# Random forest model

*Metales abdelkader*

*02/12/2019*

the whole file in my github repository

**Introduction:**

In this tutorial I will show you how to perform a random forest model using the following **titanic** data. But first we have to call the following packages :

```r
library(tidyverse)
library(randomForest)
library(caret)
library(e1071)
```

Then we call our data and display a first few rows:

```r
mydata<-read_csv("train.csv")
knitr::kable(head(mydata[,4]),caption= "titanic data")
```

Table 1: titanic data

| Name |
|---|
| Braund, Mr. Owen Harris |
| Cumings, Mrs. John Bradley (Florence Briggs Thayer) |
| Heikkinen, Miss. Laina |
| Futrelle, Mrs. Jacques Heath (Lily May Peel) |
| Allen, Mr. William Henry |
| Moran, Mr. James |

Our main goal is to predict the variable **Survived** based on the remaining features of this data.However not all these columns can be a potential predictors such as **PassengerId**, **Name**, and **Ticket** since each case has unique values for these variables. Also the variable **Cabin** is highly unlikely to provide any information about the output **Survived** besides the large number of missing values that has . therefore, it would be highly convenient to remove these variables from our analysis.

```r
mydata<-mydata[,-c(1,4,9,11)]
knitr::kable(head(mydata),caption= "titanic data")
```

Table 2: titanic data

| Survived | Pclass | Sex | Age | SibSp | Parch | Fare | Embarked |
|---|---|---|---|---|---|---|---|
| 0 | 3 | male | 22 | 1 | 0 | 7.2500 | S |
| 1 | 1 | female | 38 | 1 | 0 | 71.2833 | C |
| 1 | 3 | female | 26 | 0 | 0 | 7.9250 | S |
| 1 | 1 | female | 35 | 1 | 0 | 53.1000 | S |

| Survived | Pclass | Sex | Age | SibSp | Parch | Fare | Embarked |
|---|---|---|---|---|---|---|---|
| 0 | 3 | male | 35 | 0 | 0 | 8.0500 | S |
| 0 | 3 | male | NA | 0 | 0 | 8.4583 | Q |

Now let's check the structure of the data:

```r
glimpse(mydata)
```

```
## Observations: 891
## Variables: 8
## $ Survived <dbl> 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,...
## $ Pclass   <dbl> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3, 2,...
## $ Sex      <chr> "male", "female", "female", "female", "male", "male",...
## $ Age      <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, 39,...
## $ SibSp    <dbl> 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 0,...
## $ Parch    <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0,...
## $ Fare     <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, 51....
## $ Embarked <chr> "S", "C", "S", "S", "S", "Q", "S", "S", "S", "C", "S"...
```

we have two character variables **Sex** and **Embarked**, they are more suitable to be of factor type than character. the same note applies to the numerical variables **Survived** (our output),**Pclass**, they should be also of factor type. So we convert these variables as follows:

```r
mydata$Survived<-as.factor(mydata$Survived)
mydata$Sex<-as.factor(mydata$Sex)
mydata$Pclass<-as.factor(mydata$Pclass)
mydata$Embarked<-as.factor(mydata$Embarked)
```

No Let's get some summary about this data:

```r
summary(mydata)
```

```
##  Survived Pclass      Sex           Age            SibSp
##  0:549    1:216   female:314   Min.   : 0.42   Min.   :0.000
##  1:342    2:184   male  :577   1st Qu.:20.12   1st Qu.:0.000
##           3:491                Median :28.00   Median :0.000
##                                Mean   :29.70   Mean   :0.523
##                                3rd Qu.:38.00   3rd Qu.:1.000
##                                Max.   :80.00   Max.   :8.000
##                                NA's   :177
##      Parch             Fare         Embarked
##  Min.   :0.0000   Min.   :  0.00   C   :168
##  1st Qu.:0.0000   1st Qu.:  7.91   Q   : 77
##  Median :0.0000   Median : 14.45   S   :644
##  Mean   :0.3816   Mean   : 32.20   NA's:  2
##  3rd Qu.:0.0000   3rd Qu.: 31.00
##  Max.   :6.0000   Max.   :512.33
##
```

