TITANIC PROJECT- PART 2

Pablo Simón Martín Sánchez & Álvaro Martín Ruiz

Introduction: This report is divided in 2 parts, which correspond to the first and second task of the assignment. In the first task, we will analyze a decision tree as a complementary part of the exploratory data analysis we carry out on the first assignment. In the second part, we will look for the decision tree and random forest that best fits with our data.

TASK 1

PREPROCESSING

Before we create a decision tree with the data provided, we have to 'clean' the data set:

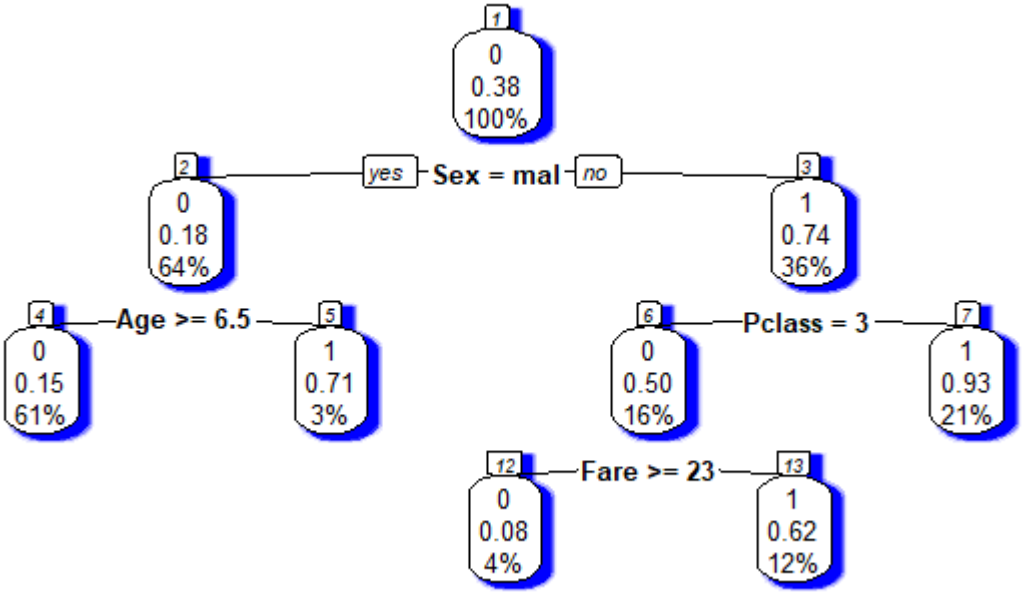
After the analysis we perform in the first assignment, we come to the conclusion that the variables Ticket and Cabin don't bring us any significant information.

So what we do is just remove these two variables from the data set. Then we also replace the 0 values in the variable 'Fare' (we assume that there is a 0 because they are missing values) by the mean, so that the values don't disrupt the information very much.

We didn't scale the data because neither classification trees nor random forests (which we will use later) work with euclidean distances or any type of metric that would be affected if we don't scale our data.

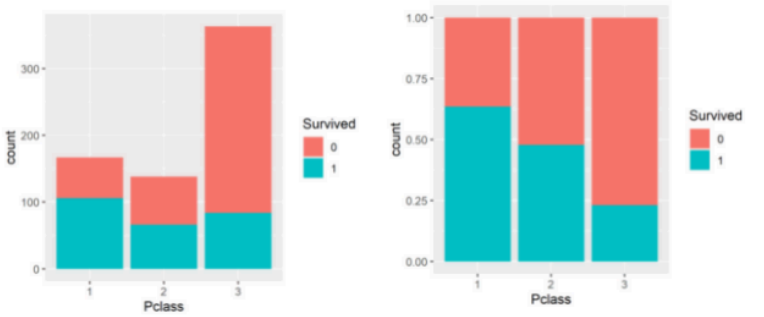
CLASSIFICATION TREE

The classification tree we get (without looking for the best hyperparameters, which is what we will do later) is as follows:



The variables that are closer to the main node are the most important variables, therefore (and as we saw in the first assignment) we can observed that surviving is strongly affected by the sex (0 = died, 1 = survived). Also, among male sex, kids (<= 6.5 years old) were the one that had a higher survival rate. This matches perfectly with the 'Women and Children first' conclusion we draw in the first part of the project.

