IBM NAAN MUDHALVAN

CREDIT CARD FRAUD DETECTION

PHASE 4:

SMOTE (Synthetic Minority Oversampling Technique)

We are creating synthetic samples by doing upsampling using SMOTE(Synthetic Minority Oversampling Technique).

# Importing SMOTE

from imblearn.over\_sampling import SMOTE

# Instantiate SMOTE

sm = SMOTE(random\_state=27)

# Fitting SMOTE to the train set

X\_train\_smote, y\_train\_smote = sm.fit\_sample(X\_train, y\_train)

print('Before SMOTE oversampling X\_train shape=',X\_train.shape)

print('After SMOTE oversampling X\_train shape=',X\_train\_smote.shape)

Before SMOTE oversampling X\_train shape= (227845, 29)

After SMOTE oversampling X\_train shape= (454898, 29)

Logistic Regression

# Creating KFold object with 5 splits

folds = KFold(n\_splits=5, shuffle=True, random\_state=4)

# Specify params

params = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as roc-auc

model\_cv = GridSearchCV(estimator = LogisticRegression(),

param\_grid = params,

scoring= 'roc\_auc',

cv = folds,

verbose = 1,

return\_train\_score=True)

# Fit the model

model\_cv.fit(X\_train\_smote, y\_train\_smote)

Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 30 out of 30 | elapsed: 1.5min finished

GridSearchCV(cv=KFold(n\_splits=5, random\_state=4, shuffle=True),

error\_score=nan,

estimator=LogisticRegression(C=1.0, class\_weight=None, dual=False,

fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None,

max\_iter=100, multi\_class='auto',

n\_jobs=None, penalty='l2',

random\_state=None, solver='lbfgs',

tol=0.0001, verbose=0,

warm\_start=False),

iid='deprecated', n\_jobs=None,

param\_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]},

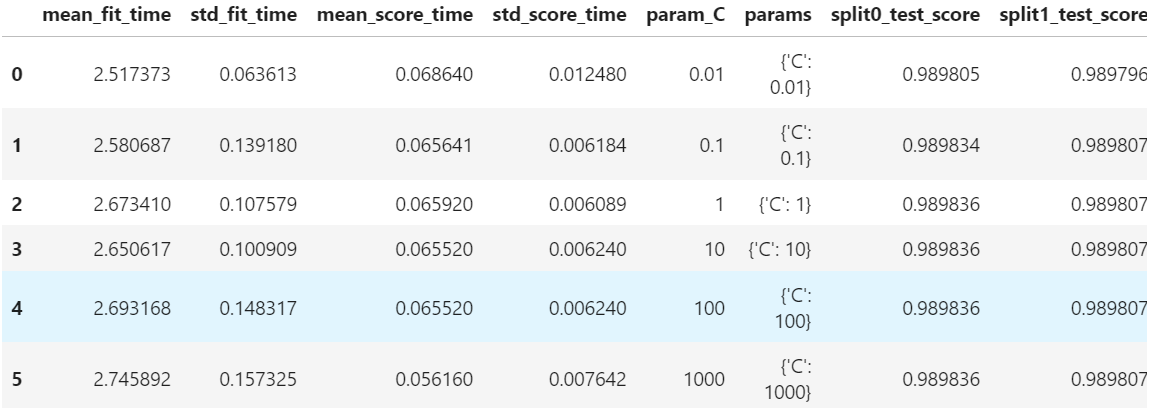
pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=True,

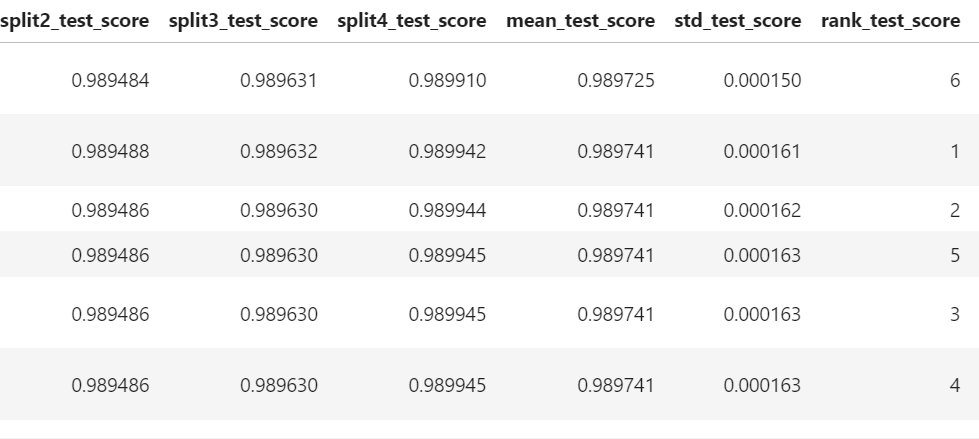
scoring='roc\_auc', verbose=1)