What attracts our attention is the large number of missing values in the **Age** variable (177), and two missing values in **Embarked**. To deal with this problem we have two alternatives, the first is to remove the entire rows that contain any missing value if we think that the removed rows number is small compared to the length of the entire data. The second alternative is to impute the missing values by making use of the many method that are designed to this purpose. (predict the missing values from the other complete values in the data). In our case we will prefer the second option since we have a moderately large number of missing values, to do so we will use the **mice** package:

```
library(mice)
impute<-mice(mydata,m=3,seed=111)
```

```
##
##  iter imp variable
##   1   1  Age  Embarked
##   1   2  Age  Embarked
##   1   3  Age  Embarked
##   2   1  Age  Embarked
##   2   2  Age  Embarked
##   2   3  Age  Embarked
##   3   1  Age  Embarked
##   3   2  Age  Embarked
##   3   3  Age  Embarked
##   4   1  Age  Embarked
##   4   2  Age  Embarked
##   4   3  Age  Embarked
##   5   1  Age  Embarked
##   5   2  Age  Embarked
##   5   3  Age  Embarked
```

```
mydata<-complete(impute,1)
```

```
summary(mydata)
```

```
##  Survived Pclass      Sex           Age             SibSp
##  0:549    1:216   female:314   Min.   : 0.42   Min.   :0.000
##  1:342    2:184   male  :577   1st Qu.:21.00   1st Qu.:0.000
##           3:491                Median :28.00   Median :0.000
##                                Mean   :29.75   Mean   :0.523
##                                3rd Qu.:38.00   3rd Qu.:1.000
##                                Max.   :80.00   Max.   :8.000
##      Parch              Fare        Embarked
##  Min.   :0.0000   Min.   :  0.00   C:168
##  1st Qu.:0.0000   1st Qu.:  7.91   Q: 77
##  Median :0.0000   Median : 14.45   S:646
##  Mean   :0.3816   Mean   : 32.20
##  3rd Qu.:0.0000   3rd Qu.: 31.00
##  Max.   :6.0000   Max.   :512.33
```

Now we do not have any missig value and hence we can go ahead.

### Data partition:

To asses our future model we should split the data into two sets one for training (80% of the data) and the second (the remaining 20%). To perform this partition we make use of **caret** package:

```r
set.seed(1234)
index <- createDataPartition(mydata$Survived,p=0.8,list=FALSE)
train <- mydata[index,]
test <- mydata[-index,]
```

## Train the model:

We will train the random forest model on the train data and print the result.

```r
set.seed(1234)
RF<-randomForest(Survived~.,data=train)
RF
```

```
##
## Call:
##  randomForest(formula = Survived ~ ., data = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 15.55%
## Confusion matrix:
##     0   1 class.error
## 0 404  36  0.08181818
## 1  75 199  0.27372263
```

with the default parameter values for this model( number of trees 500, and the selected variables number 2 at each iteration) the out of bage rate OOB is a little bit large (15.55%). however ,from the confusion matrix, the model is quite good to predict the class (0, not survived) with error rate of 8%, unlike the seconde class(1) where the prediction is poor.

if we would like to get further informations about the model , we call the attributes function:

```r
attributes(RF)
```

```
## $names
##  [1] "call"            "type"            "predicted"
##  [4] "err.rate"        "confusion"       "votes"
##  [7] "oob.times"       "classes"         "importance"
## [10] "importanceSD"    "localImportance" "proximity"
## [13] "ntree"           "mtry"            "forest"
## [16] "y"               "test"            "inbag"
## [19] "terms"
##
## $class
## [1] "randomForest.formula" "randomForest"
```

Using these informations about attributes we can pull out any information we want.

## Prediction

For prediction we use the **caret** package.

```
pred_train<-predict(RF,data=train)
head(pred_train)
```

```
## 1 3 4 6 7 9
## 0 0 1 0 0 1
## Levels: 0 1
```

To compare the predicted values with the actual values we make use of the confusion matrix function.

```
confusionMatrix(pred_train,train$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 404  75
##          1  36 199
##
##                Accuracy : 0.8445
##                  95% CI : (0.8158, 0.8703)
##     No Information Rate : 0.6162
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6622
##
##  Mcnemar's Test P-Value : 0.00031
##
##             Sensitivity : 0.9182
##             Specificity : 0.7263
##          Pos Pred Value : 0.8434
##          Neg Pred Value : 0.8468
##              Prevalence : 0.6162
##          Detection Rate : 0.5658
##    Detection Prevalence : 0.6709
##       Balanced Accuracy : 0.8222
##
##        'Positive' Class : 0
##
```

According to the accuarcy rate of the model 0.8445, the model is quite good. the sensitivity 0.9159 indicates that the model is doing better predictions for the positive class(0) than the second class where the specificity is 0.7263.

