engage_dataset_classification

March 18, 2018

1 Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import datasets
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn import metrics
        from pandas.plotting import scatter_matrix
```

```
In [2]: data=pd.read_csv("D49.csv")
        data.head()
```

```
Out[2]:    Unnamed: 0        F0        F1        F2        F3        F4        F5  \
        0           0  0.348462  0.313163 -0.883081  0.011705  0.021841  0.007599
        1           1  0.377118  0.302959 -0.875143  0.005516  0.009458  0.001826
        2           2  0.362622  0.306870 -0.879939  0.001426  0.005996  0.002257
        3           3  0.360025  0.282392 -0.888977  0.004788  0.017349  0.005781
        4           4  0.361235  0.280317 -0.889302  0.003201  0.007381  0.003100

           Label
        0      0
        1      1
        2      1
        3      1
        4      1
```

```
In [3]: data=data.drop('Unnamed: 0',axis=1)
        data.head()
```

```
Out[3]:        F0        F1        F2        F3        F4        F5  Label
        0  0.348462  0.313163 -0.883081  0.011705  0.021841  0.007599      0
        1  0.377118  0.302959 -0.875143  0.005516  0.009458  0.001826      1
        2  0.362622  0.306870 -0.879939  0.001426  0.005996  0.002257      1
        3  0.360025  0.282392 -0.888977  0.004788  0.017349  0.005781      1
        4  0.361235  0.280317 -0.889302  0.003201  0.007381  0.003100      1
```

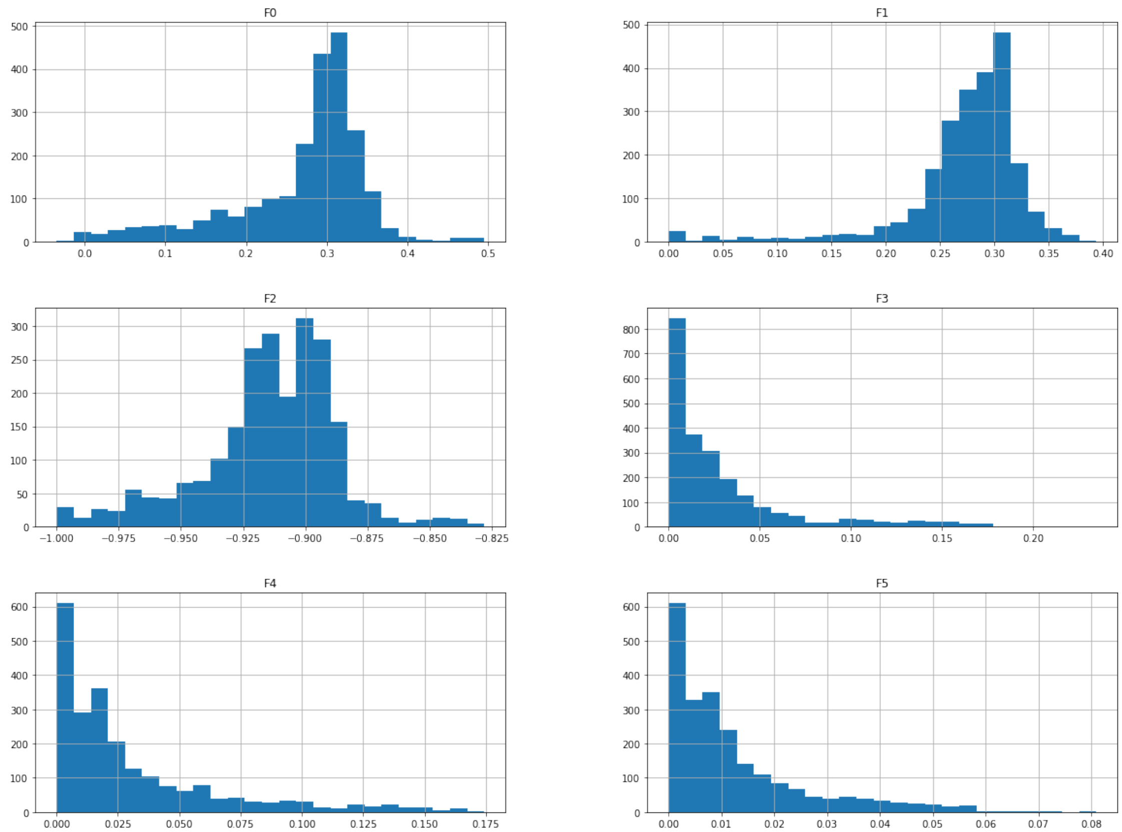
2 Exploratory Data Analysis

In [4]:
```
data1=data.drop('Label',axis = 1)
data1.describe()
```

Out[4]:
```
                F0           F1           F2           F3           F4  \
count  2254.000000  2254.000000  2254.000000  2254.000000  2254.000000
mean      0.270045     0.273486    -0.914659     0.029286     0.031333
std       0.083154     0.056479     0.027533     0.037563     0.035170
min      -0.034902     0.000000    -1.000000     0.000000     0.000000
25%       0.241176     0.257207    -0.926121     0.003580     0.006310
50%       0.295948     0.285910    -0.911985     0.016878     0.018052
75%       0.320237     0.305767    -0.896730     0.035770     0.040951
max       0.495358     0.393770    -0.827965     0.234595     0.174415

