INDEX

ABSTRACT

I have attempted this project individually and have considered the second dataset given – Bank Marketing Data Set. This project is considered to be an open ended one and I was given the opportunity to try many different things in the entire process. This problem statement is a Binary Classification problem.

To give a brief description of the dataset, it is related direct marketing campaign of a bank institution from Portuguese. Marketing campaign is nothing but phone calls to the clients to make them accept to make a term deposit with their bank. So, after a call, it is binary classified, as to no-being they client did not make a deposit and yes-being the client on call accepted to make a deposit.

The purpose of this project is to predict if the client on call would accept to make a term deposit or not based on the information of the clients. The Bank Marketing Data Set given for this project is a small portion (10%) of the entire available data set. The data set has about 4119 rows of data with 19 features and 1 column of Class information.

The main issues of the dataset are:

- Preprocessing required to fill unknown values in the dataset
- Preprocessing required to decide on usage of categorical data along with continuous data
- The data is class imbalanced (Number of class 1 (yes) is very low when compared to the number of Class 0 (no))

The remaining part of the project is organized into various sections explaining the details of experimentation and implementation:

- Understanding of the features
- Preprocessing of the features
- Different Classifiers Used
    - K-Nearest Neighbor Classifier
    - Logistic Regression
    - Naïve Bayes
    - SVM
    - Perceptrons
    - Random Forest Classifier
- Dimensionality Reduction
- Discussion on Results

Various Intuitions and Interpretations of Results are given in the last section. I thoroughly enjoyed doing this project and a great level of understanding of various Machine Learning Algorithm was obtained.

BANK MARKETING DATA SET - DESCRIPTION OF DATA SET

The Dataset I considered, the Bank Marketing Data Set is one of the famous data sets available from UCI Machine Learning Repository. As I already described in the abstract section, this dataset has features about the clients to whom calls were made. The class information is binary being yes or no for a term deposit with the bank. With this data availability, the aim of this project is to predict the yes/no (i.e) if the client would agree for a term deposit with the bank or not.

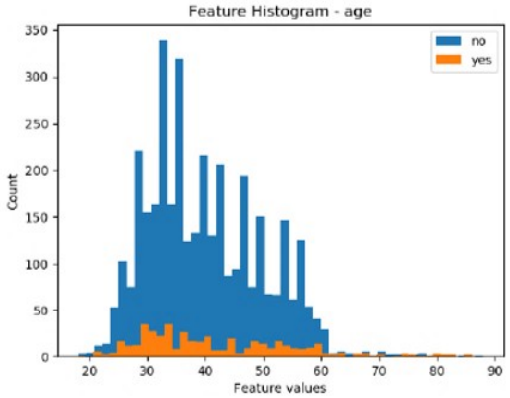
The features information is given in a table below:

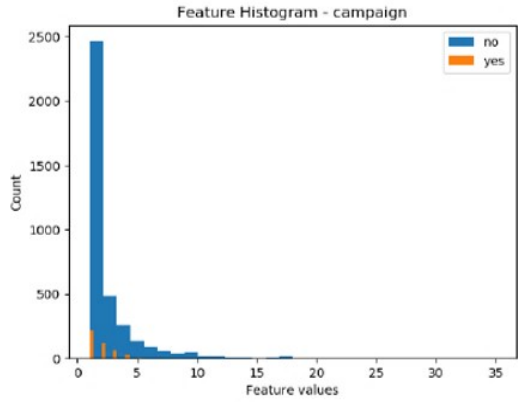
No.	Feature Name	Feature Type	Feature Description
1	Age	Numeric, Bank Client Data	Age of the client
2	Job	Categorical, Bank Client Data	Type of job the client is
3	Marital	Categorical, Bank Client Data	Marital Status of the client
4	Education	Categorical, Bank Client Data	Education level of the client
5	Default	Categorical, Bank Client Data	Has Credit in Default or not
6	Housing	Categorical, Bank Client Data	If the client has a Housing Loan or not
7	Loan	Categorical, Bank Client Data	If the client has a Personal Loan or not
8	Contact	Categorical, Related to Last Contact	Mode of contact last time
9	Month	Categorical, Related to Last Contact	Month when last contact was made
10	Day of Week	Categorical, Related to Last Contact	Day of week last contact was made
11	Campaign	Numeric, Other Attributes	Number of contacts performed during this campaign with a particular client
12	Pdays	Numeric, Other Attributes	Number of days passed by after the client was contacted from a previous campaign
13	Previous	Numeric, Other Attributes	Number of contacts performed performed before this campaign for a particular client
14	Poutcome	Categorical, Other Attributes	Outcome of the precious marketing campaign
15	Emp.var.rate	Numeric, Socio Economic Attributes	Employment Variation Rate – quarterly indicator
16	Cons.price.idx	Numeric, Socio Economic Attributes	Consumer Price Index– monthly indicator
17	Cons.conf.idex	Numeric, Socio Economic Attributes	Consumer Confidence Index – monthly indicator
18	Euribor3m	Numeric, Socio Economic Attributes	Euribor 2-month rate – daily indicator
19	Nr.employed	Numeric, Socio Economic Attributes	Number of employees – quarterly indicator

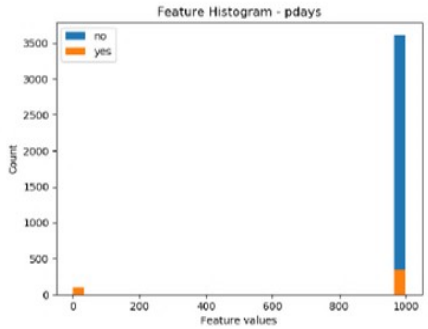
UNDERSTANDING THE FEATURES GIVEN

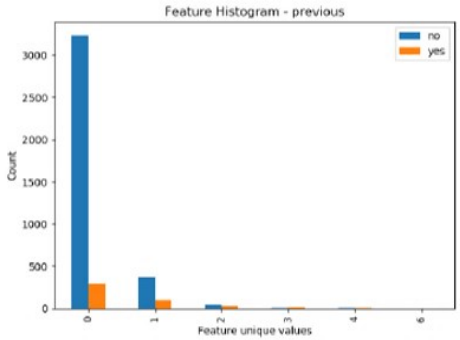
This was the first step I took to proceed the project. I plotted the histograms of each feature with respect to each class and overlaid the plot. I started it with a notion that it would help me understand which feature highly separates the class in terms of distribution. The plots for different dataset types are given below.

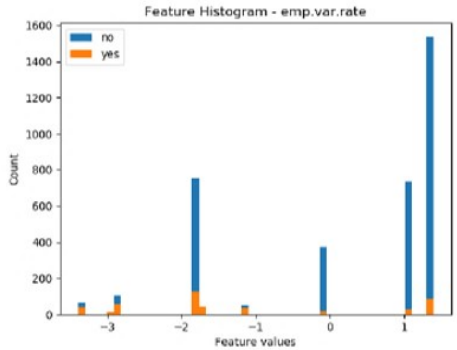
Numerical Features:

