Insurance Claims- Fraud Detection

Introduction :

Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

Insurance fraud is a serious and growing problem, and there is widespread recognition that traditional approaches to tackling fraud are inadequate. Studies of insurance fraud have typically focused upon identifying characteristics of fraudulent claims and claimants, and this focus is apparent in the current wave of forensic and data-mining technologies for fraud detection. An alternative approach is to understand and then optimize existing practices in the detection of fraud.

Definition of the Problem:

The main ethnography consisted of observation and interaction in two different departments of Company A, a major multinational insurance company. The first department had responsibility for claims pertaining to commercial vehicle policies. The second department mirrored these responsibilities for personal vehicle policies. The ethnography lasted for two months and took place during regular working hours. During this period, the ethno graphersat with claims handlers, attended meetings and training courses, and observed work practices, on occasion asking questions about work activity and encouraging staff to talk aloud whilst completing claims-handling tasks. Observations were interspersed with periods of field-note writing to ensure that each part of the day was sampled across the two-month period.

Data Pre-processing Done:

The purpose of preprocessing is to convert raw data into a form that fits machine learning. Structured and clean data allows a data scientist to get more precise results from an applied machine learning model. The technique includes data formatting, cleaning, and sampling.

- Hardware and Software Requirements and Tools Used
- Tools: Python 3.8.5, Jupyter Notebook, Numpy, Pandas, Matplotlib, Seaborn, Scikit-learn, Scipy
- Techniques: classification, Decision Tree classifier, Random Forest Classifier, Ada-Boosting Classifier, Logistic Regression , SVC
- Hardware: I3 processor, 4GB RAM

- Model/s Development and Evaluation

I have used the below models for Classifier:

1. SVC
2. Random Forest Classifier
3. Decision Tree Classifier
4. KNeighbors Classifier
5. Ada Boost Classifier
6. Logistic Regression

- ## Identification of possible problem-solving approaches (methods)

7. Read the data (from csv)
8. Identify the dependent and independent variables.
9. Check if the data has missing values or the data is categorical or not.
10. If yes, apply basic data preprocessing operations to bring the data in a go to go format.
11. Now split the data into the groups of training and testing for the respective purpose.
12. After splitting data, fit it to a most suitable model. (How to find a suitable model is answered below)
13. Validate the model. If satisfactory, then go with it, else tune the parameters and keep testing. In a few cases, you can also try different algorithms for the same problem to understand the difference between the accuracies.
14. From step 7 one can also learn about accuracy paradox.
15. Visualize the data.

- ## Testing of Identified Approaches (Algorithms)

1. KNN
2. Naive Bayes
3. Cross validation
4. Confusion matrix
5. Accuracy score

- ## Run and Evaluate selected models

I have used the below models for classification:

Logistic regression

```
lg=LogisticRegression()
lg.fit(x_train,y_train)
pred=lg.predict(x_test)
lg_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.72
[[180   0]
 [ 70   0]]
              precision    recall  f1-score   support

           0       0.72      1.00      0.84       180
           1       0.00      0.00      0.00        70

    accuracy                           0.72       250
   macro avg       0.36      0.50      0.42       250
weighted avg       0.52      0.72      0.60       250
```

RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
pred=rf.predict(x_test)
rf_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.748
[[169  11]
 [ 52  18]]
              precision    recall  f1-score   support

           0       0.76      0.94      0.84       180
           1       0.62      0.26      0.36        70

    accuracy                           0.75       250
   macro avg       0.69      0.60      0.60       250
weighted avg       0.72      0.75      0.71       250
```

DecisionTreeClassifier

```
]: dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
pred=dt.predict(x_test)
dt_accu=accuracy_score(y_test,pred)

print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.832
[[160  20]
 [ 22  48]]
              precision    recall  f1-score   support

           0       0.88      0.89      0.88       180
           1       0.71      0.69      0.70        70

    accuracy                           0.83       250
   macro avg       0.79      0.79      0.79       250
weighted avg       0.83      0.83      0.83       250
```

SVC

```
svc=SVC()
svc.fit(x_train,y_train)
pred=svc.predict(x_test)
svc_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.72
[[180   0]
 [ 70   0]]
              precision    recall  f1-score   support

           0       0.72      1.00      0.84       180
           1       0.00      0.00      0.00        70

    accuracy                           0.72       250
   macro avg       0.36      0.50      0.42       250
weighted avg       0.52      0.72      0.60       250
```

AdaBoostClassifier

```python
from sklearn.ensemble import AdaBoostClassifier
ad=AdaBoostClassifier()
ad.fit(x_train,y_train)
pred=ad.predict(x_test)
ad_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.792
[[169  11]
 [ 41  29]]
              precision    recall  f1-score   support

           0       0.80      0.94      0.87       180
           1       0.72      0.41      0.53        70

    accuracy                           0.79       250
   macro avg       0.76      0.68      0.70       250
weighted avg       0.78      0.79      0.77       250
```

```python
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
pred=knn.predict(x_test)
knn_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

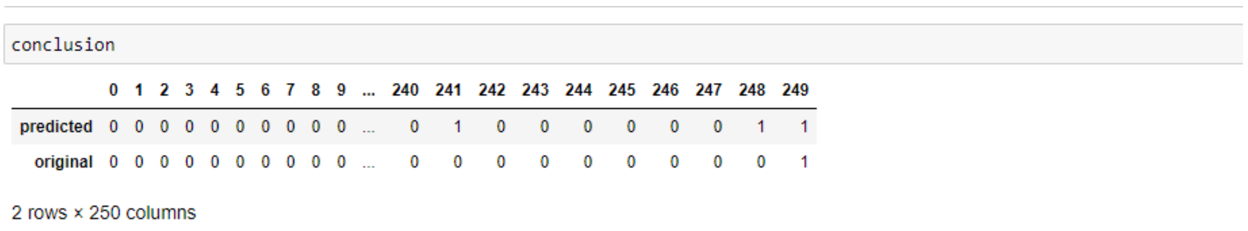
```
0.652
[[160  20]
 [ 67   3]]
              precision    recall  f1-score   support

           0       0.70      0.89      0.79       180
           1       0.13      0.04      0.06        70

    accuracy                           0.65       250
   macro avg       0.42      0.47      0.43       250
weighted avg       0.54      0.65      0.58       250
```

- Conclusion:

Decision Tree Classifier is giving Maximum Accuracy as compare to other classification algorithm. Hence Decision Tree Classifier is the best model of this dataset.



```
conclusion
```

	0	1	2	3	4	5	6	7	8	9	...	240	241	242	243	244	245	246	247	248	249
predicted	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	1	1
original	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

2 rows × 250 columns