Project Note: 2

Name: Preeti Singh

Batch Name: PGPDSBA OnlineFeb20_A

Capstone Project: Tourism

Table of Content

Model Building and Interpretation:

Let's check a data info() for df_tourism2.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   ProdTaken                4888 non-null   object
 1   PreferredLoginDevice     4888 non-null   object
 2   CityTier                 4888 non-null   object
 3   Occupation               4888 non-null   object
 4   Gender                   4888 non-null   object
 5   ProductPitched           4888 non-null   object
 6   PreferredPropertyStar    4888 non-null   object
 7   MaritalStatus            4888 non-null   object
 8   Passport                 4888 non-null   object
 9   OwnCar                   4888 non-null   object
 10  Designation              4888 non-null   object
 11  Age                      4888 non-null   float64
 12  DurationOfPitch          4888 non-null   float64
 13  NumberOfPersonVisited    4888 non-null   float64
 14  NumberOfFollowups        4888 non-null   float64
 15  NumberOfTrips            4888 non-null   float64
 16  PitchSatisfactionScore   4888 non-null   float64
 17  NumberOfChildrenVisited  4888 non-null   float64
 18  MonthlyIncome            4888 non-null   float64
dtypes: float64(8), object(11)
memory usage: 725.7+ KB
```

There are some categorical and numerical variable as well. ProdTaken is target variable. For model building all feature should be in numerical nature, so first we need to convert all categorical variable into numerical variable.

Check the imbalance level in target variable.

```
No     0.811784
Yes    0.188216
Name: ProdTaken, dtype: float64
```

We can see from the above output there is a good amount of class imbalance in t he data w.r.t the target variable i.e. ProdTaken. To take care of this imbalance w e will have to apply **SMOTE.** Before applying SMOTE we will split the data in to training and testing sets to avoid introducing bias in the test data set.

We can do model building without SMOTE also because data is not highly imbalance.

But even before that we need to convert all the categorical variables into numeri cal form so that it is conducive to modelling.

There are two types of categorical variable in the data set wherein some are or o rdinal like ProductPitched,PreferredPropertyStar, Designation which is ranked b ased and rest of all are categorical where weightage are equal for all different la bel . For ordinal categorical variable we will use map and lambda function or Categorical().code and for other categorical variable we will use one hot encodi ng and or dummy variable creation.

```
df_tourism2['ProductPitched_codes'] = df_tourism2['ProductPitched'].map({'Multi':1,'Standard':2,'Deluxe':3,'Su
per Deluxe':4,'King':5})
df_tourism2.drop('ProductPitched',inplace=True,axis=1)
df_tourism2['PreferredPropertyStar_codes'] = df_tourism2['PreferredPropertyStar'].map({'3 Star':1,'4 Star':2,'5 S
tar':3})
df_tourism2.drop('PreferredPropertyStar',inplace=True,axis=1)
df_tourism2['Designation_codes'] = df_tourism2['Designation'].map({'Executive':1,'Manager':2,'Senior Manager
':3,'AVP':4,'VP':5})
df_tourism2.drop('Designation',inplace=True,axis=1)
```

```
df_tourism2_cat = df_tourism2[categorical]
df_tourism2_dummies = pd.get_dummies(df_tourism2_cat,drop_first=True)
```

Let's check the info() of df_tourism2_dummified.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 24 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Age                                 4888 non-null   float64
 1   DurationOfPitch                     4888 non-null   float64
 2   NumberOfPersonVisited               4888 non-null   float64
 3   NumberOfFollowups                   4888 non-null   float64
 4   NumberOfTrips                       4888 non-null   float64
 5   PitchSatisfactionScore              4888 non-null   float64
 6   NumberOfChildrenVisited             4888 non-null   float64
 7   MonthlyIncome                       4888 non-null   float64
 8   ProductPitched_codes                4888 non-null   int64
 9   PreferredPropertyStar_codes         4888 non-null   int64
 10  Designation_codes                   4888 non-null   int64
 11  ProdTaken_Yes                       4888 non-null   uint8
 12  PreferredLoginDevice_Self Enquiry   4888 non-null   uint8
 13  CityTier_Tier-2                     4888 non-null   uint8
 14  CityTier_Tier-3                     4888 non-null   uint8
 15  Occupation_Large Business           4888 non-null   uint8
 16  Occupation_Salaried                 4888 non-null   uint8
 17  Occupation_Small Business           4888 non-null   uint8
 18  Gender_Male                         4888 non-null   uint8
 19  MaritalStatus_Married               4888 non-null   uint8
 20  MaritalStatus_Single                4888 non-null   uint8
 21  MaritalStatus_Unmarried             4888 non-null   uint8
 22  Passport_Yes                        4888 non-null   uint8
 23  OwnCar_Yes                          4888 non-null   uint8
dtypes: float64(8), int64(3), uint8(13)
memory usage: 482.2 KB
```

Now all features are numeric and data is ready for modelling.

A very first step of model building is that we have to divide the data into predictor (independent set) and target (dependent set) set. After that we have to split the data into train and test set. In next step, we will apply SMOTE on train set to fix the problem of imbalance problem. Here comes some basic steps that we have to follow at early stage of model building.

Dividing the data set into predictor and target variable:

```
X = df_tourism2_dummified.loc[:,df_tourism2_dummified.columns != 'ProdTaken_Yes']

Y = df_tourism2_dummified.loc[:,df_tourism2_dummified.columns == 'ProdTaken_Yes']
```

Splitting the data into (70%-30% ratio) train and test set:

```
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,random_state=1234)
```

Applying SMOTE on training data:

```
from imblearn.over_sampling import SMOTE

os = SMOTE(random_state=1234)

os_data_X,os_data_Y = os.fit_sample(X_train,Y_train)

os_data_X = pd.DataFrame(os_data_X,columns = X_train.columns)

os_data_Y = pd.DataFrame(os_data_Y,columns = Y_train.columns)
```

Checking the propensity of target variables after SMOTE:

```
ProdTaken_Yes
1                0.5
0                0.5
dtype: float64
```

We can see from above that the proportion of the minority class i.e. **ProdTaken_Yes** has been increased from approximately 19% to 50%.

Checking Outliers:

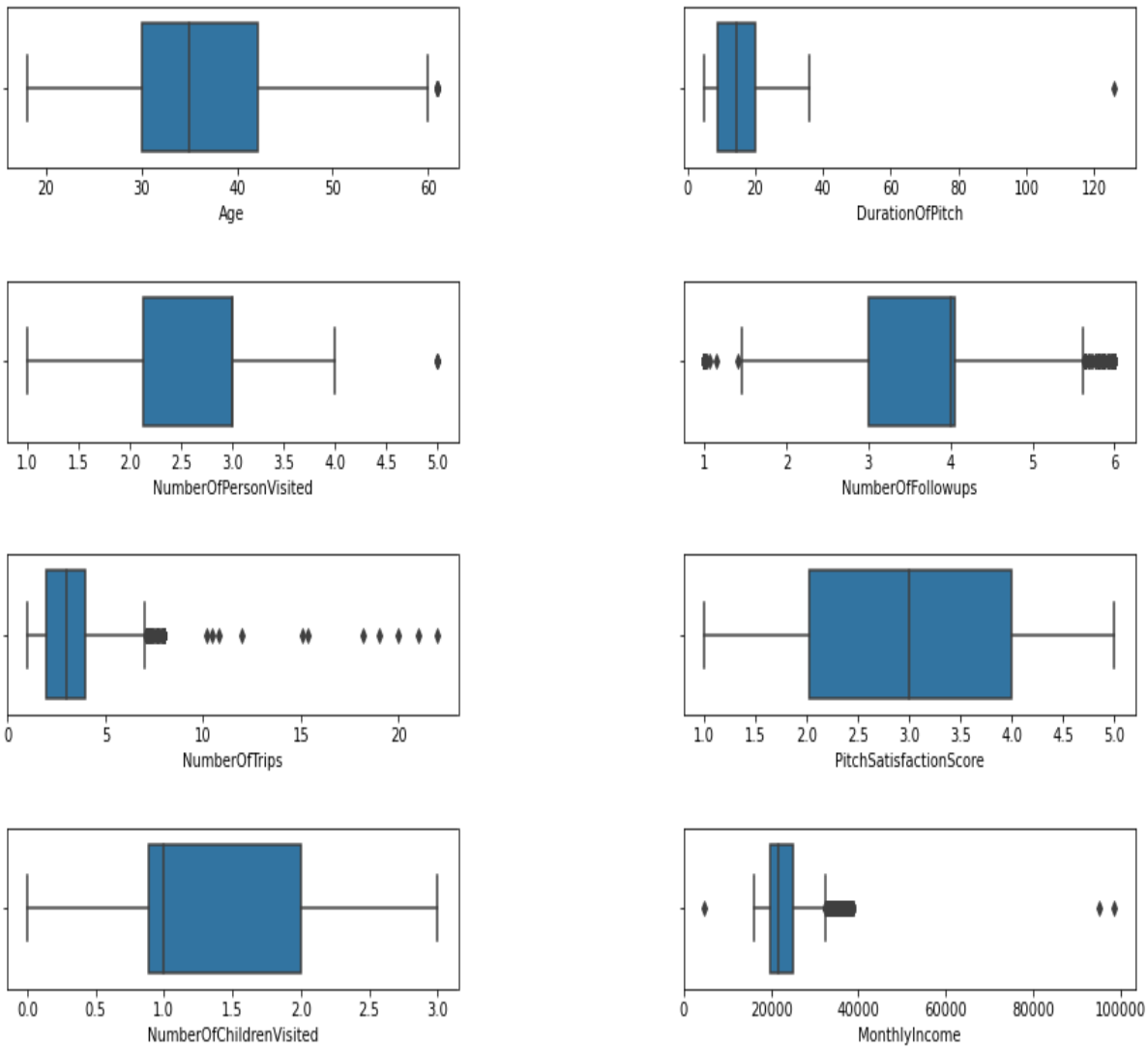

Fig.1

There are some outliers. We need to treat it.

Treating Outliers by Winsorization:

Winsorization, or winsorizing, is the process of transforming the data by limiting the extreme values, that is, the outliers, to a certain arbitrary value, closer to the mean of the distribution. Winsorizing is different from trimming because the extreme values are not removed, but are instead replaced by other values. A typical strategy involves setting outliers to a specified percentile.

In this case we used 95% winsorization, we set all data below the 5^{th} percentile to the value at the 5^{th} percentile and all data above the 95^{th} percentile to the

value at the 95th percentile.The purpose of using winsorization to avoid imposing bias into the data.

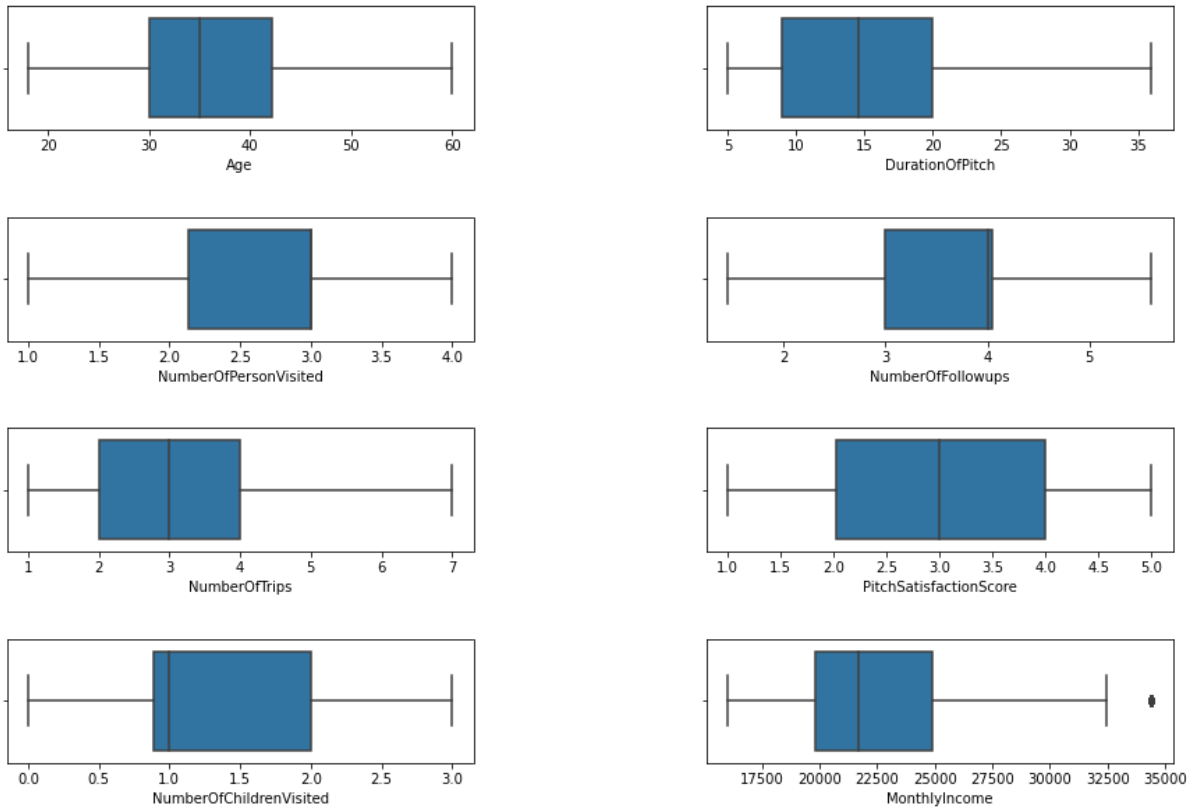

Fig: 2

Almost all outliers have removed from all features except MonthlyIncome.

Next, we are going to create base line model.The purpose of building this baseline model to get minimum level of performance from the data and to get some interpretation.

Baseline Logistic regression Model:

To build a logistic regression model, we have to first import statsmodels library. After that we have to build logistic regression model using Logit() for train set. It gives the coefficient and p_value of each feature. Here, our purpose is to figure out what all are the features significant and minimum baseline accuracy.

```
Optimization terminated successfully.
         Current function value: inf
         Iterations 8
```

Coef **Pvalue**

	Coef	Pvalue
Age	-0.005435	2.408536e-01
DurationOfPitch	0.039111	2.718492e-16
NumberOfPersonVisited	0.084538	2.302310e-01
NumberOfFollowups	0.507566	5.097726e-29
NumberOfTrips	0.058887	1.286718e-02
PitchSatisfactionScore	0.174980	6.875228e-10
NumberOfChildrenVisited	-0.273897	1.807881e-06
MonthlyIncome	0.000131	3.258623e-16
ProductPitched_codes	-0.297554	1.071177e-15
PreferredPropertyStar_codes	0.326537	6.156304e-12
Designation_codes	-0.784254	3.725798e-25
PreferredLoginDevice_Self Enquiry	-0.620127	1.484681e-15
CityTier_Tier-2	-0.121641	5.699207e-01
CityTier_Tier-3	0.488569	1.009640e-07
Occupation_Large Business	-3.996438	8.563610e-72
Occupation_Salaried	-4.207404	8.174961e-108
Occupation_Small Business	-4.126165	1.404739e-101
Gender_Male	-0.114387	1.264475e-01
MaritalStatus_Married	-1.450174	5.872600e-53
MaritalStatus_Single	0.049993	6.279424e-01
MaritalStatus_Unmarried	-0.619612	2.100732e-07
Passport_Yes	1.141454	1.587815e-45
OwnCar_Yes	-0.532684	1.705794e-12

So, to figure out significant feature we have to do hypothesis testing here.

Null Hypothesis: Feature has significant impact on dependent variable.

Alternate Hypothesis: Feature has not significant impact on dependent variable.

If p-vale > 0.05, then we have to reject null hypothesis and If p_value < 0.05, then we fail to reject null hypothesis at 95% confidence interval.

Now, we are going to **filter out those features whose p-value > 0.05**.These are the non- significant features. Later on we will drop these features while model building.

	Coef	Pvalue
Age	-0.005357	0.246912
NumberOfPersonVisited	0.086194	0.219982
CityTier_Tier-2	-0.128176	0.549969
Gender_Male	-0.105839	0.157381
MaritalStatus_Single	0.028861	0.779725

Observations:

From above output we can conclude that Age, NumberOfPersonVisited, CityTier_Tier_2, Gender_Male and MaritalStatus_Single are non-significant features. But as we have been seen in our EDA part, Age has significant impact on ProdTaken(dependent variable). So we will consider Age variable also in model building and will leave the **non-Signinficant feature NumberOfPersonVisited, CityTier_Tier_2, Gender_Male and MaritalStatus_Single.**

Building a Logistic regression model considering based on significant variable and calculating coefficients and P-value:

```
os_data_X = os_data_X[['Age', 'DurationOfPitch', 'NumberOfPersonVisited',
'NumberOfFollowups','NumberOfTrips', 'PitchSatisfactionScore', 'NumberOfChildrenVisited',
'ProductPitched_codes', 'PreferredPropertyStar_codes, 'Designation_codes',  'PreferredLoginDevice_Self
Enquiry', 'CityTier_Tier-3', 'Occupation_Large Business', 'Occupation_Salaried', 'Occupation_Small Busine,
'MaritalStatus_Married', 'MaritalStatus_Unmarried', 'Passport_Yes', 'OwnCar_Yes']]
```

X_test = X_test[['Age', 'DurationOfPitch', 'NumberOfPersonVisited', 'NumberOfFollowups', 'NumberOfTrips', 'PitchSatisfactionScore', 'NumberOfChildrenVisited', 'ProductPitched_codes', 'PreferredPropertyStar_codes', 'Designation_codes', 'PreferredLoginDevice_Self Enquiry', 'CityTier_Tier-3', 'Occupation_Large Business', 'Occupation_Salaried', 'Occupation_Small Business', 'MaritalStatus_Married', 'MaritalStatus_Unmarried', 'Passport_Yes', 'OwnCar_Yes']]

```
Optimization terminated successfully.
         Current function value: inf
         Iterations 8
```

	Coef	Pvalue
Age	-0.005428	2.404370e-01
DurationOfPitch	0.039259	1.780121e-16
NumberOfPersonVisited	0.088844	2.053901e-01
NumberOfFollowups	0.542924	9.091607e-30
NumberOfTrips	0.058852	1.298536e-02
PitchSatisfactionScore	0.172472	1.098447e-09
NumberOfChildrenVisited	-0.275016	1.573669e-06
MonthlyIncome	0.000125	9.999423e-16
ProductPitched_codes	-0.297130	1.184042e-15
PreferredPropertyStar_codes	0.322188	1.090916e-11
Designation_codes	-0.793425	6.443302e-25
PreferredLoginDevice_Self Enquiry	-0.641294	1.400271e-16
CityTier_Tier-3	0.508851	2.447838e-08
Occupation_Large Business	-4.032260	2.473993e-76
Occupation_Salaried	-4.237081	9.327497e-114

	Coef	Pvalue
Occupation_Small Business	-4.155929	6.263443e-107
MaritalStatus_Married	-1.472798	2.462015e-66
MaritalStatus_Unmarried	-0.625723	2.837693e-08
Passport_Yes	1.149069	3.429055e-46
OwnCar_Yes	-0.534483	1.423924e-12

Observations:

Now, all the above features have P_value<0.05.Hence, we can conclude these features has significant impact on dependent variable.

Interpreting the coefficient, odds and probability:

	Coef	Pvalue	Odds	Prob
Age	-0.005428	2.404370e-01	0.994587	0.498643
DurationOfPitch	0.039259	1.780121e-16	1.040039	0.509813
NumberOfPersonVisited	0.088844	2.053901e-01	1.092910	0.522196
NumberOfFollowups	0.542924	9.091607e-30	1.721032	0.632492
NumberOfTrips	0.058852	1.298536e-02	1.060619	0.514709
PitchSatisfactionScore	0.172472	1.098447e-09	1.188239	0.543011
NumberOfChildrenVisited	-0.275016	1.573669e-06	0.759560	0.431676
MonthlyIncome	0.000125	9.999423e-16	1.000125	0.500031
ProductPitched_codes	-0.297130	1.184042e-15	0.742947	0.426259

	Coef	Pvalue	Odds	Prob
PreferredPropertyStar_codes	0.322188	1.090916e-11	1.380144	0.579857
Designation_codes	-0.793425	6.443302e-25	0.452293	0.311434
PreferredLoginDevice_Self Enquiry	-0.641294	1.400271e-16	0.526611	0.344954
CityTier_Tier-3	0.508851	2.447838e-08	1.663379	0.624537
Occupation_Large Business	-4.032260	2.473993e-76	0.017734	0.017425
Occupation_Salaried	-4.237081	9.327497e-114	0.014450	0.014244
Occupation_Small Business	-4.155929	6.263443e-107	0.015671	0.015429
MaritalStatus_Married	-1.472798	2.462015e-66	0.229283	0.186518
MaritalStatus_Unmarried	-0.625723	2.837693e-08	0.534874	0.348481
Passport_Yes	1.149069	3.429055e-46	3.155254	0.759341
OwnCar_Yes	-0.534483	1.423924e-12	0.585972	0.369472

Observations:

- A customer having passport, increases the probability of taken prod by 76%.
- If the customer age increases, the probability of taken product decreases by 50%.
- A customer belongs to Tier-3, increases the probability of prod taken increases by 61%.
- If the monthly income of customer increases, the probability of prod taken increases by 50%.
- If the number of trips increases, probability of taking prod decreases by 51%.
- If the customer having small business and salaried, the probability of taking product decreases by 1%.
- A person having own car, decreases the probability of product taken by 36%.

- A customer, who has married, decreases the probability of product taken by 19%.
- A customer, whose designation is high (work as VP,AVP),decreases the probability of product taken by 31%.

Looking at baseline accuracy score:

```
Training Accuracy of Logistic Model: 0.809
Test Accuracy of Logistic Model: 0.773
```

Now, I have stored smote train independent and dependent set into new variable X_train and Y_train as per my convenience.

X_train = os_data_X

Y_train = os_data_Y

Let's build different models. ***All models are built with hyper parameter tuning. Hyper parameter tuning is the powerful tool to enhance supervised learning model- improving accuracy, recall and other important metrics by searching the optimal parameter based on different scoring methods. For hyper parameter tuning we have to use GridSearchCV() that are available in sklearn.model_selection.***

Model1: Logistic Regression:

1. First we have to import from LogisticRegression from sklearn.linear_model and GridSearchCV from skearn.model_selection.

2. Parameters are defined as follows.

```
grid = {'solver':['newton-cg','lbfgs','liblinear','sag','saga'],
    'max_iter':[1000,2000],
     'n_jobs':[2,3]
    }
```

3. After that we have to build logistic regression model and we have to pass the parameter grid into GridSearchCV. Next, we have to do model fitting.

```
model = LogisticRegression(random_state=1)
grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = 3)
```

```
grid_search.fit(X_train, Y_train.values.ravel())

LR1 = grid_search.best_estimator_

LR1.fit(X_train,Y_train.values.ravel())
```

4. Next, we can figure out best parameter for model.

```
{'max_iter': 1000, 'n_jobs': 2, 'solver': 'liblinear'}
```

5. Predicting the probability and probability class for train set.

6. Calculating model performance matrices for train and test set.

Accuracy Score:

For train set:

```
0.8131294964028777
```

For test set:

```
0.7757327880027266
```

Confusion Matrix:

For train set:

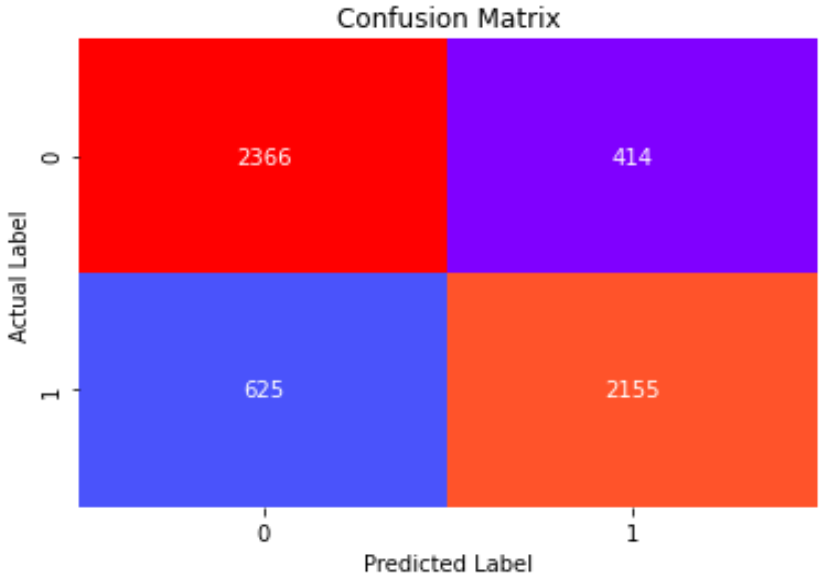

Fig.3

Classification Report:

```
              precision    recall  f1-score   support

           0       0.85      0.79      0.82      2991
           1       0.78      0.84      0.81      2569

    accuracy                           0.81      5560
```

macro avg	0.81	0.81	0.81	5560
weighted avg	0.82	0.81	0.81	5560

For test set:

Confusion matrix:

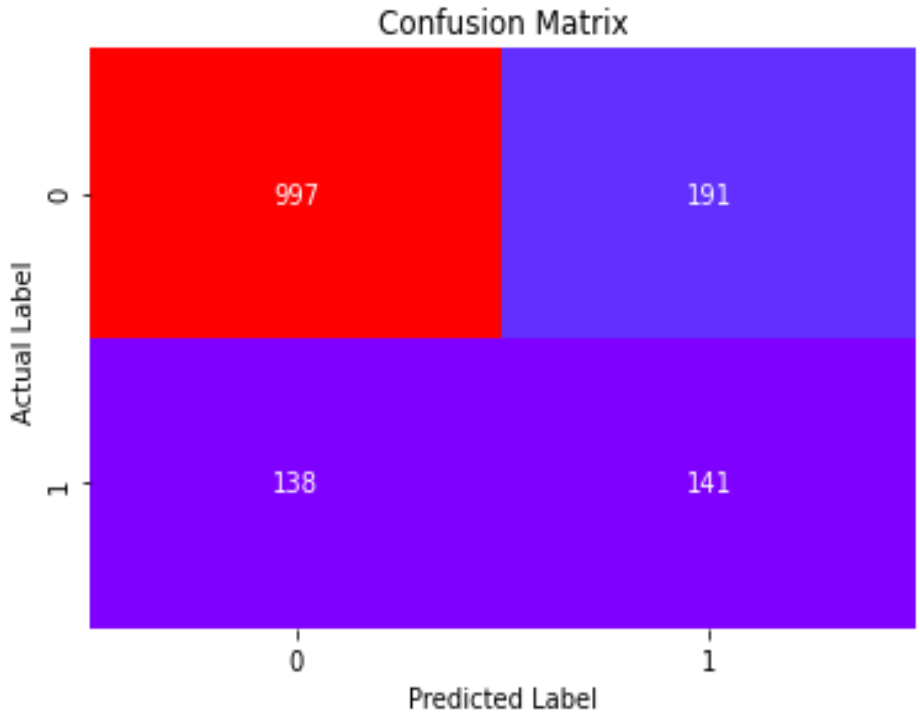

Fig.4

- Total no of correct prediction=141+997
- Total no of incorrect prediction=191+138

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.88	0.86	1135
1	0.51	0.42	0.46	332
accuracy			0.78	1467
macro avg	0.67	0.65	0.66	1467
weighted avg	0.76	0.78	0.77	1467

- 42 % customers are correctly identified as the customers who have taken product.