                F5
count  2254.000000
mean      0.012650
std       0.013287
min       0.000000
25%       0.002822
50%       0.008173
75%       0.017039
max       0.080896
```

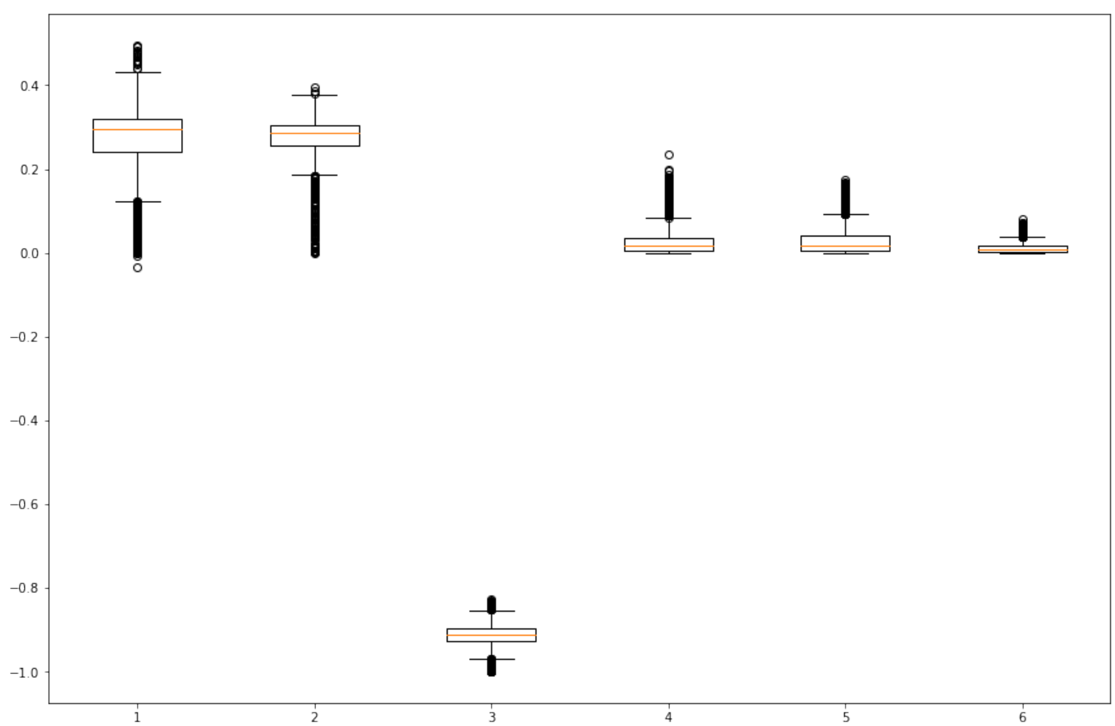
In [5]:
```
data1.hist(bins=25,figsize=(20,15))
plt.show()
```




