# Abstract

In this project, we develop a pattern recognition system that operates on Bank Marketing Data Set. The dataset consists of information on people targeted by a bank marketing campaign. With the system, we hope to accomplish the goal to predict if a client will subscribe to a long-term deposit at a certain bank after receiving a marketing call. We implemented preprocessing for the data sets and managed to reduce the irrelevant features. The Dataset was first divided into training set and testing set keeping the same percentage class distribution in comparison to the original dataset. After missing data filled and irrelevant data reduced, we also managed to keep the data balanced. Then, five various classifiers were applied in this project and the results are quite promising. It was found that ***Random Forest*** gives the best f1 score. Among all the features, we use ***random forest*** to get the importance of them and ‘***age***’, ‘***euribor3m***’, ‘***campaign***’, ‘***nr.employed***’ are the most important features which determine whether the customer would subscribe to a long-term deposit.

# Pre-processing Of Dataset

Firstly, we implement panda to extract the data from bank-additional.csv and divide the data into two parts by whether the result is ‘Yes’ or ‘No’.

1. ***Training and Testing Set Formation***

The data set is divided into two parts. One is the 19 features and the other is the label ‘y’ which indicates whether the person would subscribe to a long-term deposit after a marketing call. Among all the data sets, we select 80 percent as the training set and the other 20 percent as the testing set.

1. ***Preprocessing the Data Set***

The key approach of the whole project is preprocessing, as in Datasets provided consists of different specific name and categories instead of regular numbers which normal training classifier functions accept. Moreover, blank features along with unknown ones exist in the data set which need preprocessing.

* 1. ***Recasting the representation of Feature***

We use pandas function ‘***read\_csv***’ to import the data from csv file and the features have different types. Recasting the representation of feature is to recast the ***String*** data into ***Numeric*** data set. For many features like ‘housing’ or ‘loan’ which have only ‘yes’ or ‘no’, we use 0 and 1 to represent it.

* 1. ***Filling in the missing value***

Since the background is very detailed, many samples lack certain features. In the Dataset provided, there exists missing points which shows ‘***unknown’*** in the table. For most of the cases, ‘unknown’ features are replaced by the most common cases in that scenario. We first do statistic analyze to figure out which is the most frequent value and use that value to replace ‘unknown’. For example, ‘unknown’ in ‘marital’ is replaced by ‘married’ using python function ‘***replace***()’, as in such replacement may not change the performance of classifiers significantly and saves us lot of work determining how to deal with missing values.

* 1. ***Categorical Feature Expansion***

As described above, recasting the String data into Numeric ones are crucial. However, not all of the features are suitable with such replacement. For example, feature ‘day\_of\_week’ shows the last contact day of the week, which involves 7 situations. We can’t simply assign number 1 to 7 to this situation since Saturday is not necessary larger than Monday. Therefore, we import ***panda*s** function ‘***get\_dummies’*** which creates a several columns to represent each value. After that, we combine the original data sets and new columns together.

***Note***: Categorical feature expansion was done by merging the testing and training dataset together to prevent an unequal number of features after expansion and also prevent different features from being removed.

* 1. ***Feature dimensionality reduction***

For the reason that we have 43 features, certain feature dimensionality reduction is necessary.

* 1. ***Normalization***

Normalization is probably the most common preprocessing methods. Since most values are ‘0’ and ‘1’, and some values are range far from 0 and 1, we need to normalize our data set. We import function ‘***MinMaxScaler’*** from ‘***sklearn.preprocessing***’ to perform that.

