05/01/2018
To develop a Pattern Recognition system that works on the real-world datasets

Pattern Recognition System

EE-559: Mathematical Pattern recognition (Final Project)

Harikrishna Prabhu
3333077042
hprabhu@usc.edu

AIM:

- To develop a Pattern Recognition system that works on the real-world datasets. The data set that is being worked upon is the “Bank Marketing Data Set”.
- Project ultimatum is *to predict if a client will hold a long-term deposit in the bank* after being approached by their marketing campaign.

ABSTRACT AND MOTIVATION:

Pattern Recognition is a branch of Machine Learning that deals with the recognition of patterns that are underlying for the given dataset. Pattern by itself is essentially an arrangement of elements and is characterized by the order in which it appears.

Pattern Recognition systems are trained on the “labeled” data i.e. the labels are known in prior to the model development, and hence are called as “Supervised Learning” models. Here, we will be dealing with a number of characteristics / features such as, Job, Education, outstanding loan, Time when approached by the bank, consumer confidence index and many such features based on which our model will be trained/ will learn; and using our trained model, we will have to predict if a client, in future, will or will not hold a long-term deposit in the bank.

GIVEN DATA SET:

Bank Marketing Data Set*:* This data set contains information of people approached by the bank through their marketing campaign. This dataset is a multivariate, 2 class data set. Let us look into the features of the given data set in detail. Please refer to the table given below.

TABLE 1

Feature	Data Type	Description	Possible values if any
Age	Numeric	Age of the client approached by the bank.	Any
Job	Categorical	Type of job, the client is doing.	Admin, Blue-Collar, Entrepreneur, Housemaid, Management, Retired, Self-employed, Services, Student, technician, Unemployed and Unknown.
Marital	Categorical	Marital status of the client.	Divorced, Single, Married and Unknown.
Education	Categorical	Educational status of the client.	Basic.4y, Basic.6y, Basic.9y, High School, Illiterate, Professional Course, University Degree and Unknown.
Default	Categorical	If the client has existing credit in default.	Yes, No and Unknown.
Housing	Categorical	If the client has a Housing Loan.	Yes, No and Unknown.
Loan	Categorical	If the Client has a Personal Loan.	Yes, No and Unknown.
Contact	Categorical	Medium of communication type.	Cellular and Telephone.
Month	Categorical	In which month, the client was contacted last.	Months of the year.
Day of Week	Categorical	On which day was the client contacted last	Days of the Week.
Campaign	Numerical	Number of times the client was contacted during this campaign.	Any.
pdays	Numerical	Number of days passed since the client last contacted in the previous campaign.	Any. Note: 999 means, was not contacted

Previous	Numerical	Number of times the client was contacted before the present campaign.	Any
Poutcome	Categorical	Outcome of the previous marketing campaign	Success, Failure and Non-Existent
Emp.var.rate	Numerical	Employment Variation Rate – Quarterly variation in the employment rate.	Any
Cons.price.idx	Numerical	Consumer Price Index- Monthly variation in the price payed by the consumers for retail goods.	Any
Cons.conf.idx	Numerical	Consumer Confidence Index – Degree of optimism with which the consumer buys the goods.	Any
Euribor3m	Numerical	Euribor 3-month rate-Euro Interbank offered rate- average interest rates offered by banks across Europe	Any
Nr.employed	Numerical	Number of employees	Any
y	Binary String	Class label	Yes or No

APPROACH:

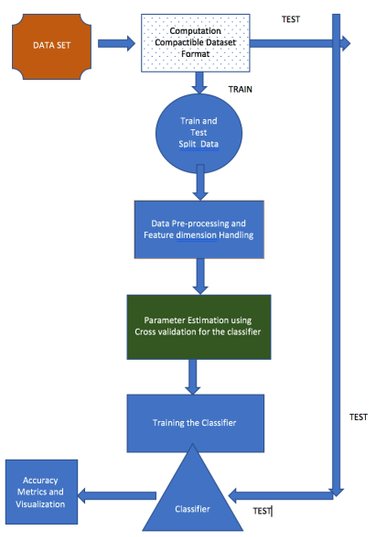

Figure 1: Flowchart of the approach for Model Development.

Each block of the flow chart is defined below.

DATA PREPROCESSING:

Data Preprocessing is a technique involving data that transforms raw data into more understandable form. In reality, the real-world data set contains more missing data and lacks consistency in its behavior. So, Data Preprocessing becomes an essential step to be followed before we develop the pattern recognition model. The following are the measures followed while preprocessing our Bank dataset.

(I) Handling missing data: It is quite obvious to find many missing data points in the real-world data set and here also we observe that the features, namely, Job, Education, Marital, Default, Housing and Loan have missing data points.

```
In [103]: runfile('/Users/rickerish_nah/Documents/trial/project_
Reloaded modules: handle_missing_data, data_preprocessing, class
Initiating the System.....
Please Wait...
Stage 1: Data Preprocessing...
Columns having unknown data in our Bank Dataset:
['job', 'marital', 'education', 'default', 'housing', 'loan']
```

Figure 2: Features in the Dataset that have missing data points.

Missing data can be of either of the following types,
(i) Missingness completely at random: Data is missing completely at random. The probability of the missing data is same irrespective of the feature type.
(ii) Missingness at random: Here, the miss is not completely at random. The probability of the missing data depends on its feature type.
(iii) Missingness due to unobserved data: Here the miss is not due to randomness but due to lack of observation of that particular feature.
(iv) Missingness that depends on the missing value itself: This type of missing data arises where, that particular feature depends on another feature which itself is missing at the first place.
In our case, the missing data are of types (ii) and especially (iii).
There are ways to handle missing data points, but handling it depends on the type of data in hand [numerical or non-numerical data types]. They are

(a) Ignore Observation: This is the simplest and most feasible type of handling missing data. It is followed where the number of missing data points is very much less compared to the size of the actual dataset. This can be applied to any type of data.

```
Size:
Original Train Data set:          3295
IGNORE_DATASET:  2465
```

Figure 3: Showing the number of data points in the original training dataset and the dataset after handling missing data by ignoring them.

[The Train and Test split will be discussed in the later parts of this report].

