CPSC-6300

Insurance Fraud Detection

Checkpoint 2

Dineshchandar Ravichandran
dravich@g.clemson.edu
Archana Lalji Saroj
asaroj@g.clemson.edu
Prashanth Reddy Kadire
pkadire@g.clemson.edu

Contents

Model choice justification

Our fraudulent claim data set comprises 1000 data points where only 24.70% of data are fraudulent, making the data set imbalanced. Furthermore, we also have labels stating which claims are fraudulent. As stated in checkpoint1, our project aims to create a classification model to classify genuine and fraudulent claims. Post going through a few of the existing works pertaining to statistical analysis and prediction on tabular data, as well as machine learning-based fraud detection models. We have narrowed our initial model implementation and prediction to the following:

1. XGBoost.
2. SVM.
3. Logistic regression.

XG-Boost:

XG-boost is widely used for statistical analysis and prediction, often time outperforming deep learning models[1]. Furthermore, we referred to similar works on fraud detection, like credit-card fraud detection[2] and auto-fraud detection paper. Which had also selected XG-Boost as it is one of the most efficient implementations of gradient-boosted decision trees and has been regarded as one of the best machine-learning algorithms, often used in Kaggle competitions[3]. Specifically designed to optimize memory usage and exploit the hardware computing power, XG-boost decreases the execution time with increased performance.

The main idea of boosting is to sequentially build sub-trees from an original tree such that each subsequent tree reduces the errors of the previous one. In such a way, the new sub-trees will update the previous residuals to reduce the error of the cost function. The following properties of XG boot further make it a staple for these classifications:

1. XG-boost is also a highly scalable end-to-end tree-boosting system.
2. The justified weighted quantile sketch is used for efficient proposal calculation.
3. The sparsity-aware algorithm is developed for parallel tree learning.
4. An effective cache-aware block structure is implemented for out-of-core tree learning.

Due to these pros, XG-boost outperforms most other machine learning algorithms in speed and accuracy.

SVM:

Support vector machines (SVMs) are supervised machine learning models that apply classification methods to two-group classification issues. The support vector machine algorithm aims to locate a hyperplane in N-dimensional space (N is the number of features) that categorizes the data points.

[1] Deep Learning, XGBoost Or Both: What Works Best For Tabular Data? (analyticsindiamag.com)
[2] LEI, Shimin & XU, Ke & HUANG, YiZhe & SHA, Xinye. (2020). An Xgboost based system for financial fraud detection. E3S Web of Conferences. 214. 02042. 10.1051/e3sconf/202021402042.
3 M. Chen, Q. Liu, S. Chen, Y. Liu, C. Zhang, and R. Liu, “XGBoostbased algorithm interpretation and application on post-fault transient stability status prediction of power system,” IEEE Access, vol. 7, pp. 13 149–13 158, January 2019.

Furthermore, SVMs are relatively memory efficient, do not experience overfitting, and perform well when there is a distinct indication of class separation. When the total number of samples is fewer than the total number of dimensions, SVM can be used and does well in terms of memory.

Various algorithms are used in machine learning for classification; however, SVM is better than most others since it produces results with higher accuracy.

SVM Classifier, compared to other classifiers, has better computational complexity. Even if the number of positive and negative examples are not the same, SVM can be used as it can normalize the data or project into the space of the decision boundary separating the two classes.

Due to these pros, we have chosen the SVM as one of the models for fraud detection.

Logistic regression:

Logistic regression is one of the most common algorithms used for classification models. It is a mathematical model used in statistics to estimate the probability of an event occurring, having been given some previous data. We are using insurance data to classify claims as fraudulent or non-fraudulent operating factors like claim amount, claim type, gender, age, and others.

Unlike decision trees or support vector machines, the logistic regression approach enables models to be quickly changed to reflect new data. It is possible to update using stochastic gradient descent.

