EE 569 PROJECT
DATE: 5/1/2018

CLASSIFICATION OF THE BANK MARKETING DATASET

AKSHATA BHAT
akshatab@usc.edu

ABSTRACT

The aim of this project was to understand and analyze the given dataset and create a pattern recognition system to perform predictions based on the given dataset. I have chosen the Bank Marketing dataset and my aim is to predict if a client will subscribe to the bank deposit after the marketing call has been made to them.
After observing the data I found that there were 19 features in the bank marketing data. It is a 2-class dataset, with “yes” and “no” as the labels. They indicate if the client will subscribe or not subscribe to a long term deposit respectively.
Out of the 19 features, only the features(columns in the .csv file) which conveyed information were retained, and others were dropped(‘pdays’ and ‘default’). Preprocessing operations were done on the labels - like label encoding, imputation. One hot encoding was performed to increase the feature dimension. Standardization and feature selection were performed and finally different classifiers were tried out. I found that the Random forest classifier performed the best when compared to the remaining classifiers as indicated by its high F1 and AUC scores

APPROACH

The brief procedure which was followed to perform classification on the bank marketing dataset is listed below:

1) Split the dataset into train and test
2) Preprocess the Training and Testing data to replace the categorical values of some of the features by numerical values
3) Impute the unknown values in the train and test data by using the most frequent value method
4) Perform one hot encoding for some of the features to ensure every feature value is given equal weightage
5) Obtain the standard deviation and mean value of the training data and using these parameters to standardize the training and testing parameters
6) Try different classifiers to perform classification and evaluate each classifier by using metrics like F1_score, AUC score and ROC curve

PREPROCESSING

Key observations and challenges faced with the banking dataset:

1) The dataset had 19 features, which meant there was a lot of data to be handled and it would be wise to drop which might not help in the classification process.
2) On closely observing the data, it was observed that the 'pdays' feature had the same value for every data point and it was not providing any extra information to differentiate the target data point from the rest.
3) The 'default' feature was also dropped because it has just two values - 'no' and 'unknown'. This meant that even if all the unknown values for the 'default' feature were imputed, it would result in a feature which had all the same value and hence not convey any new information.
4) It was also observed that for the 'education' feature, the 'illiterate' appeared only once and due this reason, when the dataset was split into train and test data, the set into which the data point with education category set to illiterate was assigned, that dataset would have one extra feature column for 'education' compared to the other dataset and there would be a feature dimension mismatch between the train and test data. Hence I dropped the data point which had the 'illiterate' value.
5) All the features which had categorical values i.e string values, were assigned numerical integer values starting from 0 to the number of unique values in that feature.
6) It was ensured that the 0 numerical value was assigned to the data points whose values were 'unknown', in order to make it easy while performing imputation.
7) Perform imputation by using the most frequent strategy using the **Imputer** function from the **sklearn.preprocessing** module for the features which had unknown values. These features were:
    a) 'job'
    b) 'marital'

c) ‘education’
d) ‘housing’
e) ‘loan’

8) The next step in the preprocessing stage was to perform one-hot encoding for the features which had numerical labels, but there was no relationship between the numerical label values and kind of data they were representing. Ex. If one of the features were temperature and there were 3 string values - low, medium and high assigned to the data points with regard to this feature - then if we assigned 0, 1, 2 to the features, then it would make sense to assign higher value to higher temperature.
9) So going by the above logic, I felt the need to perform one-hot encoding for all the features which have categorical values - contact, poutcome, job, marital, housing, education, loan, month, day_of_week by sing **get_dummies** function from pandas.
10) The last step is standardization, where the standard deviation and mean values are obtained from the training data and use that use to standardize the training and testing data.
11) The above step is performed by using the **train_test_split** function from the **sklearn.model_selection** module

FEATURE SPACE DIMENSIONALITY ADJUSTMENT

The feature space dimensionality adjustment can be done in multiple ways. Some of them are:

1) One-hot encoding - Feature dimension expansion
2) Linear discriminant analysis
3) PCA - Principal component analysis

I performed one hot encoding and Principal component analysis data components in order to expand the data. Principal component analysis did not show any improvements in the score and hence the method was dropped was later dropped.

As mentioned above the one-one hot -coding was performed in order to perform feature space expansion.

HANDLING DATA IMBALANCE

It is known that classifiers do not work properly if there is a large data imbalance. Hence we need to balance the data such that it has equal number of samples for every class. Hence undersampling or oversampling methods are used. I performed oversampling and observed that the F1 scores and AUC scores which had a difference of around 0.2 - 0.3 had approximately similar or values after performing oversampling there by confirming that the bank dataset had imbalanced data.

I used the **SMOTEENN** method from the **imblearn.combine** module to perform oversampling. The SMOTE (Synthetic Minority Oversampling Technique) generates new samples by performing interpolation between the marginal outliers and inliers[7]. It generates noisy samples and this issue is solved by cleaning up the resultant space using methods like the edited-nearest neighbours used in this project.