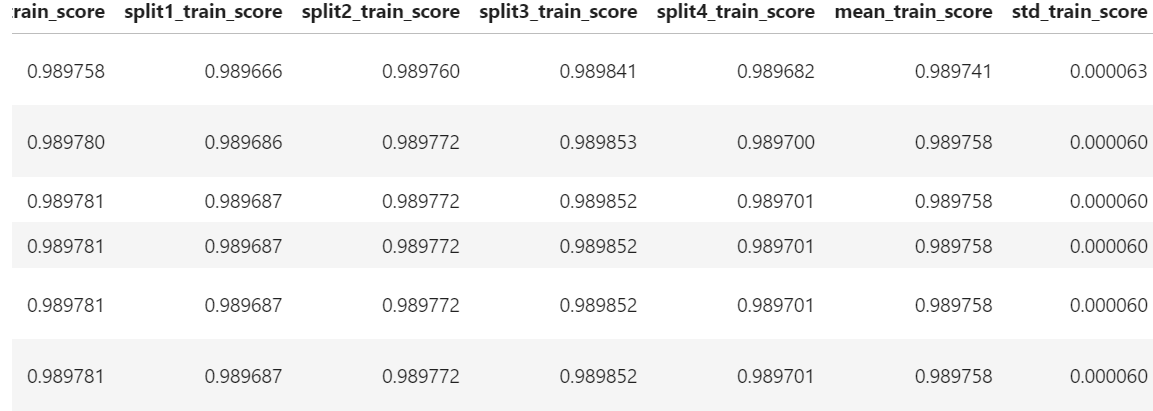
# results of grid search CV

cv\_results = pd.DataFrame(model\_cv.cv\_results\_)

cv\_results







# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))

plt.plot(cv\_results['param\_C'], cv\_results['mean\_test\_score'])

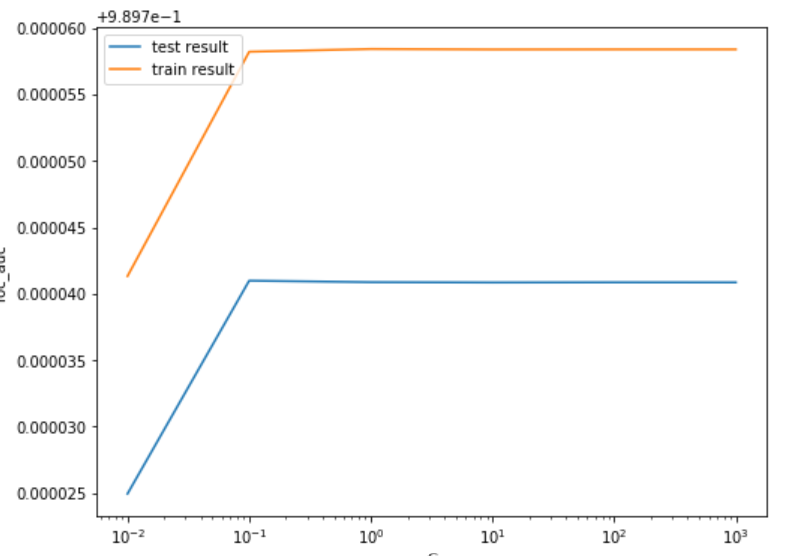
plt.plot(cv\_results['param\_C'], cv\_results['mean\_train\_score'])

plt.xlabel('C')

plt.ylabel('roc\_auc')

plt.legend(['test result', 'train result'], loc='upper left')

plt.xscale('log')



# Best score with best C

best\_score = model\_cv.best\_score\_

best\_C = model\_cv.best\_params\_['C']

print(" The highest test roc\_auc is {0} at C = {1}".format(best\_score, best\_C))

The highest test roc\_auc is 0.9897409900830768 at C = 0.1

Logistic regression with optimal C

# Instantiate the model with best C

logistic\_bal\_smote = LogisticRegression(C=0.1)

# Fit the model on the train set

logistic\_bal\_smote\_model = logistic\_bal\_smote.fit(X\_train\_smote, y\_train\_smote)

Prediction on the train set

# Predictions on the train set

y\_train\_pred = logistic\_bal\_smote\_model.predict(X\_train\_smote)

# Confusion matrix

confusion = metrics.confusion\_matrix(y\_train\_smote, y\_train\_pred)

print(confusion)

[[221911 5538]

[ 17693 209756]]

TP = confusion[1,1] # true positive

TN = confusion[0,0] # true negatives

FP = confusion[0,1] # false positives

FN = confusion[1,0] # false negatives

# Accuracy

print("Accuracy:-",metrics.accuracy\_score(y\_train\_smote, y\_train\_pred))

