Individual Project - Statistical & Machine Learning

By: Alejandra Zambrano

March 29, 2020

Introduction

The goal of this project is to describe, implement and analyze the results of five machine learning algorithms in order to get the most accurate predictions for the response to a subscription term deposit campaign in a bank.

The algorithms used are:

- Random Forest Classifier
- Logistic Regression
- Xtreme Gradient Boosting
- Gradient Boosting
- K-Nearest Neighbors

During the report, I will explain each ML algorithm, review about some advantages and disadvantages, and finally, apply the best models in the data set for testing in a Kaggle competition. This work also involves the application of some feature selection methods and cross-validation techniques.

Banking Dataset

For developing of this work, we will use a dataset from a bank's telemarketing campaign to sell long-term deposits, the target variable is the variable subscribe (numeric: 1 = 'yes', 0 = 'no') and it refers if the client subscribed the term deposit. This dataset has 7000 observations and 21 predictive variables:

- **Group 1: Bank client data:**
    1. client_id : unique ID of the client (numeric)
    2. age : client age (numeric)
    3. job : type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
    4. marital : marital status (categorical: 'divorced', 'married', 'single', 'unknown'; note: 'divorced' means divorced or widowed)
    5. education (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'unknown')
    6. default : has credit in default? (categorical: 'no', 'yes', 'unknown')
    7. housing : has housing loan? (categorical: 'no', 'yes', 'unknown')

8. loan : has personal loan? (categorical: 'no', 'yes', 'unknown')

- **Group 2: Related with the last contact of the current campaign:**
  9. contact : contact communication type (categorical: 'cellular', 'telephone')
  10. month: last contact month of year (categorical: 'jan', 'feb', 'mar', … 'nov', 'dec')
  11. dayofweek : last contact day of the week (categorical: 'mon', 'tue', 'wed', 'thu', 'fri')
- **Group 3: Other attributes:**
  12. campaign : number of contacts performed during this campaign and for this client (numeric, includes last contact)
  13. pdays : number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
  14. previous : number of contacts performed before this campaign and for this client (numeric)
  15. poutcome : outcome of the previous marketing campaign (categorical: 'failure', 'nonexistent', 'success')
- **Group 4: Social and economic context attributes:**
  16. emp.var.rate : employment variation rate - quarterly indicator (numeric)
  17. cons.price.idx : consumer price index - monthly indicator (numeric)
  18. cons.conf.idx : consumer confidence index - monthly indicator (numeric)
  19. euribor3m : euribor 3 month rate - daily indicator (numeric)
  20. nr.employed : number of employees - quarterly indicator (numeric)
- **Target variable:**
  21. subscribe : has the client subscribed a term deposit? (numeric: 1='yes', 0='no')

The selected model will be applied in a test set that contains 3000 observations and 20 variables, the same variables contained in the training set but without the variable subscribe, because this is the output that we want to predict.

Data Summary

In order to better understand the data set and get initial insights of the relationships of the variables, I will present some description, summary statistics and additional graphical representations.

Below, one can observe the type of variables in the training dataset, it contains 9 categorical variables and 10 numerical variables. The variable client_id will not be considered as a predictor variable since it does explain the behavior of the target variable and it is only an identifier for each client.

```
##      client_id            age            job        marital      education
##      "integer"      "integer"       "factor"       "factor"       "factor"
##        default        housing           loan        contact          month
##       "factor"       "factor"       "factor"       "factor"       "factor"
##    day_of_week       campaign          pdays       previous       poutcome
##       "factor"      "integer"      "integer"      "integer"       "factor"
```

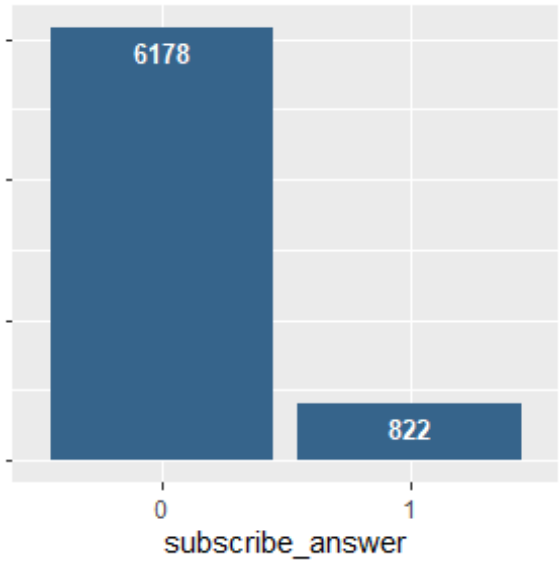
```
##   emp.var.rate cons.price.idx  cons.conf.idx      euribor3m    nr.employed
##      "numeric"      "numeric"      "numeric"      "numeric"      "numeric"
##      subscribe
##      "integer"
```