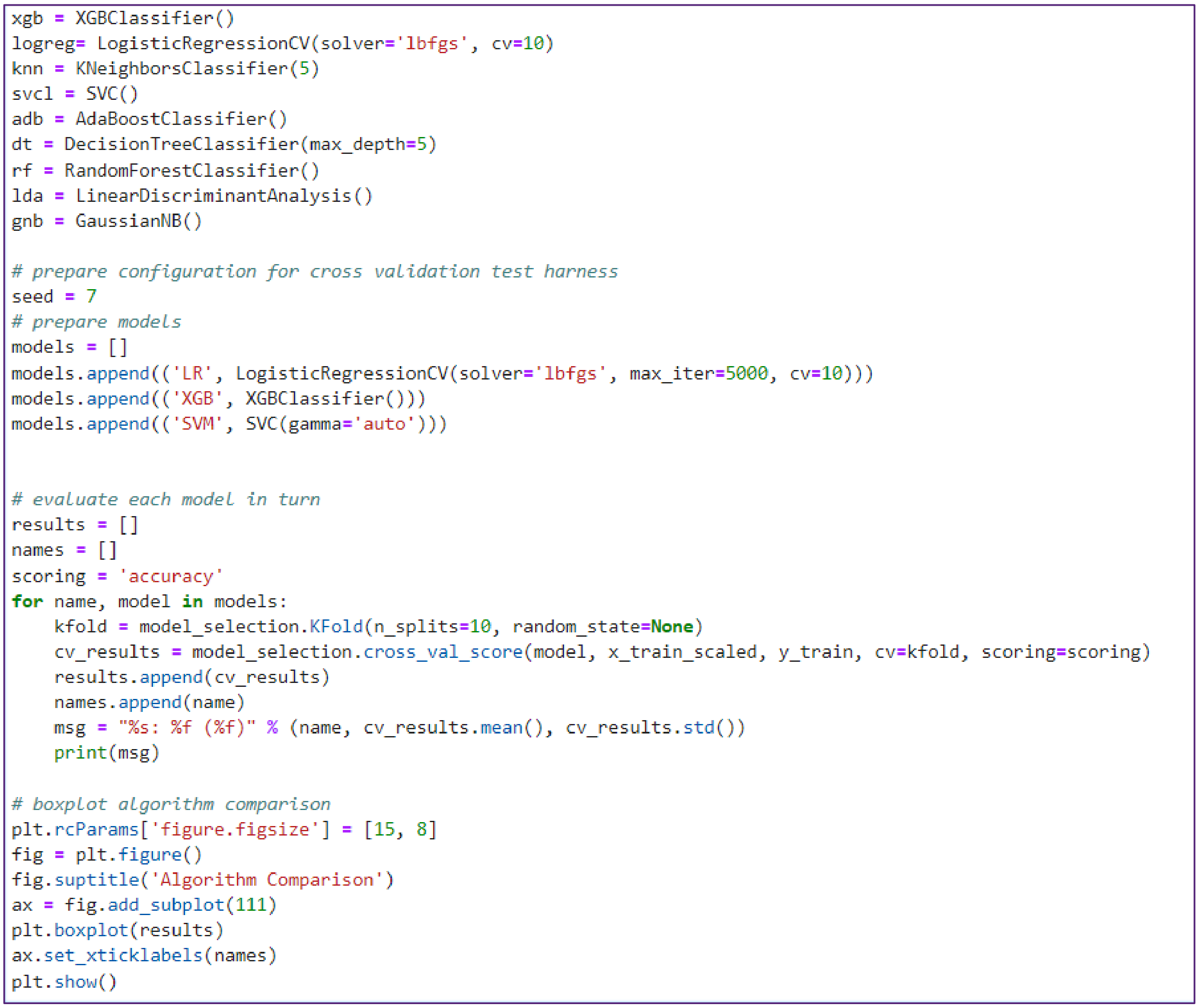
Logistic regression is less prone to overfitting in a low-dimensional dataset with sufficient training examples. Logistic regression proves very efficient when the dataset has linearly separable features.

The predicted parameters (trained weights) give an inference about the importance of each feature. The direction of the association, i.e., positive or negative, is also given. So, we can use logistic regression to determine the features' relationship.

Due to these pros, we have chosen the Logistic Regression model as one of the models for fraud detection.

