A Comparative Study of Classification Algorithms for predicting Term Deposit Status of a Bank's Customers

(Project of MTH552A)

Submitted by,

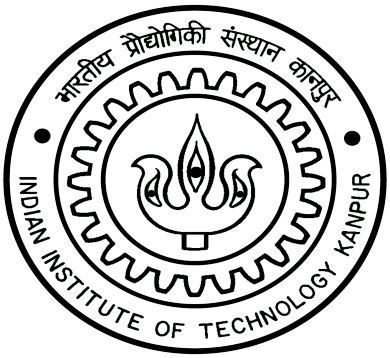
Rajdeep Saha (201380)
Soumik Karmakar (201428)

Supervised by,

Dr. Amit Mitra

Submitted on,

22^{nd} April, 2022



Contents

1 Introduction

In classification problems, we have to classify whether a particular observation will fall into a particular category using suitable classification algorithms. For example, we may have to predict based on certain diagnostic measures a patient is diabetic or not, a tumor is malignant or benign, whether a credit card will default or not. There are several classification algorithms out of which we will make a comparison study between **Logistic Regression, Decision Tree, Random Forest, AdaBoost** and **Gradient Boosting**.

2 Data Description

To carry out our analysis, we have taken **Portugeese Bank Marketing Dataset**, which can be found here. The data is associated with direct marketing campaign of a Portugeese Banking Organisation. The dataset consists of 49732 rows and 17 columns. The variable description is as follows,-

- age: age of a person.
- job:type of job of a person.
- marital_status: whether a person is married or not.
- education: educational qualification of a person.
- default: was the credit card default?
- balance: bank balance of a person.
- housing: whether a person has home loan or not.
- loan: whether a person has personal loan or not.
- contact: communication contact type.
- day: last contact day of the week.
- month: last contact month of the year.
- duration:last contact duration, measured in second.
- campaign: number of contacts performed during this campaign.
- p_days: number of days that passed by after the client was contacted last in previous campaign.
- previous: number of contacts performed before this campaign and for this client.
- p_outcome: outcome of previous marketing campaign.
- y: has the client subscribed a term deposit?

Based on these 16 predictors(out of which age, balance, duration, campaign, pdays, p_outcome are numeric and rest are categorical), we have to predict whether a client has subscribed a term deposit or not.

3 Data Preprocessing

3.1 Missing Value Check

Since missing values are problematic in modeling purpose, first we have checked for missing values and in our dataset, there are no missing values.

3.2 Label Encoding

In our dataset, we have 16 predictors , out of which 11 predictors ('job', 'marital_status', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day', 'p_outcome') are categorical. To make these columns machine readable, we have opted for **Label Encoding**. It associates with converting the labels of a categorical variables into numeric form so as to make them machine-readable.

3.3 Data Splitting

If we use our whole dataset to train a model and use that dataset to check how our model is performing in fitting purpose, it may happen that for this particular dataset, our model is performing very good but it may fail to perform that good in a completely new dataset. This is known as problem of **overfitting**. To get rid of that, we split our dataset into 3 parts, namely, **Training Set, Cross Validation Set** and **Test Set**. We train our models using Train Set and evaluate the performances of the models using Cross Validation Set. After choosing the best model depending on the performance on the Cross Validation Set, we apply that model to a completely new and unnoticed dataset, namely, Test Set.

We have split the dataset in 7:2:1 ratio, i.e., Training Set consists of 70% of the total data, Cross Validation Set consists of 20% of the total data, Test Set consists of 10% of the total data.

3.4 Check for Data Imbalance

We have checked what proportion of client has subscribed for term deposit and what proportion of client has not subscribed for term deposit. We have seen almost 88% client has not subscribed for term deposit and 12% client has subscribed for term deposit. So, we can say that our dataset is highly imbalanced.