AUC and ROC Curve:

For train set:

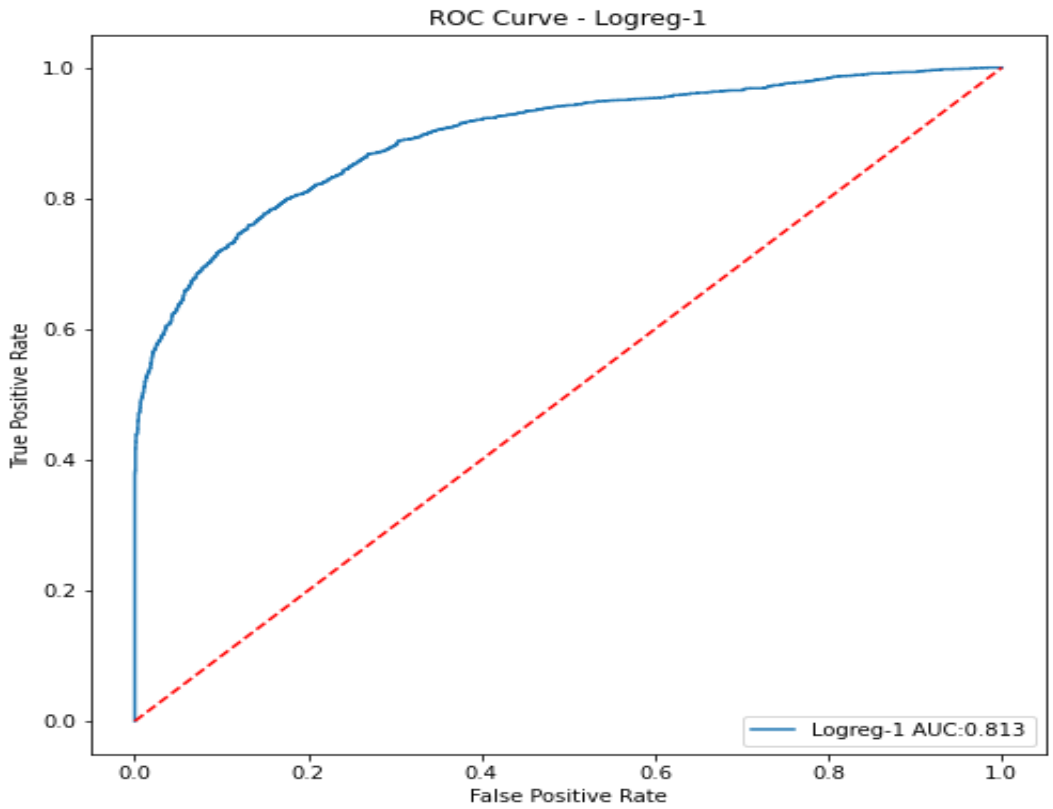

Fig.5

For test set:

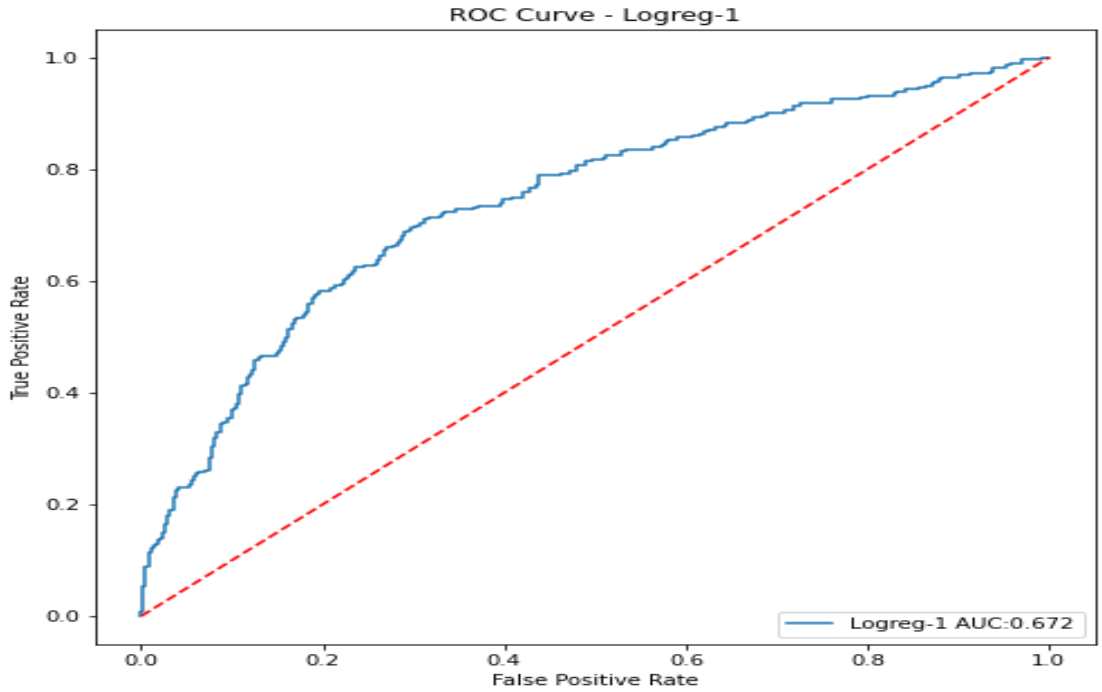

Fig.6

- For test set AUC score is less than train set. Model performs over fitting.

Model2: Logistic regression model with Recursive Feature Elimination:

RFE (Recursive Feature Elimination):RFE is feature selection algorithm that selecting those features (columns) in training data set that are more or most relevant in predicting target variable.

There are two important configuration options when using RFE: the choice in the number of features to select and the choice of the algorithm used to help choose features. Both of these hyper parameters can be explored, although the performance of the method is not strongly dependent on these hyper parameters being configured well.

1. First we have to import RFE from sklearn.feature_selection and we have to pass total no parameters that has signinficant impact on dependent variable and fit the model for train set.

```
from sklearn.feature_selection import RFE

rfe = RFE(LR1,n_features_to_select=18)
rfe = rfe.fit(X_train,Y_train.values.ravel())
```

2. After filtering out significant feature , we will divide the data into train and test set for independent set.

```
X_train_FS = X_train[col_selected]
X_test_FS = X_test[col_selected]
```

3. Next, we will build logistic regression model for X_train_FS and X_test_FS.
4. Predicting the probability and probability for train and test set.
5. Calculating performance metrics for train and test set.

Accuracy score:

For train set

```
0.810611510791367
```

For test set:

```
0.7716428084526245
```

Confusion Matrix:

For train set:

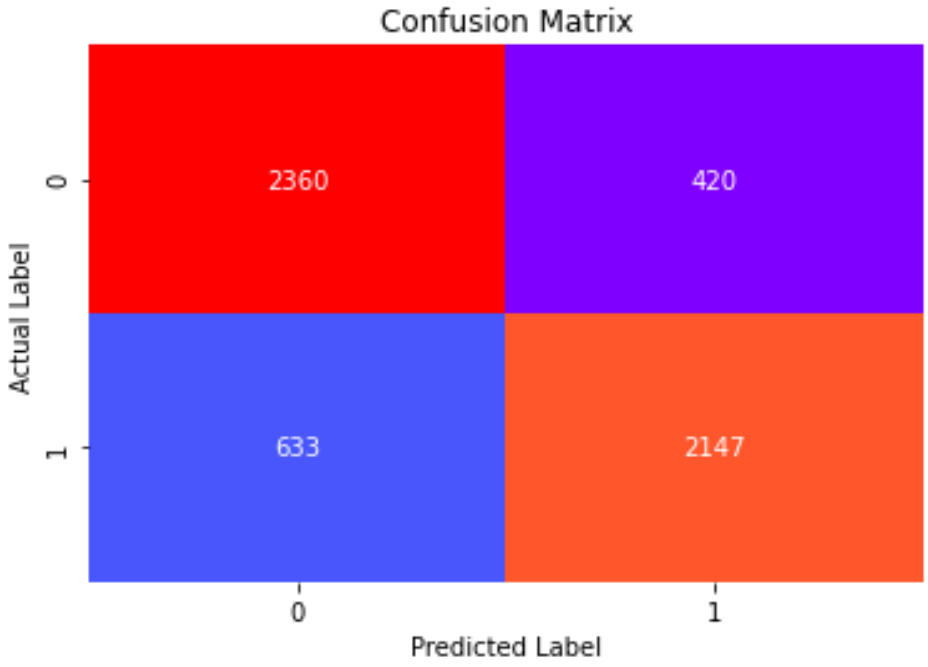

Fig.7

For test set:

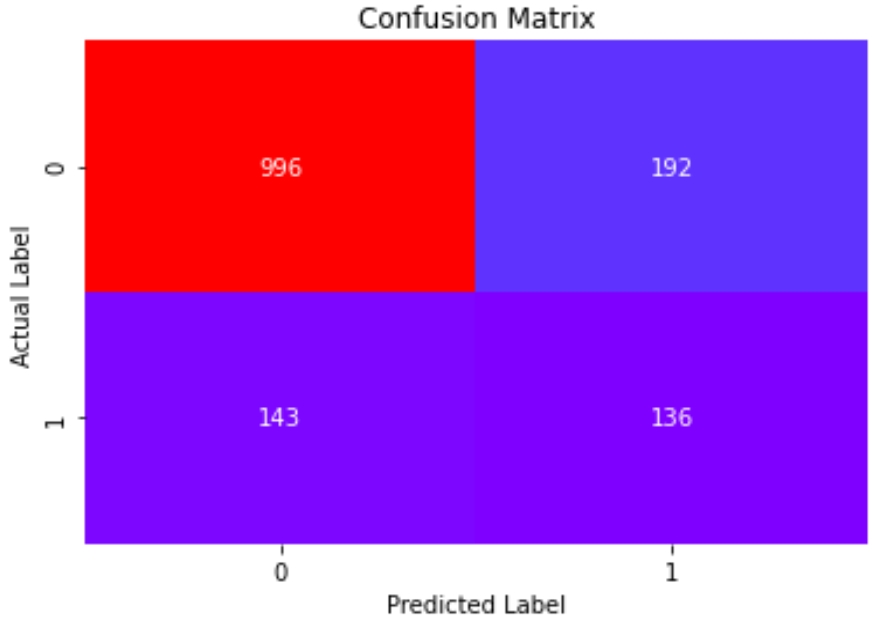

Fig.8

- Total no of correct prediction=136+996
- Total no of incorrect prediction=192+143

Classification Report:

For train set:

	precision	recall	f1-score	support
0	0.79	0.85	0.82	2780
1	0.84	0.77	0.80	2780
accuracy			0.81	5560
macro avg	0.81	0.81	0.81	5560
weighted avg	0.81	0.81	0.81	5560

For test set:

	precision	recall	f1-score	support
0	0.87	0.84	0.86	1188
1	0.41	0.49	0.45	279
accuracy			0.77	1467
macro avg	0.64	0.66	0.65	1467
weighted avg	0.79	0.77	0.78	1467

1. 49% of customers are correctly identified as those customers who have taken product.

AUC and ROC-curve:

For train set:

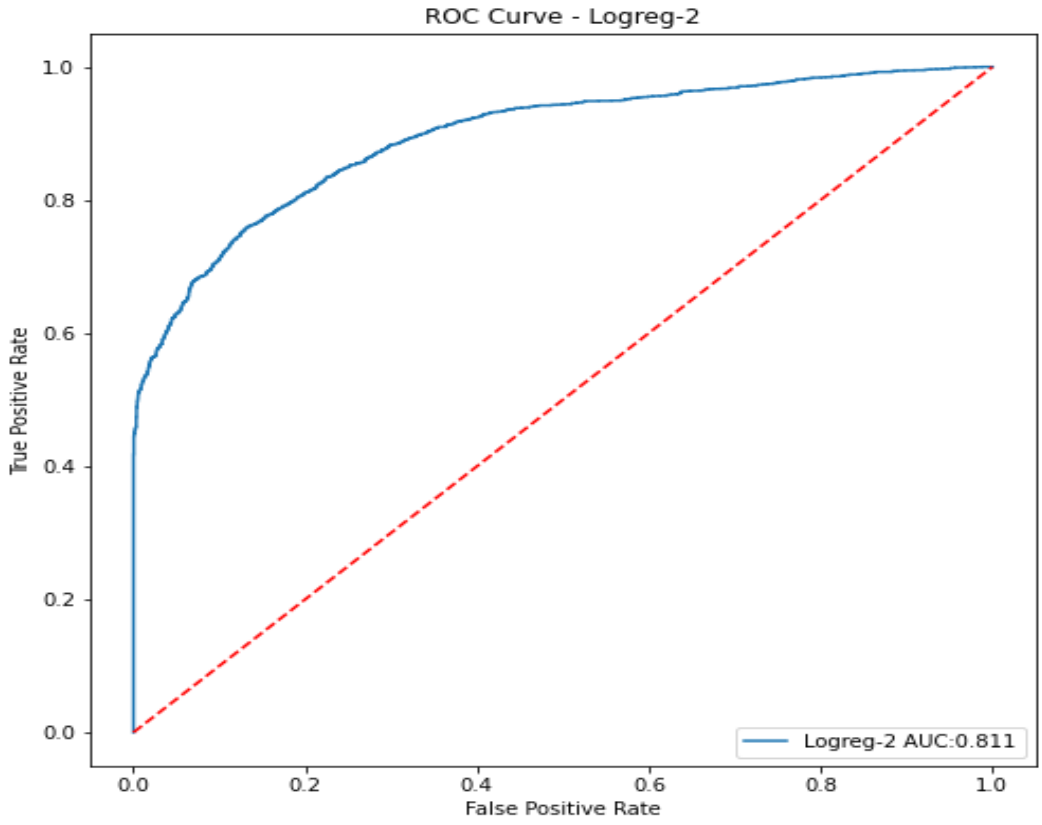

Fig.9

For test set:

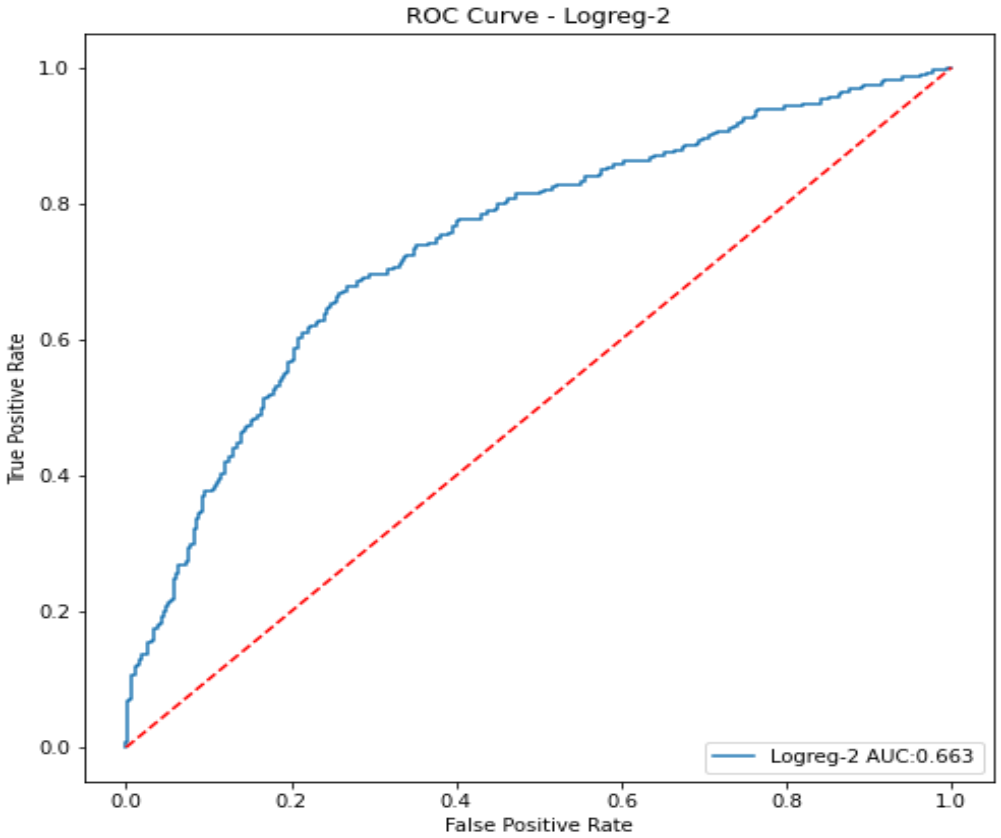

Fig.10

- Poor performance on test set: over fitting problem.

Model 3: Decision Tree with hyper parameter tuning:

1. First we have to import DecisionTreeClassifier from sklearn.tree and GridSearchCV from skearn.model_selection.
2. Parameters are defined as follows:

```
parameters = {'criterion':['gini','entropy'],
      'max_depth':[2,5,10,15],
      'min_samples_split':[2,10,15,20,25,30,60,80,100],
     'min_samples_leaf':[1,7,10,15,20,33],
     'min_impurity_decrease':[0.0001,0.001]}
grid = GridSearchCV(DT1,param_grid = parameters,cv=10,verbose=1,n_jobs=-1)
```

3. After that we have to build logistic regression model and we have to pass the parameter grid into GridSearchCV. Next, we have to do model fitting.

```
grid = GridSearchCV(DT1,param_grid = parameters,cv=10,verbose=1,n_jobs=-1)

DT1=grid.fit(X_train,Y_train.values.ravel())
```

4. Next, we can figure out best parameter for model.
5. Predicting probabilities and probability classes for train and test set.
6. Calculating performance matrices.

Accuracy score:

For train set:

```
0.9820863309352518
```

For test set:

```
0.8336741649625086
```

Confusion Matrix:

For train set:

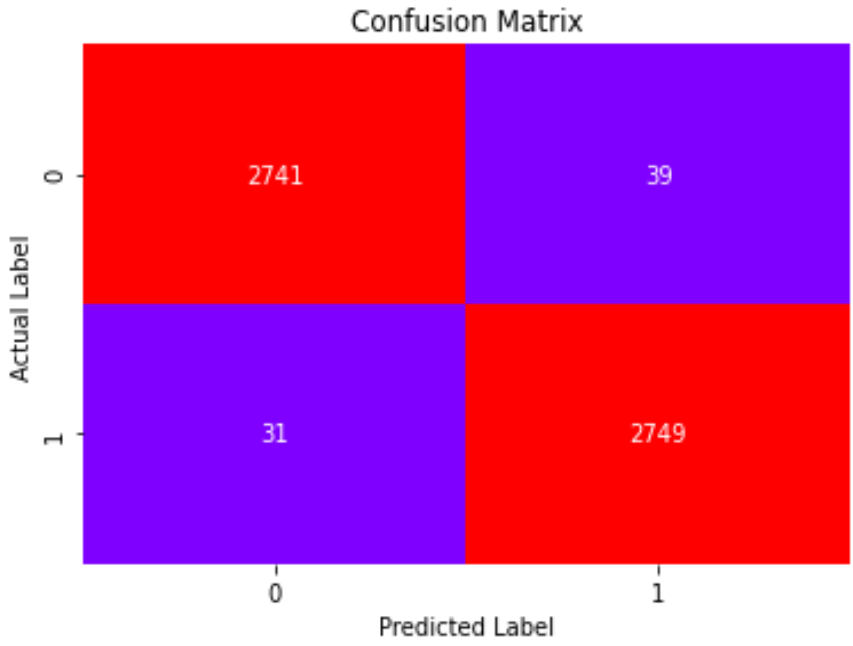

Fig.11

For test set:

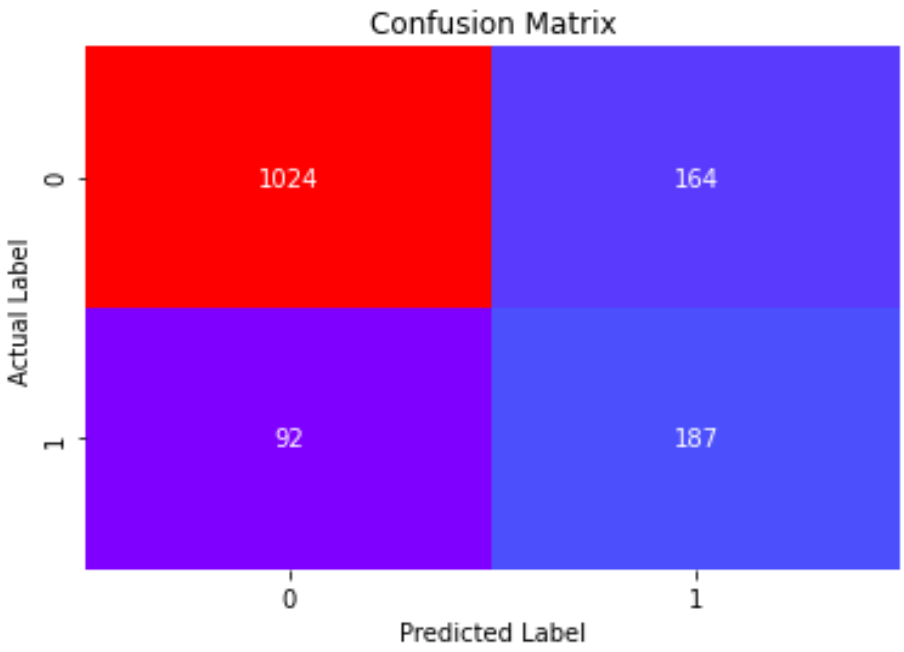

Fig.12

- Total no of correct prediction=187+1024
- Total no of incorrect prediction=164+92

Classification Report:

For train set:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	2780
1	0.99	0.99	0.99	2780
accuracy			0.99	5560
macro avg	0.99	0.99	0.99	5560
weighted avg	0.99	0.99	0.99	5560

For test set:

	precision	recall	f1-score	support
0	0.92	0.86	0.89	1188
1	0.53	0.67	0.59	279
accuracy			0.83	1467
macro avg	0.73	0.77	0.74	1467

```
weighted avg       0.84      0.83      0.83      1467
```

- 67 % customers are correctly identified as those customers who have Taken product.

