EE 559 – Mathematical Pattern Recognition

Project

Development of a Pattern Recognition System on a real – world dataset: Bank- additional.csv

Muthulakshmi Chandrasekaran

USC ID : 4486180242

muthulac@usc.edu

May 1, 2018

Table of Contents

1. OBJECTIVE

The objective of the classification problem is to predict if a customer will subscribe to a term deposit, based on certain criteria. This is based on the given dataset bank – additional.csv.

2. ABSTRACT

This project aims to find the best classifier that predicts if a customer will subscribe to a term deposit. Data is extracted from the given dataset, data is first analyzed using any of visual interpretation methods – like histogram, correlation matrix or box plots. Upon analyzing data, missing values are imputed by different methods. Then the data is checked for any outliers, and normalized. Then the data is split into training and test sets, and cross validated. For each of the classifier implemented, the optimal parameters are found using cross – validation and applied on the test set. The performance metrics like F1 Score and AUC are found, and the nest classifier is reported.

3. APPROACH

Dataset Used – The dataset used for this project is Bank – additional.csv. This is from the UCI – Machine Learning Repository. This dataset is based on the results of phone call of a Portuguese Banking Institution. The objective of the classification problem is to predict if a client will subscribe to a term deposit or not.

Python is used, that has in – built libraries like pandas and scikit – learn. The approach can be given as follows:

i. Data is first extracted from .csv file using appropriate panda's commands.
ii. Data is then analyzed to learn its unique features.
iii. The class labels are obtained from the dataset, and the data set is divided into training set and test set.
iv. To make the data more balanced, preprocessing methods are done on the training set. Preprocessing includes imputing, scaling and balancing the dataset.

These could also be applied to the test set.

v. After preprocessing, dimensionality reduction methods are used – like Logistic Regression and PCA that extract only k significant features, where k can be user – defined.

vi. Classifier models are then defined. For each classifier, the best parameters for performance are obtained by cross – validation. Classifier models implemented here are Multi-Layer Perceptron, Gaussian Naïve Bayes, Support Vector Machines, Random Forest Classifier, Decision Tree and K – Nearest Neighbors.

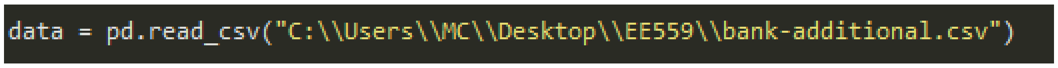
vii. For each of the classifier, the performance metrics are obtained. The evaluation criteria are training and test accuracies, Precision, Recall, F1 Score, ROC and AUC.

4. DEVELOPMENT OF PATTERN RECOGNITION SYSTEM – PROCEDURE

The various steps in developing a Pattern Recognition System are explained below.

4.1 Data Extraction:

The dataset is in a .csv file. It is extracted in Python using pandas, wherein the dataset is stored as a dataframe.

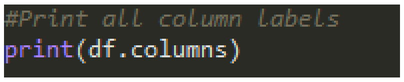


```
data = pd.read_csv("C:\\Users\\MC\\Desktop\\EE559\\bank-additional.csv")
```

Here, $data$ is the dataframe that contains the entire dataset.

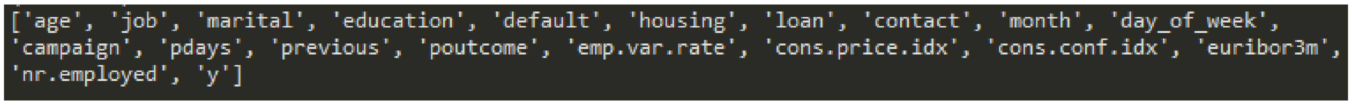
4.2 Dataset Description:

The data is analyzed fully to understand the features in the data. This $data$ has 20 columns – including the class labels. The data labels can be obtained using the command



```
#Print all column labels
print(df.columns)
```

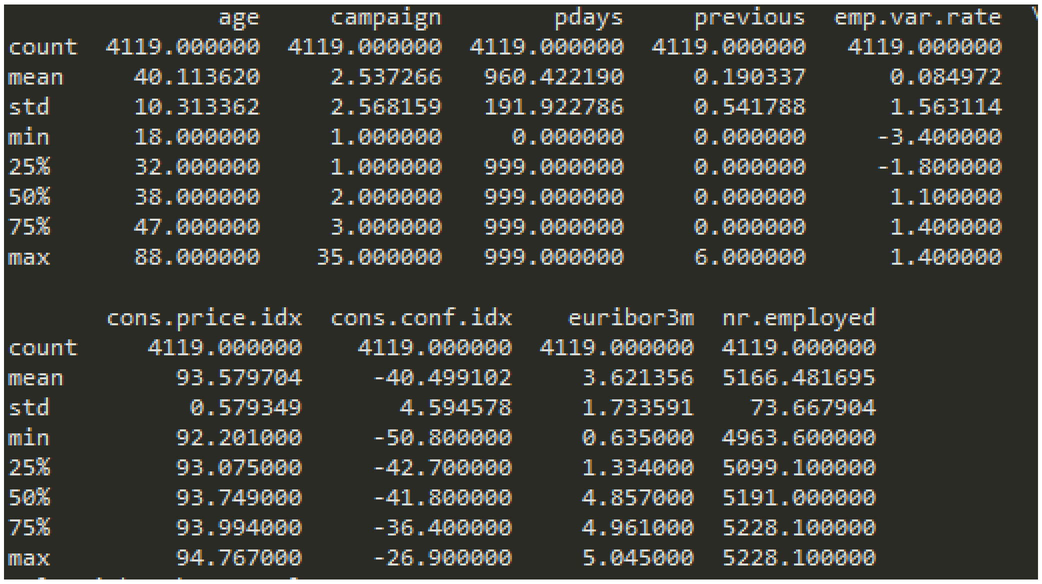
The different attributes are obtained as



```
['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week',
'campaign', 'pdays', 'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',
'nr.employed', 'y']
```

The output or the target label is y, that decides if the client has subscribed to the term deposit.

An overview of numerical data in the dataset can be given by `df.describe()`. The output is obtained as



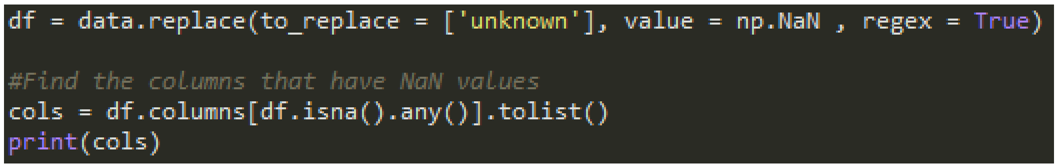
```
               age     campaign        pdays     previous  emp.var.rate
count  4119.000000  4119.000000  4119.000000  4119.000000   4119.000000
mean     40.113620     2.537266   960.422190     0.190337      0.084972
std      10.313362     2.568159   191.922786     0.541788      1.563114
min      18.000000     1.000000     0.000000     0.000000     -3.400000
25%      32.000000     1.000000   999.000000     0.000000     -1.800000
50%      38.000000     2.000000   999.000000     0.000000      1.100000
75%      47.000000     3.000000   999.000000     0.000000      1.400000
max      88.000000    35.000000   999.000000     6.000000      1.400000

       cons.price.idx  cons.conf.idx    euribor3m  nr.employed
count     4119.000000    4119.000000  4119.000000  4119.000000
mean        93.579704     -40.499102     3.621356  5166.481695
std          0.579349       4.594578     1.733591    73.667904
min         92.201000     -50.800000     0.635000  4963.600000
25%         93.075000     -42.700000     1.334000  5099.100000
50%         93.749000     -41.800000     4.857000  5191.000000
75%         93.994000     -36.400000     4.961000  5228.100000
max         94.767000     -26.900000     5.045000  5228.100000
```

The description of each feature can be given as in Appendix A.

4.3 Dataset Interpretation:

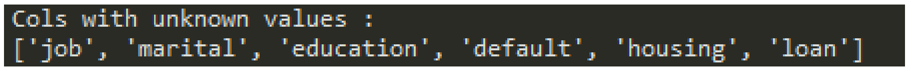
The dataset has missing values that are labelled as *unknown*. The columns that contain unknown values are found by replacing the unknown values by NaN, and using the following command.



```
df = data.replace(to_replace = ['unknown'], value = np.NaN , regex = True)

#Find the columns that have NaN values
cols = df.columns[df.isna().any()].tolist()
print(cols)
```

The columns that contain unknown values are



```
Cols with unknown values :
['job', 'marital', 'education', 'default', 'housing', 'loan']
```

The unique features are visualized using Histogram Plots, Correlation Matrix and Scatter Plots.

Figure 1 shows the histogram plots obtained for the numerical feature.