## prediction of the test data

the best evaluation of the model is to predict data that has not been seen by the model, hence we will make use of our test data.

```
predtest <- predict(RF, test)
head(predtest)
```

```
##  2  5  8 10 14 15
##  1  0  0  1  0  1
## Levels: 0 1
```

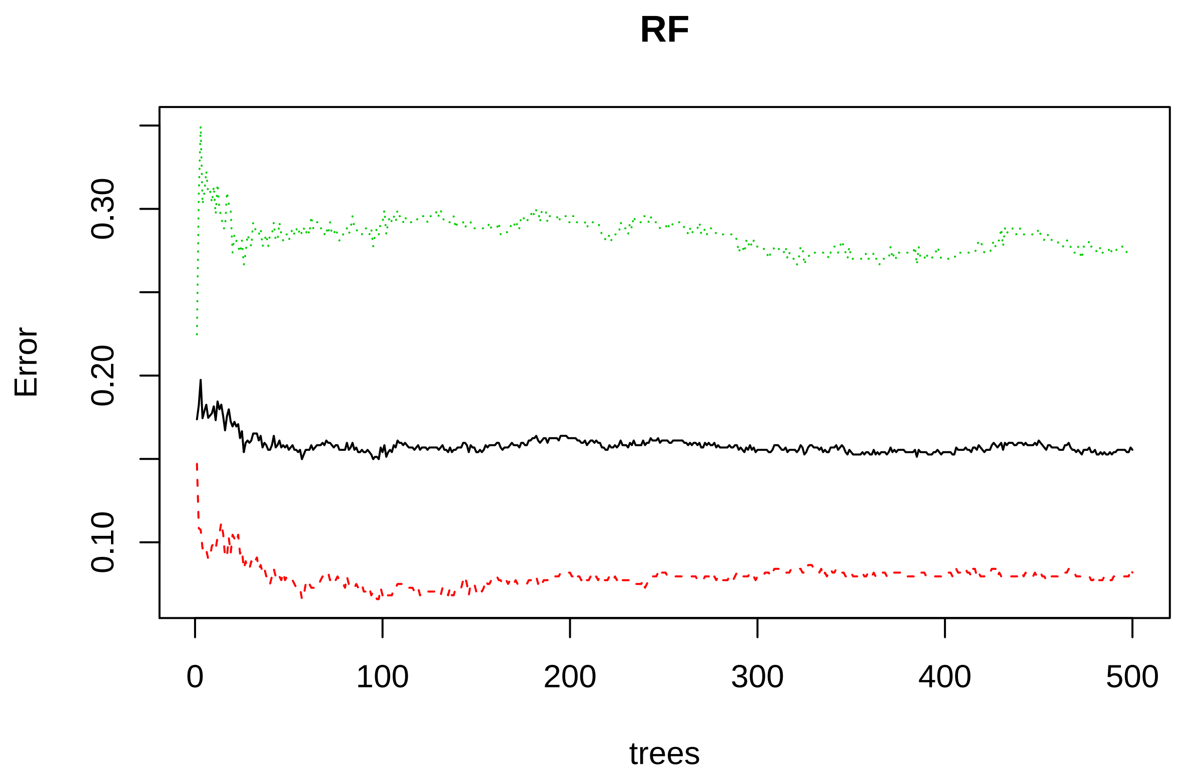
And the confusion matrix will be:

```
confusionMatrix(predtest,test$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 99 31
##          1 10 37
##
##                Accuracy : 0.7684
##                  95% CI : (0.6992, 0.8284)
##     No Information Rate : 0.6158
##     P-Value [Acc > NIR] : 1.153e-05
##
##                   Kappa : 0.4803
##
##  Mcnemar's Test P-Value : 0.001787
##
##             Sensitivity : 0.9083
##             Specificity : 0.5441
##          Pos Pred Value : 0.7615
##          Neg Pred Value : 0.7872
##              Prevalence : 0.6158
##          Detection Rate : 0.5593
##    Detection Prevalence : 0.7345
##       Balanced Accuracy : 0.7262
##
##        'Positive' Class : 0
##
```

With the test data we get less accuracy (about 77%), however we can go back and try tuning the model parameters which may improve the accuracy. But before that let's plot these parameters to guide us for the tuning operation.

```
plot(RF)
```

