**Charles PAYET**

# STATISTICAL MACHINE LEARNING

# INDIVIDUAL ASSIGNMENT

## Summary

Introduction ..... 2
Selected algotithms ..... 2
Logistic Regression ..... 2
Random Forest ..... 5
Decision Tree ..... 7
Gradient Boosting Machine ..... 9
XG Boost ..... 11
Setup ..... 14
Pre-Processing ..... 14
Features Selection ..... 14
Modelling ..... 14
Tuning Parameters: ..... 15
Results: ..... 17

## Introduction

We were asked to select 5 machine learning algorithms and then to explain the mechanism of the Algorithms.

In this report, we will as well setup a benchmark experiment to compare the 5 selected machine learning algorithms with description of our setup and result (cross-validation method, holdout, k-fold CV, etc.), evaluation metric (AUC, Accuracy, etc.), hyperparameter tuning, variable selection and resampling method (over-sampling, undersampling, etc.).

## Selected algorithms

- Logistic Regression
- Random Forest
- Decision Tree
- XG Boost
- Gradient Boosting Machine

## Logistic Regression

We are often interested in setting up models to analyze the relationship between predictors (independent variables) and target (dependent variable).

Linear regression is used when the response variable is continuous.
One assumption of linear models is that the residual errors follow a normal distribution. This assumption fails when the response variable is categorical, so an ordinary linear model is not appropriate.

This model is called dichotomous.

In ordinary linear regression, the response variable (Y) is a linear function of the coefficients (B0, B1, etc.) that correspond to the predictor variables (X1, X2, etc.). A typical model would look like:

$$Y = B0 + B1*X1 + B2*X2 + B3*X3 + \ldots + E$$

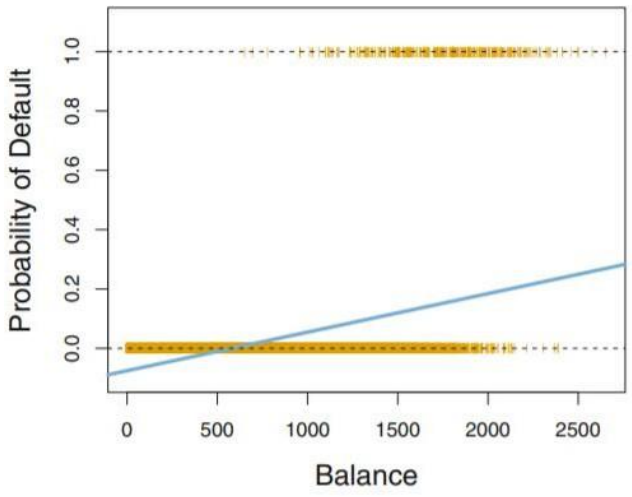
For a dichotomous target variable, we could set up a similar linear model to predict individuals' category memberships if numerical values are used to represent the two categories.
Arbitrary values of 1 and 0 are chosen for mathematical convenience.

We would assign Y = 1 if a customer subscribes and Y = 0 if he does not.

B0 is called the intercept and Bn define the slope based on the variables.

The purpose of a linear regression is to predict a value using a straight line. In fact, this method has good predicting results with linear correlated IV / DV.

However, when we are facing classification problems in which the response must 0 or 1, the linear regression does not have optimal results as we can see below:




*Linear Regression for a two level classification problem*

We can see on this graphic that linear regression have a really bad prediction score for values equal to 1. This model also predicted some values as negative, even though all values are strictly positives.

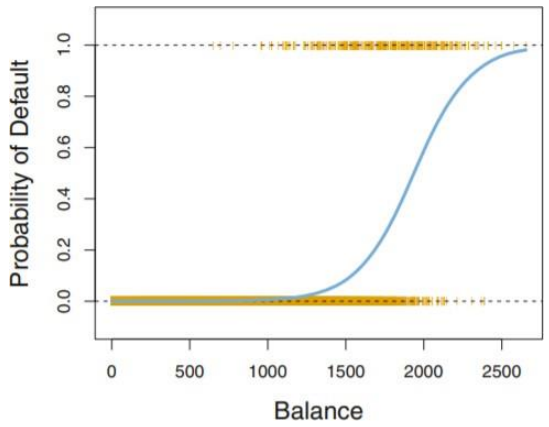
In fact, Linear Regression would not be appropriate for our model.

