EE559 Spring 2018 Project

Dataset Used: Bank Marketing Dataset

Name: Thiyagarajan Ramanathan
USC ID: 4973341255
Email: tramanat@usc.edu

Abstract:

Designing the best classifier for any given dataset is an art. The objective of any 'data scientist' will be to design the classifier which will give the best classification accuracy for the unknown (test) data. Designing the best classifier is a very important step in processing any business, or medical applications. In critical applications which involve processing and classification of data, we cannot afford to use a classifier which is not suitable for the data. Hence, the objective of the project is to analyze the data and design the best classifier model. The data given has a lot of missing datapoints and it is also imbalanced. Hence the important stage in this project will be to preprocess the data and then find the best classifier for the given data. This project has been implemented in Python using the Sci-kit Learn Package.

Aim:

To predict whether a customer will open a deposit at the bank after having a phone conversation with the bank official. The dataset is created by considering the answers provided by the customers during the phone conversation. The objective of the project is to design the best classifier which will predict whether a customer will invest in a long deposit or not. The problem can be classified as a supervised learning problem and once the model is trained, it can be used for predicting whether the customer will invest in a long-term deposit or not.

Procedure:

There are many steps that are to be followed to design a classifier. The following steps are to be followed while building the classifier:

1) Acquiring Data
2) Analyzing and Understanding the data
3) Data Pre-processing
4) Model Selection
5) Building Classifier
6) Interpreting the results

Acquiring Data:

The very first step is to acquire the data. The bank marketing dataset is obtained from the following website: 'https://archive.ics.uci.edu/ml/datasets/Bank+Marketing#'. This dataset was collected through bank marketing campaigns in Portugal. The results of the obtained data were used to predict the output labels. The 'bank-additional.csv' was used as the reduced data was used in the bank-dataset.

Analyzing and Understanding the Data:

The next step after acquiring the data is to understand it. There are about 19 features in the original dataset. Of these, many features have categorical values as well as numerical values. We have to analyze what each of the features mean to decide whether each of them will help us in classification or not. The name of each feature and their meanings are given below:

1) Age: Age of the client
2) Job: Current job of the client

Values: ‘admin’, ‘blue-collared’, ‘entrepreneur’, ‘housemaid’, ‘management’, ‘retired’, ‘self-employed’, ‘services’, ‘student’, ‘technician’, ‘unemployed’, ‘unknown’

3) Marital Status: Marital status of the client
   Values: ‘divorced’, ‘married’, ‘single’, ‘unknown’
4) Education: Education level of the client
   Values:’basic.4y’,‘basic.6y’,’basic.9y’,’high.school’,’illiterate’,’professional.course’,’university.degree’,’unknown’
5) Default: Tells whether the client has defaulted a loan earlier or not
   Values: ‘no’, ‘yes’, ‘unknown’
6) Housing: Tells whether the client has taken a housing loan or not
   Values: ‘no’, ‘yes’, ‘unknown’
7) Loan: Tells whether the client has taken a personal loan or not
   Values: ‘no’, ‘yes’, ‘unknown’
8) Contact: Tells how the client was contacted
   Values: ‘cellular’, ‘telephone’
9) Month: Tells during which month, the client was contact
   Values: All 12 months of the year
10) Day of the week: Tells on which day of the week the client was contacted
    Values: All 7 days of the week
11) Campaign: Tells how many times the client has been previously contacted during this campaign
    It has numerical values
12) PDays: Tells how many days have passed since last contacted
    Values: numerical or ‘999’ (means the client was not previously contacted)
13) Previous: Tells how many times the client has been previously contacted before the current campaign
14) Poutcome: Tells the outcome of the previous campaign
    Values: ‘failure’, ‘success’, ‘non-existent’
15) Emp.var.rate: Employment variation rate
    Numeric value, quarterly indicator
16) Cons.price.idx: Consumer Price Indicator
    Numeric value, monthly indicator
17) Cons.conf.idx: Consumer Confidence Indicator
    Numeric value, monthly indicator
18) Euribor3m: Euribor 3-month rate
    Numeric value, daily indicator
19) nr.employed: Number of employees
    Numerical value, quarterly indicator

We can perform classification using these features. But there is a lot of pre-processing to be done on the data before we can use this data for classification. Hence the next step in designing the model is, data pre-processing.

Data Pre-processing:

We have seen the different features in the given dataset. The next step is to process them. By processing them, we will be converting them to a model readable format. For example, some features have categorical outputs and hence they need to be converted to numerical values

otherwise, we cannot train the classifier using that particular feature. Also, we have seen many columns which have the value 'unknown' in cases of categorical values and 'nan' in terms of numerical values. The classifier cannot use these values directly, and hence the data needs to be pre-processed. There are many stages in pre-processing. Each of it is explained below.

i) Imputing:

This is the method followed to take care of the unknown data in the given dataset. The simplest method to deal with this problem, is to remove the datapoints which have unknown values in them. This can be done when the dataset is really large, and removing these small number of datapoints may not have any role in the model accuracy. But, when the data is small we need to find other methods to fill the missing data. There are many methods for filling missing data such as mode imputing[1], mean imputing, or using another classifier to impute the missing values. For performing these different methods of imputing, the following steps have to be followed:

Step 1: All the columns having an unknown value has to be removed

Step 2: The datapoints which don't have any unknown value will be used as training data for imputing and the datapoints which have any unknown value, will be considered as testing data for imputing.

Step 3: We have to train the classifiers unknown column by unknown column and fill the unknown data by using the trained model

Once these 3 steps are performed, we will not have any unknown datapoints in our data. This is how we have to deal with unknown data.

ii) Dealing with Imbalanced data:

The next step is to deal with imbalanced data. The meaning imbalanced data is that, datapoints from one class will be really high when compared to the number of datapoints in the other class. In the given bank marketing dataset, the number of points belonging to class 1 are around 3700 and the other class are around 400. If the classifier is just trained with these points, the classifier will give more importance to the points belonging to the larger class and will classify the points correctly from that class, but will tend to misclassify the datapoints from the other class. To avoid this, we can pass this parameter 'class_weight=balanced' to the classifier while building it. The other method used is, to perform oversampling or under-sampling. Oversampling means to increase the number of datapoints in the class which has lesser number of datapoints. There are many methods for over-sampling. One such method is SMOTE. Under-sampling tries to reduce number of data points in the class which has large number of datapoints. By performing either one of the two above mentioned methods, we would have made the dataset balanced.

iii) Label Encoding:

The classifier can only understand numerical values. In the dataset we have, there are many categorical values and hence they need to be converted to numerical values. This can be done with inbuilt commands such as 'one hot encoder' and 'label binarizer'. When a column is passed to one-hot-encoder, the categorical values become columns and if a datapoint has that particular categorical value corresponding to the separated columns, it will have value '1' for that particular column, and will have value '0' for all the other separated columns. Label binarizer is used for columns which have only 2 outputs and they can be represented by either 1 or 0. There are inbuilt commands for

label binarizing in sklearn namely 'LabelBinarize' and one hot encoding in pandas using the command 'pandas.get_dummies'.

iv) Scaling:

The final step to be followed in data pre-processing is scaling the features. For example, one feature may be present in the range of 5000-10,000 and another feature maybe present in the range of 0-100. When the classifier is trained using these parameters, they tend to give low accuracies due to the wide variance in each of the features. To deal with this problem, each of the features have to be scaled. There are two methods of scaling followed. One is Standard scaling and another is minmax scaling. In standard scaling, each of the column is scaled in such a way that their variance is 1 and the mean is 0. The second method of scaling is to use minmax scaler. The minmax scaler scales all the values to the range of 0 to 1 based on the minimum and maximum values of each feature. Once this is done, pre-processing is done. The data can now be used for building the classifier. The inbuilt command in Sklearn to perform scaling is 'Standard Scaler' and 'MinMax Scaler'.