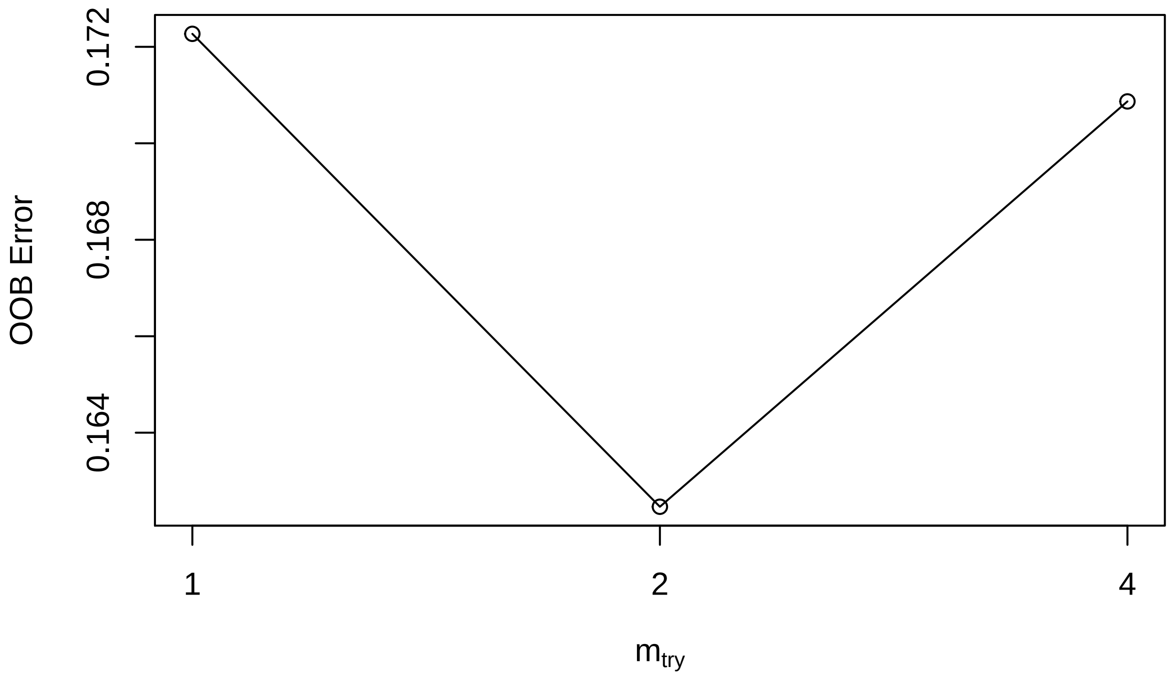


The OOB error has dropped down quikely during the first 100 iterations and then stabilaizes until the end of the process. se we can reduce the number of iterartions to see what we will get.

### Tune the model

```
tuned <- tuneRF(train[,-1],train[,1],ntreeTry = 300,improve = 0.01,
       trace = TRUE,plot = TRUE)
```

