Advanced Methods for Regression and Classification

Exercise 10

Bosse Behrens , st.id: 12347333

First we load the pdata.

```
library(ISLR)
data(Caravan)
```

Now we split the data into train and test sets, while keeping up the ratio of the train/test split for both classes Yes and No.

```
data <- Caravan
set.seed(12347333)

#data$Purchase <- ifelse(data$Purchase == "Yes", 1, 0)

data_yes <- data[data$Purchase == "Yes", ]
data_no <- data[data$Purchase == "No", ]

n_yes <- nrow(data_yes)
n_no <- nrow(data_no)

train_yes <- sample(1:n_yes, round((2/3) * n_yes))
test_yes <-(1:n_yes)[-train_yes]
train_no <- sample(1:n_no, round((2/3) * n_no))
test_no <-(1:n_no)[-train_no]


train <- c(train_yes, train_no)

test <- c(test_yes, test_no)
```

Task 1

part a)

We load the rpart package.

```
library(rpart)
```

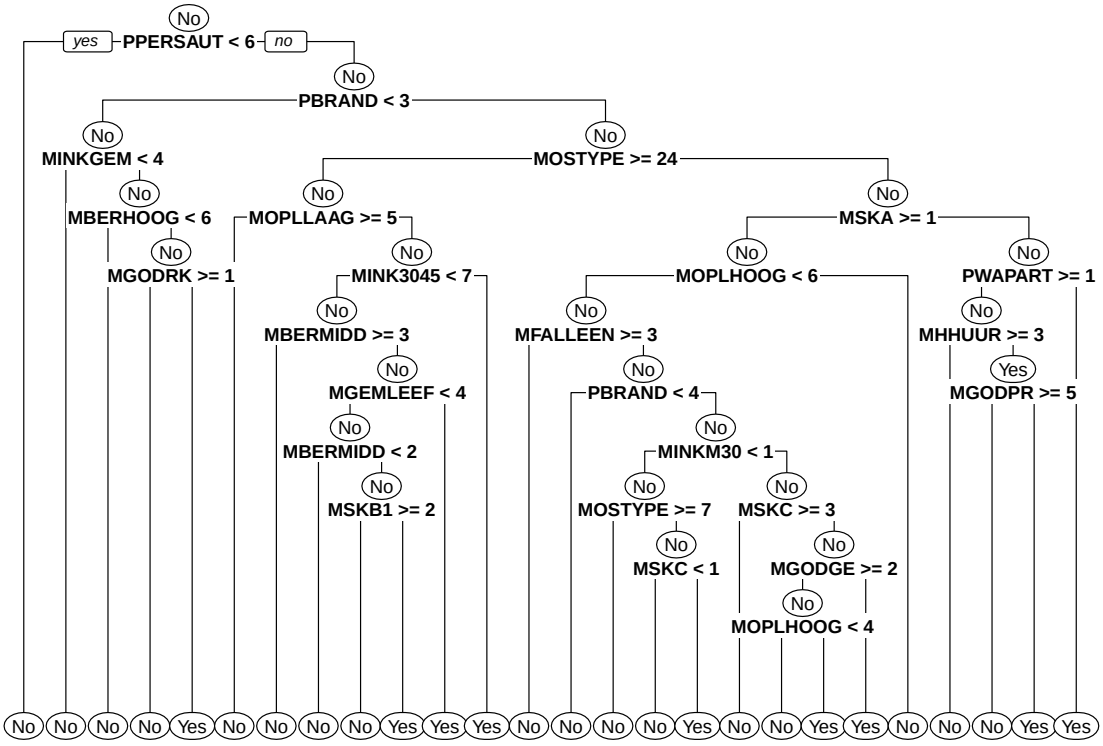
The first regression decision tree is computed. We use the same parameters as in the lecture notes.

```
tree_t0 <- rpart(Purchase~.,data=data,subset = train, xval=20, cp=0.001)
```

