roc(as.numeric(data_test$y),as.numeric(pred))

plot(x=1-single.tree.roc$specificities, y=single.tree.roc$sensitivities, ylab="Sensitivi
clip(x1=0,x2=1,y1=0,y2=1)
abline(a=0, b=1,col=2,lty=3)
abline(v=0,col=1,lty=3)
abline(h=1,col=1,lty=3)
```




```
single.tree.roc$auc
```

```
## Area under the curve: 0.651
```

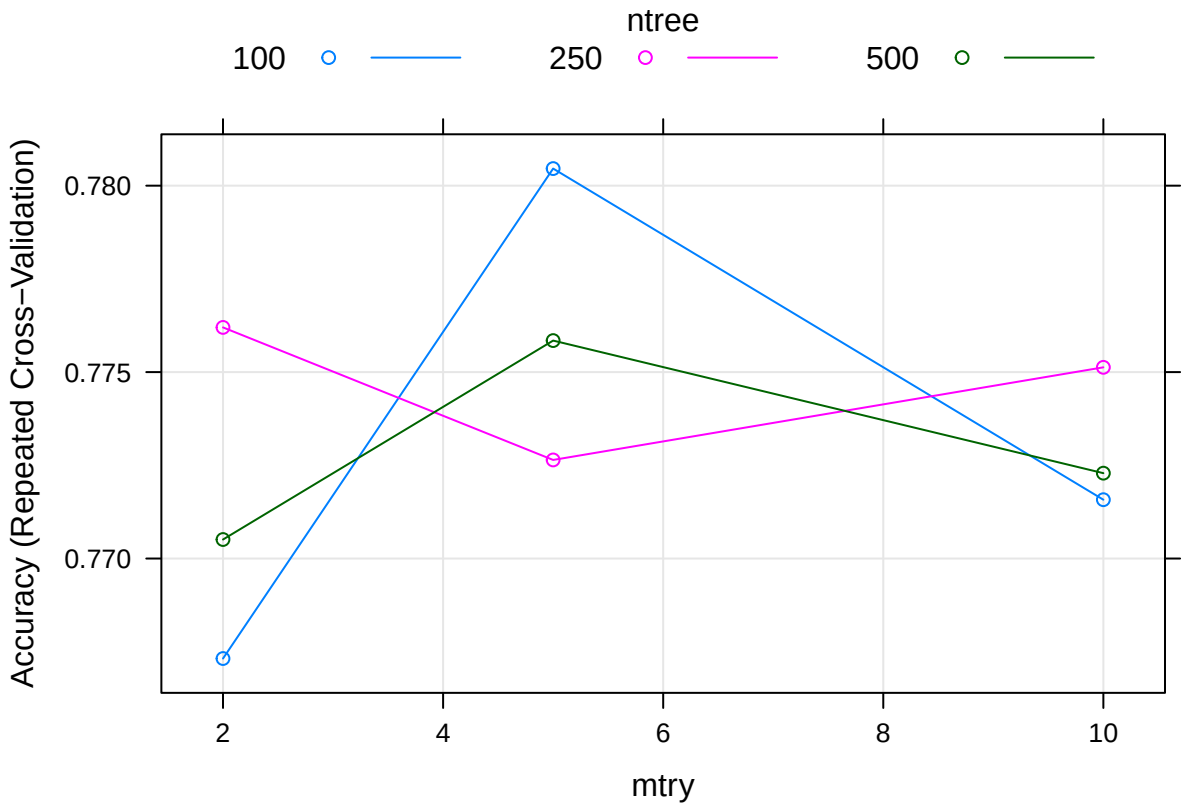
4. Fit a Random Forest (RF) classifier to predict the aqueous solubility. Tune the parameters: number of trees and number of variables in each tree, by implementing a grid search procedure. Assess the performance of RF using suitable metrics. Determine which variables are the most relevant in the solubility prediction.

