



STATISTICAL & MACHINE LEARNING

Individual Assignment



IESEG
Sébastien PAVOT

Contents

Introduction:	2
Part 1: Algorithms presentation:	2
Logistic Regression:.....	2
Linear Discriminant Analysis:	5
XGBoost:.....	7
Random Forest:.....	9
Support Vector Machines:	11
Part 2: Model Setup and Results:.....	14
Pre-Processing:	14
Features Selection:.....	14
Modelling:	15
Metric:.....	15
Tuning Parameters:.....	16
MLR Process:	18
Results:.....	19

Introduction:

In this report, we are going to focus on five algorithms. We will in a first time describe them, how they work what are their strengths and weaknesses and why we choose them in the case of this Kaggle competition, which is to predict, or not if a customer is going to subscribe to a telemarketing campaign. Then, we are going to dig in the second part where we are going to summarize all the work done in the coding part and explain the method we used to build the models in order to increase the performance of prediction.

Part 1: Algorithms presentation:

We selected five algorithms for this work. These are the following:

- Logistic Regression
- Linear Discriminant Analysis
- XGBoost
- Random Forest
- Support Vector Machines

Now, we are going to dig in more detailed explanation of each algorithm:

Logistic Regression:

The logistic regression is an algorithm that is used in classification problems. It is based on the linear regression, which has the following form:

$$y = B_0 + B_1x + \dots + \epsilon$$

With B_0 , the intercept and B_1 to B_n that define the slope based on the variables. The goal of a linear regression is to predict a value using a line. This method appears to work very well to predict value especially if the variables and the dependent variable have a linear correlation. However, in a case of a classification problem where we want to predict either it is one or zero, the linear regression will perform poorly as the graphic bellow show:

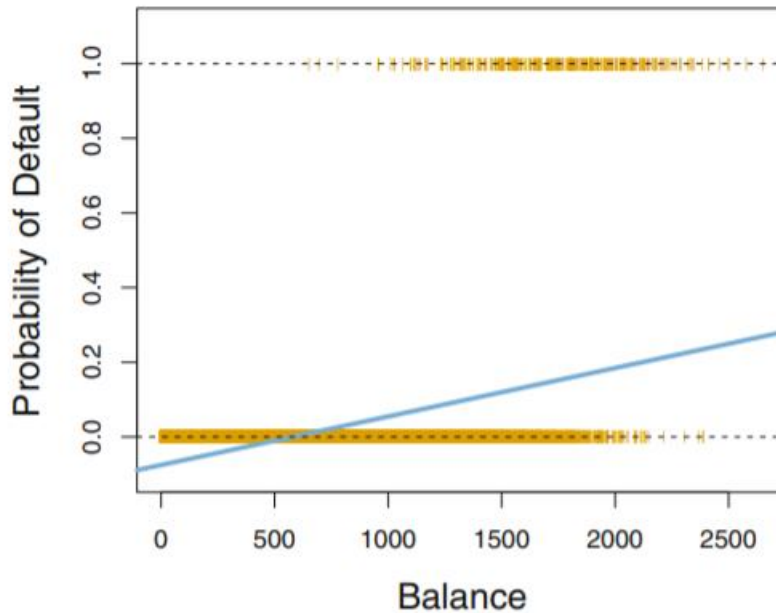


Figure 1: Linear Regression for a two level classification problem

As we can see, using a linear regression only lead to predict at maximum a probability of 0.4 to be 1 where in fact we have a lot values that are one. Using a divider like 0.5 and above probability are classified as one and the others 0, here we will predict zero values as zero. Moreover, we even predict negative probabilities where it is impossible, as the probability have to be between zero and one. This is why we use the logistic regression. The logistic regression is the same thing as the linear regression but we apply a log function to the parameters of the linear regression to make sure that probability will fit between zero and one and as the line transform into a curve that will more fit the data.

The equation for the logistic regression is:

$$P(X) = \frac{e^{B_0 + B_1 X}}{1 + e^{B_0 + B_1 X}}$$

As we can see, we divide the log function applied to the basic function of linear regression in order to always obtain a value between 0 and 1. Now, if we try to plot the function above with the same data as we did with the linear regression, we obtain:

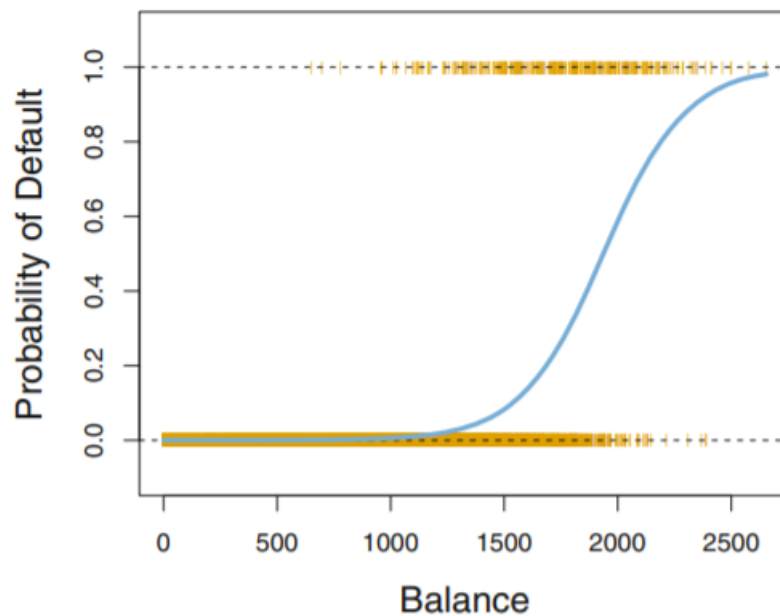


Figure 2: Logistic Regression for a two level classification

We see that now the curve is always between 0 and 1 and prediction are now possible as 1 or 0 as probabilities are contained between 0 and 1 as the difference with the logistic regression where probabilities were contained between -0.1 and 0.4 .

Now that we have seen how is constructed the Logistic Regression and the purpose of the function, we better understand why it is way more powerful than linear regression in a classification problem. The advantages of that method is that such as the linear regression, the results are easily understandable. One can use summary function in order to obtain coefficients, standard error, P value and other information in order to produce insight on which parameters has the most importance. As it is a very understandable model, it has the advantage to be useful in business case as it produce good results and remain understandable in order to give insights where non-parametric method such as the KNN classifier will not be interpretable. However, as all method, the interpretability come with a cost. Some algorithms might perform better using more deep statistical techniques. In addition, the linear regression is very useful when predicting a two level classification problem but when the number of groups to predict is above two, the logistic regression is not anymore the most useful. One last point is that research have been made that the logistic regression tends to have more trouble with many predictors and especially if some are correlated.

Nevertheless, as a conclusion, this model by his interpretability and his efficiency is one the most famous classifier algorithm used today.

Linear Discriminant Analysis:

Let us now discuss about the Linear Discriminant Analysis. This algorithm tends to be popular when we need to predict more than a two level classification problem. However, even for a two level classification problem as the one that we have for the telemetry banking dataset, the linear discriminant analysis has advantages explaining why we choose it. First of all, if the classes are well separated, the LDA will tend to be more stable than the logistic regression. In addition, if the number of observations is small and the distribution of predictors is approximatively normal, the LDA will one more time be more stable.

Let's start by explaining what is the Linear Discriminant Analysis:

First, we need to remind the Bayes classifier is the classifier that will have the lowest possible error rates out of all classifiers. But in reality, we don't know the Bayes Decision Boundary (which is equal to the prediction where we will have the lowest possible error rate). This is why we use LDA because we can't compute the Bayes Decision Boundary if we don't know it.

The LDA assume that that all the predictors are drawn from a Gaussian distribution which is the fact that each individual predictor follow a one dimensional normal distribution with some correlation between pair of predictors. The goal of the LDA is to approximate the Bayes Classifier by plugging estimates into the Bayes parameters of its function. Without going to deep in algebra, we can say the goal of this function is to approximate the Bayes Classifier by adding some estimators based on the mean of observation, weighted average of the variance foe each class. If we take an example, we can plot the in a dataset where the Bayes Decision Boundary is known the Bayes Decision Boundary and the predicted separation made by LDA for a three level classification problem:

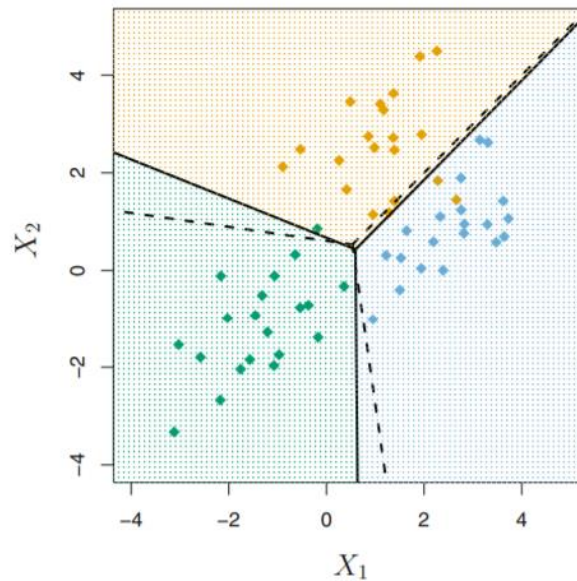


Figure 3: LDA and Bayes Decision Boundary for a three level classification problem

Here, we can see the dashed lines as the Bayes Decision Boundary, which is the lowest possible error rate. We can see that by approximating the Bayes Decision Boundary, the LDA (as solid blacked line) perform pretty well and nearly approximate the best possible decision boundary.