AUC and ROC Curve:

For train set:

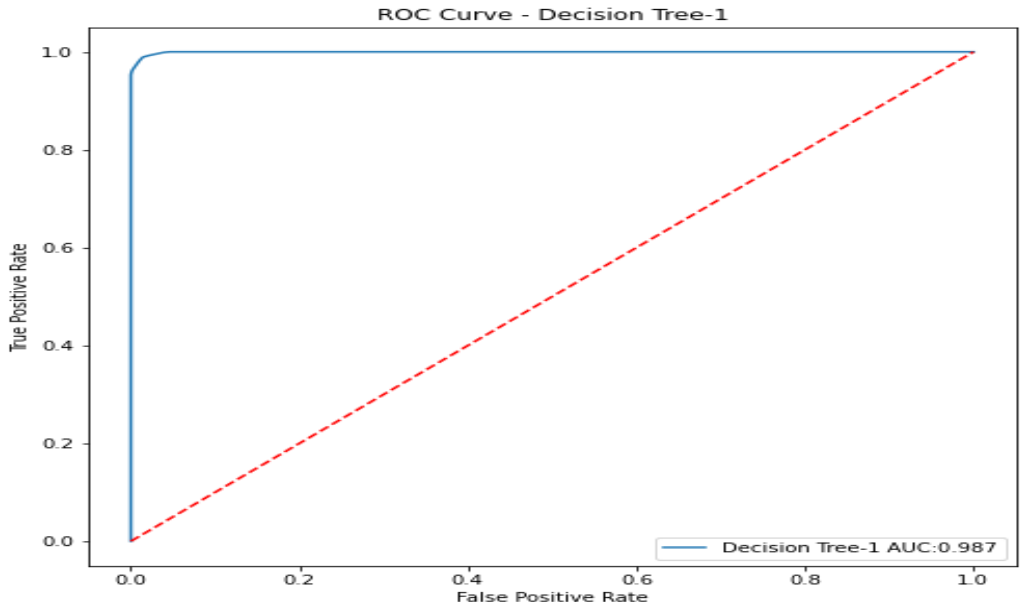

Fig.13

For test set:

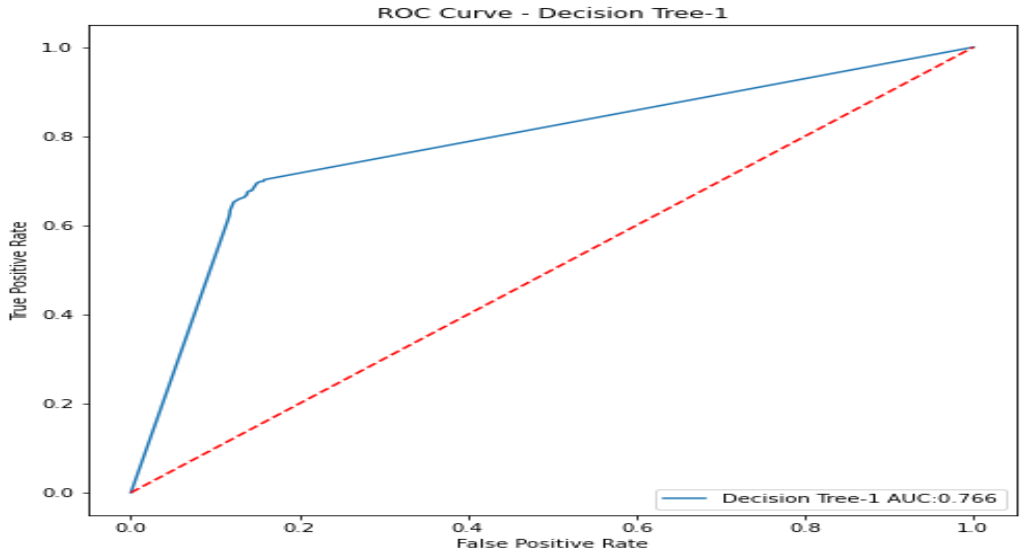

Fig.14

- AUC score is poor on test set: over fiting problem.

Model4: Random Forest with hyper parameter tuning:

1. First we have to import RandomForestClassifier from sklearn.ensemble and GridSearchCV from skearn.model_selection.

```
parameters = {'n_estimators':[60,80,100,200,300],

        'min_samples_split':[1,2,3],

        'min_samples_leaf':[1,5,10],

        'max_features': [2,3,4],

        'min_impurity_decrease':[0.00001,0.0001,0.001]}
```

2. After that we have to build logistic regression model and we have to pass the parameter grid into GridSearchCV. Next, we have to do model fitting.

```
grid = GridSearchCV(estimator=RF1,param_grid=parameters,cv=10,n_jobs=-1,verbose=1)

grid.fit(X_train,Y_train.values.ravel())
```

3. Next, we can figure out best parameter for model.

```
{'max_features': 4,
 'min_impurity_decrease': 1e-05,
 'min_samples_leaf': 1,'min_samples_split': 2,
 'n_estimators': 300}
```

4. Predicting probabilities and probability classes for train and test set.
5. Calculating performance matrices.

Accuracy score:

For train set:

```
1.0
```

For test set:

```
0.8657123381049762
```

Confusion Matrix:

For train set:

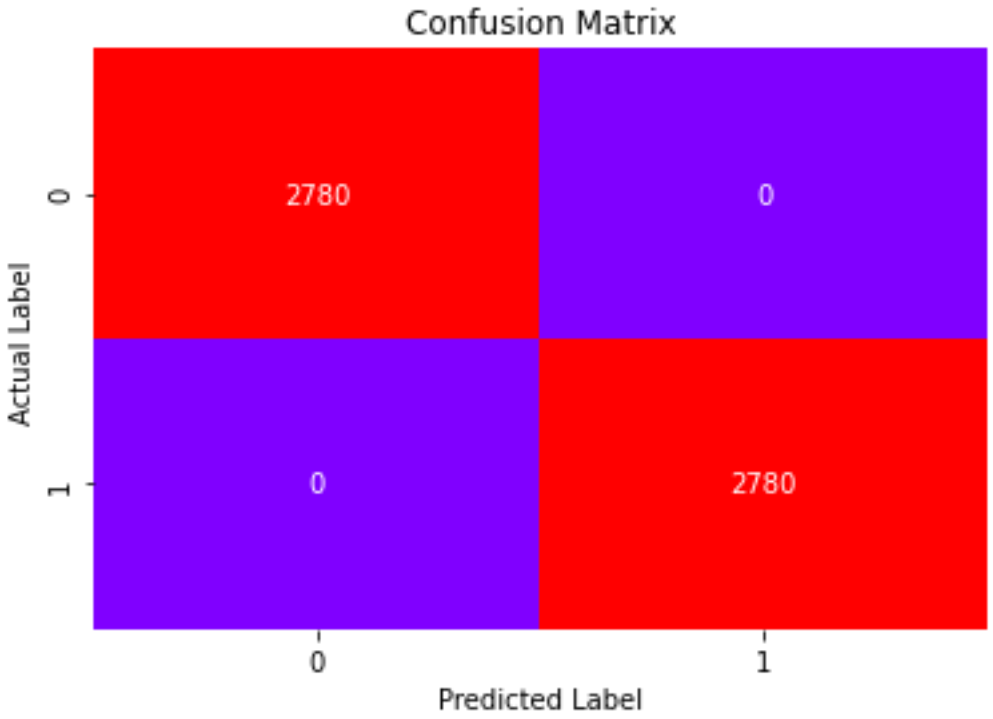

Fig.15

For test set:

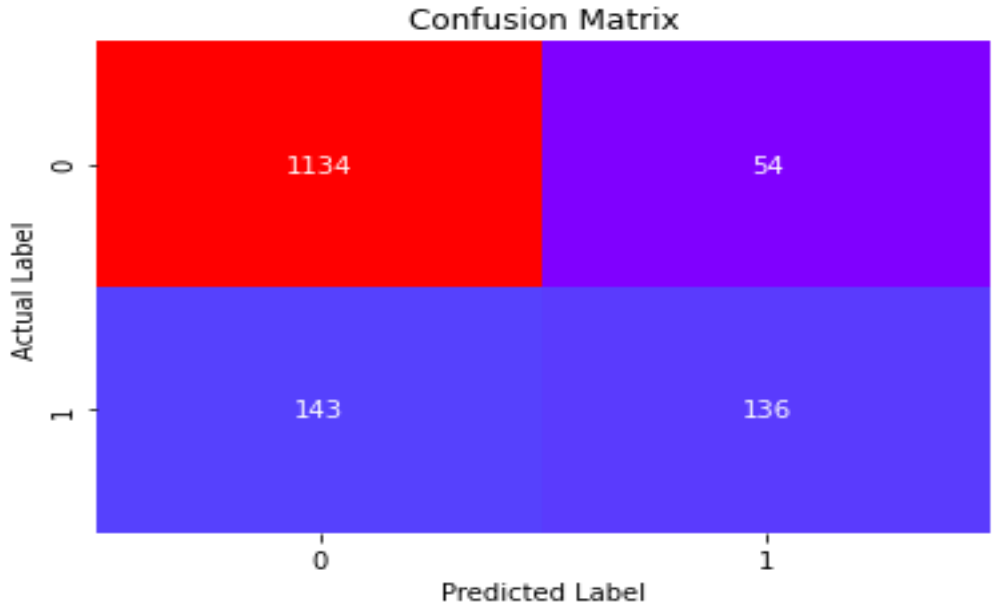

Fig.16

- Total no of correct prediction=136+1134
- Total no of incorrect prediction=143+54

Classification Report:

For train set:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2780
1	1.00	1.00	1.00	2780
accuracy			1.00	5560
macro avg	1.00	1.00	1.00	5560
weighted avg	1.00	1.00	1.00	5560

For test set:

	precision	recall	f1-score	support
0	0.89	0.95	0.92	1188
1	0.70	0.49	0.58	279
accuracy			0.87	1467
macro avg	0.79	0.72	0.75	1467
weighted avg	0.85	0.86	0.85	1467

- 49 % of customers are correctly identified as the customers who have taken product.

AUC and ROC-Curve:

For train set:

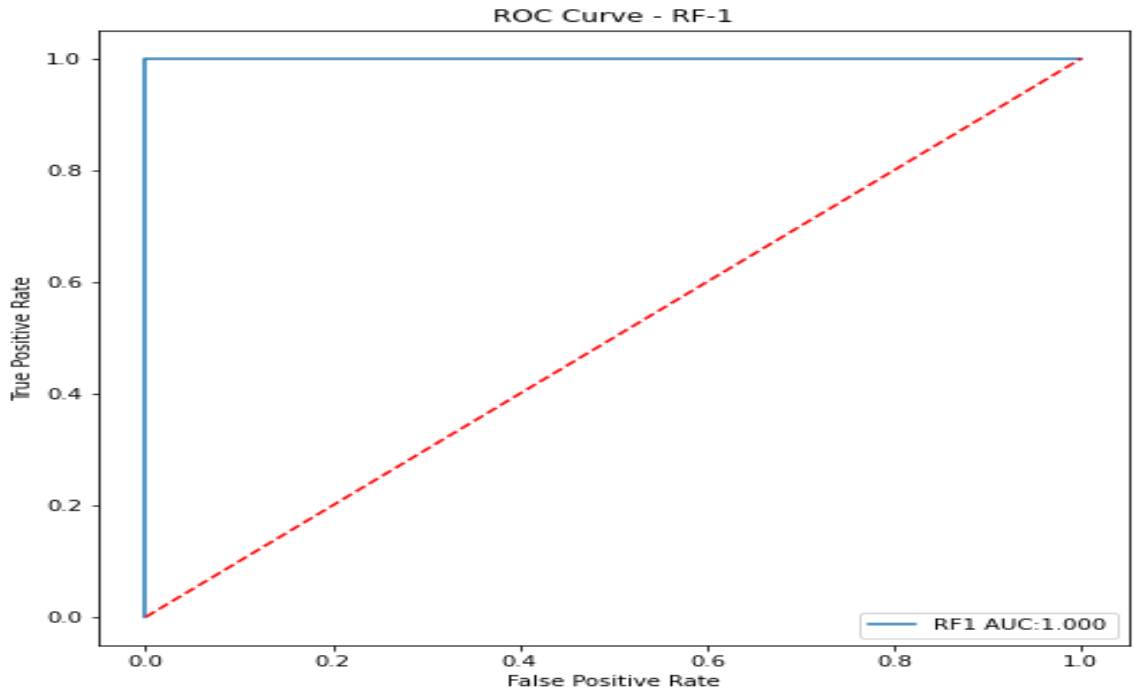

Fig.17

For test set:

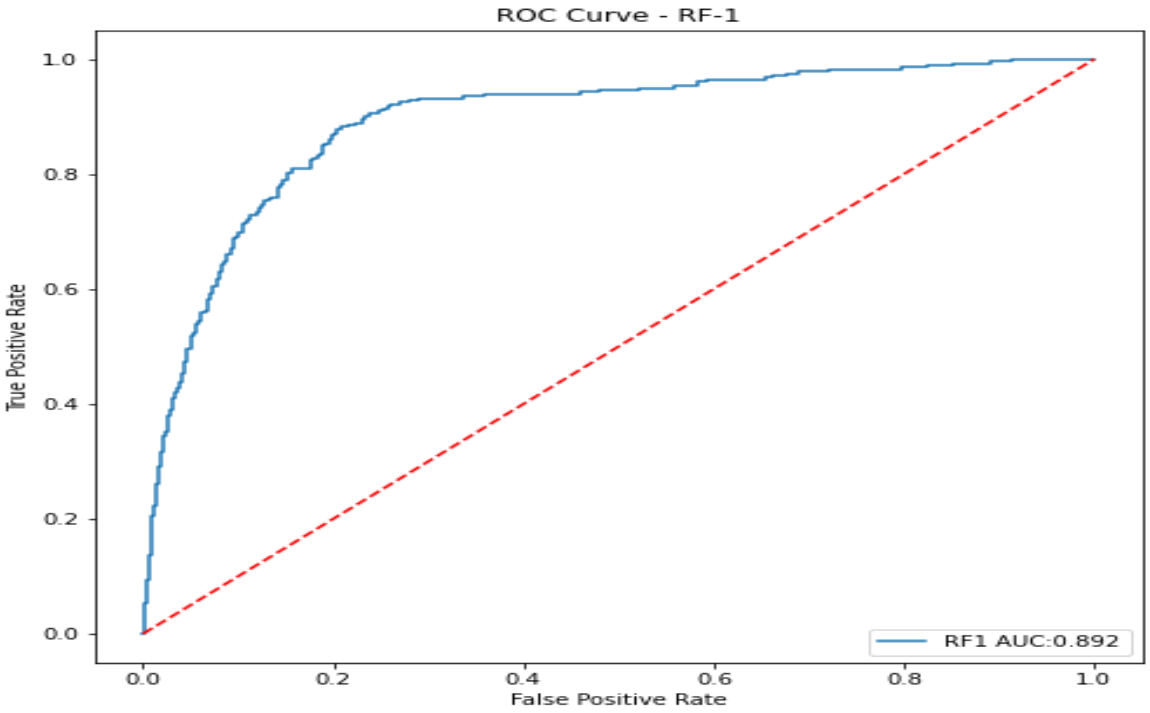

Fig.18

Looking at the Features Important:

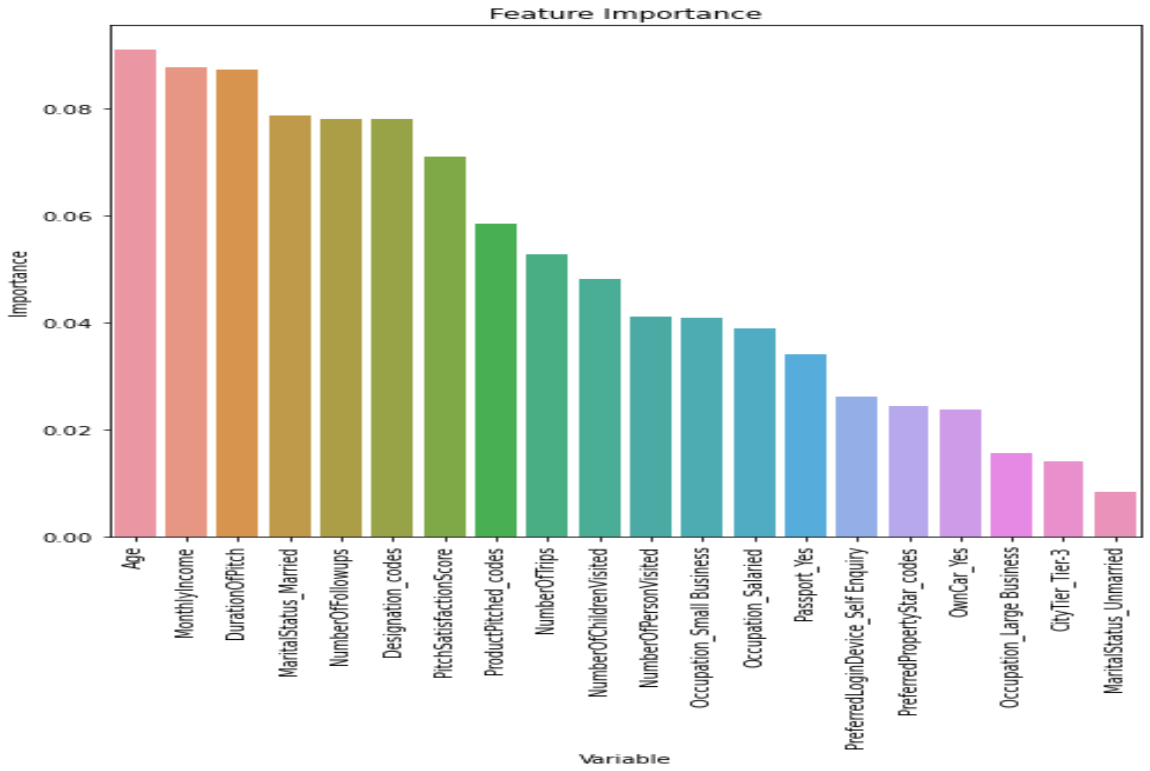

Fig.19

Observations:

Features which have longer bar, are most significant features. Age, MonthlyIncome,DurationOfpitch,Marital_status_married,NumberOfFollowups, Designation_code has significant impact on ProdTaken(dependent variable).The most important features among all is Age and MonthlyIncome.

Model5: Gradient Boosting Model:

1. First we have to import from GradientBoostingClassifier from sklearn.ensemble and GridSearchCV from skearn.model_selection.

2. Parameters are defined as follows.

```
params = {'loss':['deviance','exponential'],
        'learning_rate':[0.15,0.17,0.20],
        'n_estimators':[300,500,700]}
```

3. After that we have to build logistic regression model and we have to pass the parameter grid into GridSearchCV. Next, we have to do model fitting.

```
grid = GridSearchCV(estimator=GB1,param_grid=params,cv=10,n_jobs=-1,verbose=1)
grid.fit(X_train,Y_train.values.ravel())
GB1 = grid.best_estimator_
GB1.fit(X_train,Y_train.values.ravel())
```

4. Next, we can figure out best parameter for model.

5. Predicting the probability and probability class for train set.

6. Calculating model performance matrices for train and test set.

Accuracy Score:

For train set:

```
0.9985611510791367
```

For test set:

```
0.896387184730743
```

Confusion Matrix:

For train set:

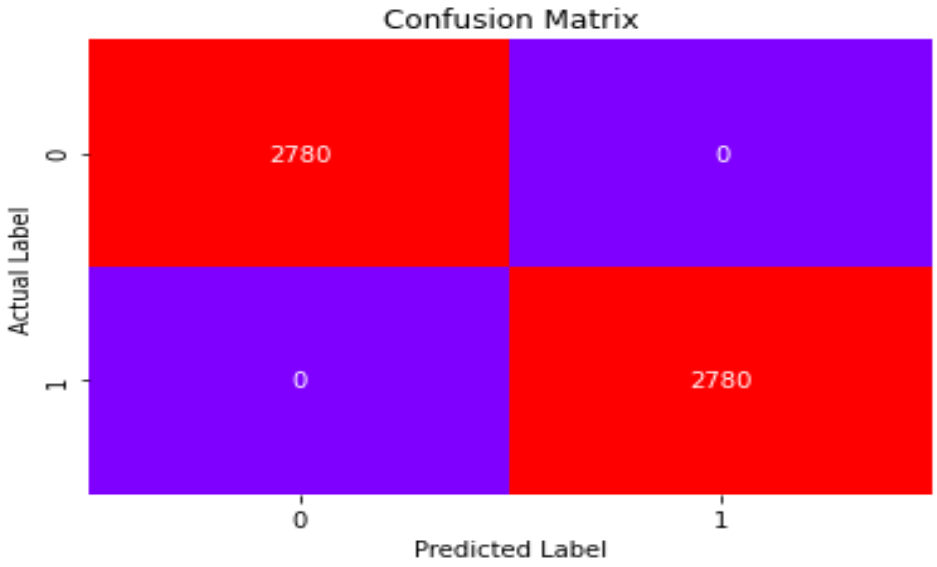

Fig.20

For test set:

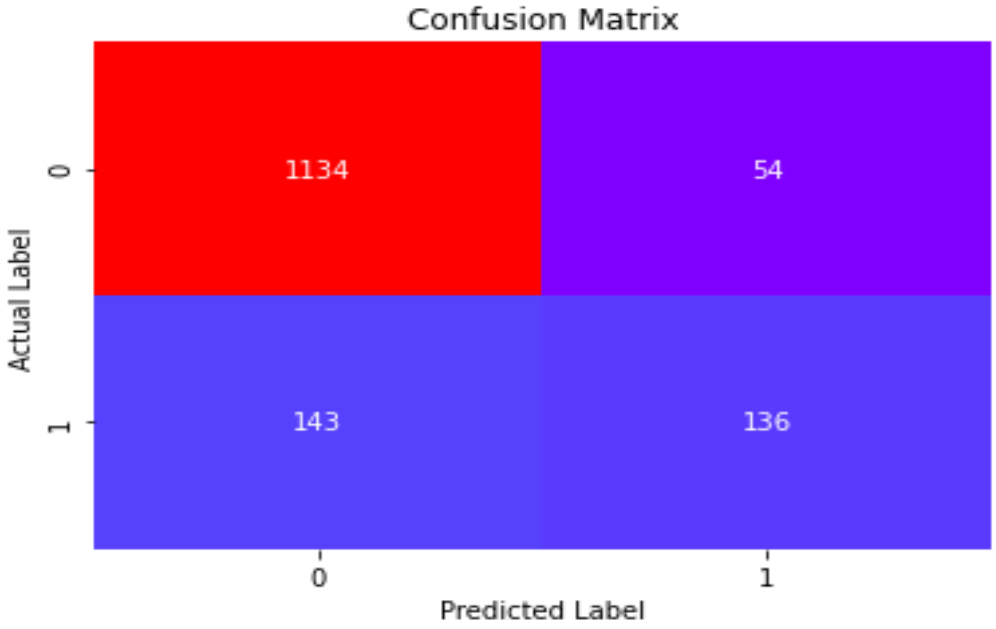

Fig.21

- Total no of correct prediction=136+1134
- Total no of incorrect prediction=143+54

Classification Report:

For train set:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2780

1	1.00	1.00	1.00	2780
accuracy			1.00	5560
macro avg	1.00	1.00	1.00	5560
weighted avg	1.00	1.00	1.00	5560

For test set:

	precision	recall	f1-score	support
0	0.89	0.95	0.92	1188
1	0.72	0.49	0.58	279
accuracy			0.87	1467
macro avg	0.80	0.72	0.75	1467
weighted avg	0.86	0.87	0.86	1467

- 49 % of customers are correctly identified as the customers who have tak en product.

