CPSC-6300 Project:Insurance Fraud Detection

Dineshchandar Ravichandran [1*] **Archana Lalji Saroj** [1*] **Prashanth Reddy Kadire** [1*]
[1]Clemson University
{dravich,asaroj,pkadire}@g.clemson.edu

1 Project Description

Project Name	Insurance Fraud Detection
Doc. release date	Dec 4th, 2022
Release	Final
Client	Dr.Riechert Mathias and Fanny Sternfeld (Allianz)
Course Number	CPSC-6300
Course Name	Applied Data Science
Semester	Fall-2022

Author	**Ownership**
Dineshchandar Ravichandran	Requirement gathering, Background Literature review, Documentation, Lazy Classifier & Bagging Classifier.
Archana Lalji Saroj	Requirement gathering, Background Literature review, Documentation, Features engineering, SVM & XG Boost.
Prashanth Reddy Kadire	Requirement gathering, Background Literature review, Documentation, Ridge Classifier & smote.

2 Introduction

The insurance industry comprises over 7,000 companies that collect over $1 trillion in premiums yearly. The massive size of the industry contributes significantly to the cost of insurance fraud by providing more opportunities and bigger incentives for committing illegal activities. The total cost of insurance fraud (non-health insurance) is estimated to be more than $40 billion annually. That means Insurance Fraud costs the average U.S. family between $400 and $700 per year in the form of increased premiums. Insurance fraud also steals at least $308.6 billion yearly from American consumers. For this project, we will focus on Automobile-insurance fraud where 25%-33% of insurance claims have an element of fraud. Automobile claim fraud and buildup added $5.6 billion-$7.7 billion in excess payments to auto-injury claims paid in the U.S. in 2012. 21 % of bodily injury (B.I.) claims and 18 % of personal injury protection (PIP) claims closed with payment had the appearance of fraud and/or buildup. Buildup involves inflating otherwise legitimate claims. The government and other organizations are responding to this by investing in technology to detect fraudulent claims, 21% of insurance institutes plan to invest in A.I. (Artificial Intelligence) in the next two years.

2.1 Motivation Goals

We will construct a supervised machine learning model-based Fraud Detection system to predict the chances of a fraudulent insurance claim. This system aims to analyze the insurance claim data set containing 38 distinctive features and generate a classification logic to identify genuine and fraudulent claims. By constructing such a fraud detection system, we can alert the insurance institutes and allow them to take necessary action against fraudulent claims.

2.2 Data Summary

We collected the insurance claims data set from Kaggle "insurance_claims_data." This data set comprises 1000 total data points. The data set is imbalanced since there are a total of 247 fraud claims and 753 genuine claims, according to the preliminary data exploration.

3 Exploratory Data Analysis

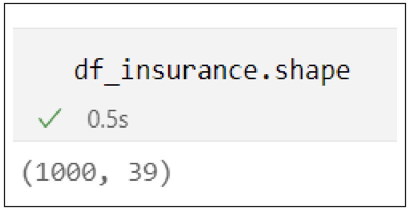
3.1 Unit of Analysis

Column Name	Data Type	Description
months_as_customer	int64	No.of months customer has been a customer to the insurance organization.
age	int64	Age of the customer.
policy_number	int64	Policy no.of the customer's insurance.
policy_bind_date	object	The moment when the insurance coverage goes into force, it's date and time specific.
policy_state	object	State in which the policy was procured.
policy_csl	object	Combined single limits are a provision of an insurance policy limiting coverage for all components of a claim to a single dollar amount. A combined single limit policy has a maximum dollar amount that covers any combination of injuries or property damage in an incident.
policy_deductable	int64	The amount customers have to pay for covered services before their insurance plan starts to pay.
policy_annual_premium	float64	Annual payment for the policy.
umbrella_limit	int64	Extra insurance that provides protection beyond existing limits and coverages of other policies. Umbrella insurance can cover injuries, property damage, certain lawsuits, and personal liability situations.
insured_zip	int64	N.A
insured_sex	object	Male/ Female.
insured_education_level	object	Education level of the customer.
insured_occupation	object	Occupation of the customer.
insured_hobbies	object	Customers' hobbies.
insured_relationship	object	Relationship of the person involved in the incedent to the actual insurance holder.
capital-gains	int64	Increase in a capital asset's value.
capital-loss	int64	Loss in a capital asset's value.
incident_date	object	Date of incident.
incident_type	object	Type of incident (Single Vehicle Collision, Vehicle Theft, Multi-vehicle Collision, and Parked Car).
collision_type	object	Type of collision (Side Collision, Rear Collision, and Front Collision).
incident_severity	object	The severity of the incident classified as per damage (Major Damage, Minor Damage, Total Loss, and Trivial Damage).

authorities_contacted	object	Type of authorities contacted after the incident.
incident_state	object	U.S. state where the incident occurred.
incident_city	object	City in which the incident occurred.
incident_location	object	Address of the incident.
incident_hour_of_the_day	int64	Time of incendent in 24hrs.
number_of_vehicles_involved	int64	No.of vehicles involved with the incident.
property_damage	object	If 'YES,' then the insurance carrier helps pay to repair the damage customer caused to other involved parties.
bodily_injuries	int64	No.of bodily injuries.
witnesses	int64	No.of witnesses to the incident.
police_report_available	object	If a police report exists of the incident.
total_claim_amount	int64	The total amount claimed by the customer fot the incedent.
injury_claim	int64	The portion of the total claim requested for the injury claim.
property_claim	int64	The portion of the total claim requested to pay for property damages.
vehicle_claim	int64	The portion of the total claim requested for vehicle damage.
auto_make	object	Manufacturer of the customer's vehicle that was involved in the incident.
auto_model	object	The specific automobile model of the customer's that was involved in the incident.
auto_year	int64	Model year
fraud_reported	object	Determining whether a claim is fraudulent or not.

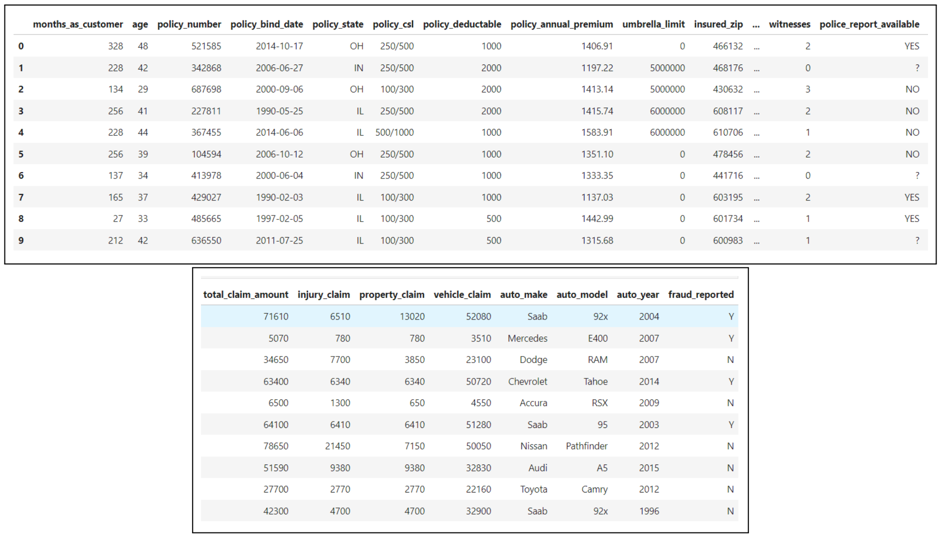
3.1.1 Observations in data set

We have a data set containing 1000 insurance claims and 38 features pertaining to it. In addition, we also have a column stating which of these insurance claims are fraudulent and which are genuine, as illustrated below:



```
df_insurance.shape
✓ 0.5s
(1000, 39)
```

Figure 1: Sample size



	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	...	witnesses	police_report_available
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0	466132	...	2	YES
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000	468176	...	0	?
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	5000000	430632	...	3	NO
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	6000000	608117	...	2	NO
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	6000000	610706	...	1	NO
5	256	39	104594	2006-10-12	OH	250/500	1000	1351.10	0	478456	...	2	NO
6	137	34	413978	2000-06-04	IN	250/500	1000	1333.35	0	441716	...	0	?
7	165	37	429027	1990-02-03	IL	100/300	1000	1137.03	0	603195	...	2	YES
8	27	33	485665	1997-02-05	IL	100/300	500	1442.99	0	601734	...	1	YES
9	212	42	636550	2011-07-25	IL	100/300	500	1315.68	0	600983	...	1	?

total_claim_amount	injury_claim	property_claim	vehicle_claim	auto_make	auto_model	auto_year	fraud_reported
71610	6510	13020	52080	Saab	92x	2004	Y
5070	780	780	3510	Mercedes	E400	2007	Y
34650	7700	3850	23100	Dodge	RAM	2007	N
63400	6340	6340	50720	Chevrolet	Tahoe	2014	Y
6500	1300	650	4550	Accura	RSX	2009	N
64100	6410	6410	51280	Saab	95	2003	Y
78650	21450	7150	50050	Nissan	Pathfinder	2012	N
51590	9380	9380	32830	Audi	A5	2015	N
27700	2770	2770	22160	Toyota	Camry	2012	N
42300	4700	4700	32900	Saab	92x	1996	N

Figure 2: Sample head

3.1.2 Unique observations

In our dataset, every claim is unique; though we do not have a unique identifier like claim I.D., the policy numbers for all the claims are unique.

3.1.3 Time period covered

Our dataset contains claims with incidents occurring from 1st Jan 2015 to 1st March 2015.

3.2 Data cleaning

Three columns had '?' Missing value:

1. Collision Type.
2. Property Damage.
3. Police Report Available.

The missing values are imputed using KNN imputer. The categorical features have been transformed into numerical features using one-hot encoding and label encoding.

3.2.1 Feature engineering

Feature engineering is a machine learning technique that uses data to generate new variables that were not present in the training set. It has the potential to generate new features for both supervised and unsupervised learning, with the goal of simplifying and speeding up data transformations while also improving model accuracy. When working with machine learning models, feature engineering is required. A bad feature will have a direct impact on the model, regardless of the data or architecture. In our case, Six interaction terms were created. Interaction between property claim amount and incident severity, vehicle claim amount and incident severity, injury claim amount and incident severity, total claim amount and incident severity, policy annual premium and total claim amount, umbrella limit and total claim amount. Nominal variables were one-hot encoded.

Additionally, we also performed data scaling to standardize the independent features present in the data set via the Standard-Scalar method. To get the unit variance, divide all the data by the standard deviation. A distribution produced by Standard-Scaler has a standard deviation of one. Because variance equals standard deviation squared, the variance is equal to 1.