```
In [6]: data1=np.array(data1)
        plt.figure(1,figsize=(15, 10))
        plt.boxplot(data1)
```

```
Out[6]: {'boxes': [<matplotlib.lines.Line2D at 0x7fe3641a5b38>,
          <matplotlib.lines.Line2D at 0x7fe3641b4f98>,
          <matplotlib.lines.Line2D at 0x7fe3641c6940>,
          <matplotlib.lines.Line2D at 0x7fe3641592b0>,
          <matplotlib.lines.Line2D at 0x7fe36416abe0>,
          <matplotlib.lines.Line2D at 0x7fe364183550>],
         'caps': [<matplotlib.lines.Line2D at 0x7fe3641a9f28>,
          <matplotlib.lines.Line2D at 0x7fe3641afe10>,
          <matplotlib.lines.Line2D at 0x7fe3641bbf60>,
          <matplotlib.lines.Line2D at 0x7fe3641c17b8>,
          <matplotlib.lines.Line2D at 0x7fe3641d08d0>,
          <matplotlib.lines.Line2D at 0x7fe3641d0a58>,
          <matplotlib.lines.Line2D at 0x7fe364160b70>,
          <matplotlib.lines.Line2D at 0x7fe364166a58>,
          <matplotlib.lines.Line2D at 0x7fe364176b70>,
          <matplotlib.lines.Line2D at 0x7fe364176cf8>,
          <matplotlib.lines.Line2D at 0x7fe364587be0>,
```