AUC and ROC Curve:

For train set:

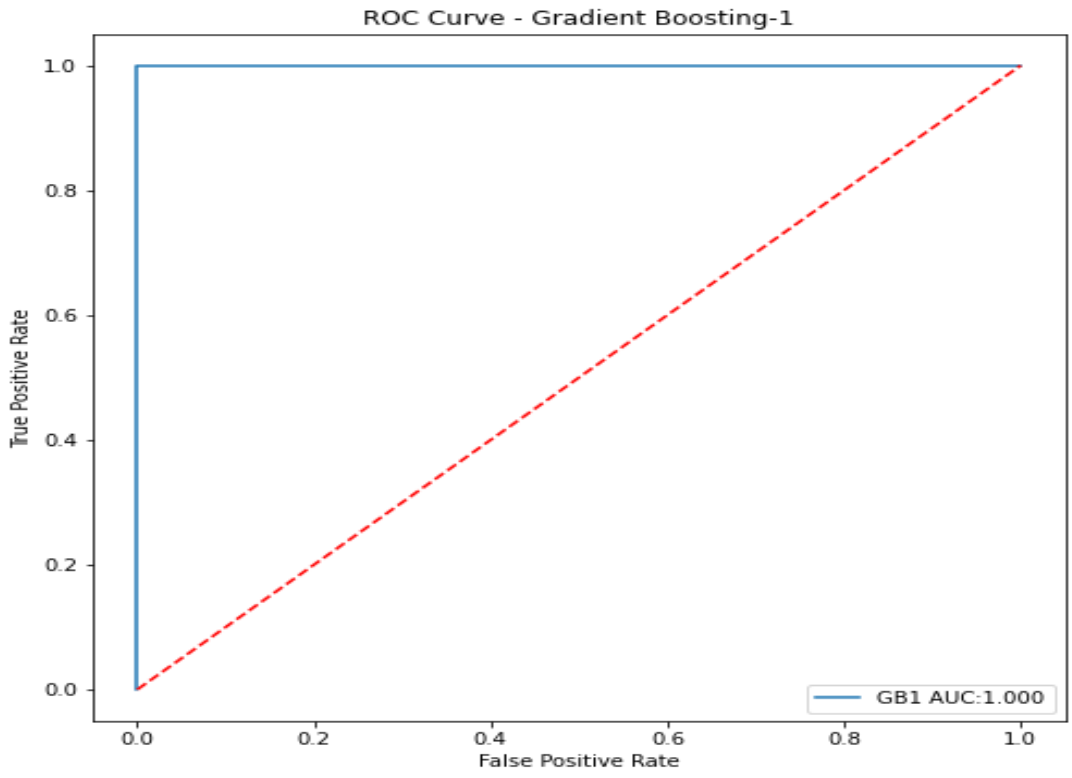

Fig.22

For test set:

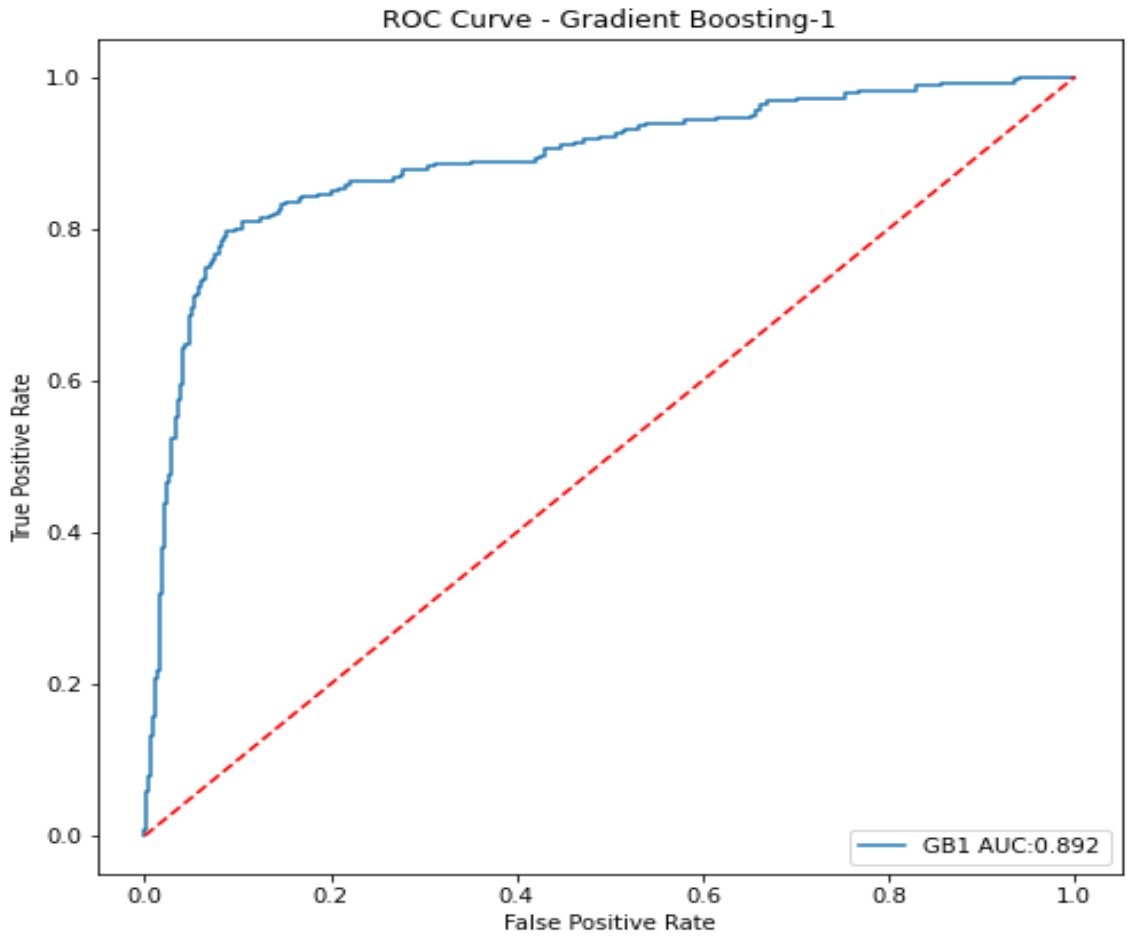

Fig.23

Looking at the features Importance:

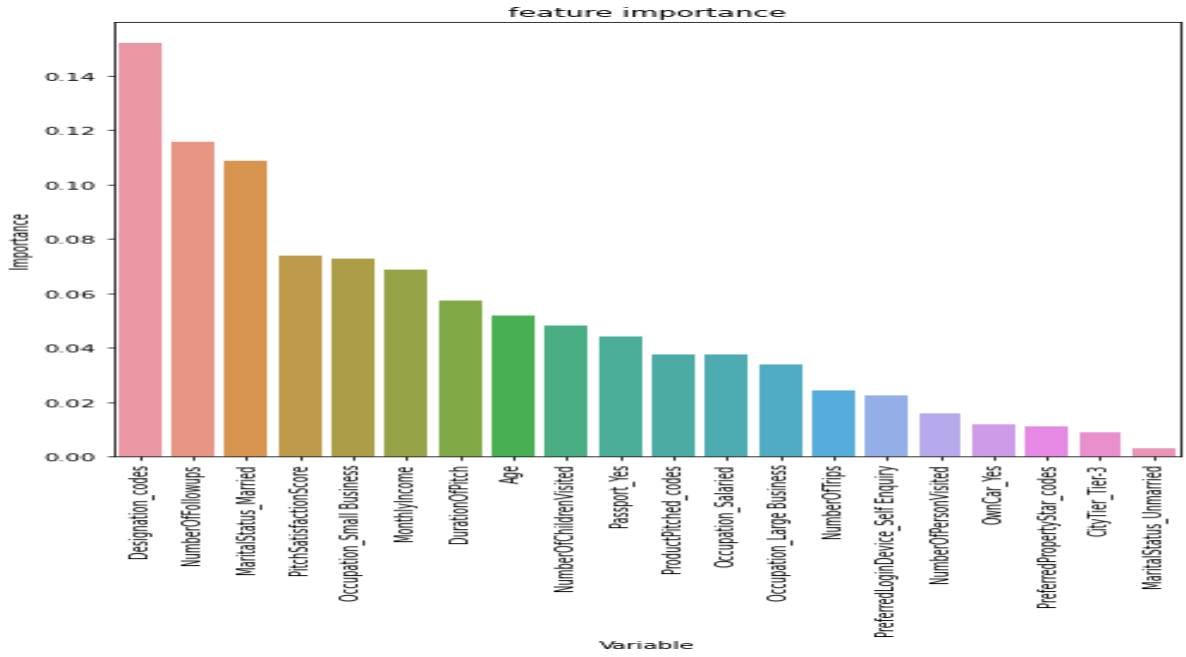

Fig.24

Observation:

In gradient boosting model the most significant feature is Designation_code followed by NumberOfFollowUps followed by Martial_Staues_Married.

Model6: KNN(K Nearest Neighbour) :

1. For building KNN model we have to scale train and test independent set.

2. First we have to import from KNeighborsClassifier from ssklearn.neighbors and GridSearchCV from skearn.model_selection.

3. Parameters are defined as follows.

```
params = {'n_neighbors':range(2,11),
        'p':[2,3],
        'metric':['manhattan','chebyshev','minkowski']}
```

4. After that we have to build logistic regression model and we have to pass the parameter grid into GridSearchCV. Next, we have to do model fitting.

```
grid = GridSearchCV(estimator=KNN1,param_grid=params,cv=10,n_jobs=-1,verbose=1)
grid.fit(X_train_std,Y_train.values.ravel())
KNN1 = grid.best_estimator_
KNN1.fit(X_train_std,Y_train.values.ravel())
```

5. Next, we can figure out best parameter for model.

```
{'metric': 'manhattan', 'n_neighbors': 2, 'p': 2}
```

6. Predicting the probability and probability class for train set.

7. Calculating model performance matrices for train and test set.

Accuracy Score:

For train set:

```
0.9739208633093526
```

For test set:

```
0.8657123381049762
```

Confusion Matrix:

For train set:

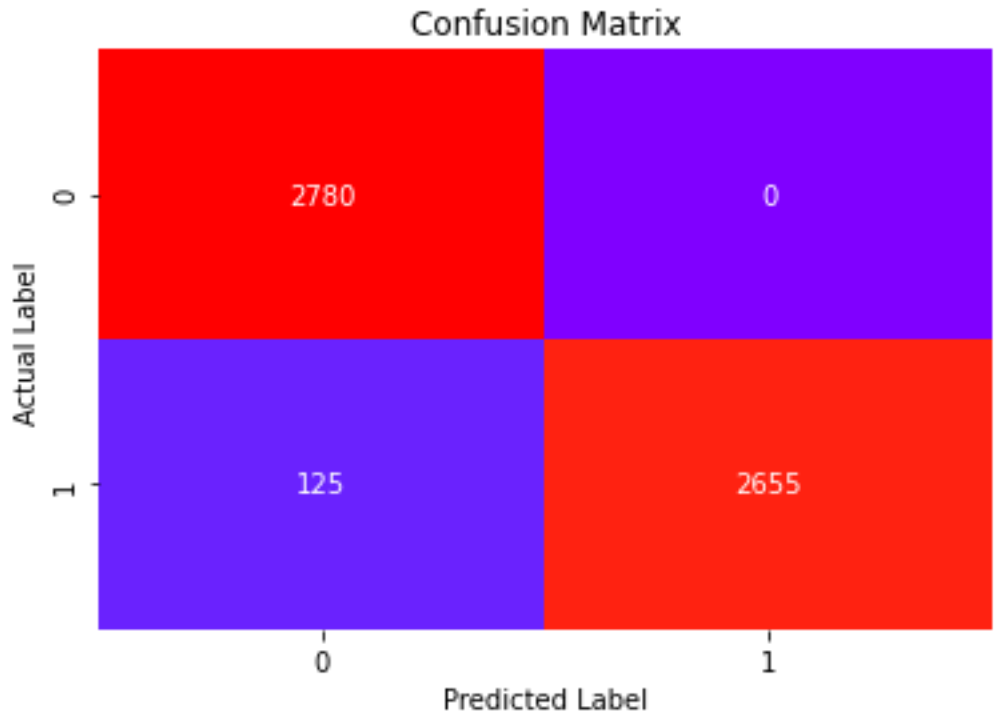

Fig: 25

For test set:

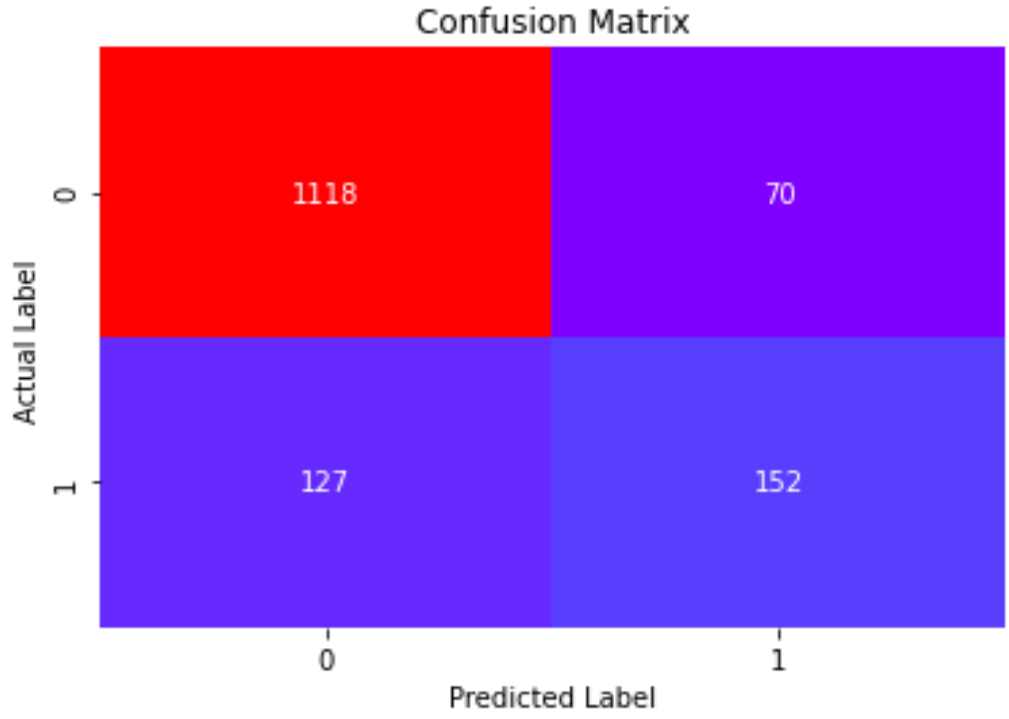

Fig.26

- Total no of correct prediction=201+1001
- Total no of correct prediction=78+187

Classification Report:

For train set:

	precision	recall	f1-score	support
0	0.98	0.97	0.97	2780
1	0.97	0.98	0.97	2780
accuracy			0.97	5560
macro avg	0.97	0.97	0.97	5560
weighted avg	0.97	0.97	0.97	5560

For test set:

	precision	recall	f1-score	support
0	0.90	0.94	0.92	1188
1	0.68	0.54	0.61	279
accuracy			0.87	1467
macro avg	0.79	0.74	0.76	1467
weighted avg	0.86	0.87	0.86	1467

- 54 % customers are correctly identified as the customers who have taken product.

AUC and ROC Curve:

For train set:

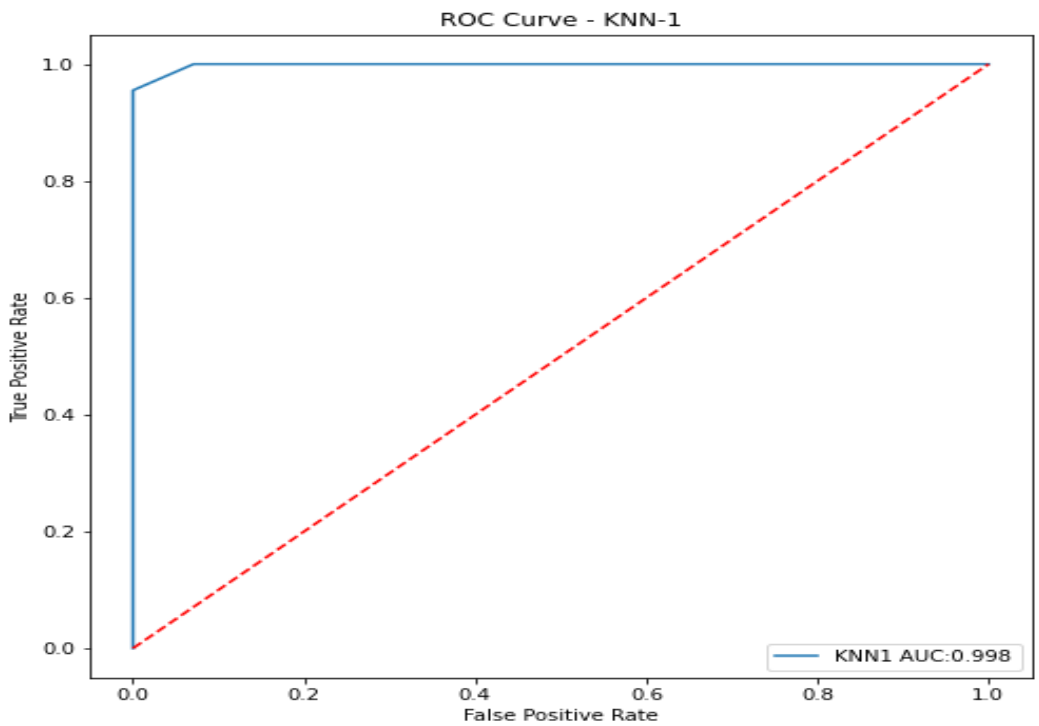

Fig.27

For test set:

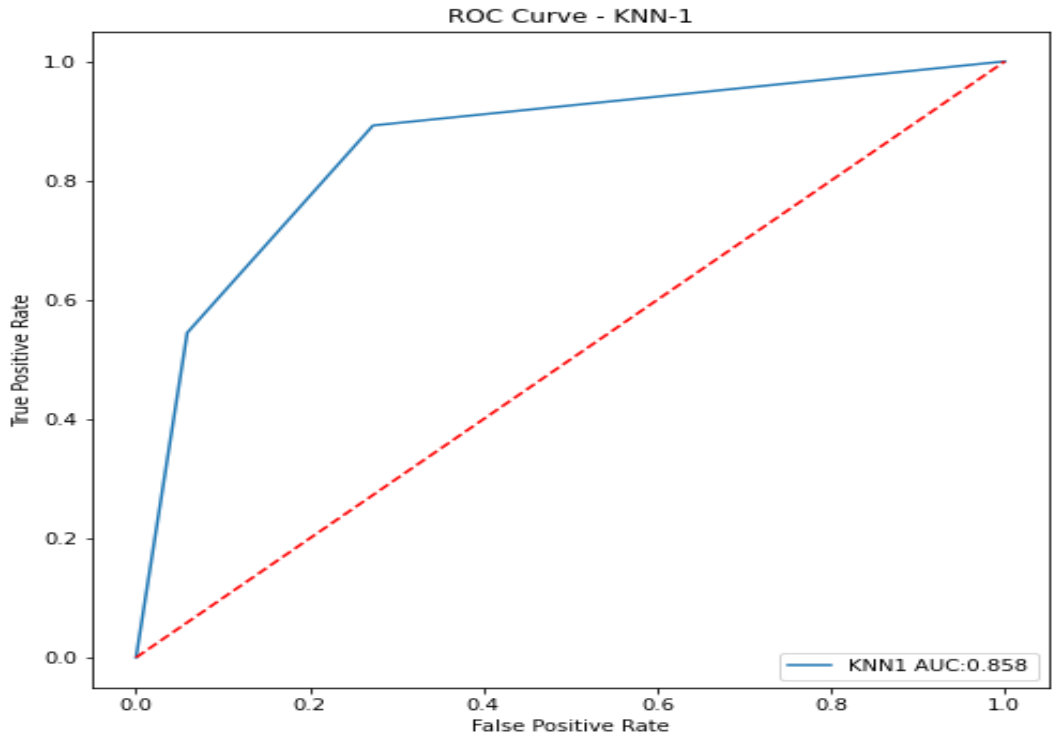

Fig.28

Model7: Neural Network:

1. For building NN model we have to scale train and test independent set.

2. First we have to import from MLPClassifier from ssklearn.neural_network and GridSearchCV from skearn.model_selection.

3. Parameters are defined as follows.

```
param_grid = {
    'activation':['logistic', 'tanh', 'relu'],
    'hidden_layer_sizes': [100,200,300,500],
    'max_iter': [5000],
    'solver': ['sgd','adam'],
    'tol': [0.001],
}
```

4. After that we have to build logistic regression model and we have to pass the parameter grid into GridSearchCV. Next, we have to do model fitting.

```
grid = GridSearchCV(estimator=NN1,param_grid=params,cv=3n_jobs=-1,verbose=1)

grid.fit(X_train_std,Y_train.values.ravel())

NN1 = grid.best_estimator_

NN1.fit(X_train_std,Y_train.values.ravel())
```

5. Next, we can figure out best parameter for model.

```
{'activation': 'tanh',
 'hidden_layer_sizes': 300,
 'max_iter': 5000,
 'solver': 'adam',
 'tol': 0.001}
```

6. Predicting the probability and probability class for train set.

7. Calculating model performance matrices for train and test set.

Accuracy Score:

For train set:

```
0.9973021582733813
```

For test set:

```
0.6046353101567825
```

Confusion Matrix:

For train set:

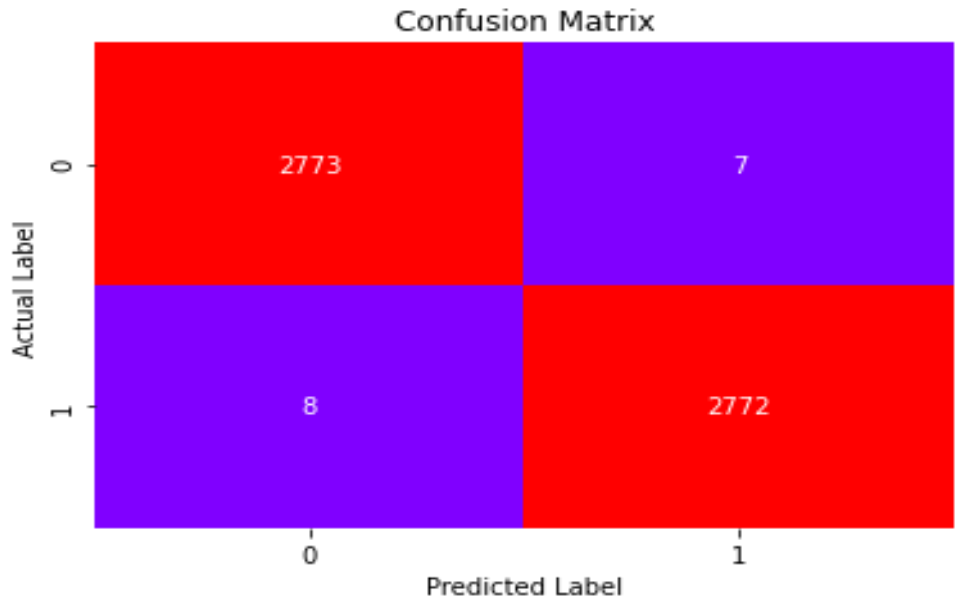

Fig.28

For test set:

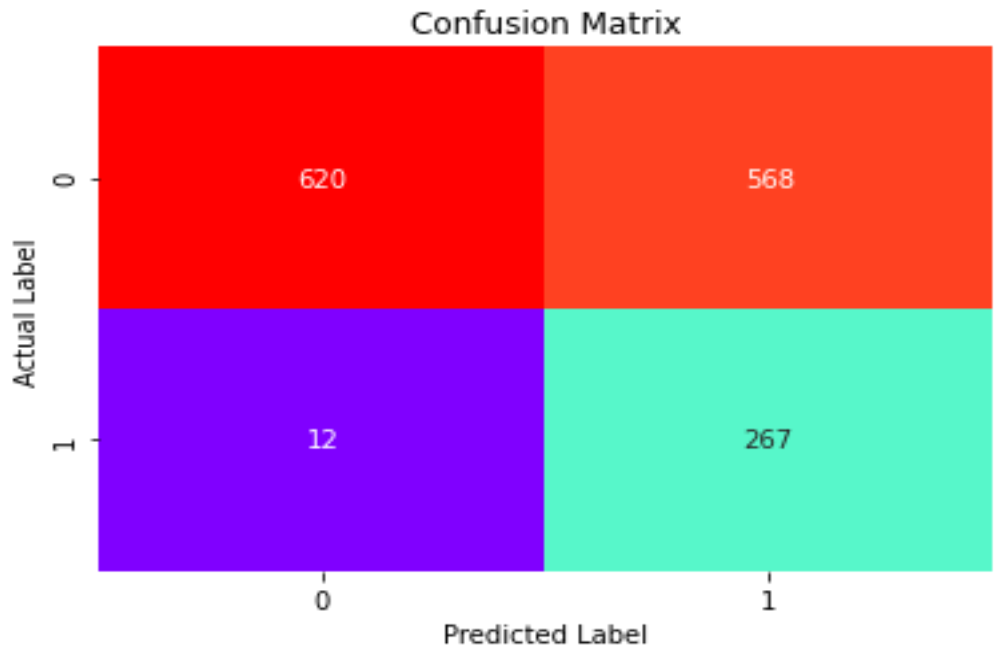

Fig.29

Classification Report:

For train set:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2780
1	1.00	1.00	1.00	2780
accuracy			1.00	5560
macro avg	1.00	1.00	1.00	5560
weighted avg	1.00	1.00	1.00	5560

For test set:

	precision	recall	f1-score	support
0	0.98	0.52	0.68	1188
1	0.32	0.96	0.48	279
accuracy			0.60	1467
macro avg	0.65	0.74	0.58	1467
weighted avg	0.86	0.60	0.64	1467

- 96 % customers are correctly identified as the customers who have taken product (good recall).