The advantages of LDA is that a stable model and that it can reach really high performance especially if the variable follow nearly a Gaussian distribution. It is a very useful algorithm as it can be used for two or more classification problems. The disadvantages of this method is that as it try to approximate the lowest possible error rate which is the Bayes Decision Boundary, in some cases it might be not useful. For example, the Bayes Decision Boundary don't care about the importance or the distribution of the groups, it just gives the lowest possible error rate. Therefore, if you are creating a fraud detection algorithm and only 2% of all the requests are fraud, the Bayes Decision Boundary can predict the best lower error rates but will not be the best for the business. In other words, misclassifying a fraud can lead to way more loss of money than the classifying more fraud people than there is in reality. In this case, it will increase the error rate but will decrease the cost. Moreover, as the LDA try to perform the lowest possible error rate following the Bayes Decision Boundary, it will maybe misclassify some fraud and will lead to way more loss of money than will do another algorithm.

To conclude, this is a pretty efficient with a range of applications in the real world but the LDA suffers from it sensitivity, which might make him not usable in some cases as, explained above.

XGBoost:

We've seen two algorithms previously, now we will to another world of algorithms. The tree based algorithms. Tree based method are really famous over the world as they are easy to understand and appears to work like human do. In a few words, for each predictor we define two possibility which can be for example having more than 100€ or less and then we regarding the amount of money you have the prediction either go in more than 100€ or less. Tree based method are useful and work most of the time very well but they tend to overfit the data. A way to compensate that is to use a Gradient Boosting Method. The XGBoost is no more than a gradient boosting algorithm with more specification. The goal of gradient boosting is simple. You create a tree, fit the model on it and compute prediction. As with all models, prediction are never 100% accurate and so you obtain residuals. Here, 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.

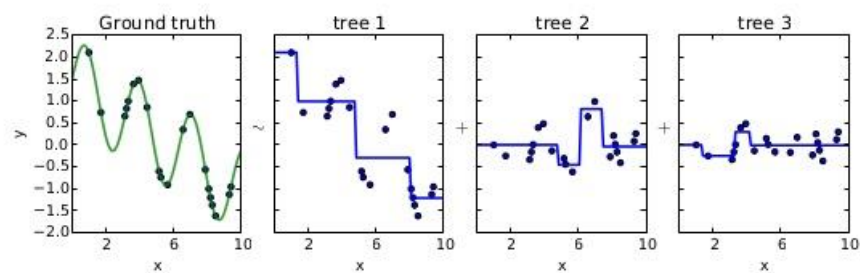
Let's take a visual example to explain it:

Gradient Boosting [J. Friedman, 1999]

Statistical view on boosting

- \Rightarrow Generalization of boosting to arbitrary loss functions

Residual fitting




 `sklearn.ensemble.GradientBoostingClassifier|Regressor`

Figure 4: Example of the steps of a Gradient Boosting algorithm

Here, we can see on the left the true function that define the data. Let's check the first tree. We see that the algorithm perform tree decision over the data and we obtain a first model. Nevertheless, as explained before, we still have residuals. It is why now on the second tree we don't perform a tree anymore on the data but on the residuals of the previous fit. It is still remaining some residuals so the residuals of tree two are used to fit the third tree that we can see on the right. At the end, we add all those three trees together as a fitted function in order to obtain a better prediction as every iteration (here three), we improve our model by learning from the mistake of the previous one.

This algorithm is an algorithm, which contains parameters meaning that difference from logistic regression for example; we can choose the way the algorithm is working. Basic gradient boosting contain three tuning parameters. The first one is the number of trees, which means the number of time we're going to compute a new tree based on the residual. The example above has a tuning parameter of number of trees equal to three as we computed three tree. We might think that more tree we will have, better the prediction will be but it's not the case as we know the risk of overfitting by the variance / bias trade off. The second parameter is the shrinkage parameter, which consist of defining the rate at which the gradient boosting learns. Finally, the number of splits in each tree is the last and third parameter which control the complexity of the boosted ensemble. This parameter has to be defined regarding the number of variables as with the maximum of the number of variables you have. Indeed, if you have a dataset with 10 predictors, you can't compute more than 10 splits as for each split you need a variable to split. So has the example above with 100€, here we computed one split on our tree so the parameter will be one as we only have one variable. Even if you have a lot of variables, computing the maximum of splits possible is not the best idea as one more time there's a high risk of overfitting.

The XGBoost is a gradient boosting algorithm but only differs as it offers more tuning parameters than a simple gradient boosting. We're not going to discuss all the different parameters but we used this one instead of a classical gradient boosting as during our pipeline we can test many parameters it's still a good idea to be able to try many different version of the model but this part will be more discussed in the section two of this report.

