

# Pattern Recognition Assignment

Siddhartha Singh Bhadauriya(2k18/se/123)

## Problem Statement

In this assignment, the problem is basically to design a bayes classifier which can classify between the notes as genuine or forged.

### Objectives:

- (a) Use 50% of the database from both the classes for training and the remaining 50% for testing. Classes are equiprobable.
- (b) Use the testing database to compute the classification accuracy  $((TP+TN)/(TP+TN+FP+FN))$  of the Bayesian model.
- (c) Plot the ROC curves between FAR vs GAR.
- (d) Repeat the above three parts when the prior of genuine class is 0.9 and forged is 0.1.

## Solution:

**Dataset used:** We will use Banknote authentication dataset, which can be taken from the given link

(<https://archive.ics.uci.edu/ml/datasets/banknote+authentication>)

## Dataset Description:

Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images.

It basically contains 1372 instances with five attributes.

```
In [23]: #reading data from .txt file
names=['w1','w2','w3','w4','labelling']
df=pd.read_csv('C:/Users/Siddhartha/data_banknote_authentication.txt',header=None,names=names)
print (df)
```

|      | w1       | w2        | w3      | w4       | labelling |
|------|----------|-----------|---------|----------|-----------|
| 0    | 3.62160  | 8.66610   | -2.8073 | -0.44699 | 0         |
| 1    | 4.54590  | 8.16740   | -2.4586 | -1.46210 | 0         |
| 2    | 3.86600  | -2.63830  | 1.9242  | 0.10645  | 0         |
| 3    | 3.45660  | 9.52280   | -4.0112 | -3.59440 | 0         |
| 4    | 0.32924  | -4.45520  | 4.5718  | -0.98880 | 0         |
| ...  | ...      | ...       | ...     | ...      | ...       |
| 1367 | 0.40614  | 1.34920   | -1.4501 | -0.55949 | 1         |
| 1368 | -1.38870 | -4.87730  | 6.4774  | 0.34179  | 1         |
| 1369 | -3.75030 | -13.45860 | 17.5932 | -2.77710 | 1         |
| 1370 | -3.56370 | -8.38270  | 12.3930 | -1.28230 | 1         |
| 1371 | -2.54190 | -0.65804  | 2.6842  | 1.19520  | 1         |

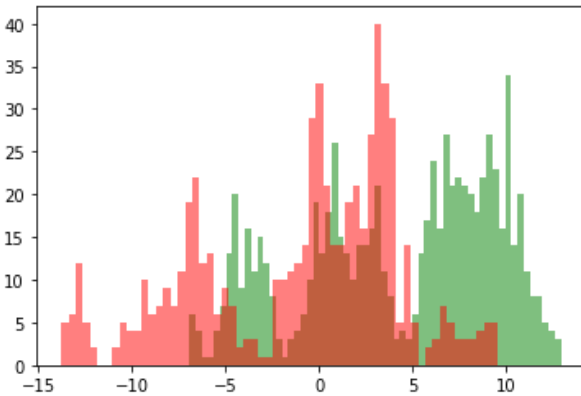
[1372 rows x 5 columns]

```
In [24]: #counting number of rows in dataset
df.count()
```

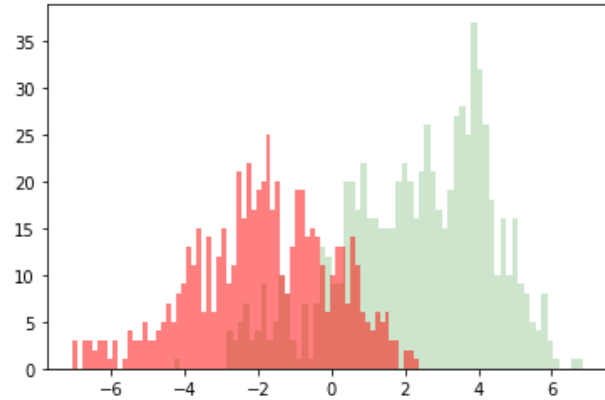
```
Out[24]: w1      1372
         w2      1372
```

## Data Visualization:

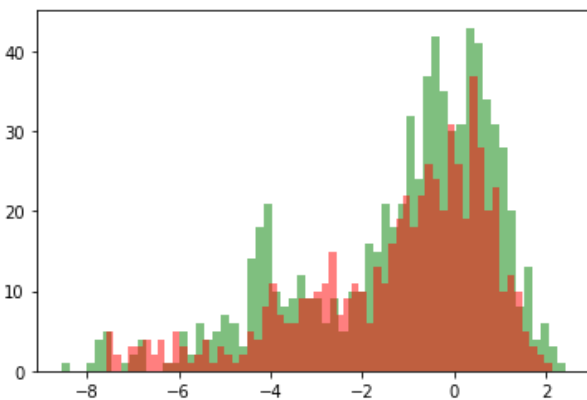
Our Aim is to visualize distribution of features, which we did using all features provided to us in the dataset.



