UNIVERSITÀ
DEGLI STUDI
DI MILANO

# Credit card churning customers classification

## A Supervised Learning Approach

Filippo Menegatti*

*Data Science and Economics, University of Milan

# Contents

**1 Introduction** **3**

**2 Short Theoretical Background** **3**
2.1 Classification Glossary . . . . . 3
2.2 Supervised Learning Techniques . . . . . 4
2.2.1 Decision Tree . . . . . 5
2.2.2 Random Forest . . . . . 6

**3 Data set** **6**
3.1 Overview of the data . . . . . 6

**4 Discussion and results** **8**
4.0.1 Decision Tree . . . . . 9
4.0.2 Random Forest . . . . . 10
4.0.3 Model Selection . . . . . 13
4.0.4 Hyperparameters Tuning . . . . . 14
4.0.5 Balance the Data . . . . . 15

**5 Conclusion** **17**

**6 References** **18**

**7 Code** **18**

# 1 Introduction

Nowadays the problem of churning clients is a real and growing issue for financial institutions. Competition in the market increased also with the boost given by fintech sector causing the creation of new subjects able to serve the needs of many more customers. Due to that banks are in a sense obliged to offer some sign-up gift to tempt people using their services. This can cause customers to take advantage of that subscribing and than stop using the service. In this essay we are going to try two different supervised statistical learning approaches in order to forecast the churning clients with the highest possible precision.

# 2 Short Theoretical Background

In this part of the essay we are going to introduce the statistical models which will be used to explain the relationships between the variables and if they can explain the behavior of the customers.

## 2.1 Classification Glossary

In classification problems there are many different indexes relevant that can be used to asses the quality of the analysis. A list of them will follow with a brief explanation:

- **Confusion Matrix:** The confusion matrix is an useful representation of the general performance of the classification algorithm, it shows the amount of True Positive, True Negative, False Positive and False Negative observation predicted.
- **Accuracy:** The accuracy is the basic index used to calculate the performance of the model, it is simply based on the sum of the right prediction made compared with the total population available.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total Pop}}$$

- **Recall:** The recall (or sensitivity) is calculated as the ratio between

the true positive values of the model and the overall number of **real** positives including those not predicted (false negatives).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **Precision:** The precision of the model - differently from recall - is calculated as the ratio between the true positives and the total positives **predicted** (including wrong predictions).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **F1 Score:** The F1 Score consists in the harmonic mean between Precision and Recall, and gives a good compromise between the two without assigning more importance to one or to the other.

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

## 2.2 Supervised Learning Techniques

In supervised learning models there is a certain target of the analysis that will guide all the learning process. Supervised learning algorithms need to be trained with different examples to be able to achieve the final goal. To do this a training set is generally used together with a test set which will be used to control the error made by the algorithm. Different methods are also used to **generalize** the measure of the error: one of the most famous is the K-Fold Cross Validation (CV). It is used to divide the data set in K folds, K-1 parts are used as training set, while the K-th is used as test set. This procedure iterates for all the K parts and permits to calculate the CV-error averaging all the errors found.
A similar procedure is used to tune the hyperparameters of the model creating also a validation set used to tune them.

### 2.2.1 Decision Tree

The Decision Tree is a supervised model useful when the features of the data are widely heterogeneous, presenting incomparable ranges or mixed data types. It is in fact used to predict the label associated with an instance going from a root node to a leaf. In our case the task concerns binary classification so the label set is represented by $Y = \{0, 1\}$. At the root node all the population is considered and the label associated with it is the most frequent. At each subsequent division two more nodes are generated, following certain arbitrarily chosen rules. If the node is a *terminal node* or a leaf, no more splits are made and it is labeled with the corresponding value, if instead it is a *decision node* the process continues and more splits are made until only leaves remain.

