CPSC-6300

Insurance Fraud Detection

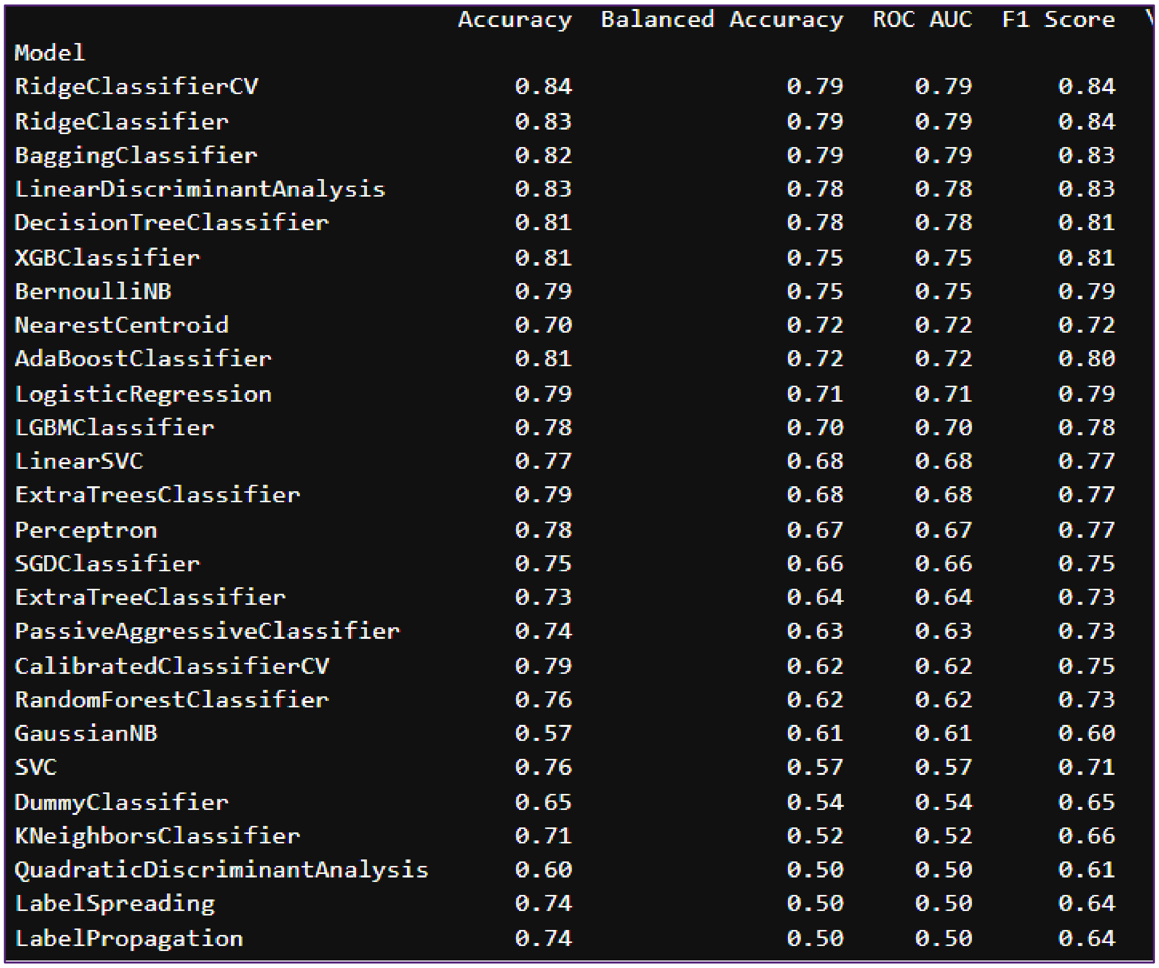
Checkpoint 3

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

Contents

Alternative model choice justification

For checkpoint 2, we achieved a test accuracy of 81.5% with XGBoost and data feature engineering. To further improve our scores, we checked with other classifiers and employed LazyClassifier and based on which we were able to generate the following scores for our data:



```
                               Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
RidgeClassifierCV                  0.84               0.79     0.79      0.84
RidgeClassifier                    0.83               0.79     0.79      0.84
BaggingClassifier                  0.82               0.79     0.79      0.83
LinearDiscriminantAnalysis         0.83               0.78     0.78      0.83
DecisionTreeClassifier             0.81               0.78     0.78      0.81
XGBClassifier                      0.81               0.75     0.75      0.81
BernoulliNB                        0.79               0.75     0.75      0.79
NearestCentroid                    0.70               0.72     0.72      0.72
AdaBoostClassifier                 0.81               0.72     0.72      0.80
LogisticRegression                 0.79               0.71     0.71      0.79
LGBMClassifier                     0.78               0.70     0.70      0.78
LinearSVC                          0.77               0.68     0.68      0.77
ExtraTreesClassifier               0.79               0.68     0.68      0.77
Perceptron                         0.78               0.67     0.67      0.77
SGDClassifier                      0.75               0.66     0.66      0.75
ExtraTreeClassifier                0.73               0.64     0.64      0.73
PassiveAggressiveClassifier        0.74               0.63     0.63      0.73
CalibratedClassifierCV             0.79               0.62     0.62      0.75
RandomForestClassifier             0.76               0.62     0.62      0.73
GaussianNB                         0.57               0.61     0.61      0.60
SVC                                0.76               0.57     0.57      0.71
DummyClassifier                    0.65               0.54     0.54      0.65
KNeighborsClassifier               0.71               0.52     0.52      0.66
QuadraticDiscriminantAnalysis      0.60               0.50     0.50      0.61