Let us look into the details of all the feature data in our dataset so that we can handle out data more precariously.

```
Decription of Non-Numeric data present in the Bank Data Set:
           job  marital           education default housing  loan   contact  \
count     3267     3288                3166    2632    3217  3217      3295
unique      11        3                   7       2       2     2         2
top     admin.  married  university.degree      no     yes    no  cellular
freq       816     2005                1020    2631    1756  2681      2114

       month day_of_week     poutcome     y
count   3295        3295         3295  3295
unique    10           5            3     2
top      may         mon  nonexistent    no
freq    1110         697         2790  2934
```

Figure 4: Description of Non-Numerical type of data. (Categorical Features)

```
Decription of Numerical data present in the Bank Data Set:
                age     campaign         pdays     previous  emp.var.rate  \
count  3295.000000  3295.000000  3295.000000  3295.000000   3295.000000
mean     40.184825     2.513505     0.237936     0.203035      0.068437
std      10.373130     2.541170     1.385310     0.558961      1.568603
min      18.000000     1.000000     0.000000     0.000000     -3.400000
25%      32.000000     1.000000     0.000000     0.000000     -1.800000
50%      38.000000     2.000000     0.000000     0.000000      1.100000
75%      47.000000     3.000000     0.000000     0.000000      1.400000
max      86.000000    35.000000    19.000000     6.000000      1.400000

       cons.price.idx  cons.conf.idx    euribor3m  nr.employed
count     3295.000000    3295.000000  3295.000000  3295.000000
mean        93.574734     -40.404825     3.608195  5165.577724
std          0.582684       4.632171     1.738085    74.286875
min         92.201000     -50.800000     0.635000  4963.600000
25%         93.075000     -42.700000     1.334000  5099.100000
50%         93.444000     -41.800000     4.857000  5191.000000
75%         93.994000     -36.400000     4.961000  5228.100000
max         94.767000     -26.900000     5.045000  5228.100000
```

Figure 5: Description of Numerical type of data.

(b) Replacing with Mode or Mean: It is also a simple way to handle missing data. Handling by replacing the missing value using mean impute, this works out well only for numerical data types. For non-numerical data such as categorical data, we use mode impute. i.e. we replace the missing data point feature with the most frequently occurring value for that particular feature. In our case, as all the missing data types are 'object' or 'categorical' data type, we have implement mode imputing.

```
Size:
Original Train Data set:          3295
MODE IMPUTED_DATASET:    3295
```

Figure 6: Showing the number of data points in the original training dataset and the dataset after handling missing data using mode impute.

(c) Handling missing data using Predictive model/algorithms: This method of handling missing data is tedious and time consuming. In this method, we split the dataset into two subsets, such that one contains data points with all the features known (train_temp) and another with data points having unknown feature points (test_temp). One the data set is segregated, we will use the train_temp dataset to train our model and predict the missing feature points using the trained classifier on the test_temp. Here, we use the feature (columns in dataframe – pandas) in the train_temp as label and predict the respective labels for test_temp. Likewise for all other features.

```
Size:
Original Train Data set:          3295
PREDICTIVE MODEL BASED IMPUTATION_DATASET:       3295
```

Figure 7: Showing the number of data points in the original training dataset and the dataset after handling missing data using predictive mode based imputation.

Here, Support Vector Machine (SVM) based prediction of unknown values is used.

NOTE: It must be taken into consideration that, imputation is to be done on class-wise dataset only, so that the behavior of the class is preserved.

The above three figures give us the intuition of dataset dimension change due to handling the missing data. The MODE and Predictive model imputation preserves the data set dimension post data handling. Whereas the ignore data set has reduced dimension.

(II) Handling Data Imbalance:

Data Imbalance can be found in almost all of the real-world data set that in unprocessed. This imbalance causes bias or tendency to move towards the dominant class in many predictive and machine learning algorithms. Thereby taking measures to reduce imbalance in data improves the classification accuracy of the training models. Data imbalance can be handled by two ways.

(a) Handling imbalance through under-sampling: Suppose we have a data set that contains 100 data points in C1 and 400 data points in C2. The as a result of under sampling, we will have a dataset that has 100 data points in both C1 and C2.

(b) Handling imbalance through over-sampling: Assuming the same scenario as mentioned above, through over-sampling we will have 400 data points in both C1 and C2.

Under sampling is done by reducing the number of data points in the dominant class type by choosing the 'n' data points, either at random or selectin the top 'n'. Likewise, for over-sampling, the non-dominant class is fit into a model and is resampled to get the final 'n' data points.

This data imbalance is handled using the python function "**SMOTE"** from the library *imblearn.over_sampling*.

(III) Data Scaling: It is always advisable to perform data scaling before developing our pattern recognition system. It is quite intuitive that scaling can be done only for numerical datatypes. Data scaling can be done in the following ways,

(a) Data Normalizing: Normalizing refers to the process of converting the numerical data following different scale to unit scale. i.e. value transforms to fall in (0,1) range. One should make a note that it is fine to normalize most of the types of data, but not. Example, it is fine to normalize currency conversion rates, but it is not fine to normalize data that represent coefficients or proportions. In our dataset 'Bank', we do not have any coefficients, so we are normalizing all of the data. This is done using MinMaxScaler of the sklearn library.

(b) Data standardization: It refers to the process of converting our data to follow zero mean and unit variance. But before standardizing, we must make sure that the data was already following Gaussian distribution. If not, it is not advisable to forcefully standardize as it would bias the original characteristics of the data. We can use StandardScaler under sklearn library to standardize our data.

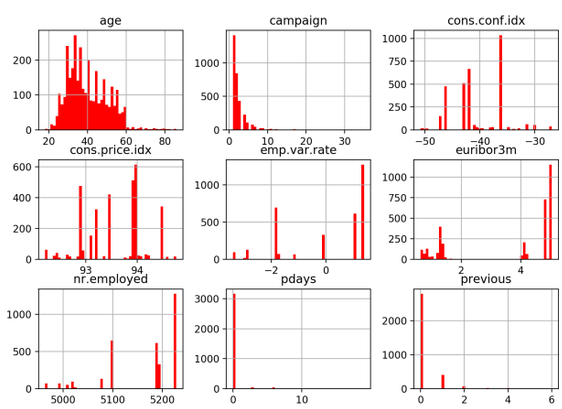

Figure 8: Histogram for numerical data type.

[age, campaign, cons.conf.idx, cons.price,idx, emp.var.rate, euribor3m, nr.employed, pdays, previous**]**

It can be seen from the above figure that none of the feature our data set approximately follow Gaussian distribution other than 'age'. And 'age' is not a very high quantitative

numerical aspect like 'nr.employed' that would bias the discriminant power of all other features. Hence it is good to go.

Data scaling (Normalizing and Standardizing) in python is done using the function **"MinMaxScaler"** and **"StandardScaler"** using the library *sklearn.preprocessing*.

(III) One-Hot Encoding and Label Encoding: These Encoding techniques are very useful to convert non-numerical data types into numerical (decimal type or binary type). Many a time, we find it difficult to handle non-numerical data, as it is difficult to do computations on it or using it. Hence it is a general mean to convert these categorical data types to numerical data type.

Label Encoding: In label encoding, each unique label is identified and is allocated a numerical value and this non-numerical to numerical conversion is remembered by means of using a library function. An example for label encoding is shown below.

Sample	Category	Numerical
1	Human	1
2	Human	1
3	Penguin	2
4	Octopus	3
5	Alien	4
6	Octopus	3
7	Alien	4

Figure 9: Label encoding.