Model Selection:

After the data is processed, we need to check the complexity of the model that will be built using the processed data. If the data dimension is very high, then the complexity of the model will be high. For reducing the complexity of the model. There is a procedure to be followed before the model is built. The following steps are to be followed:

i) Feature Selection
ii) Dimension Reduction
iii) Cross Validation

Feature Selection:

Feature selection is the first step in model selection. During this step, we have to analyze the importance of each feature. There are many methods for doing this. One such method is to use the 'kbest' method from sklearn. We can specify what method to be considered while using this command. The default function used in 'kbest' is 'f_classif'. What this method does is, it calculates the difference between all the features in the data and chooses the features that satisfy a particular threshold. For instance, the difference between a specific feature and the other features will be calculated. The features that have the highest difference will be chosen. We can also decide the number of features we want from the data. If we set k=10 in 'kbest', that will give us 10 of the best features from the dataset. The inbuilt command from sklearn is, 'SelectKBest' from sklearn. model_selection.

There are many alternatives for kbest feature selection method. Another method is variance threshold. This method calculates the variance for all the features in the dataset. Then it decides what features to retain based on the variance threshold we set. For example, if the variance threshold is set as 0.5, all the features having the variance less than threshold 0.5 will be removed. This is the idea of the variance thresholding based feature selection.

The third idea is to use a feature wrapper. In this method, we will be selecting features and combinations of features to decide the best number of features and best combinations of features. This feature chooses the best feature first, i.e., the feature which gives the best classification

accuracy. After choosing the first best feature, it chooses a combination of that specific feature with all the other features. This process is repeated until the best features are chosen.

Dimension Reduction:

Dimension Reduction and Feature selection may sound similar. But they are very different concepts. Feature selection chooses the best features but dimension reduction reduces the dimension using PCA. What PCA does is, it projects the higher dimensional data into a lower dimensional space. In feature selection, the bad features are killed, but in dimension reduction the bad features are projected onto lower dimensions which have high discriminant power. By doing this, we have reduced the dimension as well as preserved the data. Another difference between PCA and feature selection is that, in feature selection we will know what feature is preserved and not. Whereas in PCA, since the data is projected onto a lower dimension, we cannot retain the information of the features present after doing PCA. The inbuilt command is present in sklearn.decomposition.

Cross Validation:

We have discussed about selecting the best features, dimension reduction etc. But as soon as we see the data, we cannot identify which number of features will give us the best classification accuracy. Also, while building the classifiers, the model will have many parameters. We will not have any idea about which parameters are to be used for the classifier. Hence, for deciding the best parameters for a model as well as choosing the best features, we can follow the procedure for cross validation. In cross validation, the training data is split into validation set and train data. The models are trained for different parameter using the training set and tested on the validation set. Once that is done, we can choose the parameters based on the highest accuracy values. This process can be repeated for other parameters such as choosing the number of PCA components, number of KBest features, parameters for the models etc. The summary for cross validation procedure is given below:

Step 1: Split the training data into validation data and training validation data.

Step 2: Try different parameters for model, different features etc., and test them in a looping fashion on validation test set. Save the accuracies for each of the run.

Step 3: Choose the parameters for which the cross-validation accuracy is the highest.

There are inbuilt commands to for splitting the test and train data namely, ‘test_train_split’ in sklearn.model_selection.

Performance Metrics:

We have to decide on the accuracy metrics for model evaluations. Different datasets have given the correct performance metrics only if the right performance metrics are used. For example, if the dataset is imbalanced, the accuracy score will not give proper performance metrics because, the model will be inclined towards classifying the datapoints of the larger class correctly, hence the datapoints of class1 will be classified correctly but the datapoints of class2 will not be classified correctly. For these type of problems, we need to use another performance metric called as f1 score and AUC score. For explaining f1 score and AUC score, we need to define the confusion matrix.

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

From the confusion matrix, we can understand the parameters TP (True Positive), FN (False Negative), FP (False Positive) and TN (True Negative). TP stands datapoints classified as class 1 when their true label is also class 1. TN gives the number of datapoints which are actually class2 and they are also predicted as class2. In FP, the true label is class2, but the classifier predicts those points as class1. In FN, the true values are class1, but the datapoints are predicted as class2. By using these values, Precision and Recall values can be calculated.

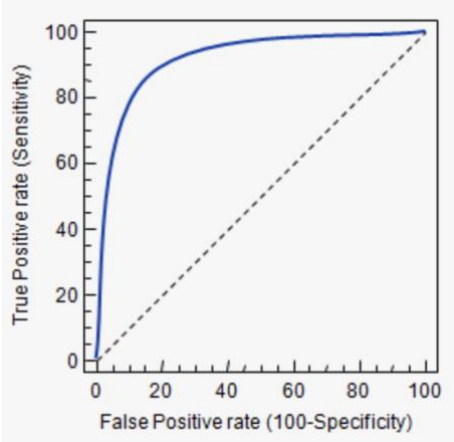
$$\text{Precision}: P = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}}$$

$$\text{Recall}: R = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}}$$

$$F = 2 \times \frac{P \times R}{P + R}$$

From the value of f1 score, we can get the performance of the model. Higher values of f1 mean the model performance is better.

Another performance metric for the imbalanced type of dataset is, ROC AUC. This also uses the True positive and True Negative values calculated from the data classified using the model. For ROC AUC, a curve is plotted.



The curve is plotted with the help of the threshold values used for classifying the datapoints as greater than a threshold and lesser than a threshold. The ROC curve is obtained by modifying the threshold values. After the curve is plotted, if the AUC (Area under the curve) score is good, then

the classifier performance is good. These are the different parameter metrics used for measuring the performance of any model. There are inbuilt commands for both f1 score and roc_auc_score in sklearn.metrics.

Building classifier:

After the data preprocessing step, we need to decide which model is to be used. Once the model is decided, the model has to be trained using the data given. In case the test data and train data not given separately, we have to split the test and train data. If the data is imbalanced, we have to use stratified-K-fold to split the given data into test and training data. Once this is done, the training data is obtained. And using this training data, we can perform cross validation to choose the best parameters of the model. This is the training stage. Once the model is trained, we need to use it to predict it for the testing data which we have separated in the first step. The classifiers implemented in this project are explained below, to understand their crux.

Support Vector Machines (SVM):

SVMs are one of the fundamental classifiers in Machine Learning. We try to find the decision boundary which classifies the datapoints correctly by trying to minimize the number of datapoints that are misclassified. The main difference between a simple minimum distance classifier is that, in SVM we consider the datapoints (support vectors) that are present very close to the decision boundary and try to maximize the margin between the decision boundary and these datapoints. There is a loss function and a penalty value for misclassifying these support vectors. This is the objective function for SVM. There are different types of kernel for SVM. They are:

i) Linear: This finds a linear boundary for the classification problem
ii) RBF: Radial Basis Function which creates a non-linear mapping (similar to phi-machine) for the features and tries to find the best non-linear decision boundary
iii) Poly: Creates a polynomial decision boundary based on different powers of each feature

There different parameters for SVM which need to be trained using cross validation. Even for different variants for SVM, the parameters vary. For example, Linear kernel has only the parameter 'C' (penalty). The RBF kernel has both 'C' and 'Gamma' and the poly kernel has parameters 'degree' and 'C'. The different parameters have to be trained using cross validation for good performance of the model.

```
##SVM Linear Training Model
clf=SVC(kernel='linear',probability=True,class_weight='balanced')
clf.fit(X_train,Y_train)
Pred_train=clf.predict(X_train)
Pred_test=clf.predict(X_test)
print("Classification model report for Training data")
train_report=classification_report(Y_train,Pred_train)
print(train_report)
print("Classification model report for Test data")
test_report=classification_report(Y_test,Pred_test)
print(test_report)
train_error=f1_score(Y_train,Pred_train,average='weighted')
train_error_2=roc_auc_score(Y_train,Pred_train,average='weighted')
test_error=f1_score(Y_test,Pred_test,average='weighted')
test_error_2=roc_auc_score(Y_test,Pred_test,average='weighted')
print("SVM Linear Model Training Error: f1={}, ROC_AUC={}".format(train_error,train_error_2))
print("SVM Linear Model Test Error: f1={}, ROC_AUC={}".format(test_error,test_error_2))
```

The SVM function is present in the sklearn package and the kernels have to be changed for implementing the poly kernel and the linear kernel.