Advantages of this algorithm is that it provide a good alternative to basic tree that tend to overfit. Also, by learning from it mistake it has the advantages to produce very good prediction in most of the case. As a tree based method, it is still remain understandable while the number of trees used isn't too high. Finally, the tuning parameter allow us to play with the model in order to try to improve it and make it more flexible as you can adapt the parameter for each type of problem you have. Disadvantages of this method is that it is sensitive to outliers as it is learning from residuals so a good data processing is needed before in order to avoid those type of problem. In

addition, it might be hard to scale up those type of models as it is learning from previous mistake. Thus, the procedure is difficult to streamline.

Random Forest:

We've seen a tree based method just before which was the gradient boosting. Now, let's focus on the Random Forest which is another tree-based method. We saw that Gradient boosting is learning from the previous tree computed and so on. Here the approach is different. The difference consist of the way that computing the tree. We saw that the gradient boosting is computing one tree and based on this one another one and another etc... Here, the Random Forest compute all tree at the same time and they aren't linked together. The schema bellow explain that:

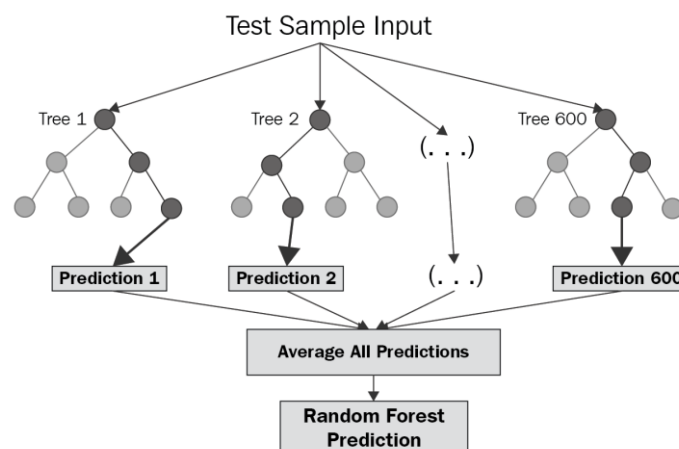


Figure 5: Schema of the Random Forest process

The Random Forest create many trees as the same time, each tree predict an outcome. When all the trees computed have done their prediction, the algorithm do the average of all the prediction and then take the prediction that is in majority for a classification problem. For example, if over 100 trees, 30 predicted 0 and 70 predicted 1, the final decision predicted by the Random Forest will be 1. Just like a democracy. However, how are computed the trees? They are computed as they will all use different data based on a bootstrap method. In few words, bootstrap will mixed the data in order to create subset of the data as many times as the number of trees computed. This method basically make that each tree is different from the other one. Indeed, computing 100 times the same tree will lead to always the same prediction so using different trees leads to diversity to train the model. This method is called a bagging method. But the Random Forest algorithm go beyond this point. Indeed, it don't stop at just creating many trees based on subset

of the data and then compute prediction based on average prediction. The Random Forest compute tree with a subset of data but also with a subset of predictors. For example, a tree computed by the random forest using a dataset of 15 variables will only use 5 random predictors overall predictors. The reason behind this is that in some dataset, some predictors might be really strong over the other one, and if we use all predictors on all the trees computed with different subset of data created by the bootstrap method, it will always lead that the predictors with an high importance will take the decision and will hide other predictors that have less importance. By doing this, the Random Forest take also in consideration predictors with less weight in the final decision by not selecting all variables for each tree. Thus, as all tree can be correlated because of high weight predictor, computing the mean of the trees will not lead to decrease the variance which is the main goal of the bagging method. Random Forest dodge this trouble by using the method just explained before.

The advantages of a Random Forest method is such as Gradient Boosting you can tune some parameters like the number of trees which make it really flexible. Also, this algorithm reduce a lot the variance that can be present in some dataset and so will produce in most of the times really good prediction. As a tree based method, it's still can be interpretable as the number of trees isn't too high. A disadvantage is that most of the time, you need a high number of trees to obtained good prediction and the interpretability is lost. This is all the dilemma of the trade of performance versus interpretability. In addition, a Random Forest have some risks to overfit the data. Finally, such as gradient boosting, a high computational algorithm make it slow to run compare to logistic regression. This is why in some cases we will prefer a logistic regression method as it faster, interpretable and in some cases can approximatively reached the precision of a Random Forest algorithm.

As a conclusion, Random Forest is a really great algorithm that can be used in many cases with good performance. But as all more evolve algorithms, it paid the price of interpretability and speed which will make it unusable in some business cases.