But the problem with label encoding is that the data that is label encoded must be used as labels and not as feature values. Plus, mathematical operations on label encoded data gives faulty result. Say we need to find the mean for the above categorical data, it is Penguin, which makes no sense. There are classifiers that can work on categorical data, example Random Forest Classifier. Yet to make sure the confidence factor of the encoding process, One-Hot Encoding is done.

One-Hot Encoding: It is a way of label encoding which gives proper structure to the encoding process, so that the training models can do better job in prediction. Here the labels are not ordinated, instead, they are binarized. Let us see the figure below to see the difference between One-Hot encoding and Label Encoding.

Sample	Human	Penguin	Octopus	Alien
1	1	0	0	0
2	1	0	0	0
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	0	0	1	0
7	0	0	0	1

Figure 10: Same categorical data as shown above is one-Hot Encoded here.

Here, even when doing computation on these data, it will provide stable result. This is because, L2 norm will be applied while making computations over the former normal distance calculator.

Given below is the dimension of our handled data sets after applying One-Hot Encoding.

```
Data sizes:
IGNORE_DATASET: (2465, 51)        LABEL: (2465, 1)
MODE_DATASET: (3295, 51)          LABEL: (3295, 1)
SVM_DATASET: (3295, 51)           LABEL: (3295, 1)
TEST_DATASET: (625, 51)           LABEL: (625, 1)
Ignore Dataset:
```

Figure 11: Data dimension after encoding

Note: Label encoding is used in the program for predicting unknown feature points and One-Hot Encoding is used on the data before the model is trained or used for classification.

Label Encoding and One Hot Encoding in python is done using the functions "**pandas.get_dummies(Data)"** and **"LabelEncoder"** using the libraries *pandas* and *sklearn.preprocessing.* While performing Label Encoding and Decoding, we discussed the need for a library function, and that in python is implemented using the function **"defaultdict"** from the library *collections*.

FEATURE SPACE HANDLING:

Here we will deal with the aspects on ***feature selection*** and ***feature dimensionality*** reduction. Before we go in depth into handling the feature space, let us look at the feature details.

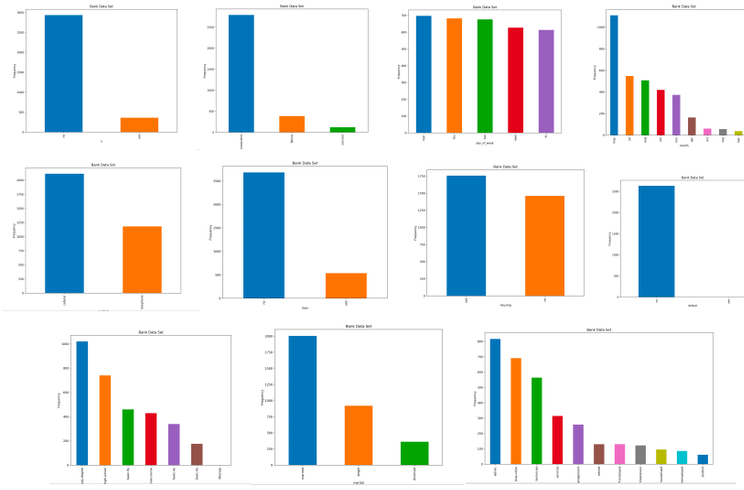

Figure 12: Histogram for Categorical data type.

[y, poutcome, day of week, month of year, contact, loan, housing, default, education, marital, job**]**

For our given dataset, we have 19 features and a label to predict if the client will hold a deposit at their bank or not. It is evident from the plots that there are many features that have strongly dominant feature values and each feature is of different scale. Hence, we do data normalization to bring all the feature values to (0,1).

Let us have a look at the correlation of all these features.

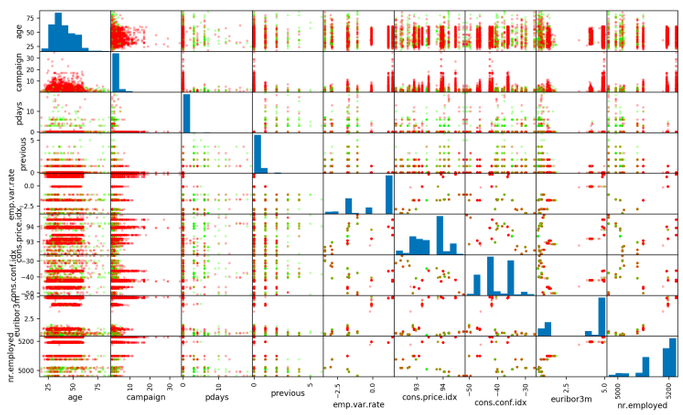

Figure 13: Correlation of features.

Here each feature is plotted against every other feature of the data set. The diagonal blocks represent the histogram of each feature and the off-diagonal ones represent the correlation plots of every pair-wise feature set. It can be observed from this correlation plot that these features aren't discriminative in nature, i.e. they are correlated. There are few features that are highly correlated. We can use this high-correlation rate as a factor to remove that particular feature, but those features are very highly correlated with some and lightly correlated with the others so it throws a manual move to keep it or drop it. Example, Age and Campaign are very highly correlated and pdays and previous are lightly correlated.

A similar and simple correlation figure is shown below,

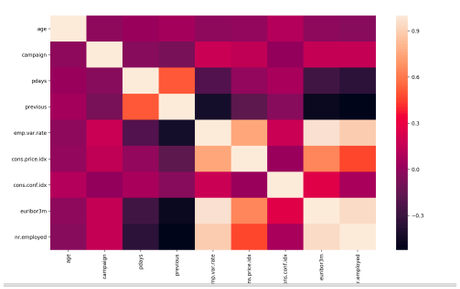

Figure 14: Correlation color maps.

Highly correlated features are shown in light color gradients and low correlation is shown by darker shades. From these correlation plots we can say that feature selection is quite difficult and becomes of trivial use. While handling missing data in our program, we come across a

situation where the feature "default" has 2632 known values of 4119 total input data, and out of which 2631 feature points are 'no' and only one is 'yes'. So even during predicting, unknown values for 'default' are predicted to 'no' values. Therefore, this feature can be termed as having zero or minimum discriminating power. Hence is it dropped from the data set. Apart from dropping this feature, we can reduce data dimensions by removing the outliers in the data set. Again, during experimenting with the program, we found out that the feature 'month December' (after one hot encoding) is not helping much as it occurs scarcely.

To reduce feature dimension, we can do the following.

(a) Principal Component Analysis: It is a statistical procedure that uses orthogonal transformation to convert correlated features to linearly un-correlated features. It is an apt methodology for dimension reduction. This PCA arranges the orthogonal space on the order of decreasing variance with the feature having most variance at the top. But it is to be made sure that the data dimension reduced using PCA are normalized as PCA is very sensitive to the data value's scale.

In python, PCA is implemented using the function **"PCA"** from the library *sklearn.decomposition*.

(b) K best feature selection: It is a feature selection algorithm rather than feature reduction. Here each feature's univariate characteristic is computed and is differenced with that of others. Those features producing largest sum of the difference is considered to the best feature pair and in turn the features are arranged accordingly. Then to top 'n' features are chosen while reducing the feature dimensions.