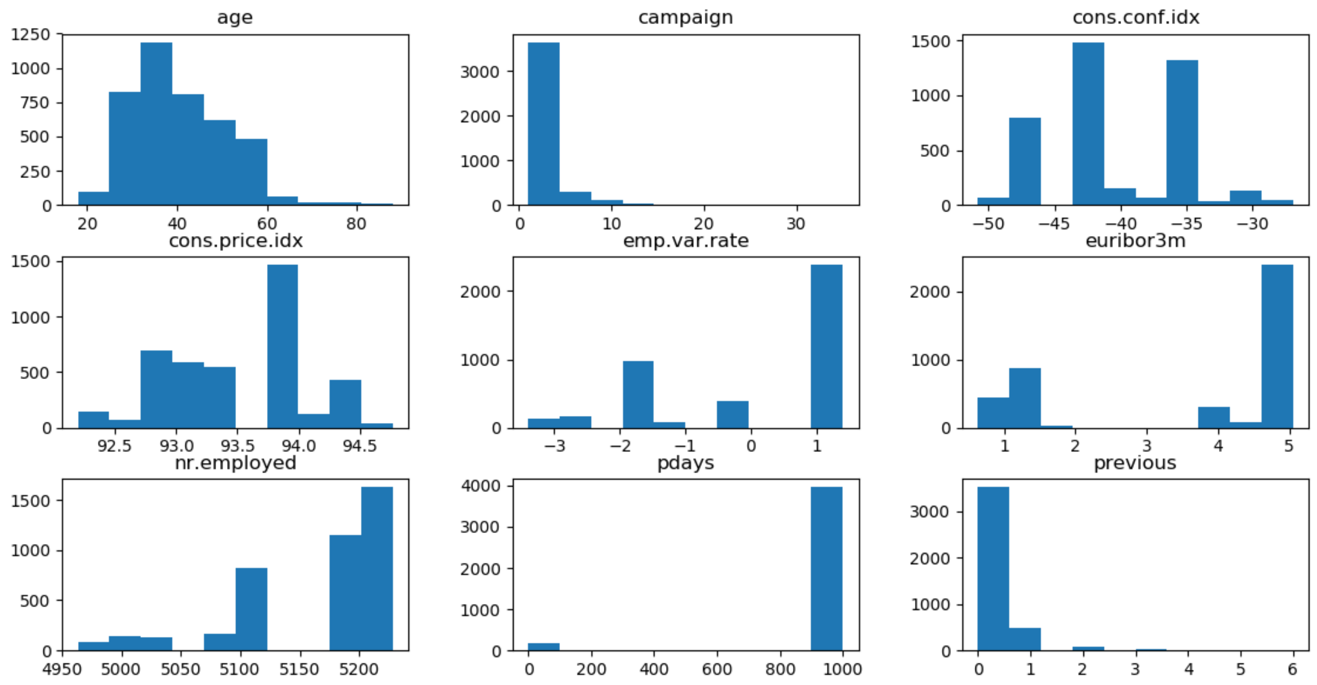


Figure 1 Histogram Visualization of the Numerical features

From this histogram, the range of the numerical features can be identified, this also gives the mode of each feature, and hence the normalization method can be found.

Figure 2 shows the Correlation Matrix obtained for the numerical features.

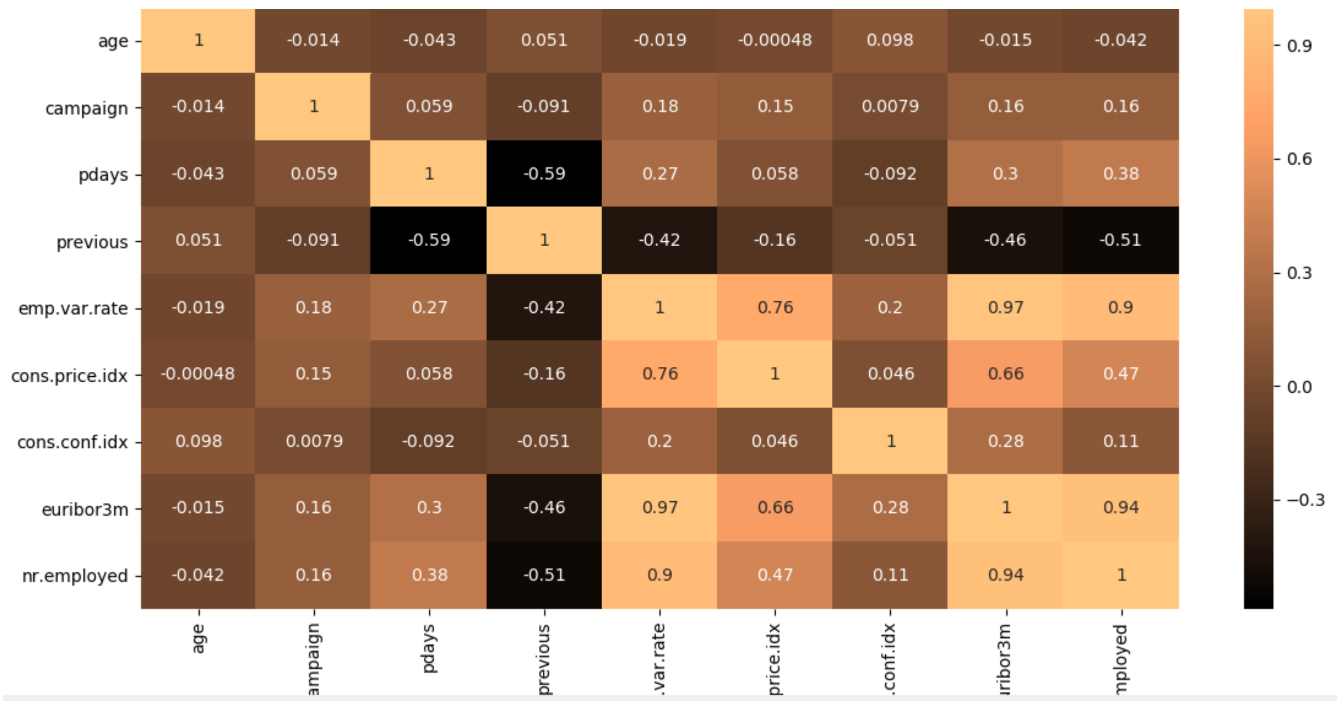


Figure 2 Correlation Matrix

The correlation matrix can be used to find the extent of correlation between numeric features, which can be later used for feature selection.

Figure 3 shows the box plots obtained for each of the numeric variables.

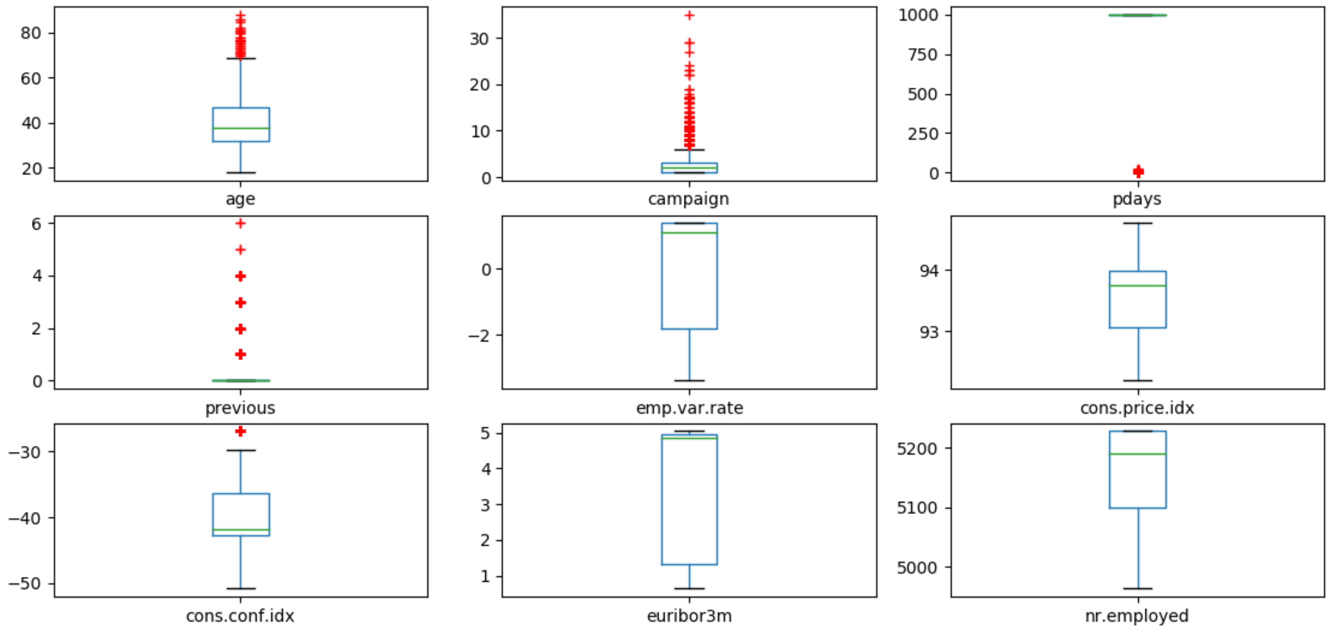


Figure 3 Box Plots for Numerical Features

Box plots give the number of outliers, if any, for each feature. In Figure 3, the number of outliers are given by the red crosses. And since there are outliers (or extreme values), corresponding preprocessing can be applied.