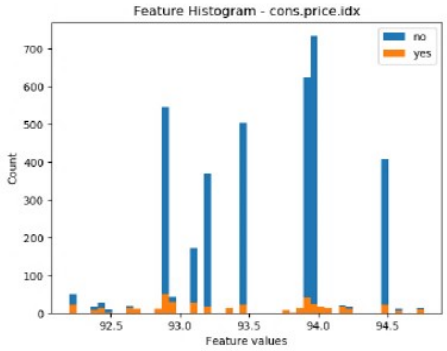


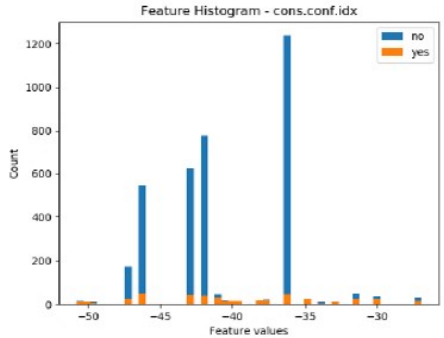


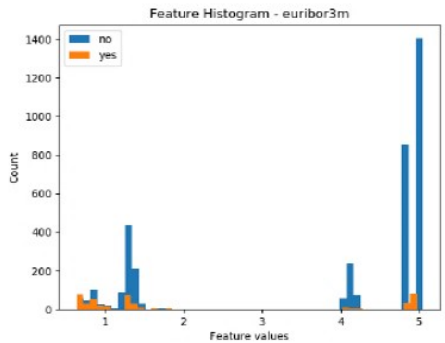

Feature Histogram - pdays
no
yes
Count
Feature values

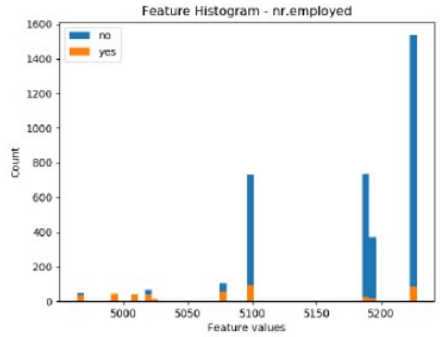

Feature Histogram - previous
no
yes
Count
Feature unique values


Feature Histogram - emp.var.rate
no
yes
Count
Feature values


Feature Histogram - cons.price.idx
no
yes
Count
Feature values

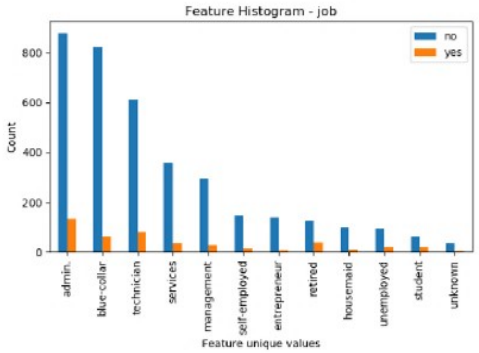

Feature Histogram - cons.conf.idx
no
yes
Count
Feature values

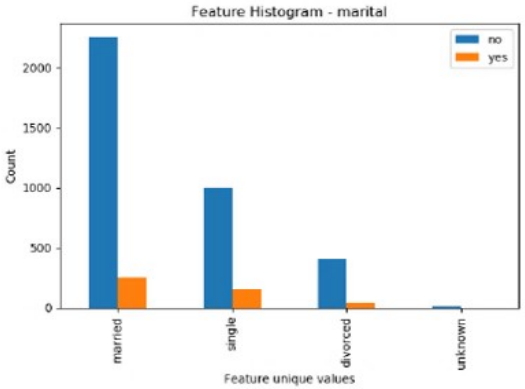

Feature Histogram - euribor3m
no
yes
Count
Feature values

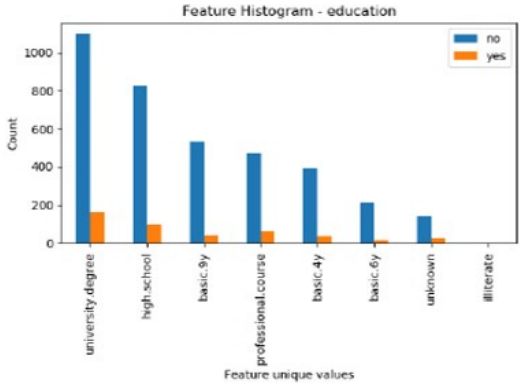


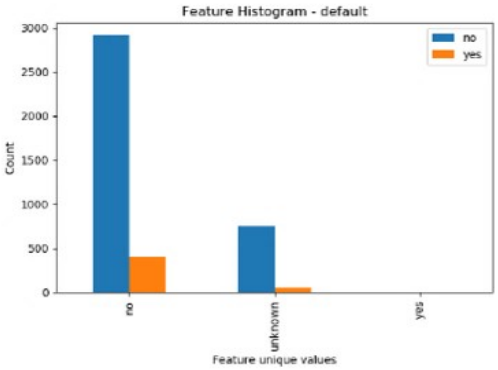
- From the plots of the continuous features shown above, these features have similar distribution in terms of classes in the data set.
- None of the distributions are clearly separated from another and it's hard to determine if a classifier will do good or not.
- Though all these features are continuous, the range of values for among the features vary the most.
- One feature has values in the range of 5000s, one feature has values in the range of 90s, some features lie in a number less than 10, and one feature has negative values.
- This varying range might affect the performance of the model. Hence, I have adopted two different strategies to make them on the same range of values.
    - Normalization
    - Standardization

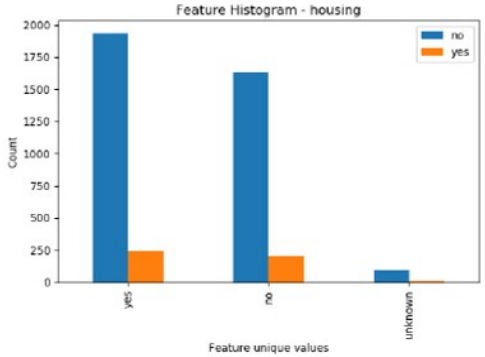
Categorical Features:

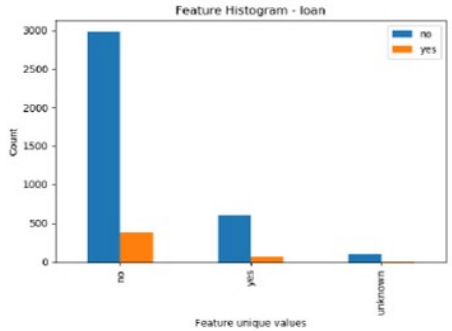


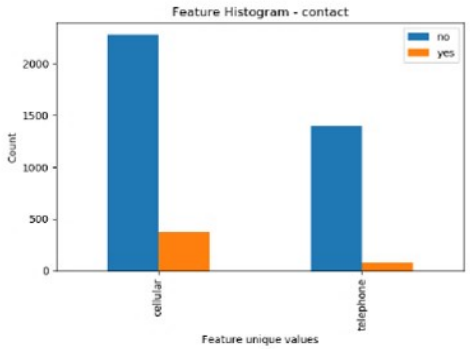


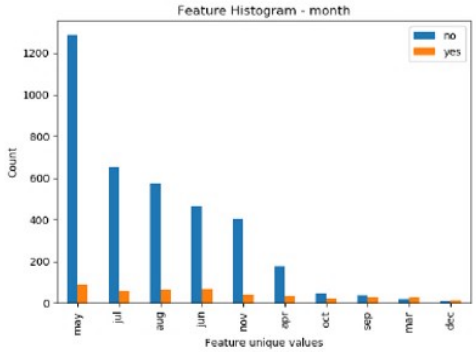


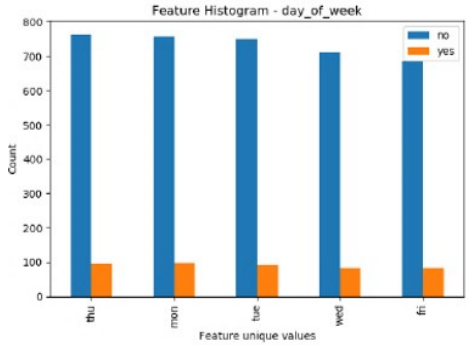


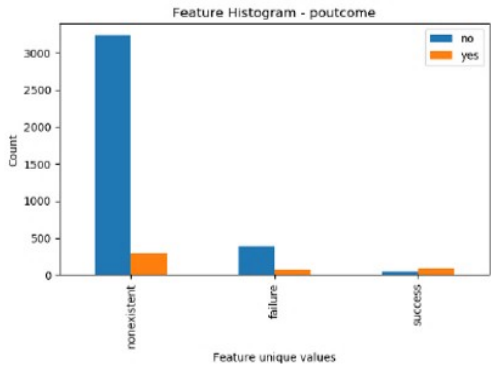

Feature Histogram - housing
no
yes
Count
2000
1750
1500
1250
1000
750
500
250
0
yes
no
unknown
Feature unique values


Feature Histogram - loan
no
yes
Count
3000
2500
2000
1500
1000
500
0
no
yes
unknown
Feature unique values


Feature Histogram - contact
no
yes
Count
2000
1500
1000
500
0
cellular
telephone
Feature unique values


Feature Histogram - month
no
yes
Count
1200
1000
800
600
400
200
0
may
jul
aug
jun
nov
apr
oct
sep
mar
dec
Feature unique values


Feature Histogram - day_of_week
no
yes
Count
800
700
600
500
400
300
200
100
0
thu
mon
tue
wed
fri
Feature unique values


Feature Histogram - poutcome
no
yes
Count
3000
2500
2000
1500
1000
500
0
nonexistent
failure
success
Feature unique values

Some key observations from the Categorical Dataset are:

- Categorical data cannot be used directly into a classifier. They need to be converted to meaningful numerical values for them to be used in a classifier
- There are two methods for converting the Categorical to Numerical Values. I have used both of them depending upon the understanding of the features. They are:
  - Labeling the unique values into some numbers. But these numbers might implicitly tell the classifier that there is some ordering among the unique feature values. Hence, I have used this only for features with some order in values makes sense
  - One hot encoding – This transform unique feature values to column labels. Then they are one hot encoded for every row of the data into 0 and 1 so as to see if that unique feature value is present or not
- Many features have unknown values in the dataset. I have used two methods to fill those unknown values. They are:
  - Mode Impute
  - SVM Impute

PREPROCESSING DATA

Many Preprocessing steps were tried to proceed before Classification. They are described below. All the preprocessing steps were done after splitting the given data into train and test data on a percentage of 75% and 25%. I used *sklearn.model_selection.train_test_split* for this process. The split was done before preprocessing since all the methods need to be fit based on training features data. Based on the parameters calculated from training data, both training and testing features need to be transformed.

Standardization:

Standardization of the dataset is transforming the features to 0 mean and 1 standard deviation. I used *sklearn.preprocessing.StandardScaler* for this process. This calculates the mean and standard deviation of every feature column and uses them in the below formula to transform the image. The function was fit using the training data features. Transformation was done on training and testing features based on min and max values calculated from training features.

$$Z = (X - \text{mean}) / \text{standard\_deviation}$$

Continuous Features (Train Set)	Mean	Variance
Age	4.01447070e+01	1.05954133e+02
Campaign	2.49627711e+00	6.08812146e+00
Previous	1.84202007e-01	2.73288785e-01
Emp.var.rate	9.04176109e-02	2.43858089e+00
Cons.price.idx	9.35811800e+01	3.36317495e-01
Cons.conf.idx	-4.05099385e+01	2.08090401e+01
Euribor3m	3.63210456e+00	2.98574344e+00
Nr.employed	5.16705199e+03	5.37721405e+03

Normalization:

Normalization of the dataset is Scaling the features between 0 and 1. This was tried only on the continuous features of the dataset. I used *sklearn.preprocessing.MinMaxScaler* for this process. This function calculates the min and max from the dataset features and scales them between 0 and 1 using the below formula. The function was fit using the training data features. Transformation was done on training and testing features based on min and max values calculated from training features.

$$Z = (X - \text{min\_value}) / (\text{max\_value} - \text{min\_value})$$

Continuous Features (Train Set)	Min Value	Max Value
Age	18.0	88.0
Campaign	1.0	35.0
Previous	0.0	6.0
Emp.var.rate	-3.4	1.400
Cons.price.idx	92.22	94.767
Cons.conf.idx	-50.08	-26.900
Euribor3m	0.635	5.0450
Nr.employed	4963.6	5228.1

Label Encoding of Categorical Features:

Some of the Categorical Features present in the data seems to be sensible when label encoding is done. The features and the encoded labels are given below. These features have some inherent order in them, and hence I have given labels with some sensible order. Though there is some order in the labels, even if the classifier understands that to be an ordered variable, this is not considered as min interpretation and would not decrease the efficiency of performance.

I haven't used any inbuilt function for this process. I have created a dictionary for every feature with categorical feature values as the keys and the labels as the encoded values. They are given below. Both training and testing features were transformed using these labels.