# Sensitivity

print("Sensitivity:-",TP / float(TP+FN))

# Specificity

print("Specificity:-", TN / float(TN+FP))

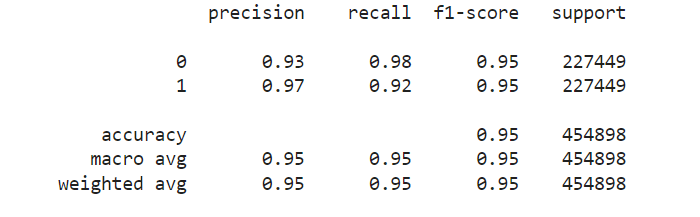
Accuracy:- 0.9489314087993352

Sensitivity:- 0.9222111330452102

Specificity:- 0.9756516845534603

# classification\_report

print(classification\_report(y\_train\_smote, y\_train\_pred))

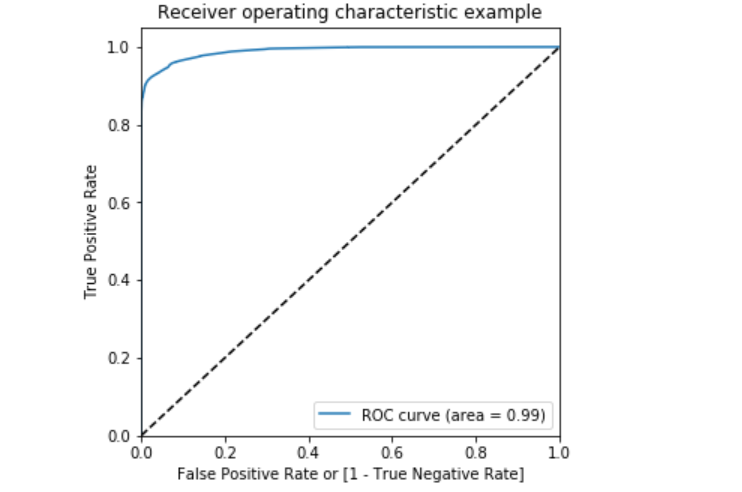


# Predicted probability

y\_train\_pred\_proba\_log\_bal\_smote = logistic\_bal\_smote\_model.predict\_proba(X\_train\_smote)[:,1]

# Plot the ROC curve

draw\_roc(y\_train\_smote, y\_train\_pred\_proba\_log\_bal\_smote)



**Prediction on the test set**

*# Prediction on the test set*

y\_test\_pred **=** logistic\_bal\_smote\_model**.**predict(X\_test)

*# Confusion matrix*

confusion **=** metrics**.**confusion\_matrix(y\_test, y\_test\_pred)

print(confusion)

[[55416 1450]

[ 10 86]]

TP **=** confusion[1,1] *# true positive*

FP **=** confusion[0,1] *# false positives*

FN **=** confusion[1,0] *# false negatives*

*# Accuracy*

print("Accuracy:-",metrics**.**accuracy\_score(y\_test, y\_test\_pred))

*# Sensitivity*

print("Sensitivity:-",TP **/** float(TP**+**FN))

*# Specificity*

print("Specificity:-", TN **/** float(TN**+**FP))

Accuracy:- 0.9743688774972789

Sensitivity:- 0.8958333333333334

Specificity:- 0.9745014595716245

*# classification\_report*

print(classification\_report(y\_test, y\_test\_pred))

precision recall f1-score support

0 1.00 0.97 0.99 56866

1 0.06 0.90 0.11 96

accuracy 0.97 56962

macro avg 0.53 0.94 0.55 56962

weighted avg 1.00 0.97 0.99 56962

**ROC on the test set**

*# Predicted probabilitY*y\_test\_pred\_proba **=** logistic\_bal\_smote\_model**.**predict\_proba(X\_test)[:,1]

*# Plot the ROC curve*

draw\_roc(y\_test, y\_test\_pred\_proba)

\***Model summary**\*

* Train set
  + Accuracy = 0.95
  + Sensitivity = 0.92
  + Specificity = 0.98
  + ROC = 0.99
* Test set
  + Accuracy = 0.97
  + Sensitivity = 0.90
  + Specificity = 0.99
  + ROC = 0.97

**XGBoost**

*# hyperparameter tuning with XGBoost*

*# creating a KFold object*

folds **=** 3

*# specify range of hyperparameters*

param\_grid **=** {'learning\_rate': [0.2, 0.6],

'subsample': [0.3, 0.6, 0.9]}

*# specify model*

xgb\_model **=** XGBClassifier(max\_depth**=**2, n\_estimators**=**200)