Now, Proceeding with imbalanced dataset can create erroneous model. Suppose, we have created a model that will simply predict that the client has not subscribed for a term deposit. Then, our model will have 88% accuracy. But, when it will be used to predict for a completely unknown dataset, it may give

very poor fit. To get rid of this **overfitting** problem, we have to deal with data imbalance.
Generally, the minority class is the class of interest in the classification problem and our goal is to achieve the best results in this class rather. If we do not treat imbalanced data beforehand, this will degrade the performance of the model. Most of the predictions will correspond to the majority class and the minority class will be treated as a noise in the data. This will result in high baias in the model.

3.5 Synthetic Minority Oversampling Technique(SMOTE)

To deal with data imbalance problem, we could have opted for **Oversampling** or **undersampling**. However, oversampling duplicates examples from minority class, which may result in duplication of same example more than once and in case of undersampling, there is a chance of losing information from our dataset.

As an alternative, we use SMOTE. It is one of the most commonly use methods to solve data imbalance problem. The objective of SMOTE is to balance class distribution by generating synthetic examples from minority class. The algorithm works as follows,-

- **Step 1**: We took an observation 'x' from minority class and we obtain **k nearest neighbours** of that point.

- **Step 2**: The synthetic examples is then created by choosing one of the k nearest neighbours 'y' at random and connecting 'x' and 'y' to form a line segment in the feature space. The synthetic examples are generated as a convex combination of the two chosen instances 'x' and 'y'. If x' is a new generated data point, it is given by,

$$\text{x'}= \alpha x + (1-\alpha)y, \text{ where, } 0 < \alpha < 1$$

We have applied SMOTE to our training data and successfully made our dataset fully balanced by generating sufficient number of synthetic examples.

4 Training Classification Algorithms

4.1 Logistic Regression

In our dataset, our response variable 'y' is binary. Logistic Discrimination model assumes that the class conditional densities of the two classes satisfy,

$$log\left(\frac{f(\mathbf{x}|Y=1)}{f(\mathbf{x}|Y=0)}\right) = \beta_0 + \beta^{'}\mathbf{x}$$

where,
$\mathbf{x}$ is the feature vector.
$f(\mathbf{x}|y=i)$ is class conditional densities, i= 0, 1.
$\beta_0, \beta^{16\times 1}$ are unknown deterministic constants.
It can be shown that,

$$P(y=1|\mathbf{x}) = \frac{e^{\beta_0^* + \beta^{'}\mathbf{x}}}{1 + e^{\beta_0^* + \beta^{'}\mathbf{x}}}$$

$$P(y=0|\mathbf{x}) = 1 - P(y=1|\mathbf{x})$$

where,

$$\beta_0^* = \beta_0 + log\left(\frac{P(y=1)}{P(y=0)}\right)$$

If $P(y=1|\mathbf{x}) \geq P(y=0|\mathbf{x})$, we assign it to class 1, otherwise we assign it to class 0.

After training the logistic regression model with our training set, we have applied the model on cross validation set and obtained the following **Confusion Matrix**,

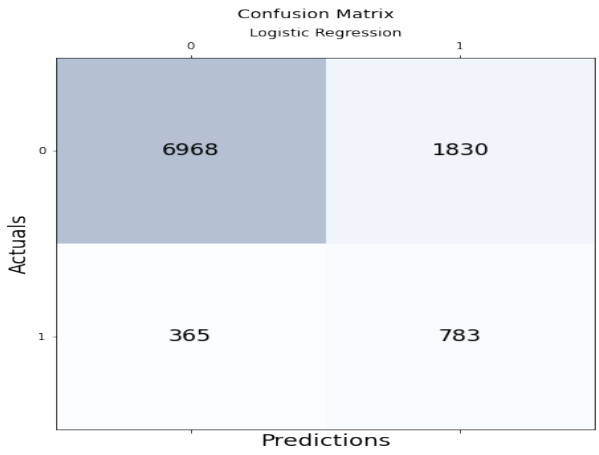


Figure 1: Confusion Matrix for Logistic Regression

4.2 Decision Tree

Decision Tree is an example of a multistage decision process. Rather than using the complete set of features jointly to make an output decision, different subsets of features are used at different levels of tree. Decision trees are drawn upside down. The points along the tree where the predictor space is split is called **internal nodes**. The topmost internal node is called **root of the tree**. The nodes, with which a class label is associated, is called **terminal nodes** or **leaf** of the tree.For a decision tree, we predict that each observation belongs to the most commonly occuring class of training observations in the region to which it belongs.