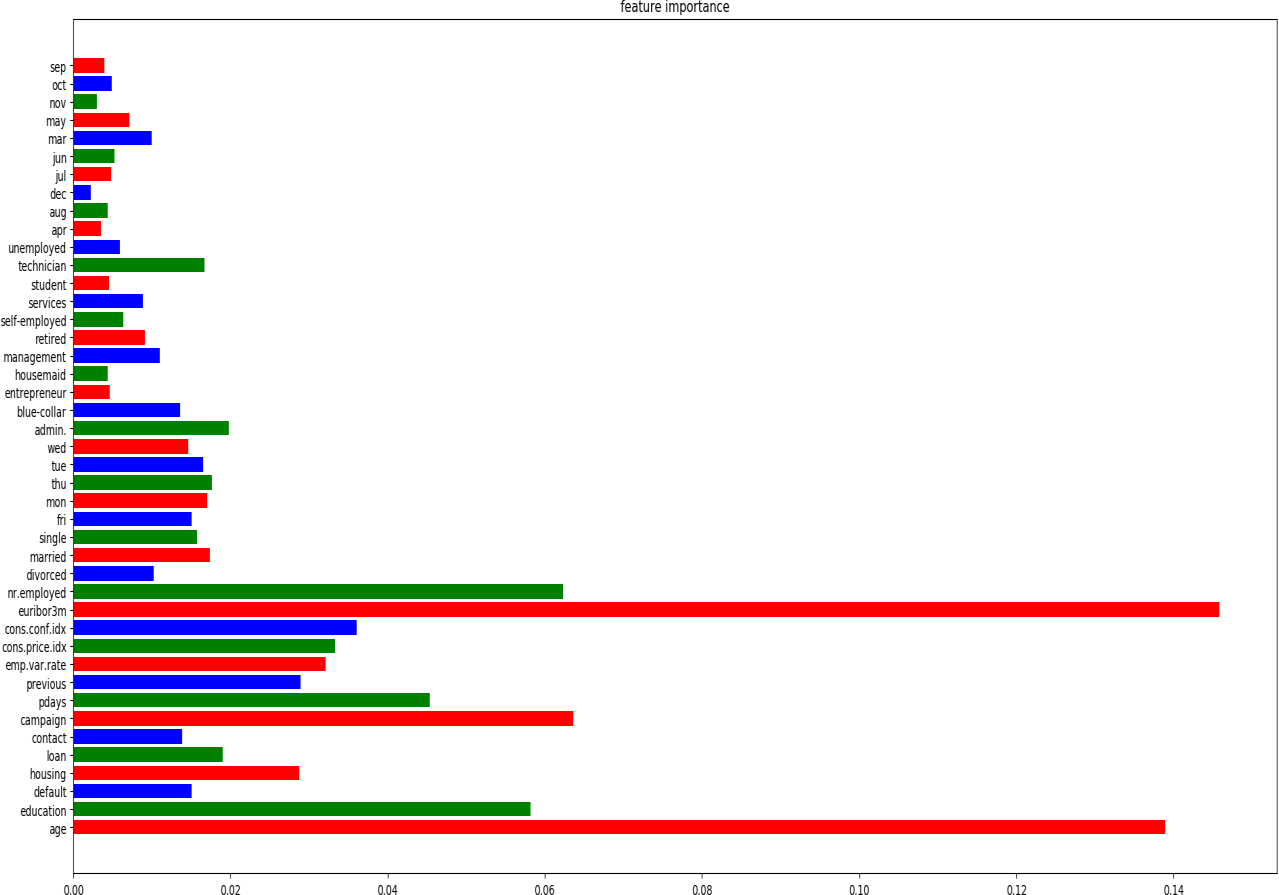
* 1. ***Balancing***

After all the pre-processing we have done, most of the classifiers still failed to meet the expectation. As we are predicting whether the target would subscribe to a long-term deposit at a certain bank. The results we got are often ‘no’, when we found out that the Dataset is actually quite unbalanced. About 90 percent the clients in the training sets choose ‘no’, which affects our training performance. We select all the data sets that choose ‘yes’ and duplicate them three times to reach a balance between positive data points and negative data points.

# Feature selection

Given the condition that we have 43 features in total, feature selection is necessary for our project. Here we implement feature dimensionally reduction based on the ***importance*** of each feature, since many features might be irrelevant features for the final decision results.