4.4 Data Pre – Processing:

Data pre – processing is an important part, since it prepares the data for further processes. It improves the dataset for suitable processing.

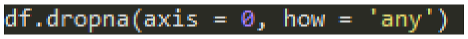
From the visualization graphs, it is observed that the **default** column has only one **yes** value, and hence that row can be removed. Now since all the values of the default column are **no**, the entire feature can be removed, as it doesn't affect the classification.

Similarly, the only row containing **illiterate** as education can also be removed, since it is only a single value, and it does not affect the classification.

The pre – processing steps done are

i. **Data Imputing**

Data Imputing is done to remove the missing data in the dataset. The methods of imputing used here are removal of missing data columns and imputing with class wise mode. Hence there are two datasets obtained.

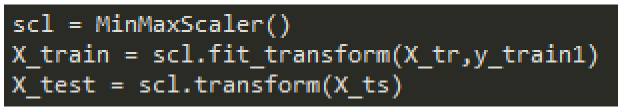
The first dataset is obtained by 
```
df.dropna(axis = 0, how = 'any')
```
, where all rows containing NaN are removed.

The second dataset is obtained by replacing the unknown values with the class wise mode of each feature.

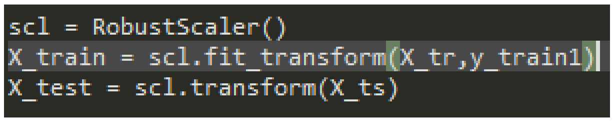
ii. **Data Scaling**

The raw data could be along various scales. Hence, all data are scaled to the same scale before further processing. The types of scaling methods used here include

- MinMax Scaler – This scale each feature to the range (0,1).


```
scl = MinMaxScaler()
X_train = scl.fit_transform(X_tr,y_train1)
X_test = scl.transform(X_ts)
```

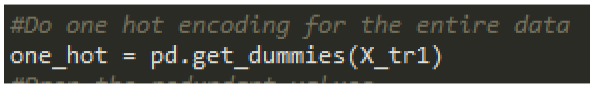
- RobustScaler – This scale to according to the quantile range, by removing median. This scaler removes any outliers present in the data.


```
scl = RobustScaler()
X_train = scl.fit_transform(X_tr,y_train1)
X_test = scl.transform(X_ts)
```

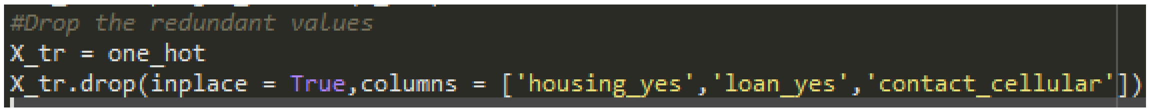
Data Balancing can also be done to make the dataset balanced.

iii. **Data Encoding**

Not all data can be processed by classifiers in the actual format. Hence categorical data must be encoded. Hence one – hot encoding is used to convert the categorical data into numbers. This enables more predictions for the output labels. It is done as


```
#Do one hot encoding for the entire data
one_hot = pd.get_dummies(X_tr1)
```

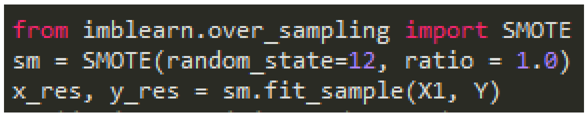
After one hot encoding is done, redundant values – i.e. features having two values – are redundant. Hence those columns are dropped for both test and train data, as shown below.



```
#Drop the redundant values
X_tr = one_hot
X_tr.drop(inplace = True,columns = ['housing_yes','loan_yes','contact_cellular'])
```

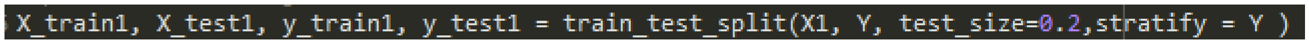
iv. **Data Sampling**

Data is unbalanced – and hence the data is balanced using SMOTE. Oversampling or under sampling can be done. Since the data set is unbalanced, oversampling leads to a balance in the number of datapoints, and hence a higher performance can be expected. It is implemented as



```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=12, ratio = 1.0)
x_res, y_res = sm.fit_sample(X1, Y)
```

The data is now split into test and train data as



```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, Y, test_size=0.2,stratify = Y )
```

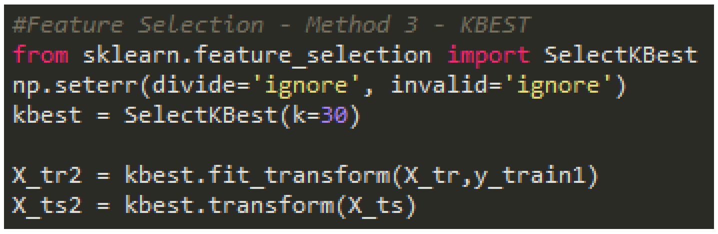
It is important that whatever preprocessing method is applied on the training set, the same must be applied on the test set. Stratify is Y to split the data in a stratified fashion. Test size of 0.2 means 20% of the data is the test data.

4.5 Model Selection:

After preprocessing is done, proper model must be identified. Dimensionality reduction is the process of reducing the number of features under consideration. This can be divided into Feature Selection and Feature Extraction. The steps in model selection can be summarized as follows:

i. **Feature Selection** – The preprocessed data contains many features. Hence only certain significant and representative features can be chosen as the input to the classifier. Here two methods of feature selection are tried – SelectKBest and Logistic Regression.

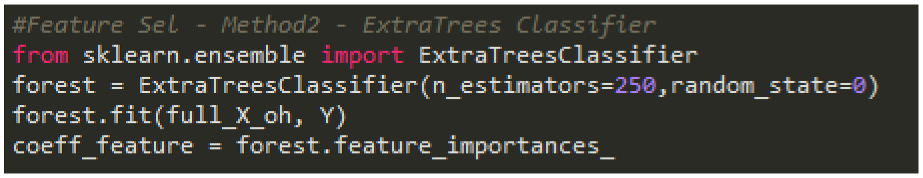
- SelectKBest – This feature selection method scores the features according to a metric (default is f_classif), and keeps only the k important features. This is a kind of feature wrapper technique. SelectKBest is implemented as shown below.



```
#Feature Selection - Method 3 - KBEST
from sklearn.feature_selection import SelectKBest
np.seterr(divide='ignore', invalid='ignore')
kbest = SelectKBest(k=30)

X_tr2 = kbest.fit_transform(X_tr,y_train1)
X_ts2 = kbest.transform(X_ts)
```

- Extra Trees Classifier – Extra Trees method (or **Ext**remely **Ra**ndom **Tree**s) is a method of feature selection using random trees. In this classifier, the entire input sample is used at each stage, and the decision boundaries are picked at random. Random splits are done each time for the maximum number of features. This is implemented as shown below.



```
#Feature Sel - Method2 - ExtraTrees Classifier
from sklearn.ensemble import ExtraTreesClassifier
forest = ExtraTreesClassifier(n_estimators=250,random_state=0)
forest.fit(full_X_oh, Y)
coeff_feature = forest.feature_importances_
```

Feature Importance graph using Extra Trees Classifier is shown in Figure 4.

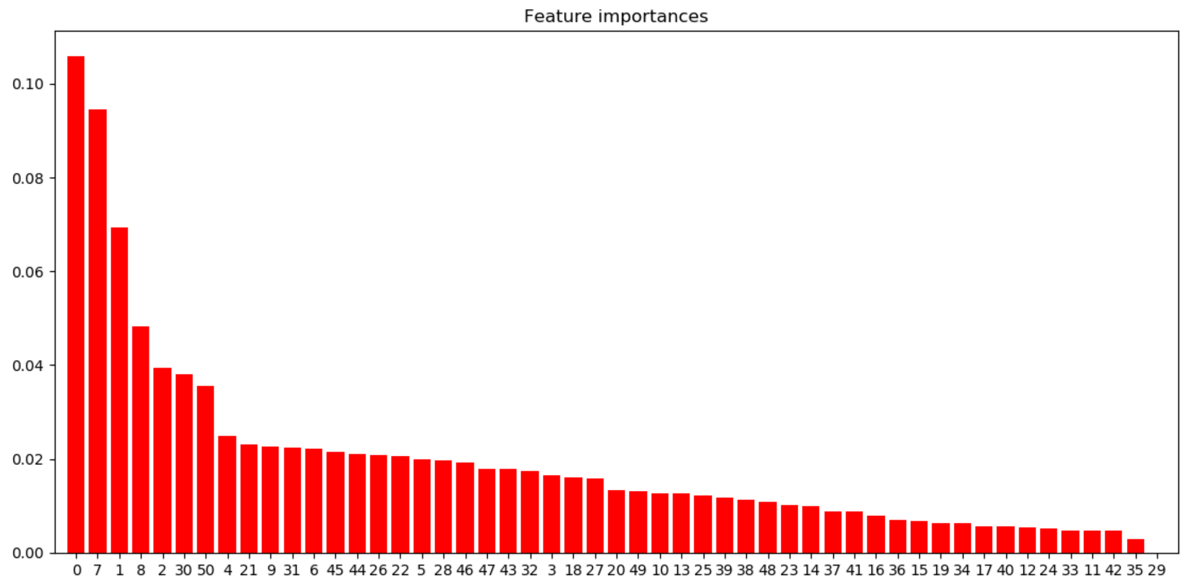
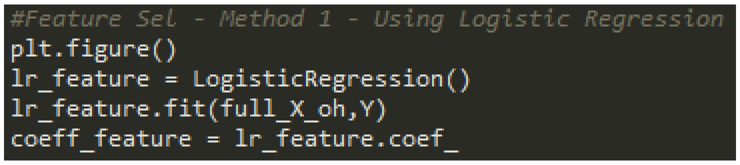


Figure 4 Feature Importance Graph using Extra Trees Classifier

The xlabels denote the column number of the features. Y axis is the importance level.