3.3 Visualization of the response

Post-data cleaning, we can see the changes in the data as illustrated below:

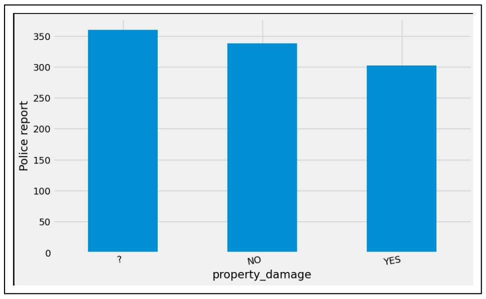


Figure 3: Data of Collision type vs police report prior data cleaning

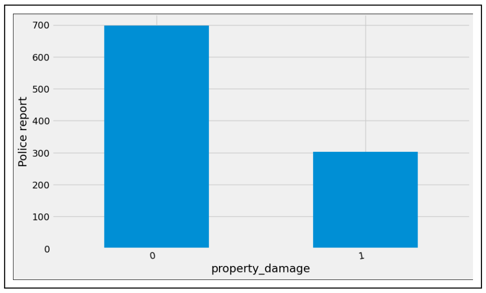
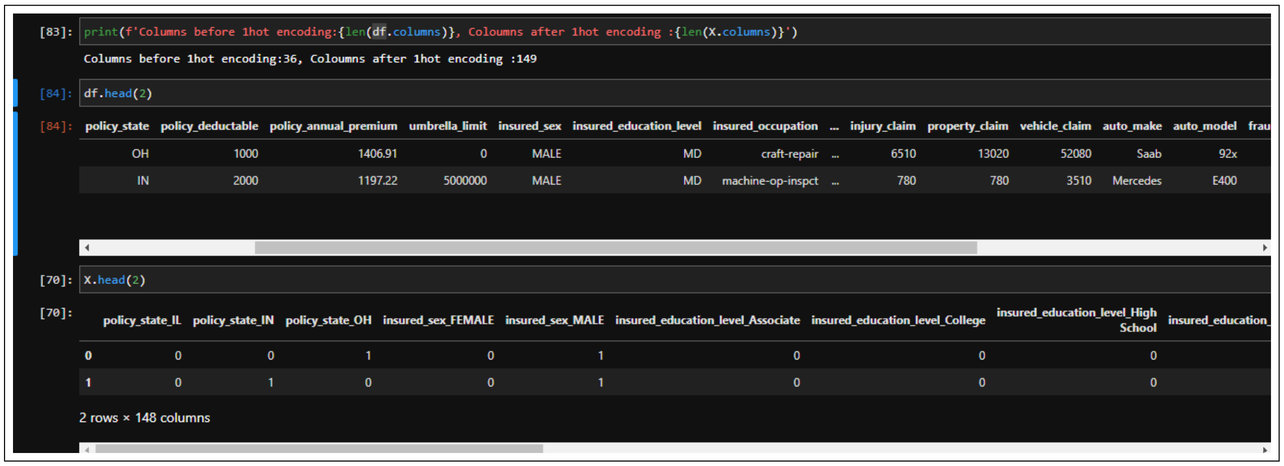


Figure 4: Data of Collision type vs police report post data cleaning

Effects of one hot-encoding:



```
[83]: print(f'Columns before 1hot encoding:{len(df.columns)}, Coloumns after 1hot encoding :{len(X.columns)}')
      Columns before 1hot encoding:36, Coloumns after 1hot encoding :149
```

```
[84]: df.head(2)
```

[84]:

policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	insured_occupation	...	injury_claim	property_claim	vehicle_claim	auto_make	auto_model	frau
OH	1000	1406.91	0	MALE	MD	craft-repair	...	6510	13020	52080	Saab	92x	
IN	2000	1197.22	5000000	MALE	MD	machine-op-inspct	...	780	780	3510	Mercedes	E400	

```
[70]: X.head(2)
```

[70]:

	policy_state_IL	policy_state_IN	policy_state_OH	insured_sex_FEMALE	insured_sex_MALE	insured_education_level_Associate	insured_education_level_College	insured_education_level_High School	insured_education_
0	0	0	1	0	1	0	0	0	
1	0	1	0	0	1	0	0	0	

2 rows × 148 columns

Figure 5: Column and data comparison before and after one-hot encoding

3.4 Visualization of key predictors

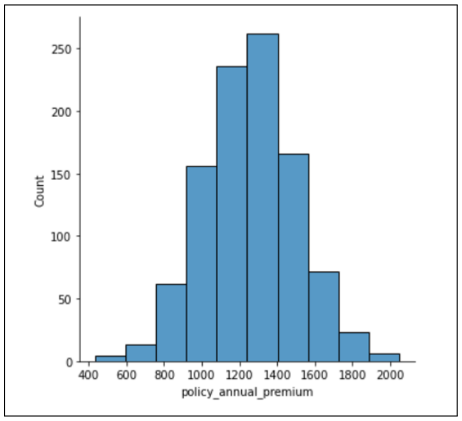


Figure 6: Annual policy premium distribution

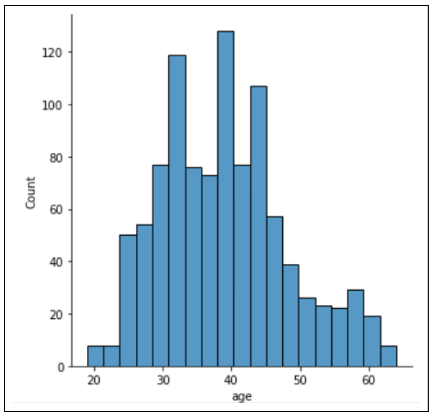


Figure 7: Age distribution of policyholders

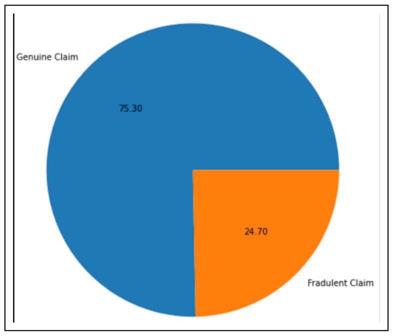


Figure 8: Genuine claims VS Fraudulent claims comparisons

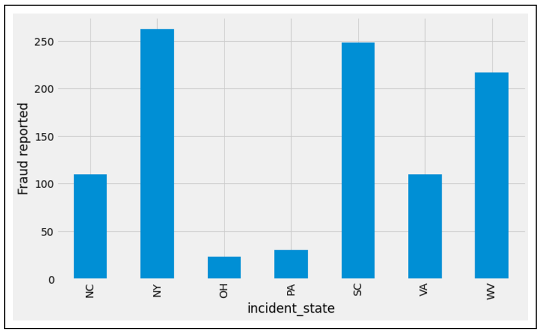


Figure 9: Incident state-wise fraudulent claims

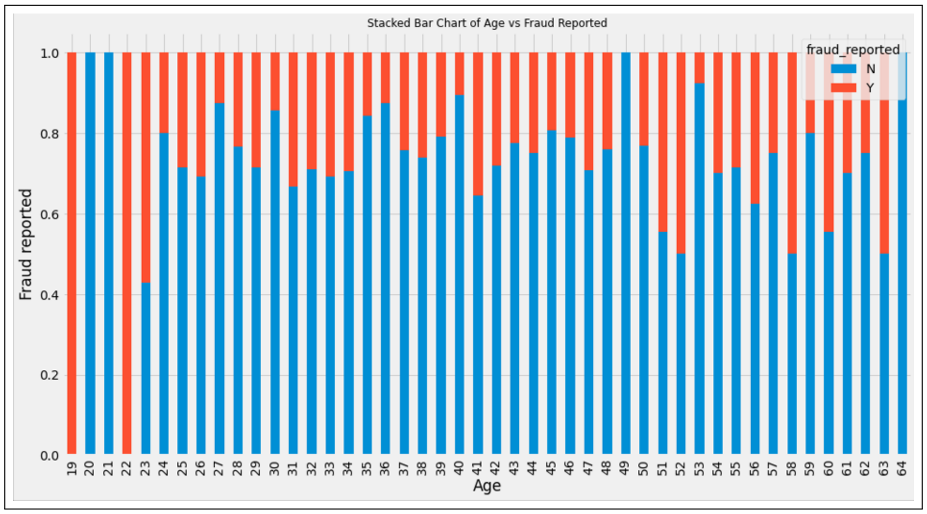


Figure 10: Age-wise fraudulent claims

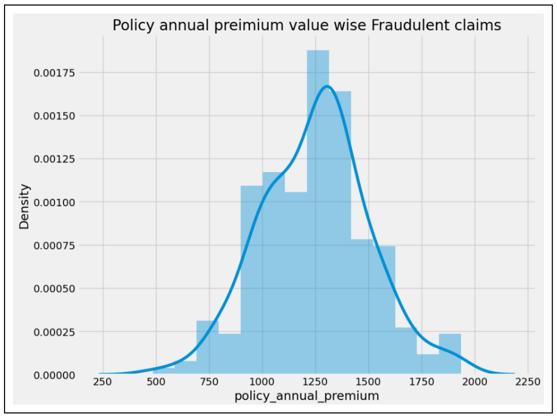


Figure 11: Policy premium relation with fraudulent claims

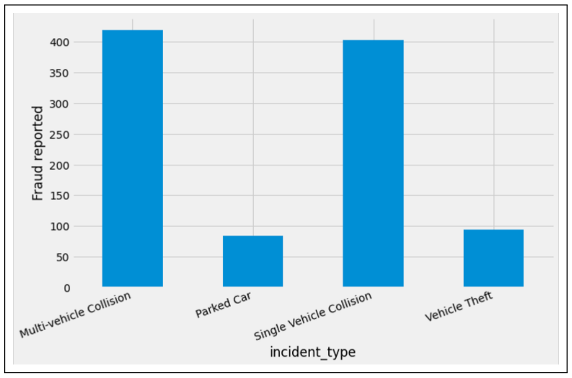


Figure 12: Incident type related to fraudulent claims

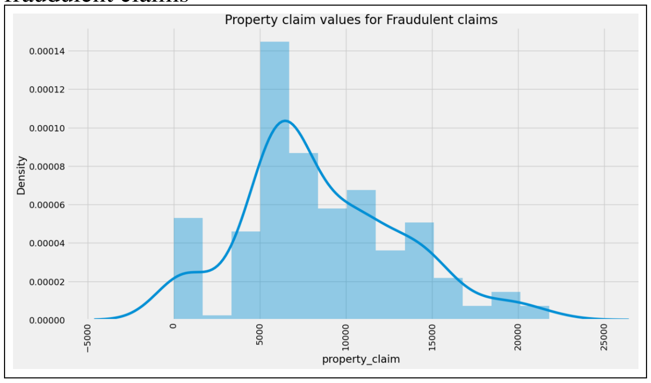


Figure 13: Property claim relation with fraudulent claims

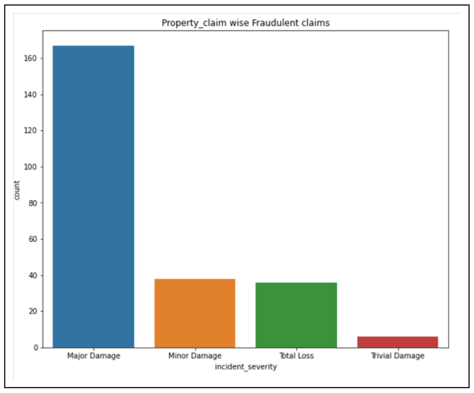


Figure 14: Damage severity related to fraudulent claims

4 Machine Learning Models

4.1 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, since we have labels stating which claims are fraudulent. As stated in the above sections, 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.