1. ***Get the importance of all the features.***

We import function ‘***RandomforestClassifier’*** from ‘***sklearn.ensemble***’. The function takes training features and labels as input and its ‘***feature\_importances\_***’ saves the importance of all features. Specifically, the code uses mean Gini gain produced by features over all trees to measure the importance.

1. ***Delete the unimportant features.***

We did a for loop to get all the indexes of features which have an importance value lower than 0.004. Such features are proved to be irrelevant to the prediction we are seeking for. Therefore, we simply delete them from the training sets. This will reduce noise interference and reduce the amount of calculations.

# Classifies and Results

In this project, we use five training methods just to be able to get more detailed results and see which classifier performs better of all.

The classifiers we used are ***random forest***, ***SVM***, ***KNN model*** and ***Naïve Bayes***.

***Cross validation***

* Set up arrays for different parameters.
* Run loops for parameters and record the ***auc*** and ***F1 score***.
* Choose the best performance combo and use that parameters to train the data set.

1. ***Random Forest***

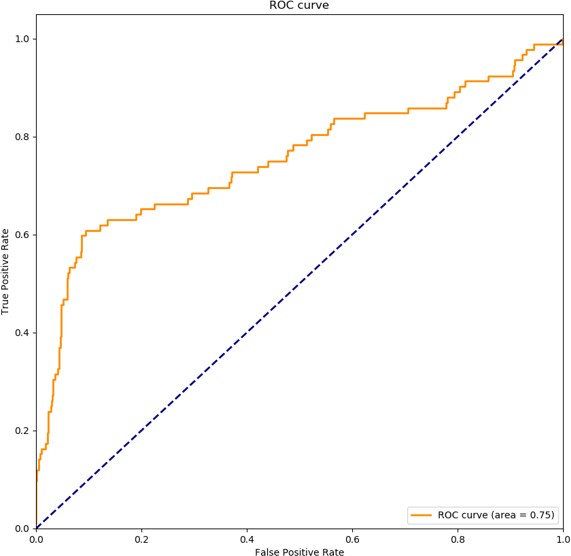
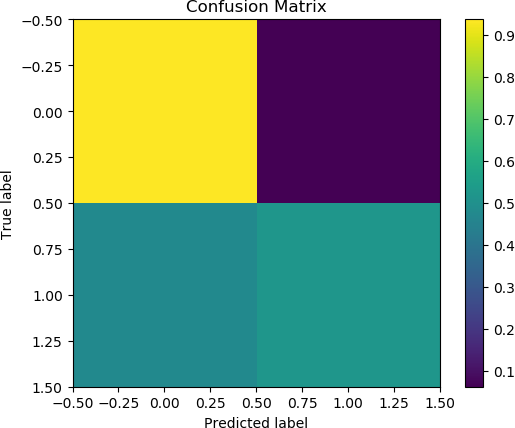
Random forest is an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification,](https://en.wikipedia.org/wiki/Statistical_classification) [regression,](https://en.wikipedia.org/wiki/Regression_analysis) that operate by constructing many [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and usually outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the all classes.

We import function ‘***RandomForestRegressor’*** from ‘sklearn.ensemble’ to implement this classifier. Then use ‘fit()’ and ‘predict()’ to perform training and predicting.

After doing cross validation, we choose following parameters and train the data:

**n\_estimaters** = 200, **max\_depth** = 5, **min\_samples\_split** = 2

*Result(Also the best result):*



Confusion Matrix:

[0.93715847 0.06284153]

[0.47826087 0.52173913]

Random Forest AUC: 0.7545883820384891

Random Forest Accuracy: 0.8907766990291263 Random Forest F Score(Binary): 0.5161290322580645

1. ***SVM***

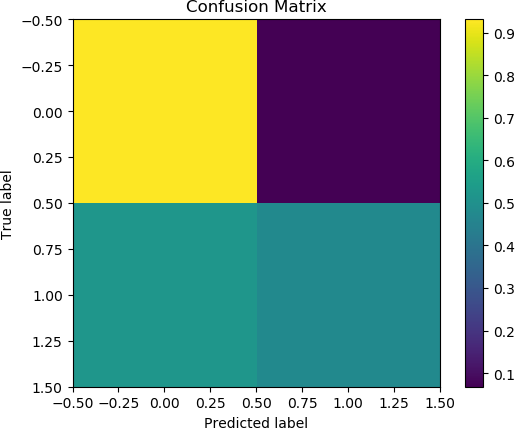
A Support Vector Machine(SVM) is a discriminative classifier formally defined by a separating hyperplane.