The below summary shows some statistic for each variable:

```
##    client_id         age                 job           marital
##  Min.   :   2   Min.   :18.00   admin.     :1756   divorced: 758
##  1st Qu.:2486   1st Qu.:32.00   blue-collar:1541   married :4306
##  Median :5018   Median :38.00   technician :1143   single  :1928
##  Mean   :5003   Mean   :40.34   services   : 668   unknown :   8
##  3rd Qu.:7512   3rd Qu.:48.00   management : 534
##  Max.   :9998   Max.   :98.00   retired    : 306
##                                 (Other)    :1052
##                education       default        housing          loan
##  university.degree  :2103   no     :5571   no     :3189   no     :5806
##  high.school        :1638   unknown:1429   unknown: 171   unknown: 171
##  basic.9y           : 993                  yes    :3640   yes    :1023
##  professional.course: 886
##  basic.4y           : 716
##  basic.6y           : 362
##  (Other)            : 302
##       contact         month      day_of_week    campaign          pdays
##  cellular :4482   may    :2323   fri:1320    Min.   : 1.000   Min.   :  0.
0
##  telephone:2518   jul    :1220   mon:1426    1st Qu.: 1.000   1st Qu.:999.
0
##                   aug    :1036   thu:1440    Median : 2.000   Median :999.
0
##                   jun    : 896   tue:1421    Mean   : 2.555   Mean   :962.
3
##                   nov    : 689   wed:1393    3rd Qu.: 3.000   3rd Qu.:999.
0
##                   apr    : 452               Max.   :33.000   Max.   :999.
0
##                   (Other): 384
##     previous             poutcome     emp.var.rate      cons.price.idx
##  Min.   :0.0000   failure    : 748   Min.   :-3.40000   Min.   :92.20
##  1st Qu.:0.0000   nonexistent:6021   1st Qu.:-1.80000   1st Qu.:93.08
##  Median :0.0000   success    : 231   Median : 1.10000   Median :93.44
##  Mean   :0.1761                      Mean   : 0.04881   Mean   :93.57
##  3rd Qu.:0.0000                      3rd Qu.: 1.40000   3rd Qu.:93.99
##  Max.   :6.0000                      Max.   : 1.40000   Max.   :94.77
##
##  cons.conf.idx      euribor3m      nr.employed     subscribe
##  Min.   :-50.80   Min.   :0.634   Min.   :4964   Min.   :0.0000
##  1st Qu.:-42.70   1st Qu.:1.334   1st Qu.:5099   1st Qu.:0.0000
##  Median :-41.80   Median :4.857   Median :5191   Median :0.0000
##  Mean   :-40.47   Mean   :3.587   Mean   :5165   Mean   :0.1174
```

```
##  3rd Qu.:-36.40   3rd Qu.:4.961   3rd Qu.:5228   3rd Qu.:0.0000
##  Max.   :-26.90   Max.   :5.045   Max.   :5228   Max.   :1.0000
##
```

The next graph shows the distribution of the target variable, the incidence of the target variable is `0.1174`



Now, we will look at the variable importance using a Random Forest model in order to explore the relationship of the most important ones and the target variable:

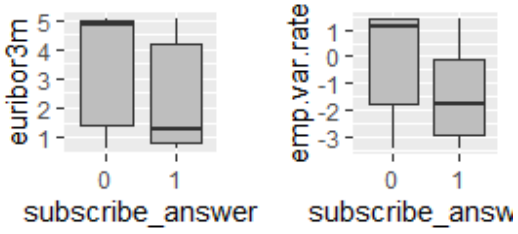
```
##          month      euribor3m cons.price.idx   emp.var.rate    nr.employed
##   0.0520432882   0.0372044316   0.0363437154   0.0296854216   0.0269201792
##  cons.conf.idx            job            age      education       poutcome
##   0.0195626072   0.0081171576   0.0071710670   0.0040330374   0.0040191164
##    day_of_week        marital          pdays        contact       previous
##   0.0027308132   0.0020486428   0.0019478786   0.0018315414   0.0015024238
##        default           loan       campaign        housing
##   0.0010423081   0.0003268686   0.0002485481  -0.0003520909
```

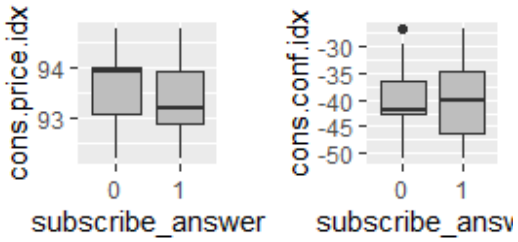
The five most important variables are month, euribor3m, emp.var.rate, cons.price.idx and cons.conf.idx. Most of these variables are related to Social and economic context attributes.

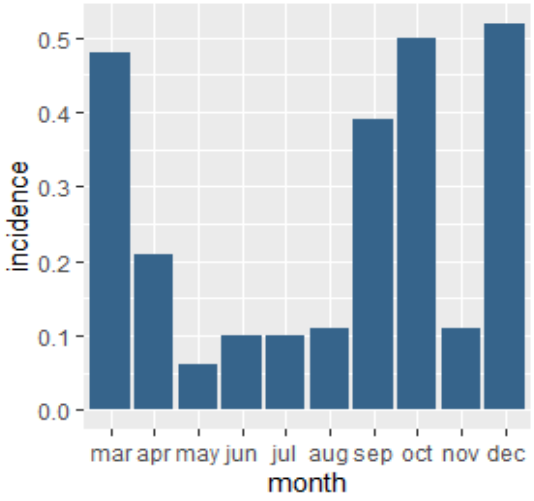
The next charts show how these variables are related to the target variable. In the scatterplot, one can observe that the target variable it does not show a clear distribution pattern within each variable. Looking at the boxplots one can conclude that when the target variable is 0 (no subscribe) the euribor3m, emp.var.rate, and cons.price.idx present higher values than when the target variable is 1 (subscribe). Both subscribe answers (0 and 1) have similar values in the cons.conf.idx variable.

Regarding the variable month, we can observe a higher incidence rate in December and October. The month with the lowest incidence is May.


subscribe
month
euribor3m
emp.var.rate
cons.price.idx
cons.conf.idx
0
1


euribor3m
emp.var.rate
subscribe_answer
subscribe_answ


cons.price.idx
cons.conf.idx
subscribe_answer
subscribe_answ



Preprocessing

In this section, I will explain the followed steps to process the data:

1. Sanity check: The dataset was clean, we did not have to fill missing values nor replacing outliers.

2. Categorical Variables: I created dummy variables for all categorical variables, also the variables job, education, month, day_of_week, age and pdays were grouped. For the grouped variables, dummy variables were created.