- Logistic Regression – This method can be also used to find the feature importance, especially when the classes are not well – separated, this logistic regression gives better decision boundaries. Logistic Regression can be used for feature selection as follows.



```
#Feature Sel - Method 1 - Using Logistic Regression
plt.figure()
lr_feature = LogisticRegression()
lr_feature.fit(full_X_oh,Y)
coeff_feature = lr_feature.coef_
```

Feature Importance graph using Logistic Regression is shown in Figure 5.

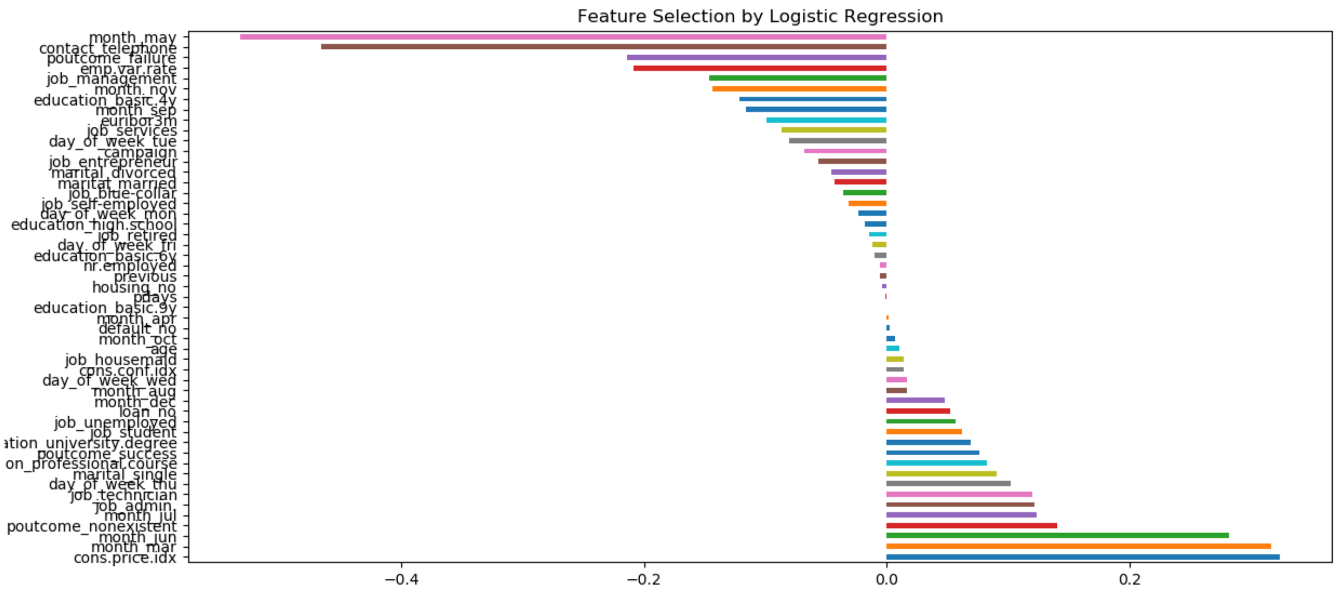
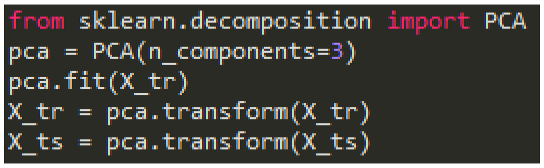


Figure 5 Feature Importance Graph using Logistic Regression

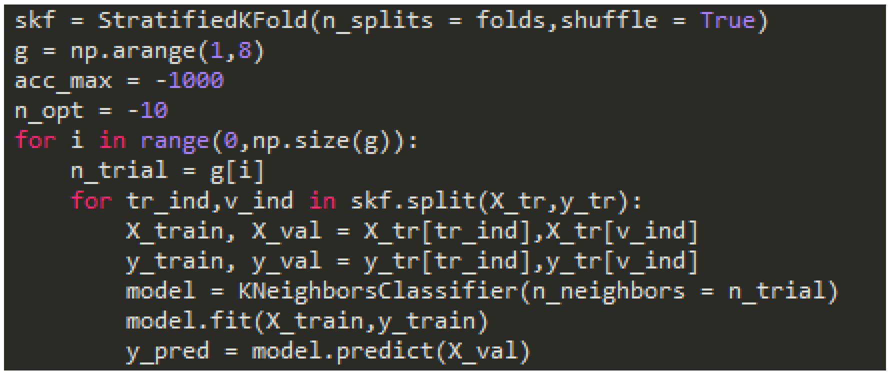
ii. **Feature Extraction –** Feature Extraction transforms the higher dimensional data into fewer dimensions using methods like PCA. PCA performs a linear mapping of data on lower dimensions. The variance of the data in lower – dimensions is reduced. PCA is implemented using the following commands. PCA does dimensionality reduction by finding the correlation between the variables. Hence PCA can give better results when applied to variables having strong correlation. PCA finds the directions of maximum variance in many dimensions and projects into smaller dimensions, while retaining essential information.



```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
pca.fit(X_tr)
X_tr = pca.transform(X_tr)
X_ts = pca.transform(X_ts)
```

The number of reduced dimensions is controlled by the parameter $n\_components$,

iii. **Cross Validation –** A model can refer to the method of describing how well the input data related to what we try to predict. K – fold cross validation tests how well a model is able to get trained by some data, and predict on the test data. To do an 80/20 split, five – fold cross validation is used. This trains the model on 80% of the data as training and 20% of the data as the test set – for five times. Hence cross – validation can be used for model checking – i.e. finding out the best parameters for each model. This can be applied to most of the classifiers that are implemented. For a classifier, K – fold cross validation is done on the training set to predict which parameters give the best results. Then it is applied on the training set. Stratified K – Fold is used because it preserves the [percentage of samples in each class. A snippet of cross – validation implementation is shown below.



```
skf = StratifiedKFold(n_splits = folds,shuffle = True)
g = np.arange(1,8)
acc_max = -1000
n_opt = -10
for i in range(0,np.size(g)):
    n_trial = g[i]
    for tr_ind,v_ind in skf.split(X_tr,y_tr):
        X_train, X_val = X_tr[tr_ind],X_tr[v_ind]
        y_train, y_val = y_tr[tr_ind],y_tr[v_ind]
        model = KNeighborsClassifier(n_neighbors = n_trial)
        model.fit(X_train,y_train)
        y_pred = model.predict(X_val)
```

4.6 Implementation of Classifiers :

Data is now complete to implement into the classifier models. The classification models implemented here are SVM, Multi Layer Perceptron, K Nearest Neighbor, Naïve Bayes and Decision Tree Classifier.

i. **Support Vector Machines**

Support Vector Machines are models used for supervised learning. This can be used for linear classification and regression. The decision boundary in a SVM is a hyperplane, that separates the two classes. SVM finds the hyperplane that has the largest margin to the training sample points i.e. large margin. An optimal SVM maximizes the margin. The different kernels used are linear, RBF and Polynomial kernels.

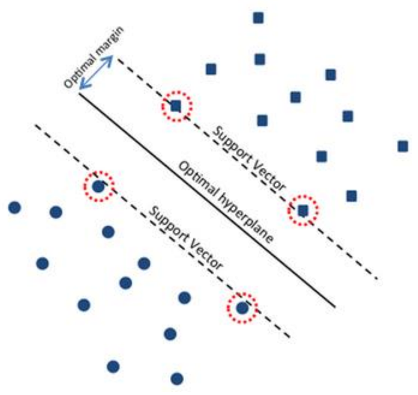
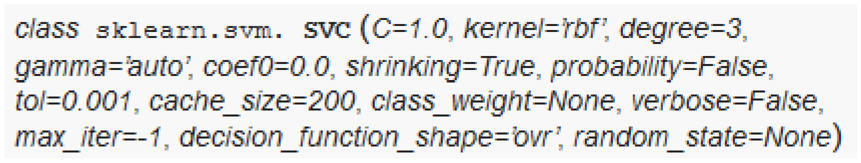


Figure 6 Support Vector Machine – An illustration[Ref :5]

In Python, the following commands are used to implement this classifier.



```
class sklearn.svm. SVC (C=1.0, kernel='rbf', degree=3,
gamma='auto', coef0=0.0, shrinking=True, probability=False,
tol=0.001, cache_size=200, class_weight=None, verbose=False,
max_iter=-1, decision_function_shape='ovr', random_state=None)
```

ii. Multi Layer Perceptron

Multi – Layer Perceptron consists of more than one perceptron, that are arranged in layers. It consists of a single input layer, multiple hidden layers for computation, and an output layer for decision making. These multi layer perceptron are supervised learning algorithms, which train on a set of input – output pairs. They learn the correlation between the inputs and outputs. An illustration of the Multi Layer Perceptron is shown below[Ref:3].

