Regression Trees Lab.

Universitat Politècnica de Catalunya

Andreu Meca, Geraldo Gariza and Jan Leyva

6 de May, 2021

Contents

```r
n <- nrow(mydataset)
p <- ncol(mydataset)

mydataset$overall_survival<- (mydataset$overall_survival)
```

1 Description data

The Molecular Taxonomy of Breast Cancer International Consortium (METABRIC) database is a Canada-UK Project which contains targeted sequencing data of 1,980 primary breast cancer samples. Clinical and genomic data was downloaded from cBioPortal. The dataset was collected by Professor Carlos Caldas from Cambridge Research Institute and Professor Sam Aparicio from the British Columbia Cancer Centre in Canada and published on Nature Communications (Pereira et al., 2016)..

Metabric dataset integrates three types of data. Clinical data are in columns 1 to 31, gene expression data are in columns 32 to 520, mutation data are in columns 521 to 693. Gene expression values are normalized to be z-score.

The data set has **1904** observations on **693** variables.

The aim of this study is to predict the `overall_survival` that specify if the patient is alive or dead. This will be done by three models (tree, a random forest and gradient boosting in the packege `adaboost`).

- Variability

```r
sort(apply(mydataset[,32:520] %>% drop_na(), 2, sd), decreasing = TRUE)[1:6]
```

```
##    nfkb1    ptprm   cyp3a7     e2f1   pdgfra     ubr5
## 1.000265 1.000265 1.000265 1.000264 1.000264 1.000264
```

The top 3 variables with the most variability have 1.000265, which is just a prove that the variables are normalized to be z-score

- Percentage missing values

```r
n_miss <- c()
for(i in 1:ncol(mydataset)){
  n_miss[i]<-length(which(is.na(mydataset[,i])))
}
knitr::kable(t(n_miss[which(n_miss != 0)]), col.names = which(n_miss != 0))
```

3	5	6	10	12	15	19	21	23	28	29	30	31	679	689	691	693
22	15	54	30	72	15	106	45	15	204	20	501	1	7	2	2	1

Most of the missing values are on the first columns of the dataset, from 1 to 32. On the figure 1 is possible see the missing percentage for the 32 first columns.

```r
cat("The percentage of NA is:", round((length(which(n_miss != 0))/ncol(mydataset)*100),2),"%")
```

```
## The percentage of NA is: 2.45 %
```

```
vis_miss(mydataset[,1:32])
```