In python, K best feature selection is implemented using the function **"SelectKBest"** from the library *sklearn.feature_selection*.

When comparing these two methods for feature dimension, K best feature selection works well as it preserves the feature label implicitly and also provides better accuracy results with feature dimension reduction. The observation of the choice of feature dimension reduction will be discussed in the experimental observation section of the report.

CLASSIFIERS AND PARAMETER ESTIMATION:

Down this section we will discuss about the types of classifiers used in our project and how we estimate the classifier's best parameter setting. Let us see in detail the classifiers used.

CLASSIFIERS USED:

(I) Perceptron: Perceptron is a linear classifier that is used in supervised learning, i.e. Pattern recognition systems. It functions to decide whether the input (may be a variable or a vector of numbers) belongs to a specific class or not. (i.e. 0 or 1). It is a function that maps a real value variable to an output value that is binary in nature, as shown below.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

here, x = data point, w is the weight and b is the bias value.

The weight update on perceptron is done using the expression given below.

$$\underline{w}(i+1) = \underline{w}(i) + \eta(i) z_n \underline{x}_n \quad \text{if } \underline{w}(i)^T z_n \underline{x}_n \le 0$$
$$\underline{w}(i+1) = \underline{w}(i) \quad \text{if } \underline{w}(i)^T z_n \underline{x}_n > 0$$

here z = -1 for misclassified data points and +1 for properly classified.

In python it is used from the library *sklearn.linear_model*, as **"from sklearn.linear_model import Perceptron"**. The parameters used in setting this classifier is *class_weight='balanced', shuffle=True*. These are set manually.

(II) Support Vector Machine (SVM):

SVM is a discriminative classifier that is defined by a separating hyperplane. In 2-D this separating hyperplane is a line. A good separation is achieved by this classifier by drawing the hyperplane that has the largest possible distance to the nearest data point possible (called as the margin of the classifier). The generalization error produced by the classifier will be lower for larger margins. It is used from the library *sklearn.svm* as **"from sklearn.svm import SVC"**. The parameters used in settings this classifier is *C, gamma, probability = True and kernel*.

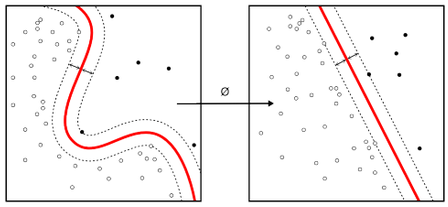

Figure 15: SVM

The fact about SVM is that it can be used as a linear as well as non-linear classifier. This can be done by changing the parameter kernel of the classifier.
(i) kernel = 'linear' for SVM to function as a Linear classifier.
(ii) kernel = 'rbf' for SVM to function as a Non-linear classifier. [rbf = radial basis funciton]
(iii) kernel = 'poly' for SVM to function where the classification boundary is of polynomial nature.
Note: SVM is an example of Parametric classifier.

(III) K Nearest Neighbors Classifier (KNN classifier):

KNN classifier is a non-parametric classifier. The output of this classifier is the class label. This is done by computing the majority votes of its neighbors and assigning it to a class that is more prevalent amongst its neighbors. Since this classifiers scope is applied locally, it is also called as lazy learning classifier. Its parameters are "number of nearest neighbors 'n'" and "norm for distance calculation 'p'", so these nearest neighbors are found by computing the distances and those 'n' data points with the least distance from the current data point is considered as its neighbors. And p can be 1 or 2 depending on the type of norm to be used. i.e Manhattan or Euclidean norm.

In python it is used from the library *sklearn.neighbors* as **"from sklearn.neighbors import KNeighborsClassifier"**.

Both SVM and KNN classifiers can be used for regression and classification process.

(IV) Decision Tree classifier:

Decision trees is one of the predictive modelling approaches used for classification processes. Decision tree is a tree like graph model. It is a map of possible outcomes of a series of related choices. This tree starts from the root (node) at the top and ends at the leaf (node) at the bottom. Each node functions to direct the decision control to its left or right node in the bottom (offspring) layer [binary in nature]. Say for a given decision rule at a node, if *True* it'll direct the flow to its right offspring and if *False*, it'll direct to the left or vice versa. This way the decision flow reaches to the leaf nodes and the probabilistic weights are assigned to the leaf nodes. For a classification problem, it functions based on the "Majority Voting" and for regression type, it uses "Mean Averaging".

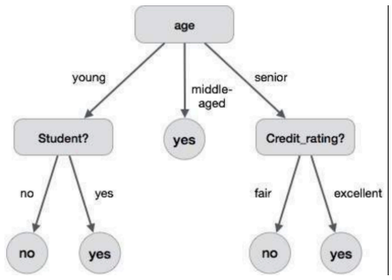

Figure 16: Example for decision tree.

In python it is used from the library *sklearn.tree* as **"from sklearn.tree import DecisionTreeClassifier"**. In our program, we are using the DecisionTree classifier with default parameter settings.

(V) Random Forests Classifier (RF classifier): RF classifiers are an ensemble learning method for classification, (ensemble = multiple layers are trained to solve the same problem and combines to develop the learning model). They contain a multitude of decision trees and operate by taking the mode of individual decision trees for classification and mean of individual decision trees.

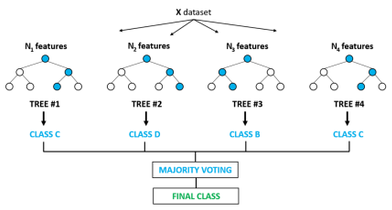

Figure 17: Random Forest

In python it is used from the library *sklearn.ensemble* as **"from sklearn.ensemble import RandomForestClassifier".** The parameters that is used in setting the classifier is "number of estimators".

(VI) Multi-Layer Perceptron (MLP) Classifier: MLP is a feed forward artificial neural network. We had discussed earlier about the Perceptron classifier. Here it is such that, that the MLP classifier is a stack of perceptron layers. In general, this artificial neural network has a minimum of 3 perceptron layers.

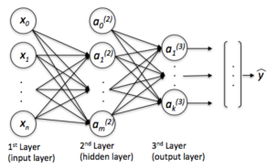

Figure 18: MLP structure.

In python, this classifier is used from *sklearn.neural_network* as **"from sklearn.neural_network import MLPClassifier"**. The parameters that is set for this classifier are "Hidden layer size" and "Max iteration".

(VII) Naïve Bayes Classifier (NB) classifier: NB classifier is one of the simplest classifier used to predict class labels. It belongs to the family of Probabilistic classifiers and more importantly follows the Bayes theorem. The Bayes theorem is expressed as,

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

This NB classifier assumes that the values in one feature is completely independent of the other features. i.e. uncorrelated or least correlation. In many applications, if at all parameter estimation is done, Maximum Likelihood estimation or priors are used. And also, it assumes that the priors are Gaussian distributed. In python it used from *sklearn.naive_bayes* as **"from sklearn.naive_bayes import GaussianNB"**.