We import function ‘***SVR’*** from ‘svm’ to implement this classifier. Then use ‘fit()’ and ‘predict()’ to perform training and predicting.

After cross validation, we choose following parameters and train the data:

**C** = 6, **gamma** = 0.002

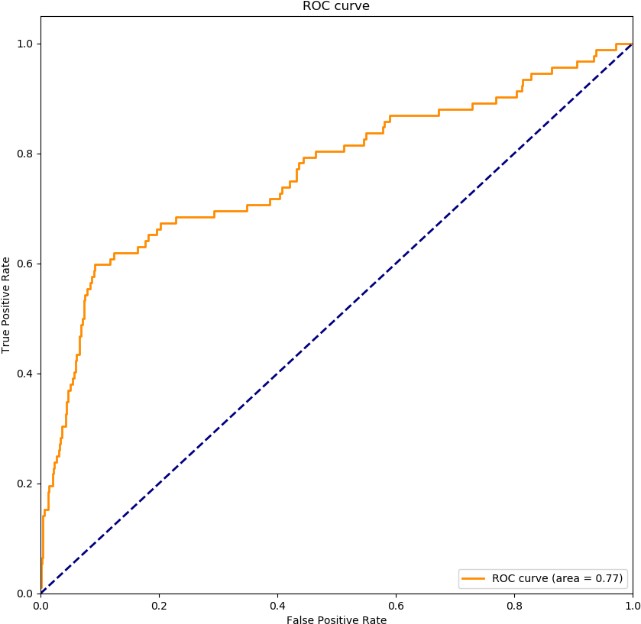
*Result:*



Confusion Matrix:

[0.93169399 0.06830601]

[0.52173913 0.47826087]



SVM AUC: 0.7677150154430981

SVM Accuracy: 0.8810679611650486

SVM F Score(Binary): 0.47311827956989244

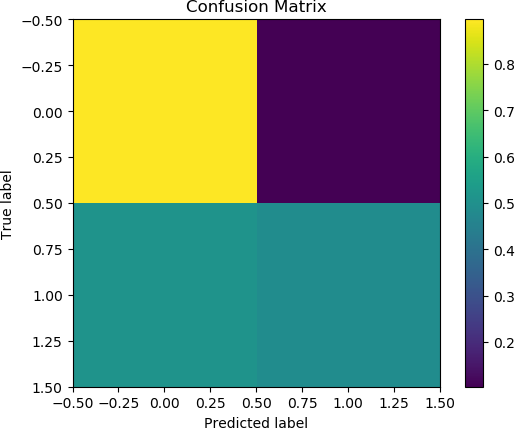
1. ***KNN models***

K nearest neighbors is a simple algorithm based on a similarity measure between data points. We import function ‘***KNeighborsClassifier’*** from ‘sklearn.neighbors’ to implement this classifier. Then use ‘fit()’ and ‘predict()’ to perform training and predicting.

After cross validation, we choose following parameters and train the data:

**n\_neighbors** = 20

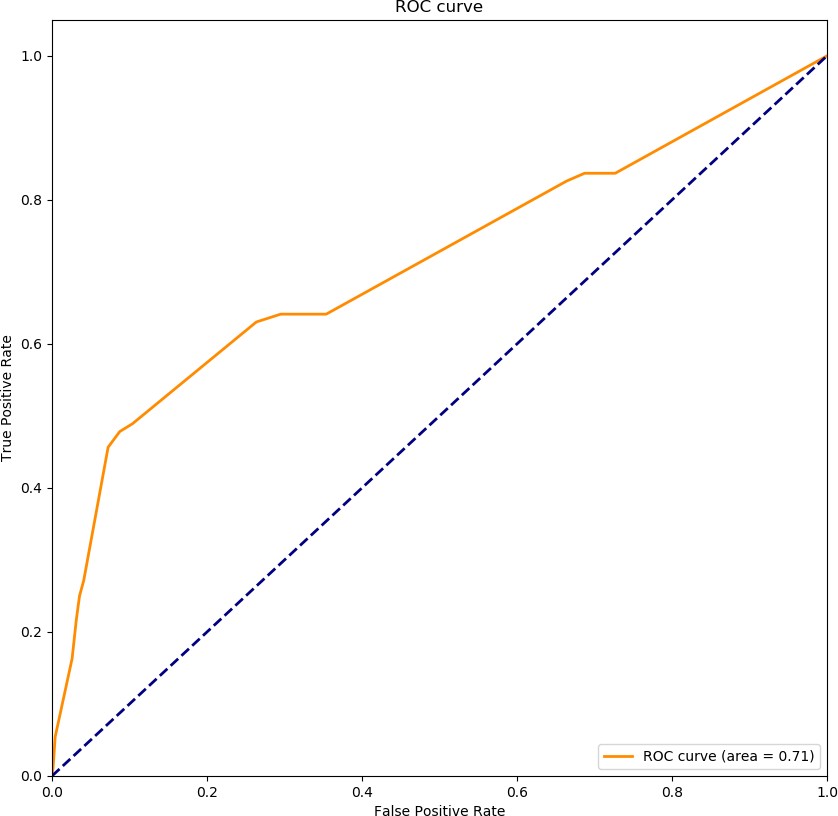
*Result:*



Confusion Matrix:

[0.89617486 0.10382514]

[0.51086957 0.48913043]



KNN AUC: 0.7103602399619863

KNN Accuracy: 0.8507281553398058

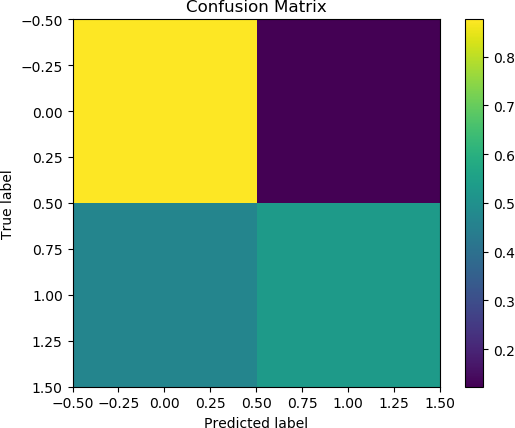
KNN F Score(Binary): 0.42253521126760557

1. ***Naïve Bayes Model***

Naive Bayes classifiers is a simple [probabilistic classifiers](https://en.wikipedia.org/wiki/Probabilistic_classifier) "based on applying [Bayes' theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with strong (naive) [independence](https://en.wikipedia.org/wiki/Statistical_independence) assumptions on the distribution of features.

We import function ‘***GaussianNB’*** from ‘sklearn.naive\_bayes’ to implement this classifier. Then use ‘fit()’ and ‘predict()’ to perform training and predicting.