Now we plot the computed tree. Since using plot() and text() results in a very unsightly and cluttered plot,w e use the package rpart.plot to get a betetr plot.

```
library(rpart.plot)

rpart.plot(tree_t0, extra = 0, under = TRUE, type = 2, box.palette = NULL, cex = 0.6)
```



As we can see, the tree has been computed. As usual for decision trees, each classification goes through the tree decision process, by checking with the predictor variables on each level and then deciding which branch to progress by comparing with the computed value threshold in the tree. For example for each data point that gets classified, first the predictor PPERSAUT is checked if its value is smaller than 6 or equal/larger. Then on the deciosion tree depending on the outcome a branch is progressed on. For example if PPERSAUT is smaller than 6, we progress on the left branch which immediately results in the classification of "No", while on the right further branches are considered.

part c)

We predict the classes using our regression tree and get the confusion amtrix as well as the balanced accuracy.

```
t0_pred <- predict(tree_t0, newdata = data[test,], type = "class")

conf_matrix <- table(data[test,]$Purchase, t0_pred, dnn = c("Actual", "Predicted"))
print(conf_matrix)
```

```
##        Predicted
## Actual   No  Yes
##    No  1790   38
##    Yes  105    8
```

```
recalls <- diag(prop.table(conf_matrix, 1))
balanced_accuracy <- mean(recalls)

cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

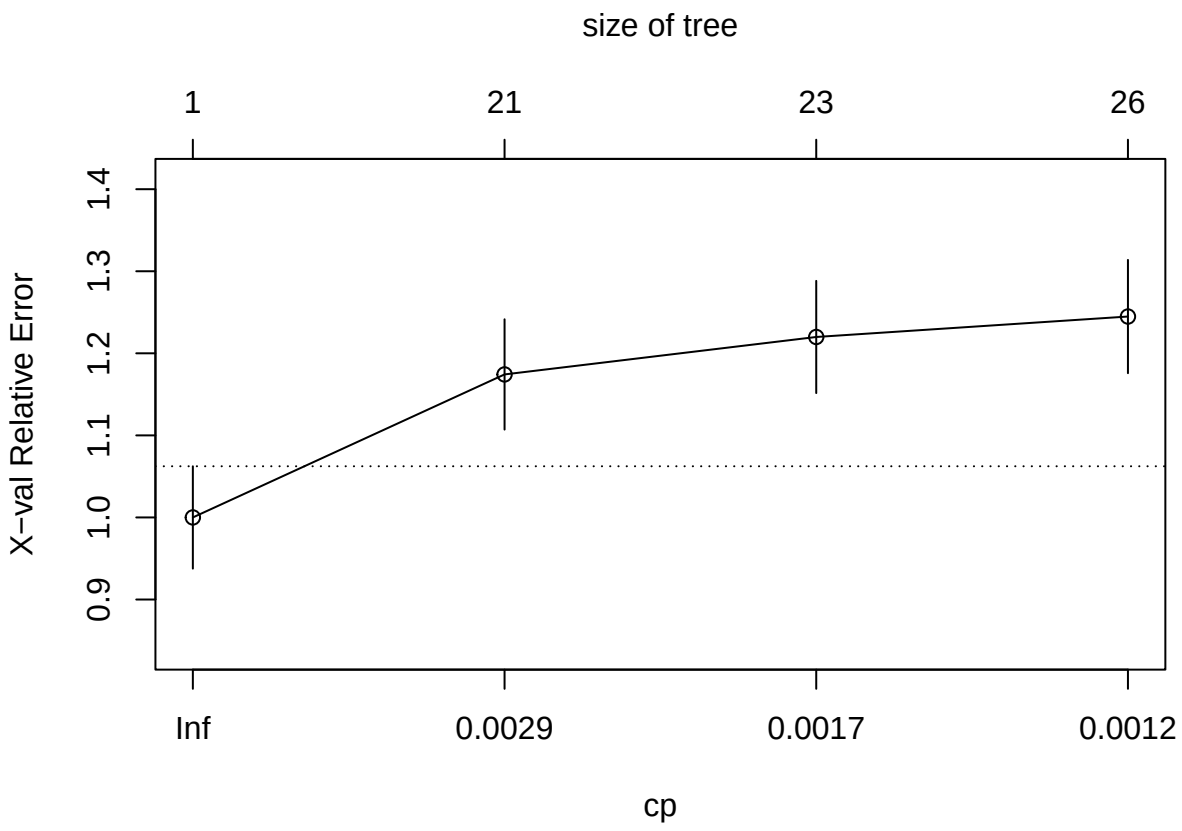
```
## Balanced Accuracy: 0.525
```

As we can see the tree is very bad at classifying the data with a balaned accuracy very close to 0.5, which is similar to random guessing.

part d)

We geth the cv-plot as well as also the complexity table witht he specific values.

