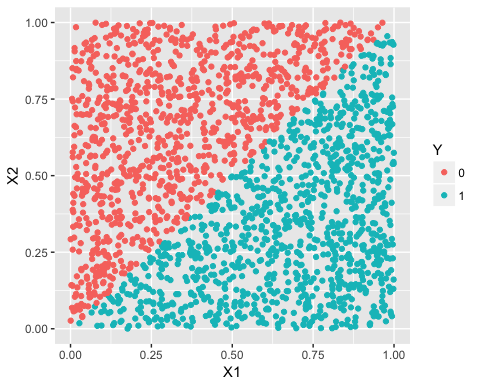
R-Lab: Random Forests

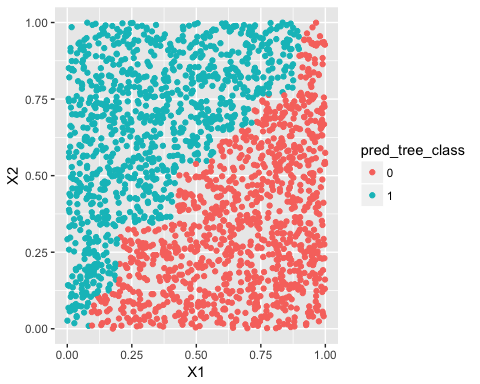
Here a data set with two predictor variables and a class variable are simulated. The classes are sepearable by a linear combination of the two predictor variables.

rm(list = ls(all = TRUE))  
library(rpart)  
library(dplyr)  
library(ggplot2)  
library(randomForest)  
ndata <- 2000  
X1 <- runif(ndata, min = 0, max = 1)  
X2 <- runif(ndata, min = 0, max = 1)  
data <- data.frame(X1, X2)  
data <- data %>% mutate(X12 = 0.5 \* (X1 - X2), Y = ifelse(X12 >= 0, 1, 0))  
data <- data %>% select(-X12) %>% mutate(Y = as.factor(as.character(Y)))  
ggplot(data, aes(x = X1, y = X2, color = Y)) + geom\_point() #+labs(title = 'Data points')

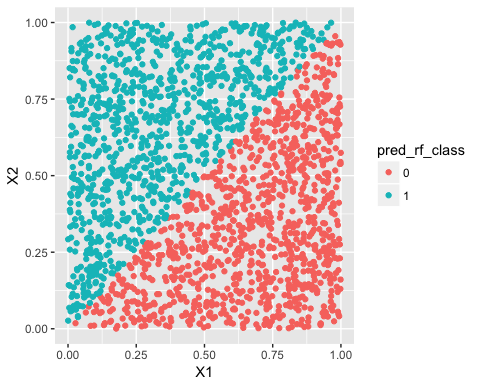


Random forests and a single decision tree are applied to the data. The predictions are plotted in the following graphs. The classification boundary of random forets are much smoother than the one of the decision tree, and is a better approximation of the linear classification boundary.

rf\_model <- randomForest(Y ~ ., data = data)  
tree\_model <- rpart(Y ~ ., data = data)  
  
pred\_rf <- predict(rf\_model, data, type = "prob")[, 1]  
pred\_tree <- predict(tree\_model, data, type = "prob")[, 1]  
data\_pred <- data %>% mutate(pred\_rf\_class = ifelse(pred\_rf < 0.5, 0, 1)) %>%   
 mutate(pred\_rf\_class = as.factor(as.character(pred\_rf\_class))) %>% mutate(pred\_tree\_class = ifelse(pred\_tree <   
 0.5, 0, 1)) %>% mutate(pred\_tree\_class = as.factor(as.character(pred\_tree\_class)))  
ggplot(data\_pred, aes(x = X1, y = X2, color = pred\_tree\_class)) + geom\_point() #+labs(title = 'Classification bounday from a single decision tree')

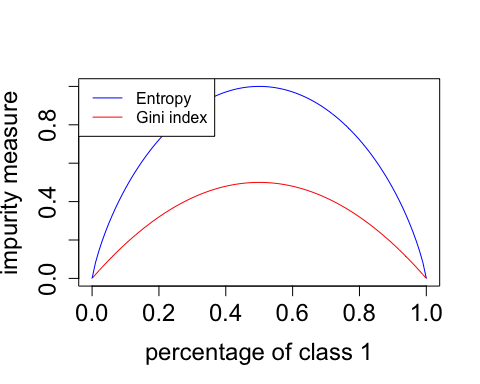


ggplot(data\_pred, aes(x = X1, y = X2, color = pred\_rf\_class)) + geom\_point() #+labs(title = 'Classification bounday from random forests')



Here we plot the Gini index and entropy values versus the percentage of class 1 (for two-class problems).

