IBM COURSERA ADVANCED DATA SCIENCE CAPSTONE

Fraud Detection : Exploring methods to classify accounts as fraud or not

-Soham Mukherjee

Outline

Use Case

Data Set

Data Assessment

Data Preprocessing

Data Feature Engineering

Architectural Choices

Model performance indicator

Base Model - Logistic Regression

Advanced Model - Neural Net

Final Model Performance

Use Case

Objective : The objective of the project is to build an automated solution to classifying an account as a fraudulent account or not.

Procedure : The process followed here is to build a data driven machine learning or deep learning model that can learn from historical data and which can be used to classify new records as fraudulent or not.

Use Case - Solution Overview

01 Get historical data with a label marking the account information as Fraud or not

02 Explore and Pre-Process data to remove data quality issues and make it fit for model training

03 Train and compare different models by measuring performances

Data Set

The Bank Account Fraud (BAF) suite of datasets has been published at **NeurIPS 2022** and it comprises a total of 6 different synthetic bank account fraud tabular datasets.

This suite of datasets is:

- Realistic, based on a present-day real-world dataset for fraud detection;
- Biased, each dataset has distinct controlled types of bias;
- Imbalanced, this setting presents a extremely low prevalence of positive class;
- Dynamic, with temporal data and observed distribution shifts;
- Privacy preserving, to protect the identity of potential applicants we have applied differential privacy techniques (noise addition), feature encoding and trained a generative model (CTGAN).

I have used 1 of the 6 datasets - Base.csv for this project

Source - https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022/data

Data Assessment

Few data assessment that has been is to check the quality issues and the balance of the data.

Missing values	Balance of data
Data dictionary mentions presence of missing values in some numeric fields. Below is the table showing the count of missing values.	There was imbalance in the feud indicator which was observed in the data

```
income_count                            | 0
name_email_similarity_count             | 0
prev_address_months_count_count         | 712920
current_address_months_count_count      | 4254
customer_age_count                      | 0
days_since_request_count                | 0
intended_balcon_amount_count            | 742523
zip_count_4w_count                      | 0
velocity_6h_count                       | 44
velocity_24h_count                      | 0
velocity_4w_count                       | 0
bank_branch_count_8w_count              | 0
date_of_birth_distinct_emails_4w_count  | 0
credit_risk_score_count                 | 14445
bank_months_count_count                 | 253635
proposed_credit_limit_count             | 0
session_length_in_minutes_count         | 2015
device_distinct_emails_8w_count         | 359
device_fraud_count_count                | 0
month_count                             | 0
```

```
[6]: df.fraud_bool.value_counts()

[6]: fraud_bool
     0    988971
     1     11029
     Name: count, dtype: int64
```

Data Preprocessing

Data preprocessing involved the following steps

Missing value - Removing columns with high number of missing values

```
# All columns counts less than 0
res = df[df[numeric_columns] < 0 ].count()
```

Scaling values

```
# Scale the numeric features in the training and testing sets using MinMaxScaler
numeric_transformer = MinMaxScaler()

# Define the ColumnTransformer object with the numeric transformer and the list of numeric fea
preprocessor = ColumnTransformer([('scaled', numeric_transformer, numeric_columns)], remainder

# Fit the preprocessor on the training set and transform both the training and testing sets
new_df_scaled = preprocessor.fit_transform(new_df)
```

One hot encoding

```
# Convert categorical variables into dummy variables using one-hot encoding
new_df = pd.DataFrame(pd.get_dummies(df, prefix=categorical_columns))
```

SMOTE - **Synthetic Minority Oversampling Technique**

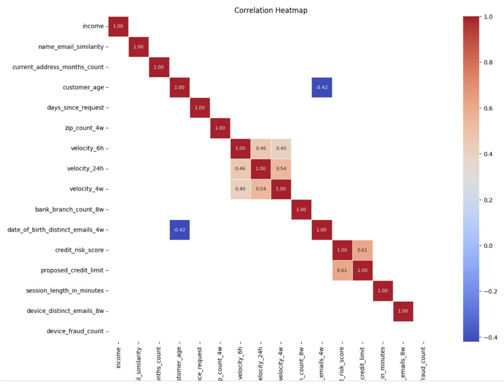
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy='auto', random_state=42)
X, y = smote.fit_resample(X, y)
```

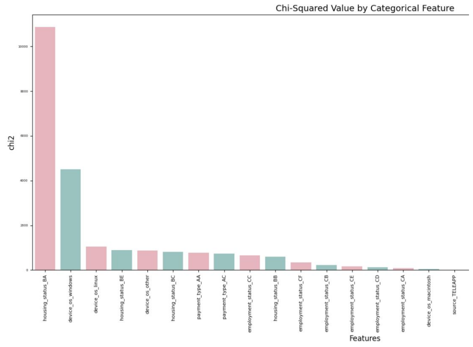
Data Feature Engineering

2 methods were used to select the best features after data preprocessing

Removed columns with high missing values → Removed highly correlated data → Categorical feature selection