```
plotcp(tree_t0)
```




```
print(tree_t0$cptable)
```

```
##            CP nsplit rel error   xerror       xstd
## 1 0.004149378      0 1.0000000 1.000000 0.06238359
## 2 0.002074689     20 0.8879668 1.174274 0.06721016
## 3 0.001383126     22 0.8838174 1.219917 0.06839912
## 4 0.001000000     25 0.8796680 1.244813 0.06903573
```

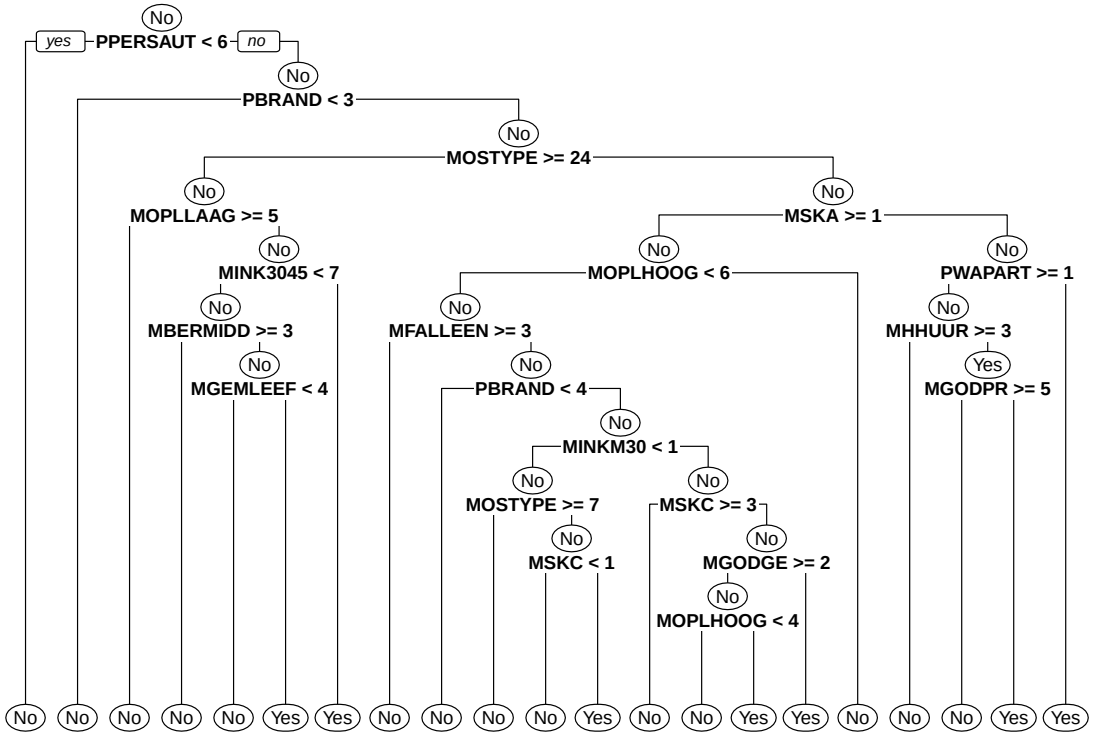
As we can see the lowest error is at cp 0.0041, but there the tree also only consist of the root node. So we take the second best which is a cp value of around 0.002 and 20 decision nodes.

part e)

We prune the tree with the new optimal complexity and plot it again.