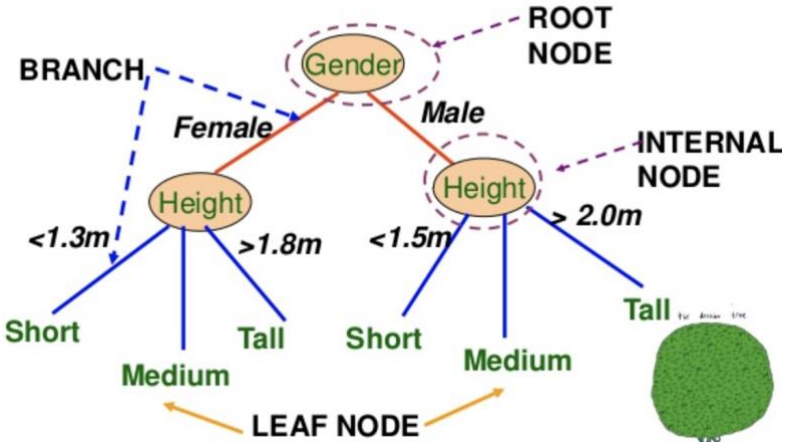
K-Nearest neighbor classifier:

This is one of the simplest supervised machine learning algorithm. The concept for KNN is very simple. For all the testing data points, we consider the 'k' nearest neighbors from the training data and assign the test datapoint to the class which contains the maximum number of datapoints in the 'k' nearest neighbors. The only parameter to train for kNN is the number of nearest neighbors to consider during the testing stage. We can use cross validation to decide on the number of nearest neighbors to be considered. Another parameter to be decided is the type of norm for considering the nearest neighbor. It can be any of the norms present, (p-norm, p varies between 0 and inf). In this project, l1 and l2 norms have been considered.

```
##Nearest Neighbors
knn=KNeighborsClassifier(n_neighbors=6,p=1,weights='uniform',algorithm='auto')
knn.fit(X_train,Y_train)
Pred_train=knn.predict(X_train)
Pred_test=knn.predict(X_test)
print("Classification model report for Training data")
train_report=classification_report(Y_train,Pred_train)
print(train_report)
print("Classification model report for Test data")
test_report=classification_report(Y_test,Pred_test)
print(test_report)
train_error=f1_score(Y_train,Pred_train,average='weighted')
train_error_2=roc_auc_score(Y_train,Pred_train,average='weighted')
test_error=f1_score(Y_test,Pred_test,average='weighted')
test_error_2=roc_auc_score(Y_test,Pred_test,average='weighted')
print("KNN Training Error: f1={}, ROC_AUC={}".format(train_error,train_error_2))
print("KNN Test Error: f1={}, ROC_AUC={}".format(test_error,test_error_2))
```

Decision Trees:

Decision trees have a simple concept behind their working but they become complex networks when a number of decision trees. The following example can help understand the working of decision trees. Consider a gender classification problem.



Decision trees start with the leaf node and end with the root, in this case which is the gender. The leaf nodes are the feature nodes. In this case, height and weight are the leaf nodes. We train the network to get the weights for each node and getting the depth of the tree. These parameters are passed as arguments to the decision tree classifier in the sklearn package. We can also decide on the number of splits in the decision tree to design the best decision tree for the given problem. We

can also find the threshold for the final root node to decide whether to consider it as class 1 or class 2 in a two-class problem. Thus, the parameters to be tuned are splits, threshold and depth.

Random Forest Classifier:

Random Forest classifiers are just an extent of the decision tree classifiers. In decision trees, only one tree is used for the classification problem. But in random forest classifier, a number of decision trees are trained for the classification problem. Another important difference is that, in decision trees we use all the features in the tree. But in random forest, the number of features and the number of datapoints considered for each of the trees are different. Thus, all the trees will give different predictions. We can use another pooling method to choose the class that was given as the output by the maximum number of trees. Using cross validation, we can train random forest classifier for different number of trees.

```
##Random Forest
rfc=RandomForestClassifier(n_estimators=14,bootstrap=True,class_weight='balanced')
rfc.fit(X_train,Y_train)
Pred_train=rfc.predict(X_train)
Pred_test=rfc.predict(X_test)
print("Classification model report for Training data")
train_report=classification_report(Y_train,Pred_train)
print(train_report)
print("Classification model report for Test data")
test_report=classification_report(Y_test,Pred_test)
print(test_report)
train_error=f1_score(Y_train,Pred_train,average='weighted')
train_error_2=roc_auc_score(Y_train,Pred_train,average='weighted')
test_error=f1_score(Y_test,Pred_test,average='weighted')
test_error_2=roc_auc_score(Y_test,Pred_test,average='weighted')
print("Random Forest Training Error: f1={}, ROC_AUC={}".format(train_error,train_error_2))
print("Random Forest Test Error: f1={}, ROC_AUC={}".format(test_error,test_error_2))
```

Perceptron Classifier:

Perceptron classifier is a single layer architecture from which the artificial neural networks were designed. The idea of the perceptron classifier is to find the weight vectors for each feature. So when the test data comes in, the datapoint is multiplied with the weight vector to get the output value to which the threshold can be applied to get the final classification. The Perceptron classifier is developed to obtained the multilayer perceptron classifier.

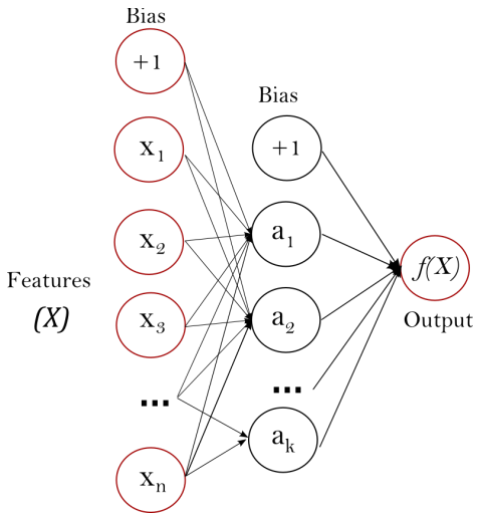
```
##Perceptron Training Model
percep = Perceptron(max_iter=10000,warm_start=False,class_weight='balanced')
percep.fit(X_train,Y_train)
Pred_train=percep.predict(X_train)
Pred_test=percep.predict(X_test)
print("Classification model report for Training data")
train_report=classification_report(Y_train,Pred_train)
print(train_report)
print("Classification model report for Test data")
test_report=classification_report(Y_test,Pred_test)
print(test_report)
train_error=f1_score(Y_train,Pred_train,average='weighted')
train_error_2=roc_auc_score(Y_train,Pred_train,average='weighted')
test_error=f1_score(Y_test,Pred_test,average='weighted')
test_error_2=roc_auc_score(Y_test,Pred_test,average='weighted')
print("Perceptron Model Training Error: f1={}, ROC_AUC={}".format(train_error,train_error_2))
print("Perceptron Model Test Error: f1={}, ROC_AUC={}".format(test_error,test_error_2))
```