If we move to the left part of the graph we see that women of a higher class were more likely to survive. This matches perfectly with the prediction we did on the first part of the assignment, where we saw that people travelling in first class had a higher survival rate. Below, there is the part of the first assignment were we predicted this:



It seems like people of a higher social-class had some preference when leaving the boat.

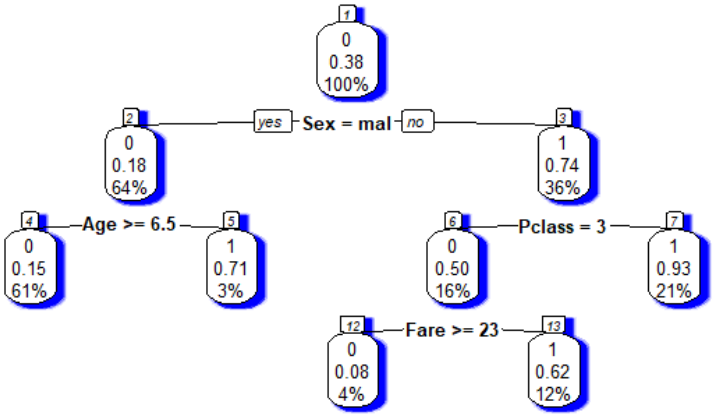
The variables that do not appear are variables that are meant to be less important (port of embarkation, and the number of sons or parents aboard the boat).

If we use an output argument called "variable.importance", we can see the importance of each variable in the decision tree.

```
> mytree$variable.importance
      Sex      Fare    Pclass       Age     Parch     SibSp  Embarked
99.892879 32.272492 23.040815 16.207753 15.227819  7.193825  4.608163
```

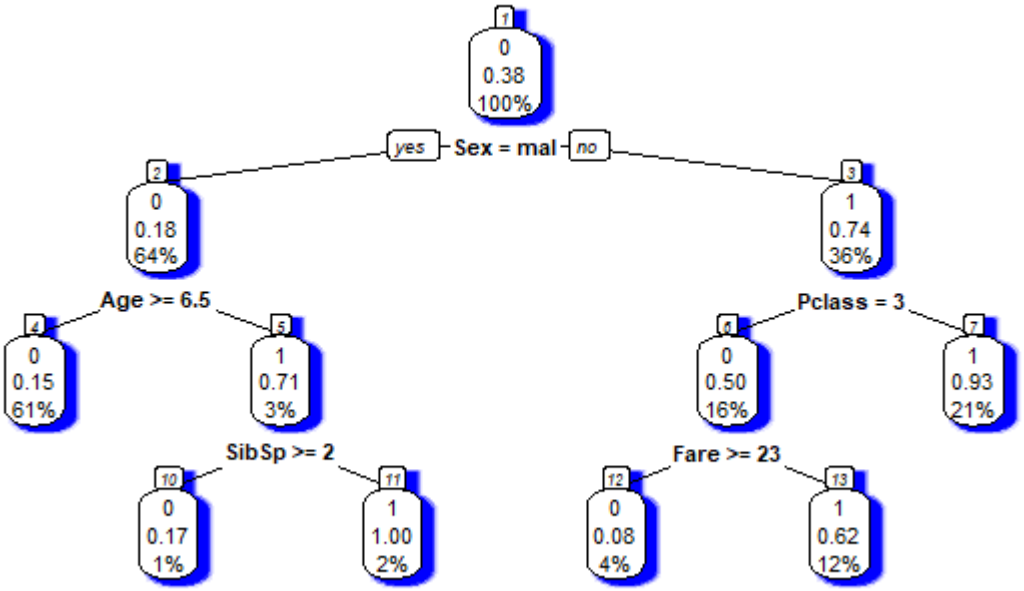
The results confirm what he have already seen, which is that the most important variable (by far) when predicting whether a person survived or not is the sex. Also, the social class (represented in Fare and Pclass) and the age of each person played an important role.