*# set up GridSearchCV()*

model\_cv **=** GridSearchCV(estimator **=** xgb\_model,

param\_grid **=** param\_grid,

scoring**=** 'roc\_auc',

cv **=** folds,

verbose **=** 1,

return\_train\_score**=True**)

*# fit the model*

model\_cv**.**fit(X\_train\_smote, y\_train\_smote)

Fitting 3 folds for each of 6 candidates, totalling 18 fits

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 18 out of 18 | elapsed: 45.8min finished

GridSearchCV(cv=3, error\_score=nan,

estimator=XGBClassifier(base\_score=None, booster=None,

colsample\_bylevel=None,

colsample\_bynode=None,

colsample\_bytree=None, gamma=None,

gpu\_id=None, importance\_type='gain',

interaction\_constraints=None,

learning\_rate=None, max\_delta\_step=None,

max\_depth=2, min\_child\_weight=None,

missing=nan, monotone\_constraints=None,

n\_estimato...

objective='binary:logistic',

random\_state=None, reg\_alpha=None,

reg\_lambda=None, scale\_pos\_weight=None,

subsample=None, tree\_method=None,

validate\_parameters=False,

verbosity=None),

iid='deprecated', n\_jobs=None,

param\_grid={'learning\_rate': [0.2, 0.6],

'subsample': [0.3, 0.6, 0.9]},

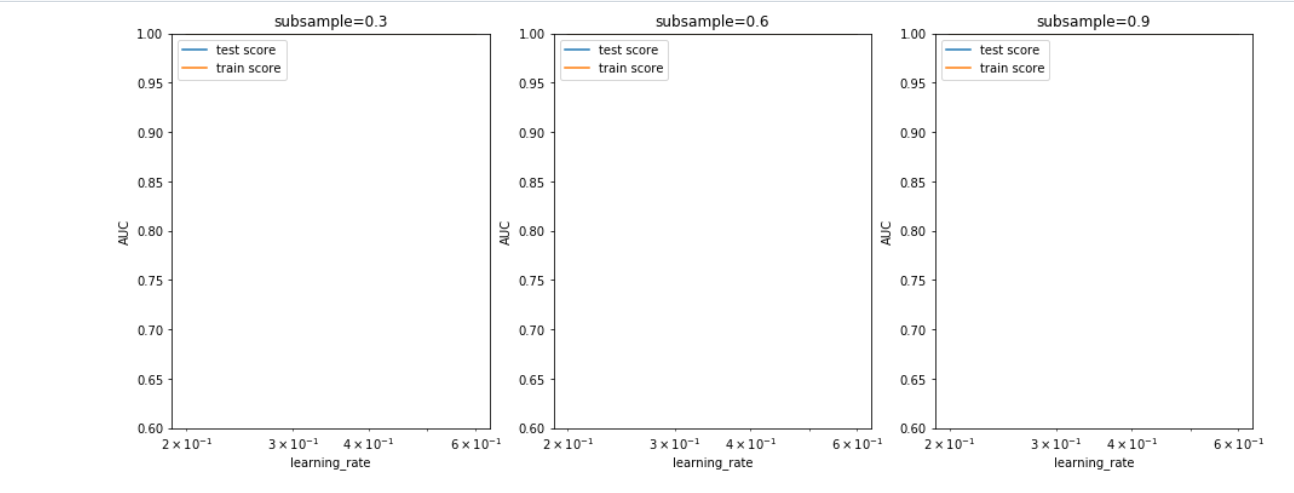
pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=True,

scoring='roc\_auc', verbose=1)

*# cv results*

cv\_results **=** pd**.**DataFrame(model\_cv**.**cv\_results\_)

cv\_results



Model with optimal hyperparameters

We see that the train score almost touches to 1. Among the hyperparameters, we can choose the best parameters as learning\_rate : 0.2 and subsample: 0.3

model\_cv.best\_params\_

{'learning\_rate': 0.6, 'subsample': 0.9}

# chosen hyperparameters

# 'objective':'binary:logistic' outputs probability rather than label, which we need for calculating auc

params = {'learning\_rate': 0.6,

'max\_depth': 2,

'n\_estimators':200,

'subsample':0.9,

'objective':'binary:logistic'}

# fit model on training data

xgb\_bal\_smote\_model = XGBClassifier(params = params)

xgb\_bal\_smote\_model.fit(X\_train\_smote, y\_train\_smote)

XGBClassifier(base\_score=0.5, booster=None, colsample\_bylevel=1,

colsample\_bynode=1, colsample\_bytree=1, gamma=0, gpu\_id=-1,

importance\_type='gain', interaction\_constraints=None,

learning\_rate=0.300000012, max\_delta\_step=0, max\_depth=6,

min\_child\_weight=1, missing=nan, monotone\_constraints=None,

n\_estimators=100, n\_jobs=0, num\_parallel\_tree=1,

objective='binary:logistic',

params={'learning\_rate': 0.6, 'max\_depth': 2, 'n\_estimators': 200,

'objective': 'binary:logistic', 'subsample': 0.9},

random\_state=0, reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1,

subsample=1, tree\_method=None, validate\_parameters=False,

verbosity=None)