AUC and ROC Curve:

For train set:

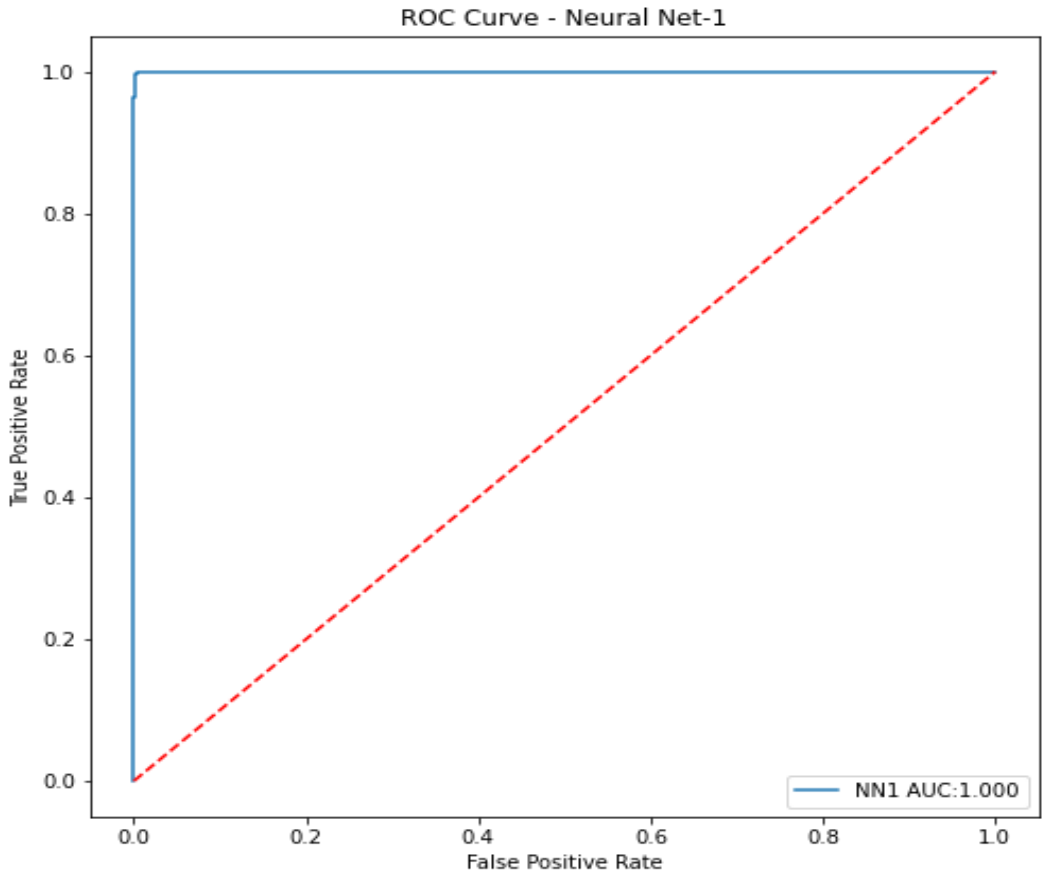

Fig.30

For test set:

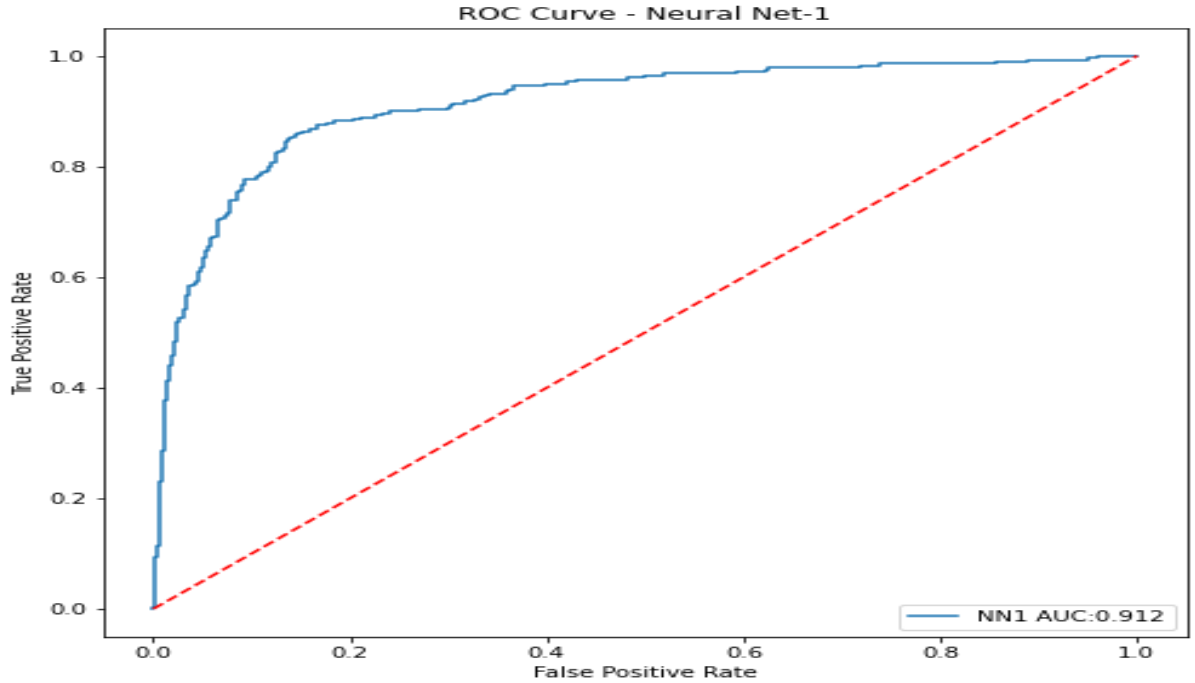

Fig.31

Model8: SVM(Support Vector Machine):

1. For building SVM model we have to scale train and test independent set.

2. First we have to import fromSVC from ssklearn.svm and GridSearchCV from skearn.model_selection.

3. Parameters are defined as follows.

```
params = {'C':[0.01,1,10,20],
        'kernel':['linear','poly','rbf','sigmoid'],
        'probability':[True]}
```

4. After that we have to build logistic regression model and we have to pass the parameter grid into GridSearchCV. Next, we have to do model fitting.

```
grid = GridSearchCV(estimator=SVC1,param_grid=params,cv=3,n_jobs=-1,verbose=1)
grid.fit(X_train_std,Y_train.values.ravel())
SVC1 =grid.best_estimator_
SVC1.fit(X_train_std,Y_train.values.ravel())
```

5. Next, we can figure out best parameter for model.

```
{'C': 20, 'kernel': 'rbf', 'probability': True}
```

6. Predicting the probability and probability class for train set.

7. Calculating model performance matrices for train and test set.

Accuracy Score:

For train set:

```
0.9920863309352518
```

For test set:

```
0.7668711656441718
```

Confusion Matrix:

For train set:

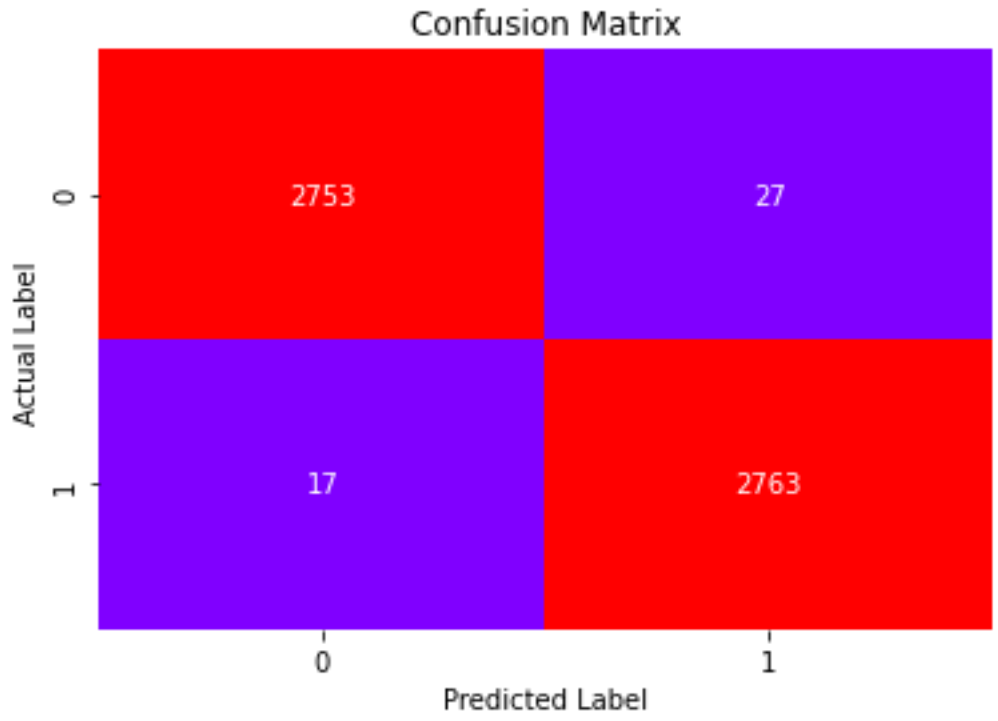

Fig.32

For test set:

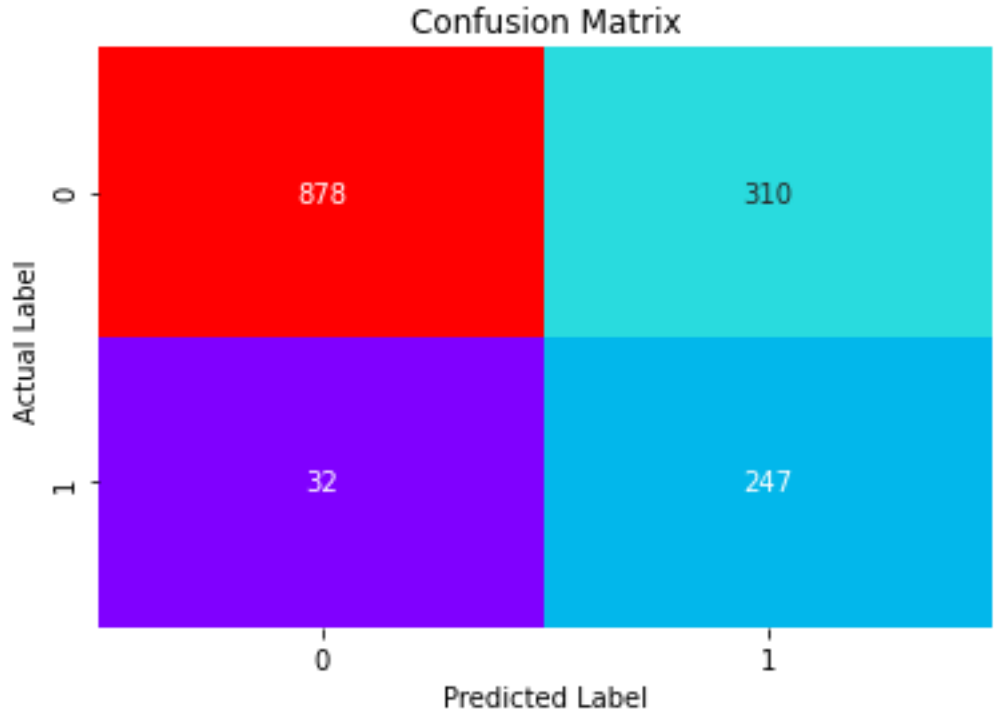

Fig.33

Classification Report:

For train set:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	2780
1	0.99	0.99	0.99	2780
accuracy			0.99	5560
macro avg	0.99	0.99	0.99	5560
weighted avg	0.99	0.99	0.99	5560

For test set:

	precision	recall	f1-score	support
0	0.96	0.74	0.84	1188
1	0.44	0.89	0.59	279
accuracy			0.77	1467
macro avg	0.70	0.81	0.71	1467
weighted avg	0.87	0.77	0.79	1467

AUC and ROC Curve:

For train set:

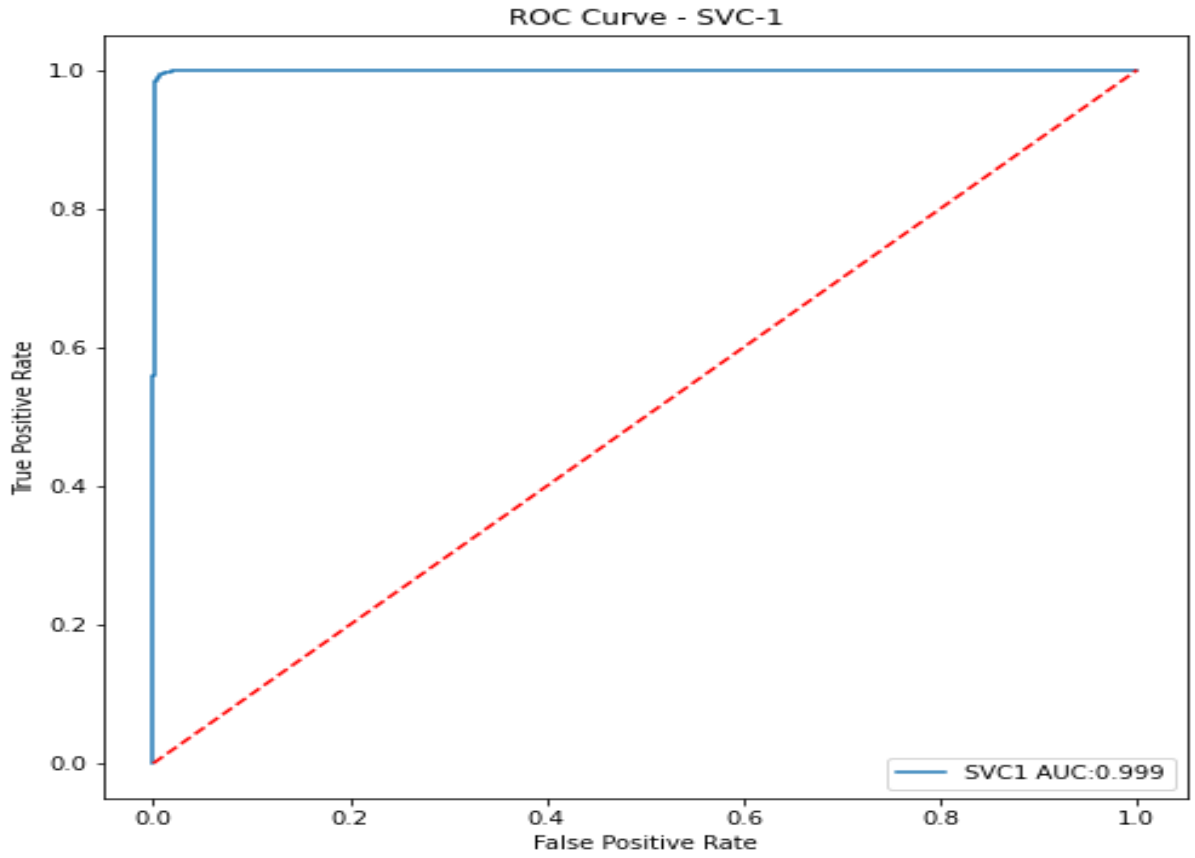

Fig.33

For test set:

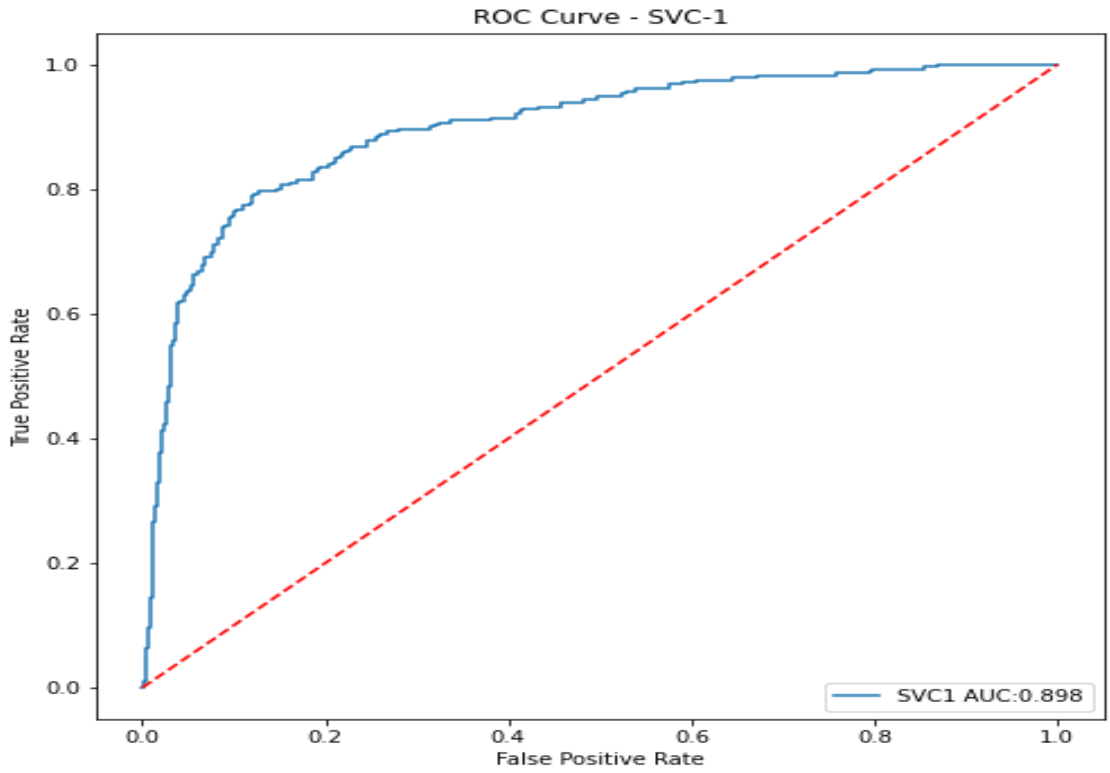

Fig.35

Model10: Naive Bayes Classifier:

1. First we have to import GaussianNB from sklearn.naive_bayes. In naïve bayes classifier we can't do hyper parameter tunning.

2. After that we have to build model and fit the model.

```
NB1 = GaussianNB()

NB1.fit(X_train,Y_train.values.ravel())
```

4. Predicting the probability and probability class for train set.

5. Calculating model performance matrices for train and test set.

Accuracy Score:

For train set:

```
0.7528776978417266
```

For test set:

```
0.7007498295841854
```

Confusion Matrix:

For train set:

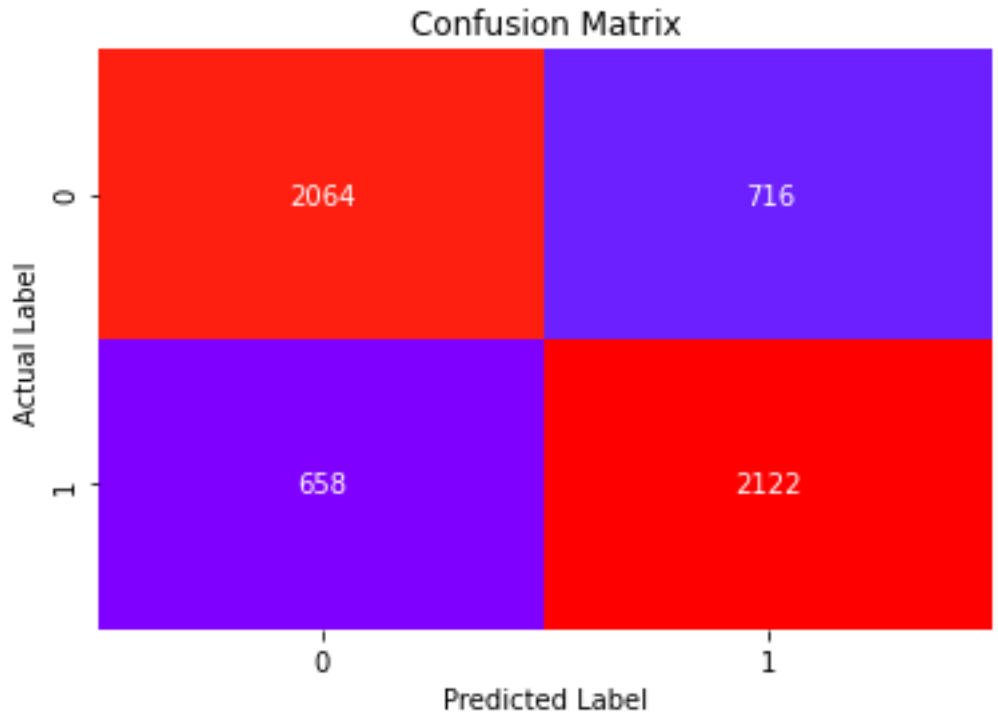

Fig.36

For test set:

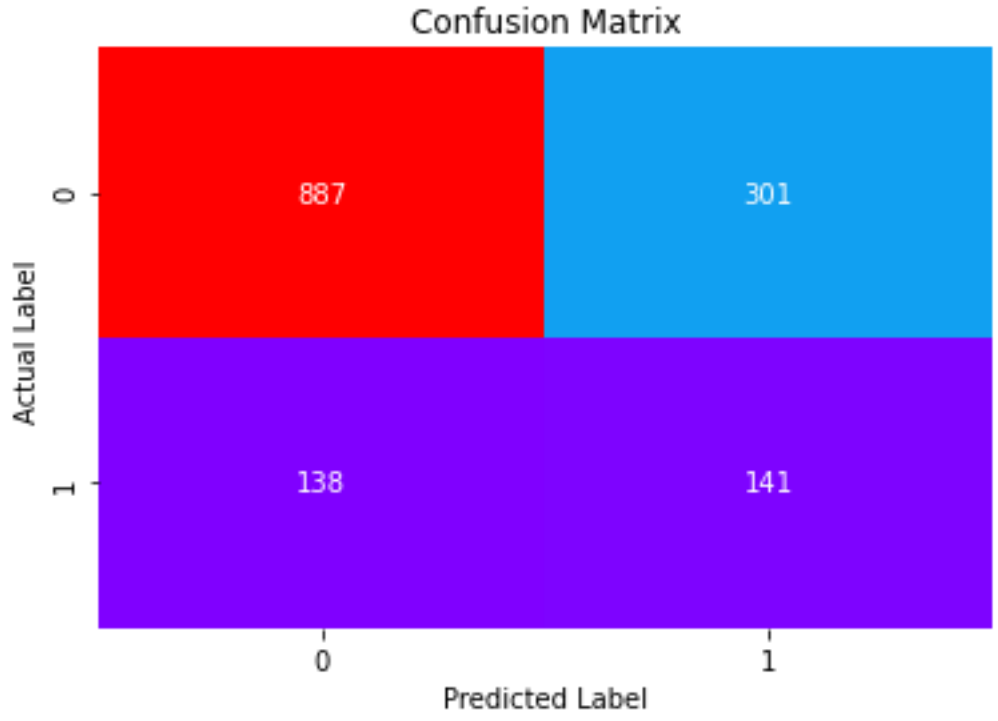

Fig.37

Classification Report:

For train set:

	precision	recall	f1-score	support
0	0.76	0.74	0.75	2780
1	0.75	0.76	0.76	2780
accuracy			0.75	5560
macro avg	0.75	0.75	0.75	5560
weighted avg	0.75	0.75	0.75	5560

For test set:

	precision	recall	f1-score	support
0	0.87	0.75	0.80	1188
1	0.32	0.51	0.39	279
accuracy			0.70	1467
macro avg	0.59	0.63	0.60	1467
weighted avg	0.76	0.70	0.72	1467

AUC and ROC Curve:

For train set:

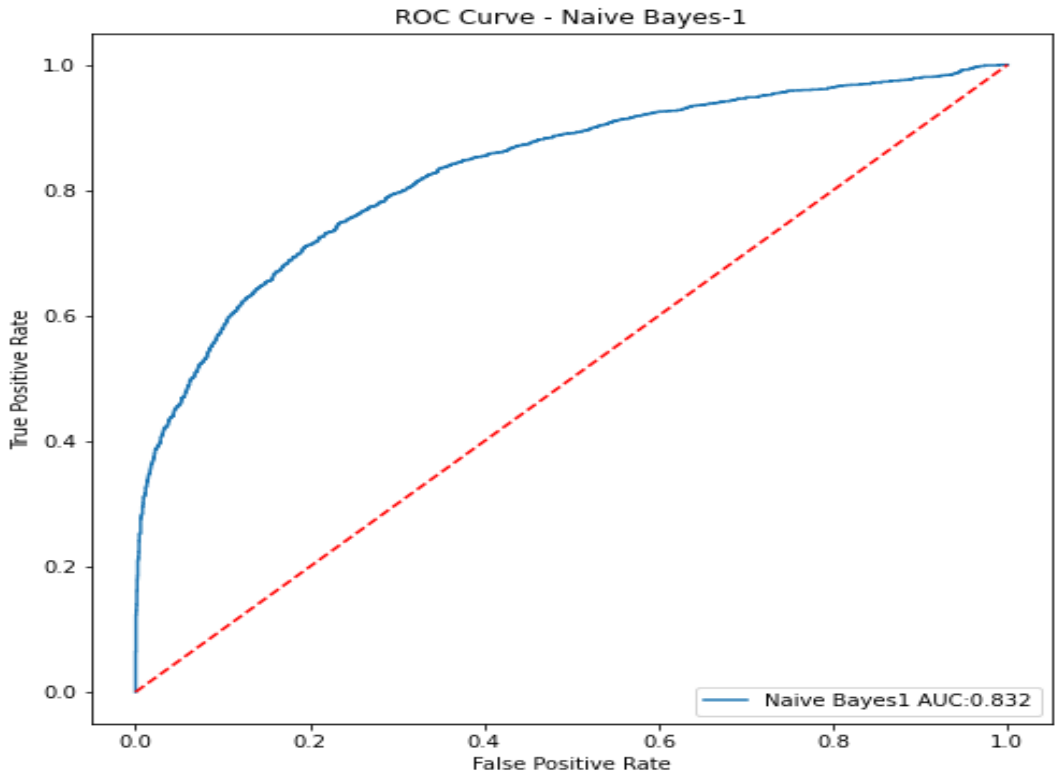

Fig.38

For test set:

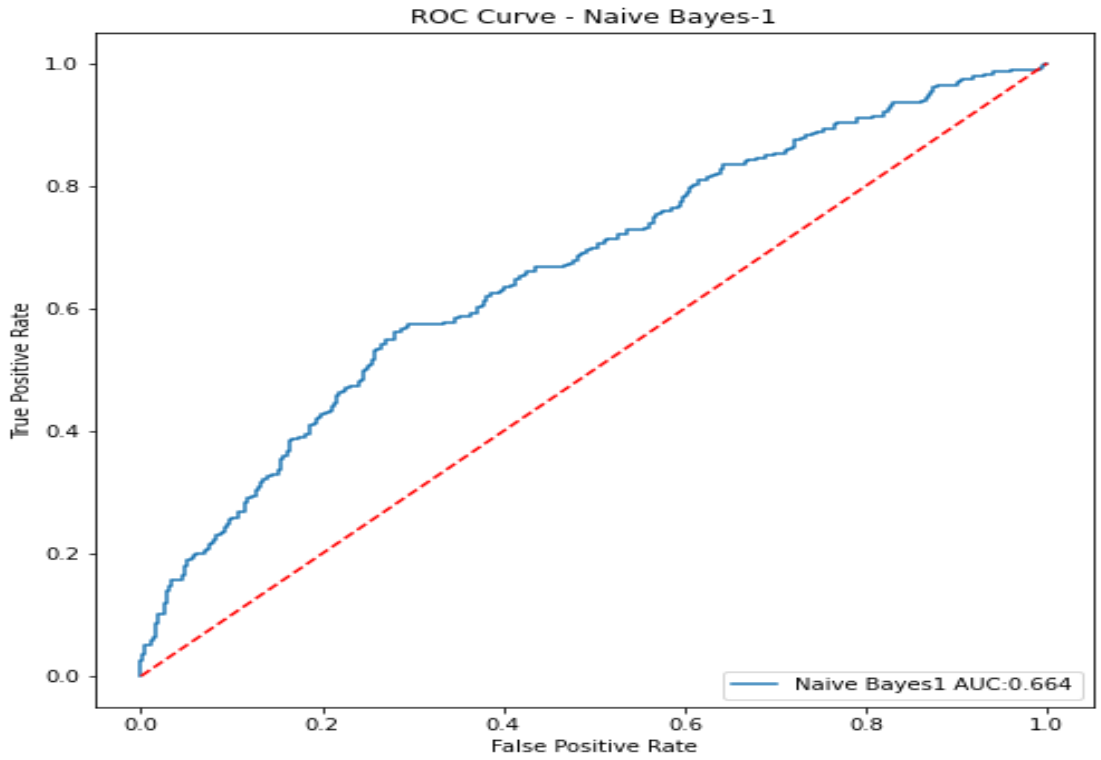

Fig.39

Model 11: Bagging Classifier Using KNN as base model:

1. First we have to import BaggingClassifier from sklearn.ensemble and KneighborsClassifier from skearn.neighbors.
2. After that we have to build Bagging classifier model by BaggingClassifier() and pass the KNN model as base model. We can also pass parameter like max_samples,max_features,n_estimators. In next step, we have to do model fitting.

```
BaggingClassifier(base_estimator=KNeighborsClassifier(metric='manhattan',
                                                      n_neighbors=2),
                  max_features=15, max_samples=1000, n_estimators=500)
```

3. Predicting the probability and probability class fir train set.
4. Calculating the model performance metrics.

Accuracy Score:

For train set:

```
0.9336330935251799
```

For test set:

```
0.7989093387866394
```

Confusion Matrix:

For train set:

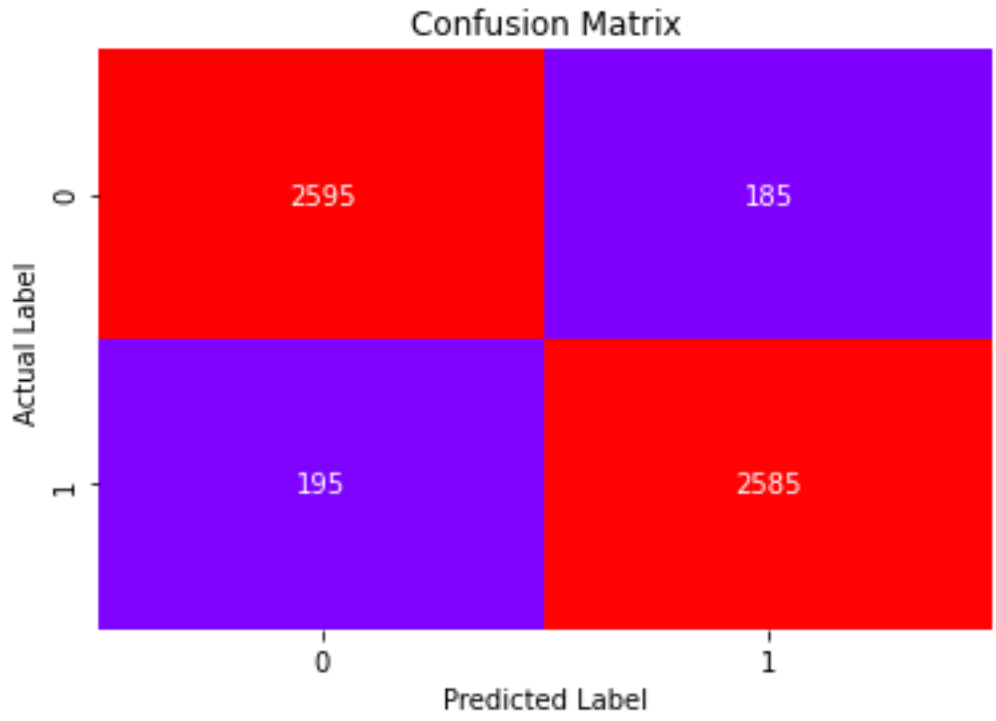

Fig.40

For test set:

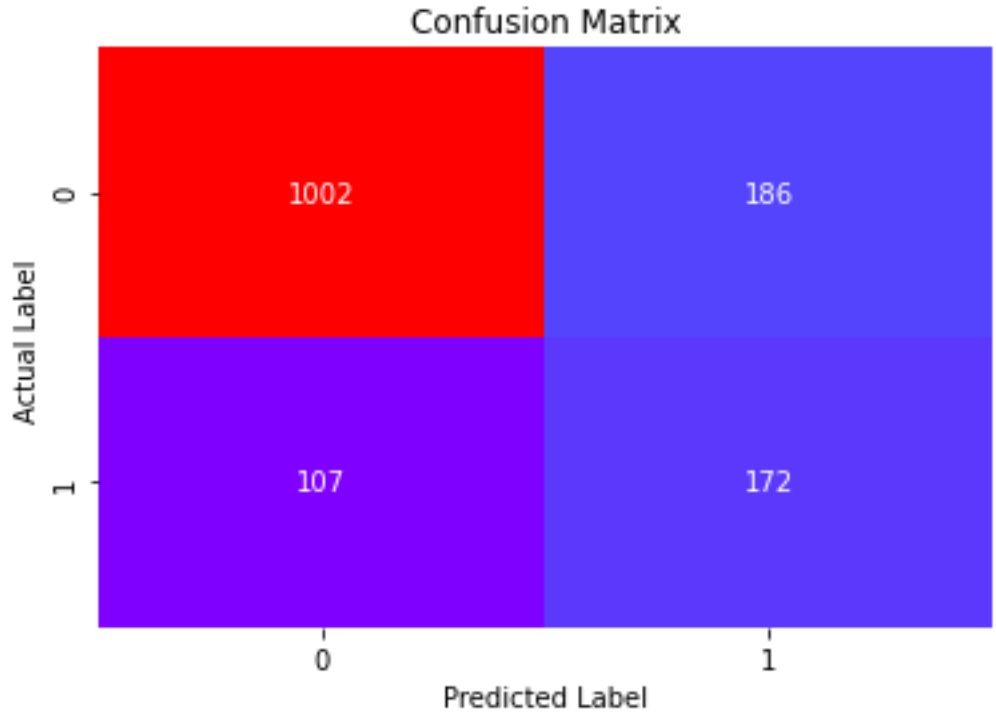

Fig.41

Classification Report:

For train set:

	precision	recall	f1-score	support
0	0.93	0.93	0.93	2780
1	0.93	0.93	0.93	2780
accuracy			0.93	5560
macro avg	0.93	0.93	0.93	5560
weighted avg	0.93	0.93	0.93	5560

For test set:

	precision	recall	f1-score	support
0	0.91	0.84	0.87	1188
1	0.48	0.62	0.54	279
accuracy			0.80	1467
macro avg	0.69	0.73	0.71	1467
weighted avg	0.82	0.80	0.81	1467

AUC and ROC Curve:

For train set:

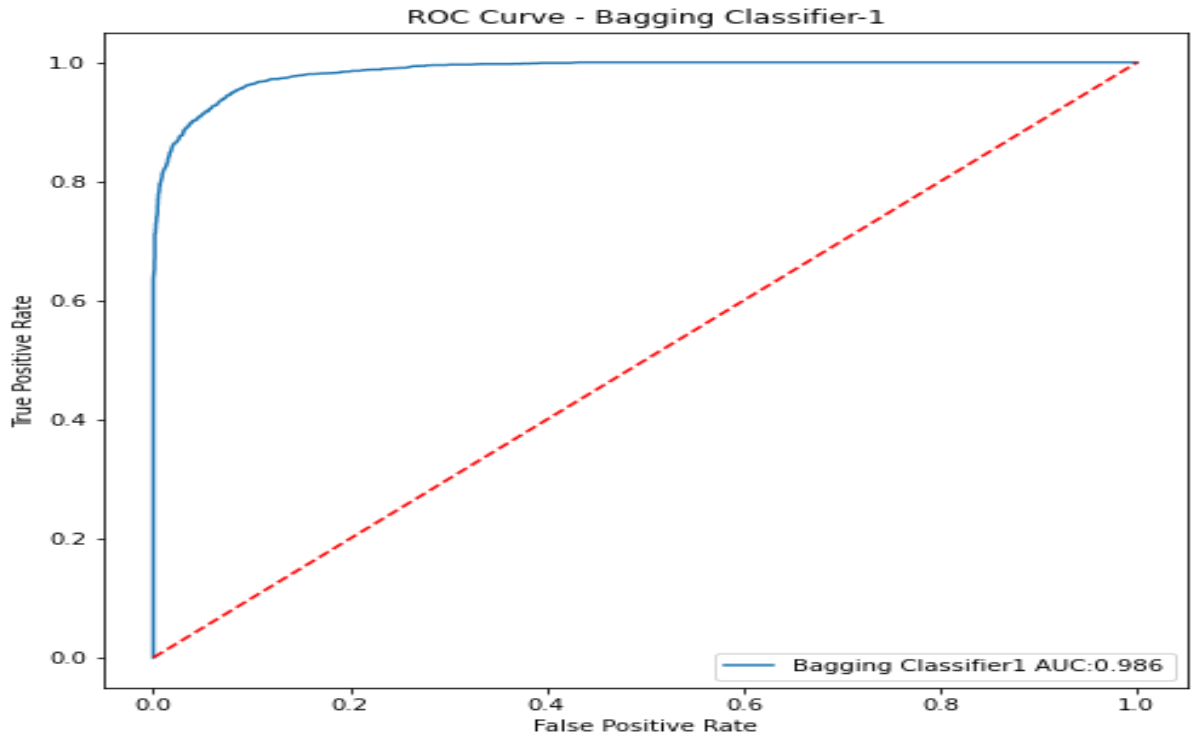

Fig.42

For test set:

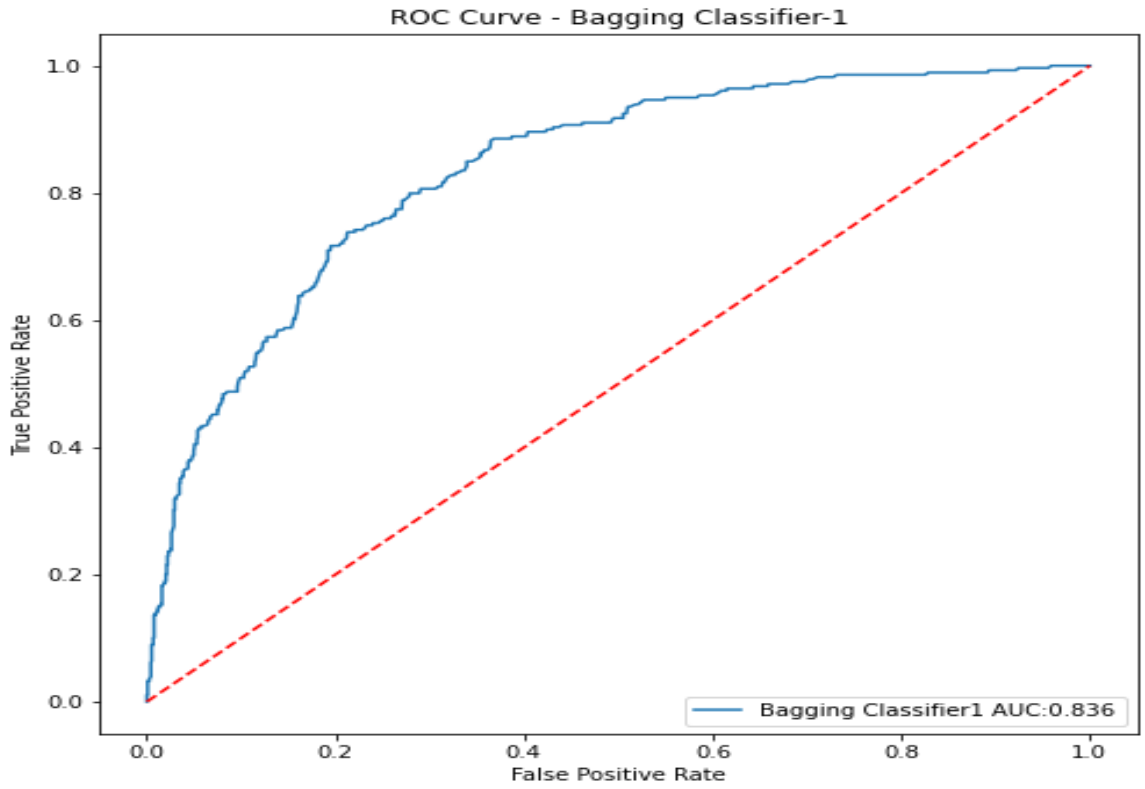

Fig.43

Model 12: Bagging Classifier with Decision tree as base model:

1. First we have to import BaggingClassifier from sklearn.ensemble and DecisionTreeClassifier from skearn.tree.
2. After that we have to build Bagging classifier model by BaggingClassifier() and pass the decision tree as base model. We can also pass parameter like max_samples,max_features,n_estimators. In next step, we have to do model fitting.

```
cart = DecisionTreeClassifier()
Bagging_model=BaggingClassifier(base_estimator=cart,n_estimators=100,max_samples=1000,
                max_features=15,random_state=1)
BC2=Bagging_model.fit(X_train, Y_train.values.ravel())
```

3. Predicting the probability and probability class fir train set.
4. Calculating the model performance metrics.

Accuracy Score:

For train set:

```
0.956294964028777
```

For test set:

```
0.8498091342876619
```

Confusion Matrix:

For train set:

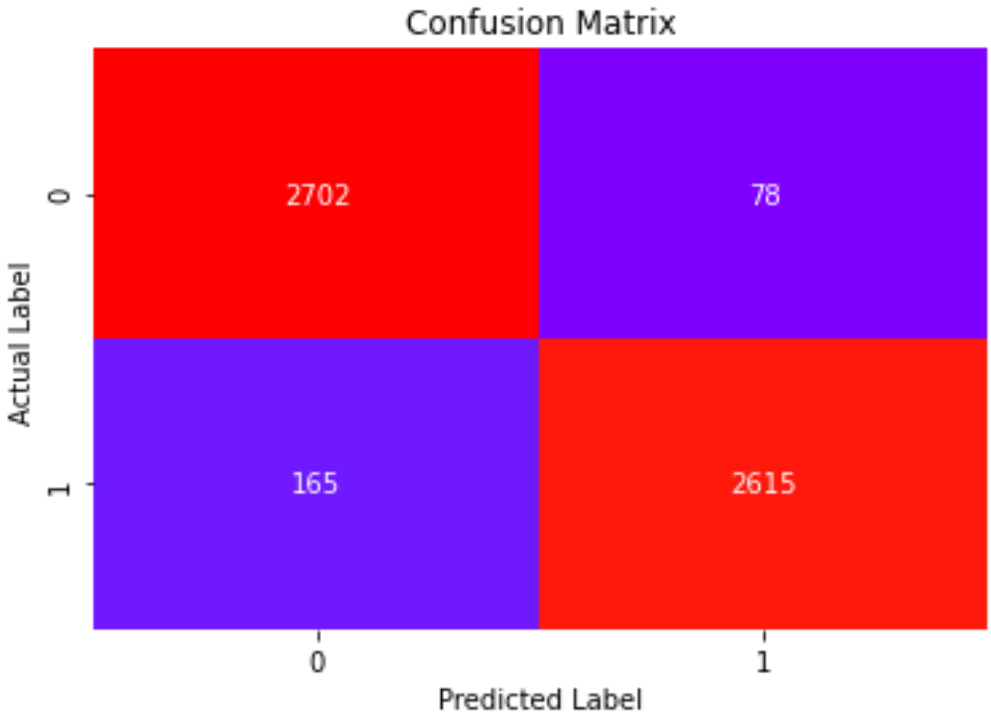

Fig.44

For test set:

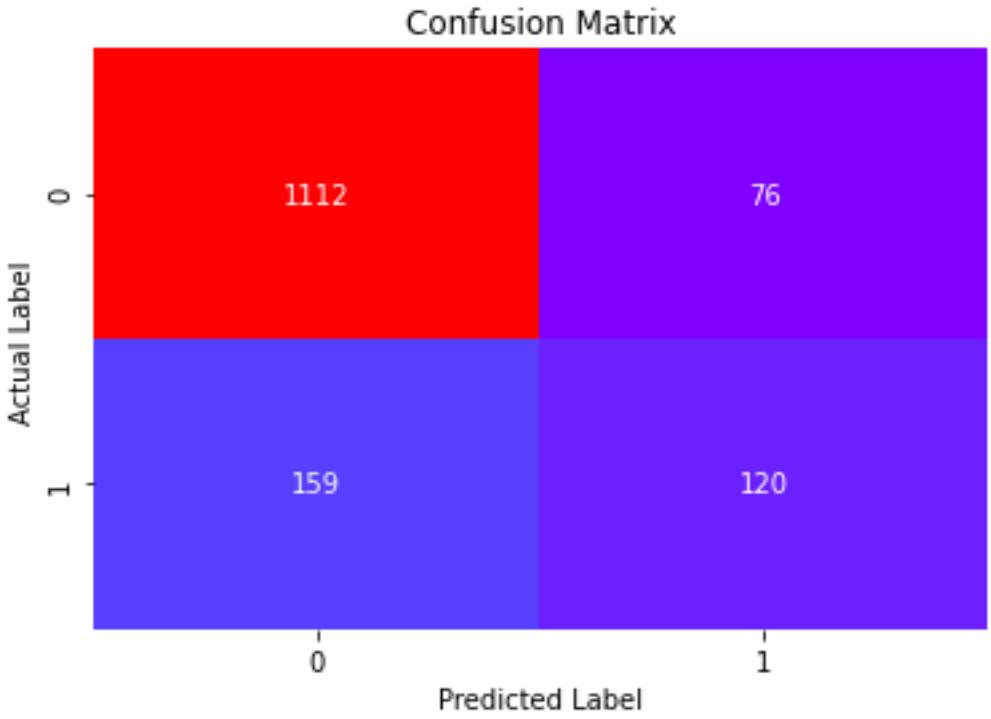

Fig.45

Classification Report:

For train set:

	precision	recall	f1-score	support
0	0.94	0.97	0.96	2780
1	0.97	0.94	0.96	2780
accuracy			0.96	5560
macro avg	0.96	0.96	0.96	5560
weighted avg	0.96	0.96	0.96	5560

For test set:

	precision	recall	f1-score	support
0	0.87	0.94	0.90	1188
1	0.61	0.43	0.51	279
accuracy			0.84	1467
macro avg	0.74	0.68	0.70	1467
weighted avg	0.82	0.84	0.83	1467

AUC and ROC Curve:

For train set:

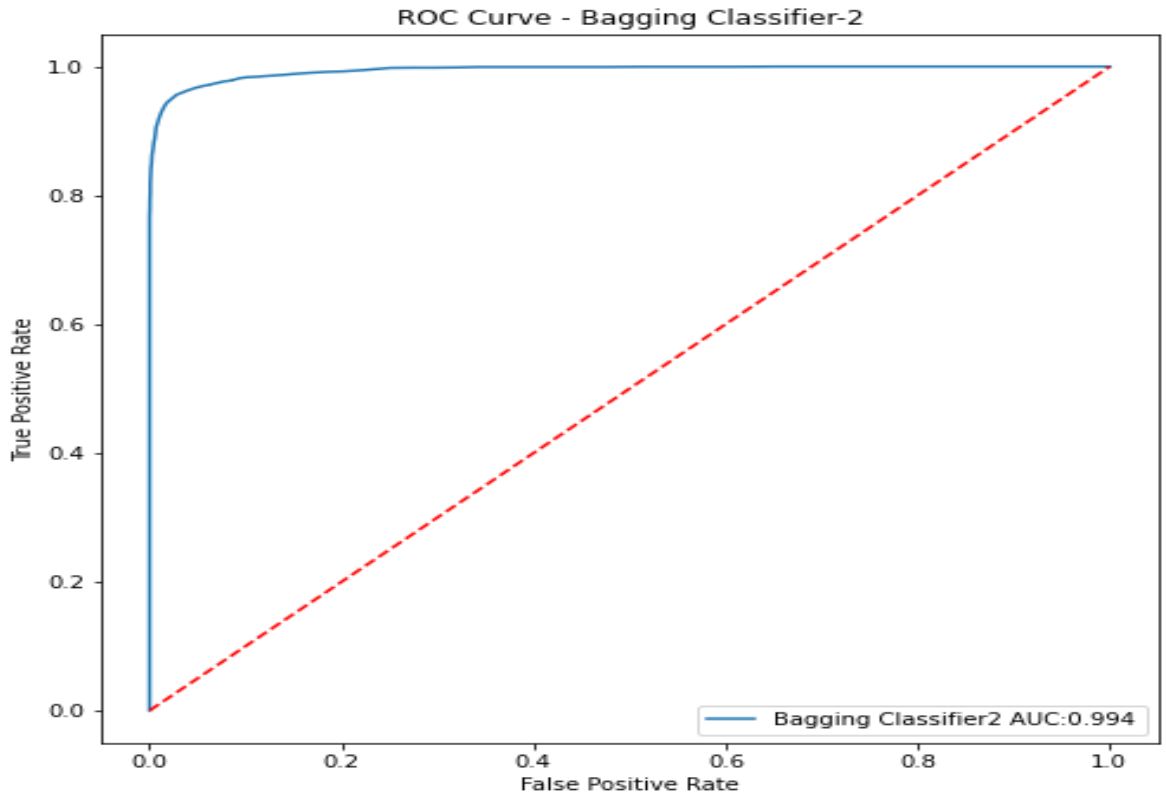

Fig.44

For test set:

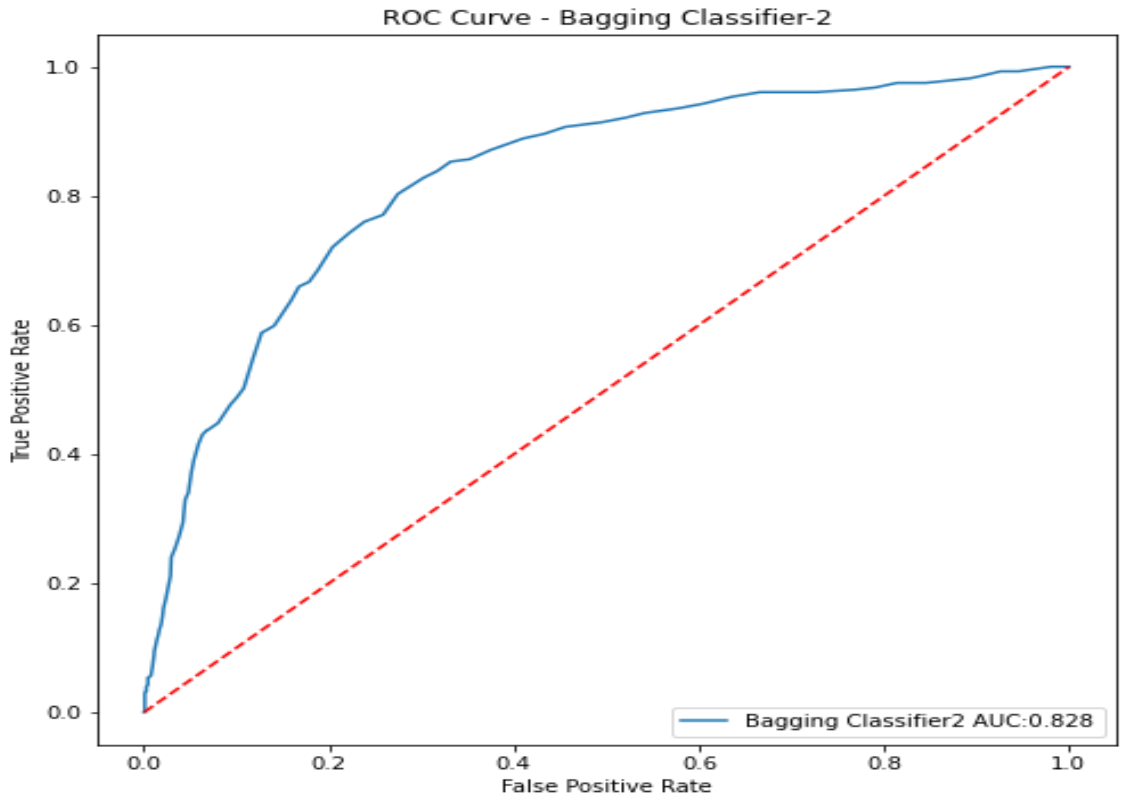

Fig.45

Now, I am going to use voting classifier to further improve the performance of model .