But as we progressed through our implementation we realised Ridge Classifier and Bagging classifier out perform logistic regression and SVM.

4.2 XG-Boost

XG-boost is widely used for statistical analysis and prediction, often time outperforming deep learning models (5). Furthermore, we referred to similar works on fraud detection, like credit-card fraud detection (4) 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.

4.3 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. 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.

4.4 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.

4.5 Ridge Classification:

Ridge classification is a technique used to analyze linear discriminant models. It is a form of regularization that penalizes model coefficients to prevent overfitting. 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 a 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 Ridge classifier is not cross-entropy loss as like Logistic Regression. Rather the loss function is mean square loss with L2 penalty. It works in the following manner for the binary classification problems by making use of Ridge regression algorithm:

1. Converts the target variable into +1 and -1 appropriately.

2. Train a Ridge model with loss function as mean square loss with L2 regularization (ridge) as penalty term.
3. During prediction, if the predicted value is less than 0, it predicted class label is -1 otherwise the predicted class label is +1.

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. Ridge classifier is trained in a one-versus-all approach for multi-class classification. LabelBinarizer is used to achieve this objective by learning one binary classifier per class.

The cost function for ridge:

$$Min(||Y - (X(theta))||^2 + \lambda||theta||^2)$$

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.

4.6 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. This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting (2). If samples are drawn with replacement, then the method is known as Bagging (1).

4.7 Performance evaluators

For this project, the accuracy, Precision, F1, Confusion Matrix, and Recall Score will be calculated on each model. These measures are considered as our unit of analysis for selecting the best model.

- **F1 score**: This is the harmonic mean of precision and recall and gives a better measure of the incorrectly classified cases than the accuracy matrix.

$$F1 - score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

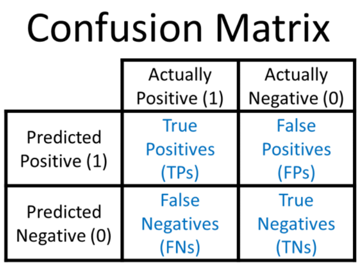
- **Precision**: It is implied as the measure of the correctly identified positive cases from all the predicted positive cases. Thus, it is useful when the costs of False Positives are high.

$$Precision = \frac{TruePositive}{(TruePositive + FalsePositive)}$$

- **Accuracy**: One of the more obvious metrics is the measure of all the correctly identified cases. It is most used when all the classes are equally important.

$$Accuracy = \frac{TruePositive + TrueNegative}{(TruePositive + FalsePositive + TrueNegative + FalseNegative)}$$

- **Confusion Matrix** : Confusion Matrix s a performance measurement for classification problems. It defines the results in the following four groups:
  True Positive: Model predicted positive value, and it's true.
  True Negative: Model predicted negative value, and it's true.
  False Positive: (Type 1 Error) Model predicted positive, but it's negative.
  False Negative: (Type 2 Error) Model predicted negative, but it's positive.



Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Figure 15: Confusion Matrix

- **Recall**: Recall is calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples. The recall measures the model's ability to detect Positive samples. The higher the recall, the more positive samples detected.

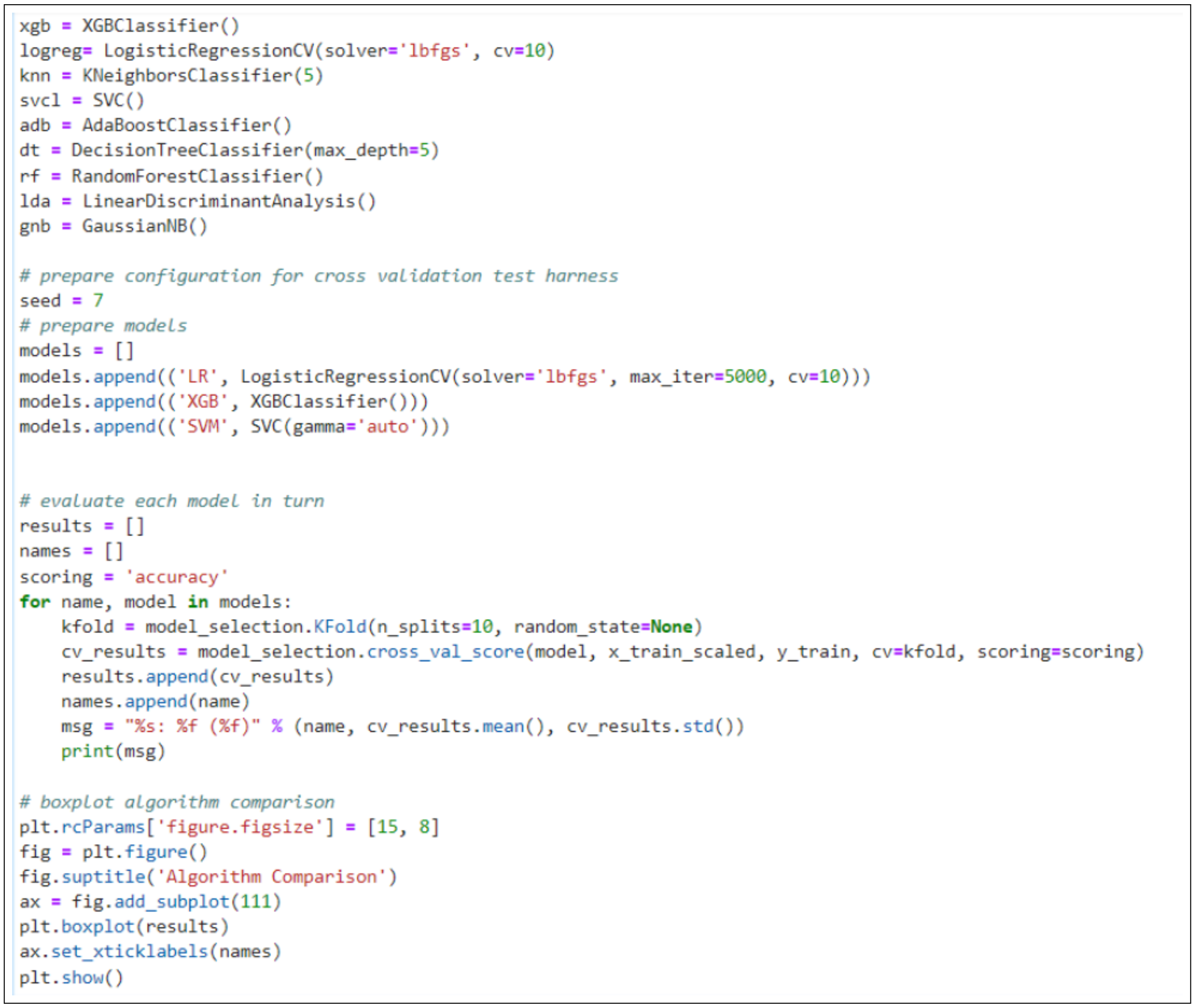
$$Recall = \frac{TruePositive}{TruePositive + FalsePositive}$$

4.8 Results of Models & Predictions

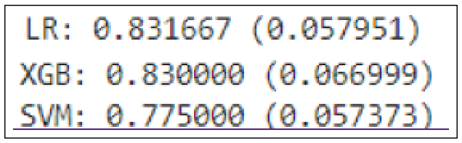
4.8.1 XG-Boost

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

In above figure we are able to 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. Based on this following best parameters 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%. The performance of tuned XG-boost on test data, where it is generating an accuracy of 81.5% and recall of 76.47%:



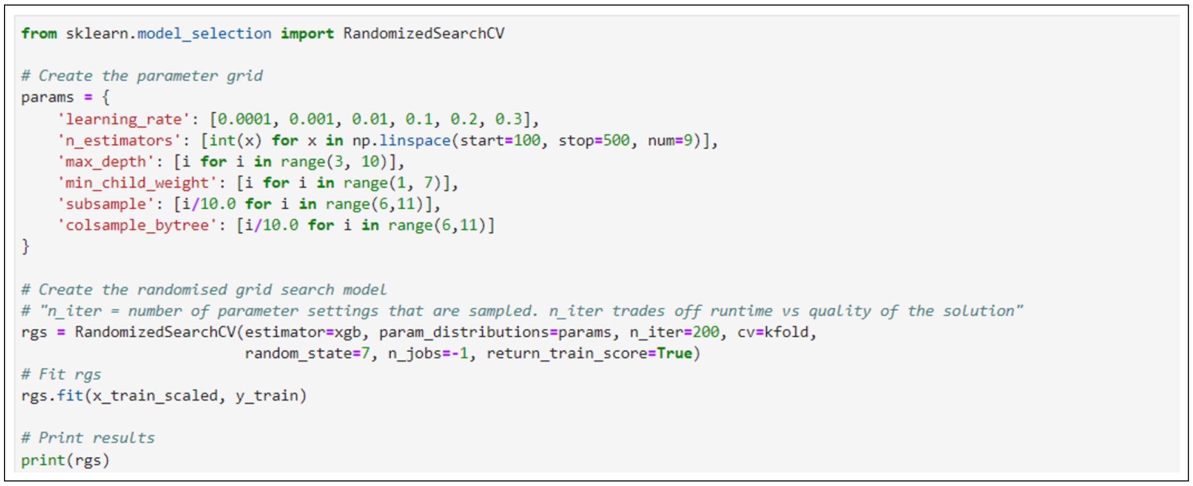
```python
xgb = XGBClassifier()
logreg= LogisticRegressionCV(solver='lbfgs', cv=10)
knn = KNeighborsClassifier(5)
svcl = SVC()
adb = AdaBoostClassifier()
dt = DecisionTreeClassifier(max_depth=5)
rf = RandomForestClassifier()
lda = LinearDiscriminantAnalysis()
gnb = GaussianNB()

# 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()
```