LabelSpreading                     0.74               0.50     0.50      0.64
LabelPropagation                   0.74               0.50     0.50      0.64
```

Based on the above scores, we have decided to implement the following models and methods to improve our overall performance:

Models:

- Ridge Classifier CV
- Bagging Classifier

Methods:

- Tsfresh

Ridge Classifier:

The Ridge Classifier, based on the Ridge regression method, converts the label data into [-1, 1] and solves the problem with the regression method. The highest value in prediction is accepted as a target class, and for multiclass data, multi. In machine learning, ridge classification is a technique used to analyze linear discriminant models. It is a form of regularization that penalizes model coefficients to prevent overfitting. Overfitting is a common issue in machine learning that occurs when a model is too complex and captures noise in the data instead of the underlying signal. This can lead to poor generalization performance on new data. Ridge classification addresses this problem by adding a penalty term to the cost function that discourages complexity. This results in a model that is better able to generalize to new data.

Ridge classification works by adding a penalty term to the cost function that discourages complexity. The penalty term is typically the sum of the squared coefficients of the features in the model. This forces the coefficients to remain small, which prevents overfitting. The amount of regularization can be controlled by changing the penalty term. A larger penalty results in more regularization and smaller coefficient values. This can be beneficial when there is little training data available. However, if the penalty term is too large, it can result in underfitting.

The loss function of the Ridge classifier is not cross-entropy loss like Logistic Regression. Rather the loss function is mean square loss with an L2 penalty. It works in the following manner for the binary classification problems by making use of the Ridge regression algorithm:

Converts the target variable into +1 and -1 appropriately

Train a Ridge model with loss function as mean square loss with L2 regularization (ridge) as penalty term

During prediction, if the predicted value is less than 0, it predicted class label is -1 otherwise the predicted class label is +1.

The cost function for ridge:

Min (||Y – X(theta)||^2 + λ||theta||^2)