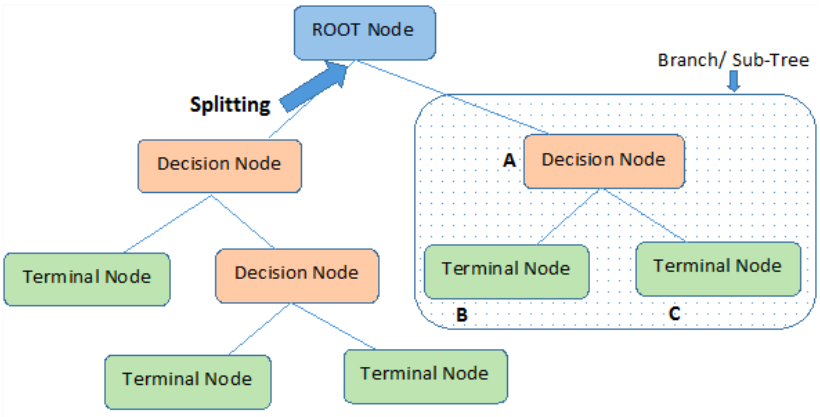



Figure 1: Example of Binary Decision Tree

It can be proved that an additional split of the tree does not cause an increase of the training error, but if their number grows too much with respect to the cardinality of the training set the tree can be affected by overfitting (high variance error), so it could perform much better on the training set than on the test set.

For these reasons many methods are used to solve this problem, like *pruning*, a method based on the selection of a smaller number of nodes, in order to "simplify" the model, and *random forest,* an algorithm to which we will dedicate the following section.

### 2.2.2 Random Forest

Random Forest [1][2] is an ensemble method derived from bagging, it used to create a certain number of trees which are uncorrelated between each other, in order to reduce the overall variance of the model injecting a certain amount of bias. The reduction in correlation between the trees is made possible by the random selection of the input variables during the process of tree growing. In this way, the creation of B independent trees with bagging permits to reduce the variance, following this formula:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

# 3 Data set

## 3.1 Overview of the data

The data set used has been taken from Kaggle website [3] and contains many data referred to the consumers of a bank, describing their characteristics and if they are credit card churners or not.
This means that they have exploited the offers available signing up with a new card but then they have quickly ended their contract.
The data set is characterized by the presence of mixed data, going through the sex and the level of education of the subject until his/her financial movements.

As it is possible to see from the summarisation of the data, there are some individuals categorized as "Unknown", but the analysis has shown that the performance does not change in a meaningful way removing them.

Here we can see some bar plots regarding the relationship between the categorical variables and the target:


Figure 2: Categorical Variables vs Target

As we can see, the categorical variables seem to be not so related to forecast the churn of a customer, however most of them use the basic Blue Card and have an income lower than 40k per year. This is quite logical because wealthy people usually don't need the sign-up gifts from banks, and it is logical to take the cheaper offer if you want to give it up soon.

Subsequently we are going to use the Spearman Correlation coefficient to

construct a correlation matrix and analyze what variables have the strongest relation between each other. Differently from the Pearson's correlation, the Spearman benchmarks the *monotonic* relationship between the variables and not the *linear* one. For this passage only we are going to treat our target variable as numeric, while then it will be treated as a factor.

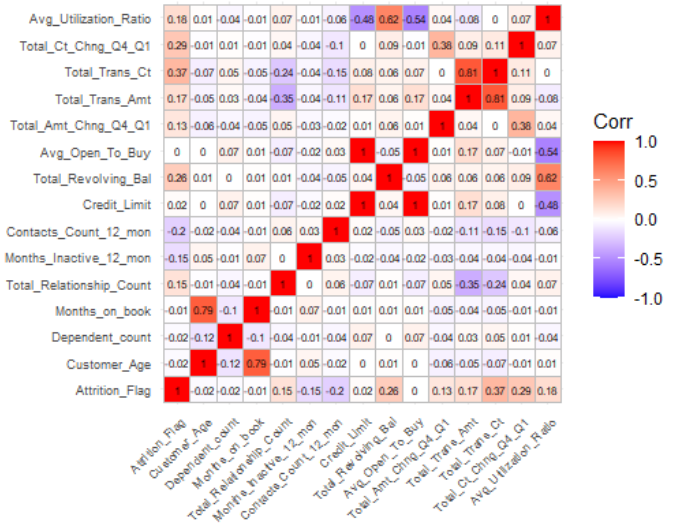



Figure 3: Spearman Correlation Matrix

As we can see there are variables related with our target, mostly positively: Total_Trans_Ct, Total_Ct_Chng_Q4_Q1, Total_Revolving_Bal, Total_Trans_Amt and Avg_Utilization_Ratio. Only Contacts_Count_12_mon and Months_Inactive_12_mon have a noteworthy negative correlation value.