3. Variable transformation: All variables were scaled. This was done in order to compare each model with the same data, in some models as K-Nearest Neighbors, scaled data is important because it uses distance measures. Also, using scaled data allows the learning process to be more stable and faster.

Feature engineering

For training the model, I created some new predictive variables. As I mentioned in the previous section, dummy variables were created for all categorical variables and grouped variables for some variables. In addition, the following variables were created:

- month_spring: it refers if the last contact was made in a spring month
- month_summer: it refers if the last contact was made in a summer month
- month_autumn: it refers if the last contact was made in a autumn month
- month_winter: it refers if the last contact was made in a winter month
- age_ge_mean: it refers if the age of the client is higher or lower than the age mean of all clients. The number 1 refers to higher and the number 0 refers to lower
- no_contacted: Taking into account the special value 999 in padays, this variable refers if the client was contacted before or not this campaign.

Variable selection

For the process of variable selection I used the fisher score. Fisher score is defined as follows:

$$\text{Fisher Score} = \frac{|\overline{x}_s - \overline{x}_{ns}|}{\sqrt{s_s^2 + s_{ns}^2}}$$

With $\overline{x}_s$ and $\overline{x}_{ns}$ the mena value, and ${S^{2}_{s}$ and S_{ns}^2 the variance of a each variable respectively subscribers and non-subscribers. Typically, the 20 variables with the highest Fisher scores, indicating good predictive power, are selected. In our case the best 20 features are:

- **nr.employed:** number of employees - quarterly indicator (numeric)
- **euribor3m:** euribor 3 month rate - daily indicator (numeric)
- **emp.var.rate:** employment variation rate - quarterly indicator (numeric)
- **no_contacted.0:** client was not contacted before (dummy)
- **month.binned.aug_+*nov*+*jul*+*jun*+_may:** grouped variables of months aug, nov, jul, jun and may (dummy)
- **month.binned.misc._level_pos.:** grouped variables months dec, mar, oct, sep (dummy)
- **previous:** number of contacts performed before this campaign and for this client (numeric)
- **poutcome.nonexistent:** outcome of the previous marketing campaign as nonexistent (dummy)
- **contact.cellular:** contact communication type as cellular (dummy)
- **pdays:** number of days that passed by after the client was last contacted from a previous campaign (numeric)
- **pdays_freq_bin.[0, 4.2):** grouped variable of pdays, number of days between 0 and 4.2 (dummy)
- **cons.price.idx:** consumer price index - monthly indicator (numeric)
- **pdays_freq_bin.[4.2, 8.4):** grouped variable of pdays, number of days between 4.2 and 8.4 (dummy)
- **month.may:** last contact in may (dummy)
- **default.no:** client has not credit in default (dummy)
- **month.oct:** last contact in oct (dummy)
- **job.binned.misc._level_pos.:** grouped variable of jobs retired, self-employed, student and unemployed (dummy)
- **month.mar:** last contact in mar (dummy)
- **job.binned.blue-collar_+_services:** grouped variable of jobs blue-collar and services (dummy)
- **month_autumn.0:** last contact was not in autum season (dummy)

Methodology

In this section, I would show the different training models, their description and some results.

1. Random Forest Classifier

Random forest model is built from the decision tree model. Since decision tree is not the best model in practice because it tends to have a lot of inaccuracy in new samples (high variance), Random Forest comes to improve this performance; by combining a lot of individual decision trees that operate as an ensemble. Each individual tree in the random forest performances a class prediction and the class with the most votes becomes the final prediction.

The process of random forest takes the following steps:

First, a number of variables 'mtry' is specified (This number represents a sample of the total predictive variables in the data, in our case the 20 selected features) such that at each node, the mtry variables are selected at random out of the total number of variables. The best split of these mtry variables is used to split the node. This value is constant while we grow all the decision trees for the model.

Then, one should also specify the number of trees we want in our random forest model, and each of these trees is grown as much as possible.

After that, we fit the model and we can predict new data by giving the prediction as the class with most votes, after evaluating each new observation in each tree of the model.

Advantages:

1. Using just a decision tree tends to overfit the data and have bad accuracy performance in unseen data. The process of combining the results of different decision trees helps to overcome this problem and give better results.
2. You can use random forest models for both regression or classification tasks, so these types of models are very versatile.
3. Random forests models create an uncorrelated forest of trees, where it forces each split to consider only a subset of the predictors; with this, all the predictors may be considered and we can be sure that the trees are independent.
4. In random forest, you select the number of predictors to use, so if you use a small value, it could be helpful in cases where we have a large number of correlated predictors.

Disadvantages:

1. You need to manually choose the number of trees you will use to build the model, so you might be losing some important results by choosing the wrong number. The same thing for the number of variables to be considered at each split, however, this could be improved by using hyper-parameter tuning or cross-validation techniques.
2. Generally, you have very little control of what the model does and the decisions it makes to build the trees.

3. Compare with single decision trees, Random forests are complex, harder and time-consuming to construct. Also in terms of graphical representations, as it requires a high number of trees, the output is very unreadable.

Model Implementation

For fitting the Random Forest model, I tuned the parameters 'ntree', 'mtry' and 'nodesize', I did the cross-validation method with 10 folds to find the best Random Forest to fit in the data.