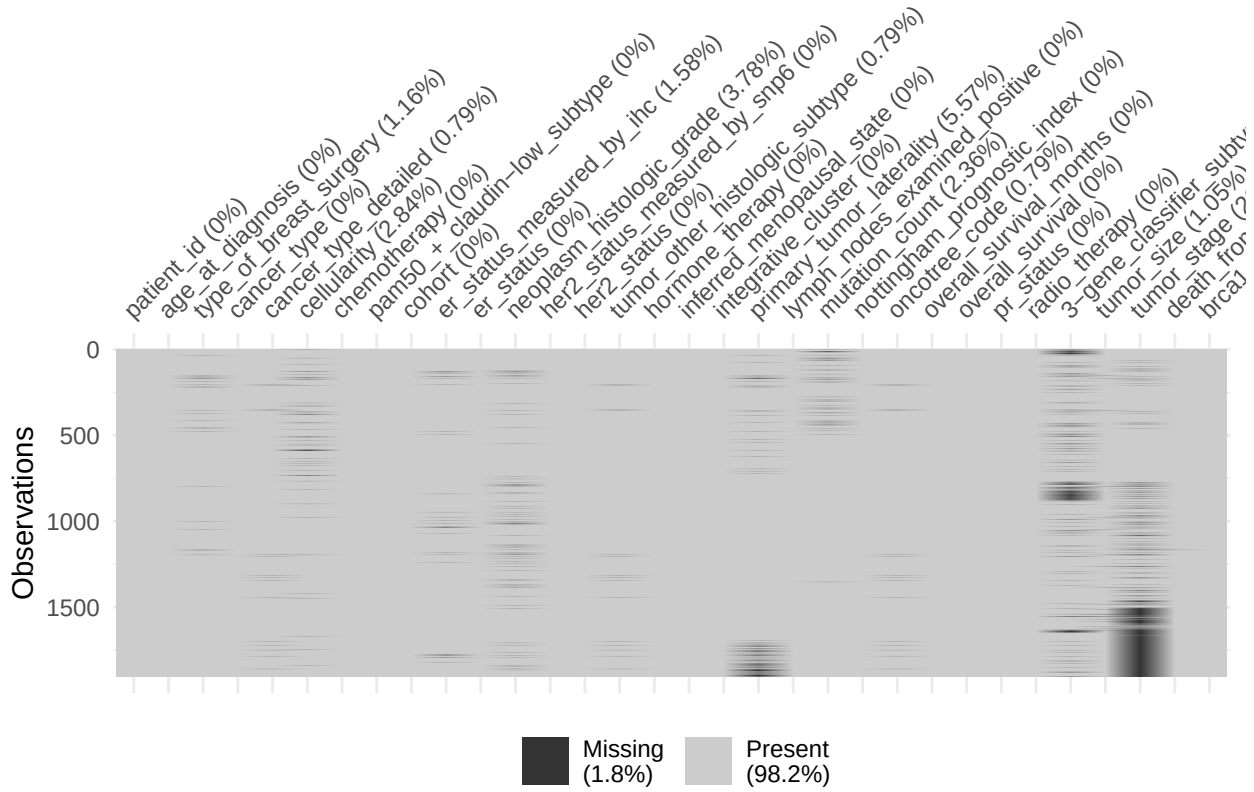


Figure 1: Missing values Col 1 to 32

2 Preprocess

In this model is not needed a data preprocess.

3 Partition of data

In order to properly evaluate the performance of a model, we must estimate the error rather than simply computing the training error. We split the observations into a training set and a test set, build the model using the training set, and evaluate its performance on the test data.

```
set.seed(params$seed)
data <-mydataset[, 32:520]
data$overall_survival <- (mydataset$overall_survival)
# trainIndex <- createDataPartition(data$brca1, p = params$partition, list = FALSE, times = 1)
```

```
set.seed(params$seed)
data_split <- initial_split(data, prop = params$partition)
train <- training(data_split)
test  <- testing(data_split)
```

The train set has 1270 observations and the test set has 634.

3. Fit a pruned single tree model to predict the overall_survival. Assess the performance of the tree by using suitable metrics. overall_survival is a target variable whether the patient is alive or dead. Additionally, you can adjust another model that adds some clinical variable that you consider relevant to the prediction.

```
fit <- rpart(overall_survival~., method="class", data=train)
fit_p <- prune(fit, cp = fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])
res_fit <- predict(fit_p, newdata = test, type = "class")
```

The package rpart fits a simple classification trees, which is stored in the variable fit and then it is pruned to not fall in an overfitting context. The prune is done minimizing the error given by the xerror parameter, computed with the cross-validation implementation.

Then we predict using the test set and store it in the res_fit object.

Now, we compute the confusion matrix and the model accuracy.

```
confusion_matrix(test$overall_survival, as.numeric(as.character(res_fit)))
```

	Actual_0	Actual_1
Predicted_0	273	89
Predicted_1	162	110

```
accuracy(test$overall_survival, as.numeric(as.character(res_fit)))
```

```
## Accuracy: 0.6041009
```

4. Fit a Random Forest (RF) classifier to predict the overall_survival. Tune the pa- rameters: number of trees and number of variables per node, by implementing a grid search procedure. Assess the performance of RF using suitable metrics. Determine which variables are the most relevant in the overall_survival prediction.

```
bag.survival=randomForest(x=select(train, -overall_survival), y=as.factor(train$overall_survival),data=

bag.survival
```