We have indeed discussed a lot about the classifiers we have used and its variety and uniqueness with respect to the other classifier put at play. In almost all of the classifiers, we have mentioned about the parameter setting. These parameters are estimated using a technique called Cross-Validation.

CROSS VALIDATION FOR PARAMETER ESTIMATION: Cross validation is a technique used to evaluate the parameters for the given classifier model used. In simple words, it is the process of randomly partitioning the original data set into 'k' sub samples (k folds cross validation) and keeping one of the subsamples as the validation set and the rest others appended together as the training data set. Iteratively this is repeated for 'k' times with different subsample as the validation set. The results over this 'k' iterations is averaged to produce a single optimized value. But in python this k fold cross validation is function is used from *sklearn.model_selection* as "**sklearn.model_selection.KFold** ". But there is flaw in this technique, the subsamples produced are not stratified, i.e. they aren't equally split based on class population so there are chances that the validation set will contain only data belonging to one class, which will give improper results. Hence we use the function **"sklearn.model_selection.StratifiedKFold "** from the library sklearn.model_selection.

The parameters that we have to keep in mind while performing cross validation are,
(i) n_splits = 'k'
(ii) shuffle = True more importantly.
So now we know how cross validation is done. We use this cross-validation technique by fitting different values for each of the different parameters iteratively and compute the single mean estimate. i.e. for each parameter setting (different parameters are set into the classifier over a separate iteration for it), using cross-validation, mean classification accuracy is computed with the validation set and that parameter(s) is set for the formal model training that gives the maximum of the validation classification accuracy. This parameter estimation using stratified K fold is done to estimate the best fitting parameter(s) for all the classifiers used in our program as mentioned above. The best set of parameters estimated using cross validation is tabulated in the experimental results section below.

ACCURACY METRICS:

Accuracy rate is used as a medium to rate the performance of the developed/trained classifier. There are many accuracy metrics that can be used to evaluate the performance statistics for the given classifier and we also have to keep in mind that our dataset is imbalanced in nature. Hence accordingly we are using only the following accuracy metrics that is described below.

(I) F measure:

In any problem statement where binary classification is to be implemented, F measure is used as the measure of the classifier's performance. It is the measure of a test's accuracy that is defined as the weighted harmonic mean of precision and recall of the given test under consideration. Precision is defined as the fraction of retrieved instances that are relevant and Recall is defined as the fraction of relevant instances that are retrieved. A pictorial representation of Precision and Recall is shown below.

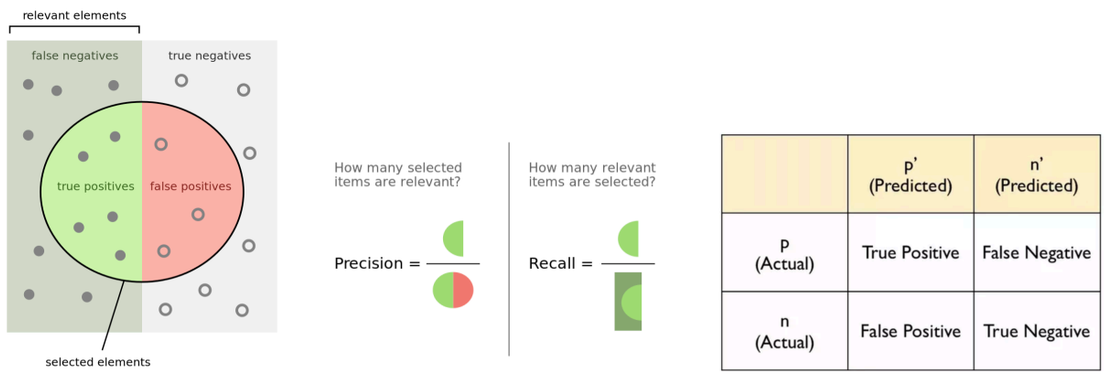

Figure 19: Computing precision and recall for F-measure.

Then F-measure is computed using the formula,

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

In python we compute the F measure using the functions from *sklearn.metrics* **classification_report** and or **precision_recall_fscore_support**.
In precision_recall_fscore_support, we will have to include a parameter called 'Average' as Weighted, whereas for classification_report, it is prevalence-weighted macro-average across classes.

(II) Receiver Operating Characteristic (ROC) and Area Under the Curve (AUC):

Statistically, a ROC curve is a graphical representation that shows the ability of a binary classifier as its discriminantion threshold is varied. This ROC curve is created using the true positive rate (TPR) against the false positive rate (FPR) at various discrimination threshold settings. The TPR is known as the sensitivity or probability of detection and FPR is known as the fall-out or probability of false alarm. So, if we know the probability distribution for both detection and false alarm, we can plot the ROC curve using its cumulative distribution function.

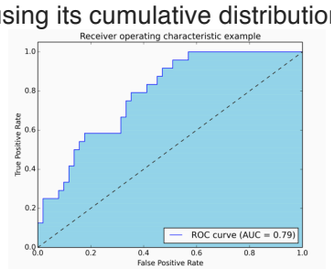

Figure 20: ROC curve and AUC

ROC is just a curve; the actual accuracy is deduced by computing the area under the ROC curve. So, in general this ROC AUC metric gives us a single numeric value proportional to the performance accuracy of our trained classifier by means of calculating the area under the curve. In python, we can implement the ROC AUC measure using the functions **"roc_auc_score"** and "**roc_curve"** from the library sklearn.metrics.
The roc_auc_score will directly give the AUC value and roc_curve is used in extracting the TPR, FPR and the discrimination threshold to plot the ROC curve and compute the AUC value separately.

DATA SET USAGE, TRAINING AND CLASSIFICATION:

We have the given data set "Bank Marketing Data set". Before even we use the data into our program it is important to split the given data into Training and Test datasets and also one has to keep in mind that this separation is stratified. Hence in python we use the function **"train_test_split"** from the library *sklearn.cross_validation*. I have used the following code to split the given data into test and train data sets.

ip_data,ip_test_data = train_test_split(inp_data, test_size=0.20, random_state=42, stratify=inp_data.y)

Where the "test_size = 0.20" indicates that the given data is separated into 80% as training dataset and the remaining 20% as the test data set. And also, another parameter "stratify" is set to the data labels so that the given data can be equi-populatedly be split.
This Test data is left untouched and only the Training dataset is proceeded with the data preprocessing and feature space handling. Test data comes into action only while testing the trained classifier.

TRAINING AND CLASSIFICATION: From the parameters estimated for each classifier using stratified K folds, we fit the training data and the training label to train that particular classifier. Once the classifier is trained, we pass the test dataset and predict the class labels. And using the accuracy metrics we had discussed, we will calculate the performance statistic for each of the classifier used. The classifier producing best result is shown in the Experimental section of this report.

EXPERIMENTAL RESULTS:

The parameters estimated for each of the classifier (best parameter setting) is shown the table given below.