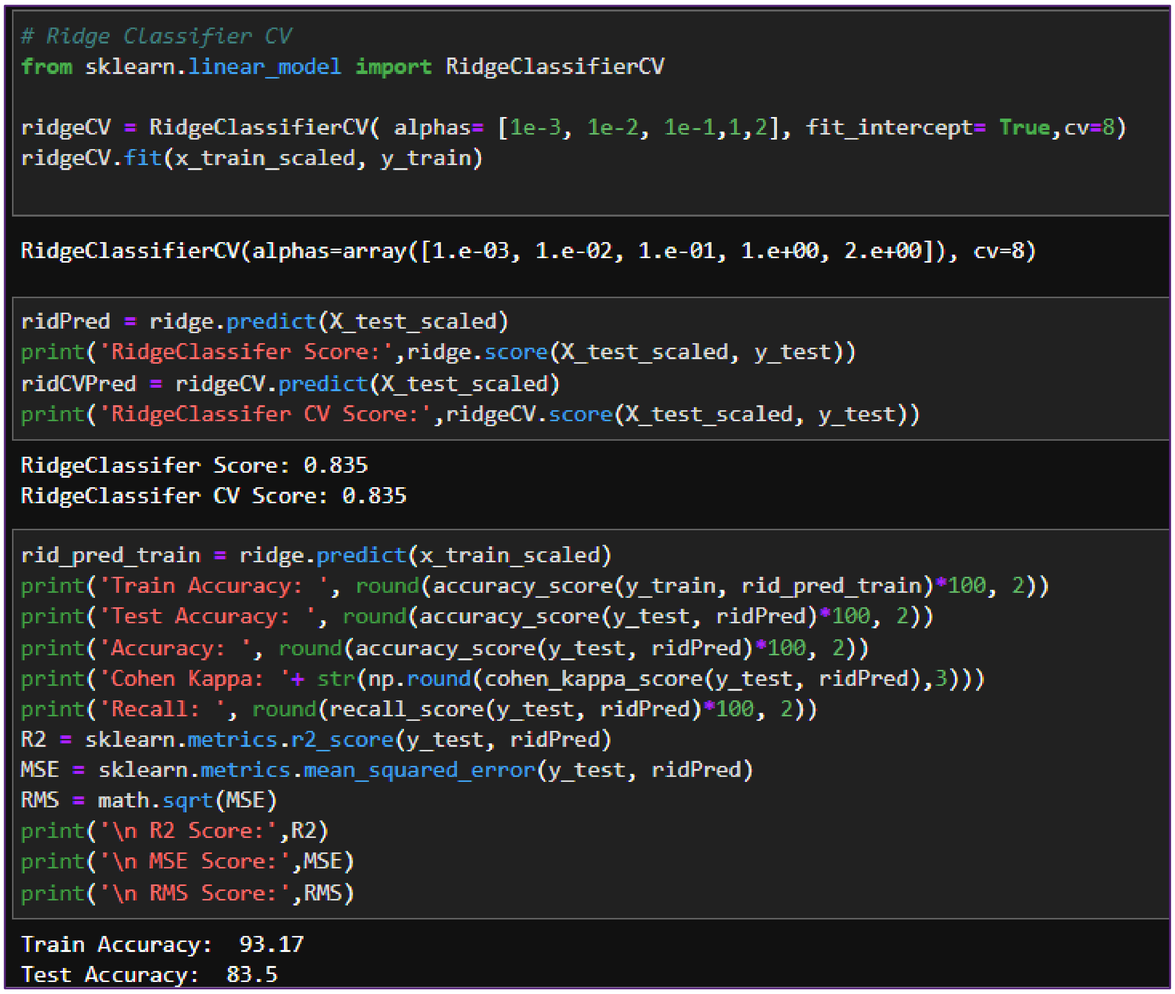
Lambda is the penalty term. λ given here is denoted by an alpha parameter in the ridge function. So, by changing the values of alpha, we are controlling the penalty term. The higher the value, the more significant the penalty, and therefore the magnitude of coefficients is reduced.
It shrinks the parameters. Therefore, it is used to prevent multicollinearity
It reduces the model complexity by coefficient shrinkage.

Bagging Classifier:

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregates their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.
Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples (or data) from the original training dataset – where N is the size of the original training set. The training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set, while others may be left out.
Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance.