The parameter 'nodesize' was considered to control the minimum size of terminal nodes and compare different results. The default value in R for classification problems is 1.

```
# Set up cross-validation
rdesc = makeResampleDesc("CV", iters=10)

# Define the model
learner <- makeLearner("classif.randomForest", predict.type="prob", fix.factors.prediction=T)

# Define the task
train_task <- makeClassifTask(id="bank_train", data=train_fitprocessed[, -1], target="subscribe")

# Set hyper parameter tuning
tune_params <- makeParamSet(
  makeDiscreteParam('ntree', value=c(100, 250, 500, 750, 1000)),
  makeDiscreteParam('mtry', value=round(sqrt((ncol(train_fitprocessed)-1) * c(0.1, 0.25, 0.5, 1, 2, 4)))),
  makeDiscreteParam('nodesize',value=c(10,20,30,40,50))
)
ctrl = makeTuneControlGrid()
```

The best tuning was

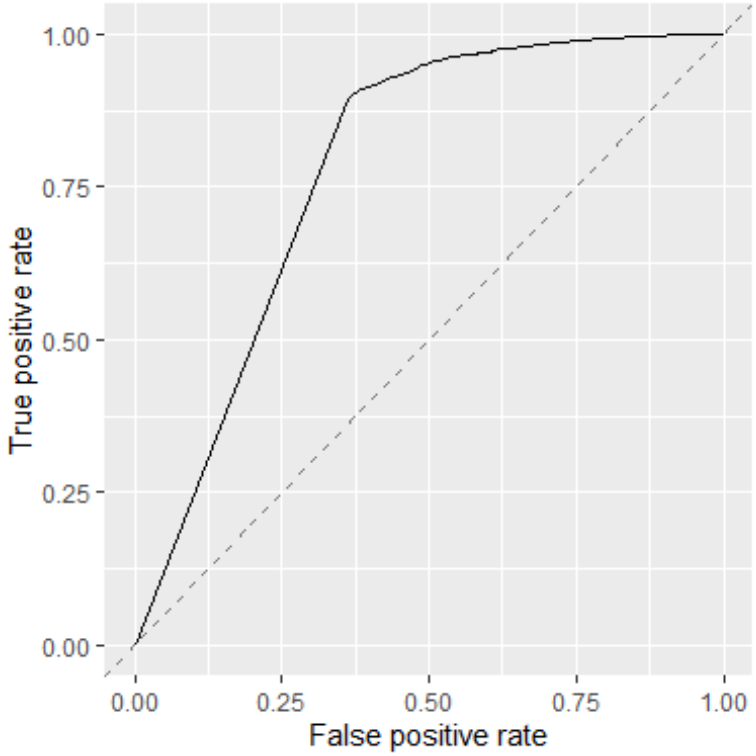
```
   Result: ntree=500; mtry=9; nodesize=40 : auc.test.mean=0.7686612
```

The confusion matrix and the train AUC are

```
##      0   1 class.error
## 0 4842 101  0.02043294
## 1  468 190  0.71124620

##       auc
## 0.7930517
```

In general, one can conclude that the model has a better performance classifying the class no subscription. The AUC train shows a good performance, but when we compare it with the mean AUC test got it in the cross-validation (0.7686), it seems that the model overfit a little the data.

Finally, we include the ROC curve of the Random Forest Model:



2. Logistic Regression

Logistic Regression is one of the most popular machine learning algorithms. Logistic Regression is similar to Linear Regression model, except that it is used to predict whether something is True or False (categorical output), instead of predicting a continuous output. In logistic regression, we fit an "S" shaped instead of a line to the data. The curve goes to zero to one. This curve shows the probability that something is True or False. If the probability is greater than a defined threshold (normally it is used 50%), we classify the observation as True (1) in other case as False (0). In general, built a logistic regression is simple.

The predictors variables can be both quantitave and qualitative. Logistic regression is given by the following equation, where each x_p represents the predictors of the model, and $p(x)$ the probability of the response variable that we are trying to predict.

$$p(x) = \frac{e^{\beta_0+\beta_1X_1+\beta_2X_2+\ldots+\beta_pX_p}}{1+e^{\beta_0+\beta_1X_1+\beta_2X_2+\ldots+\beta_pX_p}}$$

In Logistic Regression we do not used the same concept of a "residual" and least squares that is used in linear regression, instead we use a method called "maximum likelihood":

$$\iota(\beta_0,\beta_p): \prod_{i:y_{i=1}} p\,(x_i) \prod_{i\prime:y_{i\prime=0}} (1-p(x_{i\prime}))$$

The estimates β_0 and β_p are chosen to maximize this likelihood function. The maximum likelihood is the set of coefficients for which the probability of getting the data we have observed is maximum. These coefficients can also be used to explain whether there is some kind of relationship between the response variable and each one of the predictors, and also how strong is this relationship.

Advantages:

1. The logistic regression is very simple and efficient to implement. We only have to define the variables that we want to use in the model. Also after fitting the model, one can evaluate if these variables have a relationship with the target variable by analyzing the estimated coefficients and the statistics associated with them.
2. Like linear regression, logistic regression does work better when you remove attributes that are unrelated to the output variable as well as attributes that are very similar (correlated) to each other. Therefore Feature Engineering plays an important role regarding the performance of the model.

Disadvantages:

1. It can be used only for predict categorical outcomes.
2. It is not useful for resolving non - linear problems since its decision surface is linear.