Support Vector Machines:

We explained different types of algorithms, now we're going to present another type which are the support vector machines. We choose this algorithm because it's classification problem algorithm and has another approach than the previous algorithms that we've seen. The principle is pretty simple, the goal is to draw a line (if we plot in two dimensions) which divide the data into two groups as that the first one consist with all observation that are 0 for example and on the other side of the line the observation that are equal to 1. It look like this:

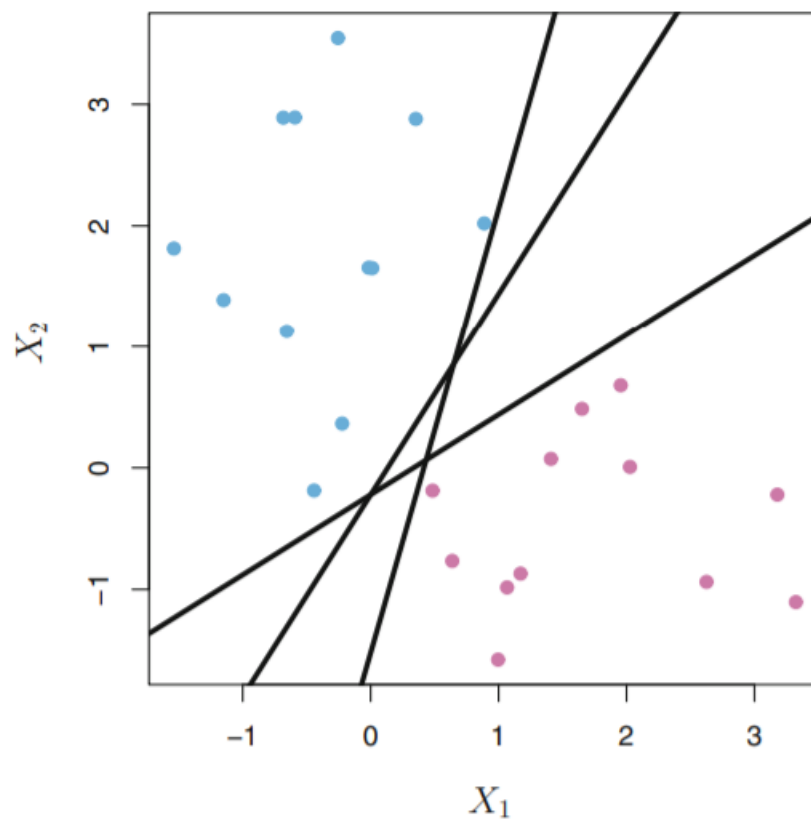


Figure 6: Different lines that perfectly split the data

Here we can see that we created few different line that perfectly split the two type of data. This is the main principle of this algorithm. But we can worry about which line to use as there's an infinity of line that can be drawn to split the data. In order to fix this trouble, we use the maximal margin classifier. The goal of this is to draw a line that perfectly split the data as the one above, but is also the furthest away from each observation group. Thus, we will be able to find the best line to split the data as this one will be as far as possible from the observations of two class so it create two margins as the plot show bellow:

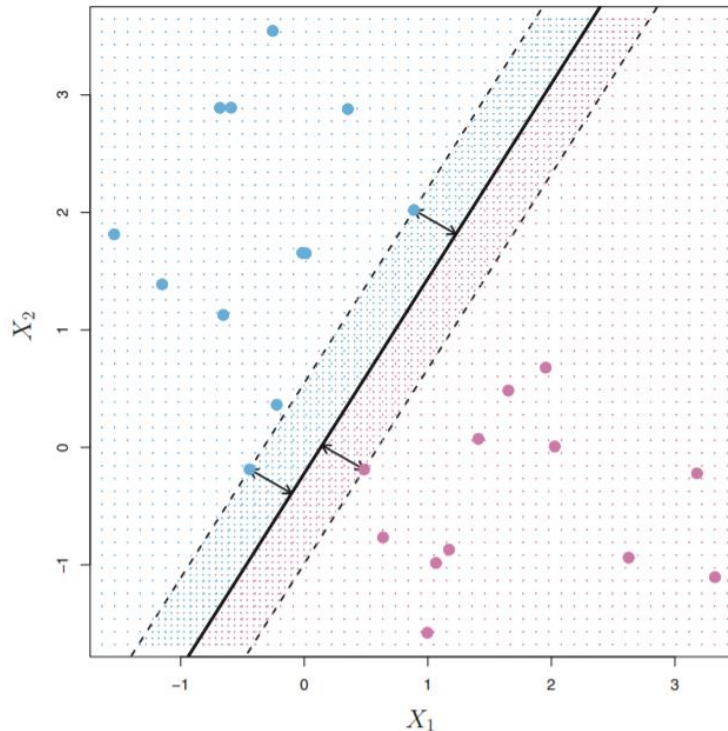


Figure 7: Maximal margin classifier

Here, we can see that we draw the line that is as far as possible from each group of observation while still perfectly splitting the data. The two-dashed line consist of the margin. The goal is to compute the margin that are as far as possible from the line so more the margin are far from the line, more the group are well split and so the prediction will be more accurate. Since the beginning, we assume that the data can perfectly be split by a line, but in reality it's never the case and so all the method explained above can't work. This is where the support vector machines take place. Based on the method explained before, the SVM algorithm work the same way but with one main difference: it allow some point to be misclassified. This possibility allow the algorithm to avoid the problem of a non-possible way to perfectly split the data. Moreover, it add to the algorithm robustness because, taking the example above, if we add one more point, the line and margin computed can consequently change as there's a new better way to split the data. SVM also contained tuning parameters. The first one is C which is a value that start from 0. If $C = 0$ then, we do not allow any points to be misclassified, but if $C > 0$ we start allowing some points to be misclassified. The goal is to find the best value for C as we will get the better prediction without avoiding to have too many points in the wrong side of the line. Furthermore, we can add some parameters to the SVM in order to make the splitting line not anymore straight but smooth such as polynomial kernel. We will not dig into here but it's a possibility to split the data in other way as show the example bellow:

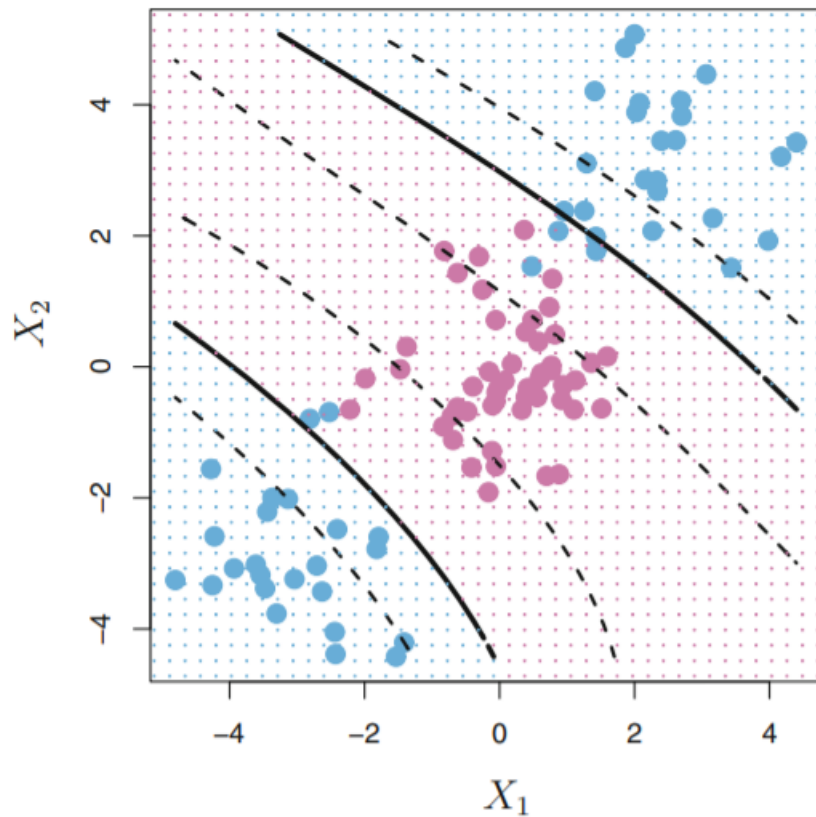


Figure 8: SVM with polynomial kernel of 3 degrees applied

The advantages of the SVM are that it is accurate in a high dimensional space and that they use subset of training point in the decision function. But the SVM are known to overfit the data. Moreover, as we show, they can't provide probability which can be a very useful information when for example we want to compute cutoff point. Finally, it can be very slow to compute with a lot of data.

To conclude, SVM is another type of algorithm, which can be really efficient but has to be used in special cases. For example, SVM are mainly use in image recognition or for text mining. But the algorithm suffer from some disadvantages discussed above which make it not very useful for a lot of situations.

Part 2: Model Setup and Results:

We've now seen and describe all the algorithms we used in this Kaggle competition in order to predict the subscription or not of a customer based on a telemetry banking data. In this part, we're going to explain the process to setup the algorithm with before a brief explanation on the pre-processing part.

Pre-Processing:

The dataset we got was a clean dataset, we didn't had any null values. What we did is that we created some new features based on the current variables and their distribution. Also, we grouped some categorical variables together in order to reduce the number of predictors in our dataset. All the detailed process are describe in the R Jupyter Notebook. We will now focus on the features selection part.