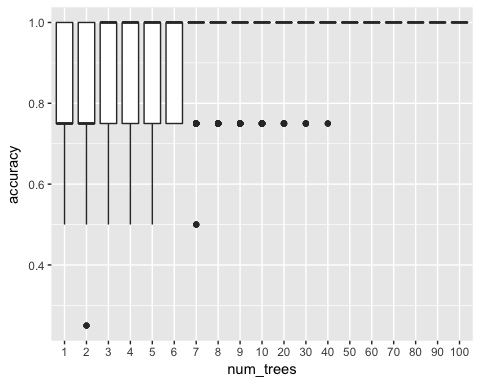
entropy <- function(p\_v) {  
 e <- 0  
 for (p in p\_v) {  
 if (p == 0) {  
 this\_term <- 0  
 } else {  
 this\_term <- -p \* log2(p)  
 }  
 e <- e + this\_term  
 }  
 return(e)  
}  
gini <- function(p\_v) {  
 e <- 0  
 for (p in p\_v) {  
 if (p == 0) {  
 this.term <- 0  
 } else {  
 this.term <- p \* (1 - p)  
 }  
 e <- e + this.term  
 }  
 return(e)  
}  
  
entropy.v <- NULL  
gini.v <- NULL  
p.v <- seq(0, 1, by = 0.01)  
for (p in p.v) {  
 entropy.v <- c(entropy.v, (entropy(c(p, 1 - p))))  
 gini.v <- c(gini.v, (gini(c(p, 1 - p))))  
}  
plot(p.v, gini.v, type = "l", ylim = c(0, 1), xlab = "percentage of class 1",   
 col = "red", ylab = "impurity measure", cex.lab = 1.5, cex.axis = 1.5, cex.main = 1.5,   
 cex.sub = 1.5)  
lines(p.v, entropy.v, col = "blue")  
legend("topleft", legend = c("Entropy", "Gini index"), col = c("blue", "red"),   
 lty = c(1, 1), cex = 1)



p <- 3/4  
gini(c(p, 1 - p))

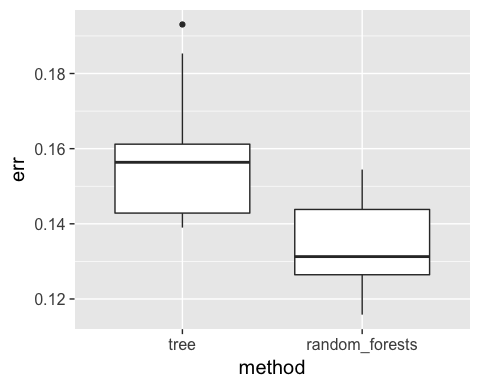
## [1] 0.375

library(dplyr)  
library(ggplot2)  
library(randomForest)  
set.seed(1)  
data <- rbind(c("0", "0", "C0"), c("1", "0", "C1"), c("0", "1", "C1"), c("0",   
 "0", "C0")) %>% as.data.frame()  
colnames(data) <- c("X1", "X2", "Classs")  
  
results <- NULL  
for (i in c(1:9, (1:10) \* 10)) {  
 for (replicate in 1:200) {  
 rf.model <- randomForest(Classs ~ ., data = data, ntree = i, keep.inbag = TRUE)  
 pred.rf <- predict(rf.model, data, type = "class")  
 err <- (length(which(pred.rf == data$Classs))/length(data$Classs))  
 results <- rbind(results, c(i, err))  
 }  
}  
colnames(results) <- c("num\_trees", "accuracy")  
results <- as.data.frame(results) %>% mutate(num\_trees = as.character(num\_trees))  
levels(results$num\_trees) <- unique(results$num\_trees)  
results$num\_trees <- factor(results$num\_trees, unique(results$num\_trees))  
ggplot() + geom\_boxplot(data = results, aes(y = accuracy, x = num\_trees)) +   
 geom\_point(size = 3)