Now, the most important question in building a tree is that when should we stop splitting. We may grow the tree till all the terminal nodes are pure (all patterns belonging to the partitions have same class label). But it may cause the problem of **overfitting**.

To address this problem, we bring the concept of **pruning** of a tree. Pruning of a grown tree after applying splitting of nodes basically means cutting branches of the tree to get subtree of the original tree.

For pruning, we use **Cost Complexity Pruning** or **Weakest Link Pruning**.

(i)Cost Complexity Pruning:

Let, r(t)=1-max P(y=i|t), i=0,1
Define, R(t)=p(t)r(t)
If α denotes the cost of complexity per terminal node, then, define,

$$R_\alpha(T) = \sum_{t\in\tilde{T}} R(t) + \alpha|\tilde{T}|$$

where, $|\tilde{T}|$ is the cardinality of trtminal node set $\tilde{T}$.
In Cost Complexity Pruning, we start from a pure tree $T_0$and for a fixed α we find a subtree T_α such that $R_\alpha(T)$ is minimum.

(ii)Weakest Link Pruning:

Let, T_t denote the subtree at node 't'. Then we define **"Strength of Link"** at node 't' as,-

$$g(t) = \frac{R(T) - R(T_t)}{|\tilde{T}| - 1}$$

We prune the tree at the node t^* for which $g(t^*)$ is minimum.

After training the decision tree model with our training set, we have applied the model on cross validation set and obtained the following **Confusion Matrix**,

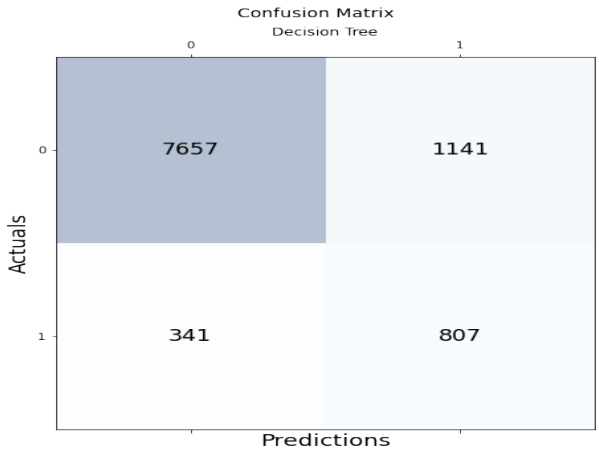


Figure 2: Confusion Matrix for Decision Tree

4.3 Random Forest

Decision trees suffer from high variance problem. To get rid of this high variance problem, we use random forest. In this method, we use the concept of **"Averaging independent observations reduce variance"**. In random forest, we take a bootstrapped sample of size N from the training data and grow a decision tree with these N samples. The only difference is that, in this case, in each split, we take a random sample of total number of predictors, preferable sample size is the square root of the number of predictors. We repeat this process, say, M number of times.

For classification setting, we note down the class predicted by each of the 'M' trees and for final prediction, we take the majority of these 'M' outputs.

The advantage of Random Forest lies in the fact that it **decorrelates** the trees. Suppose there is a strong predictor in our dataset. Then there will be a tendency to use this predictor in each top split. So the trees may look similar. In Random Forest, we are using a random sample of predictors and in some cases, the most important variables may even not be considered. We can think this as decorrelating the trees, thereby making predictions less variable.