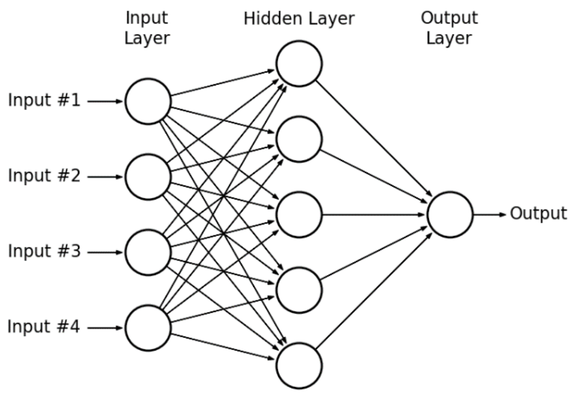
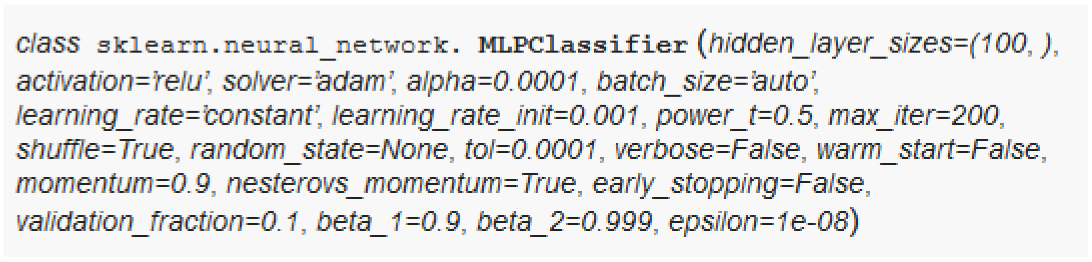


Figure 7 Multi Layer Perceptron – An illustration

The best parameters for MLP are obtained using cross validation. The parameters that are changed in MLP are the size of hidden layers. In Python, the following commands are used to implement this classifier.



```
class sklearn.neural_network. MLPClassifier (hidden_layer_sizes=(100, ),
activation='relu', solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200,
shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False,
momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```

iii. K – Nearest Neighbor

K – Nearest Neighbor is a non – parametric algorithm, which does not make any assumptions on the underlying data distribution. KNN uses the entire dataset as its representation. For an unknown data point, KNN classifies the point based on its neighbors. The data point is assigned the class of the nearest k – neighbors.

The nearest neighbor computation can be a measure of any distance – say L1 norm or L2 norm. For example, if L2 norm is taken, we compute the Euclidean distance between the point and its k nearest neighbors. The data point is assigned the class of its majority neighbors. A simple illustration of the KNN classifier is shown below[Ref : 2].

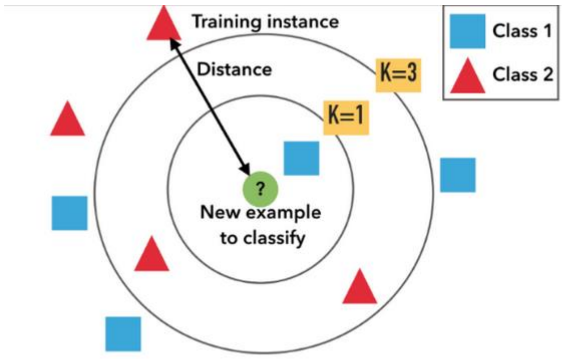
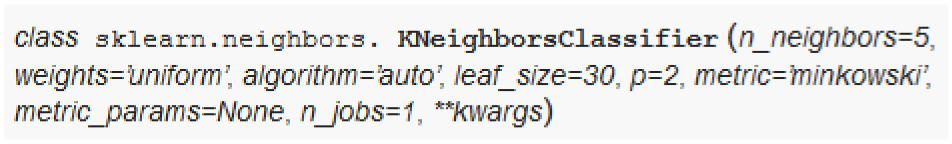


Figure 8 K Nearest Neighbor – An illustration

The best parameters for KNN are obtained by cross – validation. The parameters that are changed in KNN are the number of neighbors. In Python, the following commands are used to implement this classifier.



```
class sklearn.neighbors. KNeighborsClassifier (n_neighbors=5,
weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski',
metric_params=None, n_jobs=1, **kwargs)
```

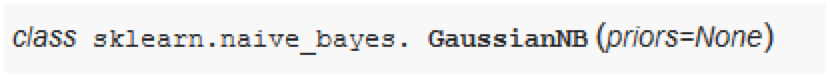
iv. Naïve Bayes Classifier

The Naive Bayes is a classification technique based on the Baye's theorem, which is given as

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Prior probabilities are calculated as a probability i.e. the number of occurrences of each sample divided by the total number of samples. Bayes classifier assumes all the variables used are independent of one another, and hence the conditional probabilities are calculated as products of individual probabilities. Naïve Bayes gives better results even with lesser training data. It also has the advantage that the convergence is quicker when the independence condition holds.

The default parameters are used in Naïve Bayes Classifier. In Python, the following commands are used to implement this classifier.

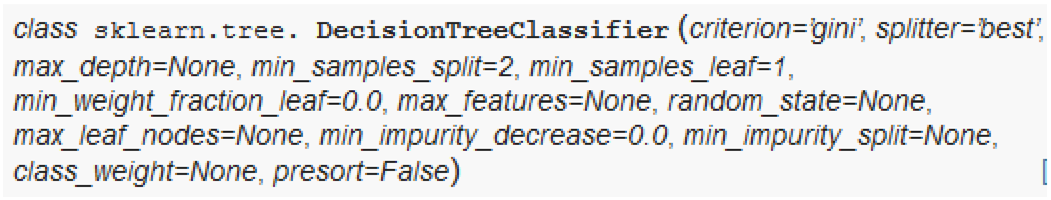


```
class sklearn.naive_bayes. GaussianNB (priors=None)
```

v. Decision Tree Classifier

A Decision Tree is a predictive learning approach, wherein the output value is based on predictions of multiple inputs. At each step of the decision tree, the target is selected based on certain criteria. A decision tree starts with a root node at the top. At each level in the tree, a node denotes a condition check on a feature, and each branch in the tree denotes the outcome of the condition check. Every node represents a class label. Decision trees follow top – down approach, wherein at each step, an optimum variable is chosen for classification. Though these decision trees are not as accurate as the other classifiers, this is simple and efficient as it can handle both categorical and numerical data. Sometimes, decision trees can also go into overfitting.

The parameters to be changed are the maximum depth of the decision trees. In Python, the following commands are used to implement this classifier.

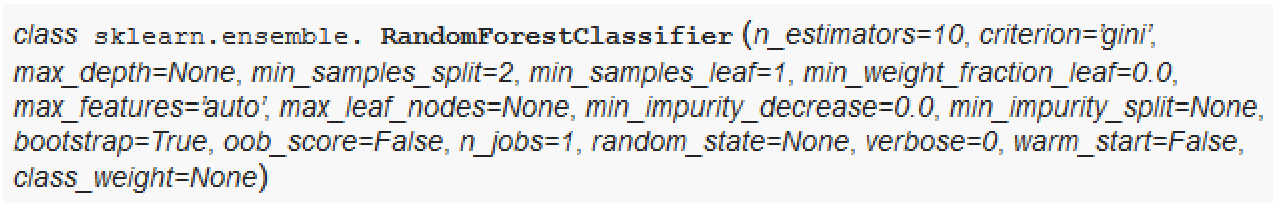


```
class sklearn.tree. DecisionTreeClassifier (criterion='gini', splitter='best',
max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
class_weight=None, presort=False)                                          [
```

vi. Random Forest Classifier

Random Forests are an extension of the decision tress. They contain multiple decision trees, and the output prediction is either the mode or mean of the individual decision trees. The advantages of random forests is that random forests reduce the risk of overfitting the data, by combining many decision trees. This reduces the variance and hence makes the model more robust.

The parameters changed in Random Forest Classifier are the number of estimators , which are calculated by cross validation. In Python, the following commands are used to implement this classifier.