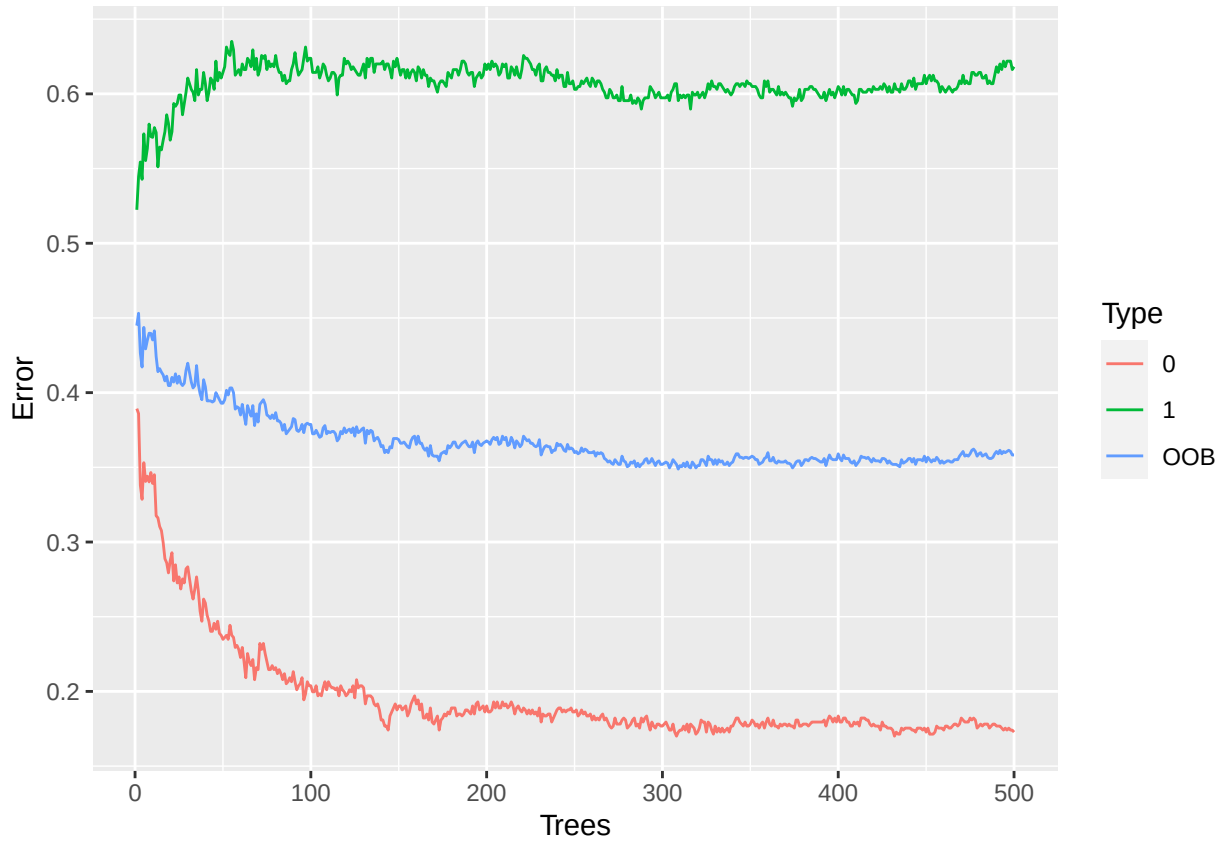
```
##
## Call:
##  randomForest(x = select(train, -overall_survival), y = as.factor(train$overall_survival),      impo
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 22
```

```
##
##         OOB estimate of  error rate: 35.83%
## Confusion matrix:
##     0   1 class.error
## 0 613 128   0.1727395
## 1 327 202   0.6181474
```

```
OBB.error<-bag.survival$err.rate[nrow(bag.survival$err.rate),1]
```

We see the error rate is 0.3582677 with the number of trees 500 and the nº of variables tried at each split = 22. We can plot the forest in order to see in that number of trees 500 is the optimum.

```
#We use the matrix of errors to make a visualization
ntrees<-nrow(bag.survival$err.rate)

oob.error.data<- data.frame(
  Trees=rep(1:ntrees, times=3),
  Type= rep(c("OOB", "0", "1"), each=ntrees),
  Error=c(bag.survival$err.rate[,"OOB"],
          bag.survival$err.rate[,"0"],
          bag.survival$err.rate[,"1"]))


ggplot(data=oob.error.data, aes(x=Trees, y=Error)) +
  geom_line(aes(color=Type))
```



Now, let's try if with a higher number of trees we obtain a better performance:

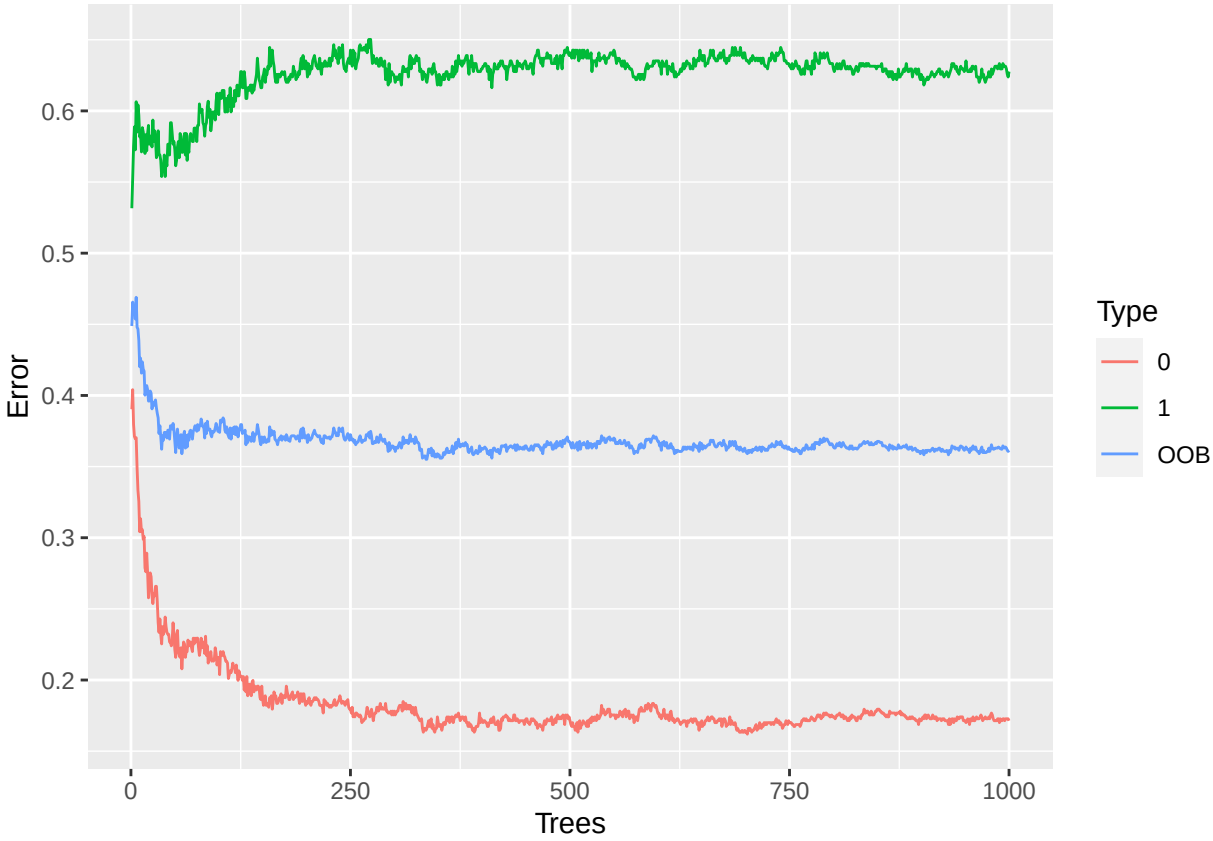
```
bag.survival2=randomForest(x=select(train, -overall_survival), y=as.factor(train$overall_survival),data=

bag.survival2
```

```
##
## Call:
##  randomForest(x = select(train, -overall_survival), y = as.factor(train$overall_survival),      ntree
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 22
##
##         OOB estimate of  error rate: 36.14%
## Confusion matrix:
##     0   1 class.error
## 0 614 127   0.1713900
## 1 332 197   0.6275992
```

```
OBB.error2<-bag.survival2$err.rate[nrow(bag.survival$err.rate),1]
```

We see the error rate is 0.3669291 with the number of trees 1000 and the nº of variables tried at each split = 22. We cannot guarantee a higher performance, thus we remain with 500.

```
ntrees2<-nrow(bag.survival2$err.rate)

oob.error.data2<- data.frame(
  Trees=rep(1:ntrees2, times=3),
  Type= rep(c("OOB", "0", "1"), each=ntrees2),
  Error=c(bag.survival2$err.rate[,"OOB"],
          bag.survival2$err.rate[,"0"],
          bag.survival2$err.rate[,"1"]))


ggplot(data=oob.error.data2, aes(x=Trees, y=Error)) +
  geom_line(aes(color=Type))
```



The plot represents stability with more than 500 trees.