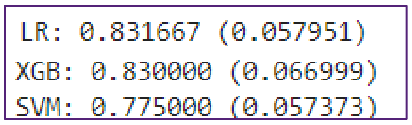
Chosen model's test error rate

As illustrated below XG-boost has better scores for training data, hence we will perform further training and testing using XG-Boost:



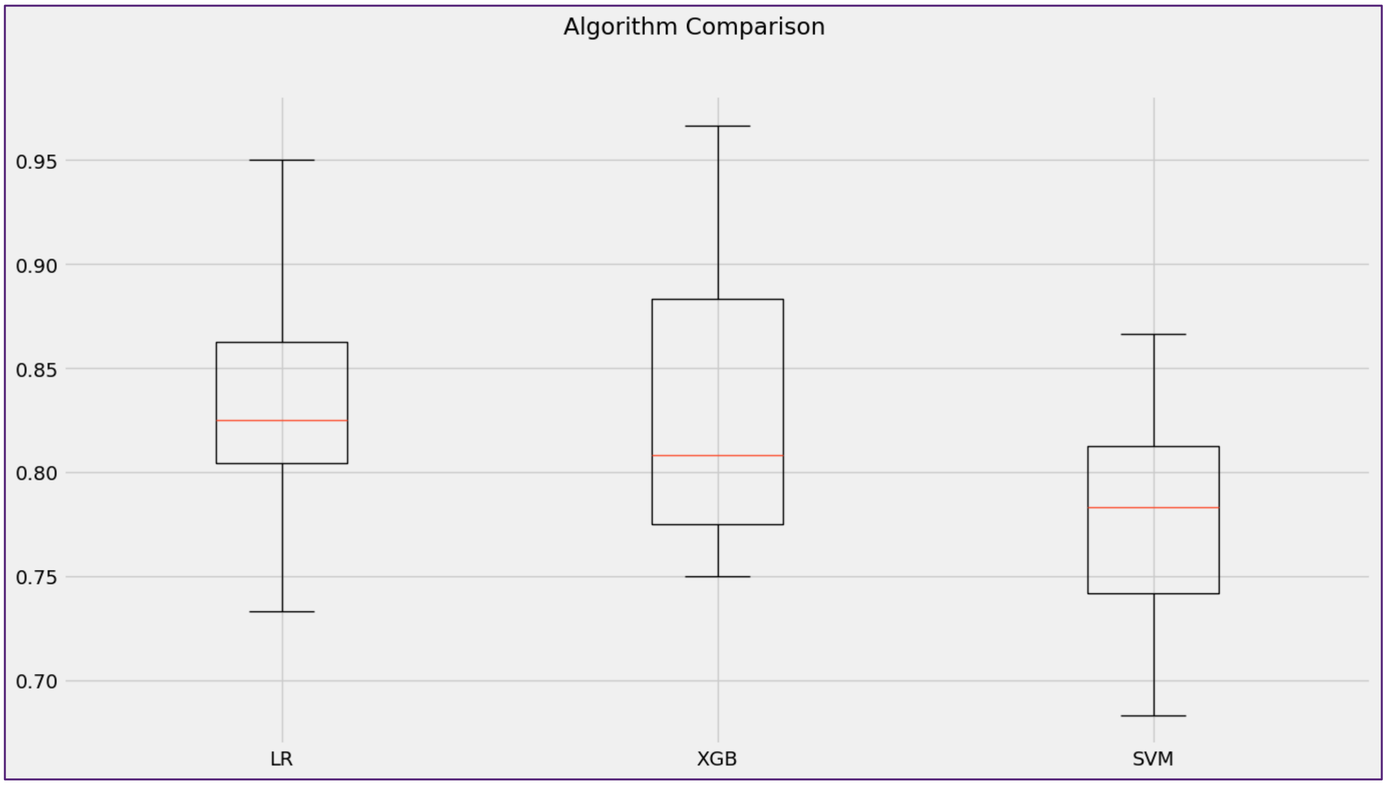
```
xgb = XGBClassifier()
logreg= LogisticRegressionCV(solver='lbfgs', cv=10)
knn = KNeighborsClassifier(5)
svcl = SVC()
adb = AdaBoostClassifier()
dt = DecisionTreeClassifier(max_depth=5)
rf = RandomForestClassifier()
lda = LinearDiscriminantAnalysis()
gnb = GaussianNB()

# prepare configuration for cross validation test harness
seed = 7
# prepare models
models = []
models.append(('LR', LogisticRegressionCV(solver='lbfgs', max_iter=5000, cv=10)))
models.append(('XGB', XGBClassifier()))
models.append(('SVM', SVC(gamma='auto')))


# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=None)