To see if maybe the variables 'SibSp' and 'Parch' may be confusing to the decision tree, we are going to replace them to one single variable which determines if people travelled alone. If we do the decision tree again, we can see that we get exactly the same decision tree as before. So, as we also predict in the first part of the project, knowing if a passenger traveled or not does not give us much information to determine whether he/she survived.



```
> mytree2$variable.importance
          Sex          Fare        Pclass           Age travels_alone
    99.892879     32.272492     23.040815     16.207753      5.755145
     Embarked
     4.608163
```

Finally, if we plot a decision tree again, but this time with the the best combination of hyperparameters (in the next part we show how to find the best combination), we can see that the results obtained are similar to the ones obtained previously.




```
> my_best_tree$variable.importance
      Sex      Fare    Pclass       Age     Parch     SibSp  Embarked
99.892879 34.968570 24.838200 16.207753 15.227819 12.585982  6.405549
```

TASK 2

In this part of the project we are going to do a model for predicting variable “survived” as a function of the rest of variables. To do so, we are going to do a classification tree and a random forest and see which one has the higher accuracy predicting survival. In order to find the best classification tree and random forest possible, we will try different combinations of the hyperparameters, and check their accuracy with some of the methods we have seen this cours (we will use k-fold cross validation for the decision tree and random subsampling for the random forest).

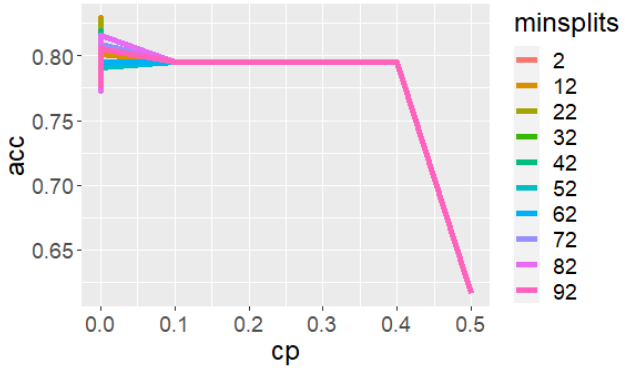
DECISION TREE

In order to find the best decision tree possible, we created loops to check multiple combinations of hyperparameters and store their results. Then, once the loop finished, we looked for the combination that gave us better results. Nevertheless, there are millions of combinations of the hyperparameters, so what we did was start by checking combinations of hyperparameters in a wide range, and then progressively reduce it until we got the best-possible combination.

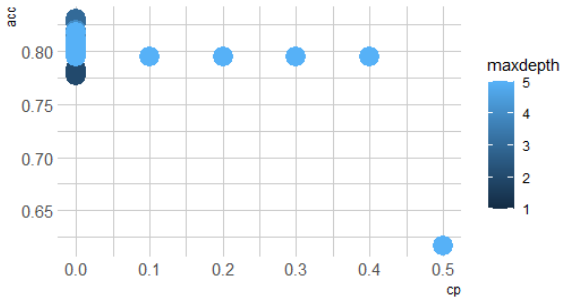
The first combination of hyperparameters we tried was:

```
minsplits = seq(2,100,5)
cp =  seq(0, 0.5, 0.1)
maxdepth = seq(1,5)
```

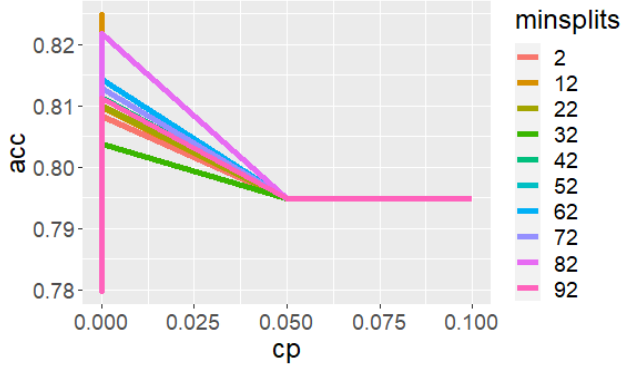
Which gave us the following graphic and the following best-combination of hyperparameters:




```
    minsplits cp maxdepth       acc
122        12  0        3 0.827874
```



After this, we see that we should fence in the range of the cp between 0 and 0.1 by 0.05, and we get the following results:

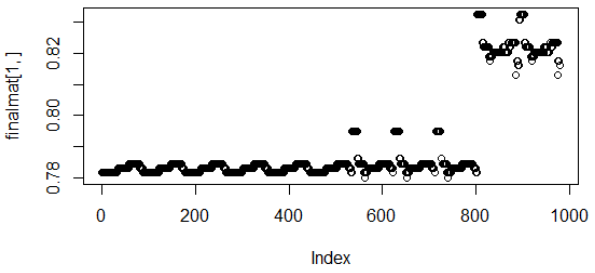


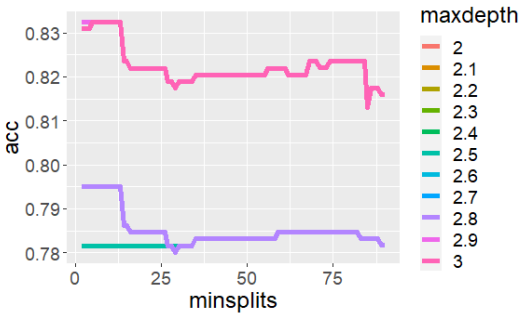

```
   minsplits cp maxdepth       acc
62        12  0        3 0.824845
```

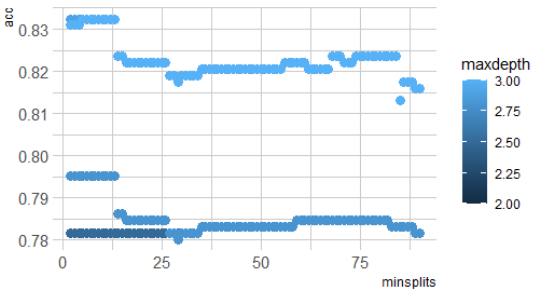
Here we observed that the optimum value for the complexity parameter is 0

So we now continue with a set value 0 for the cp, and values for minsplit and maxdepth in a tighter range every time we check the results of the combinations.

	minsplits	cp	maxdepth	acc
802	2	0	2.9	0.8324201
803	3	0	2.9	0.8324201
804	4	0	2.9	0.8324201
805	5	0	2.9	0.8324201
806	6	0	2.9	0.8324201
807	7	0	2.9	0.8324201
808	8	0	2.9	0.8324201
809	9	0	2.9	0.8324201
810	10	0	2.9	0.8324201
811	11	0	2.9	0.8324201
812	12	0	2.9	0.8324201
813	13	0	2.9	0.8324201
894	5	0	3.0	0.8324201
895	6	0	3.0	0.8324201
896	7	0	3.0	0.8324201
897	8	0	3.0	0.8324201
898	9	0	3.0	0.8324201
899	10	0	3.0	0.8324201
900	11	0	3.0	0.8324201



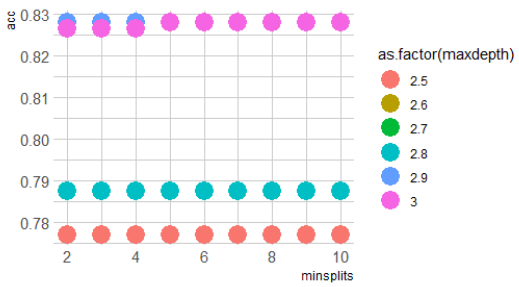




In the screenshot of the matrix where we have stored our results (which is represented on the top right chart above), we observe that the combinations that gave us the better accuracies were the ones in the positions 800 to 1000 of the matrix (approximately).

Then, In the right middle and right bottom graphs, we see that the best results are obtained when the minsplit is between 2 and 10, and the maxdepth around 2.9

We continued with our process to find the optimum values:

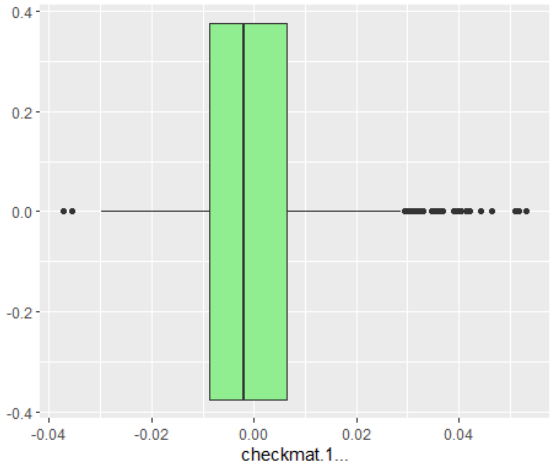


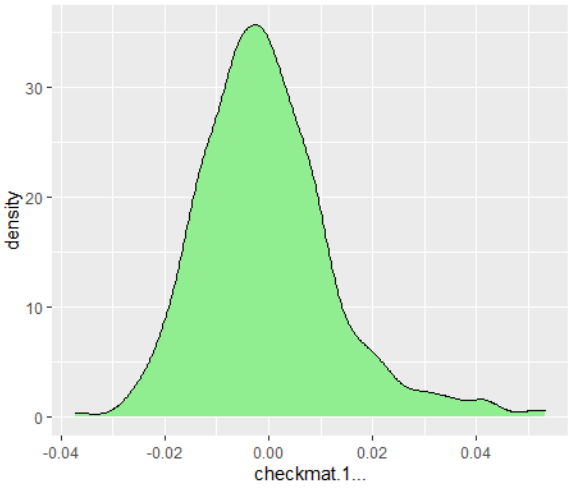
Finally, after repeating this process multiple times, we reached a very reduced and specific range of combinations, and the combination that gave us the best result is: minsplit = 2, cp= 0, maxdepth = 2.85. This model has an average accuracy of 0.823 approximately.

Repeated k-fold or repeated random subsampling ?

Classification trees and random forests' accuracy can be computed using k-fold (cross validation) and random subsampling. Both are methods used to divide the data in training and test set.
In cross validation, a number k of folds is created, each storing a different part of the data. One of these folds is used as the test set and the others are used as the training set. In repeated k-fold, you use all the different folds as the test set, getting the final accuracy of your model as the mean of all the accuracy.
On the other hand, random subsampling works by splitting the data into training and test set randomly. In the case of repeated random subsampling, you split the data an n number of times, without taking into account the previous splits and get the final accuracy of the model the same way as with k-fold .
Just by common sense, we can get to the conclusion that, in both cases, repeating the process different times gives you a more "accurate" accuracy
During the previous part of the report, we used repeated k-fold to check the accuracy of the different classification trees. But, would it have been better to do it using repeated random subsampling instead?
To check this, we will repeat the process performed previously but using random subsampling instead of k-validation. We will check for 1100 combinations of the same hyperparameters and look how the results differ by subtracting the matrix where the information for random subsampling is stored (finalmat2) from the matrix of k-validation(finalmat). As a first approach, we can compute the mean accuracy,precision and specificity got using each method and see that they are very similar

```
> rowMeans(finalmat)
[1] 0.7985190 0.8115455 0.7902958
> rowMeans(finalmat2)
[1] 0.7990217 0.8102212 0.7930204
```



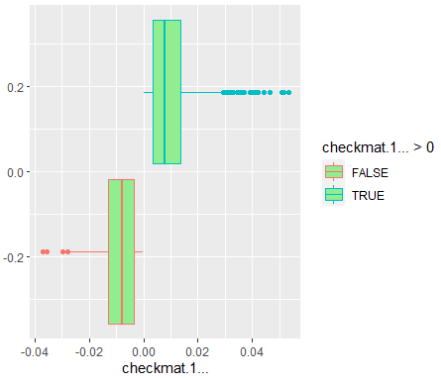



```
max(abs(checkmat[1,]))
] 0.05325112
```

```
sum(checkmat[1,] > 0)
] 474
```

```
sum(checkmat[1,] < 0)
] 626
```

From the new matrix created we can extract different information. Firstly, we see that the accuracy obtained for each specific parameter does not differ a lot. In fact, the maximum difference is just 0.05, so most values are inside a small interval. We also get that the accuracy obtained with random subsampling is greater in 626 combinations. However, as it can be observed in the boxplot and in the density graph, most extreme differences of accuracies were in favor of k-fold . That means, when k-fold gives a better accuracy than random subsampling.



These other two boxplots show how the difference is distributed depending on if the accuracy is bigger in k-folds (top one) or in random subsampling. As we stated before, the more extreme differences appear when k-fold is better.
As a last conclusion, we draw that there is not much difference between using one method or the other to compute the accuracy of our models.