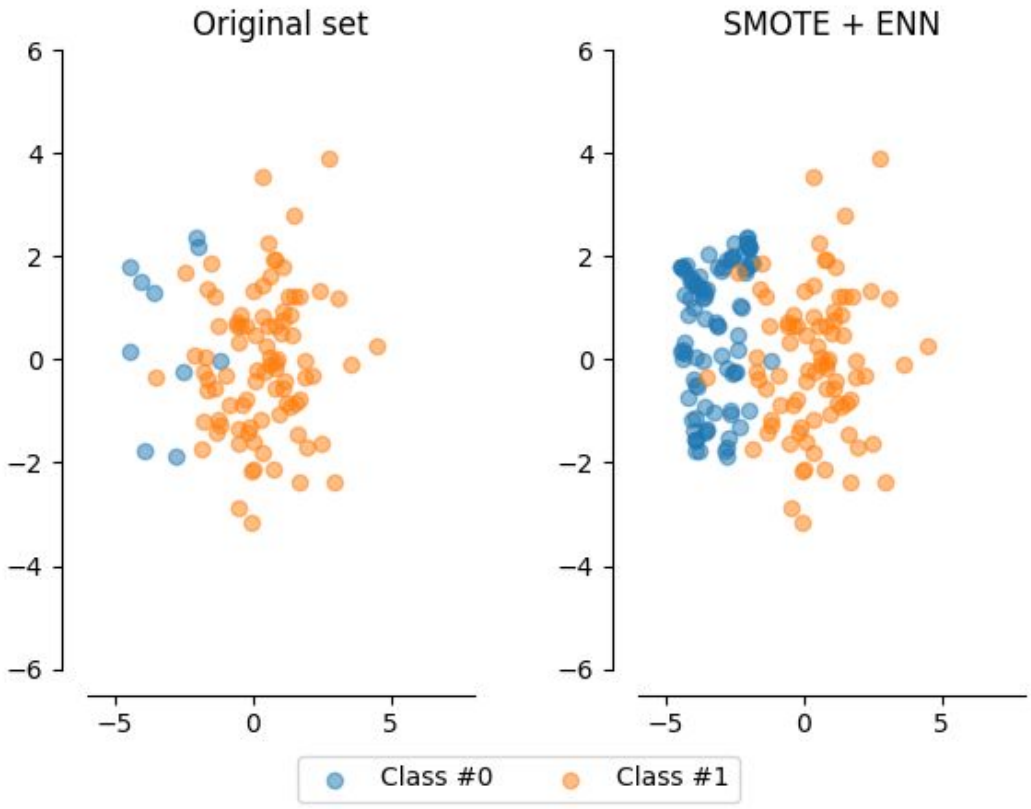


Figure to demonstrate how oversampling with SMOTE+ENN helps to generate balanced data.[7]

CROSS VALIDATION

The cross-validation step was implemented by using **sklearn's, GridSearchCV,** by passing the number of parameters and the parameter range as arguments to the function. For the Cross validation step, the best parameter settings were chosen by the GridSearch CV function among the allowed range of parameter values as given by the param_grid argument for every classifier. This step helps us choose the optimal parameters which give the best prediction, without overfitting or under fitting the model.

TRAINING AND CLASSIFICATION

I tried out multiple classifiers before finding the best performing classifier which can be inferred by the F1 and AUC scores. The different classifiers I experimented with are:

RANDOM FOREST CLASSIFIER

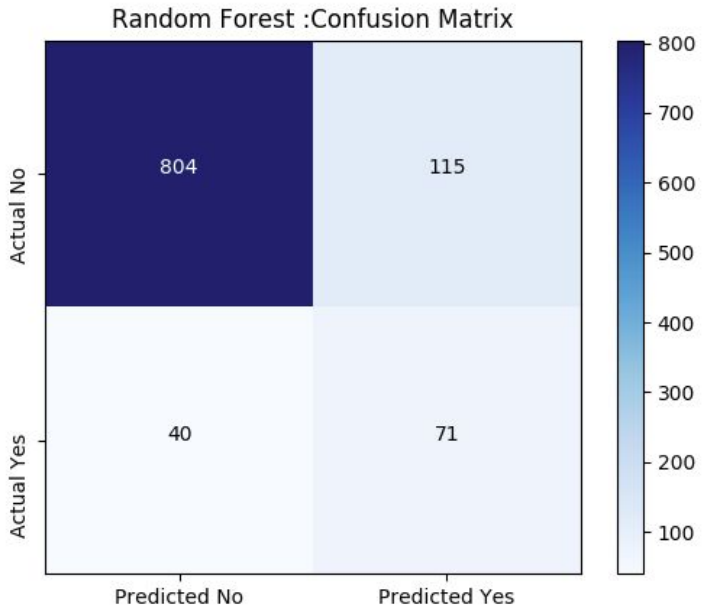
The random forest classifier is an ensemble based machine learning method, which works by constructing multiple decision trees at the time of training and providing the output which is either the mode or mean of the prediction made by all the trees in the algorithm.