```
 <matplotlib.lines.Line2D at 0x7fe36431d208>],
'fliers': [<matplotlib.lines.Line2D at 0x7fe3641b4e80>,
 <matplotlib.lines.Line2D at 0x7fe3641c6828>,
 <matplotlib.lines.Line2D at 0x7fe3641d4ac8>,
 <matplotlib.lines.Line2D at 0x7fe36416aac8>,
 <matplotlib.lines.Line2D at 0x7fe36417cd68>,
 <matplotlib.lines.Line2D at 0x7fe364187eb8>],
'means': [],
'medians': [<matplotlib.lines.Line2D at 0x7fe3641aff98>,
 <matplotlib.lines.Line2D at 0x7fe3641c1940>,
 <matplotlib.lines.Line2D at 0x7fe3641d42b0>,
 <matplotlib.lines.Line2D at 0x7fe364166be0>,
 <matplotlib.lines.Line2D at 0x7fe36417c550>,
 <matplotlib.lines.Line2D at 0x7fe3641876a0>],
'whiskers': [<matplotlib.lines.Line2D at 0x7fe3641a5dd8>,
 <matplotlib.lines.Line2D at 0x7fe3641a9da0>,
 <matplotlib.lines.Line2D at 0x7fe3641b8ef0>,
 <matplotlib.lines.Line2D at 0x7fe3641bb748>,
 <matplotlib.lines.Line2D at 0x7fe3641cb860>,
 <matplotlib.lines.Line2D at 0x7fe3641cbf98>,
 <matplotlib.lines.Line2D at 0x7fe364159b00>,
 <matplotlib.lines.Line2D at 0x7fe3641609e8>,
 <matplotlib.lines.Line2D at 0x7fe364171b00>,
 <matplotlib.lines.Line2D at 0x7fe364171c88>,
 <matplotlib.lines.Line2D at 0x7fe3602aef60>,
 <matplotlib.lines.Line2D at 0x7fe360343cc0>]}
```



3 Creating Training and Test Dataset

In [7]:
```
train_set, test_set=train_test_split(data,test_size=0.2,random_state=42)
train_data=data.copy()
```

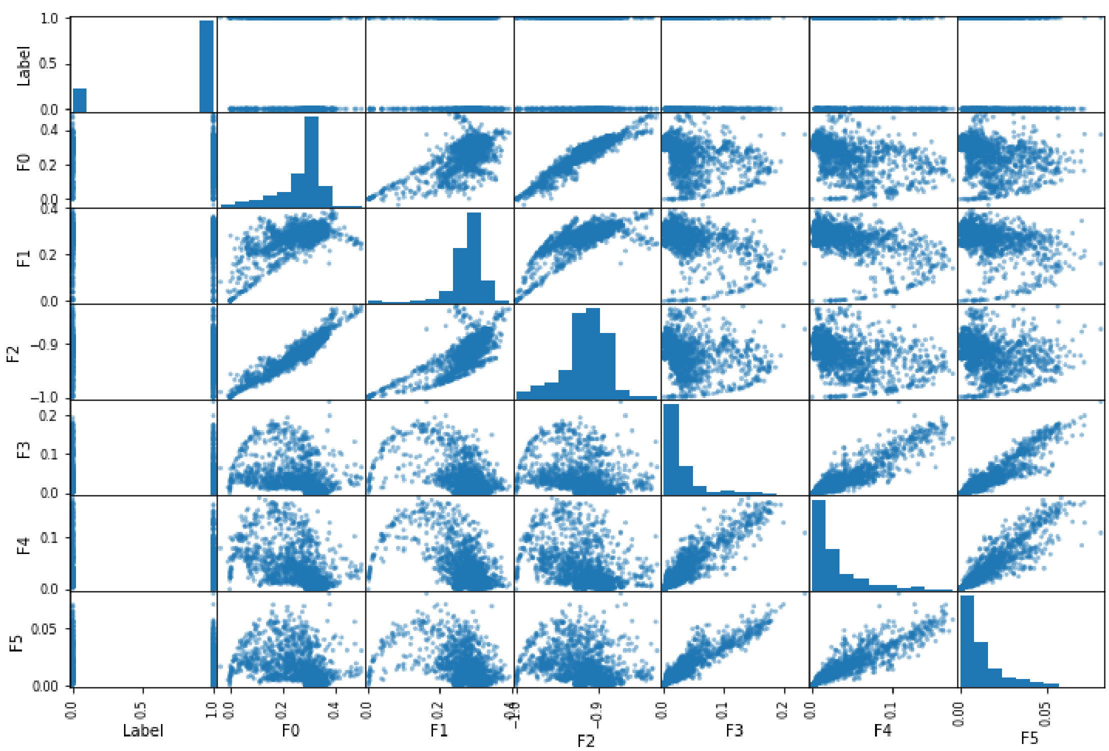
In [8]:
```
correlation_matrix=train_data.corr()
correlation_matrix['Label'].sort_values(ascending=False)
```

Out[8]:
```
Label    1.000000
F1       0.114751
F0       0.098540
F2       0.000601
F4      -0.405060
F5      -0.419292
F3      -0.423992
Name: Label, dtype: float64
```

In [9]:
```
attributes = [ "Label", "F0","F1","F2","F3","F4","F5"]
scatter_matrix(train_data[attributes], figsize=(12, 8))
```

Out[9]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fe36412d0b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe364116ef0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3640b2a58>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe364025668>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363ff2940>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3641d7b70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36426ed30>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fe3603526d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3642bdc50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36425d198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3642fa080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36436b518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363be4ef0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36034c780>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fe36443cac8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36439c908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36400f358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36441d668>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3645770b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3645523c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3644f9ef0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fe363fc6e48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363f8e8d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363ed9c18>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363e9cf98>,
```

```
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363e714e0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363e2fdd8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363e012e8>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fe364040ef0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363d79940>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363d49278>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363d06b38>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363c5ba20>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe363c1a3c8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3602736d8>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fe36023a128>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe360205080>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3601c8ac8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe360195e10>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36015f1d0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe3600ac6d8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe36006cfd0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fe36003b4e0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe35fff7e10>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe35ffce320>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe35ffe2240>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe35ff62160>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe35feae470>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fe35fe6ada0>]], dtype=object
```