Model Implementation

For fitting the logistic regression model I used a cross-validation method with 5 folds to find the best coefficients. After finding the best set of coefficients the model was fitted in the whole data training.

```
# Set up cross-validation
rdesc = makeResampleDesc("CV", iters=5, predict="both")

# Define the model
learner <- makeLearner("classif.logreg", predict.type="prob", fix.factors.pre
diction=T)

# Define the task
train_task <- makeClassifTask(id="bank_train", data=train_fitprocessed[, -1],
target="subscribe")

# Simple cross-validation
    res <- resample(learner, train_task, rdesc, measures=list(mlr::auc, setAg
gregation(mlr::auc, train.mean)))

## Resampling: cross-validation

## Measures:             auc.train   auc.test

## [Resample] iter 1:    0.7981070   0.7758514

## [Resample] iter 2:    0.7918729   0.8045532
```

```
## [Resample] iter 3:    0.7974400   0.7800658

## [Resample] iter 4:    0.7959699   0.7868902

## [Resample] iter 5:    0.7915359   0.8095670

##

## Aggregated Result: auc.test.mean=0.7913855,auc.train.mean=0.7949852

##

    best_learner <- learner
```

The model with the best performance during the cross-validation has an AUC test 0.791 and AUC train 0.794. We can conclude that the model does not overfit the data.

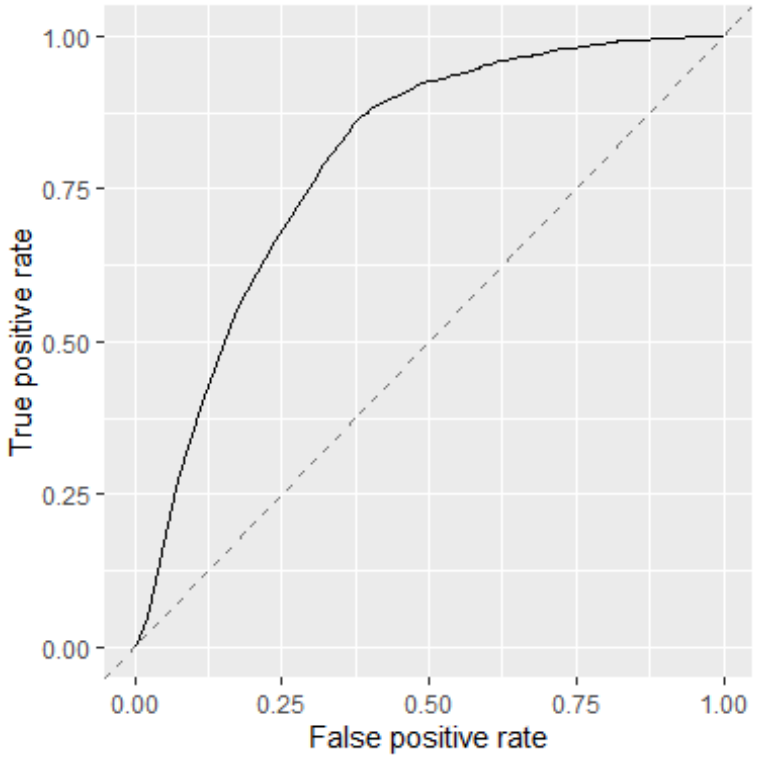
Training again the logistic regresion with the best performance in the cross-validation we show the coefficients summary, AUC train and ROC curve.

```
# Retrain the model with tbe best hyper-parameters
best_lr <- mlr::train(best_learner, train_task)
summary(best_lr$learner.model)

##
## Call:
## stats::glm(formula = f, family = "binomial", data = getTaskData(.task,
##     .subset), weights = .weights, model = FALSE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9150  -0.3915  -0.3272  -0.2716   2.7891
##
## Coefficients:
##                                      Estimate Std. Error z value Pr(>|
z|)
## (Intercept)                          -2.43499    0.05611 -43.400  < 2e
-16
## pdays                                -0.05870    0.13809  -0.425 0.670
795
## previous                              0.06525    0.07309   0.893 0.371
997
## emp_var_rate                         -1.36620    0.29395  -4.648 3.36e
-06
## cons_price_idx                        0.26963    0.13322   2.024 0.042
974
## euribor3m                             1.37342    0.36881   3.724 0.000
196
## nr_employed                          -0.83396    0.24716  -3.374 0.000
740
## default_no                            0.11590    0.06170   1.878 0.060
322
```

```
## contact_cellular                             0.34960    0.07919   4.415 1.01e
-05
## month_mar                                    0.07570    0.04165   1.818 0.069
114
## month_may                                   -0.25901    0.06957  -3.723 0.000
197
## month_oct                                    0.03870    0.04235   0.914 0.360
801
## poutcome_nonexistent                         0.16894    0.07729   2.186 0.028
826
## month_autumn_0                               0.25759    0.06233   4.132 3.59e
-05
## no_contacted_0                               0.36463    0.07573   4.815 1.47e
-06
## job_binned_blue_collar___services           -0.05166    0.05472  -0.944 0.345
091
## job_binned_misc__level_pos_                  0.08893    0.04206   2.114 0.034
483
## month_binned_aug___nov___jul___jun___may -0.04522    0.05942  -0.761 0.446
638
## month_binned_misc__level_pos_                0.06788    0.07372   0.921 0.357
101
## pdays_freq_bin__0__4_2_                     -0.03240    0.15610  -0.208 0.835
589
## pdays_freq_bin__4_2__8_4_                   -0.02683    0.09361  -0.287 0.774
372
##
## (Intercept)                                 ***
## pdays
## previous
## emp_var_rate                                ***
## cons_price_idx                              *
## euribor3m                                   ***
## nr_employed                                 ***
## default_no                                  .
## contact_cellular                            ***
## month_mar                                   .
## month_may                                   ***
## month_oct
## poutcome_nonexistent                        *
## month_autumn_0                              ***
## no_contacted_0                              ***
## job_binned_blue_collar___services
## job_binned_misc__level_pos_                 *
## month_binned_aug___nov___jul___jun___may
## month_binned_misc__level_pos_
## pdays_freq_bin__0__4_2_
## pdays_freq_bin__4_2__8_4_
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4053.7  on 5600  degrees of freedom
## Residual deviance: 3189.2  on 5580  degrees of freedom
## AIC: 3231.2
##
## Number of Fisher Scoring iterations: 6

##       auc
## 0.7949274
```

We can observe that the AUC train presents a slightly difference with the mean AUC train from the cross-validation. Also, the summary of the model shows that only ten variables have a significant relationship with the target variable.