```
#method customization
customRF <- list(type = "Classification",
                 library = "randomForest",
                 loop = NULL)

customRF$parameters <- data.frame(parameter = c("mtry", "ntree"),
                                  class = rep("numeric", 2),
                                  label = c("mtry", "ntree"))

customRF$grid <- function(x, y, len = NULL, search = "grid") {}

customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
  randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...)
}

customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata)

customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata, type = "prob")

customRF$sort <- function(x) x[order(x[,1]),]
customRF$levels <- function(x) x$classes

# train model
ctrl.rf <- trainControl(method = "repeatedcv",number = 5, repeats=1) #reduced due to co
tunegrid <- expand.grid(.mtry=c(2,5,10), .ntree=c(100, 250,500))
set.seed(1234)
random.forest <- train(y~., data=data_train, method=customRF, metric="Accuracy", trContr

random.forest
```

```
## 2815 samples
##   71 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (71), scaled (71)
## Resampling: Cross-Validated (5 fold, repeated 1 times)
```

```
## Summary of sample sizes: 2252, 2252, 2251, 2253, 2252
## Resampling results across tuning parameters:
##
##   mtry  ntree  Accuracy   Kappa
##    2    100    0.7673176  0.4808074
##    2    250    0.7761986  0.5020021
##    2    500    0.7705097  0.4888318
##    5    100    0.7804584  0.5130932
##    5    250    0.7726437  0.4957862
##    5    500    0.7758453  0.5032504
##   10    100    0.7715780  0.4949540
##   10    250    0.7751279  0.5028064
##   10    500    0.7722891  0.4965549
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5 and ntree = 100.
```

```
plot(random.forest)
```



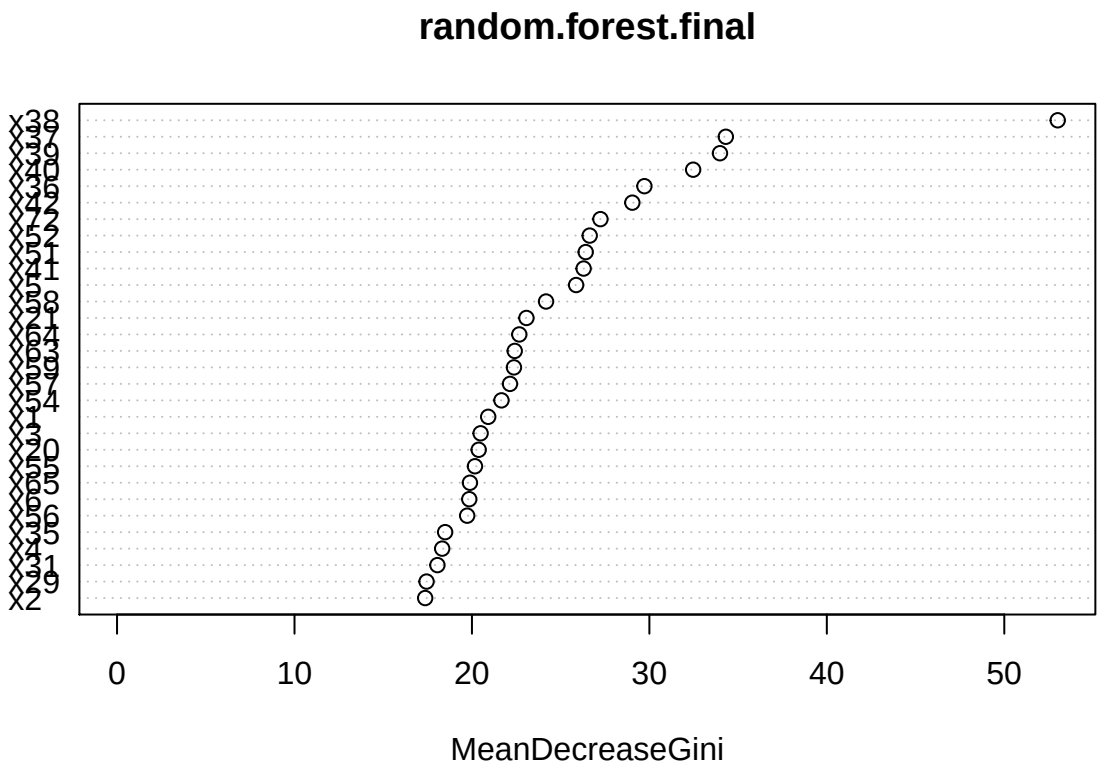
Final random forest model:

```
random.forest.final <- randomForest(y~.,data=data_train ,mtry=5, ntree = 100)
random.forest.final
```

```
##
```

```
## Call:
##  randomForest(formula = y ~ ., data = data_train, mtry = 5, ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 22.7%
## Confusion matrix:
##      0   1 class.error
## 0 1519 227   0.1300115
## 1  412 657   0.3854069
```