    cv_results = model_selection.cross_val_score(model, x_train_scaled, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

# boxplot algorithm comparison
plt.rcParams['figure.figsize'] = [15, 8]
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

Result:



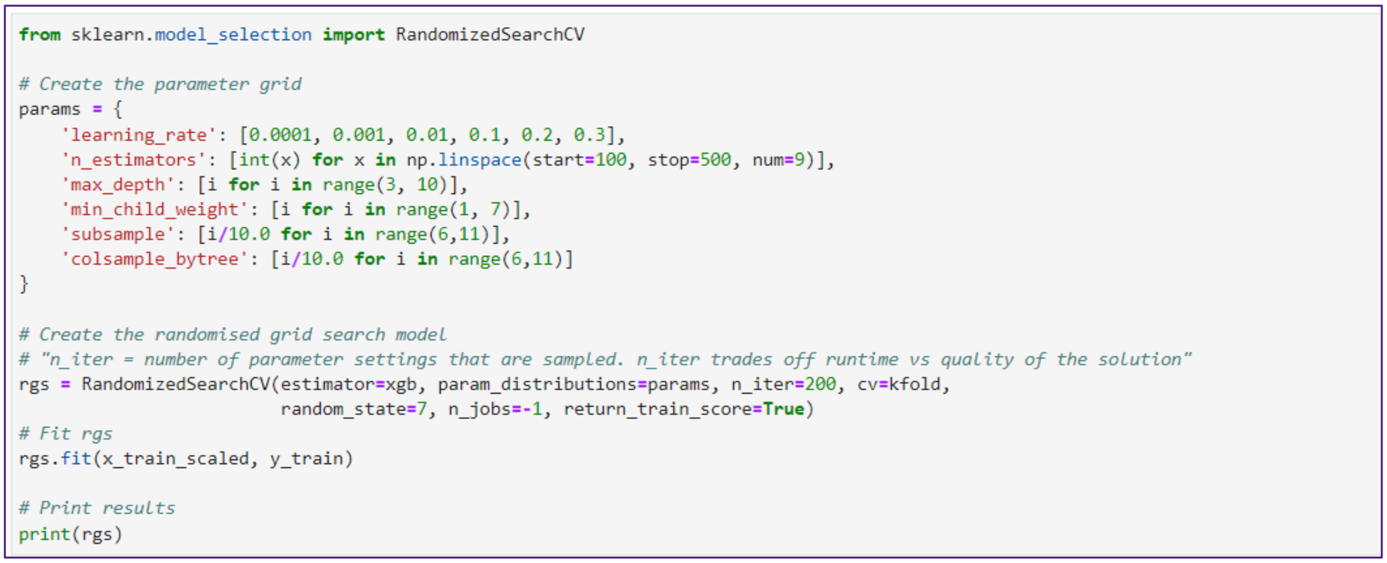
```
LR: 0.831667 (0.057951)
XGB: 0.830000 (0.066999)
SVM: 0.775000 (0.057373)
```

The following graph illustrates the performance of Logistic regression, XG-Boost, and SVM model on train data:



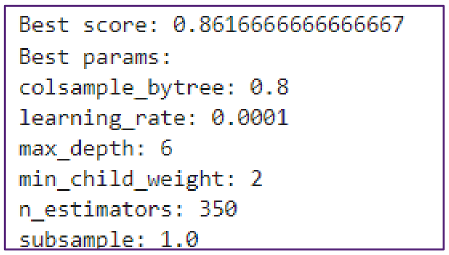
Performance of model based on the test error rate

Following the initial accuracy score, we can determine that XG-boost performs better than SVM and Logistic regression based on STD scores. So we proceeded to use XG-boost classification for our predictions and to improve the model's performance further, we performed hyperparameter tunning with Random Grid Search.

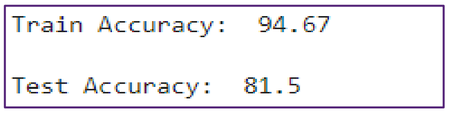