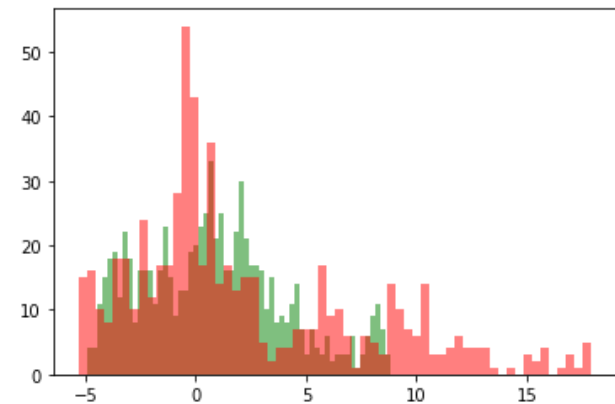
Feature 1



Feature 2

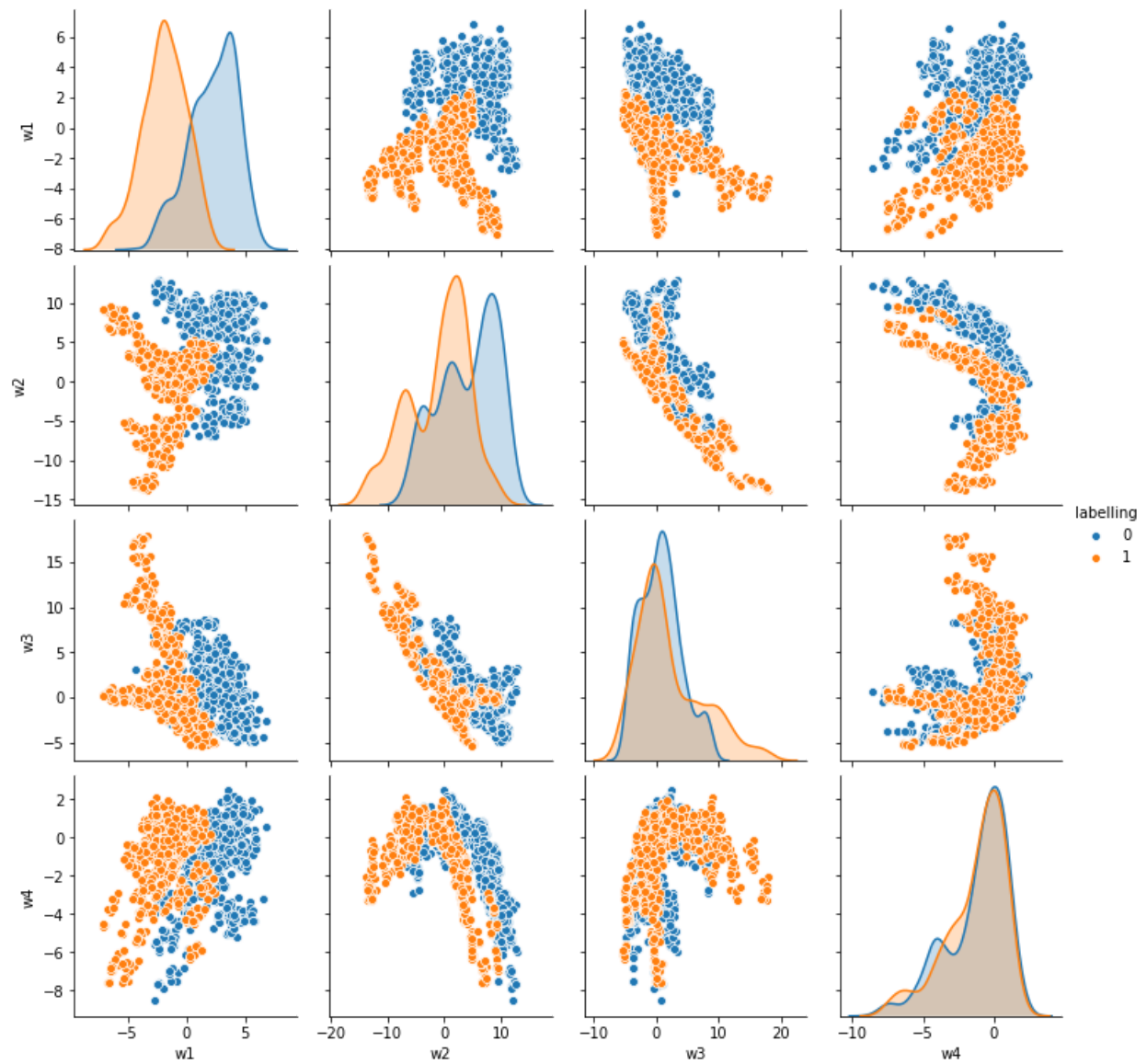


Feature 3



Feature 4

After studying this, we will plot a pairplot to get a clear picture of features are related.



## Data Preprocessing:

Now, we standardized the features by removing