Instead, we use the Logistic Regression. To transform our Linear model into a Logistic model, we apply to it a logarithm function, which will transform the line into a curve fitting 0 and 1 values.

The regression equation that results is:

$$\ln[P/(1-P)] = B0 + B1*X1 + B2*X2 + \ldots$$

When we apply this function to our example we get:




*Logistic Regression for a two level classification*

This way, we always get positives prediction values between 0 and 1.

## PROS & CONS

The advantages of Logistics Regression are:

- Results are easy to understand, visually explicit
- Easily applicable to business cases, give visual insight on data
- Logistic Regression model is not only a classification model, but also gives probabilities
- Logistic Regression can also be extended from binary classification to multi-class classification

The disadvantages of Logistic Regression are:

- Interpretation can be more difficult because of the weights (multiplicative and not additive).
- Risk of complete separation : if there is a feature that would perfectly separate the two classes, the logistic regression model can no longer be trained
- Results decreases as there is a more than two level classification to predict

## Random Forest

The random forest is a supervised learning algorithm that randomly creates and merges multiple decision trees into one “forest.”

The goal is not to rely on a single learning model, but rather a collection of decision models to improve accuracy. The primary difference between this approach and the standard decision tree algorithms is that the root nodes feature splitting nodes are generated randomly.

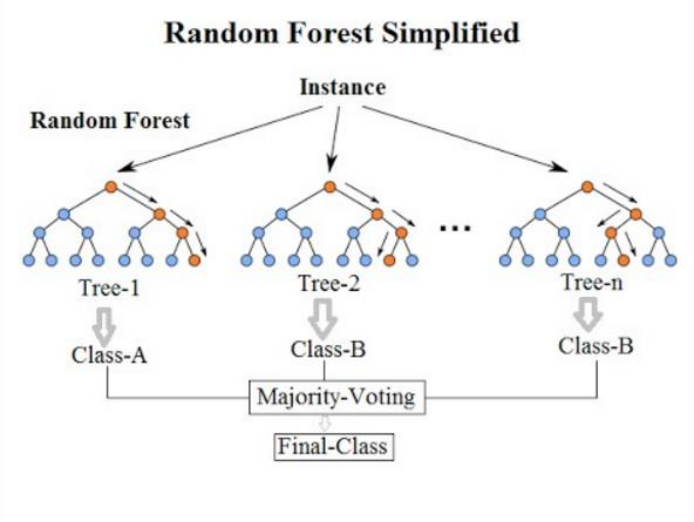
The first step is to create the random forest. The specific code various, but the general pseudocode process can be described as:

- Randomly select “K” features from total “m” features where k < m
- Among the “K” features, calculate the node “d” using the best split point
- Split the node into daughter nodes using the best split method
- Repeat the previous steps until “l” number of nodes has been reached
- Build forest by repeating all steps for “n” number times to create “n” number of trees

After the random forest decision trees and classifiers are created, predications can be made with the following steps:

- Run the test features through the rules of each decision tree to predict the outcome, then stores that predicted target outcome.
- Calculate the votes for each predicted target
- Choose the most highly voted predicted target as the final prediction

Here is a shema of Random Forest model:




*Schema of the Random Forest process*

## PROS & CONS

The advantages of Random Forest are:

- Can be used for both classification and regression tasks
- Overfitting is less likely to occur as more decision trees are added to the forest
- Classifiers can process missing values
- Classifier can also be modeled to represent categorical values

The disadvantages of Random Forest are:

- An ensemble model is inherently less interpretable than an individual decision tree
- Training a large number of deep trees can have high computational costs (but can be parallelized) and use a lot of memory
- Predictions are slower, which may create challenges for applications