```
tree_t1 <- prune(tree_t0, cp=0.002074689)

rpart.plot(tree_t1, extra = 0, under = TRUE, type = 2, box.palette = NULL, cex = 0.6)
```



As we can see, the tree is a bit less complex than our original one, e.g. it has a little fewer decision nodes.

part f)

We now predict the classes of the test data again and get the confusion matrix and balanced accuracy.

```r
t1_pred <- predict(tree_t1, newdata = data[test,], type = "class")

conf_matrix <- table(data[test,]$Purchase, t1_pred, dnn = c("Actual", "Predicted"))
print(conf_matrix)
```

```
##       Predicted
## Actual   No  Yes
##    No  1799   29
##    Yes  108    5
```

```r
recalls <- diag(prop.table(conf_matrix, 1))
balanced_accuracy <- mean(recalls)

cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.5142
```

As we can see the results have improved only very marginally but are still horrible.

g)

We now create class weights that assign each data point the weight that is 1 divided by the number of samples in its class. This assigns the observations from the minority class a higher weight and the ones from the majority class a lwoer one.

```r
class_counts <- table(data[train,]$Purchase)
class_weights <- ifelse(data[train,]$Purchase == "Yes",
                        1 / class_counts[2],
                        1 / class_counts[1])
class_counts
```

```
##
##   No  Yes
## 3640  241
```

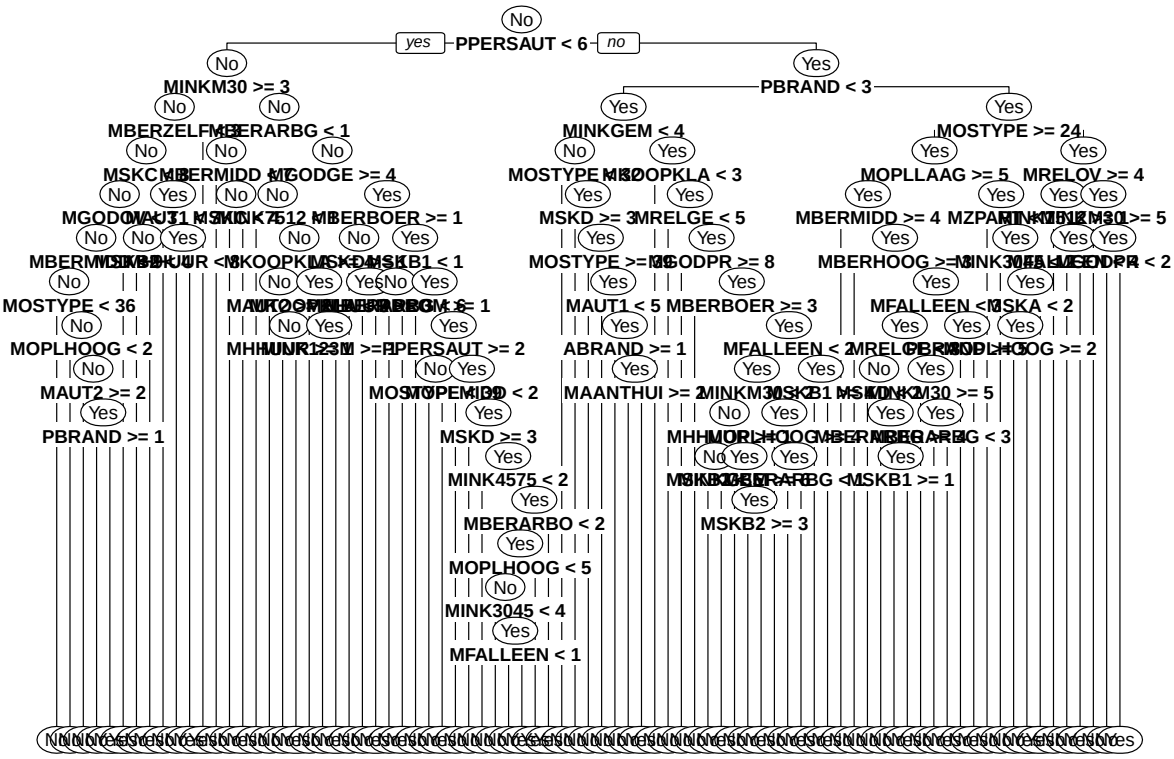
Now we construct the regression tree again and additionally use our new weights.