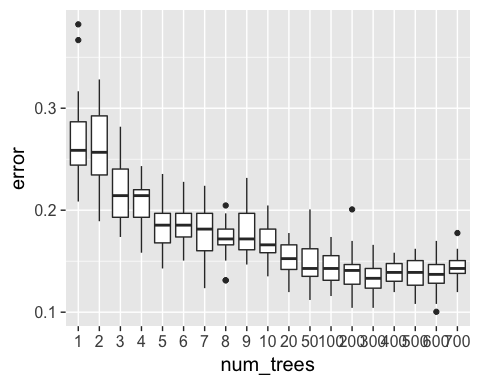
We apply both decision tree (rpart) and random forests to the re-admission data set. Half of the data stes are used for training and the other half for testing. This is run for 20 times, and the box plots of the errors from decision tree and random forests are plotted. Clearly the error rates of decision tree are higher than random forests.

library(rpart)  
library(dplyr)  
library(tidyr)  
library(ggplot2)  
library(randomForest)  
library(RCurl)  
set.seed(1)  
  
theme\_set(theme\_gray(base\_size = 15))  
  
data <- read.csv(text = getURL("https://raw.githubusercontent.com/shuailab/ind\_498/master/resource/data/AD.csv"))  
  
target\_indx <- which(colnames(data) == "DX\_bl")  
data[, target\_indx] <- as.factor(paste0("c", data[, target\_indx]))  
rm\_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))  
data <- data[, -rm\_indx]  
  
err.tree <- NULL  
err.rf <- NULL  
for (i in 1:20) {  
 train.ix <- sample(nrow(data), floor(nrow(data)/2))  
 tree <- rpart(DX\_bl ~ ., data = data[train.ix, ])  
 pred.test <- predict(tree, data[-train.ix, ], type = "class")  
 err.tree <- c(err.tree, length(which(pred.test != data[-train.ix, ]$DX\_bl))/length(pred.test))  
   
 rf <- randomForest(DX\_bl ~ ., data = data[train.ix, ])  
 pred.test <- predict(rf, data[-train.ix, ], type = "class")  
 err.rf <- c(err.rf, length(which(pred.test != data[-train.ix, ]$DX\_bl))/length(pred.test))  
}  
err.tree <- data.frame(err = err.tree, method = "tree")  
err.rf <- data.frame(err = err.rf, method = "random\_forests")  
  
ggplot() + geom\_boxplot(data = rbind(err.tree, err.rf), aes(y = err, x = method)) +   
 geom\_point(size = 3)



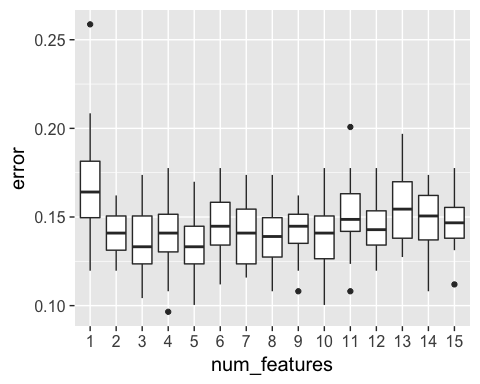
Now we investigate the impact of number of trees, and number of features on the accuracy performance of random forests. First consider the number of trees. For each number of trees, 20 runs are conducated, and the box plots for each setting are shown below. It can be seen, when the number of trees is small, particularly less than 10, the improment is substantial with additional trees added. However, the error rate is stable after the number of trees reaches 100.