After training the random forest model with our training set, we have applied the model on cross validation set and obtained the following **Confusion Matrix**,

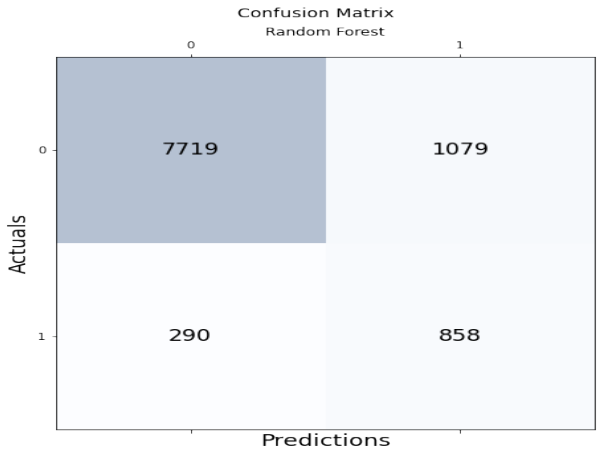


Figure 3: Confusion Matrix for Random Forest

4.4 AdaBoost

Adaptive Boosting algorithm, which is commonly known as AdaBoost algorithm is a type of ensemble boosting learning method, that is made on top of decision trees.

4.4.1 Differences with Random Forest

1. In Random Forest, in each time, we have to make a full sized tree. In a forest of trees made with AdaBoost, the trees are usually just a node and two leaves, called **stumps**. Stumps can only use one variable to make decision. So, they are **"Weak Learners"**.

2. In Random Forest each tree has equal vote on the final classification. In a forest of stumps made with AdaBoost, some stumps get **more say** in final classification than others.

3. In Random Forest, each decision tree is made independently of the others. In a forest of stumps made with AdaBoost, order is important. The errors that the first stump makes, influences how the second stump is made and so on.

4.4.2 Algorithm

1. We are given a dataset with certain number of predictors. First we create a column named **Sample Weight**, where,

$$\text{Sample Weight} = \frac{1}{\text{Total Number of Samples}}$$

2. We choose the best variable for making the first stump.(Based on Gini Index)

3. For this stump, we calculate the **Total Error** for the stump which is the sum of weights associated with the incorrectly classified sample.

4. Now, we need to determine how much say a stump will have in the final classification. We define **Amount of Say** by the following formula,

$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$

5. Now we have to modify the weights so that the next stump will take the errors that the current stump made into account. When we created the first stump, all of the sample weights were the same that meant we did not emphasize the importance of correctly classifying any particular sample. But if the first stump incorrectly classifies a number of samples then we have to emphasize the need for the next stump to correctly classify the samples by increasing their sample weight and decreasing all of the other sample weights.

   We increase the sample weights for incorrectly classified sample by the following formula,

$$\text{New Sample Weight} = \text{Sample Weight} \times e^{\text{Amount of Say}}$$

   We decrease the sample weights for correctly classified sample by the following formula,

$$\text{New Sample Weight} = \text{Sample Weight} \times e^{-\text{Amount of Say}}$$

   After that we normalize the New Sample Weights so that they sum up to 1.

6. After that, we have to create a new dataset from the previous one. In the new dataset, the frequency of incorrectly classified observations will be more than the correctly classified ones. The new dataset will be created using the normalized weights. To make a new dataset based on normalized weight, the algorithm will divide it into buckets, based on cumulative normalized weights.

7. We will stop when the new dataset will be same as the previous one.

How to make classification?