```python
from sklearn.model_selection import RandomizedSearchCV

# Create the parameter grid
params = {
    'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
    'n_estimators': [int(x) for x in np.linspace(start=100, stop=500, num=9)],
    'max_depth': [i for i in range(3, 10)],
    'min_child_weight': [i for i in range(1, 7)],
    'subsample': [i/10.0 for i in range(6,11)],
    'colsample_bytree': [i/10.0 for i in range(6,11)]
}

# Create the randomised grid search model
# "n_iter = number of parameter settings that are sampled. n_iter trades off runtime vs quality of the solution"
rgs = RandomizedSearchCV(estimator=xgb, param_distributions=params, n_iter=200, cv=kfold,
                         random_state=7, n_jobs=-1, return_train_score=True)
# Fit rgs
rgs.fit(x_train_scaled, y_train)

# Print results
print(rgs)
```

Based on this following best parameter were received, and the XG-boost was further trained with these hyperparameters to improve its overall performance, with a train accuracy increase of 11.67%.

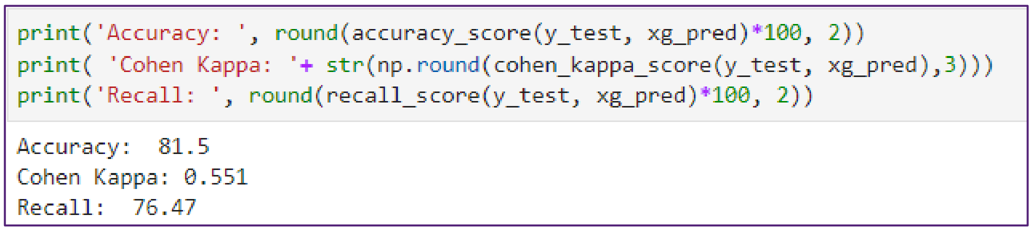


```
Best score: 0.8616666666666667
Best params:
colsample_bytree: 0.8
learning_rate: 0.0001
max_depth: 6
min_child_weight: 2
n_estimators: 350
subsample: 1.0
```