the mean and further scaling them to unit variance.

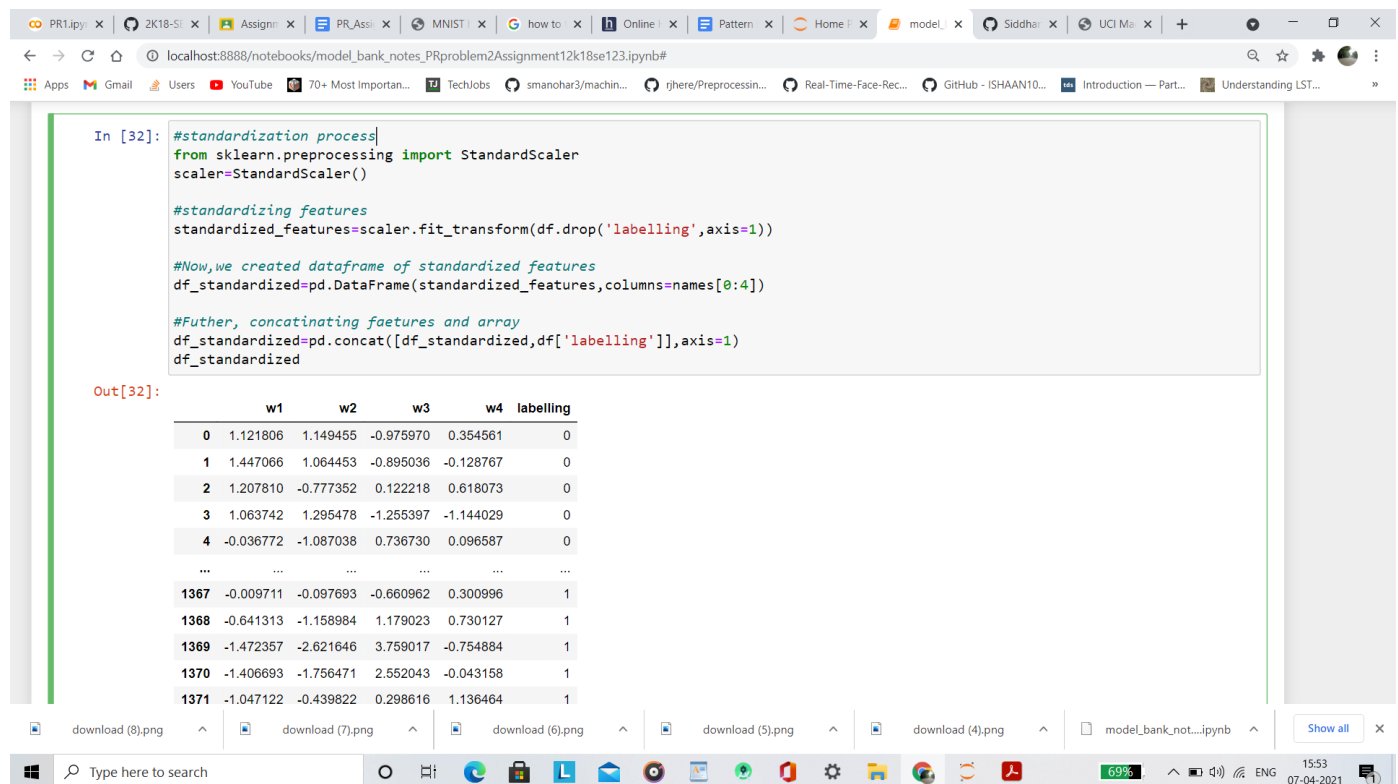
For Example:

The standard score of a sample  $x$  is calculated as:

$$z = (x - u) / s.$$

where  $u$  is the mean of the training samples, and  $s$  is the standard deviation of the training samples.