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

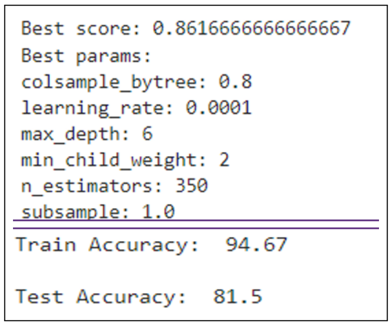
Figure 16: Chosen model's score

The model also achived an AUC score of 79.85% with the following ROC:



```
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)
```

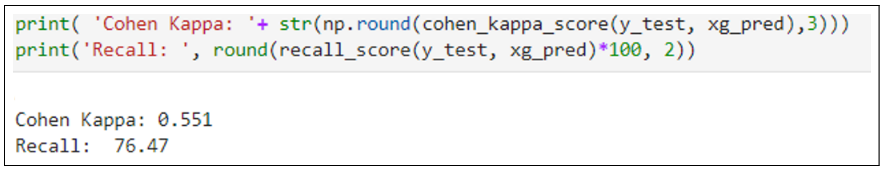
Figure 17: XG-Boost Random Grid Hyperparameter



```
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
```

Figure 18: Tuning Scores

Performance of the XG-Boost based on: F1-score, recall, precision, F-beta, and confusion matrix: 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.



```
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))

Cohen Kappa: 0.551
Recall:  76.47
```

Figure 19: XG boost recall scores

```
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
```

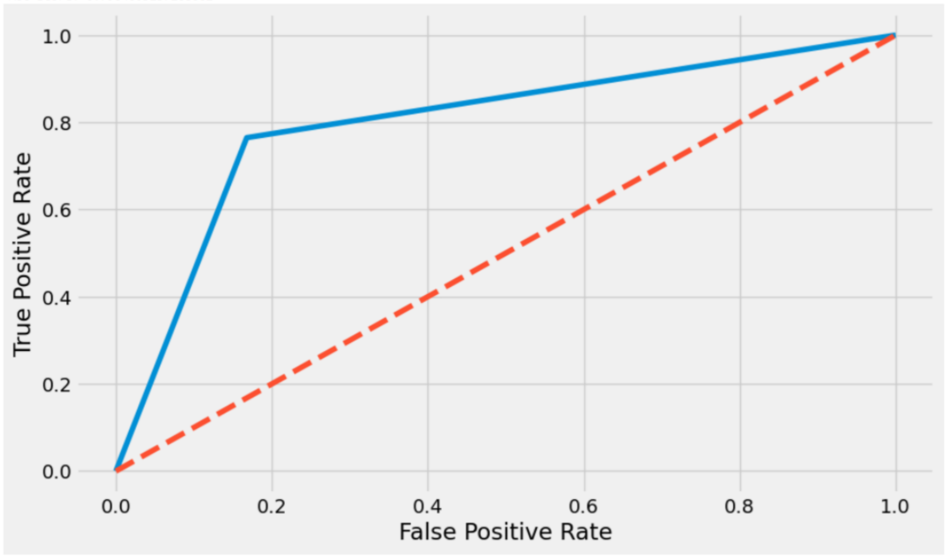
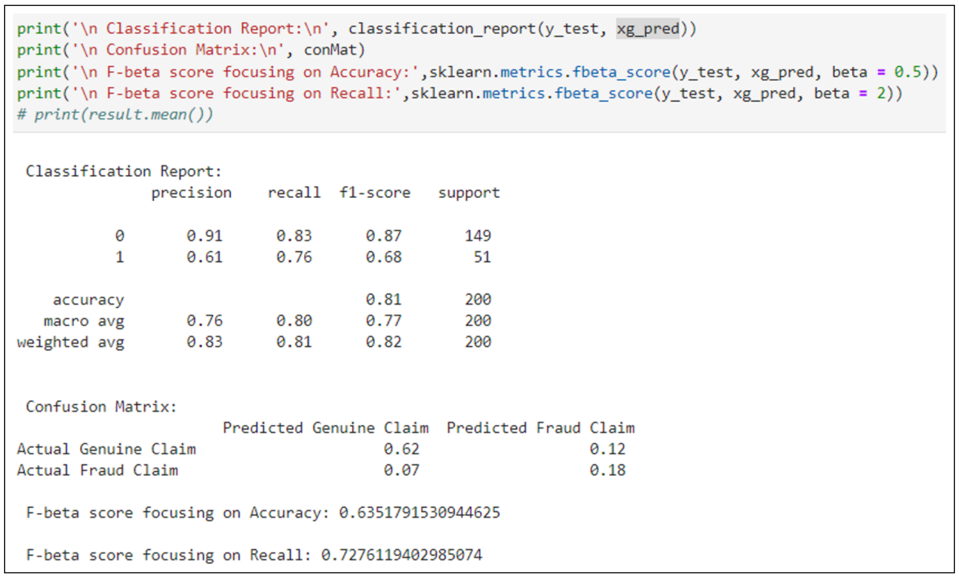


Figure 20: XG boost AUC scores

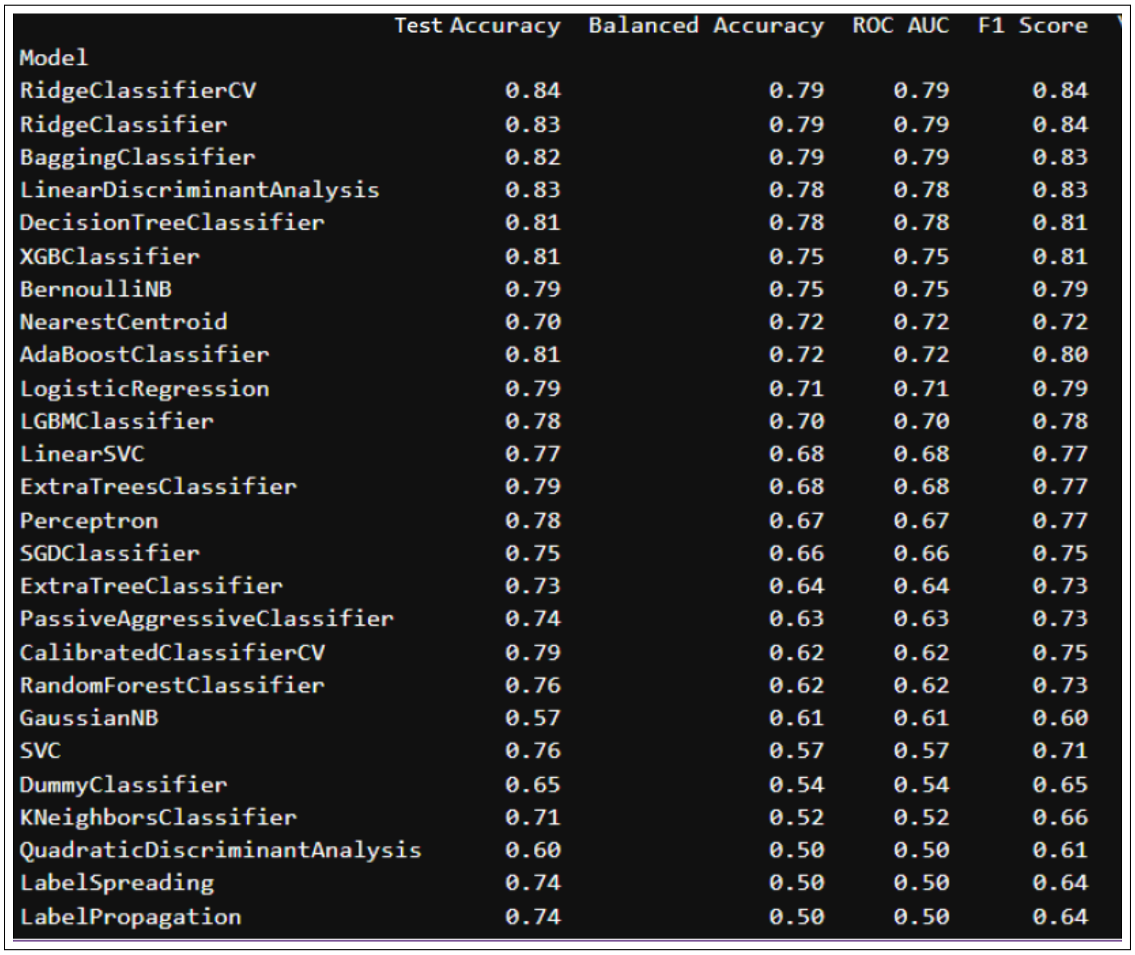


```
print('\n Classification Report:\n', classification_report(y_test, xg_pred))
print('\n Confusion Matrix:\n', conMat)
print('\n F-beta score focusing on Accuracy:',sklearn.metrics.fbeta_score(y_test, xg_pred, beta = 0.5))
print('\n F-beta score focusing on Recall:',sklearn.metrics.fbeta_score(y_test,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

Figure 21: XG boost F1, Recall, F-beta scores and Confusion Matrix

4.8.2 Lazy Classifier

To further evaluate the performances of other models we implemented Lazy Predict Classifiers, following was the performance of 26 models are:



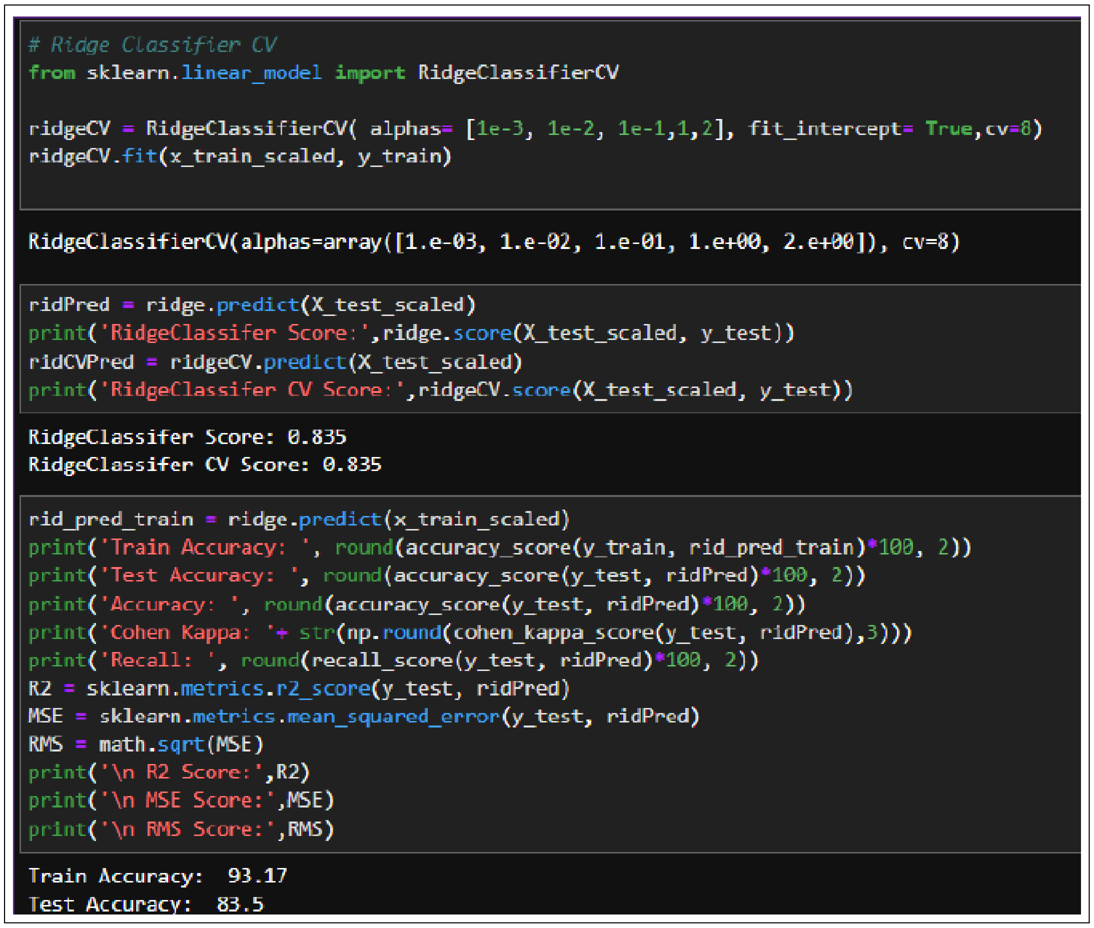
```
                               Test Accuracy  Balanced Accuracy  ROC AUC  F1 Score
Model
RidgeClassifierCV                       0.84               0.79     0.79      0.84
RidgeClassifier                         0.83               0.79     0.79      0.84
BaggingClassifier                       0.82               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
```

Figure 22: Lazy Classifier with scaled data

Based on the above results we expanded our initial model evaluation to Ridge Classifier and Bagging Classifier.

4.8.3 Ridge Classifier

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