```
Train Accuracy:   94.67

Test Accuracy:   81.5
```

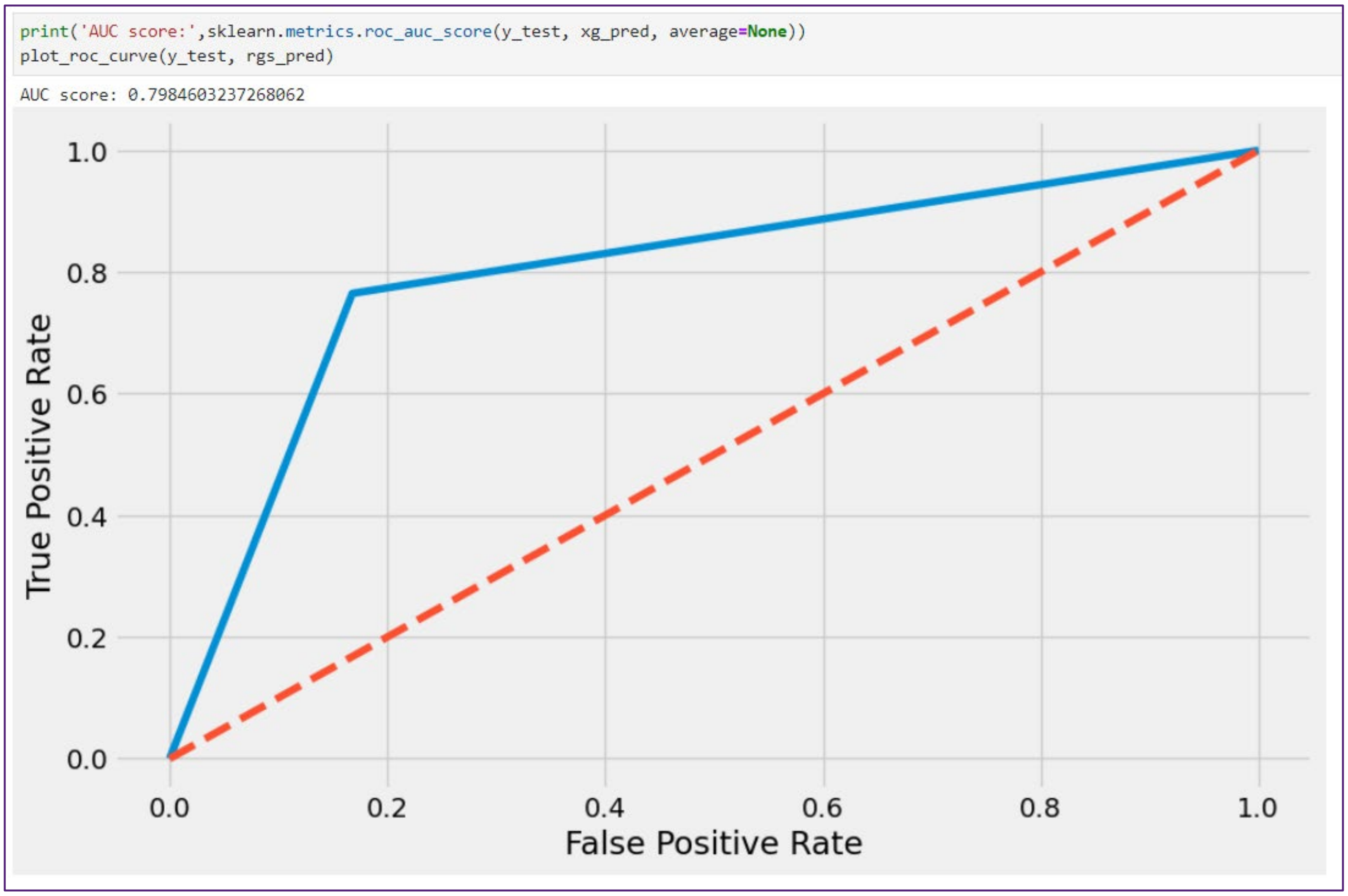
Following is the performance of tuned XG-boost on test data, where it is generating an accuracy of 81.5% and recall of 76.47%:



```python
print('Accuracy: ', round(accuracy_score(y_test, xg_pred)*100, 2))
print( 'Cohen Kappa: '+ str(np.round(cohen_kappa_score(y_test, xg_pred),3)))
print('Recall: ', round(recall_score(y_test, xg_pred)*100, 2))
```