Chosen model's test scores

Ridge Classifier



```
# Ridge Classifier CV
from sklearn.linear_model import RidgeClassifierCV

ridgeCV = RidgeClassifierCV( alphas= [1e-3, 1e-2, 1e-1,1,2], fit_intercept= True,cv=8)
ridgeCV.fit(x_train_scaled, y_train)
```

```
RidgeClassifierCV(alphas=array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 2.e+00]), cv=8)
```

```
ridPred = ridge.predict(X_test_scaled)
print('RidgeClassifer Score:',ridge.score(X_test_scaled, y_test))
ridCVPred = ridgeCV.predict(X_test_scaled)
print('RidgeClassifer CV Score:',ridgeCV.score(X_test_scaled, y_test))
```

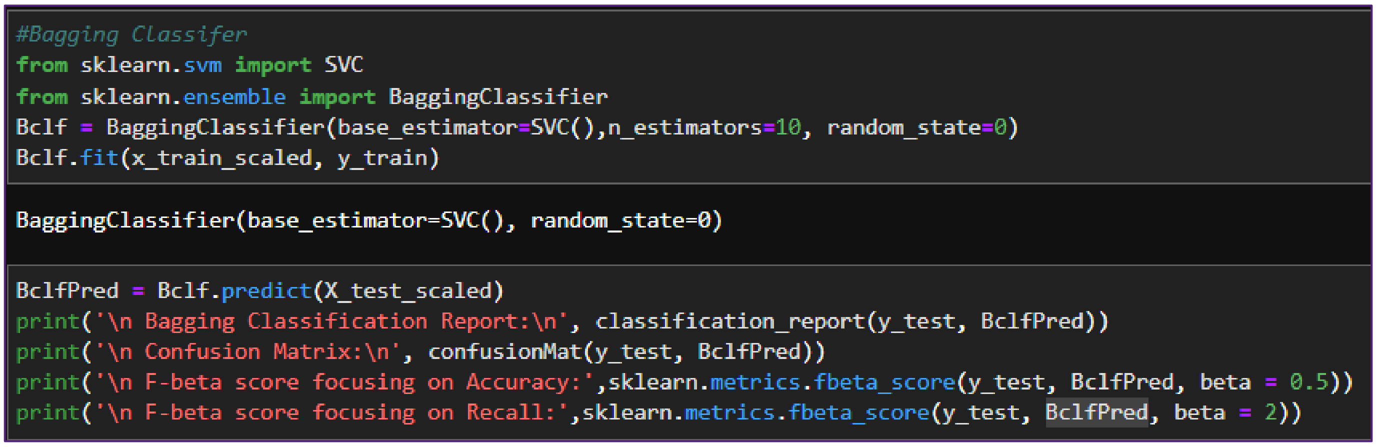
```
RidgeClassifer Score: 0.835
RidgeClassifer CV Score: 0.835
```

```
rid_pred_train = ridge.predict(x_train_scaled)
print('Train Accuracy: ', round(accuracy_score(y_train, rid_pred_train)*100, 2))
print('Test Accuracy: ', round(accuracy_score(y_test, ridPred)*100, 2))
print('Accuracy: ', round(accuracy_score(y_test, ridPred)*100, 2))
print('Cohen Kappa: '+ str(np.round(cohen_kappa_score(y_test, ridPred),3)))
print('Recall: ', round(recall_score(y_test, ridPred)*100, 2))
R2 = sklearn.metrics.r2_score(y_test, ridPred)
MSE = sklearn.metrics.mean_squared_error(y_test, ridPred)
RMS = math.sqrt(MSE)
print('\n R2 Score:',R2)
print('\n MSE Score:',MSE)
print('\n RMS Score:',RMS)
```

```
Train Accuracy:  93.17
Test Accuracy:  83.5
```

As illustrated above, the Ridge Classifier has an accuracy of 93.17% on train data and an accuracy of 83.5% on test data.

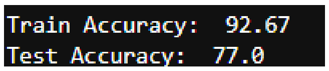
Bagging Classifier



```
#Bagging Classifer
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
Bclf = BaggingClassifier(base_estimator=SVC(),n_estimators=10, random_state=0)
Bclf.fit(x_train_scaled, y_train)
```

```
BaggingClassifier(base_estimator=SVC(), random_state=0)
```