Features Selection:

After dummy encode categorical variables and create new one, we got a dataset with 63 predictors, subscribe column as our dependent variable and client_id as our identifier column. The fact is that we were not sure that all of the 63 predictors were relevant, moreover some might be highly correlated and could be useless to have two predictors correlated instead of one. Thus, we decided to compute the Boruta method in order to reduce the number of features. This method compute a score on this relevance of each variable and then return the variables that are really important for us. By doing this, we obtained 30 predictors that were categorized as "confirmed" which means that the Boruta method think that they are relevant. We also got 7 variables classified as "tentative" which means that the method isn't certain about their usefulness or not. In this case, we had the choice to either select manually if we wanted to keep them or to let the method choose. As we don't really know all the meaning of the features and as we are missing context, choosing empirically appears to relate only to luck this is why we let the method filter those. We end with 34 variables selected by the method where we add the column subscribe as the dependent variable and the client_id column as our identifier.

Now that we have our final dataset with the selected features, we're going to in the modeling part.

Modelling:

In order to create a pipeline to build the model, we used the MLR package from R. As we explained above, the first two algorithms selected, Logistic Regression and Linear Discriminant Analysis don't have any tuning parameters. But in order to analyze their performance, we used the K fold cross validation technique with 10 iterations (or 10 folds). As a reminder, the cross validation consist of fitting the data on let's say 80% of the data that we have and predict on the 20% remaining the dependent variable. We choose to use 10 iterations as some research show that it's enough to have an idea of the test error rate. Indeed, the predicted test error rate will not be the same that the one in reality but using 10 iterations of the cross validation method, we can approximatively predict the real test error rate. So for the two first algorithm, we used the cross validation technique in order to analyze the performance of our model.

Metric:

On the way to analyze the results, we use the AUC (Area under curve) as the accuracy isn't the best metric for this model. Why ? Because over the training set, we have 6178 non-subscribers and 822 subscribers which means that 88% of the customer didn't subscribe. So computing accuracy isn't the best way as if we predict all people to be non-subscribers, we would already get 88% of accuracy. Instead, the AUC will perform a better way to predict as we will get the score based on true positive, true negative, false positive, false negative. For reminder, the AUC is computed using the confusion matrix as show bellow:

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 9: Confusion matrix

Based on this matrix, we can compute the ROC curve as bellow and calculate the AUC:

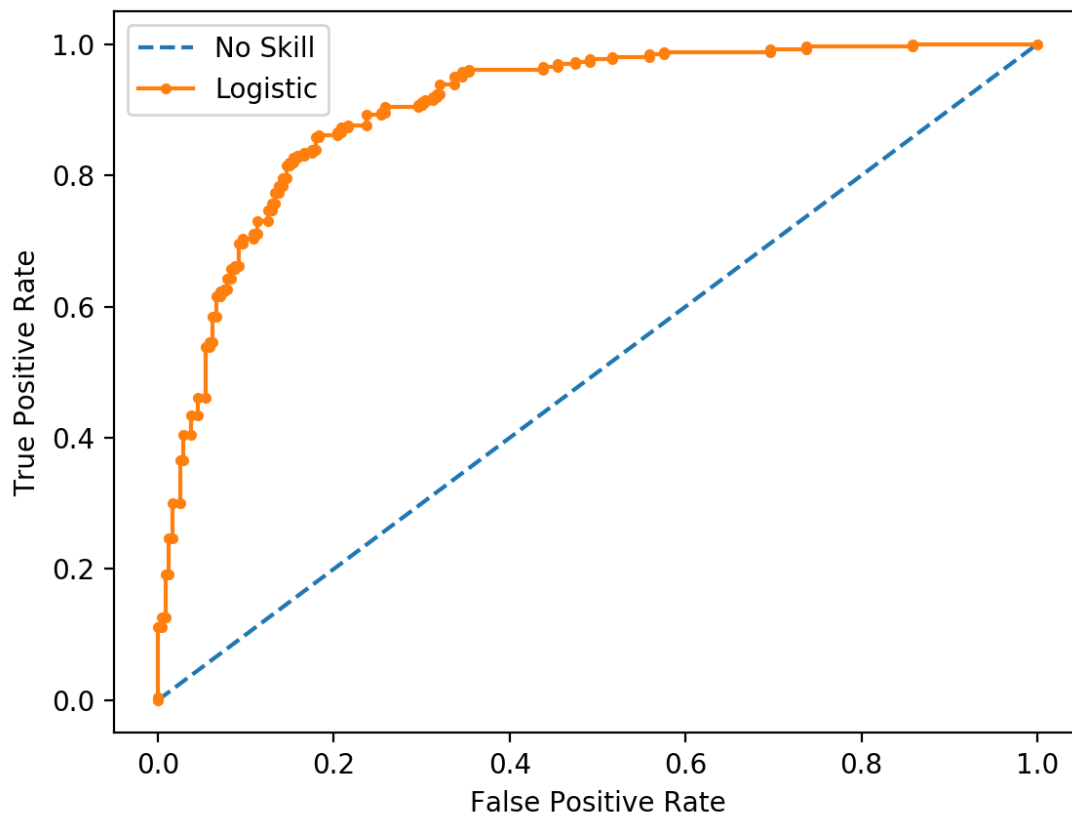


Figure 10: ROC curve and AUC

The AUC is a value between 0 and 1 which is computed by the area under the roc curve which is in orange for the example above for the logistic regression. As AUC is close to 1, better the predictions and the models are.