Categorical Features	Label Encoding - Hardcoded
Education	{'illiterate':0, 'basic.4y':4, 'basic.6y':6, 'basic.9y':9, 'high.school':11, 'professional.course':13, 'university.degree':14}
Housing	{'no':0,'yes':1}
Loan	{'no':0,'yes':1}
Contact	{'telephone':0,'cellular':1}
Month	{'jan':1,'feb':2,'mar':3,'apr':4,'may':5,'jun':6,'jul':7,'aug':8,'sep':9,'oct':10,'nov':11, 'dec':12}
Day of Week	{'mon':1,'tue':2,'wed':3,'thu':4,'fri':5,'sat':6,'sun':7}
Poutcome	{'nonexistent':0,'failure':1,'success':2}

One-Hot Encoding:

One-Hot Encoding is used in machine learning to transform categorical features to numerical feature values, so that they can be used by the classifier. Allowing the model to assume there is some ordinal relationship in the feature values might results in poor performance of the Classifier. This is where One-Hot Encoding is useful since this is mainly implemented on the features which inherently do not have any ordinal relationship. I used *sklearn.preprocessing.OneHotEncoder* for this process. I fit using the training features data. Transformation was done on both training and testing features based on train features fitting. I used One-hot Encoding for two features in the data set

- Job
- Marital

An example of One-Hot Encoding is shown below:

	Job	**Marital**
Sample 1	services	married
Sample 2	technician	single
Sample 3	housemaid	divorced

The above table would transform to below when One-Hot Encoding is done. The columns are the unique feature values present in that feature column.

Unique Values from 'Job' Feature:

'blue-collar', 'services', 'admin.', 'entrepreneur', 'self-employed', 'technician', 'management', 'student', 'retired', 'housemaid', 'unemployed'

Unique Values from 'Marital' Feature:

'married', 'single', 'divorced'

	BC	Serv	Adm	Entr	SE	Tech	Mgm	Stud	Ret	HM	UE	Marr	Sing	Div
S1	0	1	0	0	0	0	0	0	0	0	0	1	0	0
S2	0	0	0	0	0	1	0	0	0	0	0	0	1	0
S3	0	0	0	0	0	0	0	0	0	1	0	0	0	1

I have shown an example of how One-Hot Encoding was performed. The column names are given in short forms to save space so that I can represent all the transformed features in one single table and is easy to view.

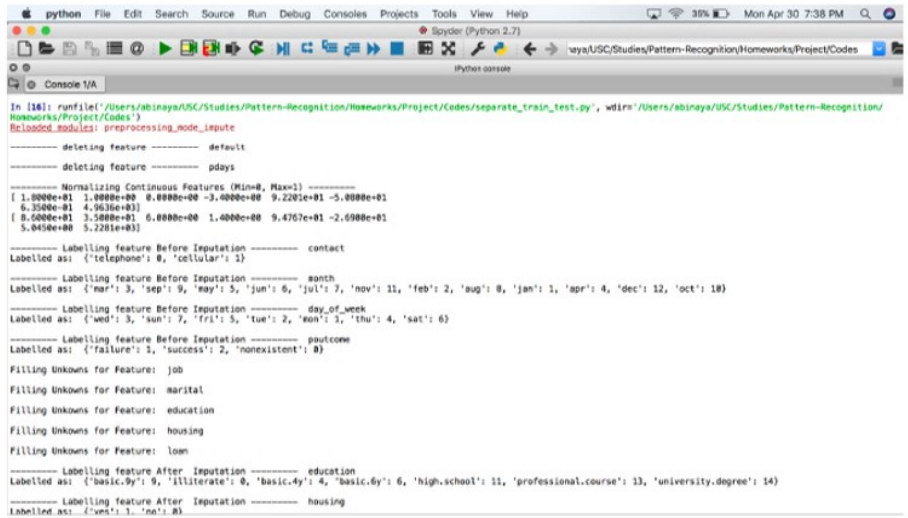
Some Notes:

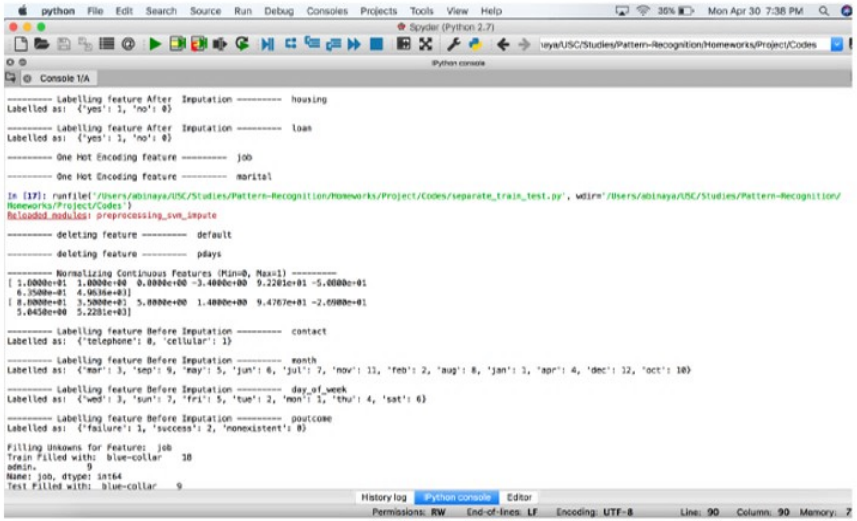
- Unique feature values would be found, and they would make the new columns
- The rows would be filled with binary values of 0 or 1 based on if that feature value is present or not for that row
- Since I could not see Job and Marital Feature with any ordinal relationship I had used One-Hot Encoding for those two features only
- All the other categorical features were transformed to numerical values based on the label encoding that was discussed above.

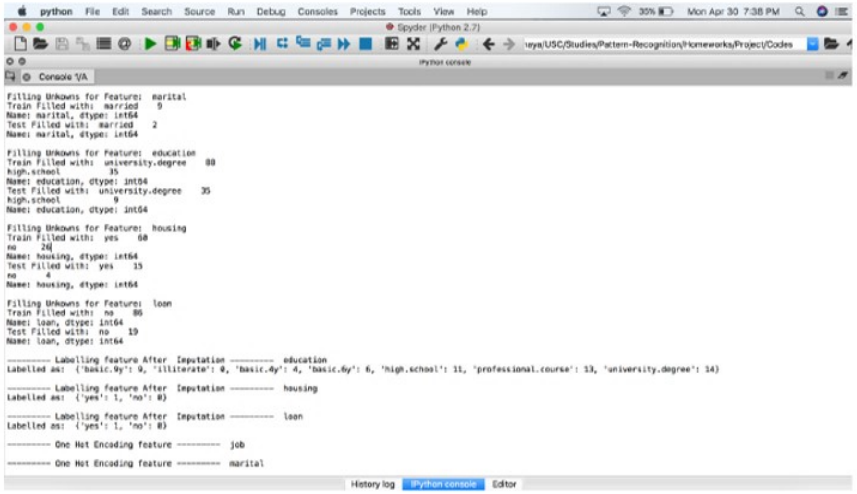
Discarding Features:

- After understanding of the features using the histogram techniques, I can see that two of the features were of no use for classification. They are:
  - Default
  - Pdays
- Both the features had only one feature value and all the other were unknowns.
- Even if we use Mode Impute or SVM Impute to fill the unknown values, all of them will get filled with that one feature value present.
- This would not give any useful information to the Classifier since all the values of the feature would be same.
- So, I conducted experiments of Classification with both including and not including those features.

Some Screenshots of the Preprocessing:

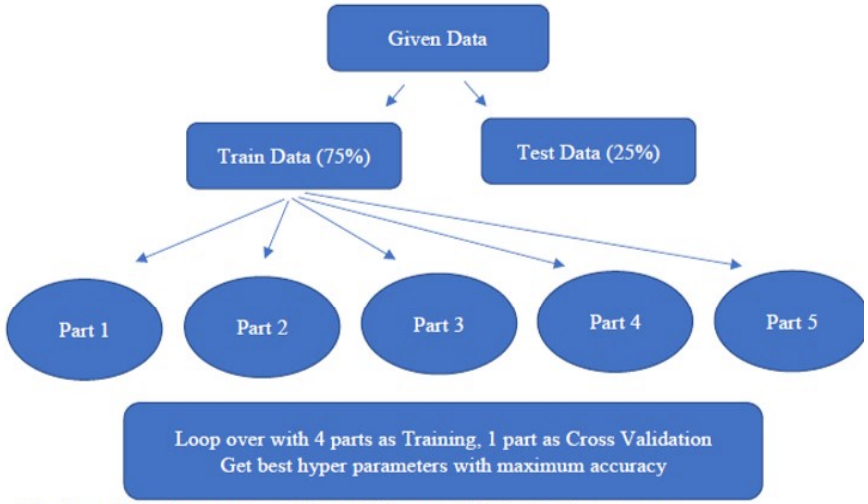






CROSS VALIDATION FOR TRAINING

- The upcoming sections, I have explained about each classifier I have used and have given classification results with some metrics in the upcoming sessions
- For some of the classifiers, where there were hyper parameters need to be tuned, I used cross validation for training the model and selecting the best hyper parameters
- This also avoided over fitting the training data which is considered to be a major problem
- I have given a flow chart explanation of how Cross Validation was done in the below mentioned sections of Classification



- The above flow chart represents Five-fold Cross validation Implementation
- Trian and Test data is split with ratio 3:1 using *sklearn.model_selection.train_test_split*
- The Train data is split into 5 equal parts using *sklearn.model_selection.StratifiedKFold*
    - Stratified K Fold is done to make sure the ratio of both the class is same in all the sub parts of the data
    - A for loop is iterated making 4 equal parts as sub training data and 1 part as cross validation
    - Initialization of new model and training is done on the combined 4 parts data
    - Cross Validation is nothing but testing data within the for loop
    - The validation accuracy is got as the average of the 5 loops run
    - This is repeated for every hyper parameter combination
    - An outer loop of maximum iterations = 20 is performed and averaged accuracies
- The best hyper parameter is chosen by considering the one which had maximum accuracy from the 5-fold stratified cross validation
- Using the best parameter, the entire a final model using the entire train data
- The model is tested both on entire training data and testing data to validate the model