```
Accuracy:  81.5
Cohen Kappa: 0.551
Recall:  76.47
```

The model is achieving an AUC score of 79.85% with the following ROC:

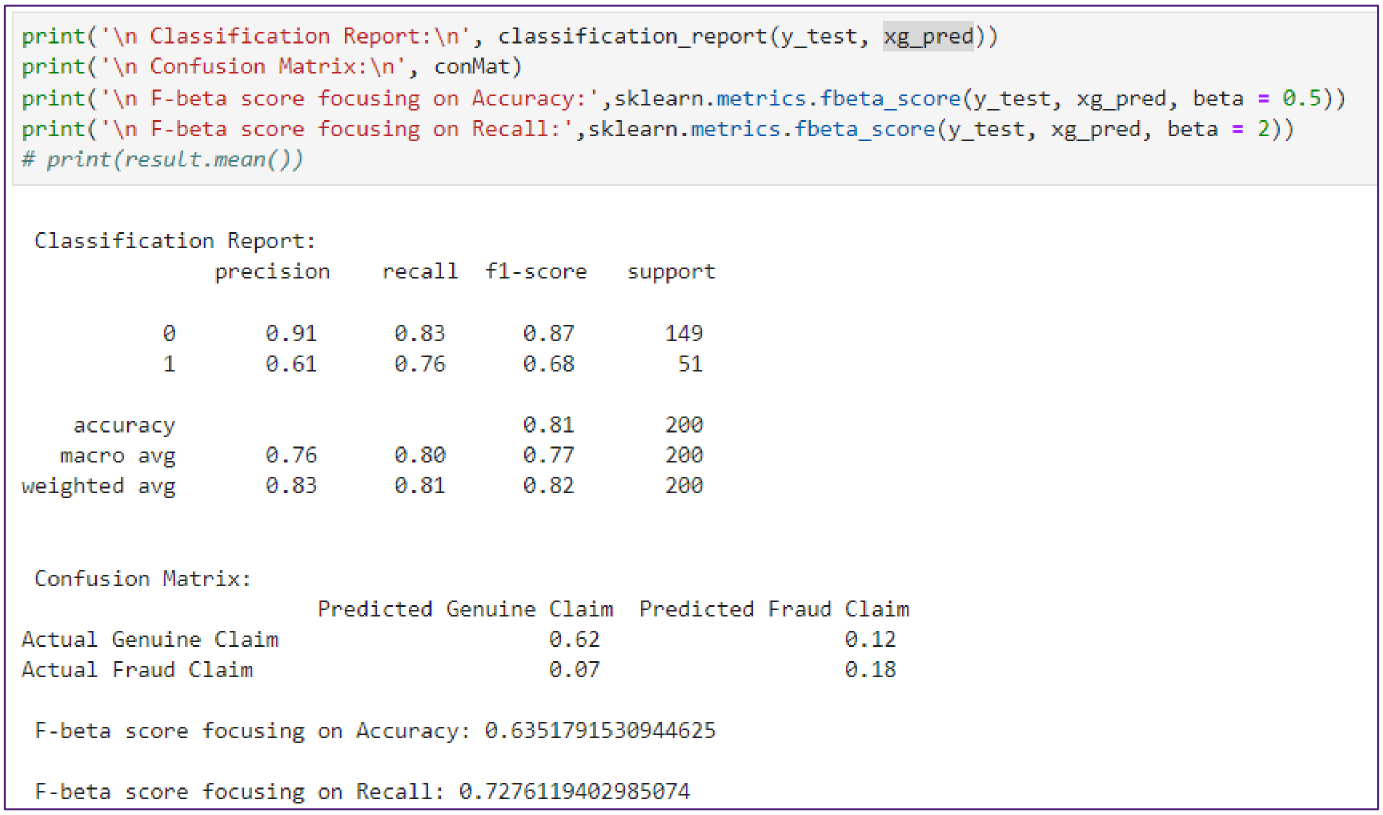


```
print('AUC score:',sklearn.metrics.roc_auc_score(y_test, xg_pred, average=None))
plot_roc_curve(y_test, rgs_pred)
```

```
AUC score: 0.7984603237268062
```

Predictions of model

We will be checking the performance of predictions of the XG-Boost based on: F1-score, recall, precision, F-beta, and confusion matrix:



```
print('\n Classification Report:\n', classification_report(y_test, xg_pred))
print('\n Confusion Matrix:\n', conMat)
print('\n F-beta score focusing on Accuracy:',sklearn.metrics.fbeta_score(y_test, xg_pred, beta = 0.5))
print('\n F-beta score focusing on Recall:',sklearn.metrics.fbeta_score(y_test, xg_pred, beta = 2))
# print(result.mean())
```

```
 Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.83      0.87       149
           1       0.61      0.76      0.68        51

    accuracy                           0.81       200
   macro avg       0.76      0.80      0.77       200
weighted avg       0.83      0.81      0.82       200


 Confusion Matrix:
                       Predicted Genuine Claim  Predicted Fraud Claim
Actual Genuine Claim                      0.62                   0.12
Actual Fraud Claim                        0.07                   0.18

 F-beta score focusing on Accuracy: 0.6351791530944625

 F-beta score focusing on Recall: 0.7276119402985074
```

As illustrated above, XG-boost has a precision of 61% and an F-1 score of 68% for fraudulent claims. Additionally, from the confusion matrix, we can observe that the model categorizes 7% of fraudulent claims as genuine claims and has an f-beta score of 72.7 % on recall. These scores show that the model can detect most fraudulent cases.

Goals for Checkpoint-3

Going further, we can focus on improving the precision and f-beta score on recall, as classifying genuine claims as fraud would be acceptable compared to classifying fraudulent claims as genuine claims. The general design of our fraud detection system will be to raise all fraudulent cases to a human verifier. In case of misclassified genuine claims, the verifiers can rectify and approve them.