Now, let's try to change the nº of variables tried each split:

```
set.seed(1234)
oob.values <- vector(length=25)
for(i in 20:25) {
  temp.model <- randomForest(x=select(train, -overall_survival), y=as.factor(train$overall_survival),da

  oob.values[i] <- temp.model$err.rate[nrow(temp.model$err.rate),1]
}
```

We observe the minimum at position 24º therefore, we use the model as default:

```
oob.values
```

```
##  [1] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [8] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [15] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.3598425 0.3787402
## [22] 0.3653543 0.3566929 0.3614173 0.3637795
```

```
## find the minimum error
oob.values<-1-oob.values

## find the optimal value for mtry...
which(oob.values == min(oob.values))
```

```
## [1] 21
```

```
## create a model for proximities using the best value for mtry
model <- randomForest(x=select(train, -overall_survival), y=as.factor(data$overall_survival),data=train
                      ,mtry=which(oob.values == min(oob.values)))
```

```
## Error in randomForest.default(x = select(train, -overall_survival), y = as.factor(data$overall_survi
```

```
oob.values
```

```
##  [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [8] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6401575 0.6212598
## [22] 0.6346457 0.6433071 0.6385827 0.6362205
```

```
## find the minimum error
oob.values<-1-oob.values

## find the optimal value for mtry...
which(oob.values == min(oob.values))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```

```
## create a model for proximities using the best value for mtry
model <- randomForest(x=select(train, -overall_survival), y=as.factor(train$overall_survival),data=trai
                      ,mtry=which(oob.values == min(oob.values)))
model
```

```
##
## Call:
##  randomForest(x = select(train, -overall_survival), y = as.factor(train$overall_survival),      ntre
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 37.64%
## Confusion matrix:
##     0   1 class.error
## 0 657  84   0.1133603
## 1 394 135   0.7448015
```

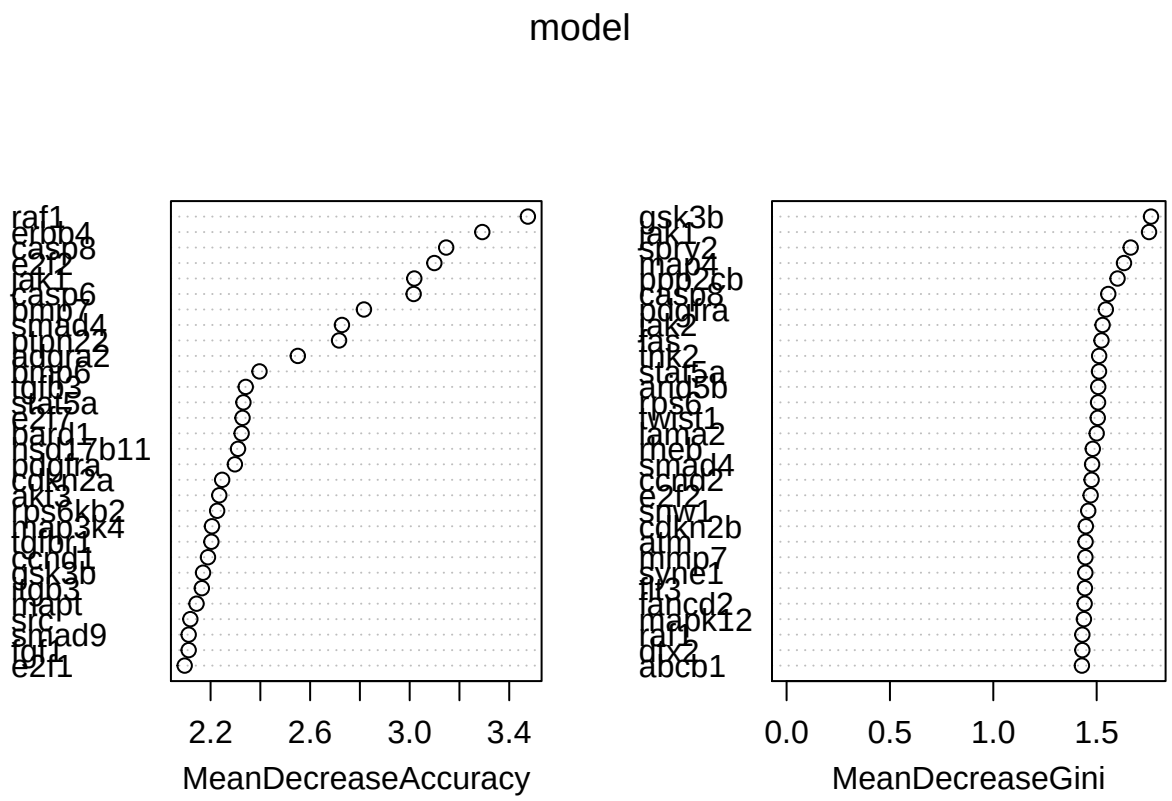
We are going to calculate the confusion matrix for the test sample:

```
yhat.bag = predict(model,newdata=test)



conf.matrix.rf<-confusionMatrix(as.factor(yhat.bag),
                                as.factor(test$overall_survival))


conf.matrix.rf$table; conf.matrix.rf$overall[1]
```

```
##           Reference
## Prediction   0   1
##          0 336 219
##          1  26  53

##  Accuracy
## 0.6135647
```

We use the varImplot function to see the importance of the variables:

```
varImpPlot(model)
```



5. Apply the gradient boosting algorithm with adaboost specification:

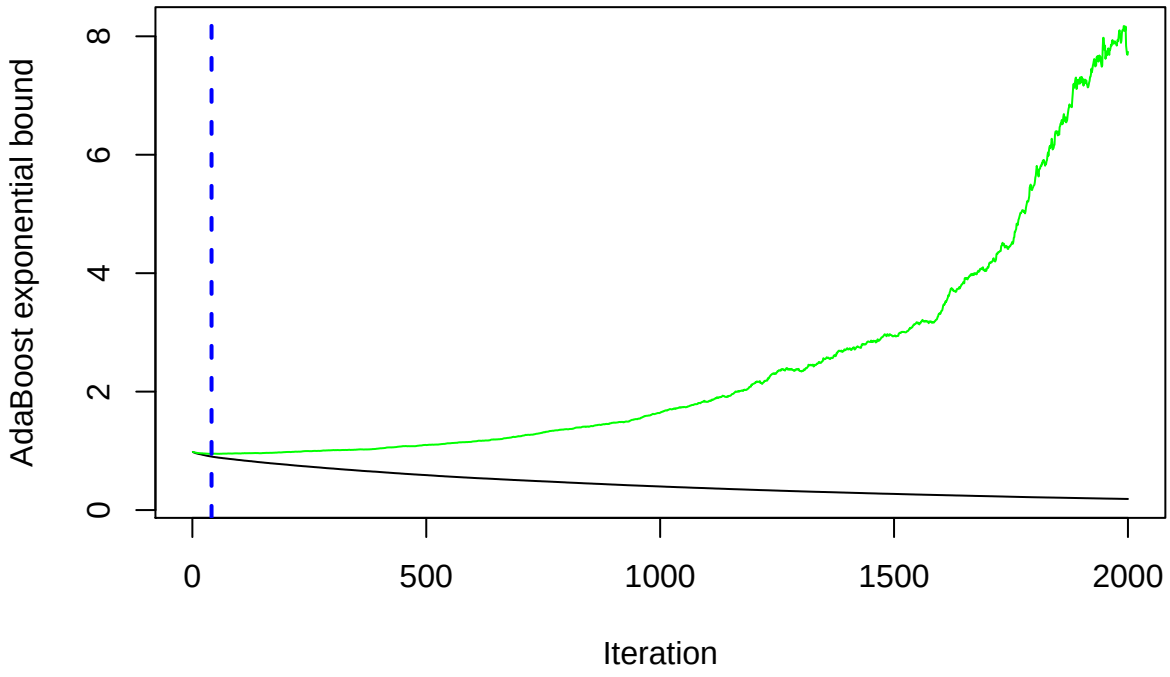
5.1. Using stumps as classification trees for `overall_survival` prediction, compute the misclassification rates of both the learning set and the test set across 2,000 iterations of gbm. Represent graphically the error as a function of the number of boosting iterations.

```
boost_model=gbm((overall_survival)~.,
                data = train,
                distribution="adaboost",
                n.trees=2000, interaction.depth=1,
                cv.folds = 3)
```

The package `gbm` have the distribution `adaboost` that perform the adaboost algorithm. In this case is a model with a 2000 trees of one deep each one (stumps). Also, it is used a cv.folds = 3.

- Represent graphically the error as a function of the number of boosting iterations.