```
varImpPlot(random.forest.final)
```



Performance of the random forest

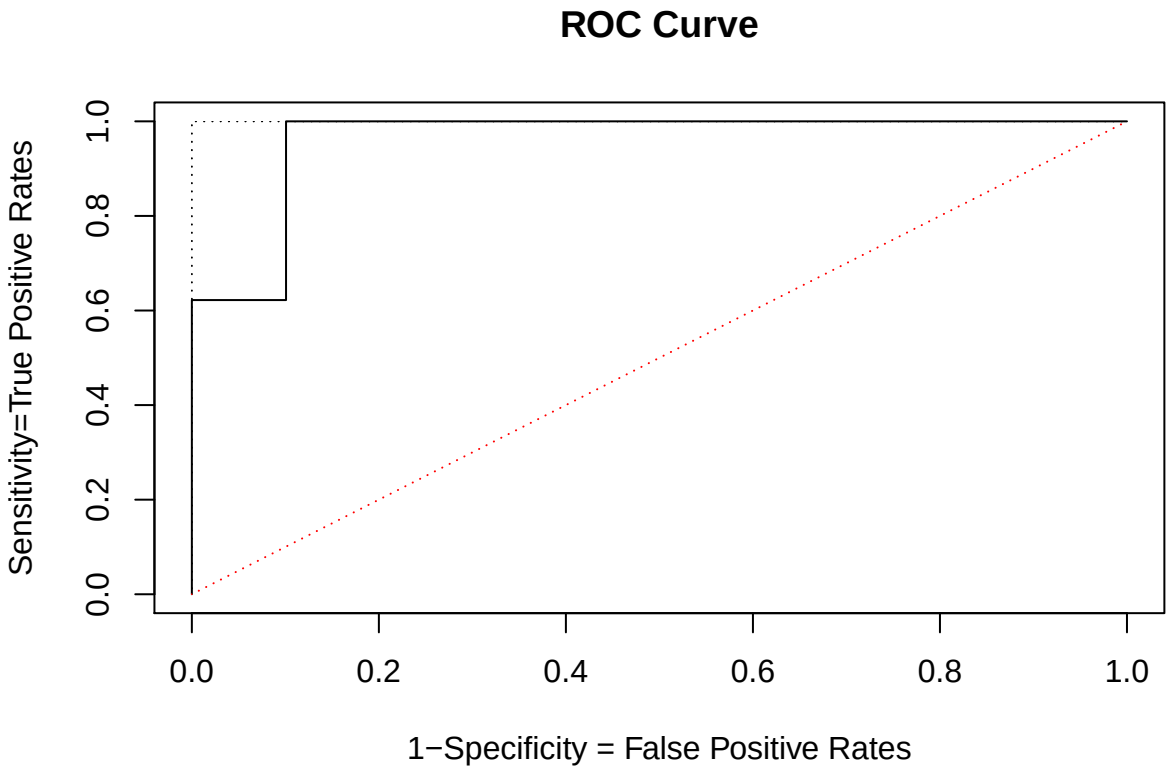
```
random.forest.pred <- predict(random.forest.final,data_test)

confusionMatrix(random.forest.pred,data_test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1571  404
##          1  176  665
```

```
##
##                Accuracy : 0.794
##                  95% CI : (0.7786, 0.8088)
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5438
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8993
##             Specificity : 0.6221
##          Pos Pred Value : 0.7954
##          Neg Pred Value : 0.7907
##              Prevalence : 0.6204
##          Detection Rate : 0.5579
##    Detection Prevalence : 0.7013
##       Balanced Accuracy : 0.7607
##
##        'Positive' Class : 0
##
```

We can also see the performance of the random forest with the ROC analysis:

```
random.forest.roc <- roc(as.numeric(data_test$y),as.numeric(random.forest.pred))

plot(x=1-random.forest.roc$specificities, y=random.forest.roc$sensitivities, ylab="Sensi
clip(x1=0,x2=1,y1=0,y2=1)
abline(a=0, b=1,col=2,lty=3)
abline(v=0,col=1,lty=3)
abline(h=1,col=1,lty=3)
```