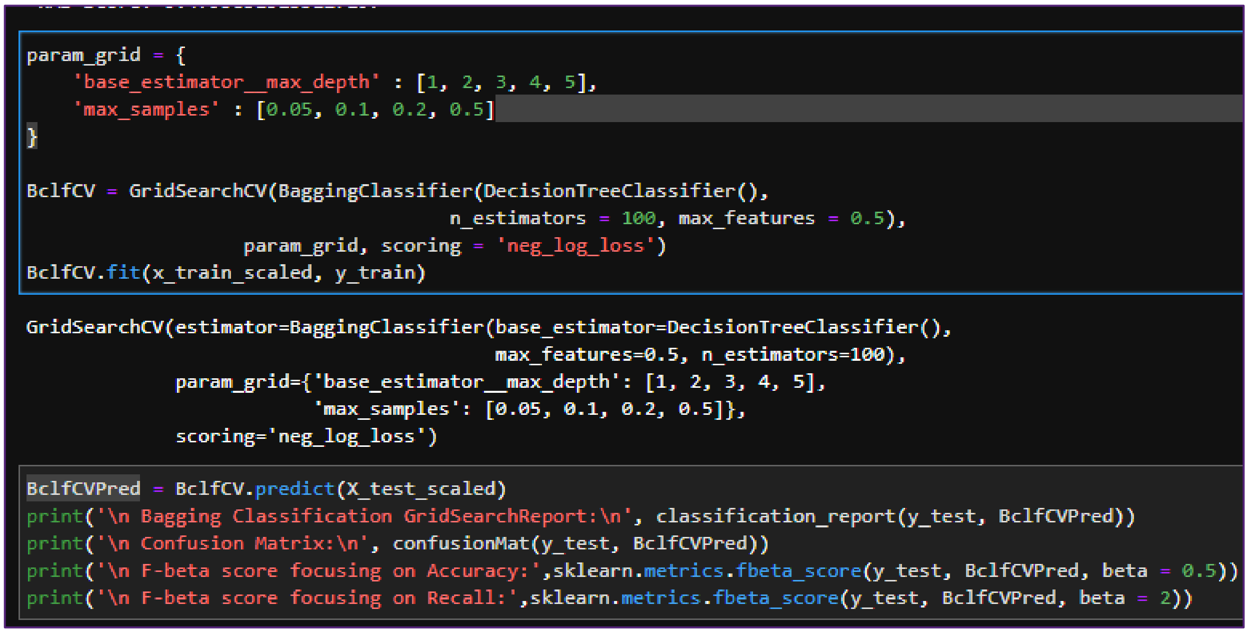
```
BclfPred = Bclf.predict(X_test_scaled)
print('\n Bagging Classification Report:\n', classification_report(y_test, BclfPred))
print('\n Confusion Matrix:\n', confusionMat(y_test, BclfPred))
print('\n F-beta score focusing on Accuracy:',sklearn.metrics.fbeta_score(y_test, BclfPred, beta = 0.5))
print('\n F-beta score focusing on Recall:',sklearn.metrics.fbeta_score(y_test, BclfPred, beta = 2))
```



```
Train Accuracy:  92.67
Test Accuracy:  77.0
```

As illustrated above, the Bagging Classifier's performance on data generates a Train accuracy of 92.67% and a Test accuracy of 77%.

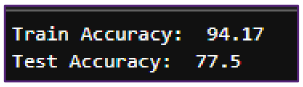
Bagging Classifier with Grid Search



```
param_grid = {
    'base_estimator__max_depth' : [1, 2, 3, 4, 5],
    'max_samples' : [0.05, 0.1, 0.2, 0.5]
}

BclfCV = GridSearchCV(BaggingClassifier(DecisionTreeClassifier(),
                                        n_estimators = 100, max_features = 0.5),
                    param_grid, scoring = 'neg_log_loss')
BclfCV.fit(x_train_scaled, y_train)
```

```
GridSearchCV(estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(),
                                         max_features=0.5, n_estimators=100),
             param_grid={'base_estimator__max_depth': [1, 2, 3, 4, 5],
                         'max_samples': [0.05, 0.1, 0.2, 0.5]},
             scoring='neg_log_loss')
```

```
BclfCVPred = BclfCV.predict(X_test_scaled)
print('\n Bagging Classification GridSearchReport:\n', classification_report(y_test, BclfCVPred))
print('\n Confusion Matrix:\n', confusionMat(y_test, BclfCVPred))
print('\n F-beta score focusing on Accuracy:',sklearn.metrics.fbeta_score(y_test, BclfCVPred, beta = 0.5))
print('\n F-beta score focusing on Recall:',sklearn.metrics.fbeta_score(y_test, BclfCVPred, beta = 2))
```