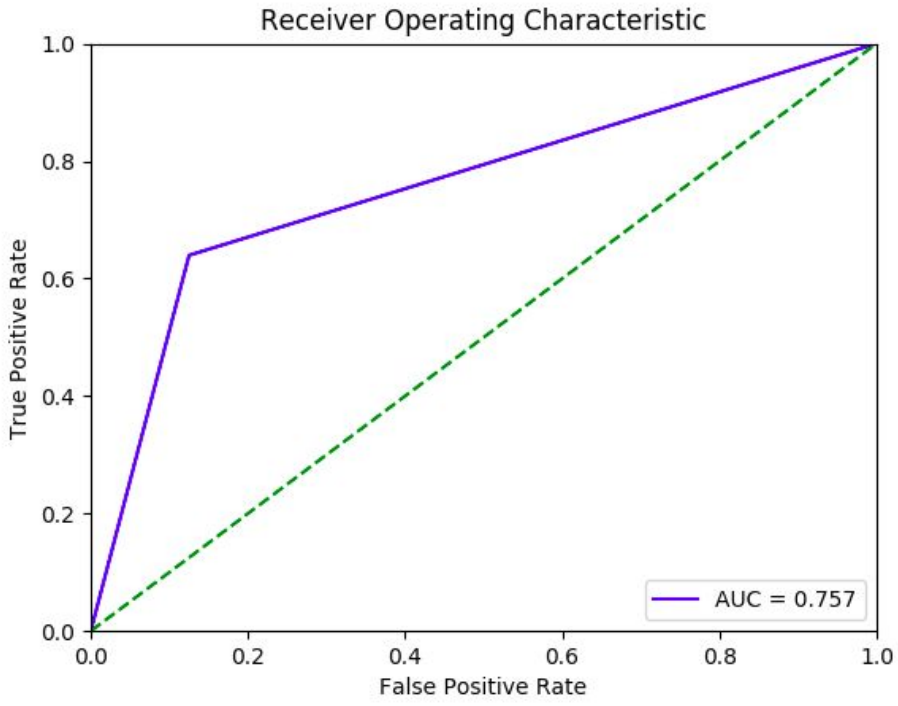
The **RandomForestClassifier** from the **sklearn.ensemble** module was used in the project and the following settings were used as parameters:
parameter_grid = {'n_estimators': range(1, 30), 'max_features': ['auto', 'sqrt', 'log2']}

The results obtained from the Random Forest classifier are:
Training data shape:
X_train: (3088, 19)
y_train: (3088,)
Test data shape:
X_test: (1030, 19)
y_test: (1030,)
Training data shape after preprocessing:
X_train: (3088, 52)
y_train: (3088,)
Test data shape after preprocessing:
X_test: (1030, 52)
y_test: (1030,)
Performing classification using Random Forest ...
Training data- Actual Label size: (4079,)
Training data- Predicted Label size: (4079,)
Random Forest Train Accuracy: 1.000
Random Forest Test Accuracy: 0.850
Random Forest F1 Score: 0.865
Random Forest AUC Score: 0.757
Classification Report scores:

Classification Report scores:

	Precision	Recall	F1-score	Support
yes	0.95	0.87	0.91	919
no	0.38	0.64	0.48	111
avg/total	0.89	0.85	0.87	1030





ROC curve obtained for Random Forest classifier

SUPPORT VECTOR MACHINE CLASSIFIER:

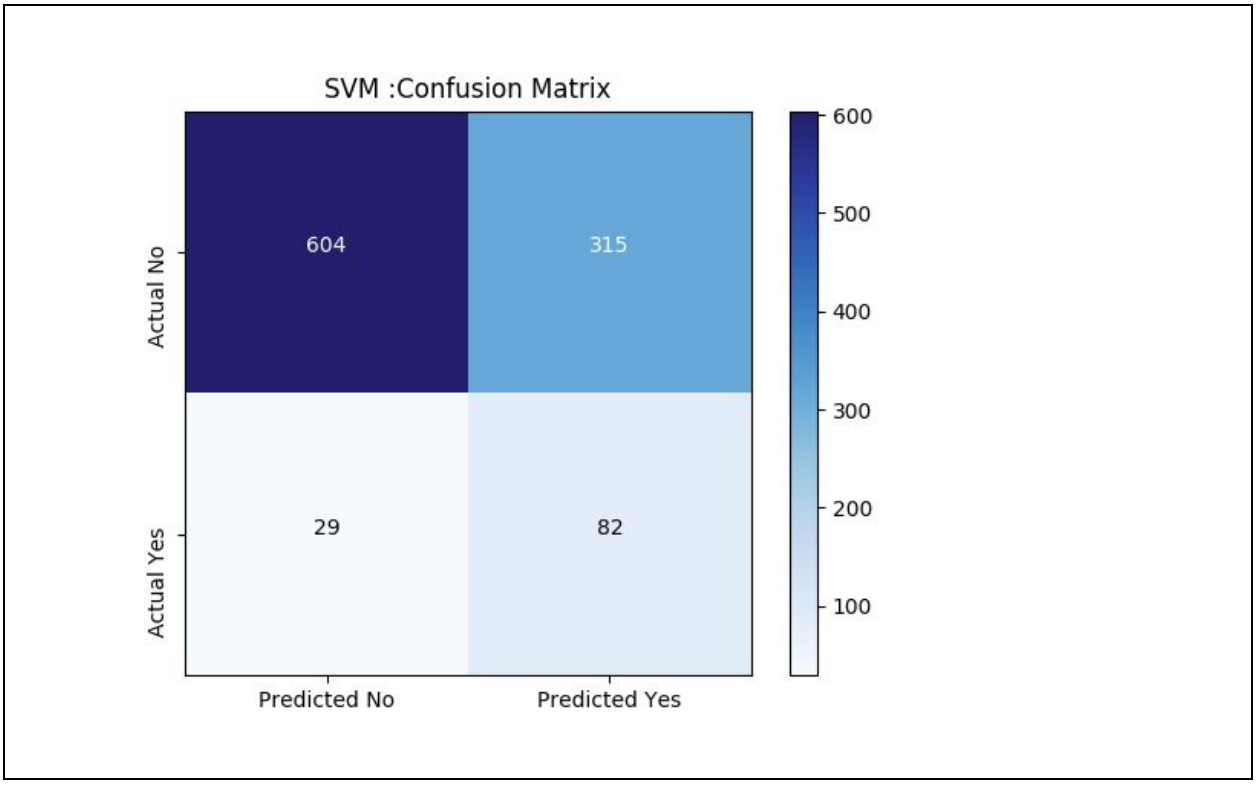
SVM is a supervised learning model which is machine learning algorithms which understand and analyze the data for problems on regression analysis and classification. SVMs also perform non-linear classification, by mapping the inputs in a high dimensional space.