```
# 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))
```

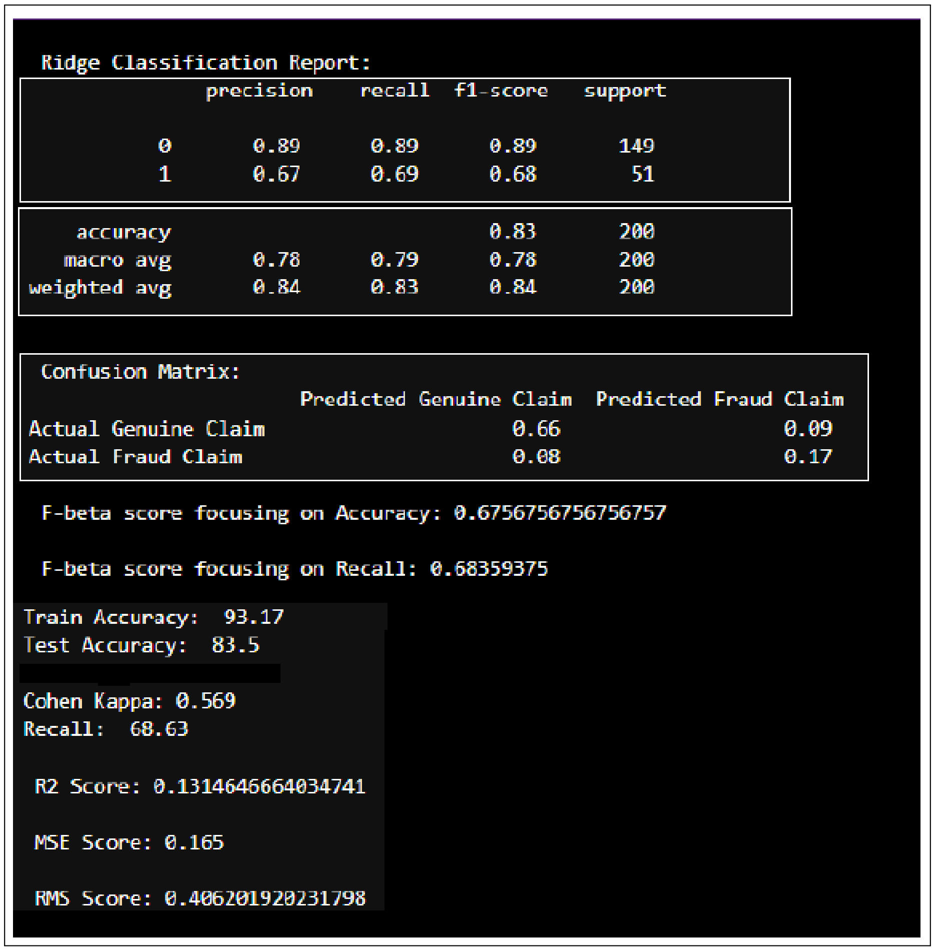
```
RidgeClassifer Score: 0.835
RidgeClassifer CV Score: 0.835
```

```
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
```

Figure 23: Ridge Classifier and its scores

For Ridge Classifier and Ridge ClassifierCV we have used sklearn.preprocessing.StandardScaler() to standardize inputs. Calling the fit function calculates the mean and standard deviation of the training set. Then, the same fitted object is used to transform the train, validation, and test sets. Scaling has an impact on the magnitude of coefficients, but it does not have any significant impact on our model predictions.

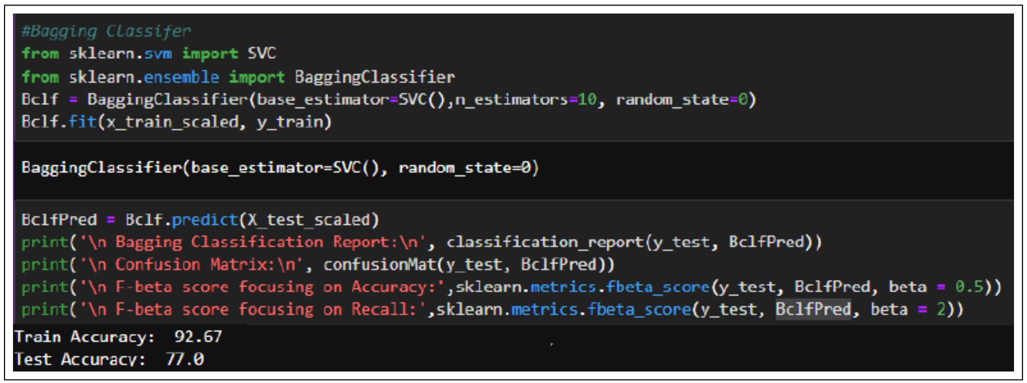


```
 Ridge Classification Report:
                precision    recall  f1-score   support

           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

Cohen Kappa: 0.569
Recall:  68.63

 R2 Score: 0.1314646664034741

 MSE Score: 0.165

 RMS Score: 0.406201920231798
```

Figure 24: Ridge Classifier performance summary

4.8.4 Bagging Classifier

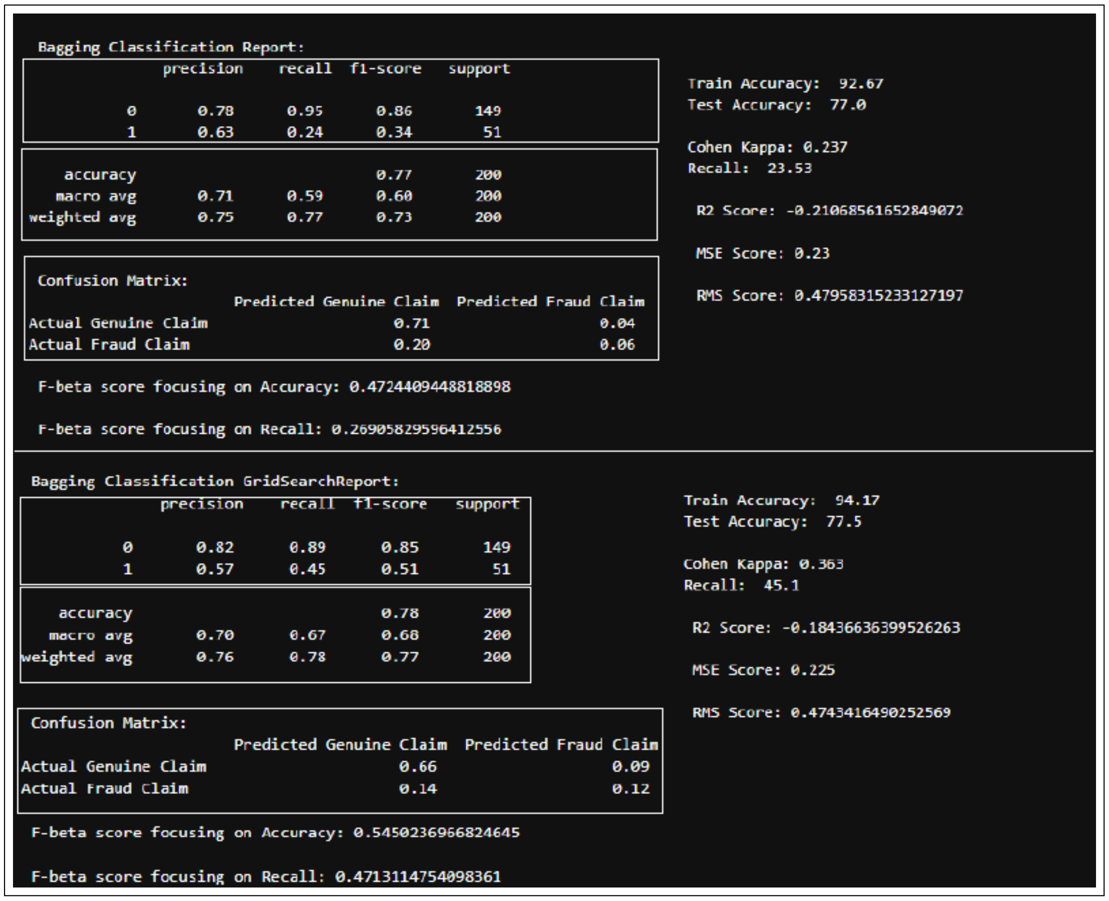
As illustrated below, the Bagging Classifier's performance on data generates a Train accuracy of 92.67% and a Test accuracy of 77%. To improve the scores we implemented grid search still the