PERCEPTRON CLASSIFIER
(TRAINED BY STOCHASTIC GRADIENT DESCENT)

THEORY:

- Perceptron is a classic Linear Learning algorithm.
- When I describe an algorithm to be linear, it means the degree of variables (i.e., the features) are always equal to one and do not increase while fitting the model
- The weights and intercept of the separator is learned by Stochastic Gradient Descent Learning Algorithm
- One very common Perceptron Training Algorithm using Basic Sequential Gradient Descent is given below:
  - Randomly shuffle the order of training data before starting the learning process
  - Initialize initial weight vector w(0)
  - Loop over max number of epochs:
    - For n in {1,2,… #no samples}
      - i = n-1
      - Update weights such that:
        w(i+1) = w(i) + learning_rate(i) * reflected data point(n)
        if present iteration data point is misclassified
      - Do not update weights if the present iteration data point is classified correctly
- Iteration can be continued until there is no change in weights over one full epoch.
- There are even other variants of Gradient Descent used to train the Perceptron Classifier. They are Batch Gradient Descent, Mini Batch Gradient Descent, Stochastic Gradient Descent etc.
- I used *sklearn.linear_model.Perceptron* for this Perceptron Implementation process.
- To address the imbalance problem, I have used 'class_weights' parameter while fitting the model in sklearn. This takes care of the imbalanced dataset.
- I have got certain evaluation metrics related to imbalanced data like
  - Confusion Matrix
  - Precision
  - Recall
  - Class 0 F1 Score
  - Class 1 F1 Score
  - F1 Score
  - Weighted F1 Score
  - ROC Curve
  - AUC value
- These metrics are specifically used for Imbalanced dataset.

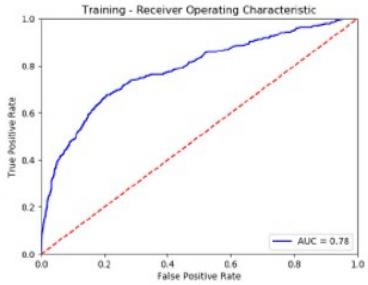
CROSS VALIDATION FOR CHOOSING BEST:

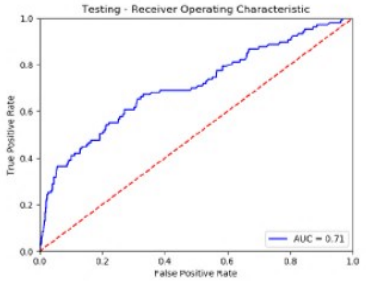
- Penalty – l1 or l2 or 'elasticnet'
- C – Inverse of Regularization Strength – 0.00001 to 1

RESULTS:

- Mode-Standardize: Chosen Penalty = elasticnet, alpha = 1e-05

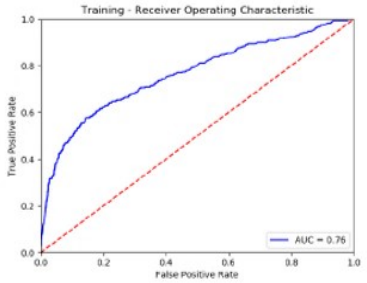
Training Confusion Matrix	Testing Confusion Matrix
[[2704 41] [282 62]]	[[916 7] [100 7]]

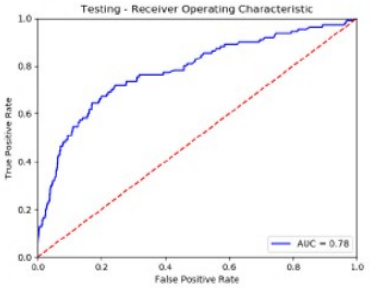




- Mode-Normalize: Chosen Penalty = l2, alpha = 1e-05

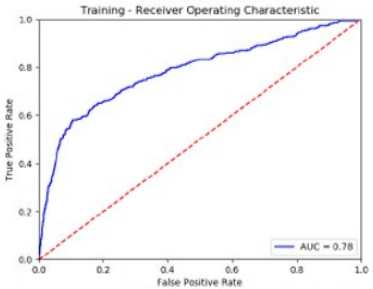
Training Confusion Matrix	Testing Confusion Matrix
[[2723 25] [298 43]	[[915 5] [97 13]]

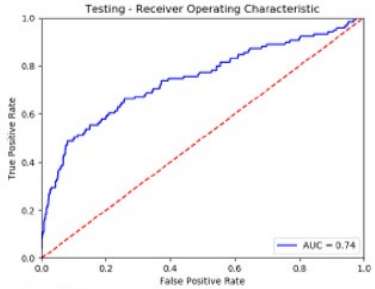




- SVM-Standardize: Chosen Penalty = l1, alpha = 1e-05

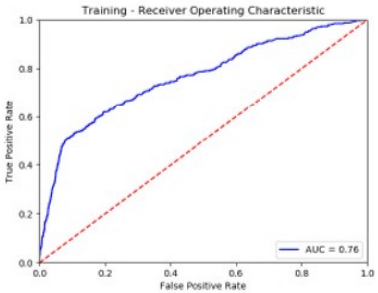
Training Confusion Matrix	Testing Confusion Matrix
[[2740 17] [305 27]]	[[907 4] [107 12]]

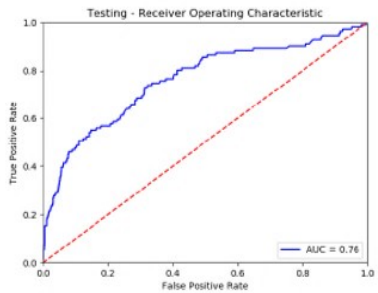




- SVM-Normalize: Chosen Penalty = l1, alpha = 1e-05

Training Confusion Matrix	Testing Confusion Matrix
[[2715 34] [298 42]]	[[914 5] [96 15]]



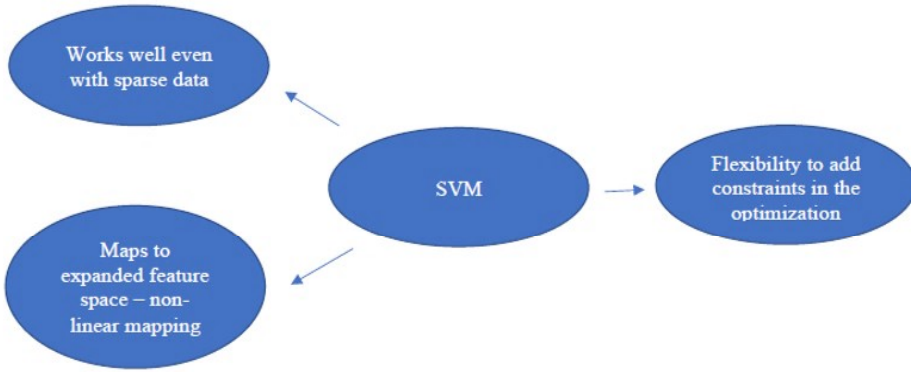


	Mode-Standardize		Mode-Normalize		SVM-Standardize		SVM-Normalize	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Accuracy	0.8954	0.8961	0.8954	0.9009	0.8957	0.89223	0.8925	0.9019
Precision	0.6019	0.5	0.6323	0.7222	0.6136	0.75	0.5526	0.75
Recall	0.18023	0.06542	0.1260	0.1181	0.0813	0.10084	0.1235	0.1351
Class 0 F1	0.94	0.94	0.94	0.95	0.94	0.94	0.94	0.95
Class 1 F1	0.28	0.12	0.21	0.20	0.14	0.18	0.20	0.23
F1 Score	0.2774	0.1157	0.2102	0.2031	0.14361	0.17777	0.2019	0.2290
Weighted F1 Score	0.8694	0.8586	0.8630	0.8677	0.85842	0.85400	0.8608	0.8701
AUC	0.7838	0.7058	0.7622	0.78033	0.7834	0.74387	0.7634	0.7597

SUPPORT VECTOR MACHINES

THEORY:

- Support Vector Machines are mainly used in Classification purposes.
- It finds the right hyper plane with margin to separate both the classes in n-feature dimension. It also has the ability to take the features to a different dimension and fit non-linear models
- This flexibility of linear and non-linear models in different dimensions is what making SVM popular in recent Machine Learning
- The Kernels are defined in such a way that it brings out the similarity between the samples of the same class. Most popularly used kernel is Gaussian (RBF)



- I used *sklearn.svm_model.SVC* for this SVM Implementation process.
- To address the imbalance problem, I have used 'class_weights' parameter while fitting the model in sklearn. This takes care of the imbalanced dataset.
- I have got certain evaluation metrics related to imbalanced data like
  - Confusion Matrix
  - Precision and Recall
  - Class 0 F1 Score and Class 1 F1 Score
  - F1 Score
  - Weighted F1 Score
  - ROC Curve
  - AUC value
- These metrics are specifically used for Imbalanced dataset.