4 Preparing the Data

```
In [10]: train_data = train_set.drop('Label', axis=1)
         train_labels = train_set['Label'].copy()
         train_data.head()
```

```
Out[10]:             F0        F1        F2        F3        F4        F5
         121   0.316162  0.293356 -0.901531  0.010711  0.030875  0.012595
         173   0.342953  0.328651 -0.879980  0.000991  0.002371  0.001007
         1245  0.157628  0.227237 -0.958732  0.040457  0.047689  0.020932
         1323  0.011324  0.099787 -0.991258  0.033611  0.077369  0.014335
         999   0.263980  0.299922 -0.915451  0.034215  0.030451  0.014590
```

```
In [11]: test_data = test_set.drop('Label', axis=1)
         test_labels = test_set['Label'].copy()
         test_data.shape
```

```
Out[11]: (451, 6)
```

4.1 Checking for any null or NAN values in training dataset

```
In [12]: sample_incomplete_rows = train_data[train_data.isnull().any(axis=1)].head()
         sample_incomplete_rows
```

```
Out[12]: Empty DataFrame
         Columns: [F0, F1, F2, F3, F4, F5]
         Index: []
```

4.2 Checking for any null or NAN values in test dataset

```
In [13]: sample_incomplete_rows = test_data[test_data.isnull().any(axis=1)].head()
         sample_incomplete_rows
```

```
Out[13]: Empty DataFrame
         Columns: [F0, F1, F2, F3, F4, F5]
         Index: []
```

5 Binary Classification

```
In [14]: X=np.array(train_data)
         X=X[:,(3,4,5)]
         Y=np.array(train_labels).flatten()
         test_data=np.array(test_data)
         test_data=test_data[:,(3,4,5)]
         test_labels=np.array(test_labels).flatten()
```

6 Nearest Neighbours

In [15]:
```
predicted_labels=[]
for i in range(len(test_data)):
    # euclidean distance
    minimum_distance=((np.dot(test_data[i],test_data[i]))-2*(np.dot(test_data[i],X[0]))
    closest_neighbour=Y[0]
    for j in range(1,len(X)):
        # euclidean distance
        distance=((np.dot(test_data[i],test_data[i]))-2*(np.dot(test_data[i],X[j]))+(np
        if(distance < minimum_distance):
            minimum_distance=distance
            closest_neighbour=Y[j]
    predicted_labels.append(closest_neighbour)
# print(predicted_labels)
```

7 Accuracy score - Nearest Neighbours

In [16]:
```
metrics.accuracy_score(test_labels,predicted_labels)
```

Out[16]: 0.77605321507760527

8 Precision - Nearest Neighbours

In [17]:
```
metrics.precision_score(test_labels,predicted_labels)
```

Out[17]: 0.85399449035812669

9 F-measure - Nearest Neighbours

In [18]:
```
metrics.f1_score(test_labels, predicted_labels)
```

Out[18]: 0.85991678224687929

10 Recall - Nearest Neighbours

In [19]:
```
metrics.recall_score(test_labels,predicted_labels)
```

Out[19]: 0.86592178770949724

11 AUC - Nearest Neighbours

In [20]:
```
fpr, tpr, thresholds = metrics.roc_curve(test_labels, predicted_labels)
metrics.auc(fpr, tpr)
```

Out[20]: 0.64801465729560881

12 Naive Bayes Classifier

```
In [21]: # Assuming data is fitted to a Gaussian
         def probability(mean, std, x):
             exponential=np.exp(-1*(x-mean)**2/(2*(std**2)))
             return ((1/(std*((22/7.0)**0.5)))*(exponential))
```

```
In [22]: # Fitting Gausian
         def gaussian_parameters(X):
             mean=np.mean(X,axis=0)
             std=np.std(X,axis=0)
             return (mean,std)
```