```
#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
```

Figure 25: Bagging Classifier scores

bagging classifier did not improve its performance, as compared below: Here we don't see any significant performance improvement of Bagging Classifier whether it trains on scaled or unscaled data. As the Bagging Classifier doesn't require feature scaling because, Bagging is the application



```
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

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
```

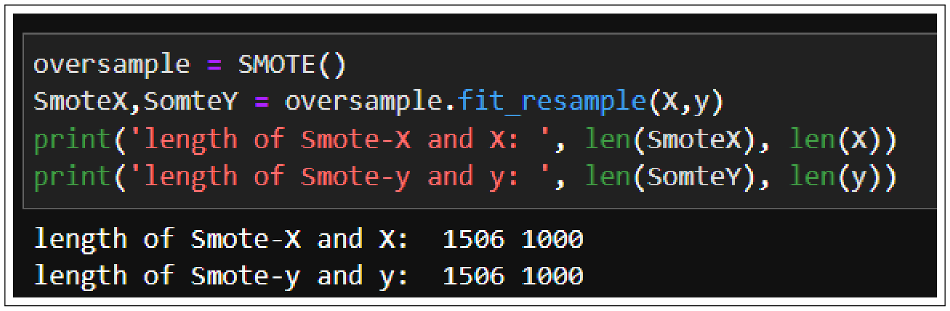
```
Train Accuracy:  94.17
Test Accuracy:  77.5

Cohen Kappa: 0.363
Recall:  45.1

R2 Score: -0.18436636399526263

MSE Score: 0.225

RMS Score: 0.4743416490252569
```

Figure 26: Bagging Classifier performance summary

of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees. There's a reason that tree-based models don't require scaling - they are invariant (that means they don't change if such a thing occurs) to monotonic transformations of any feature/input/independent variable.

4.8.5 Smote Oversampling

The performance of above models are effected by the huge imbalance of data between Genuine and Fraud insurance claims, where only 24.70% belong to fraud cases. To address this imbalanced datasets we will oversample the minority class using Synthetic Minority Oversampling Technique (SMOTE). SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Post implementing SMOTE our base dataset size increases from 1000 data-points to 1506 data-points as illustrated below:

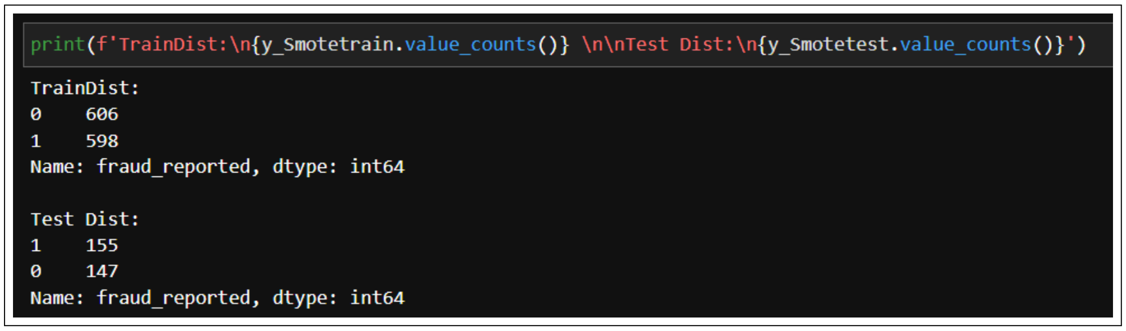


```
oversample = SMOTE()
SmoteX,SomteY = oversample.fit_resample(X,y)
print('length of Smote-X and X: ', len(SmoteX), len(X))
print('length of Smote-y and y: ', len(SomteY), len(y))
```

```
length of Smote-X and X:  1506 1000
length of Smote-y and y:  1506 1000
```

Figure 27: Smote dataset and original dataset data-size compression

Post Smote the minority class of fraudulent cases have increased from 24.70% to 49.66%, as illustrated below:



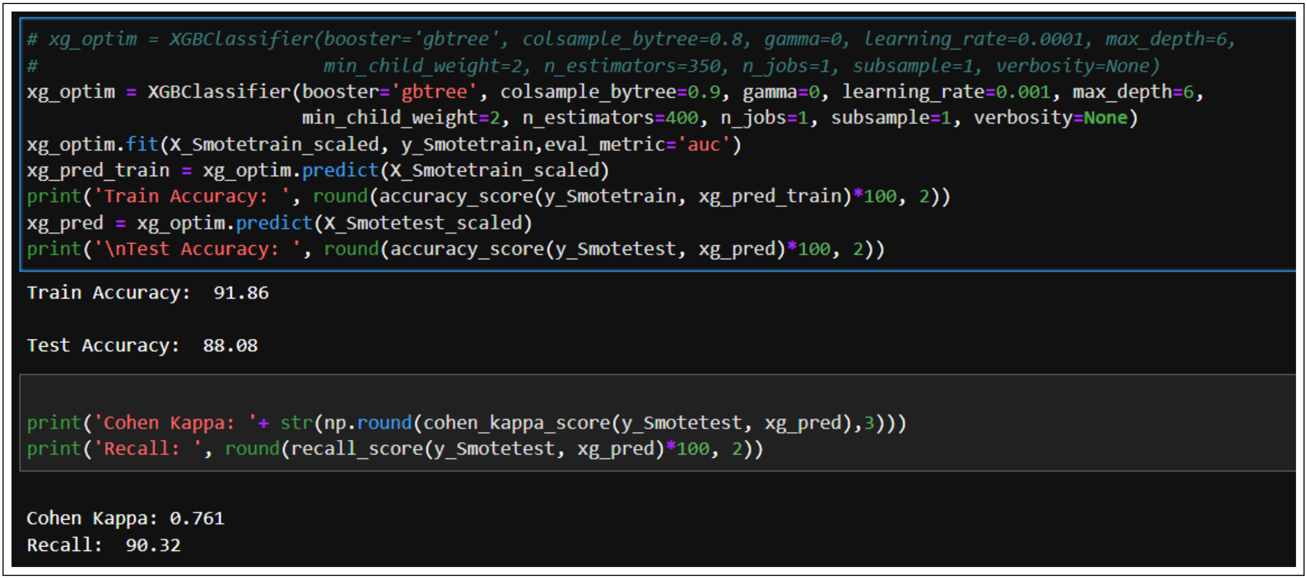
```
print(f'TrainDist:\n{y_Smotetrain.value_counts()} \n\nTest Dist:\n{y_Smotetest.value_counts()}')
```

```
TrainDist:
0    606
1    598
Name: fraud_reported, dtype: int64

Test Dist:
1    155
0    147
Name: fraud_reported, dtype: int64
```

Figure 28: Smote dataset test and train split

Post Smote oversampling, the performance of the model improved significantly. Because by generating similar examples to the existing minority points, SMOTE creates more significant and less specific decision boundaries that increase the generalization capabilities of classifiers. As illustrated below:

1. XG-Boost performance with Smote Data:



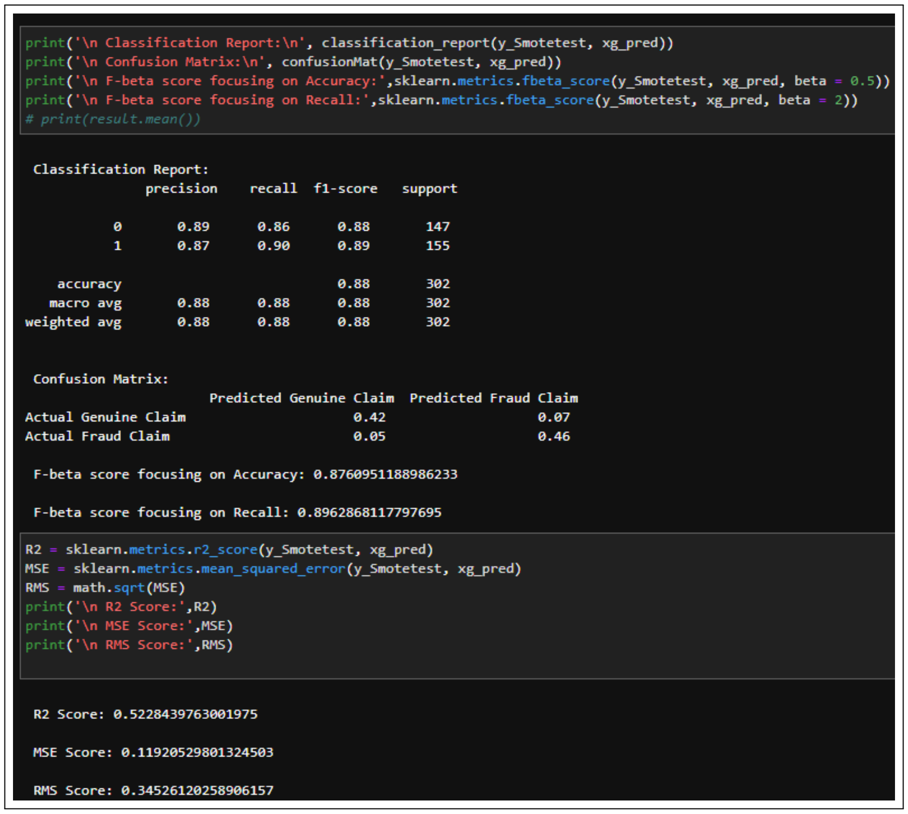
```
# xg_optim = XGBClassifier(booster='gbtree', colsample_bytree=0.8, gamma=0, learning_rate=0.0001, max_depth=6,
#                          min_child_weight=2, n_estimators=350, n_jobs=1, subsample=1, verbosity=None)
xg_optim = XGBClassifier(booster='gbtree', colsample_bytree=0.9, gamma=0, learning_rate=0.001, max_depth=6,
                         min_child_weight=2, n_estimators=400, n_jobs=1, subsample=1, verbosity=None)
xg_optim.fit(X_Smotetrain_scaled, y_Smotetrain,eval_metric='auc')
xg_pred_train = xg_optim.predict(X_Smotetrain_scaled)
print('Train Accuracy: ', round(accuracy_score(y_Smotetrain, xg_pred_train)*100, 2))
xg_pred = xg_optim.predict(X_Smotetest_scaled)
print('\nTest Accuracy: ', round(accuracy_score(y_Smotetest, xg_pred)*100, 2))
```