```r
tree_t2 <- rpart(Purchase~.,data=data[train,], xval=20, cp=0.002074689,
                 weights = class_weights)
```

We plot the tree again.

```r
rpart.plot(tree_t2, extra = 0, under = TRUE, type = 2, box.palette = NULL, cex = 0.6)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



As we can see the tree is now m uch more complex in comaprison to the 2 earlier ones. We now predict on the test data once again and compute confusion amtrix and balanced accuracy.

```
t2_pred <- predict(tree_t2, newdata = data[test,], type = "class")

conf_matrix <- table(data[test,]$Purchase, t2_pred, dnn = c("Actual", "Predicted"))
print(conf_matrix)
```

```
##       Predicted
## Actual   No  Yes
##    No  1384  444
##    Yes   61   52
```

```
recalls <- diag(prop.table(conf_matrix, 1))
balanced_accuracy <- mean(recalls)

cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.6086
```

As we can see the accuracy has improved significantly to around 0.6. This is still a very bad result though.

Task 2

part a)

We train a random forest model on our train data.

```
library(randomForest)
```

```
## randomForest 4.7-1.2
```

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

```
forest_1 <- randomForest(Purchase~.,data=data,subset=train)
```

Now we compute CM and BA again.

```
rf_pred1 <- predict(forest_1, data[test,])

conf_matrix <- table(data[test,]$Purchase, rf_pred1, dnn = c("Actual", "Predicted"))
print(conf_matrix)
```

```
##       Predicted
## Actual   No  Yes
##    No  1806   22
##    Yes   98   15
```

```
recalls <- diag(prop.table(conf_matrix, 1))
balanced_accuracy <- mean(recalls)

cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

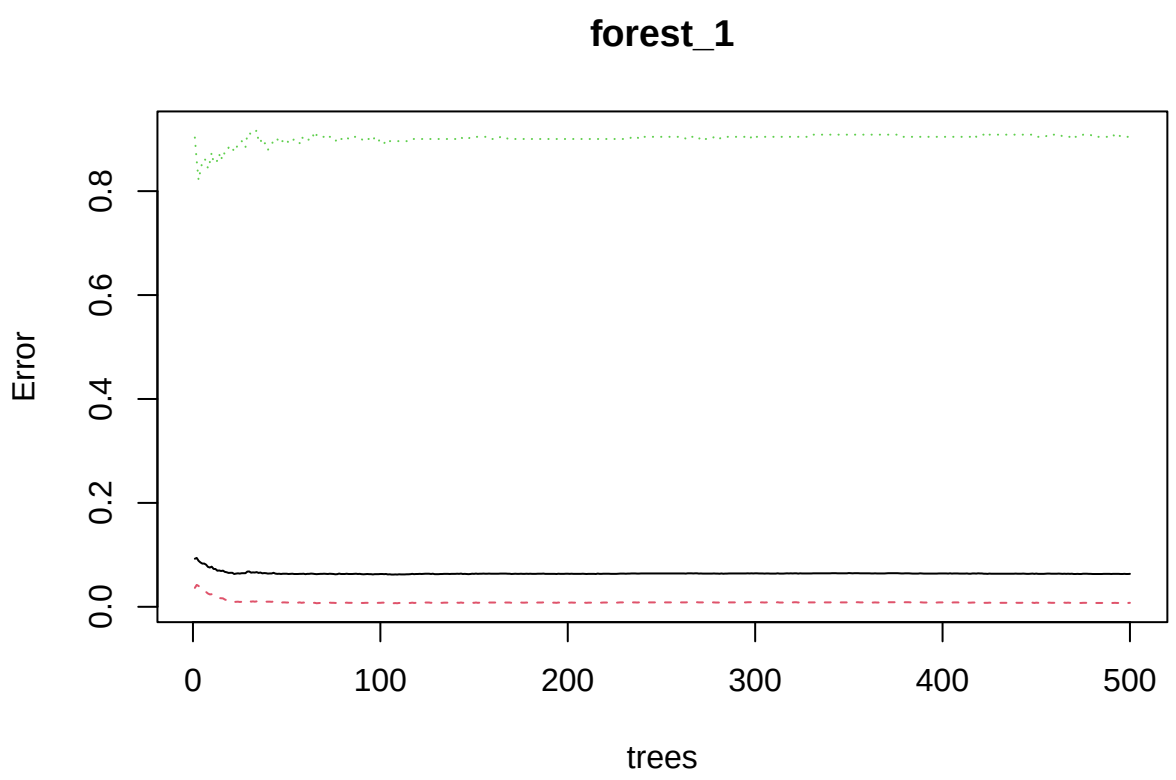
```
## Balanced Accuracy: 0.5604
```

As we can observe the BA is a bit better with a value of 0.57 than the non-optimized trees from Task 1 but still bad.

part b)

We plot the forest.