3. Extreme Gradient Boosting

EXtreme Gradient Boosting (XGBoost) is an advanced implementation of gradient boosting algorithm. XGBoost is a more regularized model to control over-fitting, which gives it better performance than a simple gradient boosting. XGBoost is one of the fastest implementations of gradient boosted trees.

One of the major inefficiencies of gradient boosted trees is considering the potential loss for all possible splits to create a new branch (if you have a lot of features, therefore you have thousands of possible splits). XGBoost overcomes this inefficiency by looking at the

distribution of the features across all data points in a leaf and using this information to reduce the search space of possible feature splits.

The goal in XGBoost is to find an output (O_{value}) for each leaf that minimizes the next equation:

$$\sum_{x=1}^{n} L\left(y_i, p_i\right)] + \frac{1}{2}\lambda 0^2$$

Where the Loss Function in the classification problems is:

$$L(y_i, p_i) = -[y_i log(p_i) + (1 - y_i) log(1 - p_i)]$$

Advantages:

1. Comparing with the gradient boosted it provides more information about the direction of gradients and how to get the minimum of our loss function. Regular gradient boosting uses the loss function as a proxy for minimizing the error of the overall model, XGBoost uses the 2nd order derivative as an approximation.
2. In this model we can apply more advanced regularization, which improves model generalization.

Disadvantages:

1. XGBoost is sensitive to outliers since every classifier is obliged to fix the errors in the predecessors; thus, the method is too dependent on outliers.
2. It has a lot of parameters and is more complex than other algorithms as logistic regression. It requires careful tuning of different hyper-parameter, and this could be a large time-consuming process.

Model Implementation

For fitting the Extreme Gradient Boosting model I used a cross-validation method with 5 folds (in order to have a less time-consuming process) to find the best hyper-parameters. After finding the best ones, the model was fitted in the whole training dataset.

```
## Set up cross-validation
rdesc = makeResampleDesc("CV", iters=5)

# Define the model
learner <- makeLearner("classif.xgboost", predict.type="prob", fix.factors.pr
ediction=T)

# Define the task
train_task <- makeClassifTask(id="bank_train", data=train_fitprocessed[, -1],
target="subscribe")

# Set hyper parameter tuning
tune_params <- makeParamSet(
```

```
  makeDiscreteParam('max_depth', value=c(2,3,4,5,6,7)),
  makeDiscreteParam('eta', value=c(0.001,0.01,0.1)),
  makeDiscreteParam('min_child_weight', value=c(2,4,6)),
  makeDiscreteParam('nrounds', value=c(400,600)),
  makeDiscreteParam('colsample_bytree', value=c(0.5, 0.6,0.7,0.8,0.9)),
  makeDiscreteParam('gamma',value=c(0,0.1,1)),
  makeDiscreteParam('subsample',value = c(0.5, 0.6,0.7,0.8,0.9))
)
ctrl = makeTuneControlGrid()
```

The best tuning was

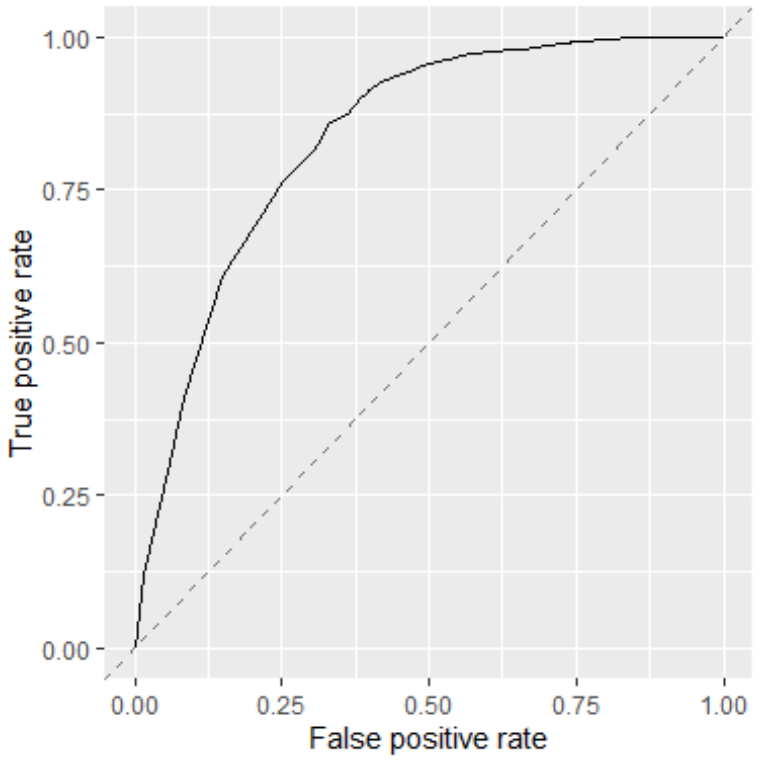
```
Result: max_depth=7; eta=0.01; min_child_weight=2; nrounds=400; colsample_byt
ree=0.6; gamma=1; subsample=0.7 auc.test.mean=0.8011255
```

The train AUC is

```
##       auc
## 0.8363746
```

In general, the AUC train shows a good performance, but when we compare it with the mean AUC test got it in the cross-validation (0.8011), it seems that the model overfit a little the data.

Finally, we include the ROC curve of the model:



4. Gradient Boosting

Gradient Boosting is an extension of the boosting method. It uses a gradient descent algorithm which can optimize any differentiable loss function. An ensemble of trees are built one by one and then individual trees are summed sequentially. The next tree, tries to recover the loss, that means the difference between actual and predicted values of the previously built tree.