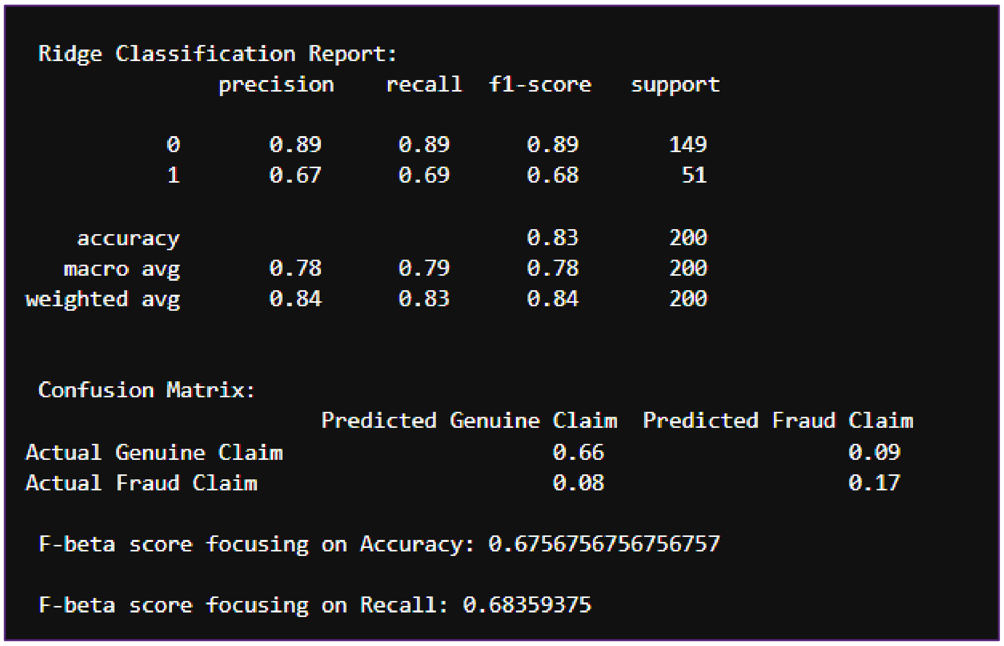
```
Train Accuracy:  94.17
Test Accuracy:  77.5
```

As illustrated above, even with hyperparameter tuning, Bagging Classifier can generate the best Train accuracy of 94.17% among all models, but only mages to achieve a Test accuracy of 77.5%. Suggesting the model is trying to overfit.

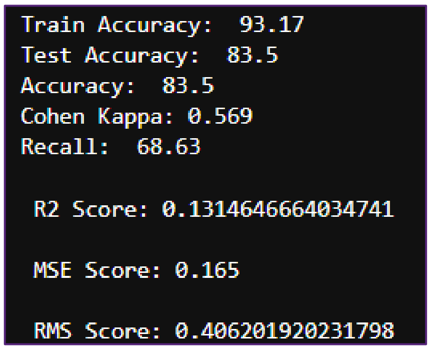
Performance of model based on the test error rate

Ridge performance summary

As illustrated below, the Ridge classifier can generate better accuracy of 83% with a total increase of 1.5%.