Suppose with our dataset, we obtained 'n' stumps. Then the test dataset will pass through all the stumps that we have constructed and for final prediction, we will take majority votes. After training the AdaBoost model with our training set, we have applied the model on cross validation set and obtained the following **Confusion Matrix**,

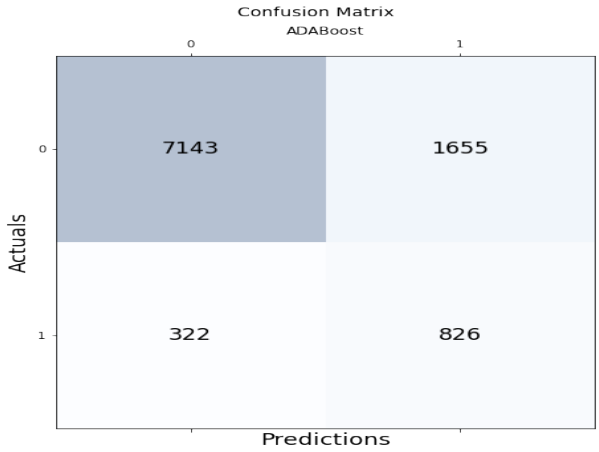


Figure 4: Confusion Matrix for AdaBoost

4.5 Gradient Boosting

4.5.1 Definition

Gradient boosting is an iterative tree based algorithm which uses gradient functions to build a prediction model as an ensemble of weak prediction models. The term "gradient" suggests that there are one or more derivative terms of the same function. This algorithm basically works by minimizing a loss function iteratively by choosing a function that points towards negative gradient.

4.5.2 A few advantages and disadvantages

- Advantages:
  - This algorithm can be optimized on several kinds of loss functions and it can also provide various hyper parameter tuning options which make the function fit very flexible.
  - There is no need of much data pre-processing in this algorithm. This works well for both numerical and categorical data.
  - In this algorithm there is no need of missing value imputation.

  - This algorithm often provides greater predictive accuracy than many other algorithms.

- Disadvantages:
  - This algorithm can be computationally expensive. Sometimes it requires to fit many (> 1000) trees and so it can be quite time consuming and take a quite lot of memory.
  - This algorithm isn't quite interpretable in nature.
  - If not proper parameters are set, this algorithm may be prone to continue to minimize all error and as a result cause over-fitting.
  - As a result of greater flexibility, more number of parameters are to be properly set, which influences the algorithm. So a larger number of grid search is required.

Before beginning let us assume that, our given data is of the form $\{X_i, y_i\}$, where $_i$ is the i^{th} feature vector and y_i is the corresponding i^{th} response. Here, as we facing a classification problem, we will use Gradient Boosting Classifier. Also, let the response y_i is of binary type i.e. of 1 and 0 type.

4.5.3 Algorithm

1. First, we will calculate the log-odds of the responses. That is, we will calculate the
$$log(Odds) = log\left(\frac{\#(y_i = 1)}{\#(y_i = 0)}\right)$$
.

2. Then we will calculate the probability using the logistic function using the log-odds and assign that as a primary guess probability for all the observation. So we will calculate
$$\hat{P}(y_i = 1) = \frac{\exp(log(Odds))}{1 + \exp(log(Odds))} = \hat{p_i}$$
(let).

3. Then we will calculate the residual as the difference between the actual responses and the probability obtained from step (2). i.e.
$$residual_i = y_i - \hat{p_i}$$

4. A regression tree is built on the regressors to predict the residuals obtained in step (3).

5. Now, for every terminal node t of the regression tree an **output** is calculated for all the patterns in that terminal node using the formula:
$$\frac{\sum_{i:X_i \in t} residual_i}{\sum_{i:X_i \in t} \hat{p_i}(1 - \hat{p_i})}$$

6. The previous $log(Odds)$ values are updated by multiplying the new output obtained from the regression tree in step (5) with a learning rate α and adding them with the previous $log(Odds)$.

7. The probabilities of $\hat{p_i}$ are updated by calculating the logistic function using the new $log(Odds)$ as it was done in step (2).

8. Again the residuals are calculated, and a regression tree is build, the $log(Odds)$ are updated and again a new set of probabilities are calculated. Thus step (2)-(7) are done iteratively until a certain amount of trees are built or the residuals become negligible.