The process of the gradient boosting takes the following steps:

For each observation, we should make an initial prediction that is the log(odds) convert it into a probability using the Logistic Function. Then, we measure the residuals like the difference between the observed and predicted values. After calculating the residuals, we should fit a decision tree to predict them. The outputs of these trees have to be transformed as follows:

$$\frac{\sum Residual_i}{\sum[PreviousProbability_i * (1 - PreviousProbability_i)]}$$

Then, the predictors are update by combining the initial leaf with the new tree. The new tree is scaled by a learning rate and after having the new prediction we transform it to a probability and we will have a new predicted probability for each observation. Then, we calculated the new residuals and repeat the previous steps. This process is repeated until we have made the maximum number of trees specified or the residuals get very small.

Advantages:

1. Gradient Boosting builds trees one at a time, where each new tree helps to correct errors made by the previously trained tree.
2. Supports different loss functions, since they are derived by optimizing an objective function, they can be used to solve different cost functions for classification and regression problems

Disadvantages:

1. Training a Gradient Boosting could take longer because trees are built sequentially.
2. Tunning could be harder because there are more parameters than in other models like Random Forest or K-Nearest Neighbors.
3. It is sensitive to outliers since every classifier is obliged to fix the errors in the predecessors; thus, the method is too dependent on outliers.

Model Implementation

For fitting the Gradient Boosting model I used a cross-validation method with 5 folds to find the best hyper-parameters. After finding the best ones the model was fitted in the whole training dataset. The hyper-parameters defined were ‘n.trees’, ‘interaction.depth’ and ‘shrinkage’. And they refer to the number of threes in the model, the depth of each tree and the learning rate.

```
# Set up cross-validation
rdesc = makeResampleDesc("CV", iters=5)

# Define the model
learner <- makeLearner("classif.gbm", predict.type="prob", fix.factors.predic
tion=T)

# Define the task
train_task <- makeClassifTask(id="bank_train", data=train_fitprocessed[, -1],
target="subscribe")

# Set hyper parameter tuning
tune_params <- makeParamSet(
   makeDiscreteParam('n.trees', value=c(300,600,900)),
   makeDiscreteParam('interaction.depth', value=c(3,5,7,9,11)),
   makeDiscreteParam('distribution', value='bernoulli'),
   makeDiscreteParam('shrinkage',value=c(0.1, 0.01, 0.001))
)
ctrl = makeTuneControlGrid()
```

After made the tuning we got

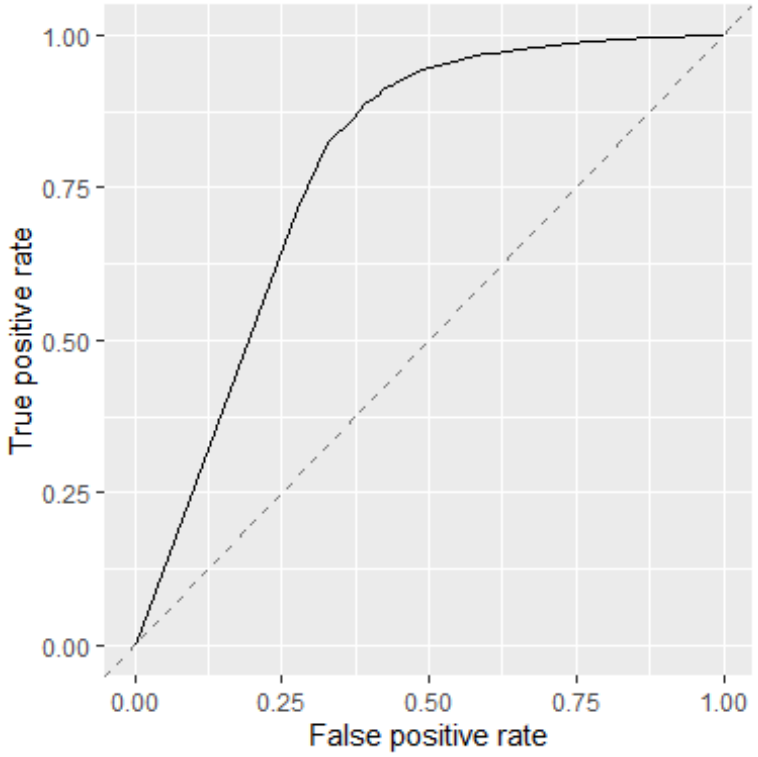
```
Result: n.trees=600; interaction.depth=11; distribution=bernoulli; shrinkage=
0.001 : auc.test.mean=0.7935094
```

The train AUC is

```
##       auc
## 0.8026587
```

This model shows a good performance, the train AUC is good and it seems that it does not overfit the data because it is close to the mean test AUC that we got in the cross-validation.

Finally, we include the ROC curve of the model:



5. K-Nearest Neighbors

This algorithm is a simple way to classify data. The KNN algorithm assumes that similar things exist in close proximity. The KNN classifier first identifies the K points in the training data that are closest to the observation x_0 represented by N_0, it then estimates the conditional probability for class j as the fraction of points whose response values are equal j:

$$Pr(Y = j|X = x_0) = \frac{1}{K}\sum_{i \in N_0} I\,(y_i = j)$$

Finally, KNN classifies the test observation x_0 to the class with the largest probability. In KNN there is no explicit training phase or it is very minimal, KNN keeps all the training data to classify the test data based on feature similarity, it measures how closely out-of-sample features resemble our training set determines how we classify a given data point.

For building a KNN model, we only have to define the K value, which is the number of nearest neighbors to define the class.