```
class sklearn.ensemble. RandomForestClassifier (n_estimators=10, criterion='gini',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
class_weight=None)
```

5. RESULTS

The performance metrics that are used for evaluation are Precision, Recall, F1 Score, Accuracy and AUC_ROC. This depends on True Positive, True Negative, False Positive and False Negatives. True Positive and True Negative are correctly classified measures, and False Positive and False Negative are misclassification measures.

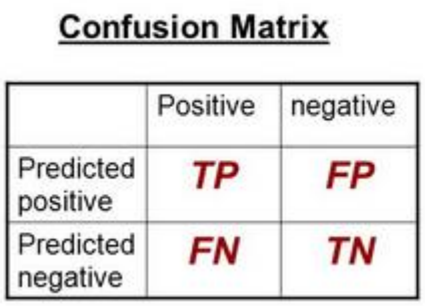
True Positive – Percentage of correctly predicted true values.

True Negative – Percentage of correctly predicted false values.

False Positive – Percentage of misclassified true values

False Negative – Percentage of misclassified false values

The confusion Matrix is defined as



Confusion Matrix

	Positive	negative
Predicted positive	TP	FP
Predicted negative	FN	TN

- Accuracy defines the percentage of correctly predicted samples, given by

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

It is calculated using

```
sklearn.metrics. accuracy_score (y_true, y_pred, normalize=True,
sample_weight=None) ¶
```

- Precision is the ratio of true positives to the total predicted positive observations.
- Recall is the ratio of true positives to the all observations in actual class.
- F1 Score is calculated based on Precision and Recall. F1 Score is the weighted average of Precision and Recall.

$$\textbf{Precision} = \frac{TP}{TP + FP} \qquad \textbf{Recall} = \frac{TP}{TP + FN}$$

$$\textbf{F1-score} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

It is calculated as

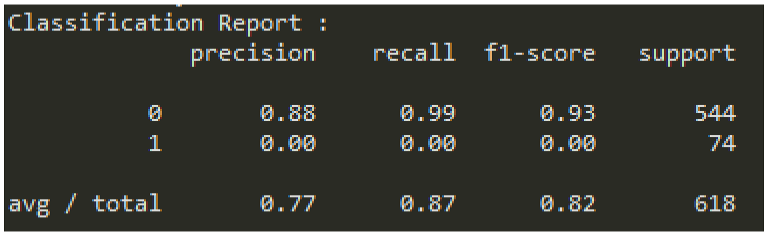
```
sklearn.metrics. precision_recall_fscore_support (y_true, y_pred, beta=1.0,
labels=None, pos_label=1, average=None, warn_for=('precision', 'recall', 'f-score'),
sample_weight=None)
```

- Receiver Operating Characteristic(ROC) is a curve of the True Positive Rate against the False Positive Rate for a set of input data. It shows the relationship between the FP and TP. A curve closer to the TP axis means the accuracy is more. A curve closer to the 45 – degree diagonal line means less accuracy. Area Under the Curve is a measure of the test accuracy. This area measures the ability of the classifier to correctly discriminate between different classes. This is helpful mostly for a binary classifier.
  It is calculated as

```
sklearn.metrics. roc_auc_score (y_true, y_score, average='macro',
sample_weight=None) ¶
```

The precision and recall are obtained using the command

```
sklearn.metrics. classification_report (y_true, y_pred, labels=None,
target_names=None, sample_weight=None, digits=2)
```



```
Classification Report :
             precision    recall  f1-score   support

          0       0.88      0.99      0.93       544
          1       0.00      0.00      0.00        74

avg / total       0.77      0.87      0.82       618
```

Two datasets are evaluated – first wherein all the rows with unknown values are removed, and the second dataset, where the values are imputed with class wise modes.

The dataset is extracted for features using SelectKBest method, and PCA is done.

In computing the values, average takes the value weighted, because the dataset is an unbalanced dataset.

SVM – Linear Classifier: The parameters of gamma and C are varied between [0.001,1000]. The best value is obtained by cross – validation. Optimal was C = 100 and gamma = 0.054.

SVM – RBF Classifier: The parameters of gamma and C are varied between [0.001,1000]. The best value is obtained by cross – validation. Optimal was C = 50.00 and gamma = 0.2

SVM – Polynomial Classifier: The parameters of degree, gamma and C are varied. Degree is varied between 1 and 6. Gamma and C are varied between [0.001,1000]. The optimal parameters were degree = 5, C = 215 and gamma = 4.356.

Multi Layer Perceptron Classifier: The parameters that were varied are the number of hidden layers. Multiple hidden layers were tested – 1 to 3. The number of neurons in each layer is varied from 10 to 200. The optimal was 132.

Naïve Bayes Classifier: The default parameters were used.

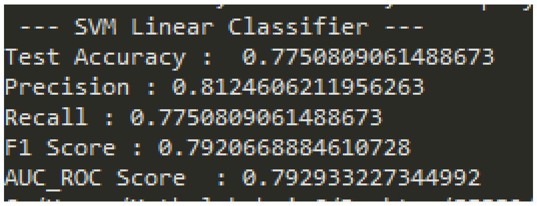
Decision Tree Classifier: Depth of the classifier is changed from 1 to 100. The optimal was 21.

Random Forest Classifier: The number of estimators is changed on a scale from 1 to 100. The optimal was 34.

K – Nearest Neighbors Classifier: The number of nearest neighbors to be considered is varied from 1 to 8. The optimal was 6.

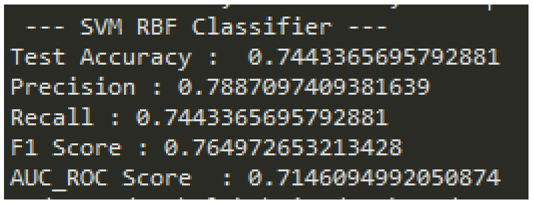
These optimal values change every trial, since shuffle in enabled.

The following results are for the first dataset – wherein the instances with unknown values are removed. These samples in Figure 9 are for the dataset 2, where mode imputation is performed.



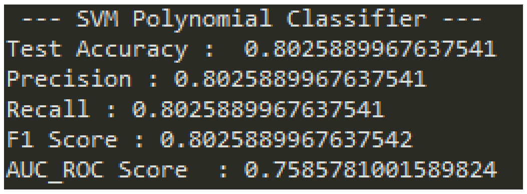
```
 --- SVM Linear Classifier ---
Test Accuracy :  0.7750809061488673
Precision : 0.8124606211956263
Recall : 0.7750809061488673
F1 Score : 0.7920668884610728
AUC_ROC Score  : 0.792933227344992
```

Figure 9(a) Output of SVM – Linear Classifier



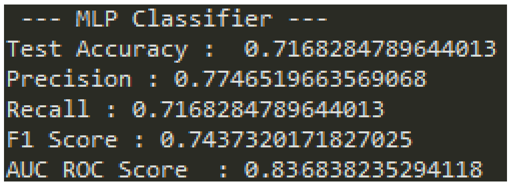
```
 --- SVM RBF Classifier ---
Test Accuracy :  0.7443365695792881
Precision : 0.7887097409381639
Recall : 0.7443365695792881
F1 Score : 0.764972653213428
AUC_ROC Score  : 0.7146094992050874
```

Figure 9(b) Output of SVM – RBF Classifier



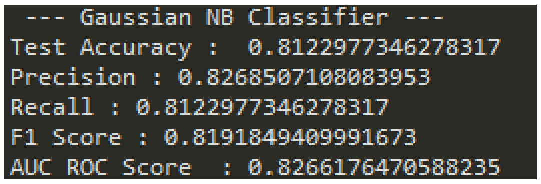
```
 --- SVM Polynomial Classifier ---
Test Accuracy :  0.8025889967637541
Precision : 0.8025889967637541
Recall : 0.8025889967637541
F1 Score : 0.8025889967637542
AUC_ROC Score  : 0.7585781001589824
```

Figure 9(c) Output of SVM – Polynomial Classifier



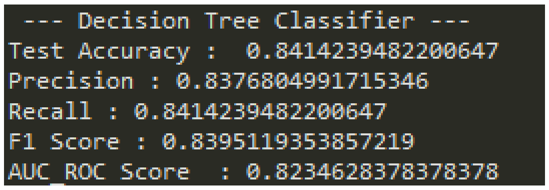
```
 --- MLP Classifier ---
Test Accuracy :  0.7168284789644013
Precision : 0.7746519663569068
Recall : 0.7168284789644013
F1 Score : 0.7437320171827025
AUC ROC Score  : 0.836838235294118
```

Figure 9(d) Output of Multi Layer Perceptron Classifier



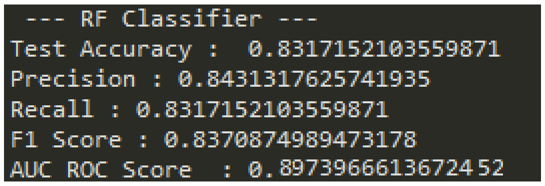
```
 --- Gaussian NB Classifier ---
Test Accuracy :  0.8122977346278317
Precision : 0.8268507108083953
Recall : 0.8122977346278317
F1 Score : 0.8191849409991673
AUC ROC Score  : 0.8266176470588235
```

Figure 9(e) Output of Naïve Bayes Classifier



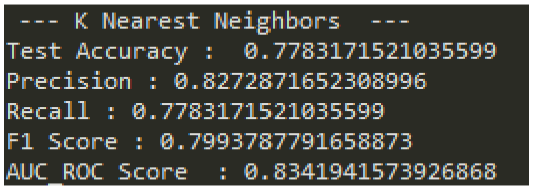
```
 --- Decision Tree Classifier ---
Test Accuracy :  0.8414239482200647
Precision : 0.8376804991715346
Recall : 0.8414239482200647
F1 Score : 0.8395119353857219
AUC_ROC Score  : 0.8234628378378378
```

Figure 9(f) Output of Decision Tree Classifier