Here, we need to standardize the data otherwise it may behave badly if the individual features somewhat look like normally distributed data.



The screenshot shows a Jupyter Notebook interface with a code cell and its output. The code cell contains the following Python code:

```
In [32]: #standardization process
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()

#standardizing features
standardized_features=scaler.fit_transform(df.drop('labelling',axis=1))

#Now, we created dataframe of standardized features
df_standardized=pd.DataFrame(standardized_features,columns=names[0:4])

#Futher, concatenating faetures and array
df_standardized=pd.concat([df_standardized,df['labelling']],axis=1)
df_standardized
```

The output of the code cell is a table with 5 columns: w1, w2, w3, w4, and labelling. The table shows the standardized features for 1371 samples. The first 5 rows are shown, followed by an ellipsis, and then the last 5 rows.

|      | w1        | w2        | w3        | w4        | labelling |
|------|-----------|-----------|-----------|-----------|-----------|
| 0    | 1.121806  | 1.149455  | -0.975970 | 0.354561  | 0         |
| 1    | 1.447066  | 1.064453  | -0.895036 | -0.128767 | 0         |
| 2    | 1.207810  | -0.777352 | 0.122218  | 0.618073  | 0         |
| 3    | 1.063742  | 1.295478  | -1.255397 | -1.144029 | 0         |
| 4    | -0.036772 | -1.087038 | 0.736730  | 0.096587  | 0         |
| ...  | ...       | ...       | ...       | ...       | ...       |
| 1367 | -0.009711 | -0.097693 | -0.660962 | 0.300996  | 1         |
| 1368 | -0.641313 | -1.158984 | 1.179023  | 0.730127  | 1         |
| 1369 | -1.472357 | -2.621646 | 3.759017  | -0.754884 | 1         |
| 1370 | -1.406693 | -1.756471 | 2.552043  | -0.043158 | 1         |
| 1371 | -1.047122 | -0.439822 | 0.298616  | 1.136464  | 1         |

## Training Our Model:

We will use the Gaussian Naive Bayes model for the given dataset. We will make two models which differentiate on their prior probabilities.

Model 1:

```
from sklearn.naive_bayes import GaussianNB
clf1=GaussianNB(priors=[0.5,0.5])
clf1.fit(X_train, Y_train)
```

Out[36]: GaussianNB(priors=[0.5, 0.5], var\_smoothing=1e-09)