Multilayer Perceptron:

Multilayer Perceptron is a part of artificial neural network. There are at least 3 layers in MLP. The first layer is the input layer, the second is the hidden layer and the third layer is the output layer. There can be a number of hidden layers. The algorithm has to be trained to decide the weights of neurons in each of the hidden layers. The meaning of this is to find of the weights for each of the

features of the dataset. So, the training process involves the training weights for each of the layers. A figure for MLP is given below:

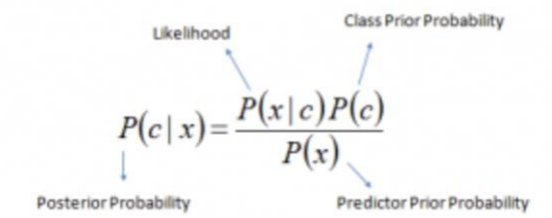
```
##MLP Classifier
gbc=MLPClassifier(hidden_layer_sizes=(160,))
gbc.fit(X_train,Y_train)
Pred_train=gbc.predict(X_train)
Pred_test=gbc.predict(X_test)
print("Classification model report for Training data")
train_report=classification_report(Y_train,Pred_train)
print(train_report)
print("Classification model report for Test data")
test_report=classification_report(Y_test,Pred_test)
print(test_report)
train_error=f1_score(Y_train,Pred_train,average='weighted')
train_error_2=roc_auc_score(Y_train,Pred_train,average='weighted')
test_error=f1_score(Y_test,Pred_test,average='weighted')
test_error_2=roc_auc_score(Y_test,Pred_test,average='weighted')
print("MLP Training Error: f1={}, ROC_AUC={}".format(train_error,train_error_2))
print("MLP Test Error: f1={}, ROC_AUC={}".format(test_error,test_error_2))
```



The training parameters for MLP includes the number of hidden layers. The other parameter is the tolerance parameter, which minimizes the loss function value. If the loss function value becomes lesser than the tolerance value, then the algorithm stops training. These parameters have to be obtained using the cross-validation procedure.

Naïve Bayes Classifier:

Naïve Bayes is an important Probabilistic classification algorithm. This is based on the Bayes theory. The Naïve Bayes classifier formula for a classification problem is given below:



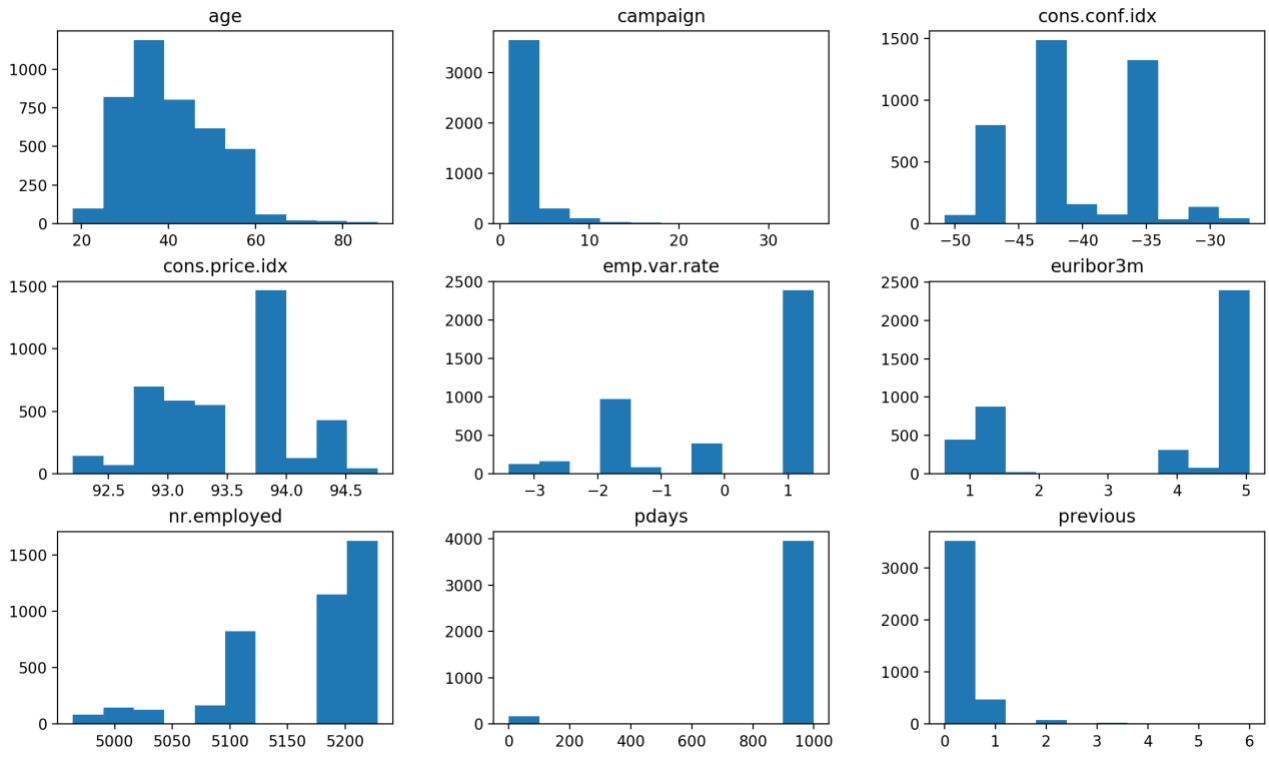
$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