```
 --- RF Classifier ---
Test Accuracy :  0.8317152103559871
Precision : 0.8431317625741935
Recall : 0.8317152103559871
F1 Score : 0.8370874989473178
AUC ROC Score  : 0.8973966613672452
```

Figure 9(g) Output of Random Forest Classifier



```
 --- K Nearest Neighbors  ---
Test Accuracy :  0.7783171521035599
Precision : 0.8272871652308996
Recall : 0.7783171521035599
F1 Score : 0.7993787791658873
AUC_ROC Score  : 0.8341941573926868
```

Figure 9(h) Output of K – Nearest Neighbor Classifier

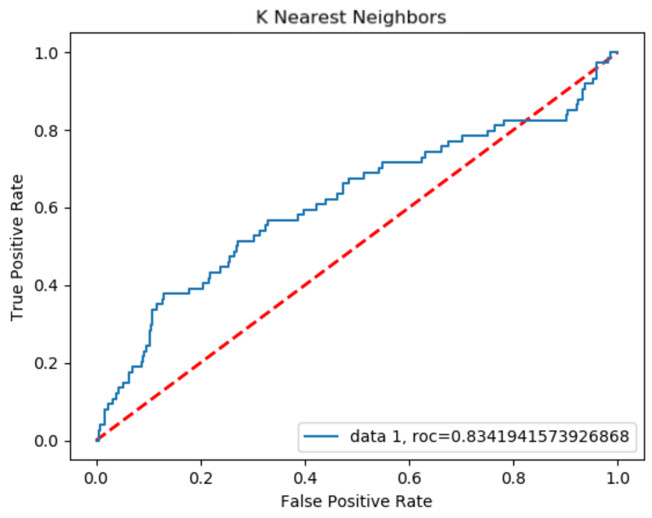
The following table shows the consolidated values of Precision and Recall for the dataset 1 – wherein all unknown values were removed.

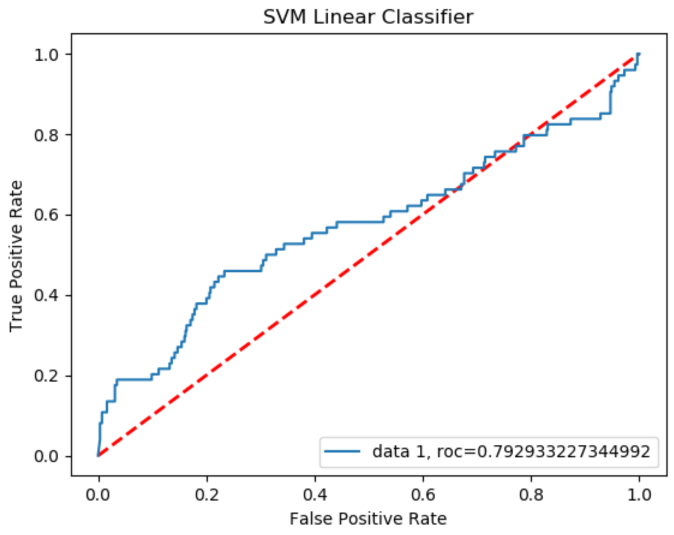
Dataset : All unknown values removed						
Classifier	**Training Accuracy**	**Test Accuracy**	**Precision**	**Recall**	**F1 Score**	**ROC_AUC Score**
SVM – Linear	0.752	0.740	0.791	0.683	0.730	0.715
SVM – RBF	0.725	0.619	0.793	0.618	0.687	0.708
SVM – Polynomial	0.735	0.728	0.799	0.728	0.760	0.699
Multi Layer Perceptron	0.89	0.884	0.792	0.88	0.836	0.738
Naïve Bayes	0.860	0.845	0.843	0.845	0.844	0.734
Decision Tree	0.87	0.854	0.793	0.888	0.838	0.709
Random Forest	0.84	0.826	0.824	0.810	0.813	0.748
K – Nearest Neighbors	0.889	0.871	0.793	0.852	0.838	0.75

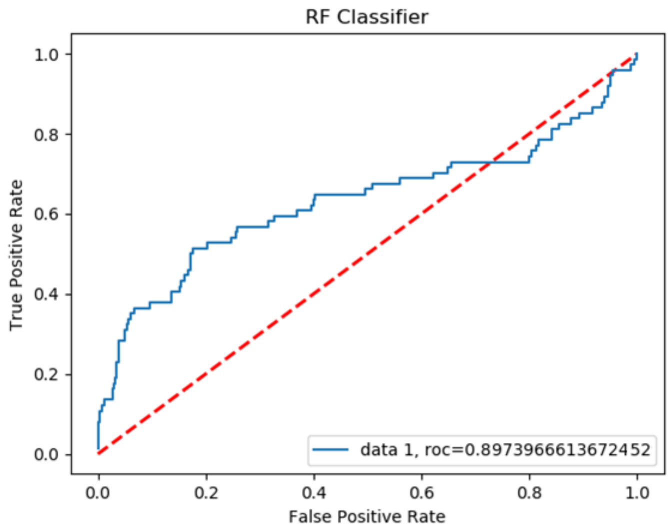
The following table shows the consolidated values of Precision and Recall for the dataset 2 – wherein all unknown values were imputed with class wise modes.

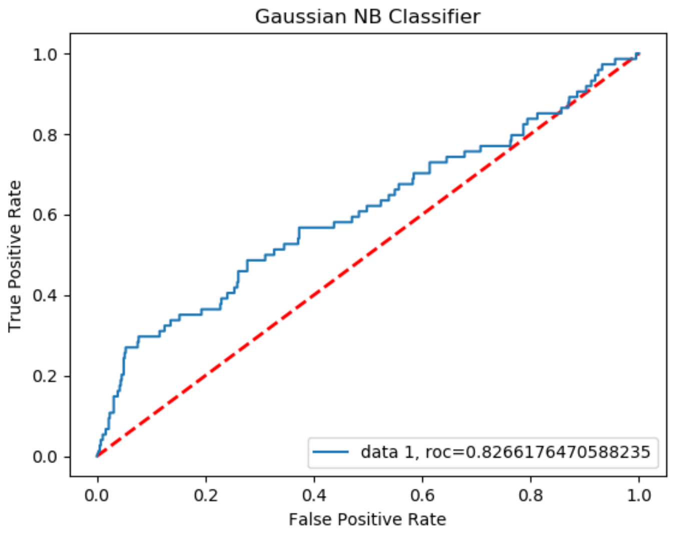
Dataset : All unknown values imputed with class wise modes						
Classifier	**Training Accuracy**	**Test Accuracy**	**Precision**	**Recall**	**F1 Score**	**ROC_AUC Score**
SVM – Linear	0.782	0.775	0.812	0.775	0.792	0.792
SVM – RBF	0.778	0.744	0.788	0.744	0.764	0.714
SVM – Polynomial	0.815	0.802	0.802	0.802	0.802	0.758
Multi Layer Perceptron	0.756	0.716	0.774	0.716	0.743	0.836
Naïve Bayes	0.820	0.812	0.826	0.812	0.819	0.826
Decision Tree	0.845	0.841	0.837	0.841	0.839	0.823
Random Forest	0.841	0.831	0.843	0.831	0.837	0.852
K – Nearest Neighbors	0.802	0.778	0.827	0.778	0.799	0.834

It is inferred that the mode imputation gives better results, and hence the Area Under Curve of the Receiver Operating Characteristic is shown only for this case. For testing other parameters, this dataset is only used. The following curves are obtained for the second dataset, imputed by class modes.









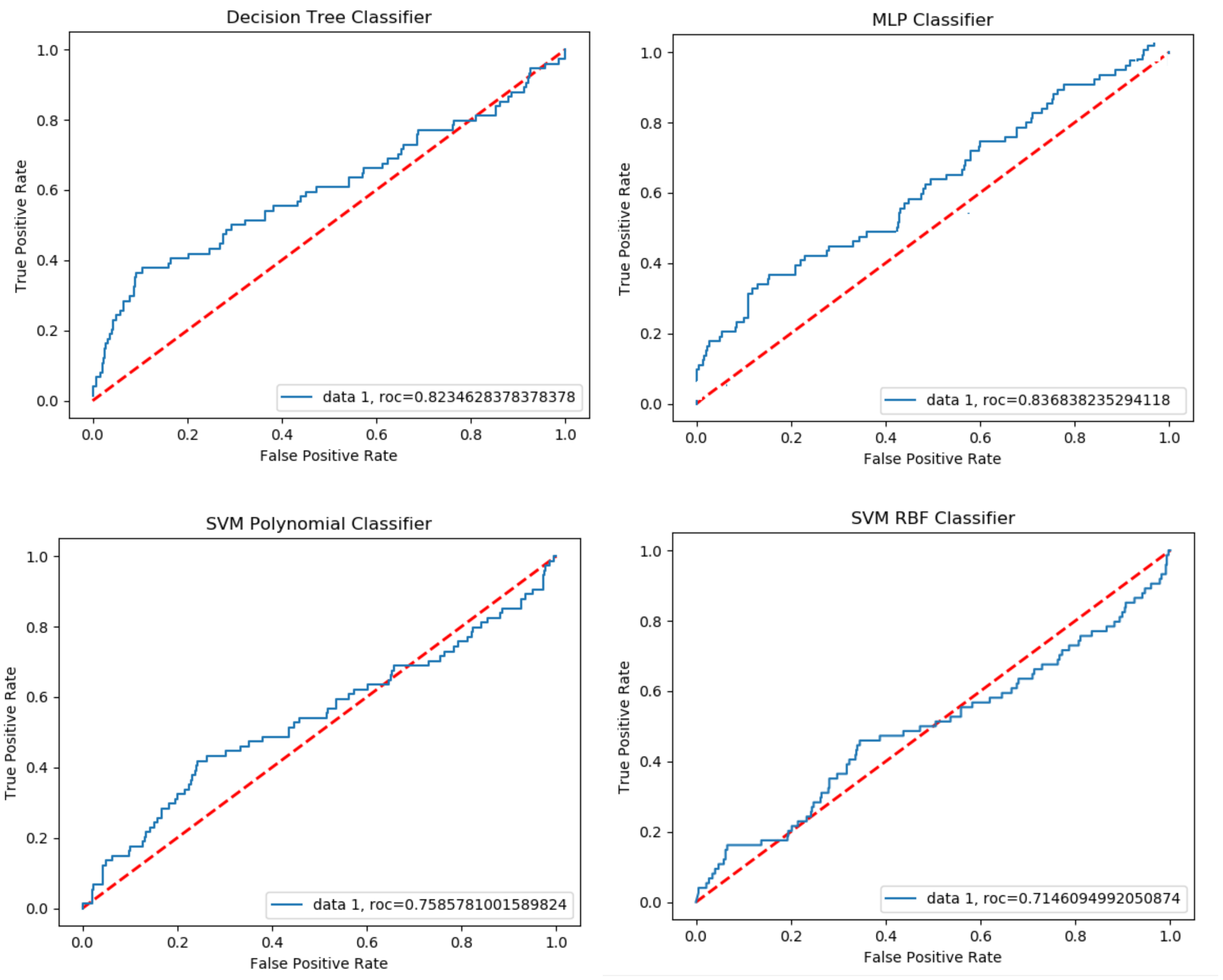


Figure 10 AUC_ROC Curves

Of all the classifiers that were computed, RF Classifier gives the Highest ROC, and RF Classifier gives the highest F1 score. It is expected that MLP has a higher F1 score, This is completely random, because the test and train data are shuffled whenever the dataset is being taken.

AUC_ROC and F1 score are both based on TP and TN, but AUC has some glitches when an imbalanced dataset is used. In practice, both the measures are used to compute the performance of the classifier. Some metrics are changed, and these classifiers are again tested.

- Different feature selection methods are used, and tested for AUC and F1_score. The results are obtained as

Choose 20 features	**Classifier**	**Test Accuracy**	**F1 Score**	**ROC_AUC Score**
Select K Best	**RF**	0.831	0.832	0.849
	MLP	0.716	0.743	0.836
Extra Tree Classifier	**RF**	0.801	0.80	0.834
	MLP	0.752	0.769	0.845
Logistic Regression	**RF**	0.762	0.801	0.84
	MLP	0.807	0.804	0.841

- PCA is done to extract the significant features. The number of features is chosen at random, and the results are obtained.

Classifier : RF				
Number of features	**Training Accuracy**	**Test Accuracy**	**F1 Score**	**ROC_AUC Score**
51	0.841	0.831	0.837	0.852
10	0.834	0.824	0.829	0.850
3	0.830	0.821	0.828	0.849

- The ROC_AUC score is computed using the metric **weighted** for average, since the dataset is imbalanced.
- The dataset is oversampled – using SMOTE .When implemented the AUC is found to improve.

6. DISCUSSION

The following observations are made.

- Data visualization techniques are useful in finding the relation between the different features. Histogram of the numerical data features gives an idea about normalizing the features. Box plots give the information about outliers. Hence based on these certain features were removed, and the dataset was preprocessed.

- The dataset is split into training set and test set , with test set being 20% of the entire dataset. The dataset is split with the parameter shuffle, and hence during every run of the program, the test set differs.