Prediction on the train set

# Predictions on the train set

y\_train\_pred = xgb\_bal\_smote\_model.predict(X\_train\_smote)

# Confusion matrix

confusion = metrics.confusion\_matrix(y\_train\_smote, y\_train\_pred)

print(confusion)

[[227447 2]

[ 0 227449]]

TP = confusion[1,1] # true positive

TN = confusion[0,0] # true negatives

FP = confusion[0,1] # false positives

FN = confusion[1,0] # false negatives

# Accuracy

print("Accuracy:-",metrics.accuracy\_score(y\_train\_smote, y\_train\_pred))

# Sensitivity

print("Sensitivity:-",TP / float(TP+FN))

# Specificity

print("Specificity:-", TN / float(TN+FP))

Accuracy:- 0.9999956034099952

Sensitivity:- 1.0

Specificity:- 0.9999912068199904

# classification\_report

print(classification\_report(y\_train\_smote, y\_train\_pred))

precision recall f1-score support

0 1.00 1.00 1.00 227449

1 1.00 1.00 1.00 227449

accuracy 1.00 454898

macro avg 1.00 1.00 1.00 454898

weighted avg 1.00 1.00 1.00 454898

# Predicted probability

y\_train\_pred\_proba = xgb\_bal\_smote\_model.predict\_proba(X\_train\_smote)[:,1]

# roc\_auc

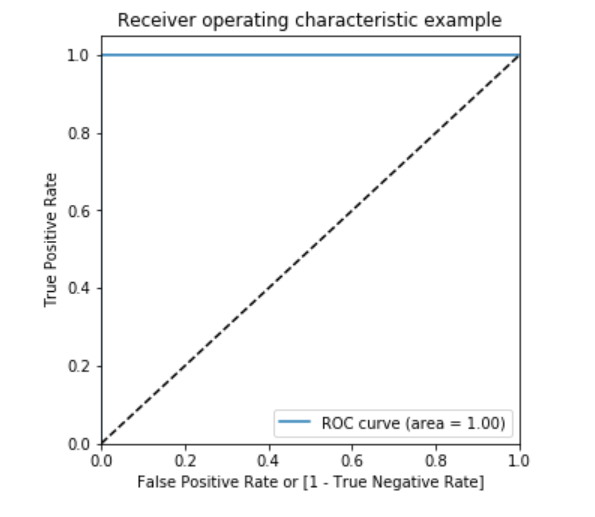
auc = metrics.roc\_auc\_score(y\_train\_smote, y\_train\_pred\_proba)

auc

1.0

# Plot the ROC curve

draw\_roc(y\_train\_smote, y\_train\_pred\_proba)



#### Prediction on the test set

*# Prediction on the test set*

y\_test\_pred **=** logistic\_bal\_adasyn\_model**.**predict(X\_test)

*# Confusion matrix*

confusion **=** metrics**.**confusion\_matrix(y\_test, y\_test\_pred)

print(confusion)

[[51642 5224]

[ 4 92]]

TP **=** confusion[1,1] *# true positive*

TN **=** confusion[0,0] *# true negatives*

FP **=** confusion[0,1] *# false positives*

FN **=** confusion[1,0] *# false negatives*

*# Accuracy*

print("Accuracy:-",metrics**.**accuracy\_score(y\_test, y\_test\_pred))

*# Sensitivity*

print("Sensitivity:-",TP **/** float(TP**+**FN))

*# Specificity*

print("Specificity:-", TN **/** float(TN**+**FP))

Accuracy:- 0.9082195147642288

Sensitivity:- 0.9583333333333334

Specificity:- 0.9081349136566665

*# classification\_report*

print(classification\_report(y\_test, y\_test\_pred))

precision recall f1-score support

0 1.00 0.91 0.95 56866

1 0.02 0.96 0.03 96

accuracy 0.91 56962

macro avg 0.51 0.93 0.49 56962

weighted avg 1.00 0.91 0.95 56962

*# Predicted probability*

y\_test\_pred\_proba **=** logistic\_bal\_adasyn\_model**.**predict\_proba(X\_test)[:,1]