## Decision Tree

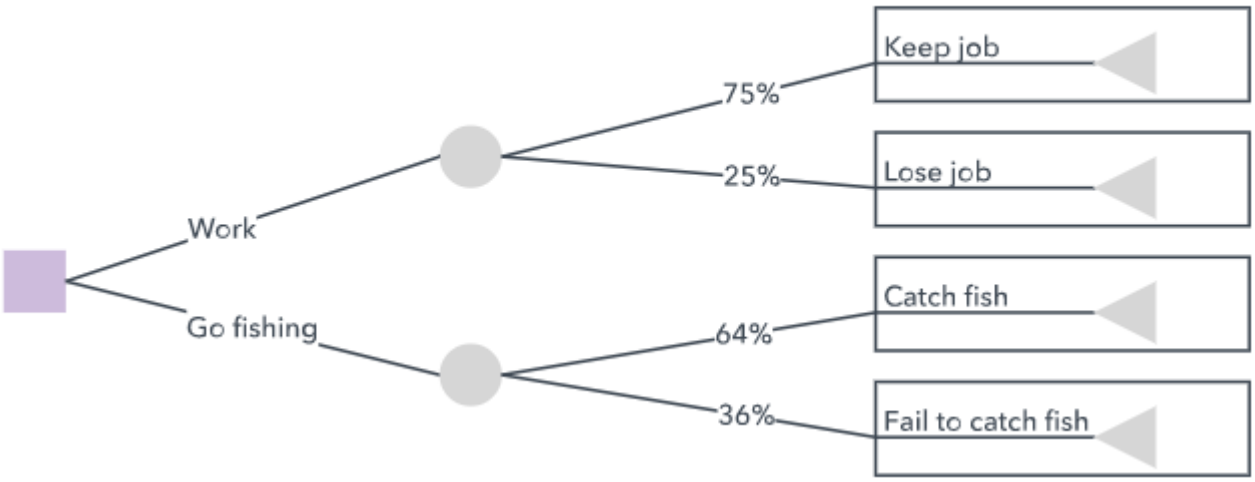
A decision tree is a graphical representation of possible solutions to a decision based on certain conditions. It's called a decision tree because it starts with a single box (or root), which then branches off into a number of solutions, just like a tree.

Decision trees are helpful, not only because they are graphics that help you 'see' what you are thinking, but also because making a decision tree requires a systematic, documented thought process. Often, the biggest limitation of our decision making is that we can only select from the known alternatives. Decision trees help formalize the brainstorming process so we can identify more potential solutions.

A decision tree typically starts with a single node, which branches into possible outcomes. Each of those outcomes leads to additional nodes, which branch off into other possibilities. This gives it a treelike shape.

There are three different types of nodes: chance nodes, decision nodes, and end nodes. A chance node, represented by a circle, shows the probabilities of certain results. A decision node, represented by a square, shows a decision to be made, and an end node shows the final outcome of a decision path.

Below is a representation of a Decision Tree:




*Schema of the Decision Tree process*

In general, Decision Tree algorithms are referred to as Classification and Regression Trees.

PROS & CONS

## PROS & CONS

The advantages of Decision Tree are:

- Simple to understand, interpret, visualize
- Decision trees implicitly perform variable screening or feature selection
- Can handle both numerical and categorical data
- Can handle multi-output problems
- Decision trees require little effort from users for data preparation
- Nonlinear relationships between parameters do not affect tree performance

The advantages of Decision Tree are:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging and boosting
- Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

## Gradient Boosting Machine

Like Random Forest, Gradient Boosting is another technique for performing supervised machine learning tasks, like classification and regression.

The implementations of this technique can have different names, most commonly we encounter Gradient Boosting machines and XGBoost.

Similar to Random Forests, Gradient Boosting is an ensemble learner.
This means it will create a final model based on a collection of individual models. The predictive power of these individual models is weak and prone to overfitting but combining many such weak models in an ensemble will lead to an overall much improved result. In Gradient Boosting machines, the most common type of weak model used is decision trees - another parallel to Random Forests.

Boosting builds models from individual so called "weak learners" in an iterative way.
In boosting, the individual models are not built on completely random subsets of data and features but sequentially by putting more weight on instances with wrong predictions and high errors.
The general idea behind this is that instances, which are hard to predict correctly ("difficult" cases) will be focused on during learning, so that the model learns from past mistakes. When we train each ensemble on a subset of the training set, we also call this Stochastic Gradient Boosting, which can help improve generalizability of our model.