library(dplyr)  
library(tidyr)  
library(ggplot2)  
require(randomForest)  
set.seed(1)  
  
theme\_set(theme\_gray(base\_size = 15))  
  
data <- read.csv(text = getURL("https://raw.githubusercontent.com/shuailab/ind\_498/master/resource/data/AD.csv"))  
  
target\_indx <- which(colnames(data) == "DX\_bl")  
data[, target\_indx] <- as.factor(paste0("c", data[, target\_indx]))  
rm\_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))  
data <- data[, -rm\_indx]  
  
results <- NULL  
for (itree in c(1:9, 10, 20, 50, 100, 200, 300, 400, 500, 600, 700)) {  
 for (i in 1:20) {  
 train.ix <- sample(nrow(data), floor(nrow(data)/2))  
 rf <- randomForest(DX\_bl ~ ., ntree = itree, data = data[train.ix, ])  
 pred.test <- predict(rf, data[-train.ix, ], type = "class")  
 this.err <- length(which(pred.test != data[-train.ix, ]$DX\_bl))/length(pred.test)  
 results <- rbind(results, c(itree, this.err))  
 # err.rf <- c(err.rf, length(which(pred.test !=  
 # data[-train.ix,]$DX\_bl))/length(pred.test) )  
 }  
}  
  
colnames(results) <- c("num\_trees", "error")  
results <- as.data.frame(results) %>% mutate(num\_trees = as.character(num\_trees))  
levels(results$num\_trees) <- unique(results$num\_trees)  
results$num\_trees <- factor(results$num\_trees, unique(results$num\_trees))  
ggplot() + geom\_boxplot(data = results, aes(y = error, x = num\_trees)) + geom\_point(size = 3)



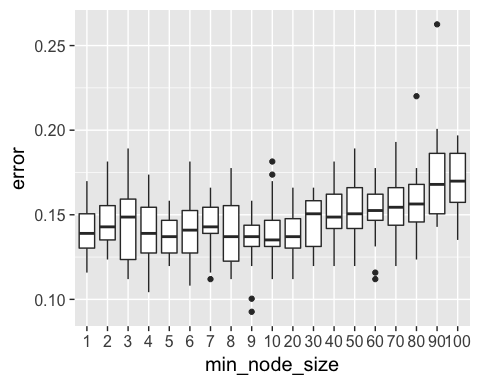
Next consider the number of features. 100 trees are used. For each number of features, 20 runs are conducated, and the box plots for each setting are shown below. In this problem, the error rates are not significantly different when the number of features changes.

library(dplyr)  
library(tidyr)  
library(ggplot2)  
require(randomForest)  
set.seed(1)  
  
theme\_set(theme\_gray(base\_size = 15))  
  
data <- read.csv(text = getURL("https://raw.githubusercontent.com/shuailab/ind\_498/master/resource/data/AD.csv"))  
  
target\_indx <- which(colnames(data) == "DX\_bl")  
data[, target\_indx] <- as.factor(paste0("c", data[, target\_indx]))  
rm\_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))  
data <- data[, -rm\_indx]  
nFea <- ncol(data) - 1  
results <- NULL  
for (iFeatures in 1:nFea) {  
 for (i in 1:20) {  
 train.ix <- sample(nrow(data), floor(nrow(data)/2))  
 rf <- randomForest(DX\_bl ~ ., mtry = iFeatures, ntree = 100, data = data[train.ix,   
 ])  
 pred.test <- predict(rf, data[-train.ix, ], type = "class")  
 this.err <- length(which(pred.test != data[-train.ix, ]$DX\_bl))/length(pred.test)  
 results <- rbind(results, c(iFeatures, this.err))  
 # err.rf <- c(err.rf, length(which(pred.test !=  
 # data[-train.ix,]$DX\_bl))/length(pred.test) )  
 }  
}  
  
colnames(results) <- c("num\_features", "error")  
results <- as.data.frame(results) %>% mutate(num\_features = as.character(num\_features))  
levels(results$num\_features) <- unique(results$num\_features)  
results$num\_features <- factor(results$num\_features, unique(results$num\_features))  
ggplot() + geom\_boxplot(data = results, aes(y = error, x = num\_features)) +   
 geom\_point(size = 3)



As mentioned, trees in random forests are fully grown. Here we experiment with the minimum node size, that is, the minimum number of isntances at a node. Again each setting is run 20 times and box plots are shown below. It can be seen, the error rates start to rise at minimum node size euqal to 40, and the error rates when minimum node size less than 40 are not substantially different. More importantly, this shows that a fully-grown tree, that is, minimum node size equal to 1, does not hurt the accuracy performance of random forests.