SVM was implements using the **SVC()** from **sklearn.svm** and following set of parameters parameter_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4], 'C': [1, 10, 100]}, {'kernel': ['linear'], 'C': [1, 10, 100]}]

The results obtained from the Support vector machine classifier are:

SVM Train Accuracy: 0.796
SVM Test Accuracy: 0.666
SVM F1 Score: 0.729
SVM AUC Score: 0.698
Classification Report scores:

	Precision	Recall	F1-score	Support
yes	0.95	0.66	0.78	919
no	0.21	0.74	0.32	111
avg/total	0.87	0.67	0.73	1030



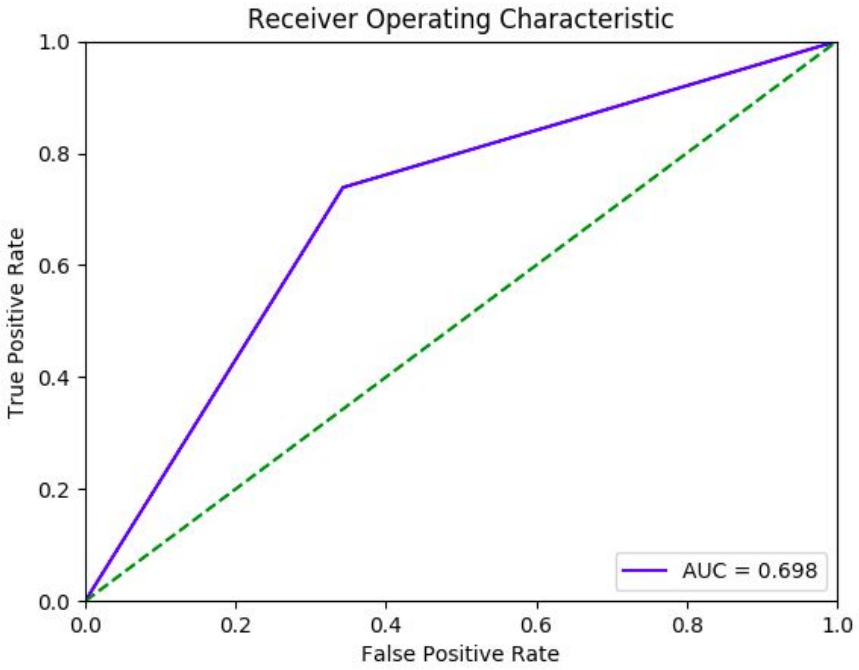


Figure: ROC score for SVM classifier

NAIVE BAYES

These are a form of probabilistic classifiers which are based on the Bayes theorem of probability with the assumption strong independence among features.

Implemented using: **GaussianNB** from **sklearn.naive_bayes**

Observations:

Naive Bayes Train Accuracy: 0.661

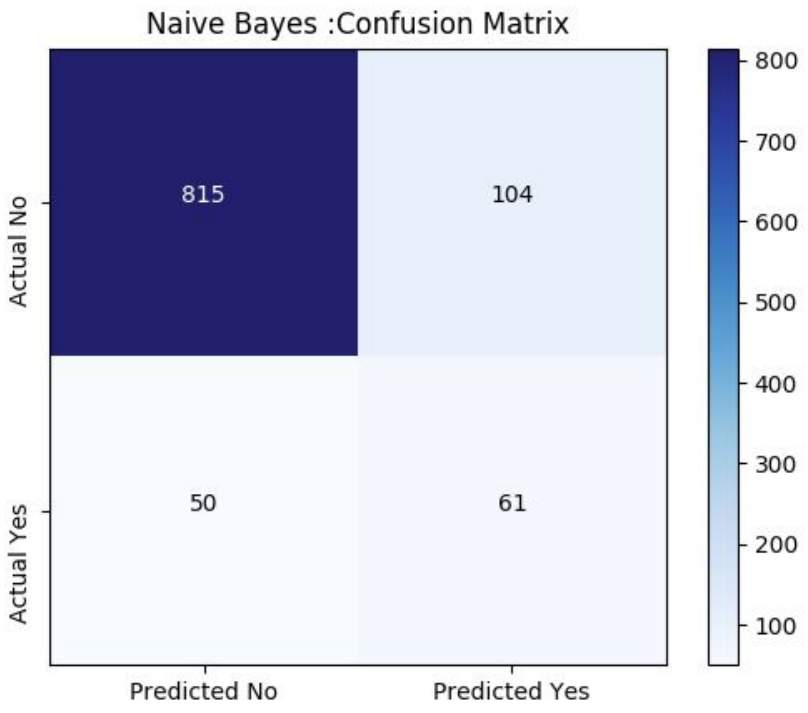
Naive Bayes Test Accuracy: 0.850

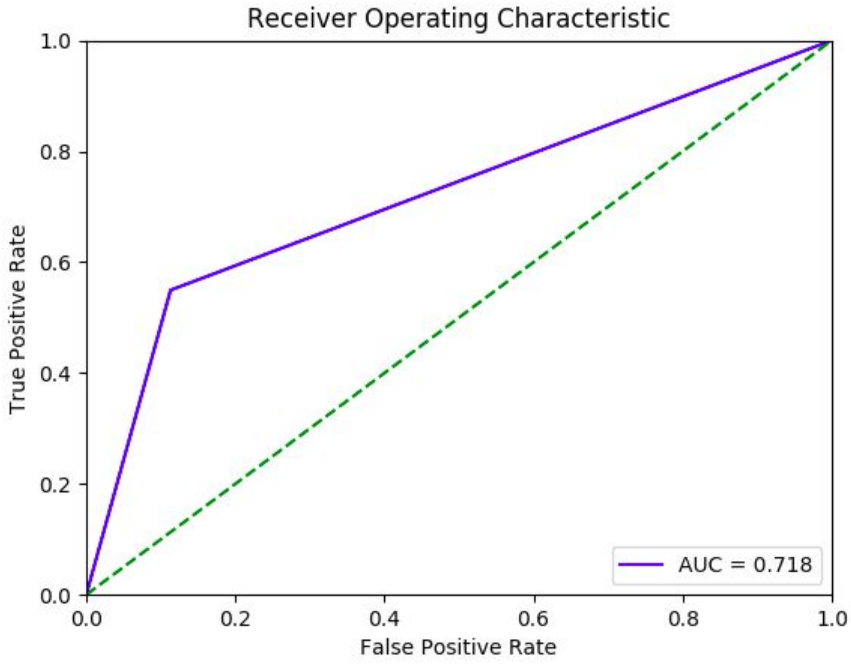
Naive Bayes F1 Score: 0.863

Naive Bayes AUC Score: 0.718

Classification Report scores:

	Precision	Recall	F1-score	Support
yes	0.94	0.89	0.91	919
no	0.37	0.55	0.44	111
avg/total	0.88	0.85	0.86	1030





ROC curve for Naive bayes classifier

PERCEPTRON CLASSIFIER

It is a linear classifier which predicts based on the linear predictor function which takes the dot product of the weights with the feature vector.