Voting Classifier:

Voting classifier is a machine learning model that trains on ensemble of numerous models and predict an output (class) based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into voting classifier and predict the output class based on the highest majority of voting.

Model 14: Voting Classifier1:

1. First we have to import VotingClassifer from sklearn.ensemble.
2. Next, we will create list of different models like logistic regression, decision tree, random forest, gradient boosting,naïve bayes, bagging with decision tree that we have to pass as base model into VotingClassifier().

   estim = [('LR1',LR1),('LR2',LR2),('DT1',DT1),('RF1',RF1),('GB1',GB1),('NB1',NB1),('BC2',BC2)]

3. Fit the model for train set.
4. Predicting the probability and probability class for train and test set.
5. Calculating performance metrics.

Accuracy score:

For train set:

```
0.9751798561151079
```

For test set:

```
0.8452624403544649
```

Confusion Matrix:

For train set:

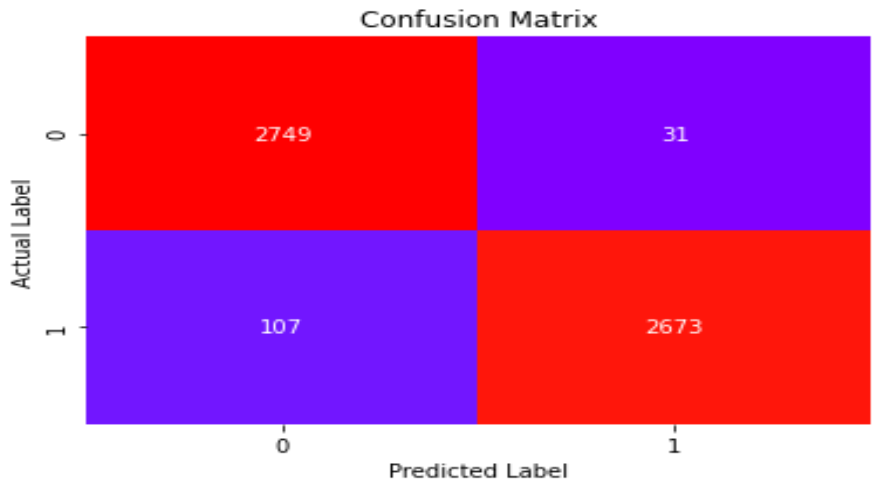

Fig.46

For test set:

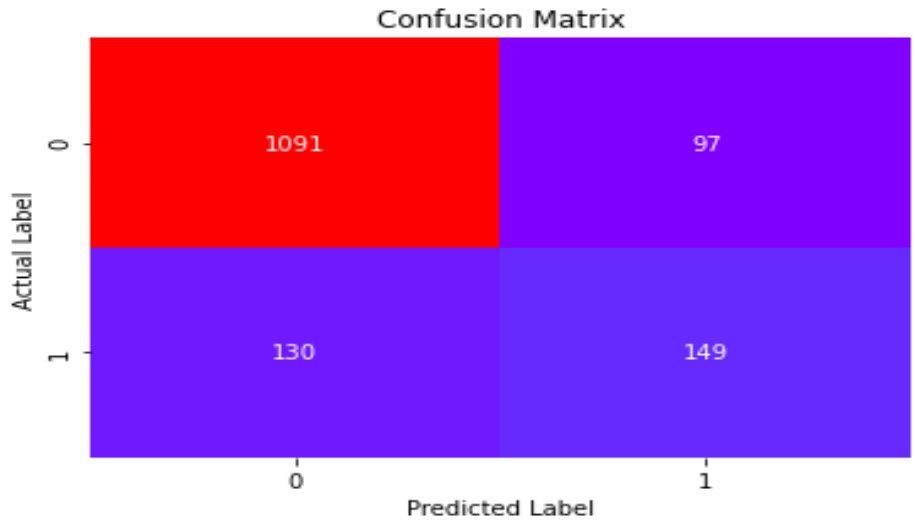

Fig.47

- Total no of correct prediction=149+1091
- Total no of incorrect prediction=130+97

Classification Report:

For train set:

	precision	recall	f1-score	support
0	0.96	0.99	0.98	2780
1	0.99	0.96	0.97	2780
accuracy			0.98	5560
macro avg	0.98	0.98	0.98	5560
weighted avg	0.98	0.98	0.98	5560

For test set:

	precision	recall	f1-score	support
0	0.89	0.92	0.91	1188
1	0.61	0.53	0.57	279
accuracy			0.85	1467
macro avg	0.75	0.73	0.74	1467
weighted avg	0.84	0.85	0.84	1467

- Only 53 % customers are correctly identified as the customers who have t aken product by Voting classifier1(VO1).

AUC and ROC curve:

For train set:

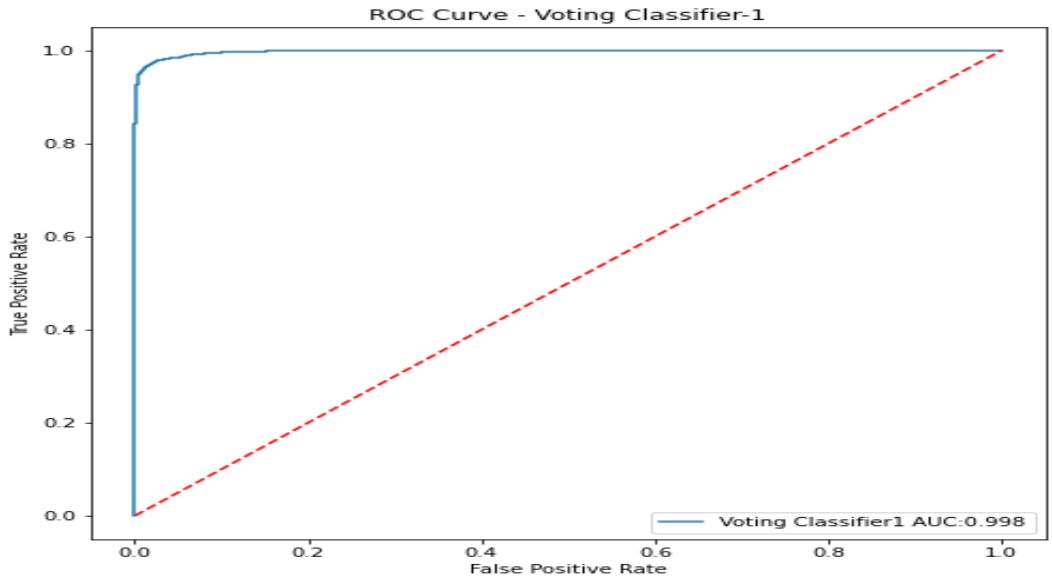

Fig.48

For test set:

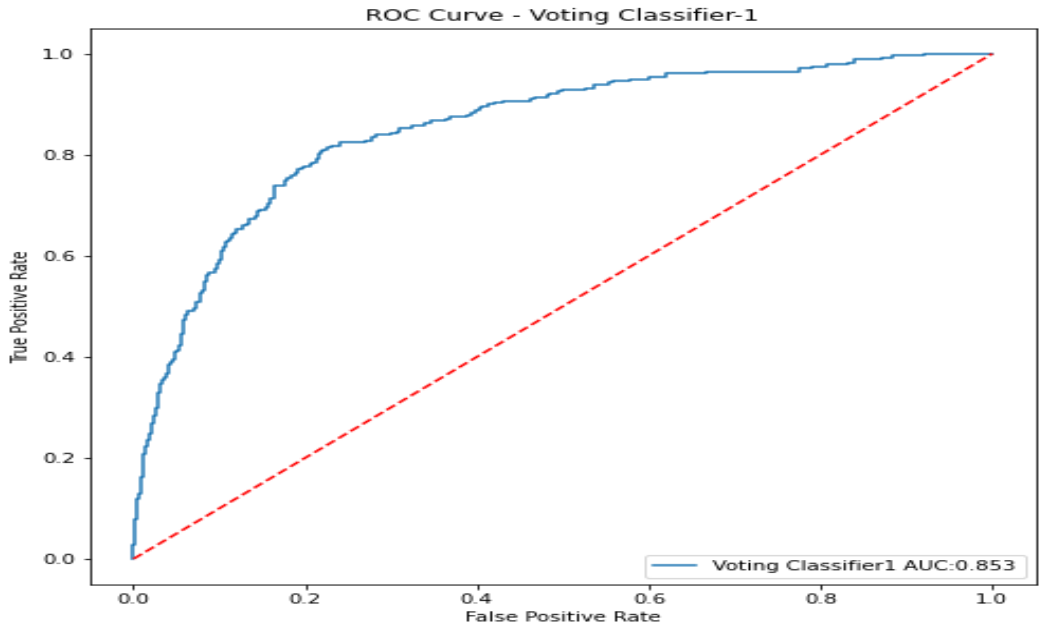

Fig.49

- AUC score for test set is not as good as train set. It means model performance is over fitting.

Model 15:VotingClassifer2:

1. First we have to import VotingClassifer from sklearn.ensemble.
2. Next, we will create list of different model likeKNN,SVC1,NN1,Bagging with decision tree(BC1) that we have to pass as base model into VotingClassifier().
   estim = [('KNN1',KNN1),('NN1',NN1),('SVC1',SVC1),('BC1',BC1)]
3. Fit the model for train set.
4. Predicting the probability and probability class for train and test set.
5. Calculating performance metrics.

Accuracy Score:

For train set:

```
0.9940647482014389
```

For test set:

```
0.8118609406952966
```

Confusion Matrix:

For train set:

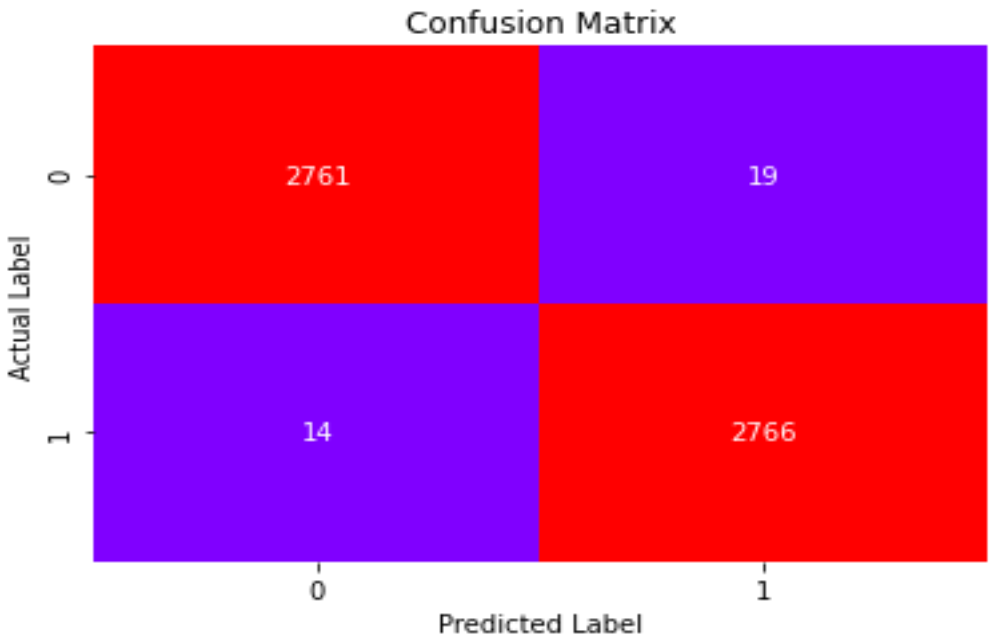

Fig.50

For test set:

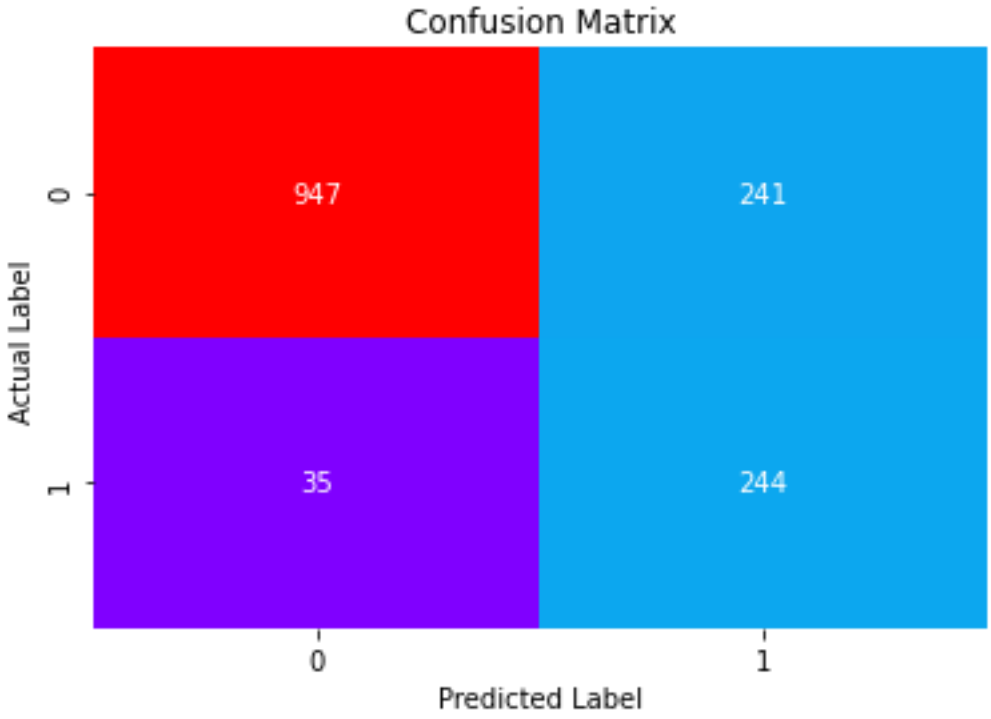

Fig.51

Classification Report:

For train set:

```
              precision    recall  f1-score   support
```

0	0.99	0.99	0.99	2780
1	0.99	0.99	0.99	2780
accuracy			0.99	5560
macro avg	0.99	0.99	0.99	5560
weighted avg	0.99	0.99	0.99	5560

For test set:

	precision	recall	f1-score	support
0	0.96	0.80	0.87	1188
1	0.50	0.87	0.64	279
accuracy			0.81	1467
macro avg	0.73	0.84	0.76	1467
weighted avg	0.88	0.81	0.83	1467

- 87 % customers are correctly identified , the customers who have taken pr oduct.

AUC and ROC Curve:

For train set:

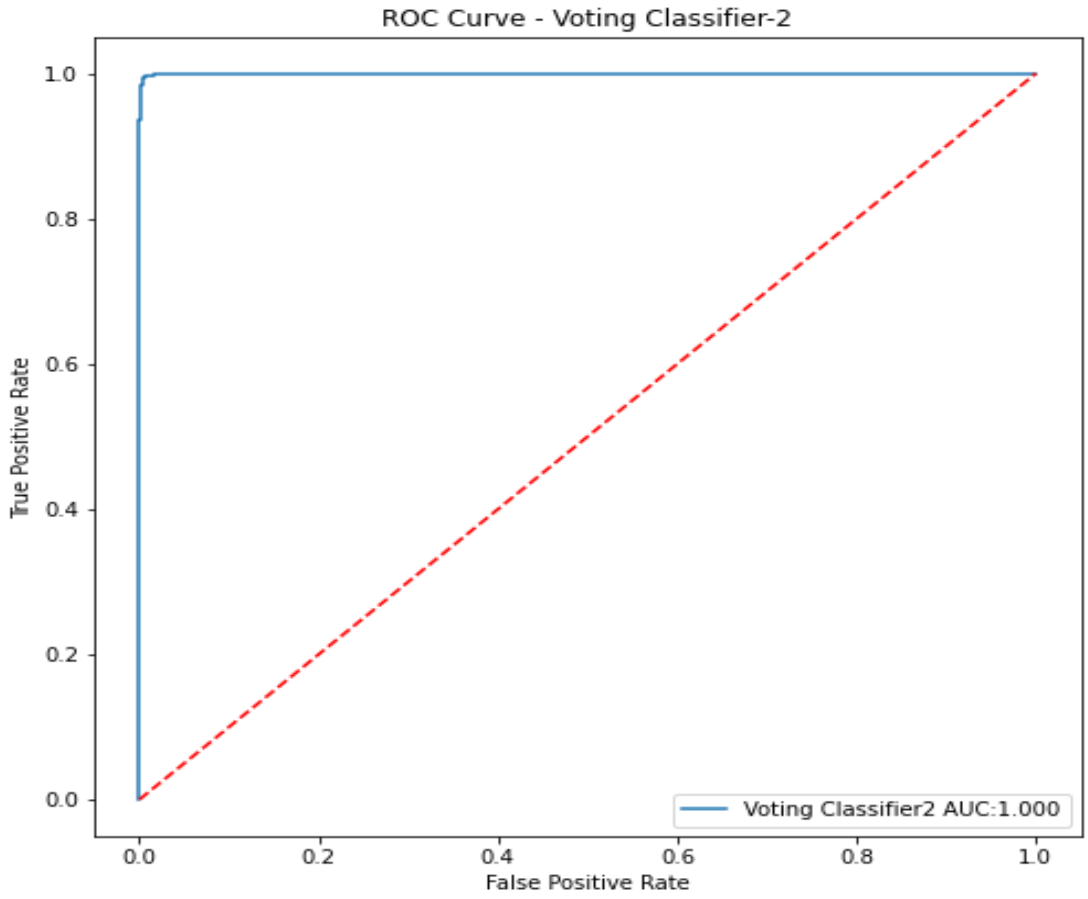

Fig.52

For test set:

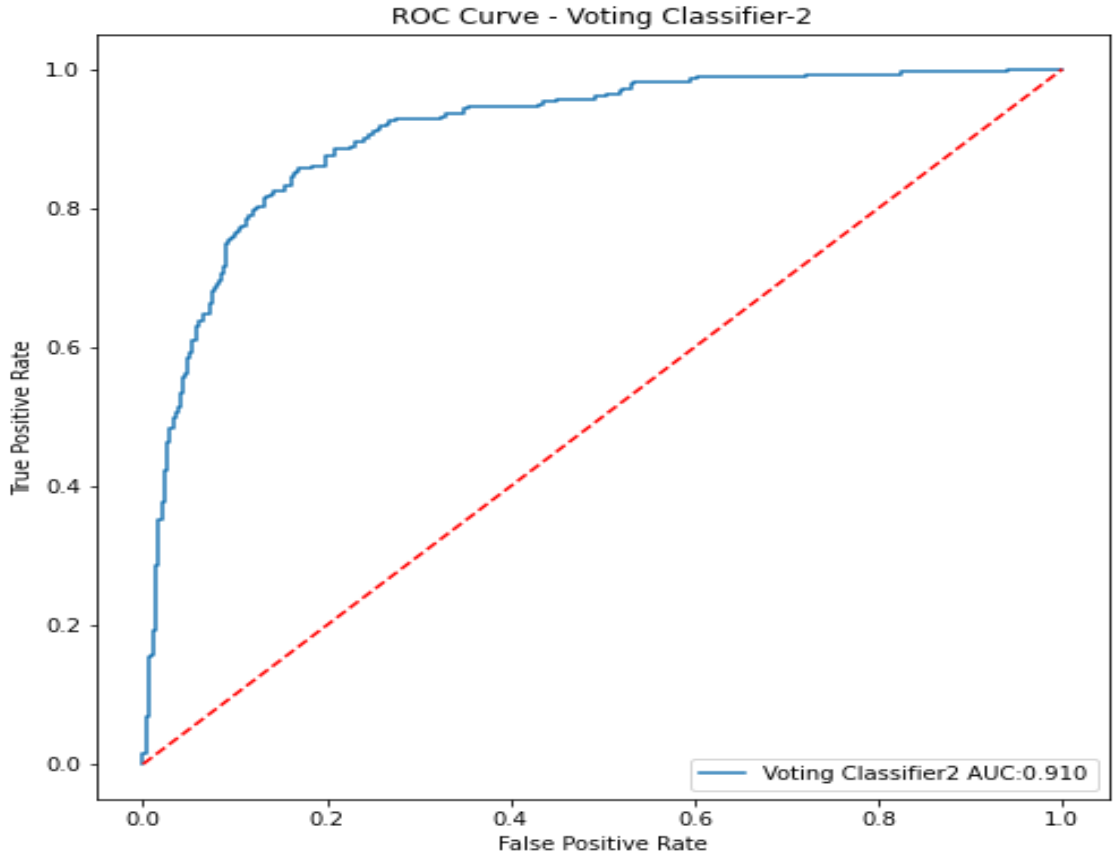

Fig.53

- AUC score is also good at test set.

Now we are going to try another classifier which is known as Stacking classifier to further improve the performance of model .

Stacking Classifier:

Stacking classifier is an ensemble learning technique to combine multiple classification models via meta-classifier. The individual classification models are trained on the complete training set, then the meta- classifier is fitted based on the outputs-meta-features-of the individual classification models in the ensemble. The meta- classifier can either be trained on the predicted class labels or probabilities from the ensemble.