library(dplyr)  
library(tidyr)  
library(ggplot2)  
require(randomForest)  
set.seed(1)  
  
theme\_set(theme\_gray(base\_size = 15))  
  
data <- read.csv(text = getURL("https://raw.githubusercontent.com/shuailab/ind\_498/master/resource/data/AD.csv"))  
  
target\_indx <- which(colnames(data) == "DX\_bl")  
data[, target\_indx] <- as.factor(paste0("c", data[, target\_indx]))  
rm\_indx <- which(colnames(data) %in% c("ID", "TOTAL13", "MMSCORE"))  
data <- data[, -rm\_indx]  
  
results <- NULL  
for (inodesize in c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80,   
 90, 100)) {  
 for (i in 1:20) {  
 train.ix <- sample(nrow(data), floor(nrow(data)/2))  
 rf <- randomForest(DX\_bl ~ ., ntree = 100, nodesize = inodesize, data = data[train.ix,   
 ])  
 pred.test <- predict(rf, data[-train.ix, ], type = "class")  
 this.err <- length(which(pred.test != data[-train.ix, ]$DX\_bl))/length(pred.test)  
 results <- rbind(results, c(inodesize, this.err))  
 # err.rf <- c(err.rf, length(which(pred.test !=  
 # data[-train.ix,]$DX\_bl))/length(pred.test) )  
 }  
}  
  
colnames(results) <- c("min\_node\_size", "error")  
results <- as.data.frame(results) %>% mutate(min\_node\_size = as.character(min\_node\_size))  
levels(results$min\_node\_size) <- unique(results$min\_node\_size)  
results$min\_node\_size <- factor(results$min\_node\_size, unique(results$min\_node\_size))  
ggplot() + geom\_boxplot(data = results, aes(y = error, x = min\_node\_size)) +   
 geom\_point(size = 3)



comb = function(n, x) {  
 return(factorial(n)/(factorial(x) \* factorial(n - x)))  
}  
comb(100, 60)

## [1] 1.374623e+28

prob <- function(n, k, p) {  
 return(comb(n, k) \* p^k \* (1 - p)^(n - k))  
}  
  
probSum <- 0  
for (i in 51:100) {  
 probSum <- probSum + prob(100, i, 0.6)  
}  
print(probSum)

## [1] 0.9729008

probSum <- 0  
for (i in 6:10) {  
 probSum <- probSum + prob(10, i, 0.6)  
}  
print(probSum)

## [1] 0.6331033

In the following we use random forests for clustering. In random forests, a synthetic data set is generated with the same size as the original data set. One can permuate each variable in the original data set to generate the synthetic data set. Then assign one class to the original data, and another class to the synthetic data, and random forests are used to classify the two data sets. Then based on the frequency of a pair of data points existing in the same node, a distance, referred to as the proximity, between the two data points can be calcualted. And clustering algorithms based on data pair distances can be applied to produce the clusters. The advantage using random forests to produce the distances, is that random forests are not sensitive to the unit of a variable and can be applied to both categorical and numerical variables. In the following we generate a data set combined by two normally distributed data sets with different means. The clusters produced from random forests are shown. It can be seen the two clusters are reasonably consistent with the underlying distributions.

rm(list = ls(all = TRUE))  
library(rpart)  
library(dplyr)  
library(ggplot2)  
library(randomForest)  
library(MASS)  
library(cluster)  
ndata <- 2000  
  
sigma <- matrix(c(1, 0, 0, 1), 2, 2)  
data1 <- mvrnorm(n = 500, rep(0, 2), sigma)  
data2 <- mvrnorm(n = 500, rep(3, 2), sigma)  
data <- rbind(data1, data2)  
rf <- randomForest(data)  
prox <- rf$proximity  
clusters <- pam(prox, 2)  
data <- as.data.frame(data)  
data$cluster <- as.character(clusters$clustering)  
ggplot(data, aes(x = V1, y = V2, color = cluster)) + geom\_point() #+labs(title = 'Data points')