Implemented by: using **sklearn's linear_model.Perceptron()**

Perceptron Train Accuracy: 0.728

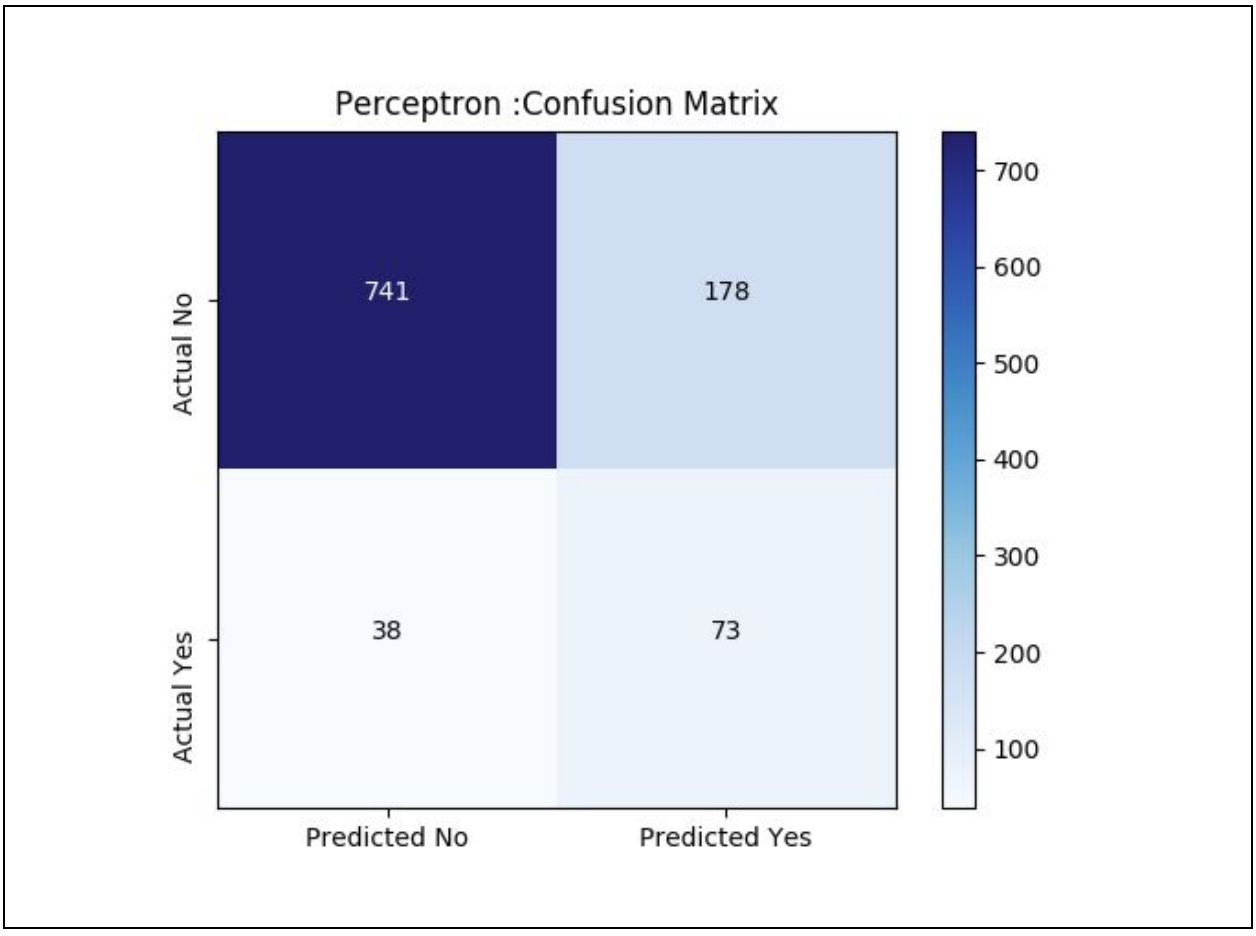
Perceptron Test Accuracy: 0.790

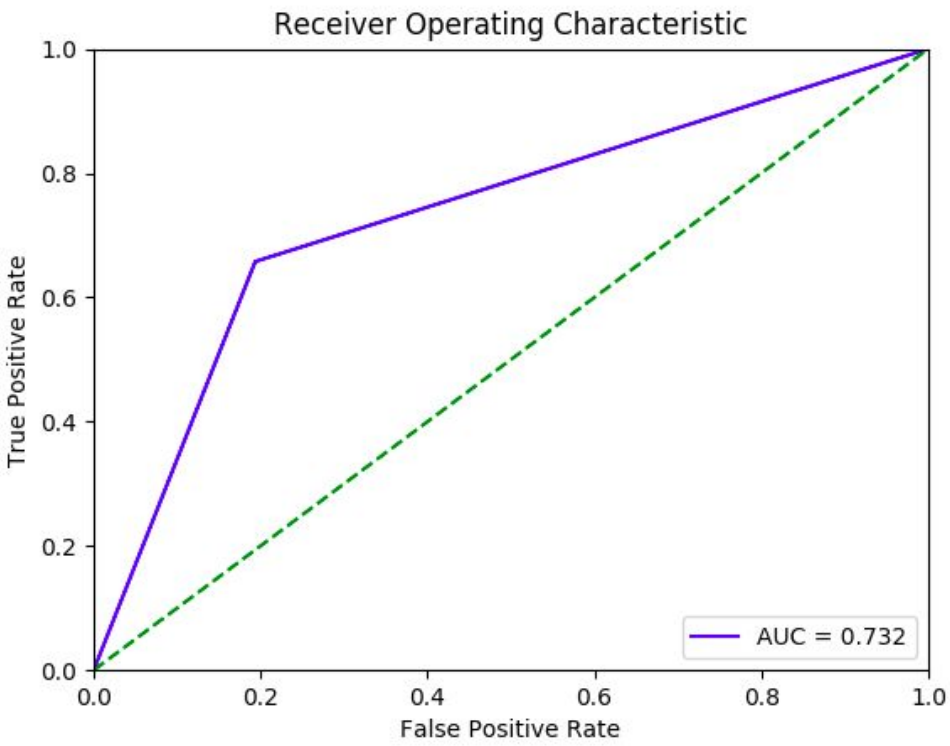
Perceptron F1 Score: 0.822

Perceptron AUC Score: 0.732

Classification Report scores:

	Precision	Recall	F1-score	Support
yes	0.95	0.81	0.87	919
no	0.29	0.66	0.40	111
avg/total	0.88	0.79	0.82	1030