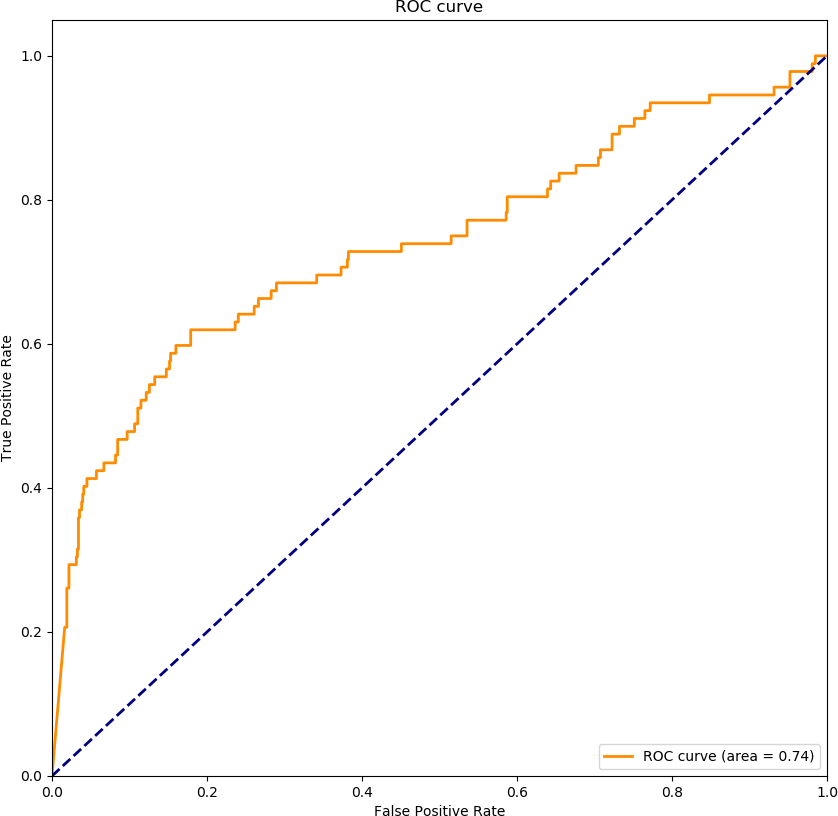
*Result:*



Confusion Matrix:

[0.87704918 0.12295082]

[0.4673913 0.5326087 ]



Naive Bayes AUC: 0.7393828700403896

Naive Bayes Accuracy: 0.8385922330097088

Naive Bayes F Score(Binary): 0.42424242424242425

# Performance Evaluation Techniques

The performance evaluation was done on the testing data sets which would go through the same preprocessing steps except for the balancing procedure. The predicted data sets were sent to a function called ***metrics.auc*** from sklearn. Also we generated a function ‘***getaccuracy’*** to get the overall accuracy. Moreover, we use ‘***f1score’*** to get the F1 score of the classifier.

# Interpretation/ conclusion

* + Being able to finish the whole projects teaches us that pre-processing is crucial to the final results no matter how advanced your algorithms are. The Dataset provided is filled with plenty unknown values, not to mention possible mistaken features.
  + Features selected in each project vary because of different models and conditions. We need to run proper algorithms to find out which features are truly useful.
  + Balancing the datasets is quite important in the way that even if you have accomplished perfect pre-processing and feature selections, the results might still fail you because of unbalanced datasets. Be careful, duplicating the positive data points should be done before splitting training sets and test sets.
  + When applying certain classifiers, cross-validation is often important as in parameters for the classifier input are highly dependent on the given features. We need certain trials to make sure proper parameters are used to the classifiers training.
  + Random Forest and SVM perform much better than Naïve Bayes and KNN. The unbalanced impact for Random Forest and SVM are relative small.
  + The best result is Random Forest, AUC: 0.7545883820384891, Accuracy: 0.8907766990291263 F Score(Binary): 0.5161290322580645

# Team work

We contributed equally in this project. For each procedure, we started our own code first. After discussion and experiments, we pick the effective parts and contribute to the final project based on the results.

# Reference

* + ***Large-Scale Support Vector Machines: Algorithms and Theory*** - Aditya Krishna Menon<https://cseweb.ucsd.edu/~akmenon/ResearchExam.pdf>
  + ***Nearest Neighbor Ensemble*** - Carlotta Domeniconi, Bojun Ya, Information & Software Engg Dept, George Mason University<https://cs.gmu.edu/~carlotta/publications/NNensemble.pdf>
  + ***Distribution of variable values and readmissions*** (population size is 69,984)<http://www.hindawi.com/journals/bmri/2014/781670/tab3/>
  + <https://www.wikipedia.org/>