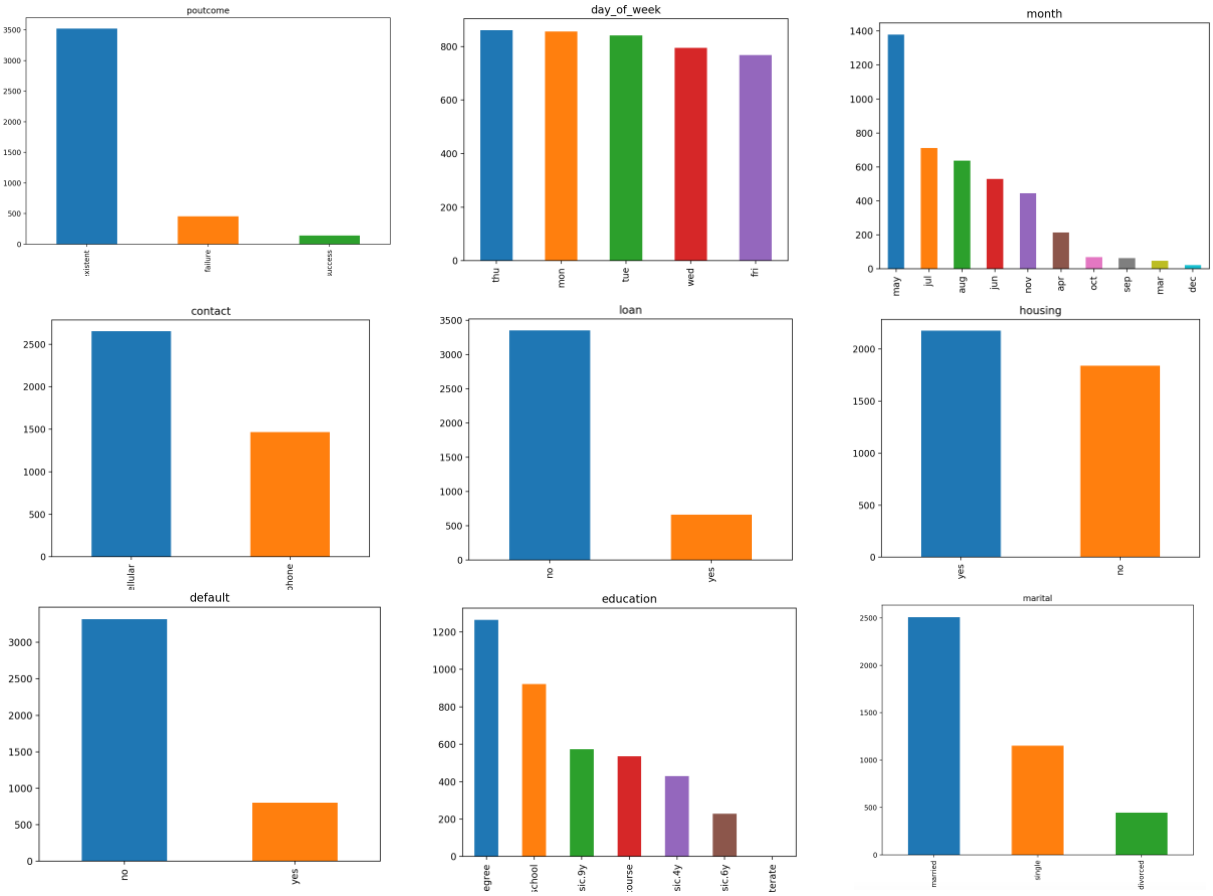
P(c/x) is the posterior probability of the class (c) given predictor (x), P(c) is the prior probability of each class, P(x/c) is the likelihood (probability of the predictor), P(x) is the prior probability of the predictor. The main assumption under which the Naïve Bayes classifier works is that, it assumes that the features are independent of each other. In most of the cases, it is impossible to get predictors which are completely independent. In most of the cases, Naïve Bayes works very well. There aren't any parameters that must be tuned with cross validation.

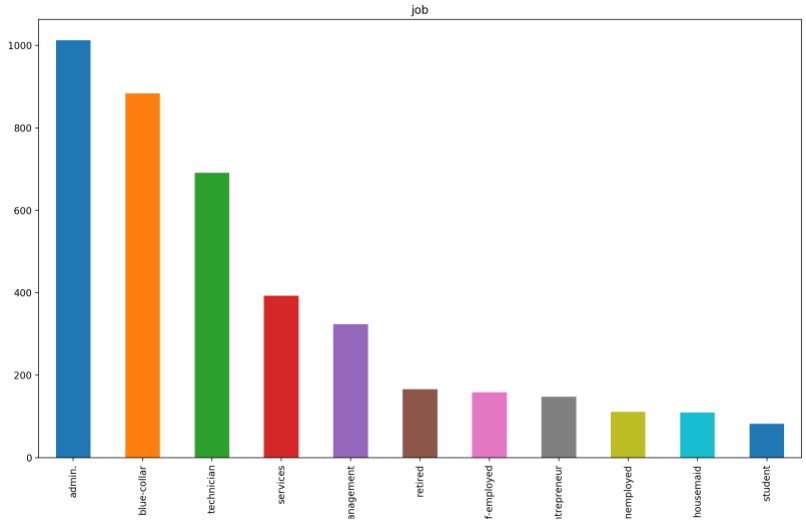
```
##Naive Bayes classifier
naiveB=GaussianNB()
naiveB.fit(X_train,Y_train)
Pred_train=naiveB.predict(X_train)
Pred_test=naiveB.predict(X_test)
print("Classification model report for Training data")
train_report=classification_report(Y_train,Pred_train)
print(train_report)
print("Classification model report for Test data")
test_report=classification_report(Y_test,Pred_test)
print(test_report)
train_error=f1_score(Y_train,Pred_train,average='weighted')
train_error_2=roc_auc_score(Y_train,Pred_train,average='weighted')
test_error=f1_score(Y_test,Pred_test,average='weighted')
test_error_2=roc_auc_score(Y_test,Pred_test,average='weighted')
print("Naive Bayes Model Training Error: f1={}, ROC_AUC={}".format(train_error,train_error_2))
print("Naive Bayes Model Test Error: f1={}, ROC_AUC={}".format(test_error,test_error_2))
```

Experimental Results:

In the first part of the experimental results we talk about the importance of each feature for the classification problem. The histograms of all the features in the dataset are given below:

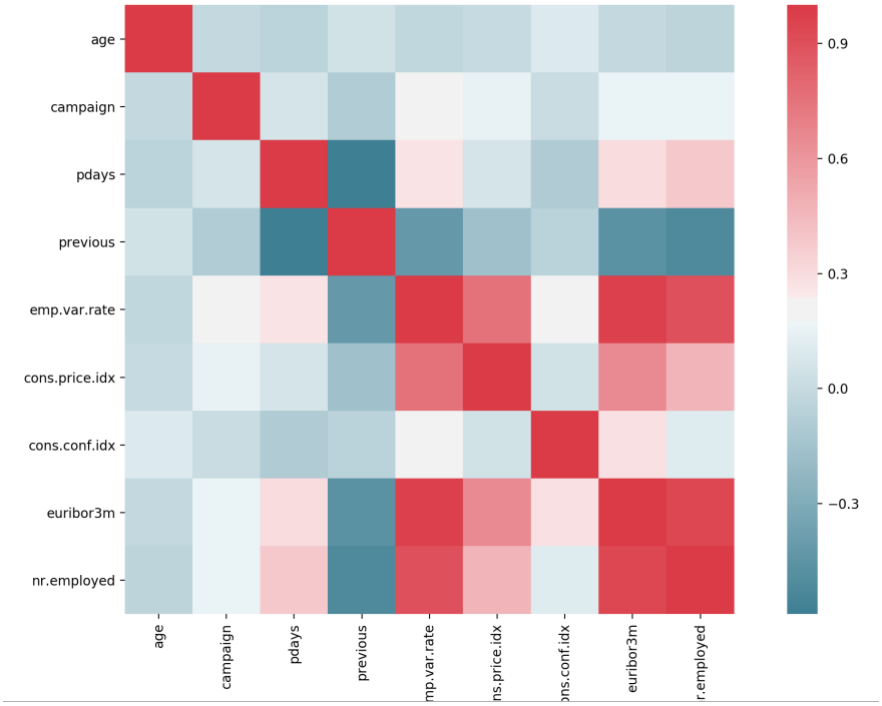

age
campaign
cons.conf.idx
cons.price.idx
emp.var.rate
euribor3m
nr.employed
pdays
previous


poutcome
day_of_week
month
contact
loan
housing
default
education
marital



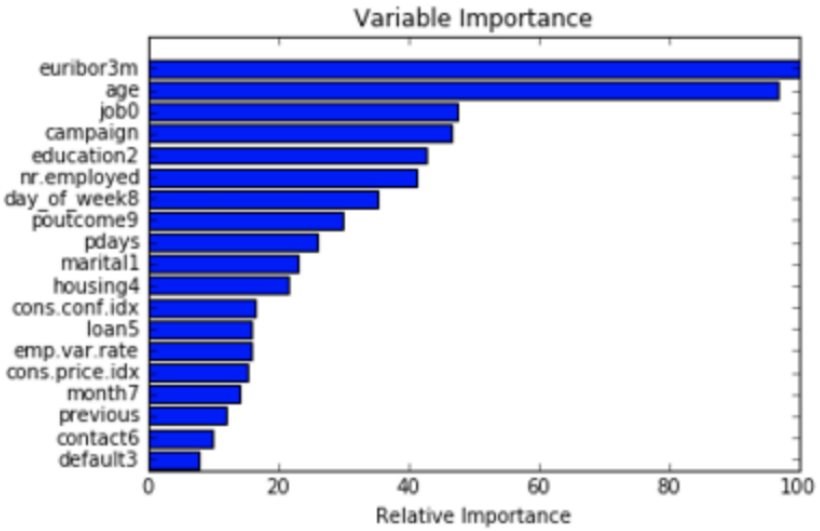
From the given histograms, we get an idea about distribution of each of the features. From this we get an idea about what parameters can help us good classification results and not.