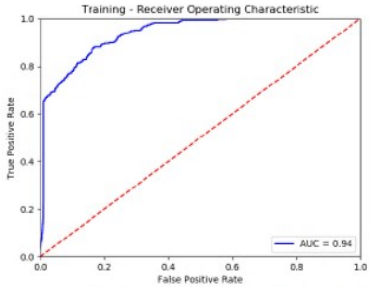
CROSS VALIDATION FOR CHOOSING BEST: (RBF kernel was used)

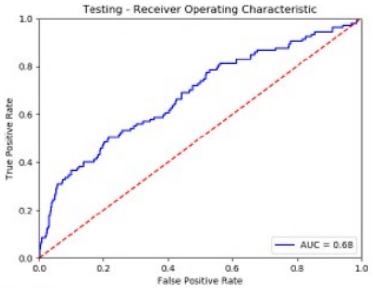
- Penalty Parameter C – 0.5 to 2
- Gamma - Kernel Coefficient – 0.01 to 0.05

RESULTS:

- Mode-Standardize: C = 2.2, Gamma = 0.03

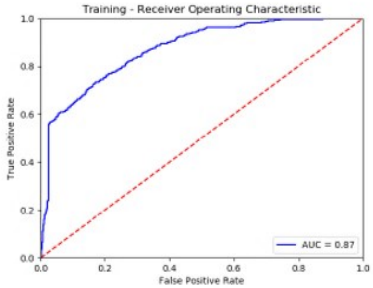
Training Confusion Matrix	Testing Confusion Matrix
[[2492 253] [79 265]	[[815 108] [67 40]]

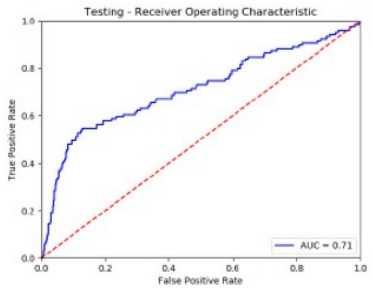




- Mode-Normalize: C = 1.9, Gamma = 0.05

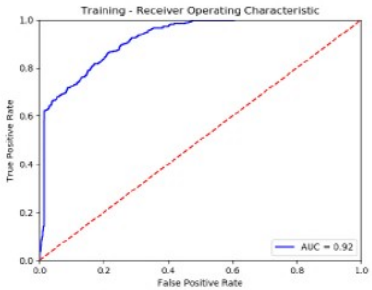
Training Confusion Matrix	Testing Confusion Matrix
[[2366 382] [105 236]]	[[776 144] [42 68]]

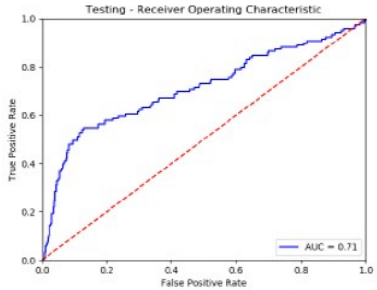




- SVM-Standardize: C = 1.5 , Gamma = 0.04

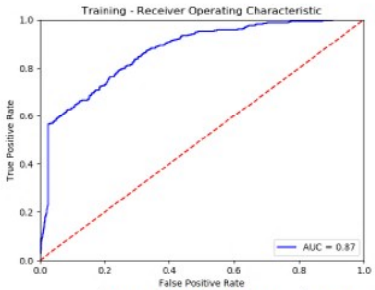
Training Confusion Matrix	Testing Confusion Matrix
[[2439 318] [88 244]]	[[791 120] [54 65]]

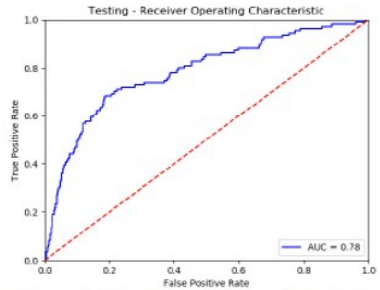




- SVM-Normalize: C = 2, Gamma = 0.05

Training Confusion Matrix	Testing Confusion Matrix
[[2362 387] [113 227]]	[[763 156] [42 69]]





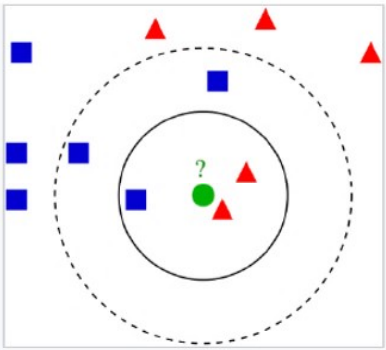
	Mode-Standardize		Mode-Normalize		SVM-Standardize		SVM-Normalize	
	Train	Test	Train	Test	Train	Test	Train	Test
Accuracy	0.8925	0.8300	0.8423	0.8194	0.8685	0.8310	0.838	0.8077
Precision	0.5115	0.2702	0.3818	0.3207	0.4341	0.3513	0.3697	0.3066
Recall	0.7703	0.3738	0.6920	0.6181	0.7349	0.5462	0.6676	0.6216
Class 0 F1	0.94	0.61	0.91	0.89	0.92	0.90	0.90	0.89
Class 1 F1	0.90	0.31	0.49	0.42	0.55	0.43	0.48	0.41
F1 Score	0.6148	0.3137	0.4921	0.4223	0.5458	0.4276	0.47589	0.41071
Weighted F1 Score	0.9016	0.8418	0.8609	0.8427	0.8826	0.8462	0.85713	0.83402
AUC	0.93775	0.6818	0.87300	0.7653	0.9201	0.7107	0.8679	0.7825

K-NEAREST NEIGBOR CLASSIFIER

THEORY:

- K-Nearest Neighbor (or KNN) is the simplest classifier which classifies the data based on the k – nearest neighbor's class information
- There is not much of training in this method. For each test data, the k – neighbors are identified based on some distance metric
- The test data is assigned the class which has the highest frequency from the neighbors
- An example from Wikipedia page is shown below
  - The green circle is the test data
  - K = 3 in this example
  - Out of the 3 neighbors, 2 are Red-Triangle Class, and 1 is Blue-Square class
  - KNN algorithm will assign Red-Triangle Class to the test data point

(Source: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)



- I used *sklearn.neighbors KNeighborsClassifier* for this KNN Implementation process.
- I have got certain evaluation metrics related to imbalanced data like
  - Confusion Matrix
  - Precision
  - Recall
  - Class 0 F1 Score and Class 1 F1 Score
  - F1 Score
  - Weighted F1 Score
- These metrics are specifically used for Imbalanced dataset.

CROSS VALIDATION FOR CHOOSING BEST:

- K from range of 3 to 15

RESULTS:

- Mode-Standardize: Chosen K = 5

Training Confusion Matrix	Testing Confusion Matrix
[[2708 37] [233 111]]	[[900 23] [90 17]]

- Mode-Normalize: Chosen K = 14

Training Confusion Matrix	Testing Confusion Matrix
[[2739 9] [288 53]]	[[909 11] [103 7]]

- SVM-Standardize: Chosen K = 13

Training Confusion Matrix	Testing Confusion Matrix
[[2733 24] [272 60]]	[[904 7] [102 17]]

- SVM-Normalize: Chosen K = 6

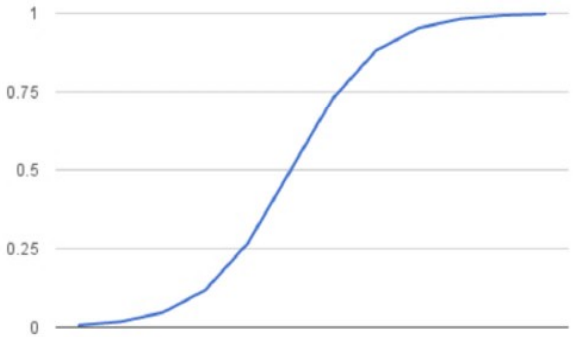
Training Confusion Matrix	Testing Confusion Matrix
[[2737 12] [320 20]]	[[914 5] [108 3]]

	Mode-Standardize		Mode-Normalize		SVM-Standardize		SVM-Normalize	
	Train	Test	Train	Test	Train	Test	Train	Test
Accuracy	0.9125	0.8902	0.9038	0.8893	0.904	0.894	0.8925	0.89029
Precision	0.75	0.425	0.8548	0.4	0.714	0.708	0.625	0.375
Recall	0.3226	0.1588	0.1554	0.0636	0.180	0.142	0.0588	0.0270
Class 0 F1	0.95	0.94	0.95	0.94	0.95	0.94	0.94	0.94
Class 1 F1	0.45	0.23	0.26	0.11	0.29	0.24	0.11	0.05
F1 Score	0.4512	0.2312	0.2630	0.1093	0.288	0.237	0.1075	0.0504
Weighted F1 Score	0.8966	0.8672	0.8728	0.8521	0.877	0.861	0.8508	0.8457

LOGISTIC REGRESSION

THEORY:

- Logistic Regression is a type of Linear Classifier. The input values are combined linearly using weights or coefficient values.
- One should not confuse Logistic Regression with Linear Regression, where Linear Regression regresses the output is a numeric value. But in Logistic Regression the outputs are either 0 or 1
- The nature of the Logistic Regression finds its way to be a binary classifier.
- The outputs are the Sigmoid function used in Logistic Regression is between 0 and 1. This output can be considered as a probability output.
- The probability represents the probability of Class 1 given the input X
- Thus, a threshold can be chosen to be 0.5 such that if the probabilities are greater than 0.5, the class output is 1 and if the probabilities are lesser than 0.5, the class output is 0
- The sigmoid function is. 1 / (1 + e ^ -value). The input X values are mapped to Y value between 0 and 1. The graph of sigmoid function is shown below:



- I used *sklearn.linear_model.LogisticRegression* for this Logistic Regression Implementation process.
- To address the imbalance problem, I have used 'class_weights' parameter while fitting the model in sklearn. This takes care of the imbalanced dataset.
- I have got certain evaluation metrics related to imbalanced data like
  - Confusion Matrix, Precision and Recall
  - Class 0 F1 Score, Class 1 F1 Score, F1 Score and Weighted F1 Score
  - ROC Curve and AUC value
- These metrics are specifically used for Imbalanced dataset.