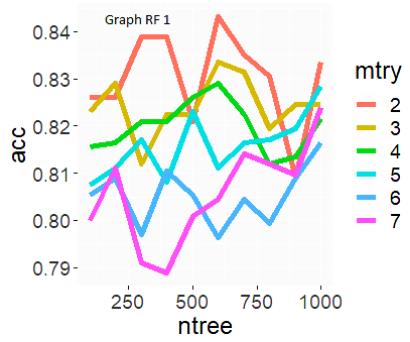
Random Forest

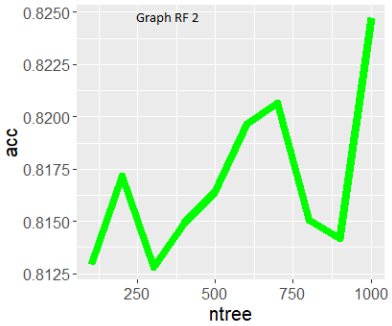
Once we had finished our deep look on decision trees and the most accurate hyperparameters that could be applied, we started looking for the best random forest. Random forest, explained roughly, is a “forest” of decision trees. The final output of the random forest (its prediction) corresponds to the prediction given by most of its trees. In order for this method to be effective, all trees must be different.

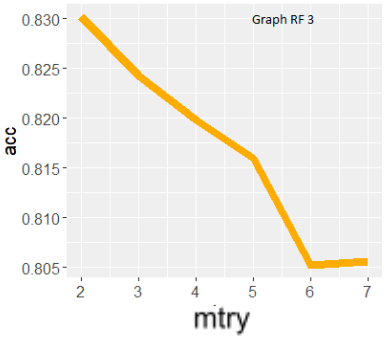
The randomForest() function inside the “randomForest” library creates the models using some hyperparameters that are set with default values if you don’t mess with them. We will try to tinker with two different hyperparameters in order to get the best possible random forest.
One of these two is “ntree”, which determines the number or trees created. With the ntree value, you don’t want to fall short, because that would mean less accuracy, but at the same time you don't want either to exceed badly the best value, because there is a point where, due to random forest don’t overfitting, creating more trees doesn’t change the output of the model and the model only consumes more memory. The other value we will try to fiddle with is “mtry”, which determines the number of variables randomly selected at every split made in each specific tree.
The default value for ntree is always 500, while the default value of mtry depends on the type of model performer(the square root of the number of variables for classification models and the number of variables divided by 3 for regression models).

To look for the best hyperparameters we use a loop in which we try all different combinations of hyperparameters(stored in a data frame using the function expand.grid()). For the first try, we give ntree values between 100 and 1000 with step 100 and mtry all possible values(from 2 to 7 by 1)

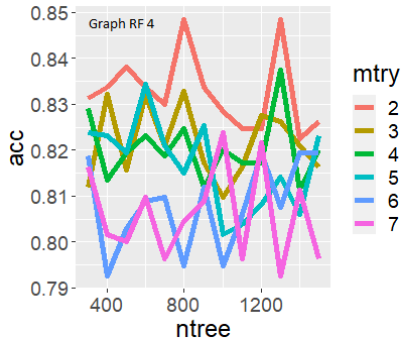
	mtry	ntree	acc
31	2	600	0.8432836

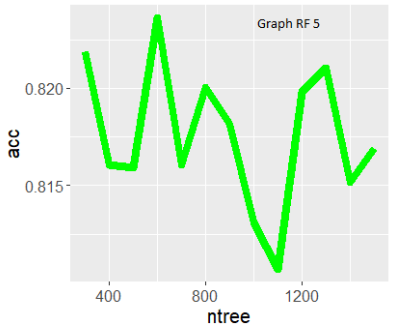


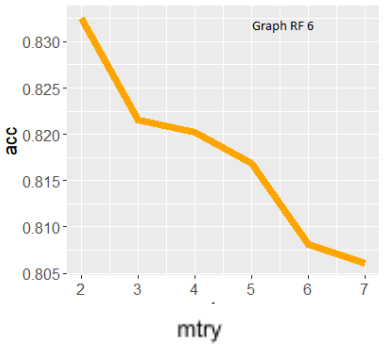




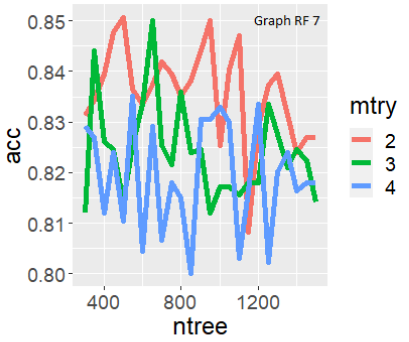
In this first trial, our best random forest gets a 0.843 accuracy. Exploring the parameters data frame with different graphs, we start to guess that the mtry should be as small as possible, as we can see in graph RF 3, which shows the mean accuracy for the different values of mtry. Moreover, graph RF 2(similar to graph RF 2 but with ntree instead of mtry) shows the mean accuracy increases from 900 to 1000, so we want to check if this trend keeps going over 1000 trees