*# roc\_auc*

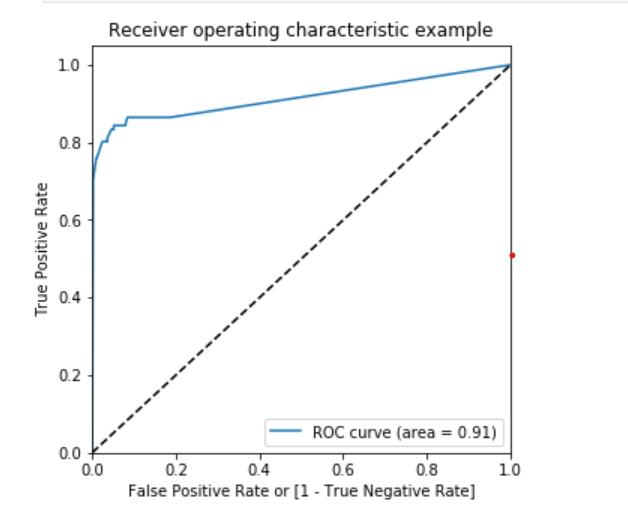
auc **=** metrics**.**roc\_auc\_score(y\_test, y\_test\_pred\_proba)

auc

0.9671573487086602

*# Plot the ROC curve*

draw\_roc(y\_test, y\_test\_pred\_proba)



\***Model summary**\*

* Train set
  + Accuracy = 0.88
  + Sensitivity = 0.86
  + Specificity = 0.91
  + ROC = 0.96
* Test set
  + Accuracy = 0.90
  + Sensitivity = 0.95
  + Specificity = 0.90
  + ROC = 0.97

### Decision Tree

*# Create the parameter grid*

param\_grid **=** {

'max\_depth': range(5, 15, 5),

'min\_samples\_leaf': range(50, 150, 50),

'min\_samples\_split': range(50, 150, 50),

}

*# Instantiate the grid search model*

dtree **=** DecisionTreeClassifier()

grid\_search **=** GridSearchCV(estimator **=** dtree,

param\_grid **=** param\_grid,

scoring**=** 'roc\_auc',

cv **=** 3,

verbose **=** 1)

*# Fit the grid search to the data*

grid\_search**.**fit(X\_train\_adasyn,y\_train\_adasyn)

Fitting 3 folds for each of 8 candidates, totalling 24 fits

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 24 out of 24 | elapsed: 5.4min finished

GridSearchCV(cv=3, error\_score=nan,

estimator=DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None,

criterion='gini', max\_depth=None,

max\_features=None,

max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0,

min\_impurity\_split=None,

min\_samples\_leaf=1,

min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0,

presort='deprecated',

random\_state=None,

splitter='best'),

iid='deprecated', n\_jobs=None,

param\_grid={'max\_depth': range(5, 15, 5),

'min\_samples\_leaf': range(50, 150, 50),

'min\_samples\_split': range pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=False,

scoring='roc\_auc', verbose=1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

*# cv results*

cv\_results **=** pd**.**DataFrame(grid\_search**.**cv\_results\_)

cv\_results

*# Printing the optimal sensitivity score and hyperparameters*

print("Best roc\_auc:-", grid\_search**.**best\_score\_)

print(grid\_search**.**best\_estimator\_)

Best roc\_auc:- 0.9414793563319087

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',

max\_depth=10, max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=100, min\_samples\_split=50,

min\_weight\_fraction\_leaf=0.0, presort='deprecated',

random\_state=None, splitter='best')

*# Model with optimal hyperparameters*

dt\_bal\_adasyn\_model **=** DecisionTreeClassifier(criterion **=** "gini",

random\_state **=** 100,

max\_depth**=**10,

min\_samples\_leaf**=**100,

min\_samples\_split**=**50)

dt\_bal\_adasyn\_model**.**fit(X\_train\_adasyn, y\_train\_adasyn)

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',

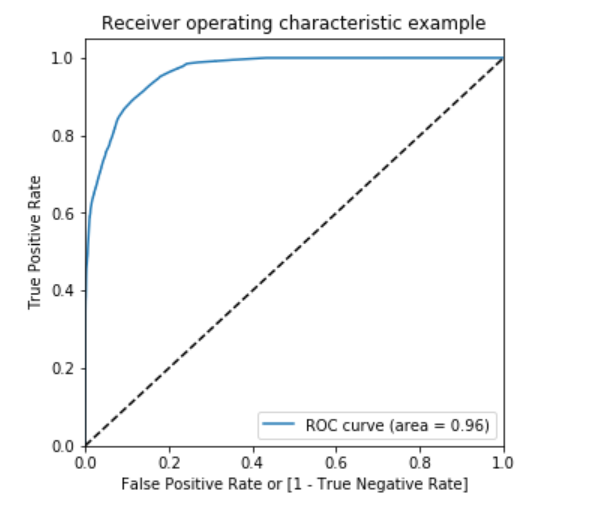
max\_depth=10, max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=100, min\_samples\_split=50,

min\_weight\_fraction\_leaf=0.0, presort='deprecated',

random\_state=100, splitter='best')