```
Train Accuracy:  91.86

Test Accuracy:  88.08
```

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

```
Cohen Kappa: 0.761
Recall:  90.32
```

Figure 29: XGBoost with Smote dataset Test Acc, Train Acc and Recall.



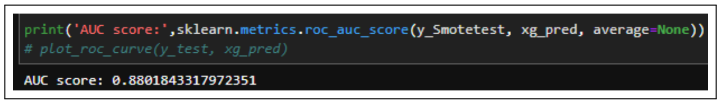
```
print('\n Classification Report:\n', classification_report(y_Smotetest, xg_pred))
print('\n Confusion Matrix:\n', confusionMat(y_Smotetest, xg_pred))
print('\n F-beta score focusing on Accuracy:',sklearn.metrics.fbeta_score(y_Smotetest, xg_pred, beta = 0.5))
print('\n F-beta score focusing on Recall:',sklearn.metrics.fbeta_score(y_Smotetest,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

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

           0       0.89      0.86      0.88       147
           1       0.87      0.90      0.89       155

    accuracy                           0.88       302
   macro avg       0.88      0.88      0.88       302
weighted avg       0.88      0.88      0.88       302


 Confusion Matrix:
                        Predicted Genuine Claim  Predicted Fraud Claim
Actual Genuine Claim                       0.42                   0.07
Actual Fraud Claim                         0.05                   0.46

 F-beta score focusing on Accuracy: 0.8760951188986233

 F-beta score focusing on Recall: 0.8962868117797695
```

```
R2 = sklearn.metrics.r2_score(y_Smotetest, xg_pred)
MSE = sklearn.metrics.mean_squared_error(y_Smotetest, xg_pred)
RMS = math.sqrt(MSE)
print('\n R2 Score:',R2)
print('\n MSE Score:',MSE)
print('\n RMS Score:',RMS)
```

```
 R2 Score: 0.5228439763001975

 MSE Score: 0.11920529801324503

 RMS Score: 0.34526120258906157
```

Figure 30: XGBoost with Smote dataset performance summary.

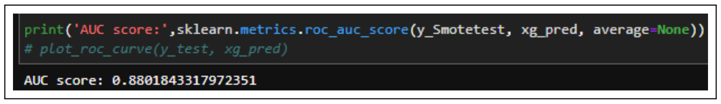


```
print('AUC score:',sklearn.metrics.roc_auc_score(y_Smotetest, xg_pred, average=None))
# plot_roc_curve(y_test, xg_pred)
```

```
AUC score: 0.8801843317972351
```

Figure 31: XGBoost with Smote dataset AUC score.

As illustrated above, the XG-boost model with smote oversampling is able to improve accuracy by 6.58% from 81.5% to 88.08%. In addition, it also has a better recall of 90.32% and a fraudulent F1 score of 89%.



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

```
AUC score: 0.8801843317972351
```

Figure 32: XGBoost feature

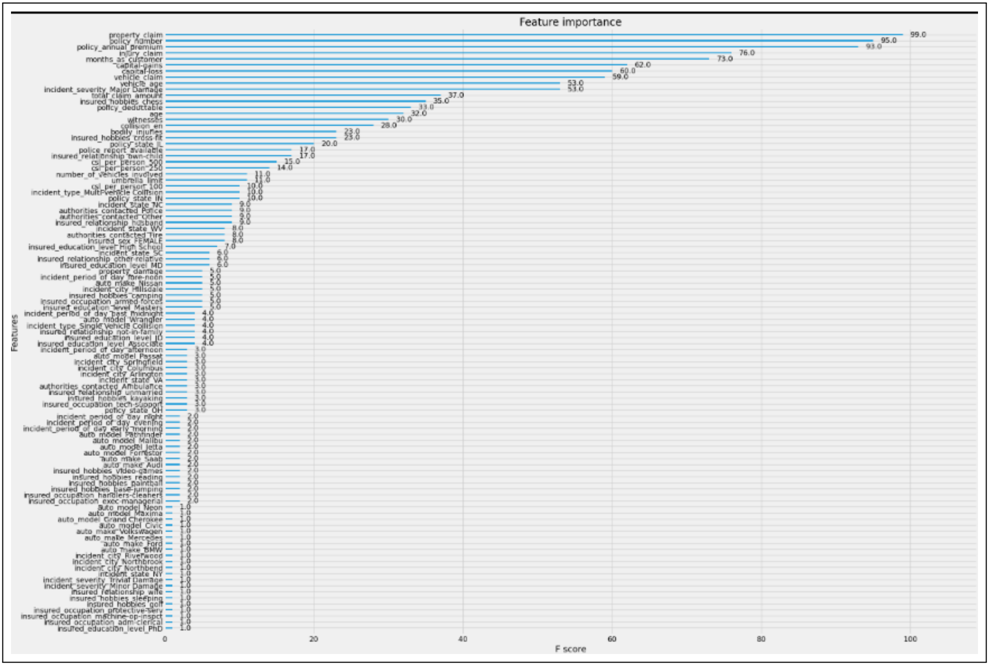
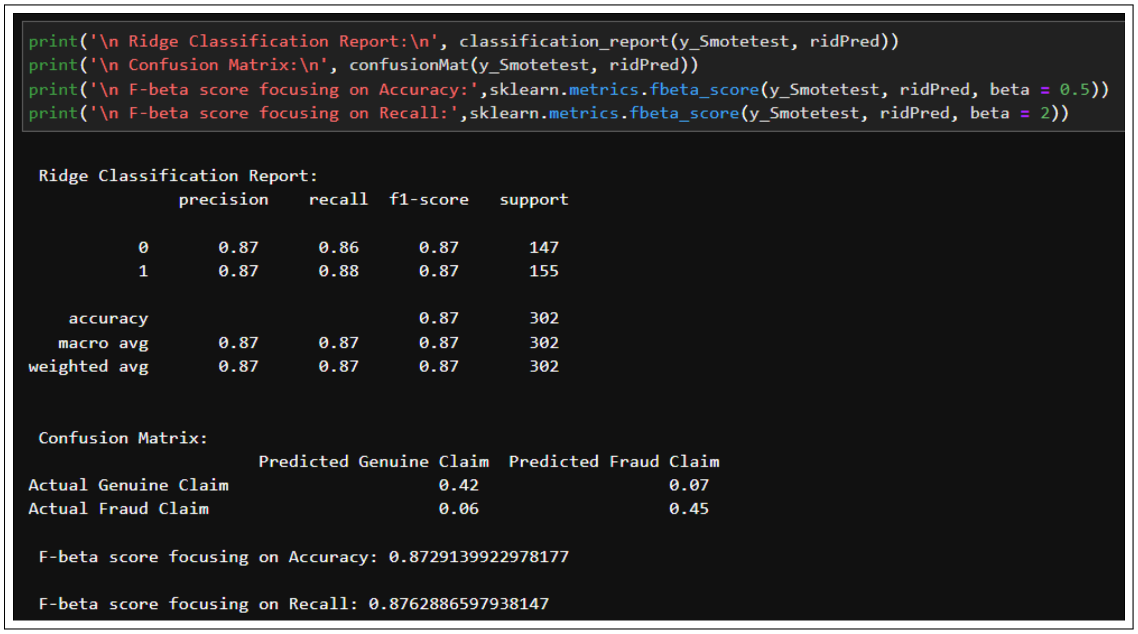


Figure 33: Feature importance of above XGboost model

2. Ridge Classifier performance with Smote Data:

   As illustrated above, the Ridge Classifier with smote oversampling can generate a test accuracy of 87.09% and a recall of 87.74% and a fraudulent F1 score of 87% as illustrated below:

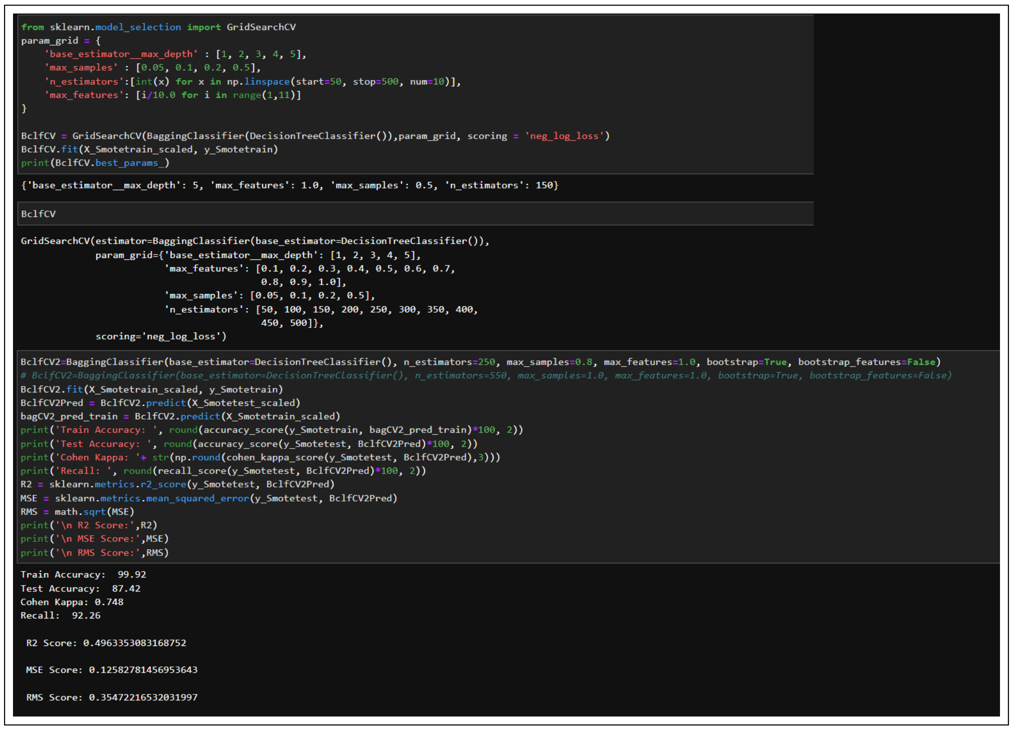


```
print('\n Ridge Classification Report:\n', classification_report(y_Smotetest, ridPred))
print('\n Confusion Matrix:\n', confusionMat(y_Smotetest, ridPred))
print('\n F-beta score focusing on Accuracy:',sklearn.metrics.fbeta_score(y_Smotetest, ridPred, beta = 0.5))
print('\n F-beta score focusing on Recall:',sklearn.metrics.fbeta_score(y_Smotetest, ridPred, beta = 2))
```

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

           0       0.87      0.86      0.87       147
           1       0.87      0.88      0.87       155

    accuracy                           0.87       302
   macro avg       0.87      0.87      0.87       302
weighted avg       0.87      0.87      0.87       302


 Confusion Matrix:
                       Predicted Genuine Claim  Predicted Fraud Claim
Actual Genuine Claim                      0.42                   0.07
Actual Fraud Claim                        0.06                   0.45

 F-beta score focusing on Accuracy: 0.8729139922978177

 F-beta score focusing on Recall: 0.8762886597938147
```

Figure 34: Ridge Classifier with Smote dataset performance summary.