Model 2:

```
clf2=GaussianNB(priors=[0.1,0.9])
clf2.fit(X_train, Y_train)
```

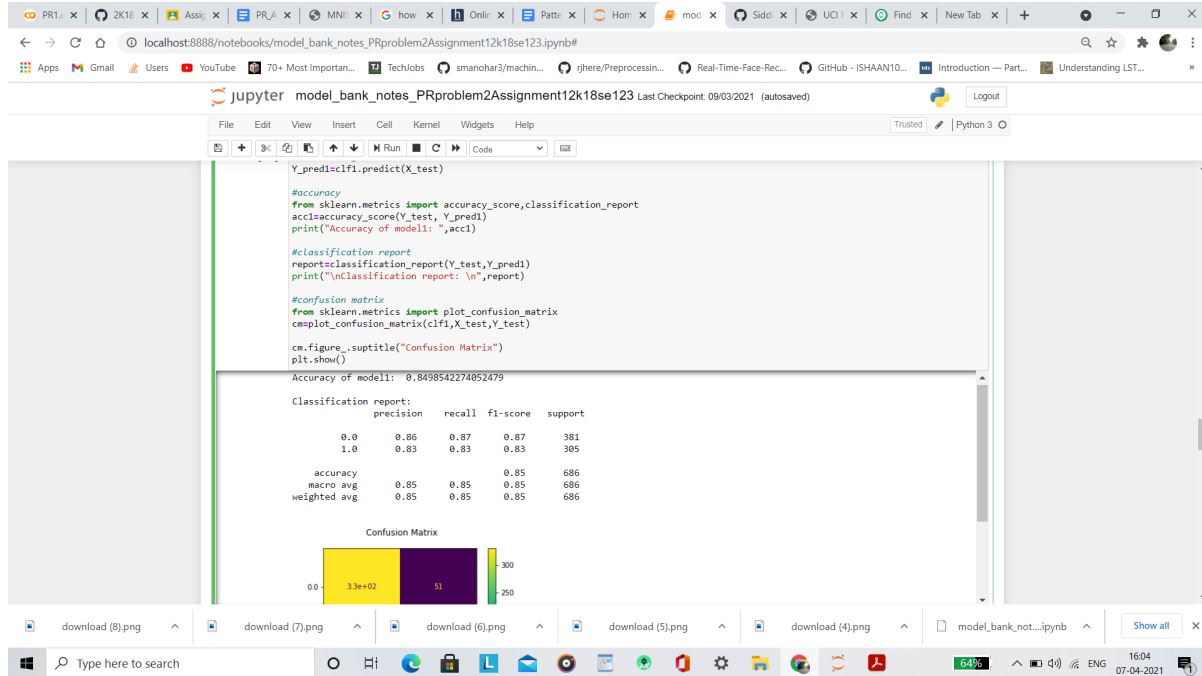
Out[49]: GaussianNB(priors=[0.1, 0.9], var\_smoothing=1e-09)

In [50]: clf2.class\_prior\_

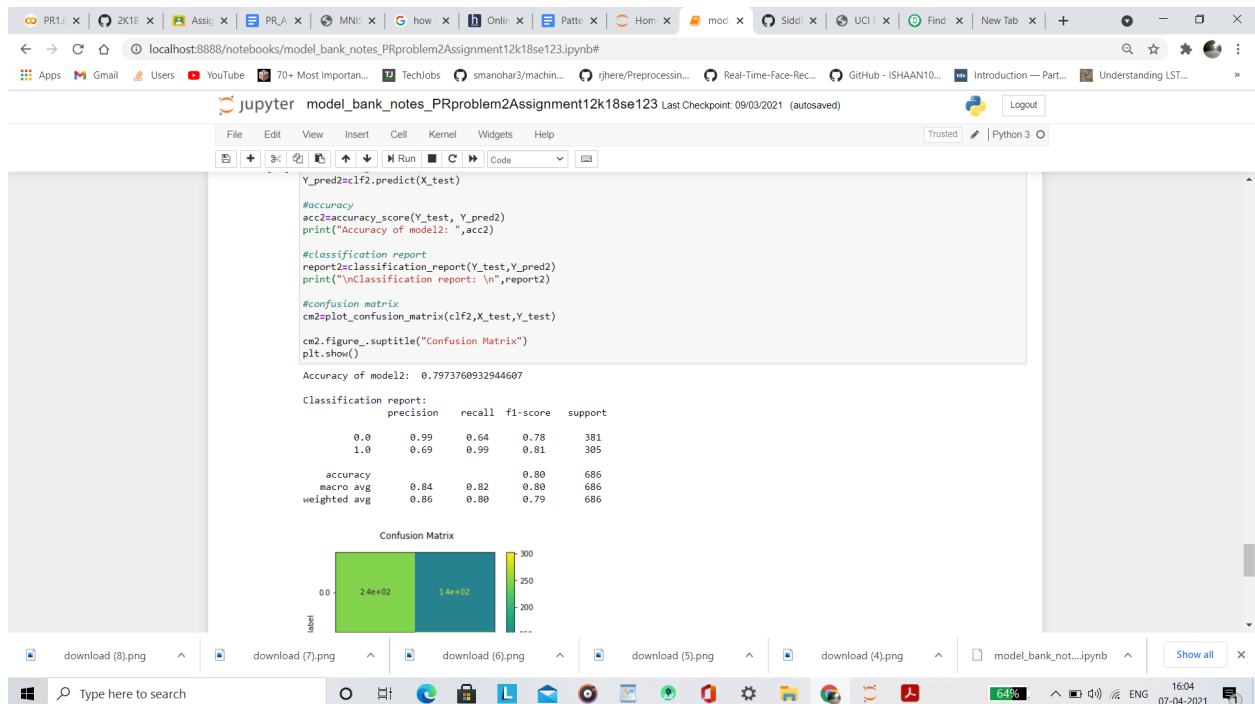
Out[50]: array([0.1, 0.9])