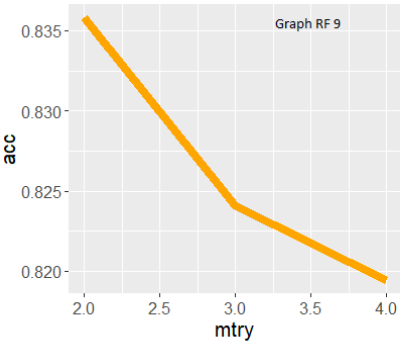






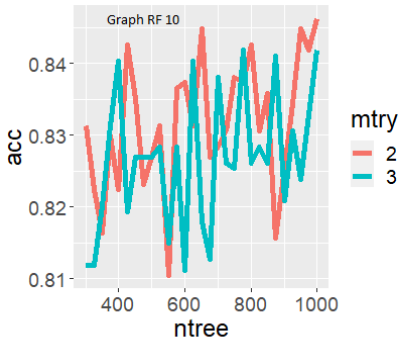
For the second trial, we expand the ntree range until 1500(starting from 300 instead of 100) and keep mtry values the same. The results indicate that after 1000 trees the model doesn't get much better, mainly due to overfitting. We double check that an mtry bigger than 4 reduces the accuracy of the model, as seen in graph RF 6

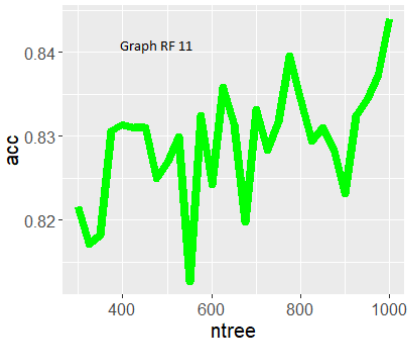




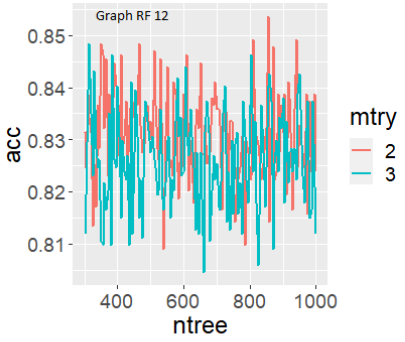
In this trial, we narrow down the mtry to 2, 3 and 4 and give ntree values in the same interval but with step 50 (300 to 1500 by 50). The graph RF 7 shows clearly how mtry equal to 4

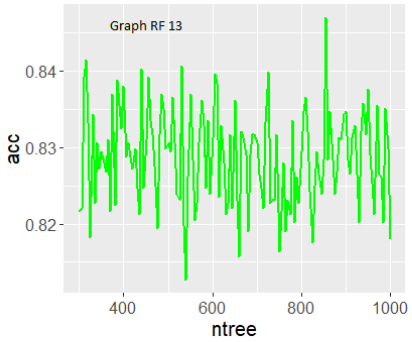
gives slightly lower accuracies than 2 and 3. In addition, graph RF 8 upholds our previous assumption that after 1000 trees, the model is overfitted

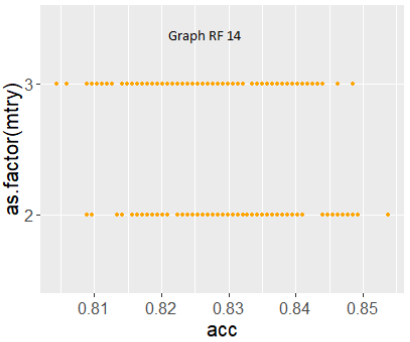




Now, we take out the 4 in mtry and limit the ntrees to 1000, reducing the step to 25 as well. Thanks to the data we got this time, we can confirm all our previous assumptions and establish clearly the boundaries in which the hyperparameters have to be included.