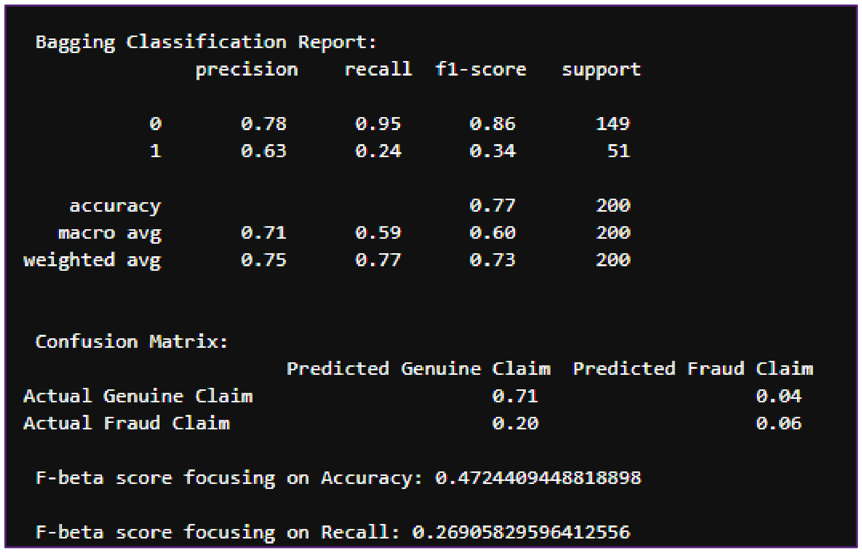


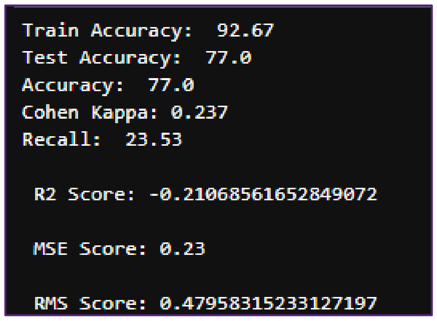
```
 Ridge Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.89      0.89       149
           1       0.67      0.69      0.68        51

    accuracy                           0.83       200
   macro avg       0.78      0.79      0.78       200
weighted avg       0.84      0.83      0.84       200


 Confusion Matrix:
                        Predicted Genuine Claim  Predicted Fraud Claim
Actual Genuine Claim                       0.66                   0.09
Actual Fraud Claim                         0.08                   0.17

 F-beta score focusing on Accuracy: 0.6756756756756757

 F-beta score focusing on Recall: 0.68359375
```

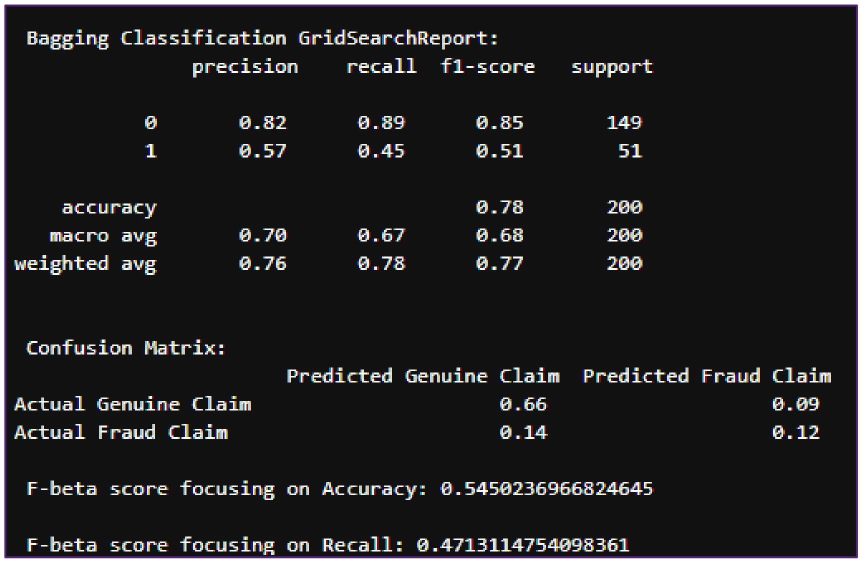


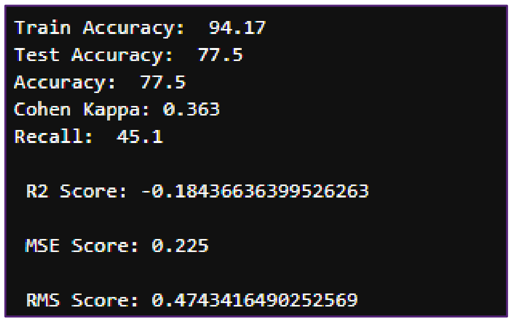
```
Train Accuracy:  93.17
Test Accuracy:  83.5
Accuracy:  83.5
Cohen Kappa: 0.569
Recall:  68.63

 R2 Score: 0.1314646664034741

 MSE Score: 0.165

 RMS Score: 0.406201920231798
```