So we can have a first idea about what variables can be useful in order to predict the customers behavior.

/pagebreak

# 4 Discussion and results

Now we are going to proceed with the analysis dividing our data set in test and training set, keeping the 80% 0f the data in it. In this way we are able

to apply our statistical methods to further analyze the data and so try to predict the best results.

### 4.0.1 Decision Tree

We proceed applying the standard Decision Tree classifier using the `rpart` function from the homonym library.

Here is the graph of the resulting tree pruned in correspondence of a complex parameter of 0.023:

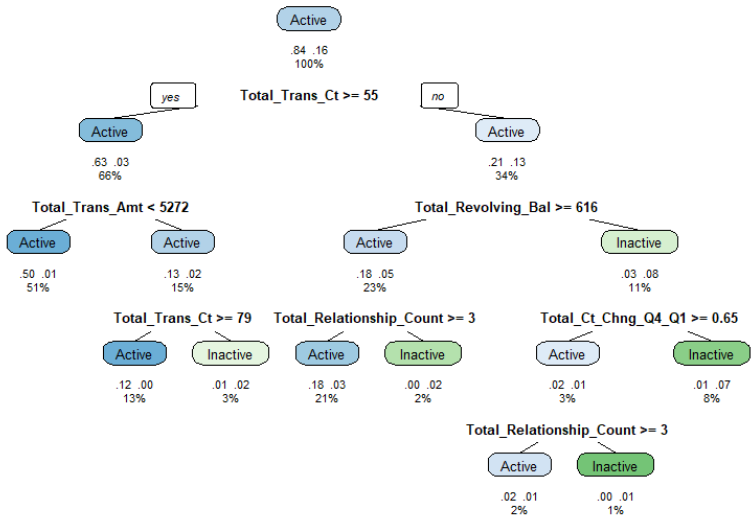



Figure 4: Decision Tree Graph 1

Table 1: Confusion Matrix

| Actual\Prediction | Active | Inactive |
|---|---|---|
| **Active** | 1637 | 52 |
| **Inactive** | 94 | 243 |

Table 2: Scores Table

| Index | Score |
|---|---|
| Precision | 0.9456961 |
| Recall | 0.9692126 |
| Accuracy | 0.9279368 |
| F1 Score | 0.9573099 |

The results are quite interesting: as we can see, we have an high value of recall with about 97%, while the accuracy is slightly lower at 93%.

#### 4.0.2 Random Forest

We try to approach the problem using the Random Forest algorithm - available in the `randomForest` package - to inject some bias in the model, trying to reduce the variance error. The results using the Random Forest are encouragingly improved, although the accuracy is again the lowest index.

Using the random forest function we are also able to obtain the *importance plot* of the model:

From the plot we can see what are the most important variables in the analysis of our target based on the mean decrease in accuracy of the model and on the mean decrease in node impurity. In the top three we have 'Total_Trans_CT', 'Total_Trans_AMT', 'Total_Relationship_Count' from one side and 'Total_Trans_AMT', 'Total_Trans_CT' and 'Total_Ct_Chng_Q4_Q1' (this is also quite consistent with the results of a previous analysis on this data set using Bayesian Networks).