The flowing plot conveys the above discrimination power of 2 features taken together. It gives the correlation of each feature with another. This is also obtained only for the numerical values as it is not possible to get the categorical values without encoding them.



From the figure given above, we can identify the correlation between each of the features. We can identify which features are similar to each other. Hence, by replacing the features which have a high correlation with just one from that particular group, we can still get the higher accuracy by

just using that single frequency. For this specific dataset, we can see that euribor3m and emp.var.rate are highly correlated, euribor3m and nr.employed are highly correlated. Hence, by taking one from these features can give us equally good classification accuracies without using these features as well. The following image gives an idea about the variance importance for each of the features.



We can see that euribor3m is the most important feature and using that feature itself will give us high classification accuracies, which we will be seeing in the results.

The parameters for the models were obtained for each of the model using the cross validation techniques. The parameters for each of the trained model are given below:

SVM Linear Kernel	C=13.73	
SVM RBF Kernel	C=100	Gamma=0.05
SVM Poly Kernel	C=210, Degree=5	Gamma=4.4
KNN classifier	Nearest Neighbors: 6	
MLP	Hidden Layers: 1	Neurons: 160
Random Forest	Number of trees: 14	

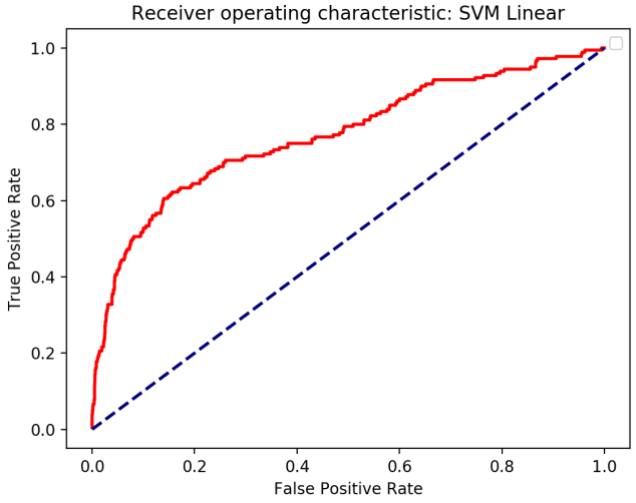
With the parameters mentioned above, the models were run and the following results were obtained. The results obtained with each of the model along with their snippets are given below. The results are obtained with the MinMax scaler and imputing the missing data with the Mode value. The model performance along with the ROC curves are given.

1) Perceptron Classifier:

```
Perceptron Model

Classification model report for Training data
             precision    recall  f1-score   support

          0       0.96      0.30      0.46      2038
          1       0.14      0.90      0.24       248

avg / total       0.87      0.37      0.43      2286

Classification model report for Test data
             precision    recall  f1-score   support

          0       0.94      0.29      0.45      1360
          1       0.13      0.85      0.22       165

avg / total       0.85      0.35      0.42      1525

Perceptron Model Training Error: f1=0.43396329564686986, ROC_AUC=0.5999893159011048
Perceptron Model Test Error: f1=0.42204627294696173, ROC_AUC=0.5705659536541889
```

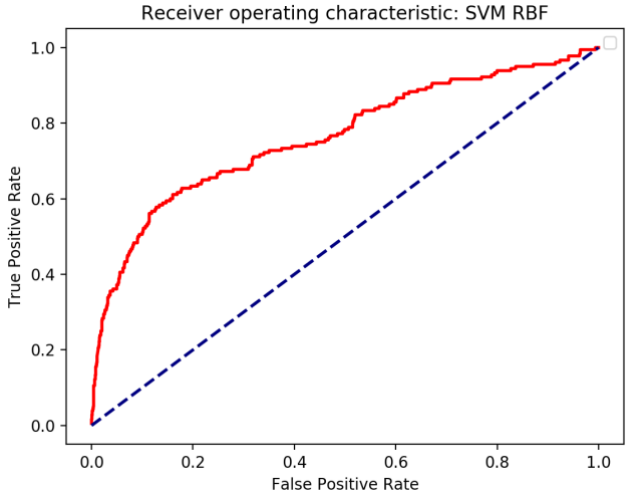
2) SVM Linear Kernel

```
Classification model report for Training data
             precision    recall  f1-score   support

          0       0.95      0.87      0.91      2038
          1       0.36      0.61      0.45       248

avg / total       0.88      0.84      0.86      2286

Classification model report for Test data
             precision    recall  f1-score   support

          0       0.94      0.87      0.90      1360
          1       0.34      0.58      0.43       165

avg / total       0.88      0.83      0.85      1525

SVM Linear Model Training Performance: f1=0.8582739363927933, ROC_AUC=0.7394207635569344
SVM Linear Model Test Performance: f1=0.852353741459588, ROC_AUC=0.723629679144385
```



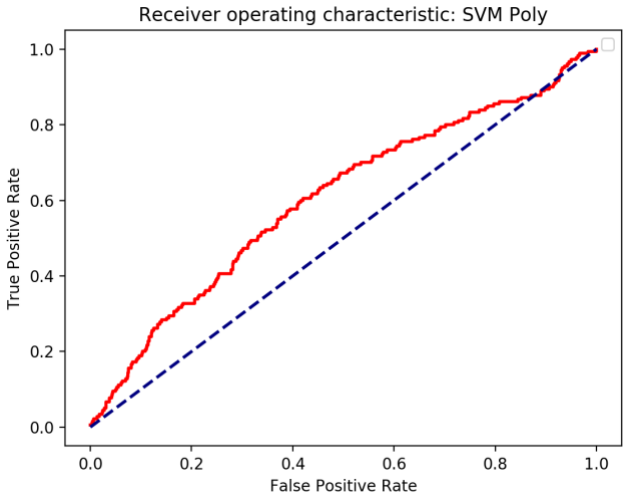
3) SVM RBF Kernel (BEST RESULT FOR ROC AUC score)

```
Classification model report for Training data
             precision    recall  f1-score   support

          0       0.96      0.89      0.92      2038
          1       0.41      0.66      0.51       248

avg / total       0.90      0.86      0.87      2286

Classification model report for Test data
             precision    recall  f1-score   support

          0       0.94      0.88      0.91      1360
          1       0.36      0.58      0.45       165

avg / total       0.88      0.85      0.86      1525

SVM RBF kernel Training Performance: f1=0.8743852227753465, ROC_AUC=0.773235936560195
SVM RBF Kernel Test Performance: f1=0.8598521154918315, ROC_AUC=0.726849376114082
```