The gradient is used to minimize a loss function, similar to how Neural Nets utilize gradient descent to optimize ("learn") weights. In each round of training, the weak learner is built and its predictions are compared to the correct outcome that we expect. The distance between prediction and truth represents the error rate of our model. These errors can now be used to calculate the gradient.
The gradient is the partial derivative of our loss function - so it describes the steepness of our error function. The gradient can be used to find the direction in which to change the model parameters in order to (maximally) reduce the error in the next round of training by "descending the gradient".

In Neural nets, gradient descent is used to look for the minimum of the loss function.
In Gradient Boosting we are combining the predictions of multiple models, so we are not optimizing the model parameters directly but the boosted model predictions.

Therefore, the gradients will be added to the running training process by fitting the next tree also to these values.

Because we apply gradient descent, we will find learning rate (the "step size" with which we descend the gradient), shrinkage (reduction of the learning rate) and loss function as hyperparameters in Gradient Boosting models - just as with Neural Nets.

Other hyperparameters of Gradient Boosting are similar to those of Random Forests:

- The number of iterations (i.e. the number of trees to ensemble)
- The number of observations in each leaf
- The tree complexity and depth
- The proportion of samples
- The proportion of features on which to train on

## PROS & CONS

The advantages of Gradient Boosting Machine are:

- Easy to read and interpret algorithm
- Prediction interpretations are easy to handle
- The prediction capability is efficient through the use of its clone methods, such as bagging or random forest, and decision trees
- Boosting is a resilient method that curbs over-fitting easily.

The disadvantages of Gradient Boosting Machine are:

- Sensitive to outliers since every classifier is obliged to fix the errors in the predecessors
- Very dependent on outliers
- Almost impossible to scale up (because every estimator bases its correctness on the previous predictors, thus making the procedure difficult to streamline)

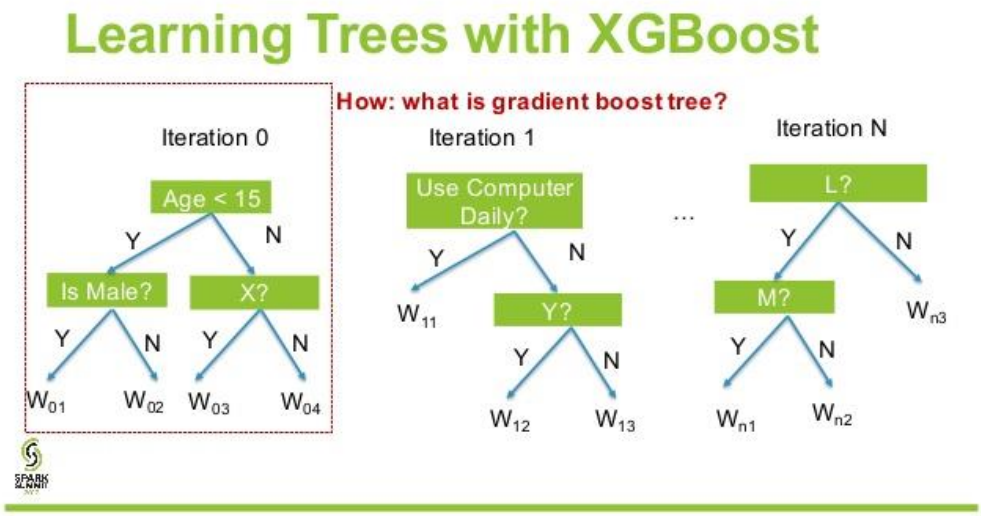
## XGBoost

The XGBoost can be defined as a gradient boosting algorithm with more specification (the name XG Boost stand for eXtreme Gradient Boosting).