Table 3: Confusion Matrix 2

| Actual\Prediction | Active | Inactive |
|---|---|---|
| **Active** | 1671 | 18 |
| **Inactive** | 48 | 289 |

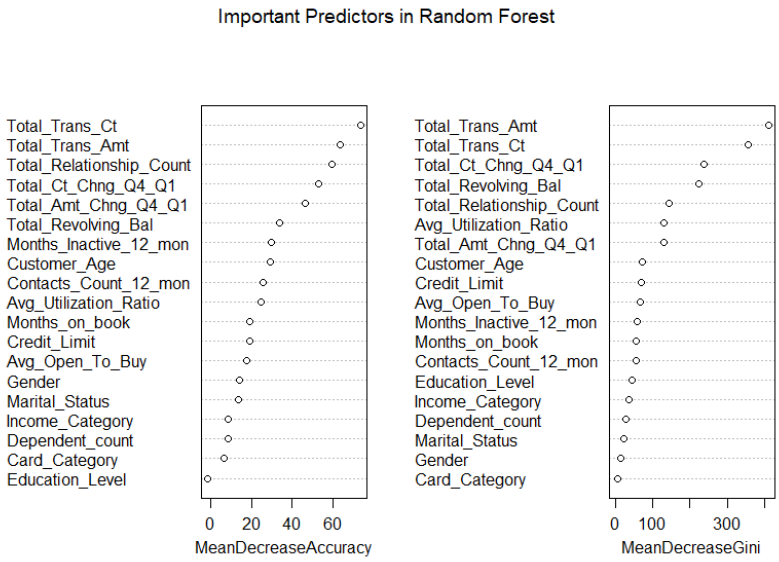



Figure 5: Random Forest - Importance Plot

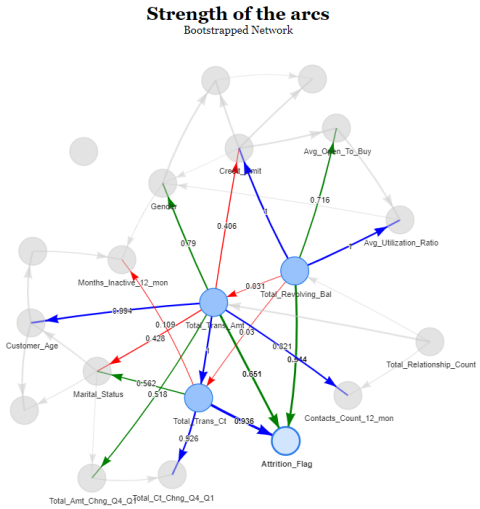



Figure 6: Here there are the variables with the strongest connection with the target, found using Bayesian Network techniques

Table 4: Score Table 2

| Index | Score |
|---|---|
| Precision | 0.9720768 |
| Recall | 0.9893428 |
| Accuracy | 0.9674235 |
| F1 Score | 0.9806338 |

The results of the Random Forest show an improvement in all the parameters, reducing the amount of false negative from 52 to 18.

### 4.0.3 Model Selection

We are now going to apply the *Decision Tree Classifier* using just the top six variables identified by the Random Forest (the top five for both the scores used) to see if the results are robust.

Looking at the indexes the result is satisfying and all the parameters are quite robust with respect to the first tree made using the complete data set. The tree has been pruned this time at a complexity level of 0.014. We represent again the output tree and the tables with the corresponding results:

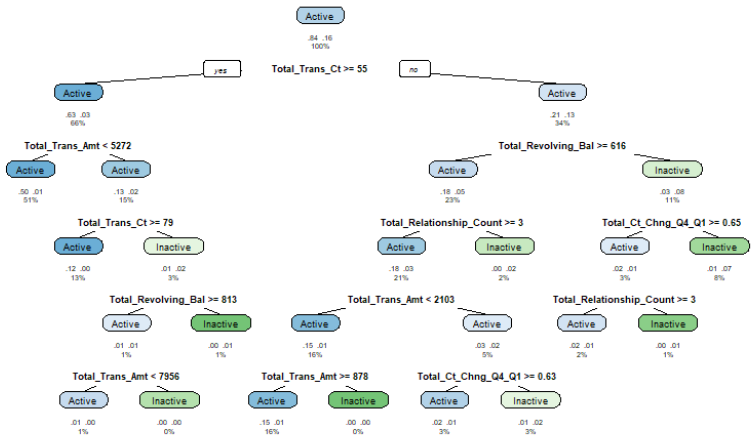



Figure 7: Decision Tree Graph 2

Table 5: Confusion Matrix 3

| Actual\Prediction | Active | Inactive |
|---|---|---|
| **Active** | 1631 | 58 |
| **Inactive** | 74 | 263 |

Table 6: Score Table 3

| Index | Score |
|---|---|
| Precision | 0.9565982 |
| Recall | 0.9656602 |
| Accuracy | 0.9348470 |
| F1 Score | 0.9611078 |

Trying the same approach with the Random Forest we can see that the scoring indexes are really similar, so the overall result is pretty acceptable considering the lower amount of data needed with this approach.