Model 16:StackingClassifier1:

1. First we have to import StackingClassifer from sklearn.ensemble.
2. Next, I have taken logistic regression model as meta classifier and LR1,LR2,DT1,GB1,NB1,BC1 as base model and then pass the parameter

estimator and final estimator into StackingClassifer and fit the model on train set.

```
stacker = LogisticRegression(penalty='none',solver='newton-cg')
STA1 = StackingClassifier(estimators=estim,final_estimator = stacker,cv=5,verbose=2,n_jobs=-1)
STA1.fit(X_train,Y_train.values.ravel())
```

3. Predicting the probabilities and probability class.
4. Calculating the performance metrics.

Accuracy Score:

For train set:

```
1.0
```

For test set:

```
0.885480572597137
```

Confusion matrix:

For train set:

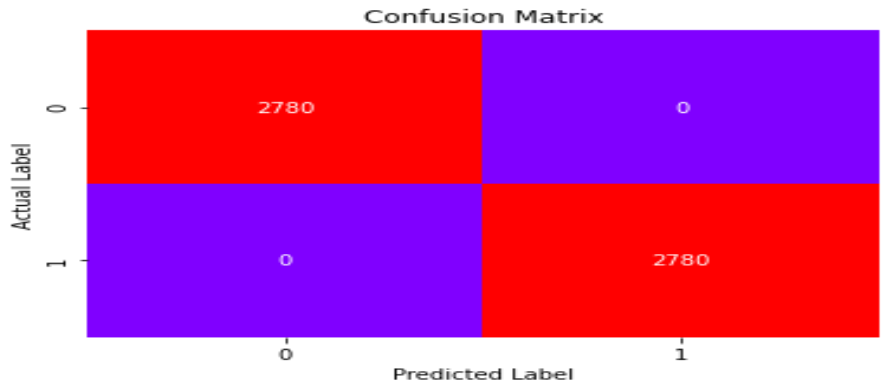

Fig.54

For test set:

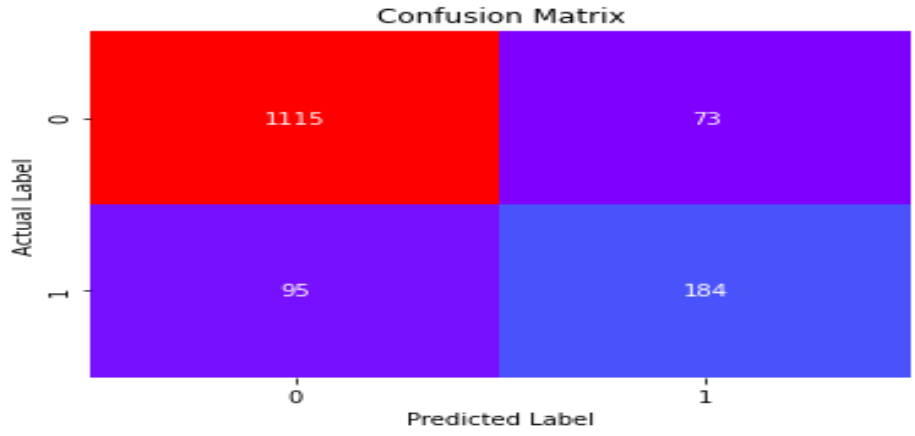

Fig.55

Classification Report:

For train set:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2780
1	1.00	1.00	1.00	2780
accuracy			1.00	5560
macro avg	1.00	1.00	1.00	5560
weighted avg	1.00	1.00	1.00	5560

For test set:

	precision	recall	f1-score	support
0	0.92	0.94	0.93	1188
1	0.72	0.66	0.69	279
accuracy			0.89	1467
macro avg	0.82	0.80	0.81	1467
weighted avg	0.88	0.89	0.88	1467

AUC and ROC Curve:

For train set:

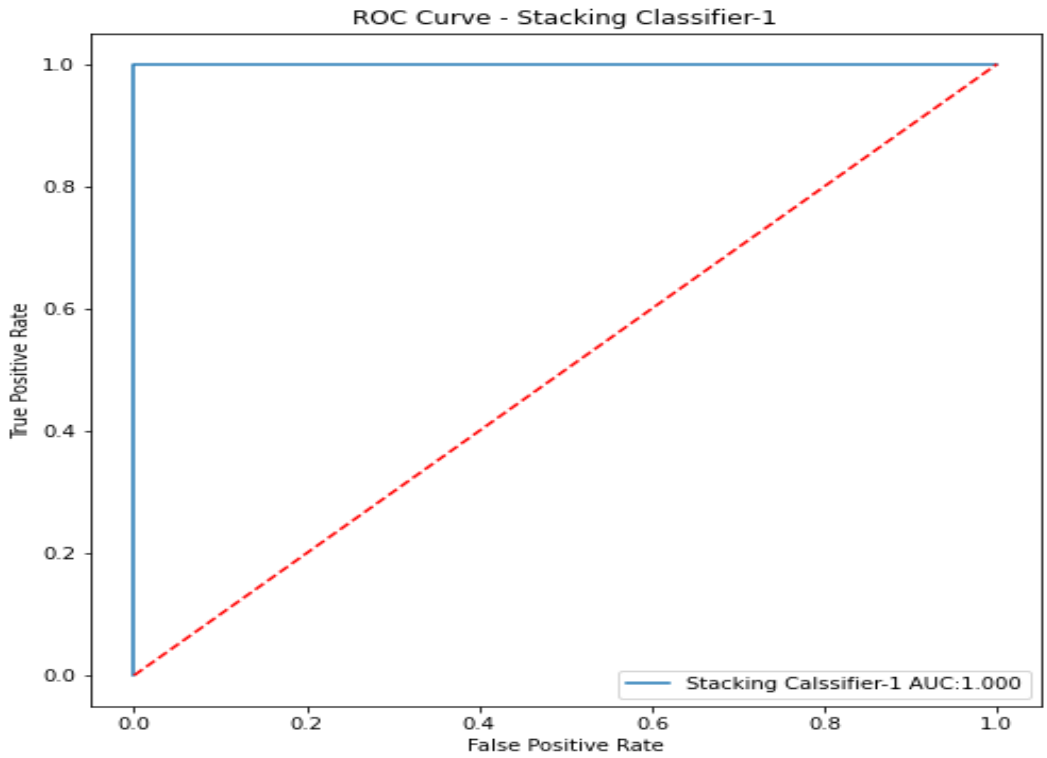

Fig.56

For test set:

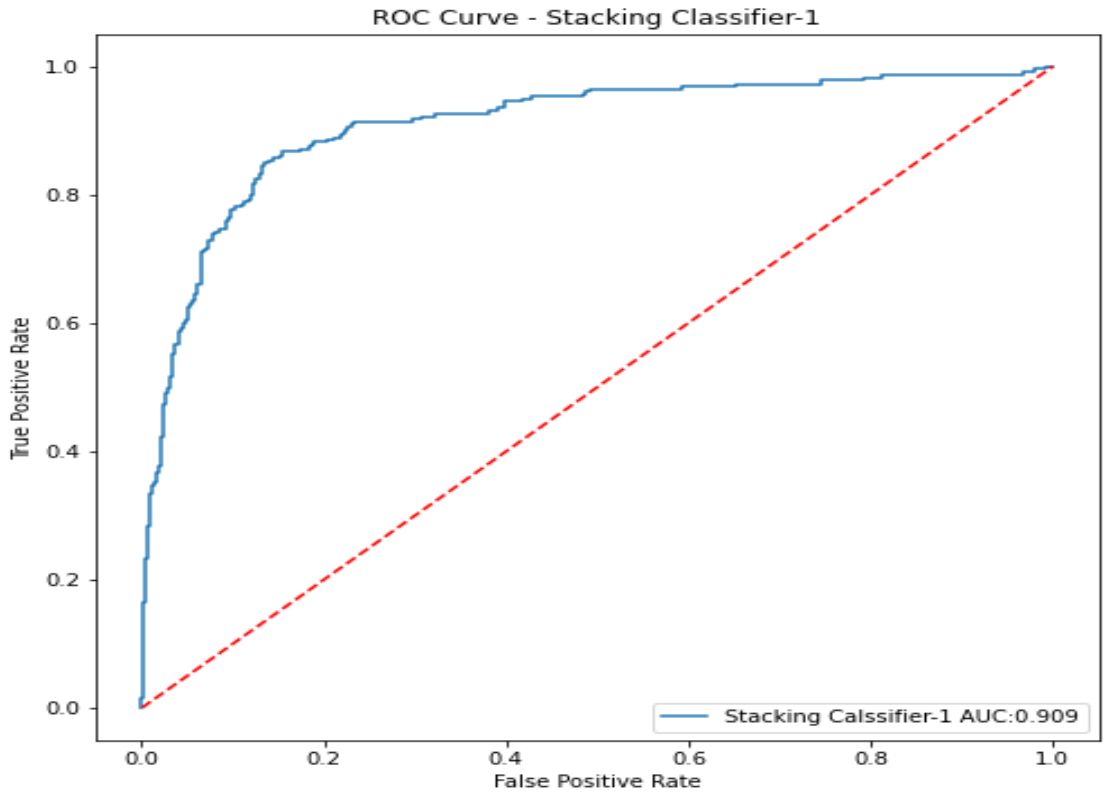

Fig.57

Model 17:StackingClassifier2:

1. First we have to import StackingClassifer from sklearn.ensemble.
2. Next, I have taken logistic regression model as meta classifier and KNN1,NN1,SVM1 and Bagging Classifier with BC1 as base model and then pass the parameter estimator and final estimator into StackingClassifer and fit the model on train set. Train and test se t(only independent set) should be scaled in this case because scaling is necessary for all model that I have taken for ensembling.

   ```
   stacker = LogisticRegression(penalty='none',solver='newton-cg')
   STA2 = StackingClassifier(estimators=estim,final_estimator = stacker,cv=5,verbose=2,n_jobs=-1)
   STA2.fit(X_train_std,Y_train.values.ravel())
   ```
3. Predicting the probabilities and probability class.
4. Calculating performance metrics.

Accuracy Score:

For train set:

```
0.9951438848920864
```

For test set:

```
0.7934560327198364
```

Confusion Matrix:

For train set:

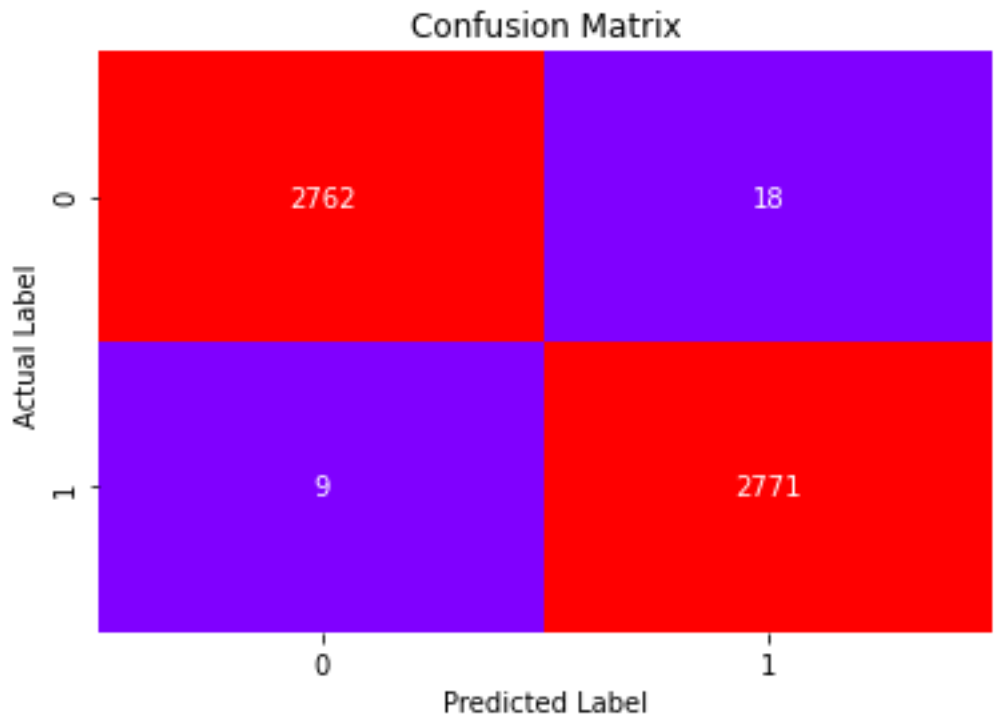

Fig.58

For test set:

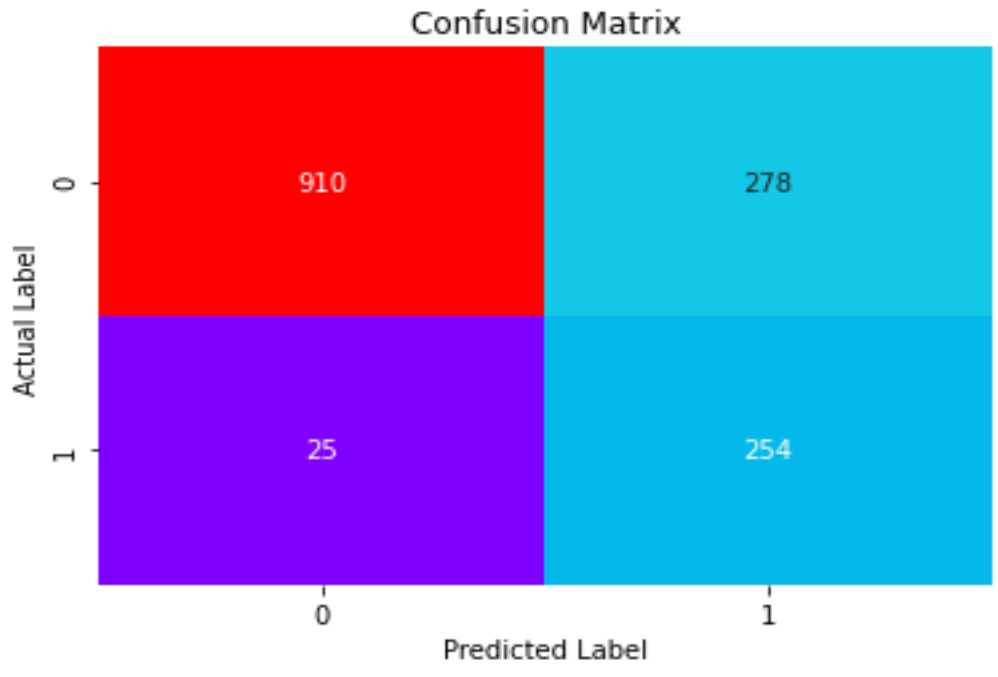

Fig.59

- For test set, total no correct prediction=254+910
- For test set, total no of incorrect prediction=25+278

Classification Report:

For train set:

	precision	recall	f1-score	support
0	1.00	0.99	1.00	2780
1	0.99	1.00	1.00	2780
accuracy			1.00	5560
macro avg	1.00	1.00	1.00	5560
weighted avg	1.00	1.00	1.00	5560

For test set:

	precision	recall	f1-score	support
0	0.97	0.77	0.86	1188
1	0.48	0.91	0.63	279
accuracy			0.79	1467
macro avg	0.73	0.84	0.74	1467
weighted avg	0.88	0.79	0.81	1467

- 91 % customers are correctly identified as the customers who have taken product.

AUC and ROC Curve:

For train set:

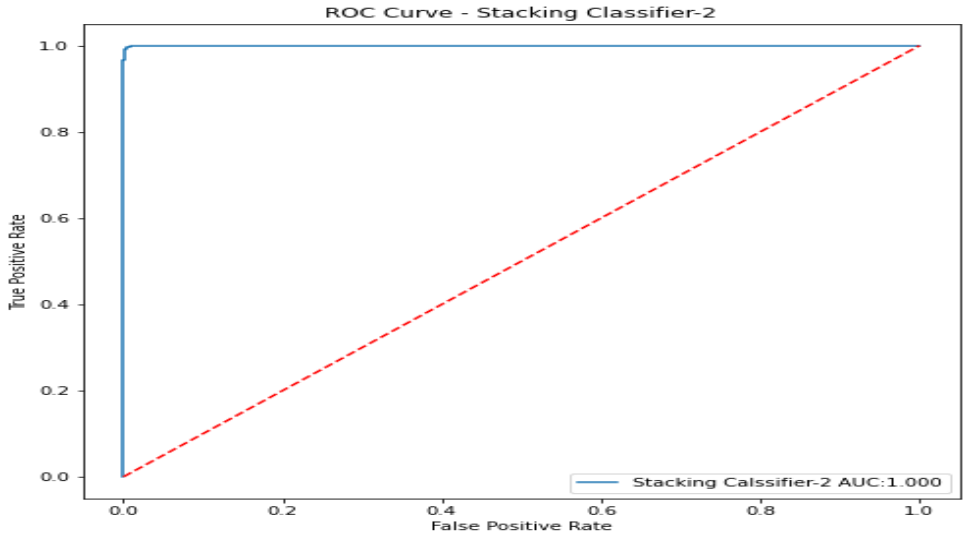

Fig.60

For test set:

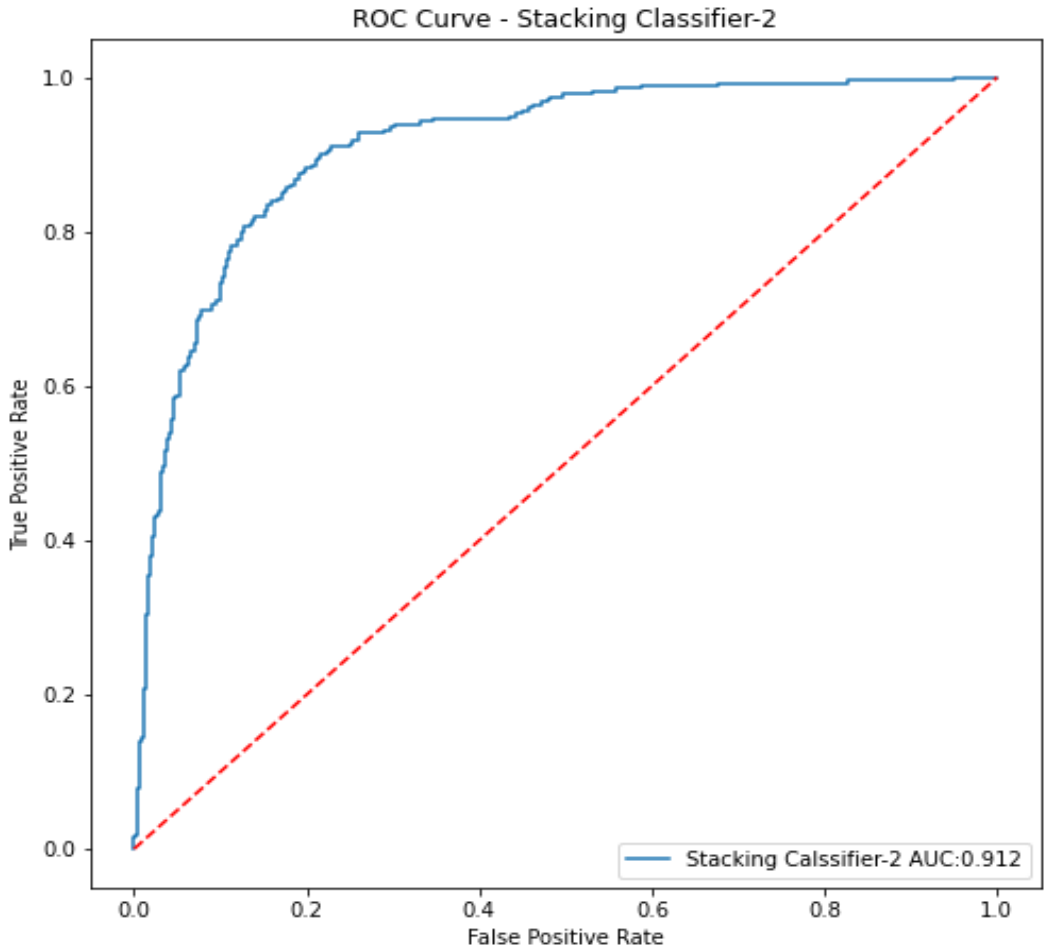

Fig.61

- AUC score is good for test set also as well train set(less difference of AUC score).We can say, model is not performing over fitting.

Comparing the model performance:

As we have to know, we have to predict, whether a customer will opt long term tourism package or not. Hence, for model selection criterion would be the model which has more accurately predicted class 1.Therefore, we have to check recall that predict how many true data points identified as true.

Recall for Neural Network (NN1) for test set is (0.96) that is more than all other model's test set that we have done so far.

Model	Accuracy		Precision		Recall		F1-score		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Logistic regression (LR1)	0.81	0.77	0.78	0.51	0.84	0.42	0.81	0.46	0.81	0.67
Logistic Regression with Recursive Feature Elimination (LR2)	0.81	0.77	0.84	0.41	0.77	0.49	0.80	0.45	0.81	0.66
Decision Tree (DT1)	0.99	0.83	0.99	0.53	0.99	0.67	0.99	0.59	0.99	0.76
Random Forest (RF1)	1	0.87	1	0.72	1	0.49	1	0.58	1	0.89
Gradient Boosting(GB1)	1	0.89	1		1		1		1	
K-Nearest Neighbours (KNN1)	0.98	0.87	1	0.68	0.96	0.54	0.98	0.61	0.99	0.85
Neural Network (NN1)	1	0.60	1	0.32	1	**0.96**	1	0.48	1	**0.91**
Support Vector Classifier(SVC1)	0.99	0.77	0.99	0.44	0.99	0.89	0.99	0.59	0.99	0.90
Naïve Bayes (NB1)	0.75	0.70	0.75	0.32	0.76	0.51	0.76	0.39	0.83	0.66
Bagging with K-Nearest Neighbours (BC1)	0.93	0.80	0.93	0.48	0.93	0.62	0.93	0.54	0.99	0.83
Bagging with Decision tree (BC2)	0.96	0.84	0.98	0.62	0.94	0.42	0.96	0.50	0.99	0.83
Voting Classifier1 (VO1)	0.98	0.85	0.99	0.61	0.96	0.53	0.97	0.57	0.99	0.85
Voting Classifier2 (VO2)	0.99	0.81	0.99	0.50	0.99	0.87	0.99	0.64	1	0.91
Stacking Classifier1 (STA1)	1	0.89	1	0.72	1	0.66	1	0.69	1	0.91
Stacking Classifier2 (STA2)	1	0.79	0.99	0.48	1	0.91	1	0.63	1	0.91

AUC score is also good (0.91) for Neural Network1 (NN1) and (STA1 and STA2) Stacking Classifier1 (0.91) is more than others model. As we know, higher the AUC score, better the model is. But recall is poor for model STA1 (0.62) and not so poor for STA2 (0.91) but less than NN1.So we conclude here, NN1 is best model. Although accuracy is not so good for NN1, we will consider it as best model because accuracy is not measure concern to decide the model is good or not.

"Neural Network (NN1) is the best model."

Visualization the model performance:

This depicts, how the models are performing with respect to different metrics.

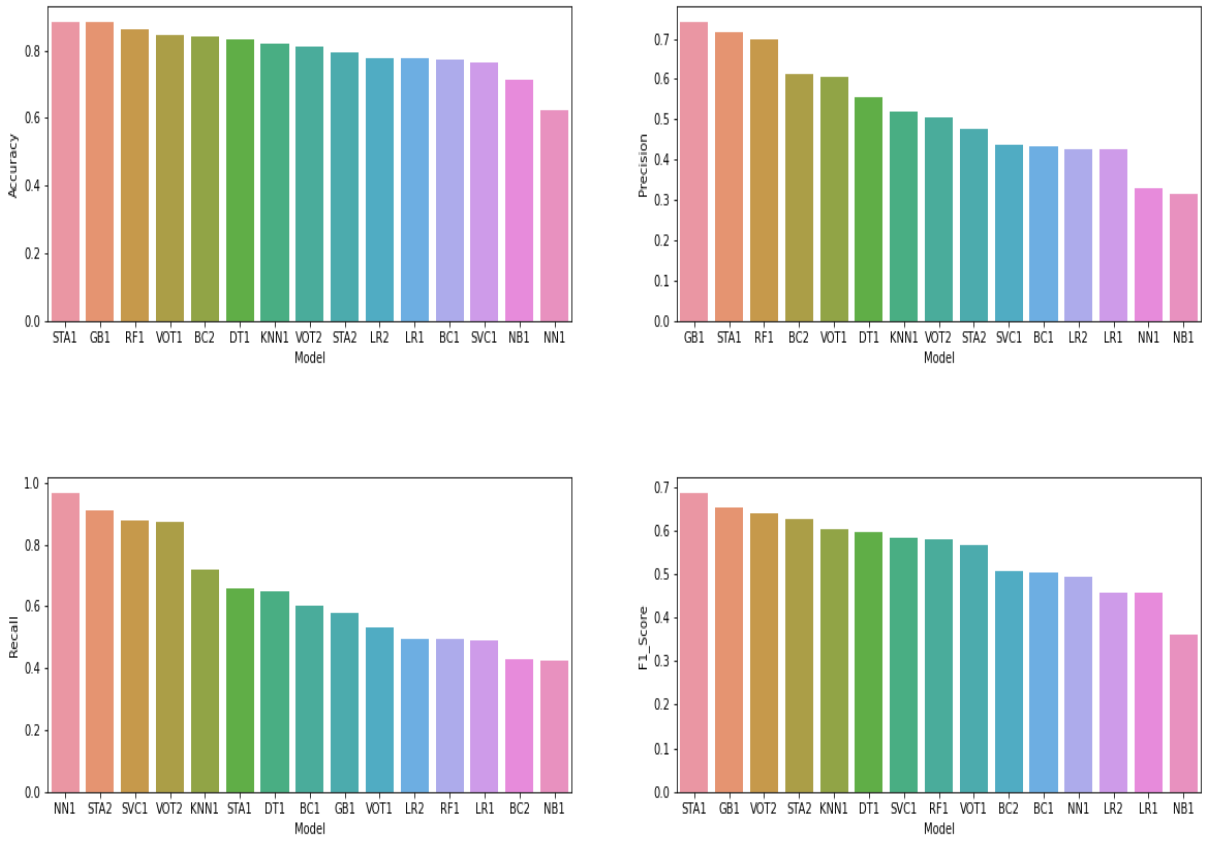

Fig.62

Observations:

- Recall is the best for Neural network model(NN1).
- AUC is the best for Stacking classifier1,2(STA1,2)andNN1).
- Precision is the best for gradient boosting (GB1).
- F1-score is the best for Stacking Classifier1(STA1).

Business Insights and Recommendations:

- 1. A customer having passport, increases the probability of taken prod by 76%.
- If the customer age increases, the probability of taken product decreases by 50%.
- A customer belongs to Tier-3, increases the probability of prod taken increases by 61%.
- If the monthly income of customer increases, the probability of prod taken increases by 50%.
- If the number of trips increases, probability of taking prod decreases by 51%.
- If the customer having small business and salaried, the probability of taking product decreases by 1%.
- A person having own car, decreases the probability of product taken by 36%.
- A customer, who has married, decreases the probability of product taken by 19%.
- A customer, whose designation is high (work as VP,AVP), decreases the probability of product taken by 31%.

From business perspective, we can say we have to target those customers who have passport and more monthly income and middle age and young age group customers that are working as executive, manger to increase the purchasing of product and business as well.