# Testing Model:

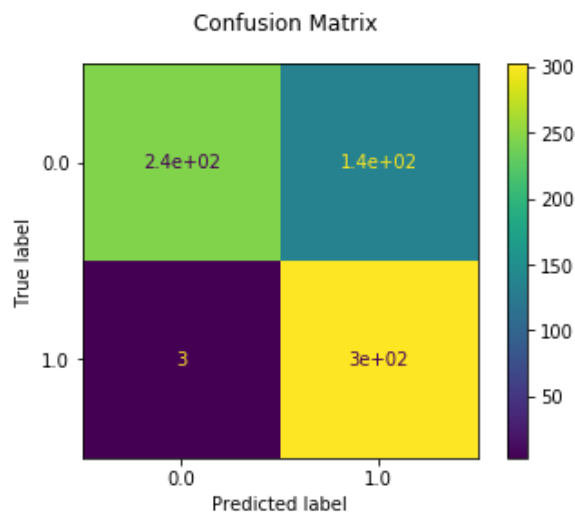
## Model 1:



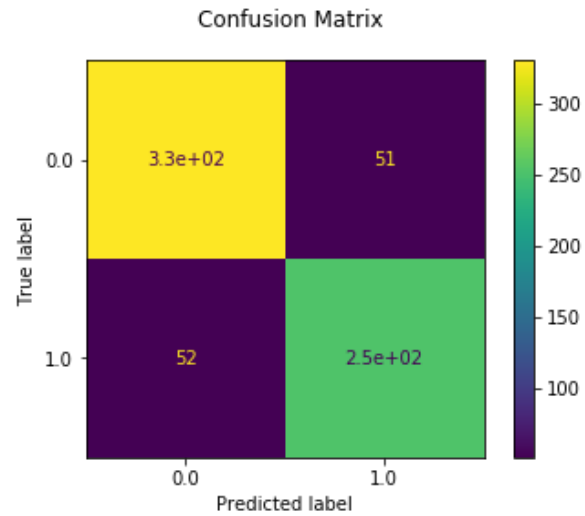
## Model 2:



After this, we printed confusion matrices as well as ROC curves for both models.

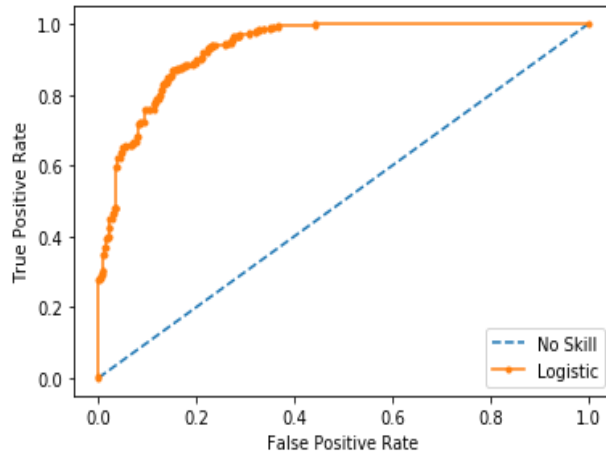


**Model 1**

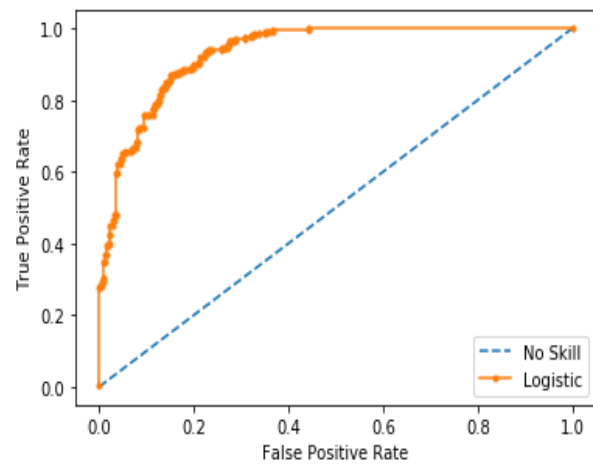


**Model 2**

**ROC Curves of both models:**



**Model 1**



**Model 2**

**Accuracy:**

**Model 1: 0.849854227405247**

**Model 2: 0.7973760932944607**