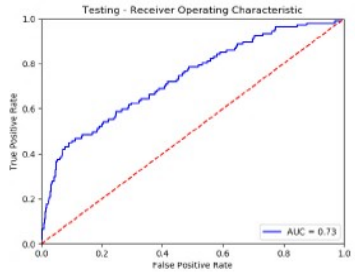
CROSS VALIDATION FOR CHOOSING BEST:

- Penalty – l1 or l2
- C – Inverse of Regularization Strength – 0 to 1

RESULTS:

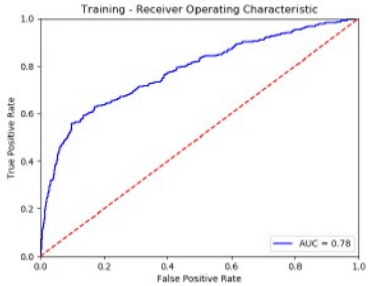
- Mode-Standardize: Chosen Penalty = l1, C = 1.0

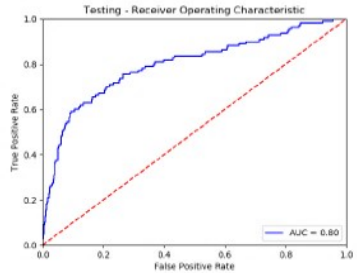
Training Confusion Matrix	Testing Confusion Matrix
[[2249 496] [115 229]]	[[758 165] [52 55]]



- Mode-Normalize: Chosen Penalty = l1, C = 0.9

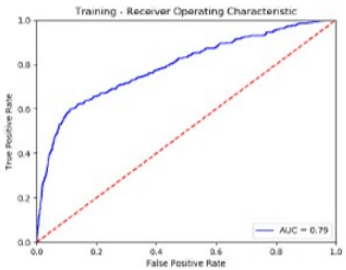
Training Confusion Matrix	Testing Confusion Matrix
[[2172 576] [123 218]]	[[739 181] [34 76]]

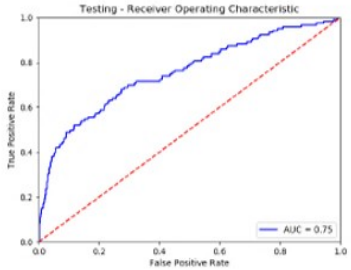




- SVM-Standardize: Chosen Penalty = l1, C = 0.8

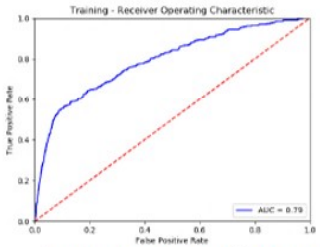
Training Confusion Matrix	Testing Confusion Matrix
[[2236 521] [116 216]]	[[729 182] [50 69]]

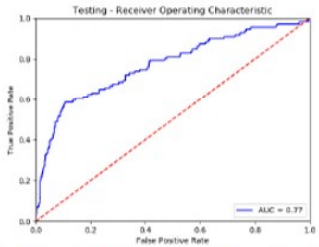




- SVM-Normalize: Chosen Penalty = l1, C = 1.0

Training Confusion Matrix	Testing Confusion Matrix
[[2259 490] [131 209]]]	[[727 192] [41 70]]





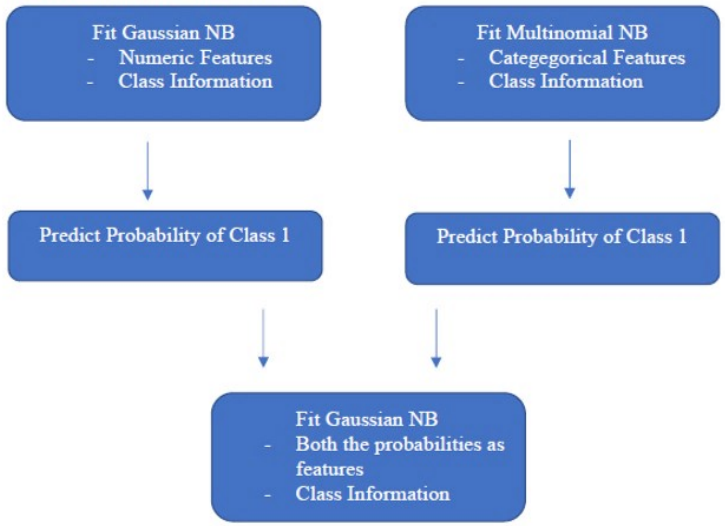
	Mode-Standardize		Mode-Normalize		SVM-Standardize		SVM-Normalize	
	Train	Test	Train	Test	Train	Test	Train	Test
Accuracy	0.802	0.789	0.7737	0.7912	0.7937	0.7747	0.79896	0.773
Precision	0.315	0.25	0.2745	0.29571	0.29308	0.2749	0.2989	0.2671
Recall	0.665	0.5140	0.6392	0.6909	0.6506	0.5798	0.6147	0.6306
Class 0 F1	0.88	0.87	0.86	0.87	0.88	0.86	0.88	0.86
Class 1 F1	0.43	0.34	0.38	0.41	0.40	0.37	0.40	0.38
F1 Score	0.4284	0.3363	0.3841	0.4141	0.40411	0.3729	0.4023	0.3753
Weighted F1 Score	0.8300	0.8188	0.8087	0.8240	0.8246	0.8061	0.8266	0.8094
AUC	0.7999	0.734	0.777	0.7971	0.79092	0.7535	0.7906	0.7717

NAÏVE BAYES CLASSIFIER

THEORY:

- Naïve Bayes comes under the category of Generative Classifier. It is based on the famously know Bayes theorm:
  - Posterior = (Likelihood * Prior) / Probability of Data
    - Where, Posterior is Probability of class given data
    - Likelihood is Probability of data given class
    - Prior is probability of class
- It learns the probability of the joint distribution of the data and Class instead of just the conditional probability of class given features
- The main advantage of the learning of joint distribution is that, we can generate samples which follows the distribution. That is why it is termed as Generative Classifier
- Also, for simplicity, the features are assumed to be independent conditioned on class information. This independence property gives us the ability to multiply the individual conditional probabilities of data given the class. This is the calculation of Likelihood. Because of this property, it is called as "Naïve" Bayes.
- I used *sklearn.naive_bayes.GaussianNB and sklearn.naive_bayes.MultinomialNB* for this Naïve Bayes Implementation process.
- Since the data is a combination of Numeric and Categorical Values, I used the below given method to implement Naïve Bayes in the data

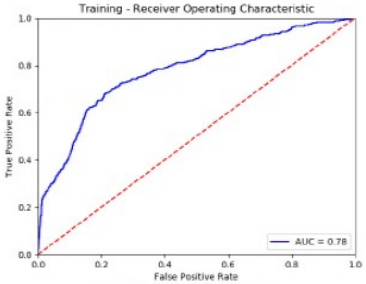
(Source: https://stackoverflow.com/questions/14254203/mixing-categorial-and-continuous-data-in-naive-bayes-classifier-using-scikit-lea)

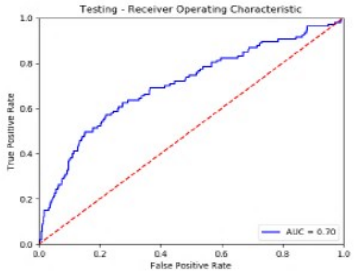


RESULTS:

- Mode-Standardize:

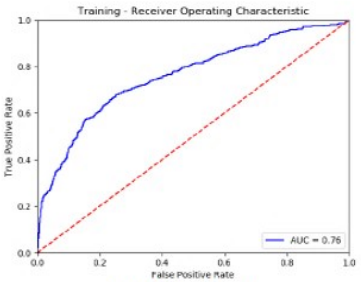
Training Confusion Matrix	Testing Confusion Matrix
[[2565 180] [225 119]]	[[864 59] [80 27]]

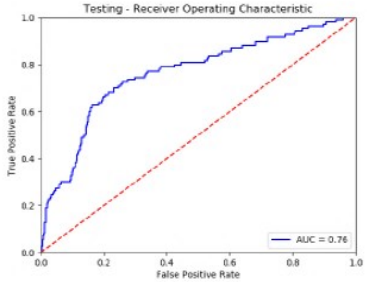




- Mode-Normalize:

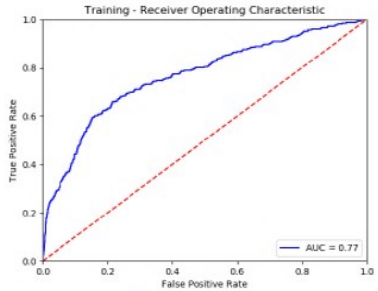
Training Confusion Matrix	Testing Confusion Matrix
[[2575 173] [221 120]]	[[868 52] [80 30]]

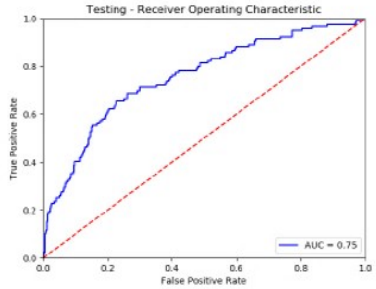




- SVM-Standardize:

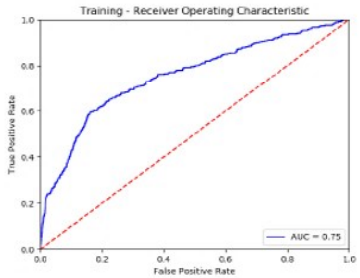
Training Confusion Matrix	Testing Confusion Matrix
[[2583 174] [219 113]]	[[857 54] [86 33]]

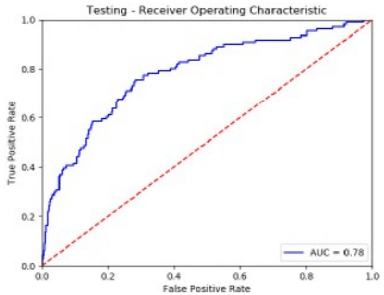




- SVM-Normalize:

Training Confusion Matrix	Testing Confusion Matrix
[[2573 176] [232 108]]	[[856 63] [67 44]]