Baggage and Baggage with Grid Search


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

           0       0.78      0.95      0.86       149
           1       0.63      0.24      0.34        51

    accuracy                           0.77       200
   macro avg       0.71      0.59      0.60       200
weighted avg       0.75      0.77      0.73       200


 Confusion Matrix:
                      Predicted Genuine Claim  Predicted Fraud Claim
Actual Genuine Claim                     0.71                   0.04
Actual Fraud Claim                       0.20                   0.06

 F-beta score focusing on Accuracy: 0.4724409448818898

 F-beta score focusing on Recall: 0.26905829596412556
```


```
Train Accuracy:  92.67
Test Accuracy:  77.0
Accuracy:  77.0
Cohen Kappa: 0.237
Recall:  23.53

 R2 Score: -0.21068561652849072

 MSE Score: 0.23

 RMS Score: 0.47958315233127197
```


```
 Bagging Classification GridSearchReport:
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       149
           1       0.57      0.45      0.51        51

    accuracy                           0.78       200
   macro avg       0.70      0.67      0.68       200
weighted avg       0.76      0.78      0.77       200


 Confusion Matrix:
                      Predicted Genuine Claim  Predicted Fraud Claim
Actual Genuine Claim                     0.66                   0.09
Actual Fraud Claim                       0.14                   0.12

 F-beta score focusing on Accuracy: 0.5450236966824645

 F-beta score focusing on Recall: 0.4713114754098361
```


```
Train Accuracy:  94.17
Test Accuracy:  77.5
Accuracy:  77.5
Cohen Kappa: 0.363
Recall:  45.1

 R2 Score: -0.18436636399526263

 MSE Score: 0.225

 RMS Score: 0.4743416490252569
```

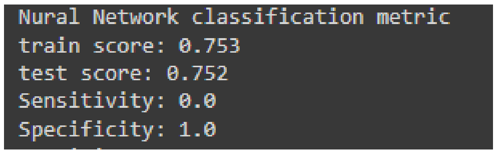
As illustrated above, Ridge Classifier has a precision of 67% and an F-1 score of 68% for fraudulent claims. Additionally, from the confusion matrix, we can observe that the model categorizes 8% of fraudulent claims as genuine claims and has an f-beta score of 68.3 % on recall. These scores show that the model can detect most fraudulent cases.

Bagging only has a precision of 63% and an F-1 score of 34% for fraudulent claims. Additionally, from the confusion matrix, we can observe that the model categorizes 20% of fraudulent claims as genuine claims and has an f-beta score of 26.9 % on recall. Bagging grid search classifiers have a precision of 57% and an F-1 score of 45% for fraudulent claims. Additionally, from the confusion matrix, we can observe that the model categorizes 14% of fraudulent claims as genuine claims and has an f-beta score of 47.1 % on recall which is pretty low.

Summary and next steps:

At the moment, Ridge Classifier is the best-performing model we have, with 83% of accuracy. But this might be the result of imbalanced data to address and check if XG-boost can perform better, we are currently working on Random Over Sample (SMOTE). Additionally, working with the feature engineering package 'Tsfresh' did not work as expected, and our manual feature engineering improved initial performance.

Additionally we have also implemented neural networks, which were only able to generate a train score of 75.3% and test score of 75.2%, which is expected as we don't have massive data. But we will check post-oversampling again to see if there are any improvements.



```
Nural Network classification metric
train score: 0.753
test score: 0.752
Sensitivity: 0.0
Specificity: 1.0
```

Code repository:
[CPSC_6300_Insurance_Project/InsuranceProject_Final.ipynb at master · DineshchandarR/CPSC_6300_Insurance_Project (github.com)]