```
(perf<-gbm.perf(boost_model))
```




```
## [1] 41
```

The gbm.perf return a plot where is specify the number of iterations need it in order to get a good performance. In this case the number of iterations for got a similar results that done with 2000 trees is 41. It means that if we perform again the `adaboost` but this time with `n.trees=` 41 we should get a similar results that the obtaineds with 2000.

```
boost_predict_train=predict(boost_model,newdata=train,
                            n.trees=2000, type = "response")
conf_matr_train<-confusionMatrix(as.factor(round(boost_predict_train)),
                 as.factor(train$overall_survival))
conf_matr_train$table; conf_matr_train$overall[1]
```

```
##            Reference
## Prediction   0   1
##          0 741   0
##          1   0 529
```

```
## Accuracy
##        1
```

As expected the performance with the same data that the algorithm is trained the performance is perfect with a 0 error rate. This is one of the problems of adaboost that tends to overfit the data, for this reason is used a split data in train and test.

```
boost_predict=predict(boost_model,newdata=test,
                          n.trees=2000, type = "response")
conf_matr<-confusionMatrix(as.factor(round(boost_predict)),
                as.factor(test$overall_survival))
conf_matr$table; conf_matr$overall[1]
```

```
##           Reference
## Prediction   0   1
##          0 262 142
##          1 100 130
```

```
##  Accuracy
## 0.6182965
```

The performance when is predicted the test data set is 0.6182965.

4 Prediction

5.2. Compare the test-set misclassification rates attained by different ensemble classifiers based on trees with maximum depth: stumps, 4-node trees, 8-node trees, and 16-node trees.

```
boost_model_4=gbm(overall_survival~.,data= train,
                  distribution="adaboost", n.trees=2000,
                  interaction.depth=4, cv.folds = 3)
boost_predict_4 =predict(boost_model_4,newdata=test,
                          n.trees=2000, type = "response")
conf_matr_4 <- confusionMatrix(as.factor(round(boost_predict_4)),
                as.factor(test$overall_survival))
conf_matr_4$table; conf_matr_4$overall[1]
```

```
##           Reference
## Prediction   0   1
##          0 268 143
##          1  94 129
```

```
## Accuracy
## 0.626183
```

```
mean(round(boost_predict_4) != test$overall_survival)
```

```
## [1] 0.373817
```

```
boost_model_8=gbm(train$overall_survival~.,data= (train), distribution="adaboost", n.trees=2000, intera
boost_predict_8 =predict(boost_model_8,newdata=test,
                          n.trees=2000, type = "response")
conf_matr_8 <- confusionMatrix(as.factor(round(boost_predict_8)),
                as.factor(test$overall_survival))
conf_matr_8$table; conf_matr_8$overall[1]
```

```
##           Reference
## Prediction   0   1
##          0 280 144
##          1  82 128
```

```
##  Accuracy
## 0.6435331
```

```
mean(round(boost_predict_8) != test$overall_survival)
```

```
## [1] 0.3564669
```

```
#summary(boost_model)
```

```
boost_model_16=gbm(train$overall_survival~.,data= train, distribution="adaboost", n.trees=2000, interac
boost_predict_16 =predict(boost_model_16,newdata=test,
                          n.trees=2000, type = "response")
conf_matr_16 <- confusionMatrix(as.factor(round(boost_predict_16)),
                as.factor(test$overall_survival))
conf_matr_16$table; conf_matr_16$overall[1]
```

```
##           Reference
## Prediction   0   1
##          0 281 147
##          1  81 125
```

```
##  Accuracy
## 0.6403785
```

```
mean(round(boost_predict_16) != test$overall_survival)
```

```
## [1] 0.3596215
```

Summary table

```
kable(matrix(c("Stump", "4-node trees", "8-node trees", "16-node trees",conf_matr$overall[1],conf_matr_
```

Deep	Accuracy
Stump	0.618296529968454
4-node trees	0.626182965299685
8-node trees	0.643533123028391
16-node trees	0.640378548895899

As we can see the difference between use stump or 16 node trees is not to much. For this could be because with one stump we got a overfitting to train dataset and is not need to increase the deep of nodes.