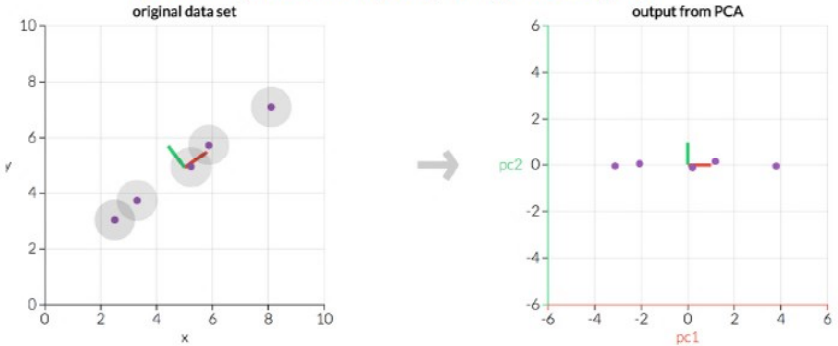


	Mode-Standardize		Mode-Normalize		SVM-Standardize		SVM-Normalize	
	Train	Test	Train	Test	Train	Test	Train	Test
Accuracy	0.8688	0.865	0.8724	0.8718	0.8727	0.8640	0.8679	0.8737
Precision	0.3979	0.3139	0.4095	0.3658	0.3937	0.3793	0.3802	0.411
Recall	0.3459	0.252	0.3519	0.2727	0.3403	0.2773	0.3176	0.3963
Class 0 F1	0.93	0.93	0.93	0.93	0.93	0.92	0.93	0.93
Class 1 F1	0.37	0.28	0.38	0.31	0.37	0.32	0.35	0.40
F1 Score	0.3701	0.2797	0.3785	0.3124	0.3651	0.3203	0.3461	0.4036
Weighted F1 Score	0.864	0.8584	0.86817	0.8634	0.8686	0.8546	0.8626	0.8727
AUC	0.7799	0.7038	0.7596	0.7633	0.7659	0.7538	0.7542	0.7833

DIMENSIONALITY REDUCTION
PRINCIPAL COMPONENT ANALYSIS

THEORY:

- Principal Component Analysis is one type of Dimensionality Reduction Techniques. This is commonly used in Machine Learning when the number of features or dimension is very high in the data to model fitted
- This method is used to emphasize variation in the features and select feature orientations that have stronger variance
- PCA finds a new coordinate system to which the entire data is projected
- The dimension of the coordinate system is the number of principal components that we choose while performing PCA of the data
- The principal components represent the directions of the data which has largest variance or discriminant power
- An example is given below:
  (Source: http://setosa.io/ev/principal-component-analysis/)



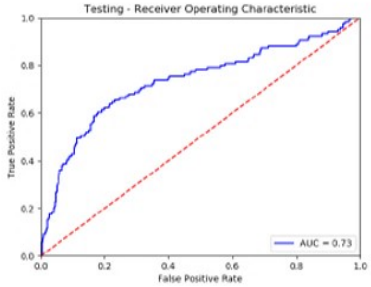
Red and Green line show the direction of principal components. The original data set has more variance in the direction of pc1. Hence when we reduce the dimension, we can project the data towards the dimension of Red (pc1)

- I used *sklearn.decompositio.PCA* for this PCA Implementation process.
- I tried PCA on the best SVM and Logistic Regression outputs and the metrics are reported below. Number of Principal Components were chosen to be 5.

RESULTS:

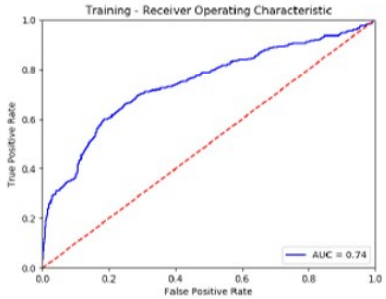
- PCA on Best SVM: C = 2.2, Gamma = 0.05

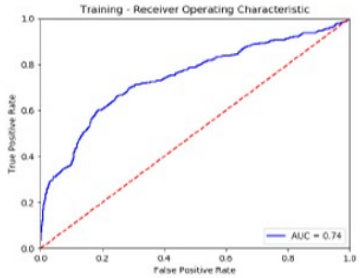
Training Confusion Matrix	Testing Confusion Matrix
[[2217 540] [113 219]]	[[729 182] [45 74]]



- PCA on Best Logistic Regression: C = 0.3, Penalty = l2

Training Confusion Matrix	Testing Confusion Matrix
[[1943 805] [103 238]]	[[649 271] [28 82]]





	PCA ON BEST SVM		Mode-Normalize	
	Train	Test	Train	Test
Accuracy	0.7886	0.7796	0.7060	0.70970
Precision	0.2885	0.289	0.2281	0.2322
Recall	0.6596	0.6218	0.6979	0.7454
Class 0 F1	0.87	0.87	0.81	0.34
Class 1 F1	0.40	0.39	0.81	0.35
F1 Score	0.40146	0.3946	0.3439	0.3542
Weighted F1 Score	0.8211	0.8109	0.7590	0.7638
AUC	0.8221	0.7339	0.7430	0.7612

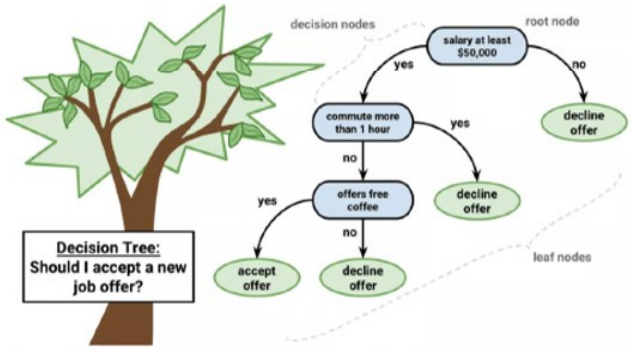
ENSEMBLE MODEL
RANDOM FOREST CLASSIFIER

THEORY:

Random Forest Classifier is becoming to be widely used in all applications because of its various advantages and applications. Some of them are:

- It is very generic and can be used for both classification and regression
- Random Forest Classifier by itself can handle missing values
- Even when more decision trees are added to the classifier, there will be no problem of overfitting. People from various industries suffer with overfitting problem.
- Again, Random Forest Classifier can handle both Categorical and Continuous Numerical values.

As the name suggest, Random Forest Classifier builds a forest with many number of trees (i.e.) decision trees. Higher the number of decision trees, the accuracy is high. As mentioned above there is no problem of overfitting. First, let us understand the concept of decision trees using a simple example given below:
(Source: https://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/)



- I used *sklearn.ensemble.RandomForestClassifiern* for this Implementation
- To address the imbalance problem, I have used 'class_weights' parameter while fitting the model in sklearn. This takes care of the imbalanced dataset.
- I have got certain evaluation metrics related to imbalanced data like
  - Confusion Matrix, Precision and Recall
  - Class 0 F1 Score, Class 1 F1 Score, F1 Score and Weighted F1 Score
  - ROC Curve and AUC value
- These metrics are specifically used for Imbalanced dataset.

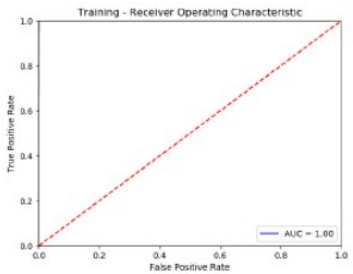
CROSS VALIDATION FOR CHOOSING BEST:

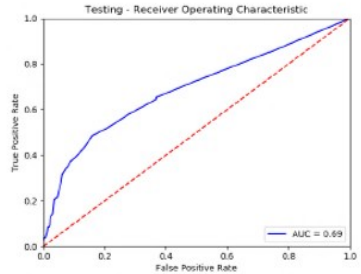
- Number of estimators: 5 to 50 (decision treess)

RESULTS:

- Mode-Standardize: Chosen number of estimators = 40

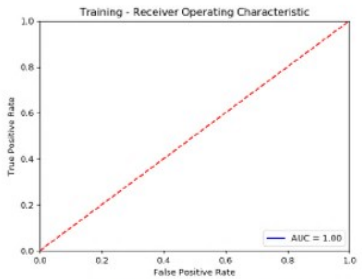
Training Confusion Matrix	Testing Confusion Matrix
[[2743 2] [10 334]]	[[907 16] [98 9]]

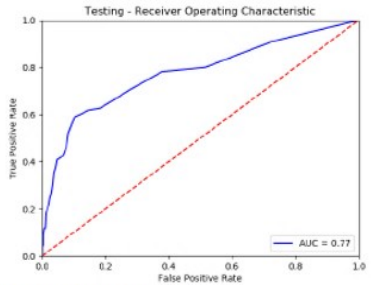




- Mode-Normalize: Chosen number of estimators = 40

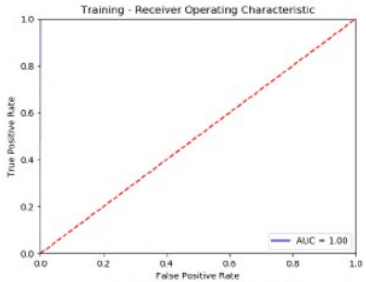
Training Confusion Matrix	Testing Confusion Matrix
[[2748 0] [3 338]]	[[900 20] [86 24]]

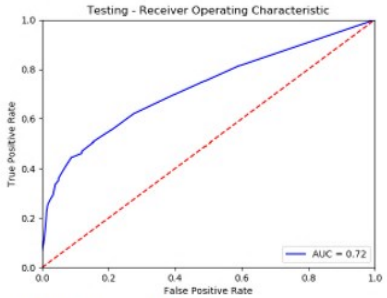




- SVM-Standardize: Chosen number of estimators = 40

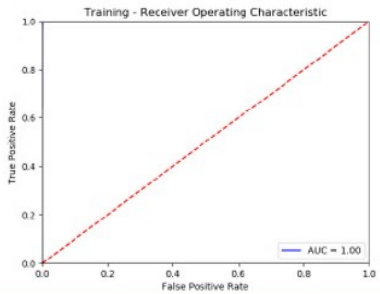
Training Confusion Matrix	Testing Confusion Matrix
[[2755 2] [6 326]]	[[896 15] [89 30]]

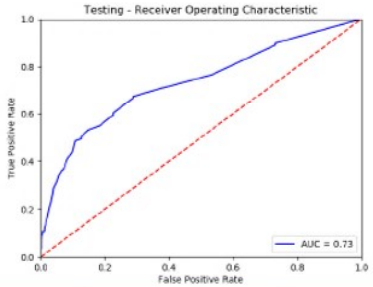




- SVM-Normalize: Chosen number of estimators = 40

Training Confusion Matrix	Testing Confusion Matrix
[[2747 2] [6 334]]	[[905 14] [95 16]]