Advantages:

1. K-Nearest Neighbors algorithm is easy to implement.
2. It is useful for no-linear data because it does make assumptions about the underlying data distribution.
3. Few parameters to tunes, K and the distance metric.

Disadvantages:

1. It is computationally expensive because the algorithm stores all the training data and each new observation is classified by computing the distances between the nearest points.
2. There is not a method to define the best value of K. Low values of K can overfit the data.
3. Sensitiveness to very unbalanced datasets, where most entities belong to one or a few classes. If we set K with large values, we will tend to classify the new observations in the same category (category with more samples over the all training dataset)

Model Implementation

For fitting the K-Nearest Neighbors model I used a cross-validation method with 10 folds to find the best K value.

```
# Set up cross-validation
rdesc = makeResampleDesc("CV", iters=10)

# Define the model
learner <- makeLearner("classif.kknn", predict.type="prob", fix.factors.predi
ction=T)

# Define the task
train_task <- makeClassifTask(id="bank_train", data=train_fitprocessed[, -1],
target="subscribe")

# Set hyper parameter tuning
tune_params <- makeParamSet(
  makeDiscreteParam('k', value=c(1:30))
)
ctrl = makeTuneControlGrid()
```

The best value for k is

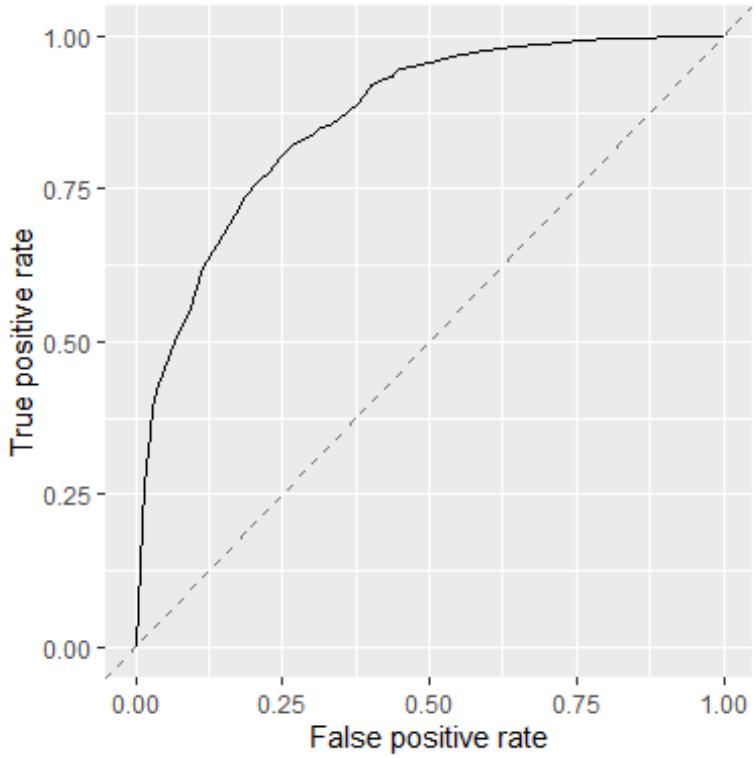
```
                Result: k=30 : auc.test.mean=0.7711269

## Loading required package: kknn

##
## Attaching package: 'kknn'

## The following object is masked from 'package:caret':
##
##     contr.dummy
```

The train AUC is

```
##       auc
## 0.8641367
```

This model shows a good performance using the train AUC, however it is higher than the mean test AUC (0.7711) calculated in the cross-validation method what suggests that the model overfit the data.

Finally, we include the ROC curve of the model:



Model benchmarking

For comparing the performance of each model the full dataset was divided into train and valid. The models Random Forest Classifier, Logistic Regression, Extreme Gradient Boosting, Gradient Boosting, and K-Nearest Neighbors were fitted in the training set and then they were evaluated in the valid set using the AUC metric.

The results showed in the previous section were gotten by fitting each model in the training dataset. In all models, cross-validation was performed and we used the same data for training and validation.

The table below shows the different performances of each model in the valid dataset:

```
##                            auc       acc
## Random_Forest        0.7425274 0.8963545
## Logistic_Regression  0.8055791 0.8949249
## XGBoosting           0.7916412 0.8956397
## Gradient_Boosting    0.8025649 0.8827734
## KNN                  0.7708996 0.8913510
```

We can observe that the models with the best performance are the Logistic Regression and the Gradient Boosting taking into account only the AUC score, regarding accuracy it seems that all of them have a similar performance. If we consider the train AUC scores, the Logistic Regression and Gradient Boosting show to be more stables than the other models.

In the submission in the Kaggle competition with the Gradient Boosting model I got in the 30% of the test holdout data set an AUC of 0.793 and in the 70% an AUC of 0.758.

In the submission in the Kaggle competition with the Logistic Regression model I got in the 30% of the test holdout data set an AUC of 0.768 and in the 70% an AUC of 0.789. It seems that the logistic regression is the model more stable.

References

Cao, X.H., Stojkovic, I. & Obradovic, Z. A robust data scaling algorithm to improve classification accuracies in biomedical data. BMC Bioinformatics 17, 359 (2016). https://doi.org/10.1186/s12859-016-1236-x

Verbeke, Wouter & Dejaeger, Karel & Martens, David & Hur, Joon & Baesens, Bart. (2012). New insights into churn prediction in the telecommunication sector: A profit driven data mining approach. European Journal of Operational Research. 218. 211-229. 10.1016/j.ejor.2011.09.031.

T. Chen, C. Guestrin, XGBoost: A Scalable Tree Boosting System, 2016

James, G., Witten, D., Hastie, T. and Tibshirani, R. (n.d.). An introduction to statistical learning.