##### Prediction on the train set

*# Predictions on the train set*

y\_train\_pred **=** dt\_bal\_adasyn\_model**.**predict(X\_train\_adasyn)

*# Confusion matrix*

confusion **=** metrics**.**confusion\_matrix(y\_train\_adasyn, y\_train\_pred)

print(confusion)

[[215929 11520]

[ 1118 226330]]

TP **=** confusion[1,1] *# true positive*

TN **=** confusion[0,0] *# true negatives*

FP **=** confusion[0,1] *# false positives*

FN **=** confusion[1,0] *# false negatives*

*# Accuracy*

print("Accuracy:-",metrics**.**accuracy\_score(y\_train\_adasyn, y\_train\_pred))

*# Sensitivity*

print("Sensitivity:-",TP **/** float(TP**+**FN))

*# Specificity*

print("Specificity:-", TN **/** float(TN**+**FP))

Accuracy:- 0.9722178866864367

Sensitivity:- 0.9950845907636031

Specificity:- 0.9493512831447929

*# classification\_report*

print(classification\_report(y\_train\_adasyn, y\_train\_pred))

precision recall f1-score support

0 0.99 0.95 0.97 227449

1 0.95 1.00 0.97 227448

accuracy 0.97 454897

macro avg 0.97 0.97 0.97 454897

weighted avg 0.97 0.97 0.97 454897

*# Predicted probability*

y\_train\_pred\_proba **=** dt\_bal\_adasyn\_model**.**predict\_proba(X\_train\_adasyn)[:,1]

*# roc\_auc*

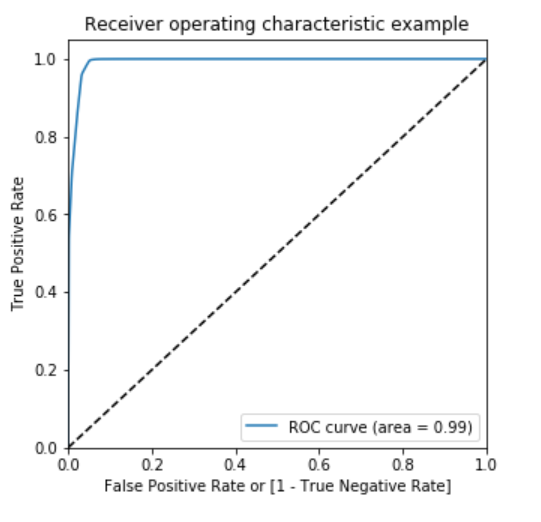
auc **=** metrics**.**roc\_auc\_score(y\_train\_adasyn, y\_train\_pred\_proba)

auc

0.9917591040224101

*# Plot the ROC curve*

draw\_roc(y\_train\_adasyn, y\_train\_pred\_proba)



##### Prediction on the test set

*# Predictions on the test set*

y\_test\_pred **=** dt\_bal\_adasyn\_model**.**predict(X\_test)

*# Confusion matrix*

confusion **=** metrics**.**confusion\_matrix(y\_test, y\_test\_pred)

print(confusion)

[[53880 2986]

[ 15 81]]

TP **=** confusion[1,1] *# true positive*

TN **=** confusion[0,0] *# true negatives*

FP **=** confusion[0,1] *# false positives*

FN **=** confusion[1,0] *# false negatives*

*# Accuracy*

print("Accuracy:-",metrics**.**accuracy\_score(y\_test, y\_test\_pred))

*# Sensitivity*

print("Sensitivity:-",TP **/** float(TP**+**FN))

*# Specificity*

print("Specificity:-", TN **/** float(TN**+**FP))

Accuracy:- 0.9473157543625575

Sensitivity:- 0.84375

Specificity:- 0.9474905919178419

*# classification\_report*

print(classification\_report(y\_test, y\_test\_pred))

precision recall f1-score support

0 1.00 0.95 0.97 56866

1 0.03 0.84 0.05 96

accuracy 0.95 56962

macro avg 0.51 0.90 0.51 56962

weighted avg 1.00 0.95 0.97 56962

*# Predicted probability*

y\_test\_pred\_proba **=** dt\_bal\_adasyn\_model**.**predict\_proba(X\_test)[:,1]

*# roc\_auc*

auc **=** metrics**.**roc\_auc\_score(y\_test, y\_test\_pred\_proba)

auc

0.9141440147305362

*# Plot the ROC curve*

draw\_roc(y\_test, y\_test\_pred\_proba)

\***Model summary**\*

* Train set
  + Accuracy = 0.97
  + Sensitivity = 0.99
  + Specificity = 0.95
  + ROC-AUC = 0.99
* Test set
  + Accuracy = 0.95
  + Sensitivity = 0.84
  + Specificity = 0.95
  + ROC-AUC = 0.91