After training the Gradient Boosting model with our training set, we have applied the model on cross validation set and obtained the following **Confusion Matrix**,

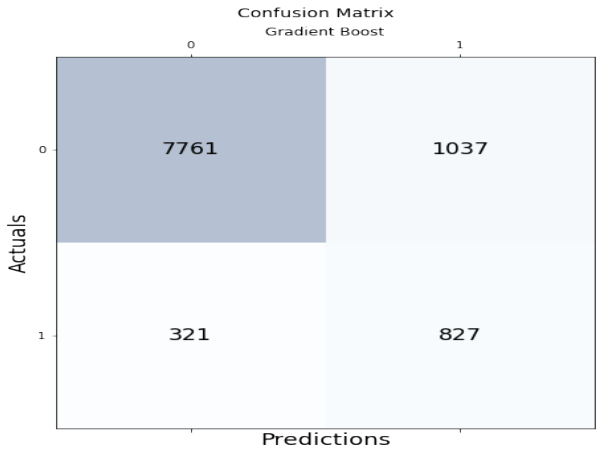


Figure 5: Confusion Matrix for Gradient Boosting

5 Final Model Selection based on Evaluation Metrics

5.1 F-Score

To define F-Score, we have to define **Precision** and **Recall** first.

$$\text{Precision (P)} = \frac{\text{True Positive}}{\text{Total Number of Predicted Positive}}$$

$$\text{Recall (R)} = \frac{\text{True Positive}}{\text{Total Number of Actual Positive}}$$

Then, F-Score is given by,-

$$\text{F-Score} = \frac{2\text{PR}}{\text{P+R}}$$

The higher the F-Score of a model, the greater the accuracy is.

5.2 ROC-AUC Curve

It is a curve of **True Positive Rate (TPR) vs False Positive Rate (FPR)**, where,

$$\text{TPR} = \frac{\text{True Positive}}{\text{Total Number of Actual Positive}}$$

$$\text{FPR} = \frac{\text{False Positive}}{\text{Total Number of Actual Negative}}$$

The area under this curve is known as AUC (Area Under the Curve). The higher the AUC of an ROC curve, the better the model is.
Now, we will represent the ROC Curve for respective models.