Table 7: Confusion Matrix 4

| Actual\Prediction | Active | Inactive |
|---|---|---|
| **Active** | 1664 | 25 |
| **Inactive** | 36 | 298 |

Table 8: Score Table 4

| Index | Score |
|---|---|
| Precision | 0.9788235 |
| Recall | 0.9851983 |
| Accuracy | 0.9698914 |
| F1 Score | 0.9820006 |

#### 4.0.4 Hyperparameters Tuning

We are going to proceed with the tuning of the Random Forest's hyperparameters. We will stick with the new variables setting and so we are going to keep just the six features used before. We train our model trying 500, 1000, 1500, 2000 trees and the mtry, the number of variables randomly sampled as candidates at each split equal to the square root of the number of columns

of the data set (default value), 4 and 6, for the reduced data set, and default, 5, 10, 15, 20 for the other, using a 10 folds Cross Validation to control the result.

The 500 trees model using a default mtry is the one with the best performances for the model selection data set, confirming the previous results.

The complete data set random forest outperformed the previous one in all the indexes calculated using a mtry of 10 and 1000 trees:

Table 9: Confusion Matrix 5

| Actual\Prediction | Active | Inactive |
|---|---|---|
| **Active** | 1673 | 16 |
| **Inactive** | 35 | 302 |

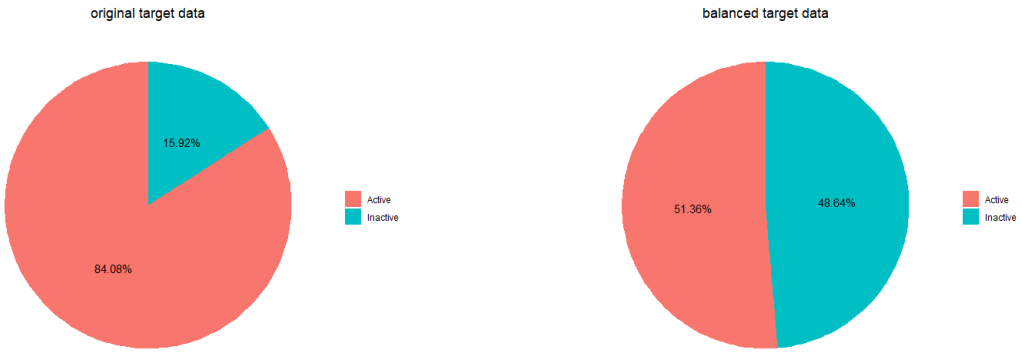
Table 10: Score Table 5

| Index | Score |
|---|---|
| Precision | 0.9795082 |
| Recall | 0.9905269 |
| Accuracy | 0.9748272 |
| F1 Score | 0.9849868 |

#### 4.0.5 Balance the Data

Now we will try to modify our approach applying the `SMOTE` function from the `smootefamily` package, and balancing the target categories contained in the training set. The algorithm used is deeply described in the paper from Chawala Et Al. 2002 [4], but to summarize its functions it uses e the K Nearest Neighbor algorithm to create some synthetic observations similar to the other already present in the sample so to balance our data.