The following code is to get data points corresponding to each class

```
In [23]: data_class1= [X[i] for i in range(len(Y)) if Y[i]==1] # class1 refers to data correspon
         data_class2= [X[i] for i in range(len(Y)) if Y[i]==0] # class2 refers to data does not
```

```
In [24]: (mean_class1,std_class1)=gaussian_parameters(data_class1) # get each features gaussian
         (mean_class2,std_class2)=gaussian_parameters(data_class2) # get each features gaussian
         print(mean_class1,std_class1)
         print(mean_class2,std_class2)
         total_class1=0
         for i in range(len(Y)):
             if(Y[i]==1):
                 total_class1=total_class1+1
         class1_probability=float(total_class1)/len(Y)
         class2_probability=1-class1_probability
```

```
[ 0.02149636  0.02447165  0.00998435] [ 0.028639    0.02749351  0.01060775]
[ 0.05881004  0.0575178   0.02286286] [ 0.0486533   0.04528142  0.01641784]
```

```
In [25]: predicted_labels=[]
         for i in range(len(test_data)):
             probability_class1=1
             probability_class2=1
             for j in range(len(test_data[i])):
                 probability_class1=probability_class1*probability(mean_class1[j],std_class1[j],
                 probability_class2=probability_class2*probability(mean_class2[j],std_class2[j],
             probability_class1=probability_class1*class1_probability
             probability_class2=probability_class2*class2_probability
             if(probability_class1>probability_class2):
                 predicted_labels.append(1)
             else:
                 predicted_labels.append(0)
```

13 Accuracy score - Naive Bayes Classifier

```
In [26]: metrics.accuracy_score(test_labels,predicted_labels)

Out[26]: 0.84257206208425717
```

14 Precision - Naive Bayes Classifier

```
In [27]: metrics.precision_score(test_labels,predicted_labels)

Out[27]: 0.87862796833773082
```

15 F-measure - Naive Bayes Classifier

```
In [28]: metrics.f1_score(test_labels, predicted_labels)

Out[28]: 0.90366350067842593
```

16 Recall - Naive Bayes Classifier

```
In [29]: metrics.recall_score(test_labels,predicted_labels)

Out[29]: 0.93016759776536317
```

17 AUC - Naive Bayes Classifier

```
In [30]: fpr, tpr, thresholds = metrics.roc_curve(test_labels, predicted_labels)
         metrics.auc(fpr, tpr)

Out[30]: 0.71777197092569234
```

18 Logistic Regression - Gradient Descent

Create a copy of features of test data and insert value "1" as first feature in every data point of test_data

```
In [31]: X_data=np.copy(X)
         X_data=np.insert(X_data, 0, values=[1], axis=1)

In [32]: def sigmoid(z):
             return 1.0/(1+np.exp(-1*z))
         def gradient_descent_logistic_regression(X_data,Y,learning_rate,number_iterations):
             theta=np.zeros(X_data.shape[1])
             for i in range(number_iterations):
                 z=np.dot(X_data,theta)
                 p=sigmoid(z)
                 gradient=np.dot(X_data.T, (p - Y)) / Y.size
                 theta=theta-learning_rate*gradient
             return theta
```

```
In [33]: learning_rate=0.1
         number_iterations=30000
         theta=gradient_descent_logistic_regression(X_data,Y,learning_rate,number_iterations)
         print(theta)
```

```
[ 2.06401973 -9.42539448 -8.05800168 -3.31029465]
```

```
In [34]: test_data_new=np.copy(test_data)
         test_data_new=np.insert(test_data_new, 0, values=[1], axis=1);
         predicted_labels=[]
         for i in range(len(test_data_new)):
             if(sigmoid(np.dot(test_data_new[i],theta))>0.5):
                 predicted_labels.append(1)
             else:
                 predicted_labels.append(0)
         # print(predicted_labels)
```

19 Accuracy score - Logistic Regression (Gradient Descent)

```
In [35]: metrics.accuracy_score(test_labels,predicted_labels)
```

```
Out[35]: 0.8314855875831486
```