```
random.forest.roc$auc
```

```
## Area under the curve: 0.7607
```

5. In view of the above metrics, compare the classifiers in 3) and 4).

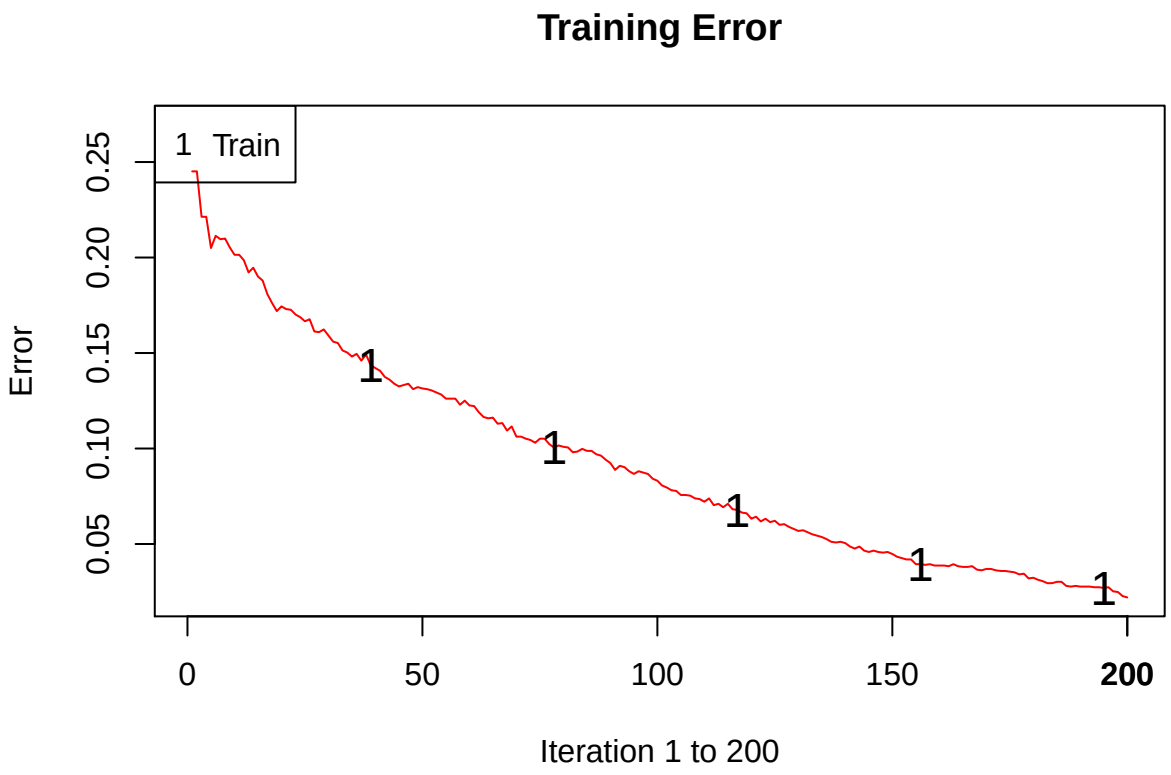
We can clearly see that the random forest obtains a better results in terms of accuracy in the classification of the data tested (8.95% more accuracy in the predictions).

6. Apply the discrete AdaBoost algorithm (with an exponential loss function).

6.1. Using "stumps" as classification trees compute the misclassification rates of both the learning set and the test set across 2,000 iterations of AdaBoost. Represent these error as a function of the number of boosting iterations. Due to computational limitations for the AdaBoost, we did only 200 iterations.

```
AdaBoost <- ada(x=data_train[,-72],
                y=data_train[,72],
                loss="exponential",
                iter=200,
```

```
                rpart.control(maxdepth = 1, cp = -1, minsplit=0))


plot(AdaBoost)
```




```
AdaBoost
```

```
## Call:
## ada(data_train[, -72], y = data_train[, 72], test.x = rpart.control(maxdepth = 1,
##     cp = -1, minsplit = 0), loss = "exponential", iter = 200)
##
## Loss: exponential Method: discrete   Iteration: 200
##
## Final Confusion Matrix for Data:
##           Final Prediction
## True value    0    1
##          0 1724   22
##          1   40 1029
##
## Train Error: 0.022
##
## Out-Of-Bag Error:  0.077  iteration= 200
##
## Additional Estimates of number of iterations:
```

```
##
## train.err1 train.kap1
##        200        200
```

Performance in the learning set:

```
AdaBoost.pred.train <- predict(AdaBoost,newdata=data_train,n.trees=200)

confusionMatrix(as.factor(AdaBoost.pred.train),data_train$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1724   40
##          1   22 1029
##
##                Accuracy : 0.978
##                  95% CI : (0.9719, 0.9831)
##     No Information Rate : 0.6202
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9531
##  Mcnemar's Test P-Value : 0.03085
##
##             Sensitivity : 0.9874
##             Specificity : 0.9626
##          Pos Pred Value : 0.9773
##          Neg Pred Value : 0.9791
##              Prevalence : 0.6202
##          Detection Rate : 0.6124
##    Detection Prevalence : 0.6266
##       Balanced Accuracy : 0.9750
##
##        'Positive' Class : 0
##
```

Performance in the test set:

```
AdaBoost.pred.test <- predict(AdaBoost,newdata=data_test,n.trees=200)

confusionMatrix(as.factor(AdaBoost.pred.test),data_test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
```

```
##          0 1525  388
##          1  222  681
##
##                Accuracy : 0.7834
##                  95% CI : (0.7677, 0.7985)
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5258
##  Mcnemar's Test P-Value : 2.379e-11
##
##             Sensitivity : 0.8729
##             Specificity : 0.6370
##          Pos Pred Value : 0.7972
##          Neg Pred Value : 0.7542
##              Prevalence : 0.6204
##          Detection Rate : 0.5415
##    Detection Prevalence : 0.6793
##       Balanced Accuracy : 0.7550
##
##        'Positive' Class : 0
##
```

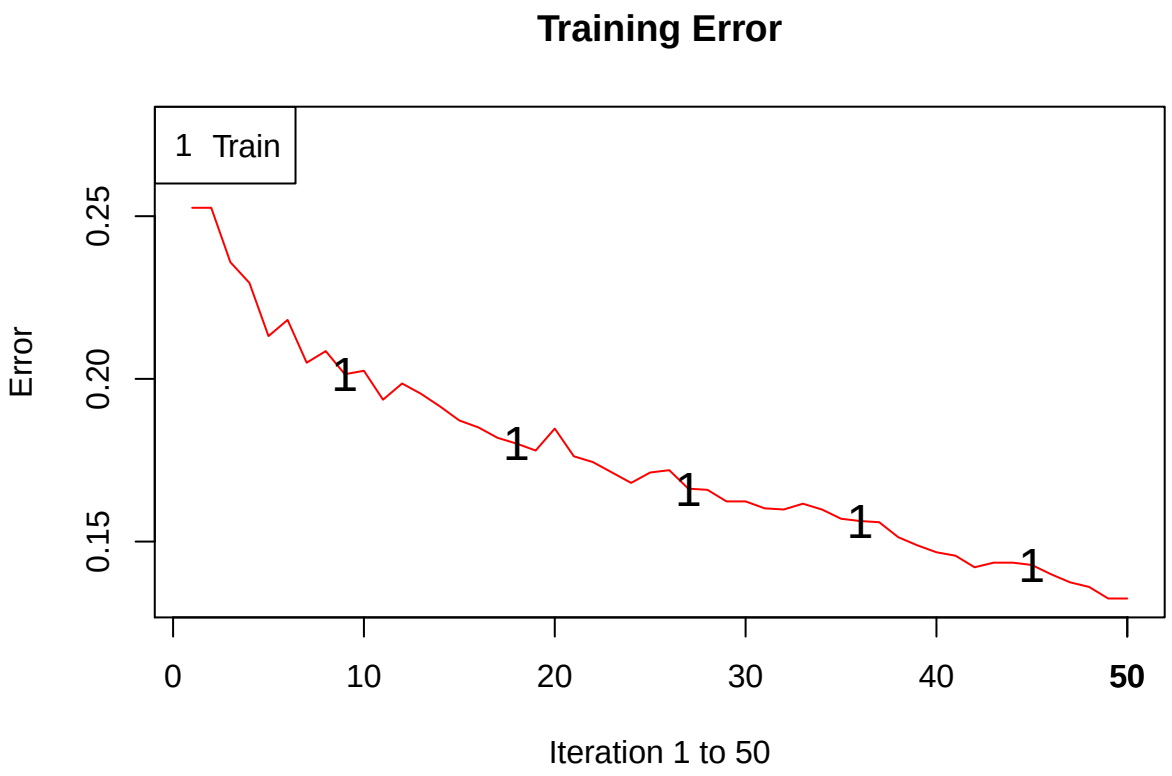
6.2. Compare the test-set misclassification rates attained by different ensemble classifiers based on trees of sizes: stumps, 4-node trees, 8-node trees, and 16-node trees.