3. Bagging Classifier performance with Smote Data:



```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'base_estimator__max_depth' : [1, 2, 3, 4, 5],
    'max_samples' : [0.05, 0.1, 0.2, 0.5],
    'n_estimators':[int(x) for x in np.linspace(start=50, stop=500, num=10)],
    'max_features': [i/10.0 for i in range(1,11)]
}

BclfCV = GridSearchCV(BaggingClassifier(DecisionTreeClassifier()),param_grid, scoring = 'neg_log_loss')
BclfCV.fit(X_Smotetrain_scaled, y_Smotetrain)
print(BclfCV.best_params_)
```

```
{'base_estimator__max_depth': 5, 'max_features': 1.0, 'max_samples': 0.5, 'n_estimators': 150}
```

```
BclfCV
```

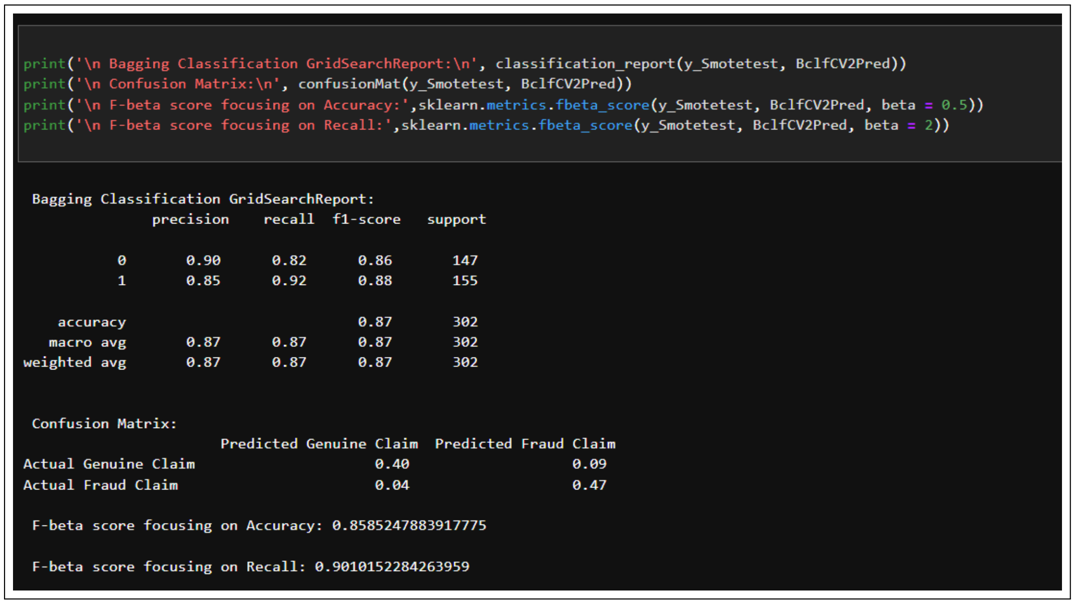
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier()),
             param_grid={'base_estimator__max_depth': [1, 2, 3, 4, 5],
                         'max_features': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
                                          0.8, 0.9, 1.0],
                         'max_samples': [0.05, 0.1, 0.2, 0.5],
                         'n_estimators': [50, 100, 150, 200, 250, 300, 350, 400,
                                          450, 500]},
             scoring='neg_log_loss')
```

```
BclfCV2=BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=250, max_samples=0.8, max_features=1.0, bootstrap=True, bootstrap_features=False)
# BclfCV2=BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=550, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False)
BclfCV2.fit(X_Smotetrain_scaled, y_Smotetrain)
BclfCV2Pred = BclfCV2.predict(X_Smotetest_scaled)
bagCV2_pred_train = BclfCV2.predict(X_Smotetrain_scaled)
print('Train Accuracy: ', round(accuracy_score(y_Smotetrain, bagCV2_pred_train)*100, 2))
print('Test Accuracy: ', round(accuracy_score(y_Smotetest, BclfCV2Pred)*100, 2))
print('Cohen Kappa: '+ str(np.round(cohen_kappa_score(y_Smotetest, BclfCV2Pred),3)))
print('Recall: ', round(recall_score(y_Smotetest, BclfCV2Pred)*100, 2))
R2 = sklearn.metrics.r2_score(y_Smotetest, BclfCV2Pred)
MSE = sklearn.metrics.mean_squared_error(y_Smotetest, BclfCV2Pred)
RMS = math.sqrt(MSE)
print('\n R2 Score:',R2)
print('\n MSE Score:',MSE)
print('\n RMS Score:',RMS)
```

```
Train Accuracy:  99.92
Test Accuracy:  87.42
Cohen Kappa: 0.748
Recall:  92.26

 R2 Score: 0.4963353083168752

 MSE Score: 0.12582781456953643

 RMS Score: 0.35472216532031997
```

Figure 35: Bagging Classifier with Smote dataset Test Acc, Train Acc and Recall.

As illustrated above, the Bagging Classifier with smote oversampling can generate a test accuracy of 87.42% and a recall of 92.26%, and a fraudulent F1 score of 88% as illustrated below:



```
print('\n Bagging Classification GridSearchReport:\n', classification_report(y_Smotetest, BclfCV2Pred))
print('\n Confusion Matrix:\n', confusionMat(y_Smotetest, BclfCV2Pred))
print('\n F-beta score focusing on Accuracy:',sklearn.metrics.fbeta_score(y_Smotetest, BclfCV2Pred, beta = 0.5))
print('\n F-beta score focusing on Recall:',sklearn.metrics.fbeta_score(y_Smotetest, BclfCV2Pred, beta = 2))
```

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

           0       0.90      0.82      0.86       147
           1       0.85      0.92      0.88       155

    accuracy                           0.87       302
   macro avg       0.87      0.87      0.87       302
weighted avg       0.87      0.87      0.87       302


 Confusion Matrix:
                       Predicted Genuine Claim  Predicted Fraud Claim
Actual Genuine Claim                     0.40                   0.09
Actual Fraud Claim                       0.04                   0.47

 F-beta score focusing on Accuracy: 0.8585247883917775

 F-beta score focusing on Recall: 0.9010152284263959
```

Figure 36: Bagging Classifier with Smote dataset performance summary.

4.9 Best performing model

As observed from the above sections, XG-boost with smote data generates a test accuracy of 88.08%, Recall of 90.32 %, and fraudulent F1 score of 89%. It miss-classifies 7% of fraud cases and has an F-beta score of 89.62% on recall.

5 Summary & Conclusion

Fraud accounted for between 15 % and 21 % of total claims payments for auto insurance bodily injury in 2012, according to an Insurance Research Council (IRC) study. The study estimated that between \$5.6 billion and \$7.7 billion was fraudulently added to paid claims for auto insurance bodily injury payments in 2012, compared with a range of \$4.3 billion to \$5.8 billion in 2002.

We have built a model to detect auto insurance fraud in this project. The challenge behind fraud detection in machine learning is that frauds are far less common than legit insurance claims.

We used the following models project: logistic regression, SVM, XG-Boost, Lazy Classifier Ridge Classifier, and Bagging Classifier. And handled oversampling with SMOTE.

The best and final fitted model was a weighted XGBoost that yelled an accuracy of 88.08% F1 score of 89%, Recall of 90.32 %, and a ROC AUC of 87%. The model performed better than our initial F1 score of 0.68 and ROC AUC of 0.63. XG boost F1 score and ROC AUC scores were the highest amongst the other models. In conclusion, the model could correctly distinguish between fraud claims and legit claims with high accuracy, with only miss-classifying 7% fraudulent claims as genuine claims.

As observed in the above sections, based on the model, we can suggest that the following features carry higher weightage for our model learning:

1. Property Claim
2. Policy annual premium
3. Injury Claim
4. Capital gains

Hence we suggest the field experts focus on these parameters. Apart from this, the imbalance of the data was a huge hindrance for our models to perform better; hence they can also focus on creating synthetic data to offset this imbalance. And further, also look into more historic as well as other credit details to make informed decisions on the same.

The study has several restrictions. The first limitation of this study is the limited sample size. Larger data sets increase the stability of statistical models. During the second meeting, this issue was brought up with the clients, but, owing to privacy issues, they were unable to share the real industry of data. Additionally, because it represents a larger fraction of the actual population, it generalizes better. Additionally, the data only includes event claims from three states between January 1 and March 1, 2015. As a result, we are unsure of what percentage of auto insurance policyholders experienced no occurrences vs those who did. We cannot include instances older than two months, so the picture we get may not represent the entire year. This is significant since there may be seasonal variations in occurrence rates. Future research may look toward collecting a larger data set spanning several years. Due to the delicate nature of fraud and the sensitive information attached to such data, this may still be difficult. We can also compare data from cross-insurance platforms for the same users, like health, housing, etc. To create new features and give us more insights into the user's behaviors and patterns. Additionally given more time we can also explore ensemble learning, which, when we explored in the later stages of our projects, was able to generate better accuracy of approx. 91% but due to deadlines we are unable to explore this avenue further. Apart from this, in the future, we can also implement an incremental input feature model (sub-set of input parameters) based on the importance of features as mentioned by the field experts for more custom-tailored solutions.

6 Data Links

1. **Insurance Dataset:** `https://www.kaggle.com/datasets/ssmohanty/insurance-claims-data?resource=download`
2. **GitHub Code Repository:** `https://github.com/DineshchandarR/CPSC_6300_Insurance_Project/blob/master/InsuranceProject_Final.ipynb`

References

[1] BREIMAN, L. Bagging predictors. *Machine Learning 24* (1996).

[2] BREIMAN, L. Pasting small votes for classification in large databases and on-line. *Machine Learning 36* (2004), 85–103.

[3] CHEN, M., LIU, Q., CHEN, S., LIU, Y., ZHANG, C.-H., AND LIU, R. Xgboost-based algorithm interpretation and application on post-fault transient stability status prediction of power system. *IEEE Access 7* (2019), 13149–13158.

[4] LEI, S., XU, K., HUANG, Y., AND SHA, X. An xgboost based system for financial fraud detection. *E3S Web of Conferences 214* (01 2020), 02042.

[5] SAGAR, R. Deep Learning, XGBoost, Or Both: What Works Best For Tabular Data?