	Mode-Standardize		Mode-Normalize		SVM-Standardize		SVM-Normalize	
	Train	Test	Train	Test	Train	Test	Train	Test
Accuracy	0.9961	0.8893	0.9990	0.89708	0.9974	0.8990	0.9974	0.8941
Precision	0.99404	0.36	1.0	0.5454	0.9939	0.6666	0.9940	0.5333
Recall	0.9709	0.0841	0.9912	0.21818	0.9819	0.2521	0.982	0.1441
Class 0 F1	1.0	0.95	1.00	0.94	1.00	0.95	1.0	0.94
Class 1 F1	0.98	0.14	1.00	0.31	0.99	0.37	0.99	0.23
F1 Score	0.9823	0.9823	0.9955	0.3116	0.9878	0.36585	0.9881	0.226
Weighted F1 Score	0.9960	0.8572	0.99902	0.8768	0.9974	0.8782	0.9974	0.8660
AUC	0.9998	0.6881	1.0	0.7746	0.9997	0.72484	0.9998	0.7320

INTERPRETATION OF RESULTS

Some Interpretations I could come up with the above reported results and non-reported results:

- **The data used had some unknowns in the feature values. To fill the unknowns, I used two methods – Mode Impute and SVM Impute**
  - Both the methods did not show much of a difference in the results
  - This was mainly because, the number of unknowns in the data is not very much. The method of imputation did not hence matter much
  - The percentage of unknowns present in the data varied from 0.2% to 4% for the features that were considered for the classification process
  - Hence this is not a large percentage of data to have unknowns. I believe this is the reason, the method of imputation did not matter.
  - If the percentage of unknowns in the data was huge, definitely the method of imputation would have changed the underlying distribution of the features and the Classification results would have also been different

- **Removing two features in the data:**
  - As I have mentioned in the preprocessing step, I have removed two features - default and pdays. This was decided on the intuition after seeing the histogram plots
  - But still I wanted to try most of the classifiers with those features also. So, on my trial runs I saw that these features weren't of much importance. The results using them were either have less F1 Score or did not change anything.
  - If the F1 score does not change, even after including those features, it means those features add no value and can be removed to reduce the dimension

- **The data used was class imbalanced.**
  - The percentage of Class 1 was exactly 10.94% of the entire data set. Since I used inbuilt function to separate the dataset into train and test set, the percentage of Class 1 in both of them was still maintained to be around 10%
  - I used two different methods to address the class imbalance in the dataset
    - Used class weight parameter while fitting most of the classifier in scikit-learn
    - Tried SMOTE technique from *imblearn.over_sampling.SMOTE* This oversamples the data which is has less frequency in terms of classes. So, I oversampled the data in such a way that the number of data samples from Class 1 and Class 0 are equal
  - I have reported all the above results using the class weight parameter passed with argument as "balanced". I haven't reported SMOTE technique results since they were approximately the same as using the class weight parameter while fitting the models.
  - I also haven't reported results which I tried without both the techniques since those F1 Scores were comparatively less than the scores I obtained after using these methods to address imbalance in the data.

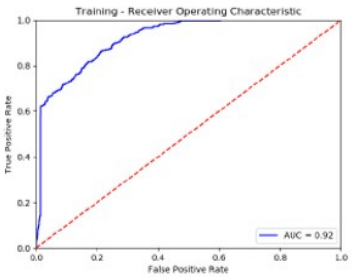
- **Dimensionality Reduction results:**
  - I have showed the dimensionality reduction results on the best performance I had before. I reduced the dimension of the features from 30 to 5 based on the principal components of the train features
  - As we can see, dimensionality reduction like PCA did not improve the performance of the Classifier. It either remains approximately same as in the case of SVM or it reduces the performance as in the case of Logistic Regression
  - Some reasons I could think of reduction in performance after performing dimensionality reduction might be:
    - The dimensions are not too high to be reduced
    - Given more time, I could have tried different other methods of Dimensionality reduction that could have improved the performance instead of reducing it
    - The other methods include Linear Discriminant Analysis, Graph Based Kernel PCA, Auto encoders etc.

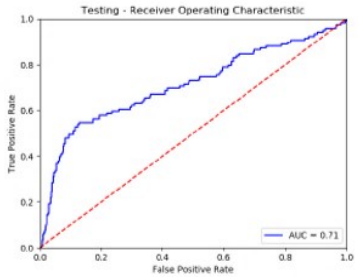
- **Classifier Results Interpretation:**
  - From the above results I have tried 6 different Classifiers.
    - The Poorest performing classifiers: Perceptron, K-Nearest Neighbors
    - Average Performing Classifiers: Random Forest,
    - Good Performing Classifiers: SVM, Logistic Regression, Naïve Bayes
  - Poor Performance was seen in Perceptron and K-Nearest Neighbors since their normal F1 Score was around 0.2
  - Random Forest had average performance since the F1 score was around 0.3
  - Good Performing Classifiers had their F1 score greater than 0.4
  - All the classifiers had best weighted F1 score greater than 0.8 and AUC of the ROC curve is greater than 0.7
  - I tried implementing Multilayer Perceptrons for Neural Network using *sklearn.neural_network.MultiLayerPerceptrons*
    - I haven't reported the results since it was one of the poor performing Classifier.
    - My take on this poor performance of Neural Network is that the number of training samples is very less.
    - This results in over-fitting of the Multilayer Perceptrons. Again, resulting in poor performance on Testing data
    - I could not avoid over-fitting of the data by determining the hyper parameters using cross-validation too.
    - This is the case with ensemble modeling also. The Random Forest Classifier reported has very high training accuracy and the AUC = 1 due to over-fitting.
    - If we had used the entire Bank Marketing Dataset instead of just a subset of it, I think Neural Network or Ensemble models would have outperformed any other Classifier

- **Best Performing Classifier Results**
  - Mode of Imputation: SVM
  - Continuous features were Standardized – 0 mean and unit variance
  - SVM was Trained using Cross Validation:
    - Hyper parameter C = 1.5 was chosen
    - Hyper parameter Gamma = 1.5 was chosen

Training Confusion Matrix	Testing Confusion Matrix
[[2439 318] [88 244]]	[[791 120] [54 65]]

	SVM-Standardize	
	Train	**Test**
Accuracy	0.8685	0.8310
Precision	0.4341	0.3513
Recall	0.7349	0.5462
Class 0 F1	0.92	0.90
Class 1 F1	0.55	0.43
F1 Score	0.5458	0.4276
Weighted F1 Score	0.8826	0.8462
AUC	0.9201	0.7107





Mode Impute (Filling Unkowns):

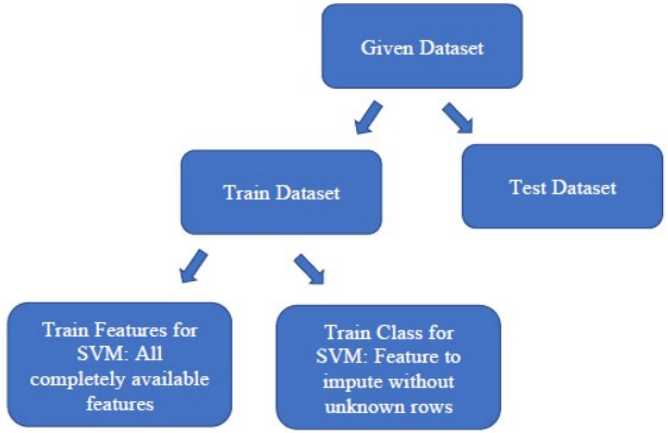
As I showed in the histogram of features, there are many unknown values. These unknown values should be filled with some other feature values present in the training data.

- One method of filling the unknown values in the train and test features is using Mode Impute Method
- For this Method, I first segregated rows with feature values without unknowns
- Based on the new train features, I found the feature value which has the highest frequency. That is, I found the feature value which had occurred maximum number of times. This is the mode of the feature. Let us call it as 'Mode Feature'
- All the unknowns of that feature were filled by 'Mode Feature'
- This was repeated for all feature columns having unknown values
- This can be easily verified from the histogram plots I provided in the previous section. The 'Mode Feature' value selected would have the bar with largest height.
- The Categorical Features with unknown values, and the mode feature value found to replace is given in the table below

Feature with Unknown Values	'Mode Feature' value used to replace
Job	admin
Marital	married
Education	university.degree
Housing	yes
Loan	no

SVM Impute (Filling Unknowns):

- In the above section I explained how Mode Impute concept was used to fill unknown values in the features.
- The main drawback of Mode Imputing is that the all the unknowns are filled with one single value.
- If the distribution of the feature is skewed with many feature values being the same, the unknowns will also be filled with the same highest occurring value. This will make the distribution of the feature still more skewed
- This might have huge effect when there are huge number of unknowns during the classification process since the feature might become useless and noisy
- Thus, another method was tried to fill unknown values which is called SVM Impute
- The main idea behind this SVM impute is that, the features which do not have unknowns and are completely available in nature are considered to be features for SVM classifier. The Class would be the feature value that has unknowns.
- A complete flow chart is given below on how this was done so that it is better understandable.



- Thus, the SVM model is trained using those Train Features and Train Class Information
- The trained SVM model is used to predict Feature Values where there are unknowns
- New SVM is trained for every Feature that has Unknown Values
- By this method, there were different values chosen for filling based on other features of the data. Meaning, all the unknowns were not filled by one single value.
- Filling was done on both Train and Test Data. But the SVM model was fit only based on Training data with feature values without unknowns
- I used *sklearn.svm.SVC* for this process. I used the default parameters for initialization.
- The summary of SVM Imputation is shown below

Feature with Unknown Values	Counts of Unknowns filled with: (Train Data)	Counts of Unknowns filled with: (Test Data)
Job	Blue Collar – 18 Admin - 9	Blue Collar – 9 Admin - 3
Marital	Married – 9	Married - 2
Education	university.degree – 88 high.school - 35	university.degree – 35 high.school - 9
Housing	yes – 60 no - 26	yes – 15 no - 4
Loan	no - 86	no - 19

- On comparison with the Mode Impute, SVM impute works better since we can see that not all unknowns were mapped to single feature value in 3 out of 5 cases
- For example, all job unknowns were filled with 'Admin' using Mode Impute. But using SVM Impute most unknowns were filled with 'Blue Collar' and few were filled with 'admin'