Steps to create a XG Boost model could be summarized as:

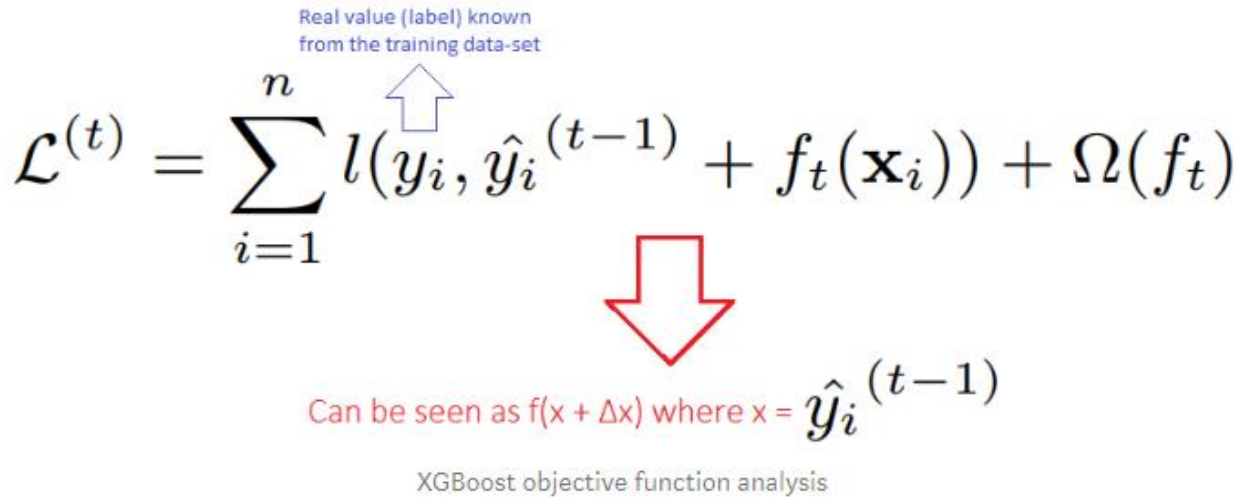
- Create a tree
- Fit the model on it
- Compute predictions

After computing the first tree, we compute a second one based on the residuals in order to adapt our model.
We repeat the manipulation many times and every time, our model is learning from his mistakes. Every time a new tree is computed, it's added into the function so it will increase the precision of the model.




*Gradient Boosting schema*

The objective function (loss function and regularization) at iteration $t$ that we need to minimize is the following:




XGBoost objective function analysis

We can see that the XGBoost objective is a function of functions. It cannot be optimized using traditional optimization methods in Euclidean space.

While decision trees are one of the most easily interpretable models, they exhibit highly variable behavior. Let's consider a single training dataset that we randomly split into two parts and use each part to train a decision tree in order to obtain two models.

When we fit both these models, they would yield different results. Decision trees are said to be associated with high variance due to this behavior. Bagging or boosting aggregation helps to reduce the variance in any learner. Several decision trees which are generated in parallel, form the base learners of bagging technique. Data sampled with replacement is fed to these learners for training. The final prediction is the averaged output from all the learners.

In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

The base learners in boosting are weak learners in which the bias is high, and the predictive power is just a tad better than random guessing. Each of these weak learners contributes some vital information for prediction, enabling the boosting technique to produce a strong learner by effectively combining these weak learners. The final strong learner brings down both the bias and the variance.

In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits. Such small trees, which are not very deep, are highly interpretable. Parameters like the number of trees or iterations, the rate at which the

gradient boosting learns, and the depth of the tree, could be optimally selected through validation techniques like k-fold cross validation. Having a large number of trees might lead to overfitting. So, it is necessary to carefully choose the stopping criteria for boosting.

## PROS & CONS

The advantages of XG Boost are:

- Extremely fast (parallel computation)
- Highly efficient
- Versatile (can be used for classification, regression or ranking)
- Can be used to extract variable importance
- Do not require feature engineering (missing values imputation, scaling and normalization)

The disadvantages of XG Boost are:

- Only work with numeric features
- Leads to overfitting if hyperparameters are not tuned properly

## Setup

We are now going to describe the process to setting up the different algorithms.