Here's an example of the target variable before and after the application of the algorithm:




The new setting, applied only to the model selection data set, which contains only numerical variables, took us to a lower performance for the tree predictor, pruned at a cp of 0.016, with an higher precision (0.9791535), but lower recall (0.9177028), accuracy (0.9151037) and F1 Score (0.9474328) with respect to the previous trees. The tuned Random Forest performed quite good, with the following results:

Table 11: Confusion Matrix 6

| Actual\Prediction | Active | Inactive |
|---|---|---|
| **Active** | 1649 | 40 |
| **Inactive** | 26 | 311 |

Table 12: Score Table 6

| Index | Score |
|---|---|
| Precision | 0.9844776 |
| Recall | 0.9763173 |
| Accuracy | 0.9674235 |
| F1 Score | 0.9803805 |

# 5 Conclusion

Different methods are available to increase the performance of a simple decision tree, which remains the best in terms of visualization of the results. The use of random forest algorithm permits to increase the overall performance under every point of view and also to perform of variable selection process. Anyhow the algorithms performed better with the complete data set. Finally, the use of the SMOTE algorithm to balance the training set data did take us to some improvements, at least using the random forest, and specifically in the precision of the model.

# 6 References

[1] Kaggle Dataset

[2] “Random Forest” - J. Breiman (2001)

[3] “Elements of Statistical Learning” - T. Hastie Et Al. (2009)

[4] “SMOTE: Synthetic Minority Over-sampling Technique” - N. V. Chawla Et. Al. (2002)