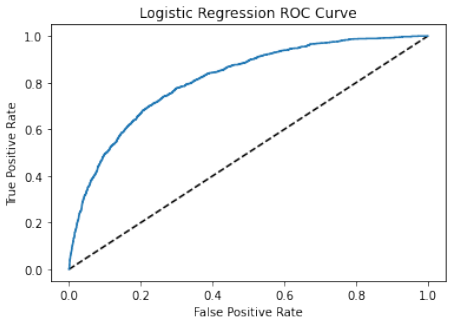


Figure 6: ROC Curve for Logistic Regression

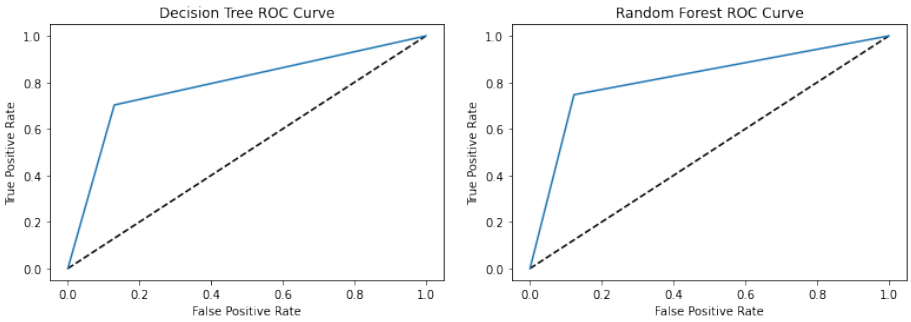


Figure 7: ROC Curve for Decision Tree and Random Forest

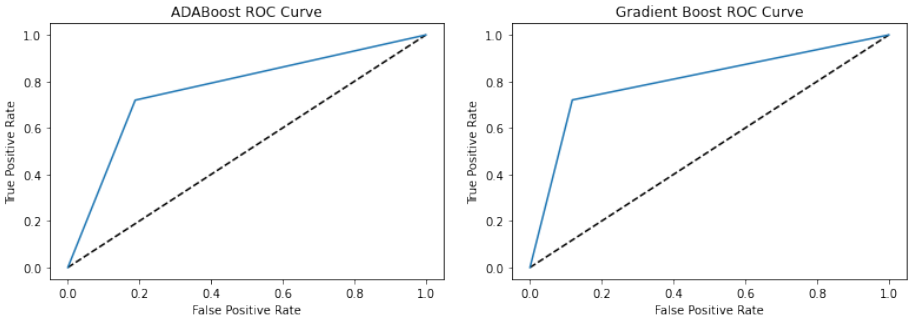


Figure 8: ROC Curve for AdaBoost and Gradient Boosting

Now, we will tabulate the values of F-Score and AUC for the respective models and we will choose that model to be the best one for which we get the maximum F-Score and AUC.

	Accuracy	F-Score	AUC
Logistic Regression	0.78	0.42	0.79
Decision Tree	0.85	0.52	0.79
Random Forest	0.86	0.56	0.81
AdaBoost	0.80	0.46	0.77
Gradient Boosting	0.86	0.55	0.80

Table 1: F-Score and AUC for Various Models

From the above table, we can see that Random Forest has the highest F-Score and AUC among all models. So, we have chosen this model to predict whether a client will subscribe term deposit or not.

6 Applying the Model on Test Set

After choosing Random Forest to be the best model, we have applied the model on test set to see the performance of this model on a completely new dataset.

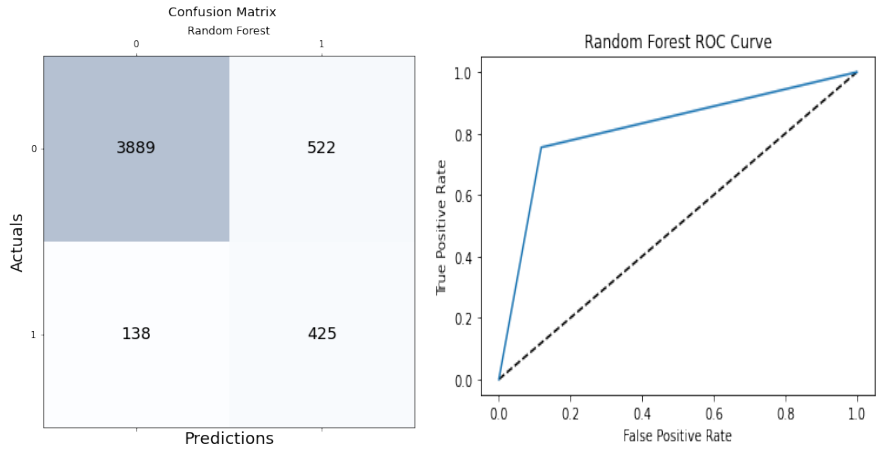


Figure 9: Confusion Matrix and ROC Curve for Final Model

Fianlly we tabulate the values of accuracy, F-Score and AUC for this model.

	Accuracy	F-Score	AUC
Random Forest	0.87	0.56	0.82

Table 2: F-Score and AUC for Final Model

7 Conclusion

From all our analysis, we can conclude that, for this particular dataset, where the objective is to predict whether a client has subscribed a term deposit or not, **Random Forest outperforms all other classification algorithms**.

8 References

1. James, G., Witten, D., Hastie, T., Tibshirani, R., An Introduction to Statistical Learning- with Applications in R, Springer.
2. https://towardsdatascience.com/
3. https://www.analyticsvidhya.com/
4. https://statquest.org/

9 Appendix

The python code for this project can be found here.

10 Acknowledgement

We would like to express our deeply felt gratitude towards **Prof. Dr. Amit Mitra**, whose continuous support and exemplary guidance has helped us in the fulfillment of this project.