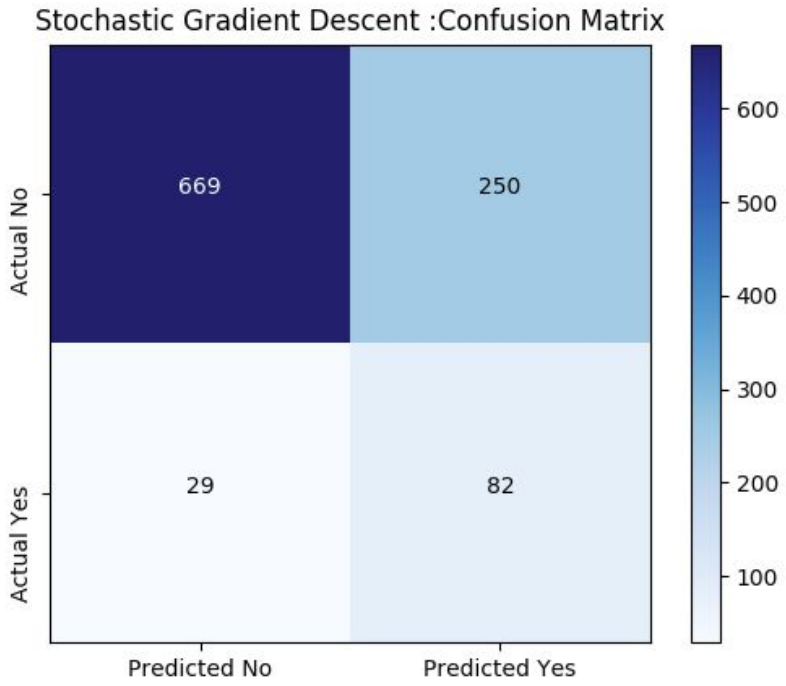
ROC Curve For Perceptron classifier

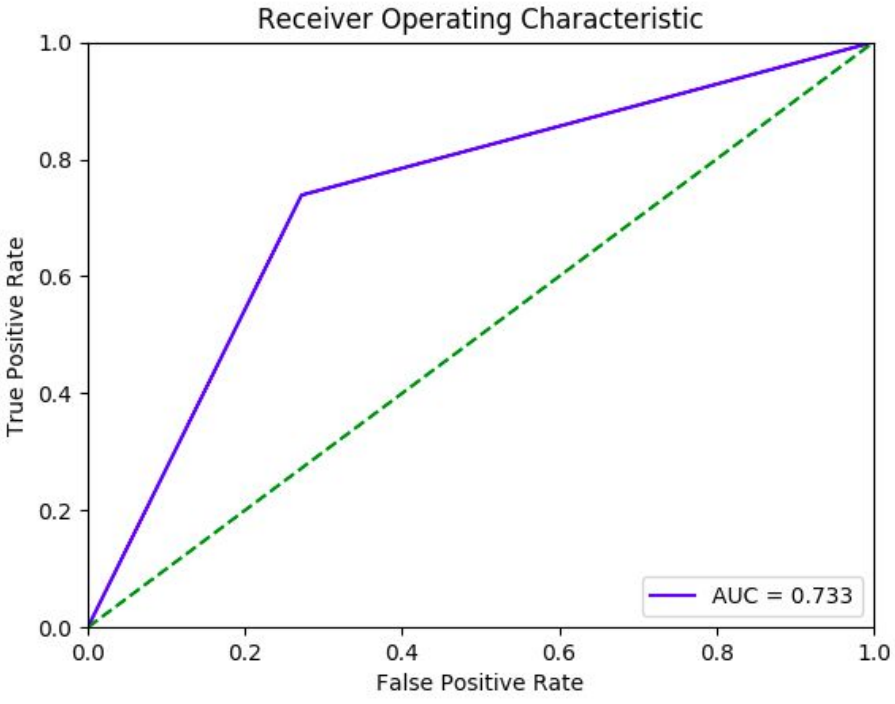
STOCHASTIC GRADIENT DESCENT CLASSIFIER

Stochastic Gradient Descent Train Accuracy: 0.749
Stochastic Gradient Descent Test Accuracy: 0.729
Stochastic Gradient Descent F1 Score: 0.778
Stochastic Gradient Descent AUC Score: 0.733

Classification Report scores:

	Precision	Recall	F1-score	Support
yes	0.96	0.73	0.83	919
no	0.25	0.74	0.37	111
avg/total	0.88	0.73	0.78	1030

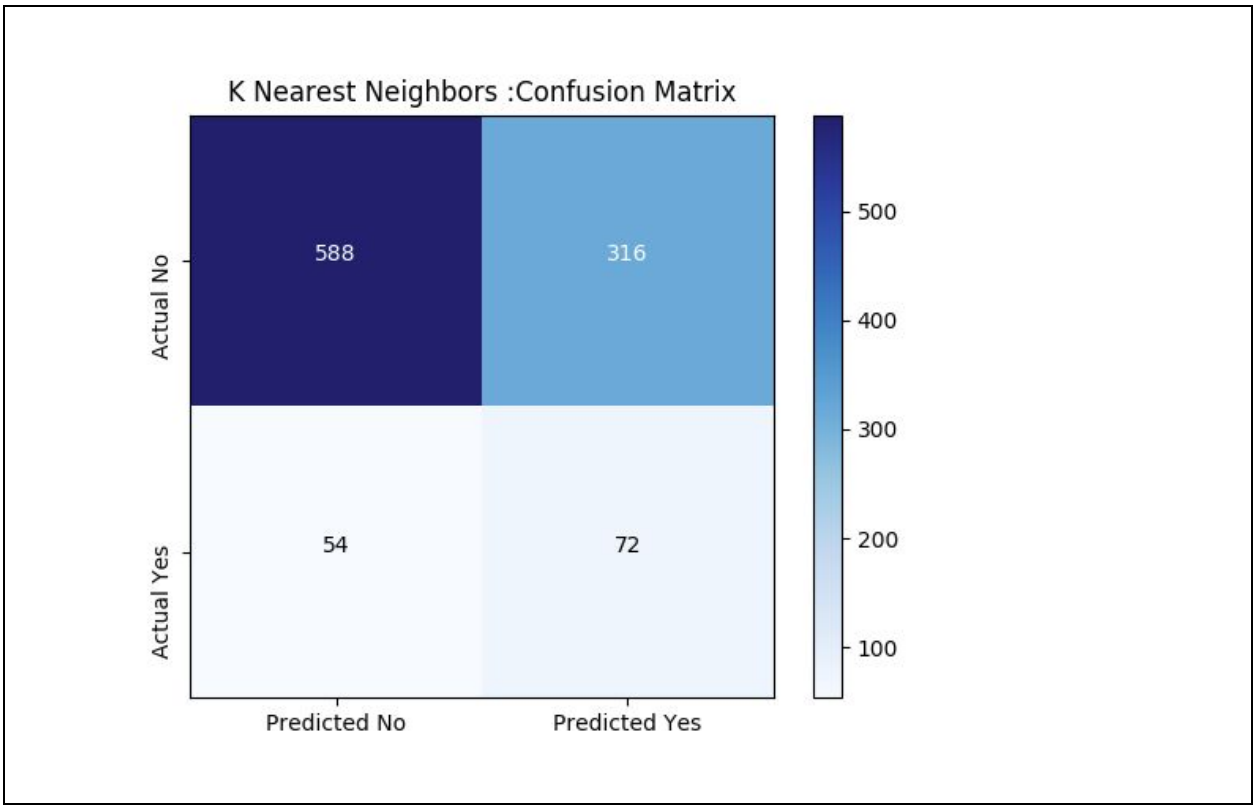

Stochastic Gradient Descent :Confusion Matrix
669
250
29
82
Actual No
Actual Yes
Predicted No
Predicted Yes
600
500
400
300
200
100