# 7 Code

```
library(readr)
library(mlbench)
library(plyr)
library(caret)
library(ggplot2)
library(smotefamily)
library(randomForest)
library(rpart)
library(dplyr)
library(FactoMineR)
library(ggcorrplot)
library(rpart.plot)

confusion_matrix <- function(y_test, model, x_test){

    conf.matrix <- table(y_test,
      predict(model, newdata = x_test, type="class"))

    rownames(conf.matrix) <- paste("Actual",
      rownames(conf.matrix), sep = ":")

    colnames(conf.matrix) <- paste("Pred",
      colnames(conf.matrix), sep = ":")
```

```
    return(conf.matrix)
}

class_scoring <- function(conf_matrix){

    recall <- conf_matrix[1,1]/(conf_matrix[1,1] +
      conf_matrix[1,2])

    precision <- conf_matrix[1,1]/(conf_matrix[1,1] +
      conf_matrix[2,1])

    accuracy <- (conf_matrix[1,1]+conf_matrix[2,2])/
        (conf_matrix[1,1]+conf_matrix[1,2] +
        conf_matrix[2,1]+conf_matrix[2,2])

    f1_score <- 2*((precision*recall)
      /(precision+recall))

    results <- c(recall, accuracy, precision, f1_score)

    df <- data.frame(results)

    rownames(df) <- c('Recall', 'Accuracy', 'Precision', 'F1 Score')

    return(df)

}


BankChurners <- read_csv("BankChurners.csv")

BankChurners$Attrition_Flag <- revalue(BankChurners$Attrition_Flag,
  c("Existing Customer"= "Active",
    "Attrited Customer"="Inactive"))
```

```r
data <- BankChurners

data[, c(1,3,5:8)] <- lapply(data[, c(1,3,5:8)],
as.factor)

summary(data)

# some graphs about categorical variables wrt the target

ggplot(data) +
    aes(x = Education_Level, fill = Attrition_Flag) +
    geom_bar() +
    scale_fill_hue() +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45,
    vjust = 0.5, hjust=0.5))

ggplot(data) +
    aes(x = Income_Category, fill = Attrition_Flag) +
    geom_bar() +
    scale_fill_hue() +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45,
    vjust = 0.5, hjust=0.5))

ggplot(data) +
    aes(x = Card_Category, fill = Attrition_Flag) +
    geom_bar() +
    scale_fill_hue() +
    theme_minimal()

ggplot(data) +
    aes(x = Marital_Status, fill = Attrition_Flag) +
    geom_bar() +
```

```
    scale_fill_hue() +
    theme_minimal()

ggplot(data) +
    aes(x = Gender, fill = Attrition_Flag) +
    geom_bar() +
    scale_fill_hue() +
    theme_minimal()

seed = 4321
set.seed(seed)

cor_spearman <- cor(data[, sapply(data, is.numeric)],
  method = 'spearman')

as.matrix(data.frame(cor_spearman)) %>%
    round(3) %>% #round
    ggcorrplot('square','full', lab = TRUE)

# Split data into training (80%) and test (20%)
dt = sort(sample(nrow(BankChurners), nrow(BankChurners)*.8))
train<-BankChurners[dt,]
test<-BankChurners[-dt,]

train[, c(1,3,5:8)] <- lapply(train[, c(1,3,5:8)], as.factor)
test[, c(1,3,5:8)] <- lapply(test[, c(1,3,5:8)], as.factor)

### FIRST TREE


tree.mod = rpart(formula = Attrition_Flag ~ ., data = train)

plotcp(tree.mod)

tree.mod = prune.rpart(tree.mod, 0.023)
```

```
rpart.plot(tree.mod, fallen.leaves = FALSE,
extra = 109, under = TRUE,
tweak = 1.6, clip.facs = TRUE,
compress = FALSE, space = 0.35)

a <- caret::confusionMatrix(data = predict(tree.mod,
  newdata = test, type="class"),
  reference = test$Attrition_Flag,
  mode = 'everything')

matrix1 <- confusion_matrix(test$Attrition_Flag,
  model = tree.mod,
  x_test = test)

scores1 <- class_scoring(matrix1)

### FIRST RANDOM FOREST ###

rf <- randomForest(Attrition_Flag~.,
  data=train, importance=TRUE)

varImpPlot(rf, main = "Important Predictors in Random Forest")

b <- caret::confusionMatrix(data = predict(rf,
  newdata = test, type="class"),
  reference = test$Attrition_Flag,
  mode = 'everything')

matrix2 <- confusion_matrix(test$Attrition_Flag,
  model = rf,
  x_test = test)
scores2 <- class_scoring(matrix2)

################################################
```

```
###                model selection                ###
#####################################################

train.top = subset(train, select=c(Attrition_Flag,Total_Trans_Ct,
  Total_Trans_Amt,Total_Relationship_Count,
  Total_Ct_Chng_Q4_Q1,Total_Amt_Chng_Q4_Q1,
  Total_Revolving_Bal))

test.top = subset(test, select=c(Attrition_Flag,Total_Trans_Ct,
  Total_Trans_Amt,Total_Relationship_Count,
  Total_Ct_Chng_Q4_Q1,Total_Amt_Chng_Q4_Q1,
  Total_Revolving_Bal))


### SECOND TREE ###

tree.mod.top = rpart(formula = Attrition_Flag ~.,
  data = train.top)

plotcp(tree.mod.top)

tree.mod.top = prune(tree.mod.top, cp = 0.011)

rpart.plot(tree.mod.top, fallen.leaves = FALSE,
  extra = 109, under = TRUE,
  tweak = 1.7, clip.facs = TRUE,
  compress = TRUE, space = 0.35)