- The same set of preprocessing is applied to the test set and the training set, to avoid discrepancies.
- Preprocessing includes processing the dataset to improve features. The three steps in preprocessing that were done are Data Imputation, Data Normalization and Data encoding.
- Data imputation is the process of removing the unknown or missing values – either by dropping the values or filling with class wise mode of the values. It is found that imputing with class wise modes gives the better accuracy results – for all classifiers.
- In Data Normalization, MinMaxScaler and RobustScaler are used. Both the scalers scale the vale to the range [0,1]. MinMax Scaler is found to give less accuracy. RobustScaler gives more accuracy because the outliers are removed.
- Data Encoding is done – one hot encoding on the entire dataset is done, and for features with binary values, one column is removed because it is redundant column. One hot encoding gives the flexibility of analyzing each categorical feature, and finding the importance of each feature.
- The data is properly split into Test and training sets, with test set being 20% of the dataset. This ratio is maintained constant throughout.
- The classifiers are implemented – and the choice of optimal parameters obtained are computed through five – fold cross validation.
- On analyzing the results, it is found that Random Forest Classifier gives the best accuracy and F1 score. This changes for every iteration, since shuffle is enabled. RF gives higher output because it reduces overfitting. The AUC curve for RF is also found to be closer to the left border, and hence higher AUC.
- Selecting features doesn’t make any significant change here, since the F1 score and AUC are found to be almost the same for all feature selection methods.
- PCA gives best accuracy only if all features are selected. When the dimensions are reduced, the F1 Score and AUC decreases.

7. CONCLUSION

This project is on Python. The best classification model that could be applied on the bank – dataset is Random Forest. This is based on the obtained F1 Score and AUC Curves. It is essential that preprocessing has to be done properly to achieve higher accuracy, and F1 score. Since this is a binary classifier, AUC will be a best metric to evaluate the performance. The AUC is improved by Oversampling.

REFERENCES

[1] Lecture Notes of EE559

[2] http://mymljourney.com/classification-with-knn-and-cancer-diagnosis-example/

[3]https://www.researchgate.net/figure/A-hypothetical-example-of-Multilayer-Perceptron-Network_fig4_303875065

[4] http://www.statsoft.com/textbook/naive-bayes-classifier

[5] http://manufacturingscience.asmedigitalcollection.asme.org/article.aspx?articleid=2475086

Appendix A

Description of Features in bank – additional.csv

#	Feature	Description	Type	Values
1	Age	Age of the client	Numeric	18 to 88
2	Job	Type of job	Categorical	Admin,blue-collar,entrepreneur, housemaid, management, retired, self-employed, services,student, technician, Unemployed, Unknown
3	Marital	Marital Status	Categorical	Divorced, Married, Single,unknown
4	Education	Level of education	Categorical	basic.4y,basic.6y, basic.9y,high.school, illiterate, professional.course, university.degree, unknown
5	Default	Credit in default?	Categorical	Yes, No, Unknown
6	Housing	Has housing Loan?	Categorical	Yes, No, Unknown
7	Loan	Has Personal Loan?	Categorical	Yes, No, Unknown
8	Contact	Type of communication	Categorical	Cellular, Telephone
9	Month	Month last contacted	Categorical	Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
10	Day	Day last contacted	Categorical	Mon, Tue, Wed, Thu, Fri
11	Campaign	number of contacts performed during this campaign for this client	Numeric	1 to 35
12	pdays	number of days that passed by after the client was last contacted from a previous campaign	Numeric	0 to 21, 999 means clients was not previously contacted
13	previous	number of contacts performed before this campaign for this client	Numeric	0 to 6
14	poutcome	outcome of the previous marketing campaign	Categorical	Failure, Nonexistent, Success
15	emp.var.rate	employment variation rate - quarterly indicator	Numeric	-3.4 to 1.4

16	cons.price.idx	consumer price index - monthly	Numeric	92.201 to 94.767
17	cons.conf.idx	consumer confidence index - monthly indicator	Numeric	-50.8 to -26.9
18	euribor3m	Euribor 3-month rate	Numeric	0.635 to 5.045
19	nr. employed	Number of employee – quarterly indicator	Numeric	4963.6 to 5228.1
20	y	Client has subscribed a term deposit?	Binary	Yes, No