P – Perceptron : KNN – K Nearest Neighbours : SVML – SVM Linear : SVMR – SVM RBF :
SVMP – SVM Poly : DT – Decision Tree : RF – Random Forest : MLP- Multi-Layer Perceptron
NB – Naïve Bayes

TABLE 2: PARAMETER ESTIMATION USING CROSS-VALIDATION

	Classifier (Across)	**KNN**	**SVML**	**SVMR**	**SVMP**	**RF**	**MLP**
Parameter (Down)							
n_neighbors		6					
p		1					
C			13.73	0.021544	215		
gamma				0.004641	4.356		
degree					5		
n_estimators						14	
hidden_layer_sizes							160
max_iter							200

The Table given below shows the F Measure and Area Under the Curve values for each of the handled dataset using different classifiers.

TABLE 3: F Measure and AUC score

Data set (Down)	**Classifier (Across)**	**P**	**KNN**	**SVML**	**SVMR**	**SVMP**	**DT**	**RF**	**MLP**	**NB**
Mode Imputed DataSet	F Measure	0.79	0.85	0.88	0.83	0.8	0.84	0.87	0.86	0.87
	Area Under the Curve	0.68	0.65	0.69	0.73	0.62	0.59	0.75	0.7	0.76
Ignored DataSet	F Measure	0.84	0.85	0.88	0.83	0.81	0.84	o.87	0.87	0.87
	Area Under the Curve	0.61	0.68	0.69	0.68	0.61	0.6	0.77	0.69	0.76
SVM Imputed DataSet	F Measure	0.02	0.65	0.84	0.02	0.02	0.83	0.64	0.02	0.83
	Area Under the Curve	0.5	0.62	0.53	0.5	0.5	0.57	0.54	0.5	0.5

```
- Perceptron -
F Measure Report:
             precision    recall  f1-score   support

          0       0.91      0.90      0.91       554
          1       0.30      0.34      0.32        71

avg / total       0.84      0.83      0.84       625

roc_auc:         0.6175700411857425
```

```
Using-KNN -
F measure Report:
             precision    recall  f1-score   support

          0       0.89      0.99      0.94       554
          1       0.56      0.07      0.12        71

avg / total       0.85      0.89      0.85       625

ROC_Curve:       0.6856536329892714
```

Figure 21: Showing Precision, Recall, F Measure and AUC value

```
Using-SVM Linear -
F measure Report:
             precision    recall  f1-score   support

          0       0.91      0.99      0.95       554
          1       0.75      0.21      0.33        71

avg / total       0.89      0.90      0.88       625

ROC_Curve:       0.6903442314537042
```

```
Using-SVM Poly -
F measure Report:
             precision    recall  f1-score   support

          0       0.91      0.85      0.88       554
          1       0.22      0.32      0.26        71

avg / total       0.83      0.79      0.81       625

ROC_Curve:       0.6147353434687547
```

Figure 22: Showing Precision, Recall, F Measure and AUC value

```
Using-Decision Tree -
F measure Report:
             precision    recall  f1-score   support

          0       0.91      0.91      0.91       554
          1       0.29      0.30      0.29        71

avg / total       0.84      0.84      0.84       625

ROC_Curve:       0.601858443077236
```

```
Using-Random Forest -
F measure Report:
             precision    recall  f1-score   support

          0       0.91      0.98      0.94       554
          1       0.58      0.21      0.31        71

avg / total       0.87      0.89      0.87       625

ROC_Curve:       0.7752071998779682
```

Figure 23: Showing Precision, Recall, F Measure and AUC value

```
Using-SVM gausian -
F measure Report:
             precision    recall  f1-score   support

          0       0.89      1.00      0.94       554
          1       0.00      0.00      0.00        71

avg / total       0.79      0.89      0.83       625

ROC_Curve:       0.6895561092184879
```

```
Using-Neural Net:MLP -
F measure Report:
             precision    recall  f1-score   support

          0       0.91      0.98      0.94       554
          1       0.59      0.24      0.34        71

avg / total       0.87      0.89      0.87       625

ROC_Curve:       0.6926577515635327
```

Figure 24: Showing Precision, Recall, F Measure and AUC value

```
Using-Naive Bayes -
F measure Report:
             precision    recall  f1-score   support

          0       0.93      0.92      0.93       554
          1       0.44      0.49      0.46        71

avg / total       0.88      0.87      0.87       625

ROC_Curve:       0.7622413179437637
```