c <- caret::confusionMatrix(data = predict(tree.mod.top,
  newdata = test.top, type="class"),
  reference = test.top$Attrition_Flag,
  mode = 'everything')

matrix3 <- confusion_matrix(test$Attrition_Flag,
  model = tree.mod.top,
```

```
  x_test = test.top)

scores3 <- class_scoring(matrix3)

### SECOND RANDOM FOREST ###

rf.top <- randomForest(formula, data=train.top,
  importance=FALSE)

d <- caret::confusionMatrix(data = predict(rf.top,
  newdata = test.top, type="class"),
  reference = test.top$Attrition_Flag,
  mode = 'everything')

matrix4 <- confusion_matrix(test$Attrition_Flag,
  model = tree.mod.top,
  x_test = test.top)

scores4 <- class_scoring(matrix4)

##################################################
###                  Tuning                    ###
##################################################

### TUNED RANDOM FOREST WITH MODEL SELECTION ###

tuned.rf1 <- tune.randomForest(x = train.top[, -1],
  y = train.top[, c('Attrition_Flag')],
  ntree = c(500, 1000, 1500, 2000),
  mtry = c(sqrt(ncol(train.top)), 4, 6))

e <- caret::confusionMatrix(data =
  predict(tuned.rf1$best.model,
  newdata = test.top, type="class"),
  reference = test.top$Attrition_Flag,
```

```
  mode = 'everything')

matrix4 <- confusion_matrix(test.top$Attrition_Flag,
  model = tuned.rf1$best.model,
  x_test = test.top)

scores4 <- class_scoring(matrix4)

### TUNED RANDOM FOREST WITHOUT MODEL SELECTION ###

tuned.rf1bis <- tune.randomForest(x = train[, -1],
  y = train$Attrition_Flag,
  ntree = c(500, 1000, 1500, 2000),
  mtry = c(sqrt(ncol(train)), 5, 10, 15, 20))

f <- caret::confusionMatrix(data =
  predict(tuned.rf1bis$best.model,
  newdata = test.top, type="class"),
  reference = test.top$Attrition_Flag,
  mode = 'everything')

matrix5 <- confusion_matrix(test.top$Attrition_Flag,
  model = tuned.rf1bis$best.model,
  x_test = test.top)

scores5 <- class_scoring(matrix5)


################################################
###      SMOTE on the training test          ###
################################################

train.smote <- SMOTE(target = train.top$Attrition_Flag,
                     X = as.data.frame(train.top[,-1]),
                     K = 3, dup_size = 2)
```

```
train.smote <- rbind(train.smote$data,
  train.smote$syn_data)
names(train.smote)[names(train.smote) == 'class'] <- 'Attrition_Flag'
train.smote[, 7] <- as.factor(train.smote[, 7])

prob.or <- round((prop.table(table(train$Attrition_Flag))*100), 2)
orig <- data.frame(prob.or)


pie = ggplot(orig, aes(x="", y=Freq, fill=Var1)) +
  geom_bar(stat="identity", width=1)

pie = pie + coord_polar("y", start=0) +
  geom_text(aes(label = paste0(Freq, "%")),
  position = position_stack(vjust = 0.5))

pie = pie + labs(x = NULL, y = NULL,
  fill = NULL,
  title = "original target data")

pie = pie + theme_classic() +
  theme(axis.line = element_blank(),
  axis.text = element_blank(),
  axis.ticks = element_blank(),
  plot.title = element_text(hjust = 0.5))

# pie chart with balanced classes

prob.smote <- round((prop.table(table(train.smote$Attrition_Flag))*100), 2)
smote <- data.frame(prob.smote)

pie1 = ggplot(smote, aes(x="", y=Freq, fill=Var1)) +
  geom_bar(stat="identity", width=1)

pie1 = pie1 +
```

```
  coord_polar("y", start=0) +
  geom_text(aes(label = paste0(Freq, "%")),
  position = position_stack(vjust = 0.5))

pie1 = pie1 +
  labs(x = NULL, y = NULL,
      fill = NULL, title = "balanced target data")

pie1 = pie1 +
  theme_classic() +
  theme(axis.line = element_blank(),
    axis.text = element_blank(),
    axis.ticks = element_blank(),
    plot.title = element_text(hjust = 0.5))

### THIRD DECISION TREE with SMOTE ###

tree.mod2 = rpart(formula = Attrition_Flag ~ ., data = train.smote)

plotcp(tree.mod2)
tree.mod2 = prune(tree.mod2, cp=0.016)


g <- caret::confusionMatrix(data = predict(tree.mod2,
  newdata = test.top, type="class"),
  reference = test.top$Attrition_Flag,
  mode = 'everything')

matrix6 <- confusion_matrix(test.top$Attrition_Flag,
  model = tree.mod2,
  x_test = test.top)

scores6 <- class_scoring(matrix6)

### TUNED SMOOTE RANDOM FOREST
```

```
tuned.rf2 <- tune.randomForest(x = train.smote[, -7],
  y = train.smote[, c('Attrition_Flag')],
  ntree = c(500, 1000, 1500, 2000),
  mtry = c(sqrt(ncol(train.smote)), 4, 6))

tuned.rf2$best.model

h <- caret::confusionMatrix(data = predict(tuned.rf2$best.model,
  newdata = test.top, type="class"),
  reference = test.top$Attrition_Flag, mode = 'everything')

matrix7 <- confusion_matrix(test.top$Attrition_Flag,
  model = tuned.rf2,
  x_test = test.top)

scores7 <- class_scoring(matrix7)
```