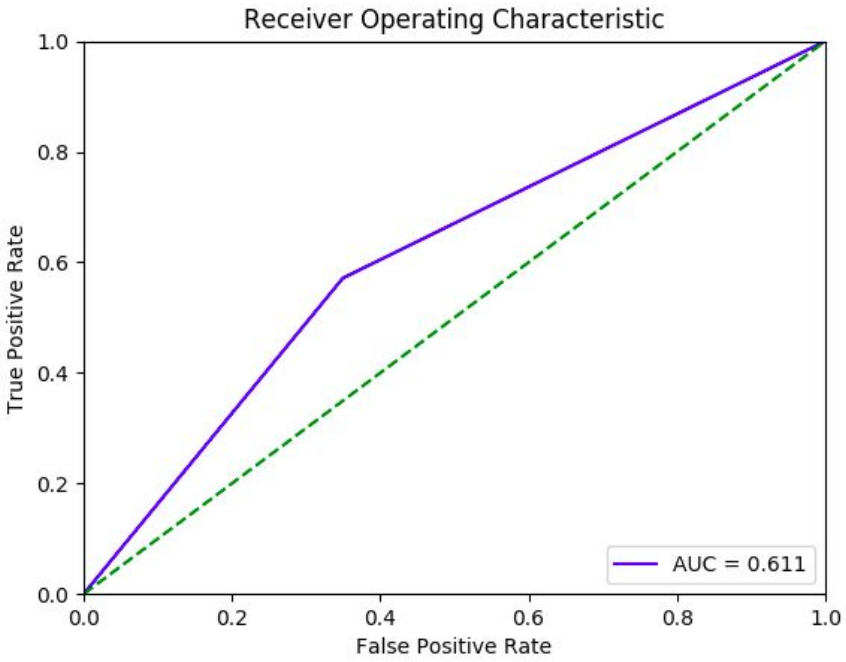

Receiver Operating Characteristic
True Positive Rate
False Positive Rate
AUC = 0.733
0.0
0.2
0.4
0.6
0.8
1.0

K NEAREST NEIGHBORS CLASSIFIER

Training data- Actual Label size: (4137,)
Training data- Predicted Label size: (4137,)
K Nearest Neighbors Train Accuracy: 1.000
K Nearest Neighbors Test Accuracy: 0.641
K Nearest Neighbors F1 Score: 0.702
K Nearest Neighbors AUC Score: 0.611
Classification Report scores:

	Precision	Recall	F1-score	Support
yes	0.92	0.65	0.76	904
no	0.19	0.57	0.28	126
avg/total	0.83	0.64	0.70	1030





DATASET SPLIT - The dataset was split into train and test datasets before performing the preprocessing and all the preprocessing, imputation and standardization is done after the splitting.
Sklearn.train_test_split is used to implement the splitting into train and test data. This division is performed using the default split ratio of 0.25 with 25% test data and 75% train data.

INTERPRETATION

Classifier	F1-Score	AUC Score
KNN	0.702	0.611
Stochastic	0.77	0.733
Perceptron	0.822	0.732
Naive Bayes	0.863	0.718
SVM	0.727	0.698
Random forest	0.865	0.75

From observing the above table and the metrics obtained for each classifier, I could conclude that the **Random Forest** performed the best given the label encoding, imputation and standardization performed by me.
It had the following scores:
Random Forest Train Accuracy: 1.000
Random Forest Test Accuracy: 0.850
Random Forest F1 Score: 0.865
Random Forest AUC Score: 0.757

We can observe that despite test accuracy scores being around 85% and the Train accuracy being 100%, the classifier performed better compared to the remaining classifiers as indicated by the F1 score which takes the precision and recall concepts into account while indicating the classification accuracy of a classifier. A high F1 score indicates that the system is performing good with higher number of true positives and true negatives compared to the rest. If we look at the confusion matrix of the Naive Bayes classifier, we see that the true negatives are very high compared to other categories(false positive, true positive, false negative)which is accurate since the number of 'no' s in the bank marketing dataset is also very high(approx 3600 out of 4200), where compared to the other categories.
The AUC score and F1 score should be around the same range for a classifier and a high score for both of them indicates a good classifier. There is some difference in the range between the AUC and F1 score, but they are very close and considerably high helping us judge them as the best classifiers.
The next best classifiers are the Naive Bayes and Perceptron classifiers with the scores as listed below:
Perceptron F1 Score: 0.822
Perceptron AUC Score: 0.732
Naive Bayes F1 Score: 0.863
Naive Bayes AUC Score: 0.718

REFERENCES:

1) https://en.wikipedia.org/wiki/Naive_Bayes_classifier
2) https://www.digitalocean.com/community/tutorials/how-to-build-a-machine-learning-classifier-in-python-with-scikit-learn
3) https://en.wikipedia.org/wiki/Perceptron
4) https://www.quora.com/What-is-an-intuitive-explanation-of-F-score
5) https://en.wikipedia.org/wiki/Support_vector_machine
6) https://en.wikipedia.org/wiki/Random_forest
7) http://contrib.scikit-learn.org/imbalanced-learn/stable/auto_examples/combine/plot_smote_enn.html#sphx-glr-auto-examples-combine-plot-smote-enn-py