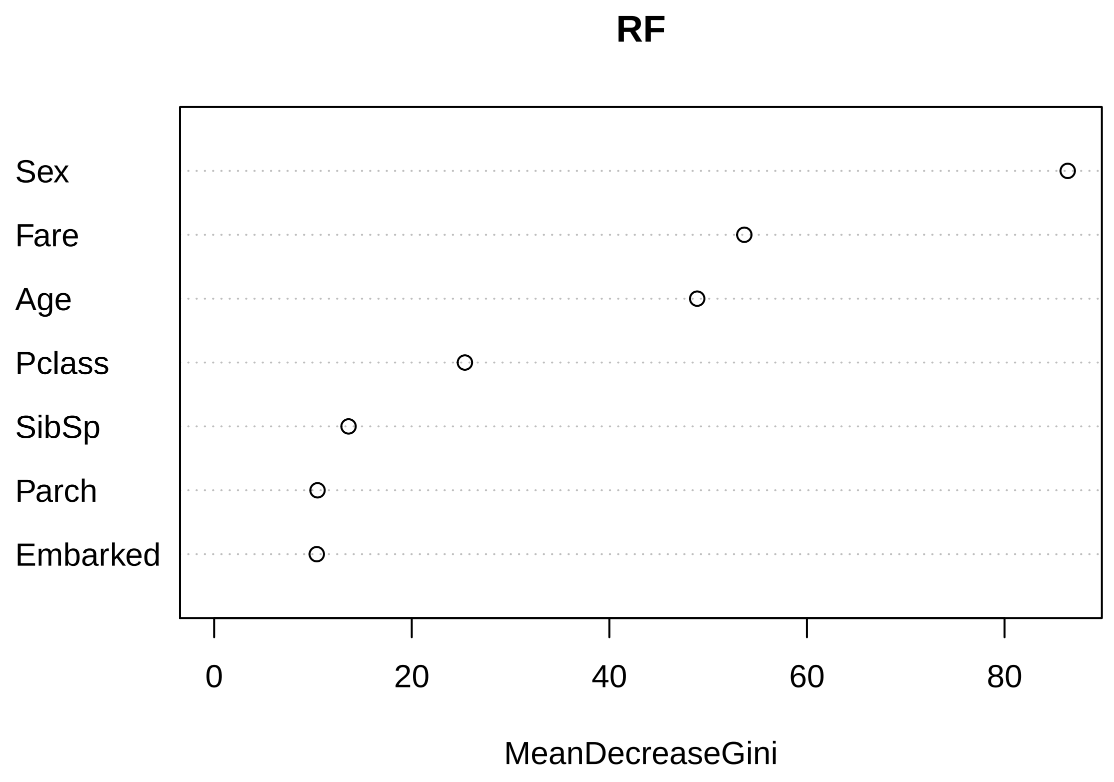
```
## mtry = 2  OOB error = 16.25%
## Searching left ...
## mtry = 1     OOB error = 17.23%
## -0.06034483 0.01
## Searching right ...
## mtry = 4     OOB error = 17.09%
## -0.05172414 0.01
```



I have tried many different values for ntree and the best values that i could get is 300 instead of the default of 500 but with the same oob (15,55%), in this case it is advisable to use the new model with ntree value 300 sinse it is less complex, or However you can also stick with your original model.

we can also update our model by adding or removing some predictors, but first let's check the most significante predictors.

```r
varImpPlot(RF)
```



From this plot we see that **sex** has the most importante contribution on this model followed by **Fare** and **Age** , and the remaining ones are less importante but still have some small contributions. let's try to remove the last one **Embarked** and see what happens.

```r
set.seed(1234)
RF1<-randomForest(Survived~.-Embarked,data=train,importance=TRUE)
RF1
```

```
##
## Call:
##  randomForest(formula = Survived ~ . - Embarked, data = train,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 14.99%
## Confusion matrix:
##     0   1 class.error
## 0 406  34  0.07727273
## 1  73 201  0.26642336
```

removing this variable results in a slight improvment in the OOB error rate from 15.55% to 14.99%. now let's check the accuracy in the training set using this model.

```r
pred_train1<-predict(RF1,data=train)
confusionMatrix(pred_train1,train$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 406  73
##          1  34 201
##
##                Accuracy : 0.8501
##                  95% CI : (0.8218, 0.8755)
##     No Information Rate : 0.6162
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6744
##
##  Mcnemar's Test P-Value : 0.0002392
##
##             Sensitivity : 0.9227
##             Specificity : 0.7336
##          Pos Pred Value : 0.8476
##          Neg Pred Value : 0.8553
##              Prevalence : 0.6162
##          Detection Rate : 0.5686
##    Detection Prevalence : 0.6709
##       Balanced Accuracy : 0.8282
##
##        'Positive' Class : 0
##
```

The accuracy is slightly improved by a very small amount from 84.45% to 85.01%.

Let's check the accuracy for the test data.

```r
predtest1 <- predict(RF1, test)
confusionMatrix(predtest1,test$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 101  29
##          1   8  39
##
##                Accuracy : 0.791
##                  95% CI : (0.7236, 0.8483)
##     No Information Rate : 0.6158
##     P-Value [Acc > NIR] : 4.548e-07
##
##                   Kappa : 0.531
##
##  Mcnemar's Test P-Value : 0.001009
```

```
##
##             Sensitivity : 0.9266
##             Specificity : 0.5735
##          Pos Pred Value : 0.7769
##          Neg Pred Value : 0.8298
##              Prevalence : 0.6158
##          Detection Rate : 0.5706
##    Detection Prevalence : 0.7345
##       Balanced Accuracy : 0.7501
##
##        'Positive' Class : 0
##
```

The same improvment has happned for the test data where the accuracy rate has increased from 76.84% to 79.1%.

Finaly we can save our model for further use.

```r
saveRDS(RF1,"RFtitanic.RDS")
```