Figure 25: Showing Precision, Recall, F Measure and AUC value

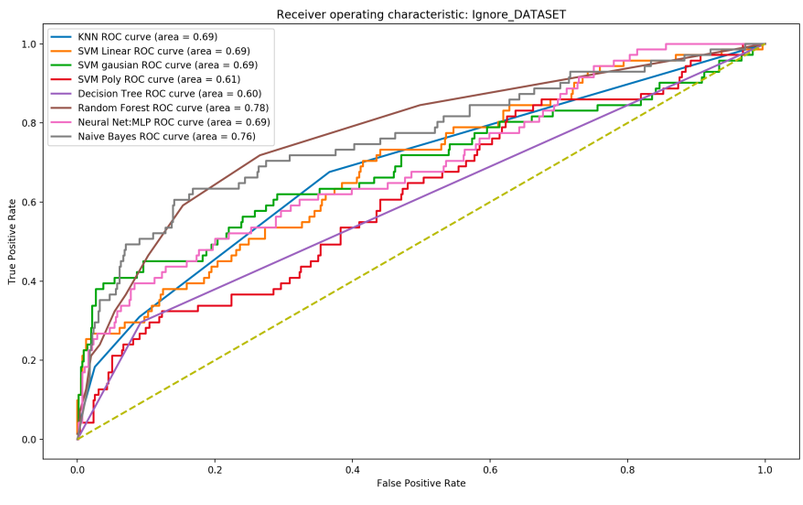

FIGURE 26: AUC scores for all the classifiers used in Ignore_DataSet

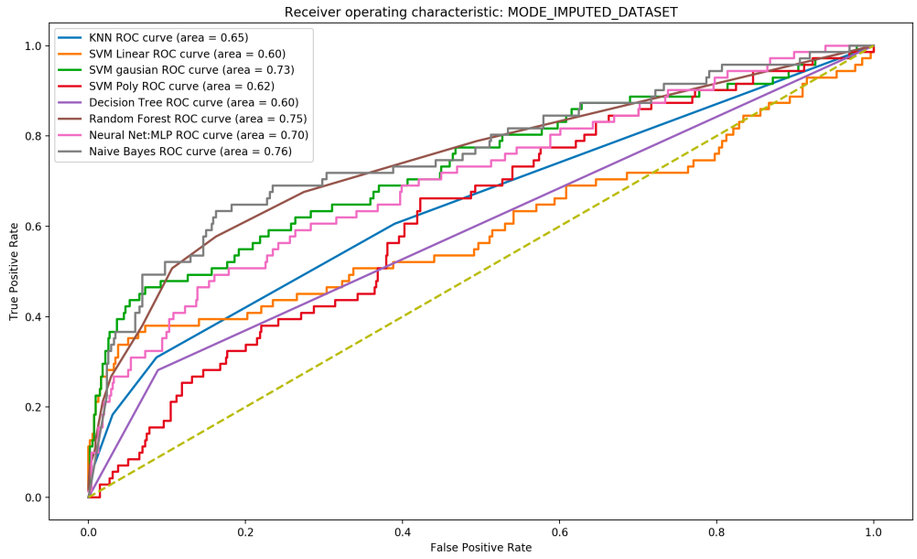

FIGURE 27: AUC scores for all the classifiers used in Mode_Imputed_DataSet

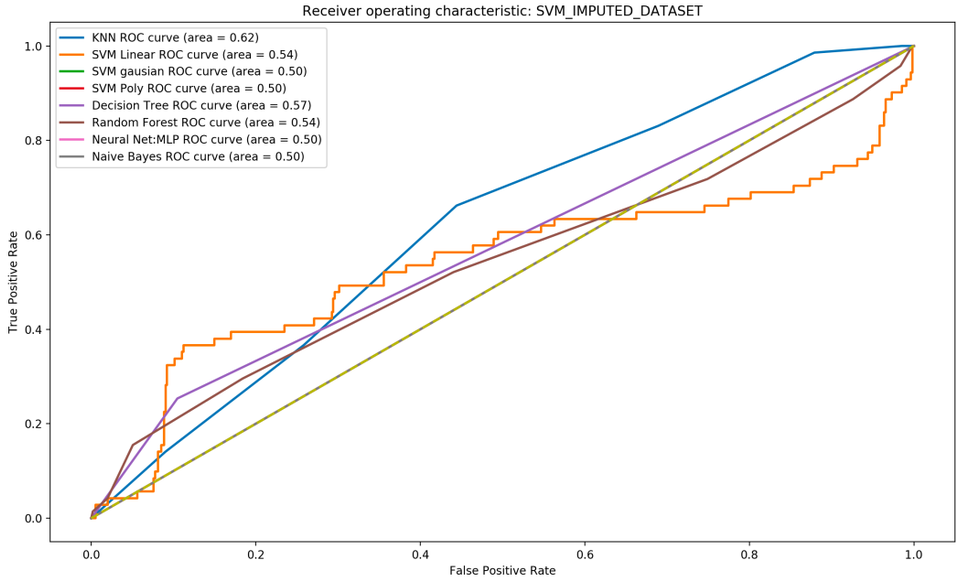

FIGURE 28: AUC scores for all the classifiers used in SVM_Imputed_DataSet

After Dimension reduction of the Features, we computed the AUC value for each of the dimension reduced data set.

```
After Feature reduction:
For  1  reduced features, The AUC is:  0.7702623684344333
For  2  reduced features, The AUC is:  0.7759698988152743
For  3  reduced features, The AUC is:  0.6851070320842019
For  4  reduced features, The AUC is:  0.7112421823358928
For  5  reduced features, The AUC is:  0.6784842629785935
For  6  reduced features, The AUC is:  0.6813698072914018
For  7  reduced features, The AUC is:  0.7005898205115167
For  8  reduced features, The AUC is:  0.6895942441653532
For  9  reduced features, The AUC is:  0.6978186810393043
For  10  reduced features, The AUC is:  0.710784562973509
For  11  reduced features, The AUC is:  0.7251614379417298
For  12  reduced features, The AUC is:  0.7143056897340723
For  13  reduced features, The AUC is:  0.7351146590735751
```

```
For  14  reduced features, The AUC is:  0.6945644989067981
For  15  reduced features, The AUC is:  0.7413433670615752
For  16  reduced features, The AUC is:  0.7257334621447094
For  17  reduced features, The AUC is:  0.7413433670615752
For  18  reduced features, The AUC is:  0.7553516042100982
For  19  reduced features, The AUC is:  0.7441145065337875
```

Figure 29: AUC values for dimension reduced data set.

Table 4: Showing AUC value for over sampled data.

		P	KNN	SVML	SVMR	SVMP	DT	RF	MLP	NB
OverSampled Data	F Measure	0.60	0.62	0.76	0.74	0.64	0.84	0.85	0.67	0.73
	Area Under the Curve	0.63	0.70	0.82	0.78	0.73	0.83	0.94	0.82	0.78
UnderSampled Data	F Measure	0.69	0.68	0.73	0.72	0.61	0.61	0.74	0.69	0.68
	Area Under the Curve	0.69	0.74	0.77	0.5	0.63	0.61	0.79	0.78	0.78

```
Using-Random Forest -
F measure Report:
              precision    recall  f1-score   support

          0       0.79      0.96      0.87       554
          1       0.95      0.75      0.84       554

avg / total       0.87      0.85      0.85      1108

ROC_Curve:       0.9418619426813852
```

Figure 30: Performance Statistics for over-sampling data type.

```
Before Balancing the Data:
(3295, 51) (625, 51)
After Balancing the Data Over-Sampling:
(5868, 51) (1108, 51)
After Balancing the Data Under-Sampling:
(722, 51) (142, 51)
Comparing different models:
```

Figure 31: Dimension comparison between pre and post over-sampling and under-sampling.

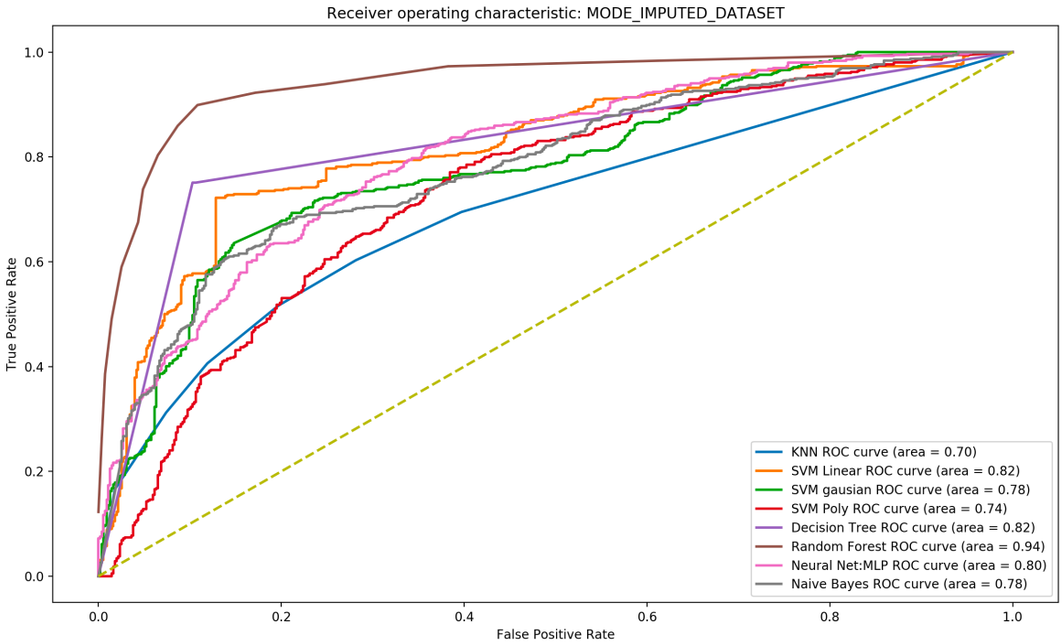

Figure 32: AUC score for over-sampled reduced data set.