### Pre-Processing

Below are the main steps we were threw for the pre-processing steps:

- Managing null values: there was no null values in our dataset
- Create new features, based on available values and their distribution
- Cluster categorical variables to reduce the number of (useful) predictors

## Features Selection

Below are the main steps we were threw for the features selection:

- Dummy encoding of categorical variables
- Client_id is our identifier variable, subscribe is our DV and others variables are IV.
- Compute Boruta method to reduce the number of features and only keep those that are important for prediction (assign score to every selected variables to keep only the relevant). 30 were classified as 'confirmed' (relevant) and 7 as 'tentative' (not sure if it is relevant).

## Modelling

Below are the main steps we were threw for the modelling part:

- Create a pipeline using MLR package
- Compute K fold cross validation (with 10 iterations). 10 iterations is enough to get an overview of the test error rate
- Compute AUC (Area Under the Curve) which will compute the score based on TP, TN, FP and FN (confusion matrix). The closer AUC is from 1, the best will be predictions.
  We do not compute accuracy here because it would not be relevant, taking in account that more than 85% of customers did not subscribe (predicting every customer who did not subscribe would lead to 85% accuracy).
- Set up tuning parameters of Decision Tree, GBM and XG Boost.

## Tuning Parameters

XGBoost's parameters used:

- nrounds | lower=200,upper=600
- max_depth | lower=3,upper=20
    - The maximum depth of a tree, same as GBM.
      Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
- lambda | lower=0.55,upper=0.60
    - This is used to handle the regularization part of XGBoost.
- eta | lower = 0.001, upper = 0.5
    - Makes the model more robust by shrinking the weights on each step
- subsample | lower = 0.10, upper = 0.80
    - Same as the subsample of GBM. Denotes the fraction of observations to be randomly samples for each tree.
      Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.
- min_child_weight | lower=1,upper=5
    - Defines the minimum sum of weights of all observations required in a child.
      This is similar to min_child_leaf in GBM but not exactly. This refers to min "sum of weights" of observations while GBM has min "number of observations".
      Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree. Too high values can lead to under-fitting hence, it should be tuned using CV.
- colsample_bytree | lower = 0.2,upper = 0.8
    - Similar to max_features in GBM. Denotes the fraction of columns to be randomly samples for each tree.

Random Forest's parameters used:

- Ntree | 100, 250, 500, 750, 1000
    - Number of trees built by the algorithm
- Mtry | 0.1, 0.25, 0.5, 1, 2, 4
    - Number of variables tested at each division. Ntree and Mtry purpose are to minimize the Out Of Bag (OOB).

SVM's parameters used:

- C | 2^c(-8,-4,-2,0)
- Sigma | 2^c(-8,-4,0,4)
-

Some notable properties between C and Sigma are :

- With low value of C (<1) error rate increases with higher value of sigma.
- Large values of C counter balance the bias introduced by large sigma
- Very small value of sigma may result in good training data error rate but wont be useful in the case of test data recognition error rate.
- With large value of sigma the Gaussian kernel of RBF becomes almost linear.
- A stable(where error rate is almost non fluctuating) region of sigma and C values should be searched (from graph if possible) for better test data recognition accuracy.
- Number of SVs is not a reliable way of determining the 'goodness' of the classifier.

Gradient Boosting Machine's parameters used:

- Distribution | bernoulli
- n.trees | lower = 100, upper = 1000
    - Number of trees (the number of gradient boosting iteration) i.e. N. Increasing N reduces the error on training set, but setting it too high may lead to over-fitting
- interaction.depth | lower = 2, upper = 10
    - Number of splits it has to perform on a tree (starting from a single node)
- n.minobsinnode | lower = 10, upper = 80
    - The minimum number of observations in trees' terminal nodes
- shrinkage | lower = 0.01, upper = 1
    - It is considered as a learning rate

We did not use any tuning parameters for Logistic Regression and Decision Tree.

## Results

Finally, we compute our best AUC on below table:

| Model: | AUC: |
|---|---|
| Logistic Regression | 0.80124 |
| Random Forest Random Search | 0.72736 |
| Decision Tree | 0.70837 |
| Support Vector Machines | 0.65100 |
| Gradient Boosting Machine | 0.77700 |
| XG Boost | 0.79580 |

Looking at the results, we can conclude that Logistic Regression and XG Boost outperformed the others algorithms in term of AUC and should be chosen to predict our IV variable 'subscribe' in this dataset.