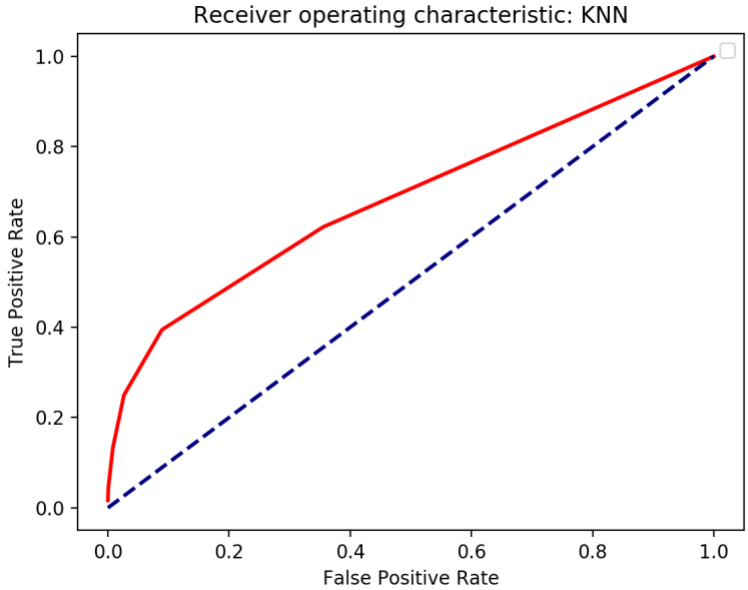
4) SVM Poly Kernel

```
Classification model report for Training data
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      2038
          1       1.00      1.00      1.00       248

avg / total       1.00      1.00      1.00      2286

Classification model report for Test data
             precision    recall  f1-score   support

          0       0.91      0.87      0.89      1360
          1       0.21      0.27      0.23       165

avg / total       0.83      0.81      0.82      1525

SVM Poly Kernel Training Performance: f1=1.0, ROC_AUC=1.0
SVM Poly Kernel Test Performance: f1=0.8188159872960044, ROC_AUC=0.5723930481283422
```



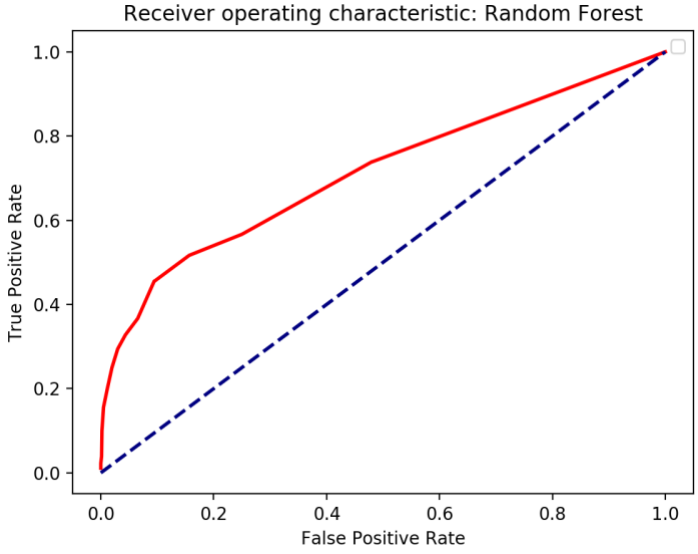
5) KNN classifier Model

```
Classification model report for Training data
             precision    recall  f1-score   support

          0       0.91      0.99      0.95      2038
          1       0.78      0.16      0.27       248

avg / total       0.89      0.90      0.87      2286

Classification model report for Test data
             precision    recall  f1-score   support

          0       0.90      0.99      0.95      1360
          1       0.67      0.12      0.21       165

avg / total       0.88      0.90      0.87      1525

KNN Training Performance: f1=0.8748481372496579, ROC_AUC=0.5779464370508721
KNN Test Performance: f1=0.8655808375827898, ROC_AUC=0.5569295900178253
```



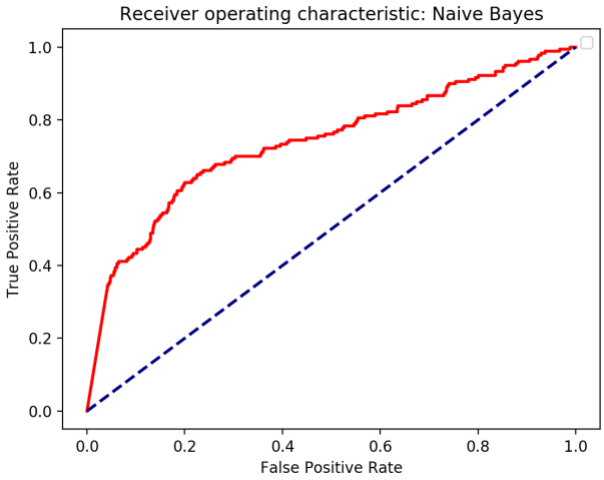
6) Random Forest Classification Model

```
Classification model report for Training data
             precision    recall  f1-score   support

          0       0.99      1.00      0.99      2038
          1       1.00      0.90      0.95       248

avg / total       0.99      0.99      0.99      2286

Classification model report for Test data
             precision    recall  f1-score   support

          0       0.91      0.98      0.94      1360
          1       0.55      0.20      0.29       165

avg / total       0.87      0.90      0.87      1525

Random Forest Training Performance: f1=0.9892651244142231, ROC_AUC=0.9516129032258065
Random Forest Test Performance: f1=0.8733474539387784, ROC_AUC=0.5900735294117646
```



7) Gaussian Naïve Bayes Classification model

```
Classification model report for Training data
             precision    recall  f1-score   support

          0       0.95      0.78      0.86      2038
          1       0.27      0.67      0.39       248

avg / total       0.88      0.77      0.81      2286

Classification model report for Test data
             precision    recall  f1-score   support

          0       0.94      0.77      0.85      1360
          1       0.25      0.62      0.36       165

avg / total       0.87      0.75      0.80      1525

Naive Bayes Model Training Performance: f1=0.807876108733091, ROC_AUC=0.7262377726423755
Naive Bayes Model Test Performance: f1=0.795197588701501, ROC_AUC=0.697415329768271
```



8) Decision Tree

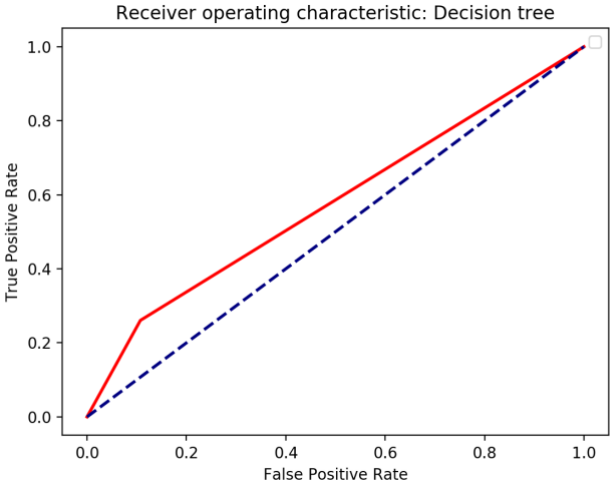
Classification model report for Training data

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2038
1	1.00	1.00	1.00	248
avg / total	1.00	1.00	1.00	2286

Classification model report for Test data

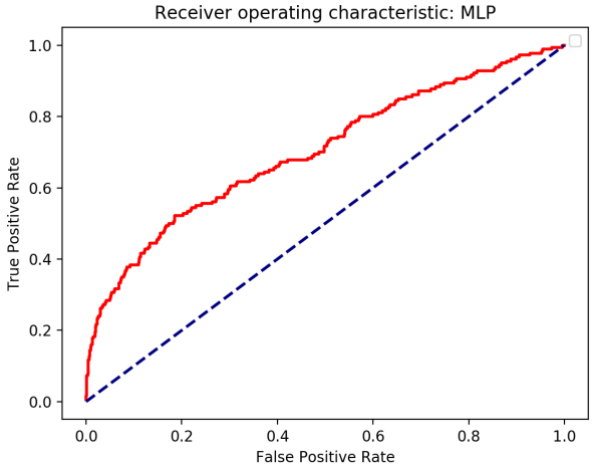
	precision	recall	f1-score	support
0	0.92	0.89	0.91	1360
1	0.30	0.37	0.33	165
avg / total	0.85	0.84	0.84	1525