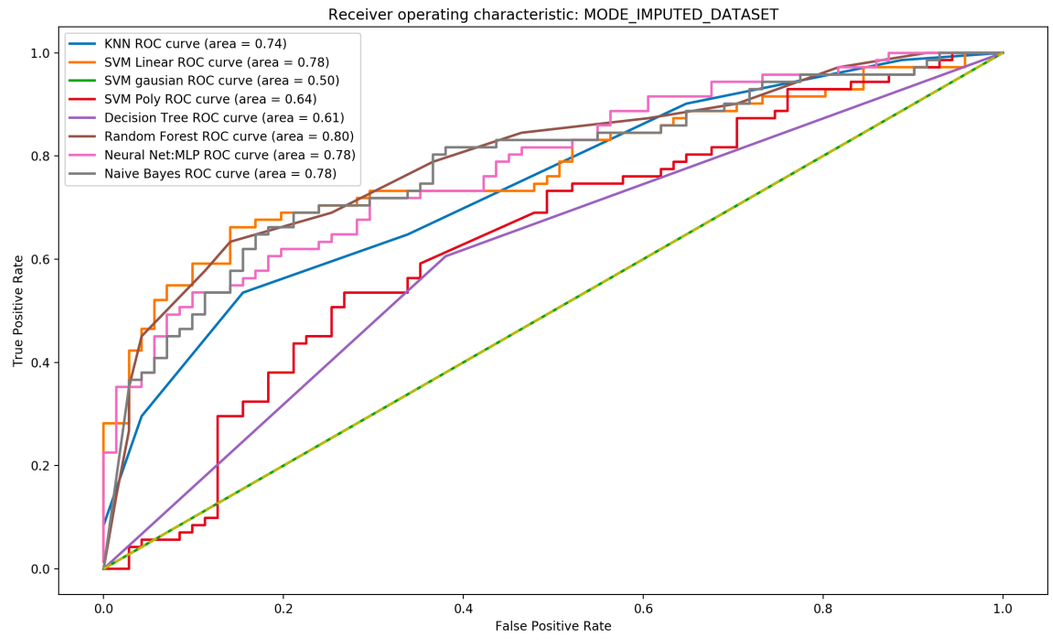

Figure 33: AUC score for under-sampled reduced data set.

RESULT ANALYSIS AND DISCUSSION:

We have used Stratified K Folds cross-validation technique to find the best parameter fit for all our classifiers used in our program. These best parameters estimated is shown in Table 2 above. Now once our parameters are estimated, we use these classifiers and train them using all the 3 of our training data sets (That is preprocessed and feature handled). And its corresponding test set is used for prediction and in turn their accuracy metrics (classification accuracy) are measured, namely F Measure and AUC value. For each of the data sets (Ignore, Mode imputed and SVM imputed) we calculate the F measure and AUC curve and is tabulated as shown in Table3.

With consideration of both accuracy measures, the SVM_DataSet produces very poor results. i.e. all the classifiers used for the above two data sets performed badly for this data set. Hence, we can come to a consensus the SVM based imputed data is not properly handled. The other two data sets show similar performance evaluations on almost all the classifiers used upon it. Their respective ROC plots are shown in the figure 26,27 and 28. These plots itself gives an idea that

the data set SVM_Imputed_DataSet is of not much use and for further study, we will use either of the other two data sets only.
Figures 21 through 25 show the performance statistics of each classifier on the dataset Ignore_DataSet. They contain, Precision, Recall, F measure and ROC_AUC values for all the classifiers used on our data set.
How to select the best Classifier or the Classification model? There is no such thing as the best classifier. The classifier performance varies largely depending on the data set it is trained on.
So, for each data there is a unique classifier or a bunch that equally work out well with the classification analysis. In our case we see that the Maximum F Measure = 0.88 is obtained by the SVM- Linear classifier and for the metric AUC value, Naïve Bayes and Random forest equally work out well for classification with an AUC value of 0.76 on average. And also, we can note that based on the variance in F Measure and AUC value, F Measure values vary over small range only and AUC values vary little more than the F Measures. Hence, we can consider AUC values as a standard benchmark to compare the Accuracy scores.
In our case, the AUC value (and also F measure) for both Random Forest classifier and Naïve Bayes classifier produce similar classification accuracy. Therefore, we compare the other available statistical data and put forth Random Forest as the best classifier as it as equally likely produced good classification accuracy for the SVM imputed Dataset.

So, ***Random Forests*** Classifier.
So far, we had worked on original preprocessed and feature space handled data. We had not performed any further feature dimension reduction. From our description of methods for feature space dimension reduction, we choose Select K Best features to select top 'n' best features. After feature dimension reduction, we compare its respective AUC value and is shown in figure 9. From there we can see that, feature reduction shows no improvement in classification accuracy, but rather drops. So, feature space dimensionality reduction cannot be used to improve the classification results. This was lightly intuitive when observing the correlation plots between different features.
Since the data is imbalanced,

(i) we performed data balancing (with over-sampling) using **SMOT**E function from the library *imblearn.over_sampling*. This function resamples the data of non-dominant class type and makes the dataset equally weighted. When this data set is passed through the round of classifiers, we get the corresponding F Measure and AUC values as shown in Table 4. Its corresponding ROC_AUC plot is shown in figure 32. There too we can see that our chosen classifier has out-performed the other classifiers and gives us a ROC_AUC value of 0.94.

(ii) We also performed data balancing (with under-sampling) using **RandomUnderSampler** from the library imblearn.under_sampling. This function resamples the dataset such that the class wise data dimensions become equal to the non-dominant class data points. This resampled data is run through different classifiers and its corresponding F Measure and AUC values is noted down as shown in Table 4. Its corresponding ROC_AUC plot is shown in figure 33. For our selected classifier, we get an ROC_AUC value of 0.80.

Based on the table and the figures, we can say that over sampling improves classification accuracy while in under-sampling also, the performance has not been much impacted, but we can say that it is more or less the same.

LINKS:

[1] http://www.stat.columbia.edu/~gelman/arm/missing.pdf (HANDLING MISSING DATA)
[2] https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/ (Imbalance handling)
[3] https://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn/ (Data Preprocessing)
[4] https://machinelearningmastery.com/scale-machine-learning-data-scratch-python/ (Data Scaling)
[5] http://scikit-learn.org/