This is why we choose this method.

Tuning Parameters:

As we discussed above, the two first algorithms didn't needed any tuning parameters. But for XGBoost, Random Forest and SVM, we can tune some parameters.

Let's first focus on the two tree based method. For XGBoost, we mentioned before the parameters of the gradient boosting algorithm. XGBoost comes with other parameters that we used:

- Nrounds
- Max_depth
- Lambda
- Eta
- Subsample
- Min_cild_weight
- Col_sample_bytree

Those parameters are the one we used for the XGBoost. We are not going to explain each of the parameters as we already explain some before but we will explain the process we made.

For the Random Forest algorithm, we used the following tuning parameters:

- Ntree
- Mtry
- Nodesize

Once more, we will not explain the detailed meaning of the parameter but the approach of the tuning parameters.

For those two algorithms, we used two different approaches. The first one is using randomness. We assign to each parameter a range of value (for example 1 to 100) and we assign to the grid control which is the parameter to define how many times the model has to try parameter a number. For XGBoost and Random Forest, we tried over 250 times for each model to fit this model with the parameter randomly selected in the range of value defined. This approach has the advantages to be able to find by randomness a good mix of features that can lead to an improvement of the AUC. But this approach need a lot of computation and might not always be efficient.

The other approach that we also used for SVM is the selected parameters. Here, we defined in each parameters a number of possible value (like 1-5-10). In this case we don't define the number of iterations as it is define by the number of possible model regarding all different possibilities in the feature. This approach is faster and can lead to great performance as we can specify by ourselves the best parameter we think. The only cons of this method is that we can maybe miss a better model as we are specifying specific values.

For the SVM, we used C and sigma as our tuning parameters.

MLR Process:

To explain the process of MLR, it's pretty easy. We explained above the two ways of selection of tuning parameters. Since we specified the parameters, the MLR package process the following way. If it's a random selection way, he select randomly parameters within the specified range, fit the model and compute the AUC using the K-fold cross validation with 10 folds.

NOTE: We used a 10 K fold cross validation for all of the 5 algorithms.

After computing the AUC average over the 10 folds with the specific parameters, the model try another mix of parameters values and one more time return the average value. The process stop when it reach the number of iteration specified. In the case of parameters that we specified, it's the same as above but this time the process stop when he tried all possibilities. We say that random selection need a lot of computation because for 250 iterations, the algorithm need to fit 250 different models with each time do a 10K fold cross validation which return in fitting 2500 times the model for 250 iterations.

At the end of the process, the MLR package retain the model that had the best AUC over the 10K fold cross validation. It keep the parameter and so we obtain the model that perform the best regarding the different tuning parameters we set. After this, we use the best model to compute predictions on our predict set and it is done.

As a conclusion, we describe all the process for the modelling part that we use in order to perform the model with the best score. Here, the goal is only to get the best AUC for the Kaggle competition but in real life more parameters have to be taken into account making that the algorithm that perform the best might not be the selected one at the end.

NOTE: At the end of the notebook, we tried some bagging predictions of different algorithms. These are not describe here but some comments will be available in the notebook.

Results:

Model:	AUC:
Logistic Regression	0.7904695
Linear Discriminant Analysis	0.7854110
XGBoost Random Search	0.7970922
XGBoost Grid Search	0.7954835
Random Forest Random Search	0.7859184
Random Forest Grid Search	0.7863417
Support Vector Machines	0.7312467

Here are the results we obtained on the test set validated with 10K folds cross validation. Regarding the results, we can clearly see that the Gradient Boosting, either with a Random Search parameters or with specified parameters tuned outperformed the others algorithms in term of AUC. The interesting part is that the Random Search parameters approach produce better results than the specified one showing that by randomness, we can approach the best parameters for a model.

However, if check, even if support vector machines performed poorly, others algorithms nearly achieved the same AUC as the Gradient Boosting. Especially for the Logistic Regression which is on the podium without being a really complex model. This can lead to us to think what would be the point to use a gradient boosting algorithm in a business environment where it only lead to a slight increase of performance, but we can't interpret the results and the importance of features in decision. Also, reminder is to compute the Gradient Boosting with such good parameters, it took around 2 hours to try all different parameters in order to output a good mix of them. One can logically choose the Logistic Regression as the performance are really close to the XGBoost score but has the advantage to be fast and interpretable. Nevertheless, here, in a Kaggle Competition where only the results matter, we would choose the XGBoost model.