### XGBoost

*# hyperparameter tuning with XGBoost*

*# creating a KFold object*

folds **=** 3

*# specify range of hyperparameters*

param\_grid **=** {'learning\_rate': [0.2, 0.6],

'subsample': [0.3, 0.6, 0.9]}

*# specify model*

xgb\_model **=** XGBClassifier(max\_depth**=**2, n\_estimators**=**200)

*# set up GridSearchCV()*

model\_cv **=** GridSearchCV(estimator **=** xgb\_model,

param\_grid **=** param\_grid,

scoring**=** 'roc\_auc',

cv **=** folds,

verbose **=** 1,

return\_train\_score**=True**)

*# fit the model*

model\_cv**.**fit(X\_train\_adasyn, y\_train\_adasyn)

Fitting 3 folds for each of 6 candidates, totalling 18 fits

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 18 out of 18 | elapsed: 42.5min finished

GridSearchCV(cv=3, error\_score=nan,

estimator=XGBClassifier(base\_score=None, booster=None,

colsample\_bylevel=None,

colsample\_bynode=None,

colsample\_bytree=None, gamma=None,

gpu\_id=None, importance\_type='gain',

interaction\_constraints=None,

learning\_rate=None, max\_delta\_step=None,

max\_depth=2, min\_child\_weight=None,

missing=nan, monotone\_constraints=None,

n\_estimato...

objective='binary:logistic',

random\_state=None, reg\_alpha=None,

reg\_lambda=None, scale\_pos\_weight=None,

subsample=None, tree\_method=None,

validate\_parameters=False,

verbosity=None),

iid='deprecated', n\_jobs=None,

param\_grid={'learning\_rate': [0.2, 0.6],

'subsample': [0.3, 0.6, 0.9]},

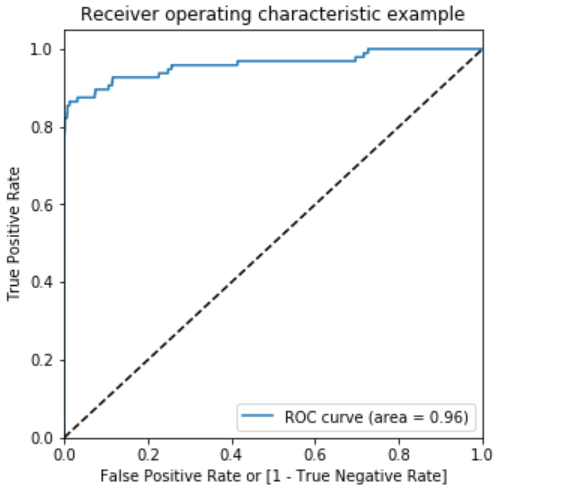
pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=True,

scoring='roc\_auc', verbose=1)

*# cv results*

cv\_results **=** pd**.**DataFrame(model\_cv**.**cv\_results\_)

cv\_results



Model summary\*

Train set

Accuracy = 0.99

Sensitivity = 1.0

Specificity = 1.0

ROC-AUC = 1.0

Test set

Accuracy = 0.99

Sensitivity = 0.78

Specificity = 0.99

ROC-AUC = 0.96

Choosing best model on the balanced data

He we balanced the data with various approach such as Undersampling, Oversampling, SMOTE and Adasy. With every data balancing thechnique we built several models such as Logistic, XGBoost, Decision Tree, and Random Forest.

We can see that almost all the models performed more or less good. But we should be interested in the best model.

Though the Undersampling technique models performed well, we should keep mind that by doing the undersampling some imformation were lost. Hence, it is better not to consider the undersampling models.

Whereas the SMOTE and Adasyn models performed well. Among those models the simplist model Logistic regression has ROC score 0.99 in the train set and 0.97 on the test set. We can consider the Logistic model as the best model to choose because of the easy interpretation of the models and also the resourse requirements to build the mdoel is lesser than the other heavy models such as Random forest or XGBoost.

Hence, we can conclude that the Logistic regression model with SMOTE is the best model for its simlicity and less resource requirement.

Print the FPR,TPR & select the best threshold from the roc curve for the best model

print('Train auc =', metrics.roc\_auc\_score(y\_train\_smote, y\_train\_pred\_proba\_log\_bal\_smote))

fpr, tpr, thresholds = metrics.roc\_curve(y\_train\_smote, y\_train\_pred\_proba\_log\_bal\_smote)

threshold = thresholds[np.argmax(tpr-fpr)]

print("Threshold=",threshold)

Train auc = 0.9897539730968845

Threshold= 0.5311563613510013

We can see that the threshold is 0.53, for which the TPR is the highest and FPR is the lowest and we got the best ROC score.