```
plot(forest_1)
```



We see the overall error as well as the individual error for each class for the numbers of trees in the forest. As we can see after at most around 50 trees the error seems to have converged and stabilized and stays roughly the same for an increasing number of trees after that.

part c)

We try different methods to improve the balanced accuracy. The biggest problem is the imbalanced class distribution, so we are trying several methods to counter that imbalance.\ First we modify the parameter sampsize. This parameter specifies how many samples should be used. By default it is using all the data, which means it is using way more data of the majority class than the other one. From Task 1 we know that in the minority class "Yes" there are 241 samples in the train data. We therefore set sampsize to 241 for both classes, so it uses all the samples from "Yes" and the same number from "No". This is similar to undersampling.

```
forest_2 <- randomForest(Purchase~.,data=data,subset=train, sampsize=c(241,241))
```

```
rf_pred2 <- predict(forest_2, data[test,])

conf_matrix <- table(data[test,]$Purchase, rf_pred2, dnn = c("Actual", "Predicted"))
print(conf_matrix)
```

```
##       Predicted
## Actual   No  Yes
##    No  1508  320
##    Yes   54   59
```

```r
recalls <- diag(prop.table(conf_matrix, 1))
balanced_accuracy <- mean(recalls)

cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.6735
```

As we can see the BA improved significantly up to 0.67, which is still not evry good but way better than before.Now we try to use the parameter classwt to improve BA. This parameter again takes weights for the data. We do the same approach as in task 1 by giving each observations the weight of 1 divided by the number of samples in its class. This gives the fewer "Yes" observations mroe weight.

```r
class_weights1 <- c(1 / table(data[train,]$Purchase)[1],
                    1 / table(data[train,]$Purchase)[2])
```

Now we train the forest again and compute CM and BA.