```
AdaBoost.stump=ada(x=data_train[,-72],
                   y=data_train[,72],
                   loss="exponential",
                   iter=50,
                   rpart.control(maxdepth = 1, cp = -1, minsplit=0))

AdaBoost.4=ada(x=data_train[,-72],
                   y=data_train[,72],
                   loss="exponential",
                   iter=50,
                   rpart.control(maxdepth = 4, cp = -1, minsplit=0))

AdaBoost.8=ada(x=data_train[,-72],
                   y=data_train[,72],
                   loss="exponential",
                   iter=50,
```

```
                rpart.control(maxdepth = 8, cp = -1, minsplit=0))

AdaBoost.16=ada(x=data_train[,-72],
                y=data_train[,72],
                loss="exponential",
                iter=50,
                rpart.control(maxdepth = 16, cp = -1, minsplit=0))
```

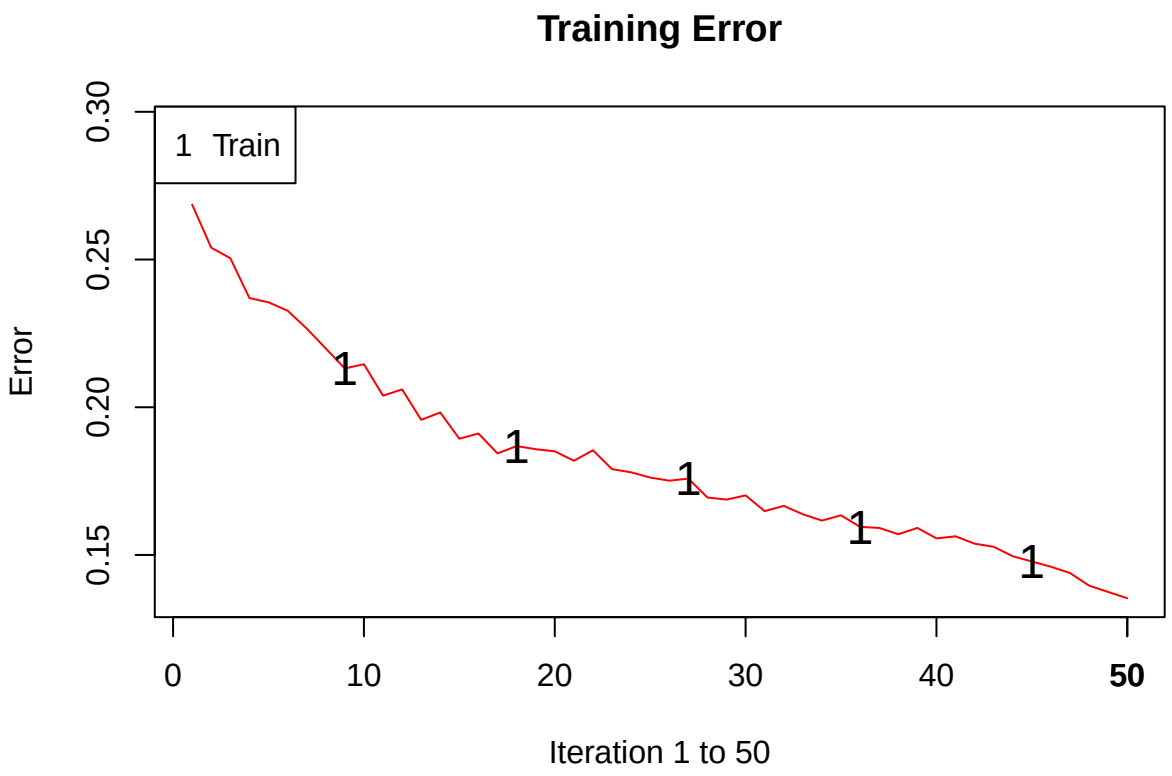
Stump Tree:

```
plot(AdaBoost.stump)
```



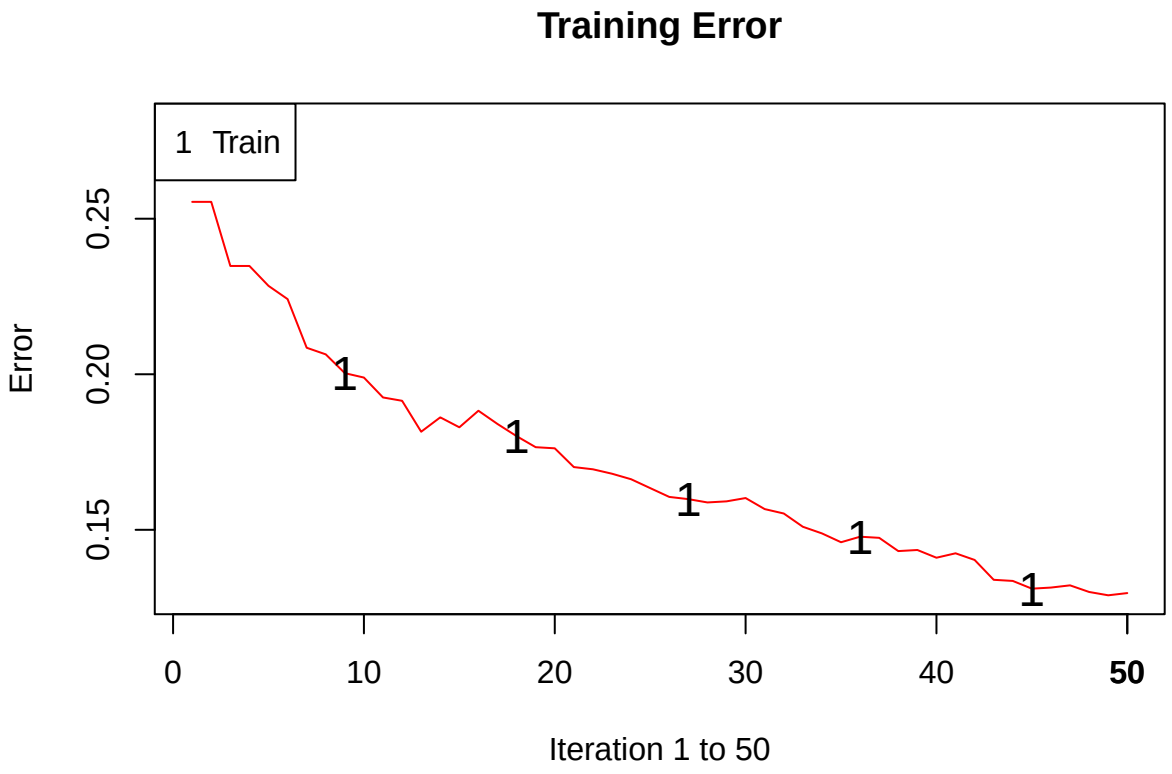
4-node Tree:

```
plot(AdaBoost.4)
```



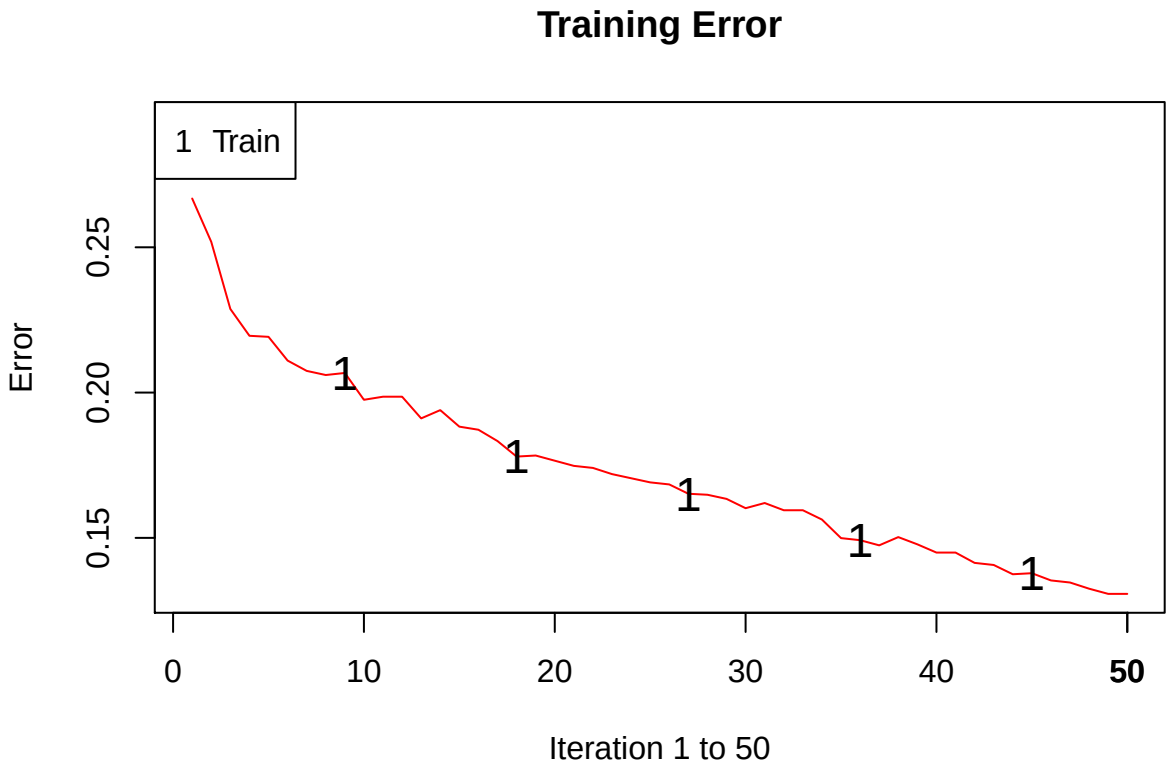
8-node Tree:

```
plot(AdaBoost.8)
```



16-node Tree:

```r
plot(AdaBoost.16)
```



Performance in the test set for each tree:

Stump tree:

```r
AdaBoost.stump.test <- predict(AdaBoost.stump,newdata=data_test,n.trees=50)

confusionMatrix(as.factor(AdaBoost.stump.test),data_test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1545  428
##          1  202  641
##
##                Accuracy : 0.7763
##                  95% CI : (0.7604, 0.7916)
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5047
##  Mcnemar's Test P-Value : < 2.2e-16
##
```

```
##             Sensitivity : 0.8844
##             Specificity : 0.5996
##          Pos Pred Value : 0.7831
##          Neg Pred Value : 0.7604
##              Prevalence : 0.6204
##          Detection Rate : 0.5487
##    Detection Prevalence : 0.7006
##       Balanced Accuracy : 0.7420
##
##        'Positive' Class : 0
##
```

4-node tree:

```
AdaBoost.4.test <- predict(AdaBoost.4,newdata=data_test,n.trees=50)

confusionMatrix(as.factor(AdaBoost.4.test),data_test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1543  434
##          1  204  635
##
##                Accuracy : 0.7734
##                  95% CI : (0.7575, 0.7888)
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.498
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8832
##             Specificity : 0.5940
##          Pos Pred Value : 0.7805
##          Neg Pred Value : 0.7569
##              Prevalence : 0.6204
##          Detection Rate : 0.5479
##    Detection Prevalence : 0.7021
##       Balanced Accuracy : 0.7386
##
##        'Positive' Class : 0
##
```

8-node tree:

```
AdaBoost.8.test <- predict(AdaBoost.8,newdata=data_test,n.trees=50)

confusionMatrix(as.factor(AdaBoost.8.test),data_test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1557  440
##          1  190  629
##
##                Accuracy : 0.7763
##                  95% CI : (0.7604, 0.7916)
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5024
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8912
##             Specificity : 0.5884
##          Pos Pred Value : 0.7797
##          Neg Pred Value : 0.7680
##              Prevalence : 0.6204
##          Detection Rate : 0.5529
##    Detection Prevalence : 0.7092
##       Balanced Accuracy : 0.7398
##
##        'Positive' Class : 0
##
```

16-node tree:

```
AdaBoost.16.test <- predict(AdaBoost.16,newdata=data_test,n.trees=50)

confusionMatrix(as.factor(AdaBoost.16.test),data_test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1550  424
##          1  197  645
##
##                Accuracy : 0.7795
##                  95% CI : (0.7637, 0.7947)
```

```
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5117
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8872
##             Specificity : 0.6034
##          Pos Pred Value : 0.7852
##          Neg Pred Value : 0.7660
##              Prevalence : 0.6204
##          Detection Rate : 0.5504
##    Detection Prevalence : 0.7010
##       Balanced Accuracy : 0.7453
##
##        'Positive' Class : 0
##
```