Decision Tree Training Performance: f1=1.0, ROC_AUC=1.0
Decision Test Performance: f1=0.8449538748437697, ROC_AUC=0.6319073083778965



9) MLP

```
Classification model report for Training data
             precision    recall  f1-score   support

          0       0.97      1.00      0.99      2038
          1       0.99      0.78      0.87       248

avg / total       0.98      0.98      0.97      2286

Classification model report for Test data
             precision    recall  f1-score   support

          0       0.92      0.96      0.94      1360
          1       0.46      0.27      0.34       165

avg / total       0.87      0.89      0.87      1525

MLP Training Performance: f1=0.974222849474635, ROC_AUC=0.8906383551236189
MLP Test Performance: f1=0.8734451540585861, ROC_AUC=0.6145833333333334
```



Comparison Table:

Classifier	F1 test score	AUC test Score
Perceptron	0.422	0.457
SVM Linear	0.85	0.72
SVM RBF	0.86	0.72
SVM Poly	0.81	0.57
KNN	0.86	0.56
Random Forest	0.87	0.59
Decision Tree	0.85	0.63
Naïve Bayes	0.79	0.69
MLP	0.87	0.61

The results of the 9 classifiers are summarized above. Also, the weighted f1 score for all the classifiers are very similar, and hence we cannot use this feature to decide the best classifier. We consider the roc auc score for deciding the best classifier. Both SVM Linear and SVM RBF kernels

have a very good roc score. And SVM RBF kernel is chosen arbitrarily between SVM Linear kernel and SVM RBF kernel. The results are taken from the test classifying accuracies. We can use this model to further analyze the feature selection, dimension reduction as well as use the oversampling and under sampling techniques to get the best model selection variant.

Feature Selection:

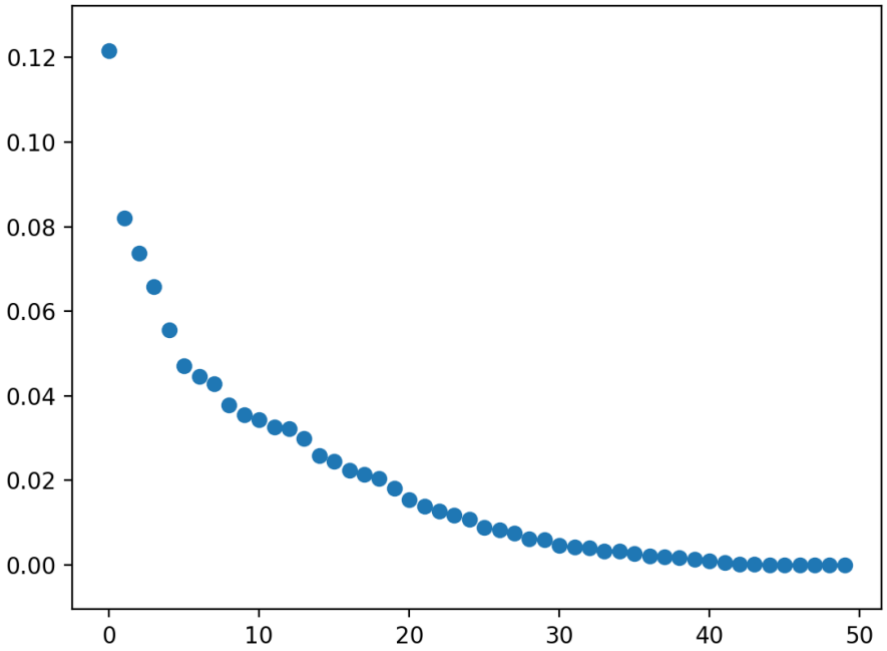
K-Best Features	ROC Scores
5	0.7112
10	0.7214
15	0.721
25	0.7289
30	0.716
50	0.725

From the results, we can see that ROC scores are very similar to that of the original dimension score. This is because of the fact that the most important features in the dataset provide very good classification results. And hence even if the features with the lower importance is removed, the accuracy does not change.

Next, we try different PCA components to check whether the number of components affect the classification accuracy or not.

PCA Components	ROC Scores
5	0.677
10	0.701
15	0.7155
40	0.70
50	0.723

As we can see, dimension reduction does not change the accuracy. The accuracy is very good for the dataset without any dimension reduction. The scatter plot conveying the proportion of variance explained for pca when the dimension is 50 is given below:



As we can see, since some of the feature axes are more important in terms of variance when compared to the other features, even if the dimensionality is reduced the result will be the same. Hence, there is no improvement in applying PCA to this dataset. And since the dataset is not very large, training 56 features for 4000 data points is very fast and does not make the model training slower.

In the following section, different types of imputing are performed and the ROC scores are compared.

Type of Imputing	ROC Score
SVM Imputing	0.7232
Removing the unknown datapoints	0.688

When other imputing methods are performed, the ROC score decreases. Hence the mode imputing gives better result when compared to the other types of imputing such as SVM Imputing. Also, removing the unknown datapoints also gives a lower ROC score. Hence the best type of Imputing for this particular dataset would be Mode imputing. Another variant that can be performed is to use the standard scaler instead of the minmax scalar. There is not any difference when the scaler is changed. Both Minmax and Standard scalar give very similar results.

We also try different methods for balancing the given imbalanced dataset via two approaches. One is Over sampling datapoints from the class which has lesser number of datapoints and the second

method is under sample the datapoints from the class which has higher number of datapoints. The results are mentioned below.

Oversampling	ROC: 0.704
Under sampling	ROC: 0.723

Both the methods of over sampling and under sampling give lower scores when compared to that of the obtained score without using these methods. This is because, the 'class_weight' parameter in each of the classifier is made to be 'balanced' and hence there is no advantage in over sampling or under sampling. Over sampling and Under sampling were performed using the package named 'imblearn'. For over sampling, SMOTE was used and for under sampling KNN was used. SMOTE uses both bootstrapping and KNN to sample data from the class that has less datapoints. For under sampling, the datapoints from the class that has larger number of datapoints is taken, and if there are too many points in the surrounding region (found by KNN), then some of the points are removed, effectively under sampling the datapoints to be taken for classification.

In the next part, we will remove certain columns and check if the accuracy is the same or not:

Dropped feature	ROC score
Default	0.702
Default, Day, Month	0.697

We can see that the accuracy is reducing due to the removal of these features. The accuracy does not drop that much even when more than one features such as 'default', 'day' and 'month'. This is because of the fact that the discrimination power for default is low. Since we are one hot encoding the dataset, for this particular dataset, dropping particular columns does not influence the classification model that much.

Best Classifier and Specifications:

Thus, the best classifier for bank dataset is SVM with RBF kernel. The imputing followed is 'mode' and there is no dimension reduction or PCA involved before the data is used to train the model. The highest test roc score is 0.73 obtained with trained model.

Conclusion:

Thus, the objective of the project was achieved. The best classifier for the bank dataset was designed and tested on the test data and the performance metrics are obtained. This project gave practical idea about how the data can be processed and information can be obtained from raw data. Another important thing to note is, the most important part of designing the classifier is to handle the missing data and categorical values. Once these issues are resolved, then designing the classifier is an easy job.

References:

[1] Imputer, S. (2018). *Sklearn: Categorical Imputer*. [online] Stackoverflow.com. Available at: https://stackoverflow.com/questions/42846345/sklearn-categorical-imputer?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa [Accessed 1 May 2018].