```r
forest_3 <- randomForest(Purchase~.,data=data,subset=train, classwt=class_weights1)
```

```r
rf_pred3 <- predict(forest_3, data[test,])

conf_matrix <- table(data[test,]$Purchase, rf_pred3, dnn = c("Actual", "Predicted"))
print(conf_matrix)
```

```
##       Predicted
## Actual   No  Yes
##    No  1793   35
##    Yes   91   22
```

```r
recalls <- diag(prop.table(conf_matrix, 1))
balanced_accuracy <- mean(recalls)

cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.5878
```

As we can see it also improves performance to 0.59 in comparison to the standard random forest, but not as much as the sampsize modifying. Now we try as a last step to modify the cutoff value, emaning at which value from 0 to 1 data would get calssified as Yes or No. By default this is of course 0.5, but we can increase it so everything for example below 0.9 will be classified as yes and only under 0.1 as no. After trying a bit we get the same resutls for 0.9/0.1 split. We now train the forest with this cutoff value and compute again CM and BA.

```r
forest_4 <- randomForest(Purchase~.,data=data,subset=train, cutoff=c(0.9,0.1))
```

```r
rf_pred4 <- predict(forest_4, data[test,])

conf_matrix <- table(data[test,]$Purchase, rf_pred4, dnn = c("Actual", "Predicted"))
print(conf_matrix)
```

```
##       Predicted
## Actual   No  Yes
##    No  1510  318
##    Yes   62   51
```

```
recalls <- diag(prop.table(conf_matrix, 1))
balanced_accuracy <- mean(recalls)

cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.6387
```

As we can see this also significantly improves BA compared to no optimization and also is better than with the class.weights, but still worse than the sampsize.

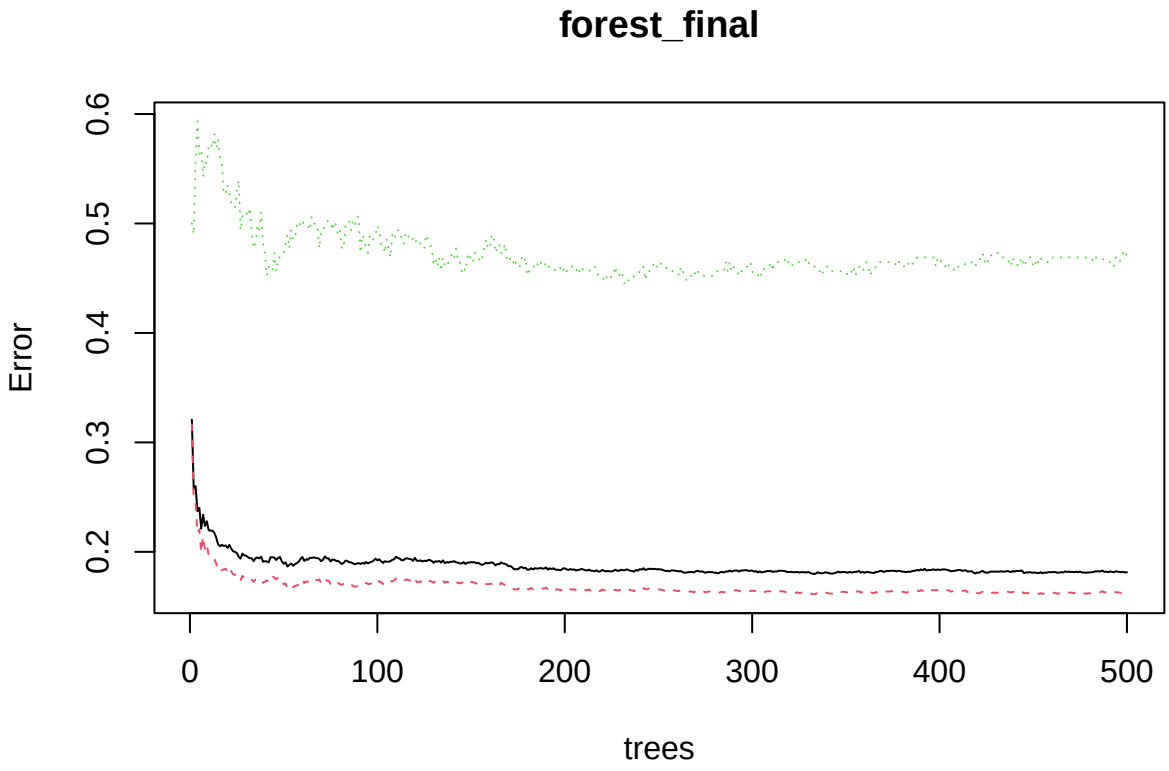
part d

We run the random Forest with the adjusted sampsize, since this obtained the best BA in part c. Combining the different methods form before also doesn't make sense since each is trying to counter the class imbalance and together they would just create a new one towards the other class.

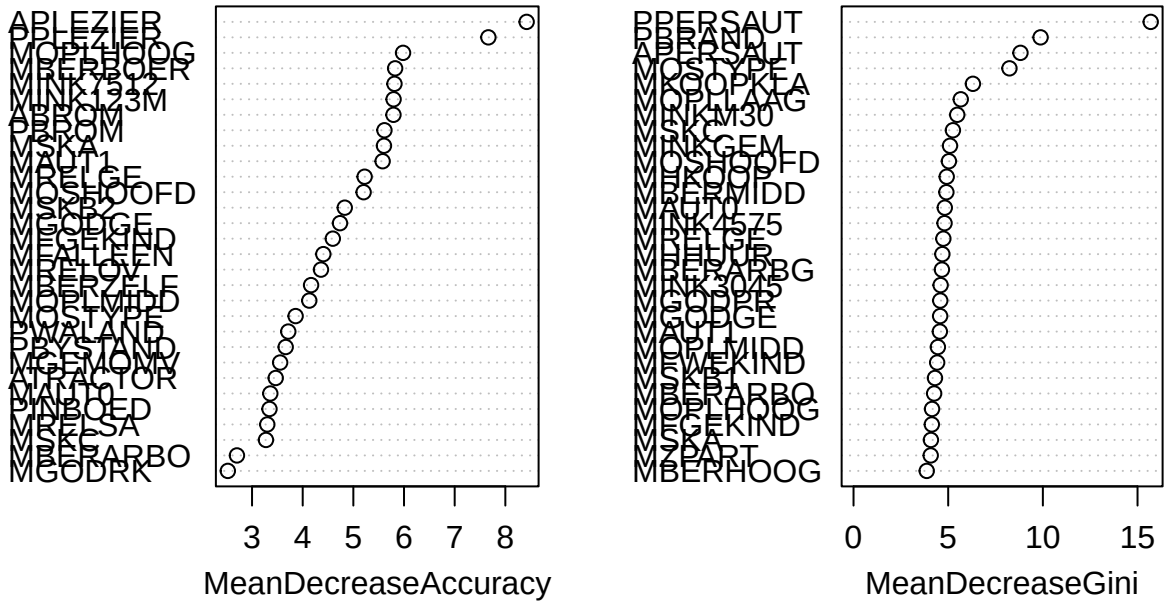
```
forest_final <- randomForest(Purchase~.,data=data,subset=train,
                             sampsize=c(241,241), importance=TRUE)
```

We plot again the error plot as well as the varaince importance plot.

```
plot(forest_final)
```




```
varImpPlot(forest_final)
```



As we can see from the error plot, this time the error for the "Yes" class stabilizes and converges way later than with the defat settings at around 250 trees. For the majority class and the overall error it stays similar with stabilizing at about 50. The importance tables show us the importance of the predictors. The first table is the Mean Decrease Accuracy, which means by how much the accuracy decreases when axcluding the specific predictor. It seems that APLEZIER and PPLEZIER will have the most decrease in accuracy when excluded which would make them important. The second table shows the Mean Decrease Gini which measures how nmuch a predictor contributes to reducing node impurity. Higher values would imply that the predictor helps splitting the data mroe efficiently. The highest values are PPERSAUT and PBRAND which indicates they are important as well.