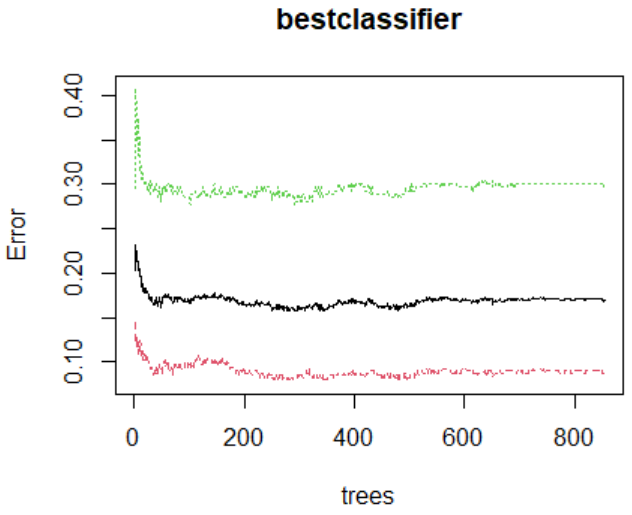
Finally, we run one last time the loop using the boundaries we got from the previous process in which we try a wide range of values for ntree(300 to 1000 step 5), while leaving mtry between 2 and 3. In this case, the whole process gives us a lot of information, as we can see in the graphs. If we analyze all the data, the random forest with the best accuracy(0.8537313) when we set ntree as 855 and mtry as 2.

```
mtry ntree       acc
   2   855 0.8537313
```

WHICH IS THE BEST MODEL?

Finally, we have our best decision tree and our best random forest, so the question now is, which is the better supervised machine learning method to predict survival between these two?
Looking at the error estimation obtained with each of the models, we can state that the random forest gives us a more accurate precision (which was something we could expect, given that random forests are more 'complex' than decision trees).

If we plot the best random forest obtained in the previous part of the report, we see three different functions. The top one shows the estimated error in the worst case scenario, while the floor one states the opposite, the estimated error in the best case scenario. Finally, the middle function shows the average error. In this case, the average estimated error is around 0.16. If we print the forest, we get that the out of bag error rate is equal to 15.87%, which condenses the information seen in the plot.



In order to be able to export all of our work so that we can predict the survival variable given a different dataset, we store all the processes that are necessary inside a function that needs a given parameter. Inside the function, we predict the values of the variable survive using the best random forest we got, returning the prediction, the confusion matrix and the values of accuracy, precision and specificity.

Inside the function, we clean the data set given as a parameter. First, we look for empty values in the variables used in the random forest and substitute them for either the mode in categorical data and or the mean in numeric data. Then, we replace the values in the column fare that are equal to zero with the mean.

Finally, our function looks as follows:

```
function(test_set){

  library("randomForest")

  test_set$Fare[ is.na(test_set$Fare)]=mean(test_set$Fare, na.rm = TRUE)
  test_set$Survived[ is.na(test_set$Survived)]= Mode(test_set$Survived)
  test_set$Pclass[ is.na(test_set$Pclass)]=Mode(test_set$Pclass)
  test_set$Sex[ is.na(test_set$Sex)]=Mode(test_set$Sex)
  test_set$Age[ is.na(test_set$Age)]=mean(test_set$Age, na.rm = TRUE)
  test_set$SibSp[ is.na(test_set$SibSp)]=Mode(test_set$SibSp)
  test_set$Parch[ is.na(test_set$Parch)]=Mode(test_set$Parch)
  test_set$Embarked[ is.na(test_set$Embarked)]=Mode(test_set$Embarked)

  aux2 = which(test_set$Fare == 0)
  meanfare = mean(test_set$Fare[-aux2])
  test_set$Fare[aux2] = rep(meanfare)

  pred = predict(bestclassifier, test_set, type = "class")
  conf_matrix = table(test_set$Survived,pred,dnn=c("Actual value","Classifier prediction"))
  accuracy = sum(diag(conf_matrix))/sum(conf_matrix)
  precision = conf_matrix[1,1]/sum(conf_matrix[,1])
  specificity = conf_matrix[2,2]/sum(conf_matrix[,2])
  return(list(prediction = pred, conf_matrix = conf_matrix,accuracy = accuracy,
              specificity = specificity, precision = precision, test_set = test_set))
}
```