20 Precision - Logistic Regression (Gradient Descent)

```
In [36]: metrics.precision_score(test_labels,predicted_labels)
```

```
Out[36]: 0.8373205741626795
```

21 F-measure - Logistic Regression (Gradient Descent)

```
In [37]: metrics.f1_score(test_labels, predicted_labels)
```

```
Out[37]: 0.9020618556701031
```

22 Recall - Logistic Regression (Gradient Descent)

```
In [38]: metrics.recall_score(test_labels,predicted_labels)
```

```
Out[38]: 0.9776536312849162
```

23 AUC - Logistic Regression (Gradient Descent)

```
In [39]: fpr, tpr, thresholds = metrics.roc_curve(test_labels, predicted_labels)
         metrics.auc(fpr, tpr)
```

```
Out[39]: 0.6232354177929957
```

24 Logistic Regression - Newton's method

```
In [40]: def newton_method_logistic_regression(X_data,Y,number_iterations):
             theta=np.zeros(X_data.shape[1])
             for i in range(number_iterations):
                 z=np.dot(X_data,theta)
                 p=sigmoid(z)
                 gradient=np.dot(X_data.T, (p - Y)) / Y.size
                 column=(np.ones(p.size)).T
                 prob_product = np.dot(p,column-p)
                 learning_rate=np.linalg.inv(np.dot(prob_product,np.dot(X_data.T,X_data)/ Y.size
                 theta=theta-np.dot(learning_rate,gradient)
             return theta
```

```
In [41]: theta=newton_method_logistic_regression(X_data,Y,number_iterations)
         print(theta)
```

```
[  2.33655874 -13.07344745  -1.28614056 -28.14997027]
```

```
In [42]: predicted_labels=[]
         for i in range(len(test_data_new)):
             if(sigmoid(np.dot(test_data_new[i],theta))>0.5):
                 predicted_labels.append(1)
             else:
                 predicted_labels.append(0)
```

25 Accuracy score - Logistic Regression (Newton's method)

```
In [43]: metrics.accuracy_score(test_labels,predicted_labels)
```

```
Out[43]: 0.84035476718403546
```

26 Precision - Logistic Regression (Newton's method)

```
In [44]: metrics.precision_score(test_labels,predicted_labels)
```

```
Out[44]: 0.84708737864077666
```

27 F-measure - Logistic Regression (Newton's method)

```
In [45]: metrics.f1_score(test_labels, predicted_labels)
```

```
Out[45]: 0.90649350649350646
```

28 Recall - Logistic Regression (Newton's method)

```
In [46]: metrics.recall_score(test_labels,predicted_labels)

Out[46]: 0.97486033519553073
```

29 AUC - Logistic Regression (Newton's method)

```
In [47]: fpr, tpr, thresholds = metrics.roc_curve(test_labels, predicted_labels)
         metrics.auc(fpr, tpr)

Out[47]: 0.64872049017841049
```

30 Logistic Regression (Library)

```
In [48]: logistic_regression=LogisticRegression()
         logistic_regression.fit(X,Y)

Out[48]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [49]: predicted_labels=logistic_regression.predict(test_data) # prediction of labels for test
```

31 Accuracy score - Logistic Regression (library)

```
In [50]: metrics.accuracy_score(test_labels,predicted_labels)

Out[50]: 0.80266075388026603
```

32 Precision - Logistic Regression (library)

```
In [51]: metrics.precision_score(test_labels,predicted_labels)

Out[51]: 0.80498866213151932
```

33 F-measure - Logistic Regression (library)

```
In [52]: metrics.f1_score(test_labels, predicted_labels)

Out[52]: 0.888610763454318
```

34 Recall - Logistic Regression (library)

```
In [53]: metrics.recall_score(test_labels,predicted_labels)

Out[53]: 0.99162011173184361
```

35 AUC - Logistic Regression (library)

In [54]:
```
fpr, tpr, thresholds = metrics.roc_curve(test_labels, predicted_labels)
metrics.auc(fpr, tpr)
```

Out[54]:
```
0.53344446446807237
```