```
income_count                            | 0
name_email_similarity_count             | 0
prev_address_months_count_count         | 712920
current_address_months_count_count      | 4254
customer_age_count                      | 0
days_since_request_count                | 0
intended_balcon_amount_count            | 742523
zip_count_4w_count                      | 0
velocity_6h_count                       | 44
velocity_24h_count                      | 0
velocity_4w_count                       | 0
bank_branch_count_8w_count              | 0
date_of_birth_distinct_emails_4w_count  | 0
credit_risk_score_count                 | 14445
bank_months_count_count                 | 253635
proposed_credit_limit_count             | 0
session_length_in_minutes_count         | 2015
device_distinct_emails_8w_count         | 359
device_fraud_count_count                | 0
month_count                             | 0
```





Architectural Choices

Data Source	CSV
Data Repository	Local File System
Discovery and Exploration	Jupyter notebook , Python, Pandas / Spark, Matplotlib, Seaborn

Model Performance Indicator

This is a classification problem . Thus the following were used to assess model performance :

- Confusion Matrix
- Accuracy
- Precision
- Recall
- F1-score
- Area Under ROC

Base Model - Logistic Regression

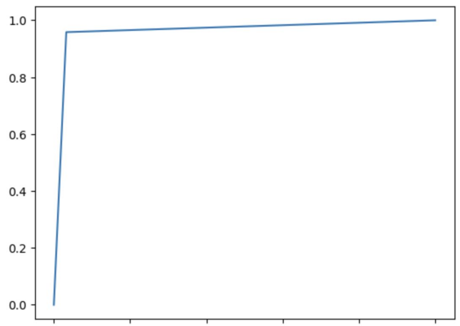
Data was split - X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

A cross validated Logistic Regression was built and the following was the model performance.

clf = LogisticRegressionCV(cv=2, random_state=0,max_iter=5,penalty = 'l2',n_jobs=4).fit(X_train, y_train)

```
fpr, tpr, thresholds = roc_curve(y_test,y_pred)
plt.plot(fpr,tpr)
```

```
[<matplotlib.lines.Line2D at 0x7dbff12de980>]
```




```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.96    299654
           1       0.96      0.97      0.96    293729

    accuracy                           0.96    593383
   macro avg       0.96      0.96      0.96    593383
weighted avg       0.96      0.96      0.96    593383
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[287300   9661]
 [ 12354 284068]]
```

Advanced Model - Neural Net

Keras was used to build a Convolution Neural Network as this is considered to work great with classification tasks.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
```

```
feature_dim = X_train.shape[1]
feature_dim
```

```
37
```

+ Code + Markdown

```
# Define the first CNN model
cnn_model_1 = Sequential()
cnn_model_1.add(Conv1D(filters=16, kernel_size=3, activation='relu', input_shape=(feature_dim, 1)))
cnn_model_1.add(MaxPooling1D(pool_size=2))
cnn_model_1.add(Flatten())
cnn_model_1.add(Dense(16, activation='relu'))
cnn_model_1.add(Dense(1, activation='sigmoid'))
```

```
# Compile all models
cnn_model_1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Advanced Model - Neural Net

Model Performance

```
cnn_model_1.fit(X_train, y_train, epochs=15, batch_size=1000, validation_data=(X_test, y_test))
```

```
Epoch 1/15
1385/1385 [==============================] - 10s 4ms/step - loss: 0.1430 - accuracy: 0.9471 - val_loss: 0.1019 - val_accuracy: 0.9626
Epoch 2/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0992 - accuracy: 0.9633 - val_loss: 0.0961 - val_accuracy: 0.9642
Epoch 3/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0949 - accuracy: 0.9646 - val_loss: 0.0928 - val_accuracy: 0.9653
Epoch 4/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0925 - accuracy: 0.9652 - val_loss: 0.0911 - val_accuracy: 0.9657
Epoch 5/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0911 - accuracy: 0.9658 - val_loss: 0.0910 - val_accuracy: 0.9657
Epoch 6/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0903 - accuracy: 0.9661 - val_loss: 0.0895 - val_accuracy: 0.9664
Epoch 7/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0896 - accuracy: 0.9663 - val_loss: 0.0896 - val_accuracy: 0.9659
Epoch 8/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0892 - accuracy: 0.9664 - val_loss: 0.0892 - val_accuracy: 0.9663
Epoch 9/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0888 - accuracy: 0.9665 - val_loss: 0.0884 - val_accuracy: 0.9669
Epoch 10/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0884 - accuracy: 0.9667 - val_loss: 0.0884 - val_accuracy: 0.9666
Epoch 11/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0881 - accuracy: 0.9668 - val_loss: 0.0875 - val_accuracy: 0.9671
Epoch 12/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0881 - accuracy: 0.9667 - val_loss: 0.0885 - val_accuracy: 0.9664
Epoch 13/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0877 - accuracy: 0.9669 - val_loss: 0.0883 - val_accuracy: 0.9664
Epoch 14/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0874 - accuracy: 0.9670 - val_loss: 0.0871 - val_accuracy: 0.9671
Epoch 15/15
1385/1385 [==============================] - 6s 4ms/step - loss: 0.0871 - accuracy: 0.9671 - val_loss: 0.0882 - val_accuracy: 0.9665
```

Final Model - Neural Net

Model Performance

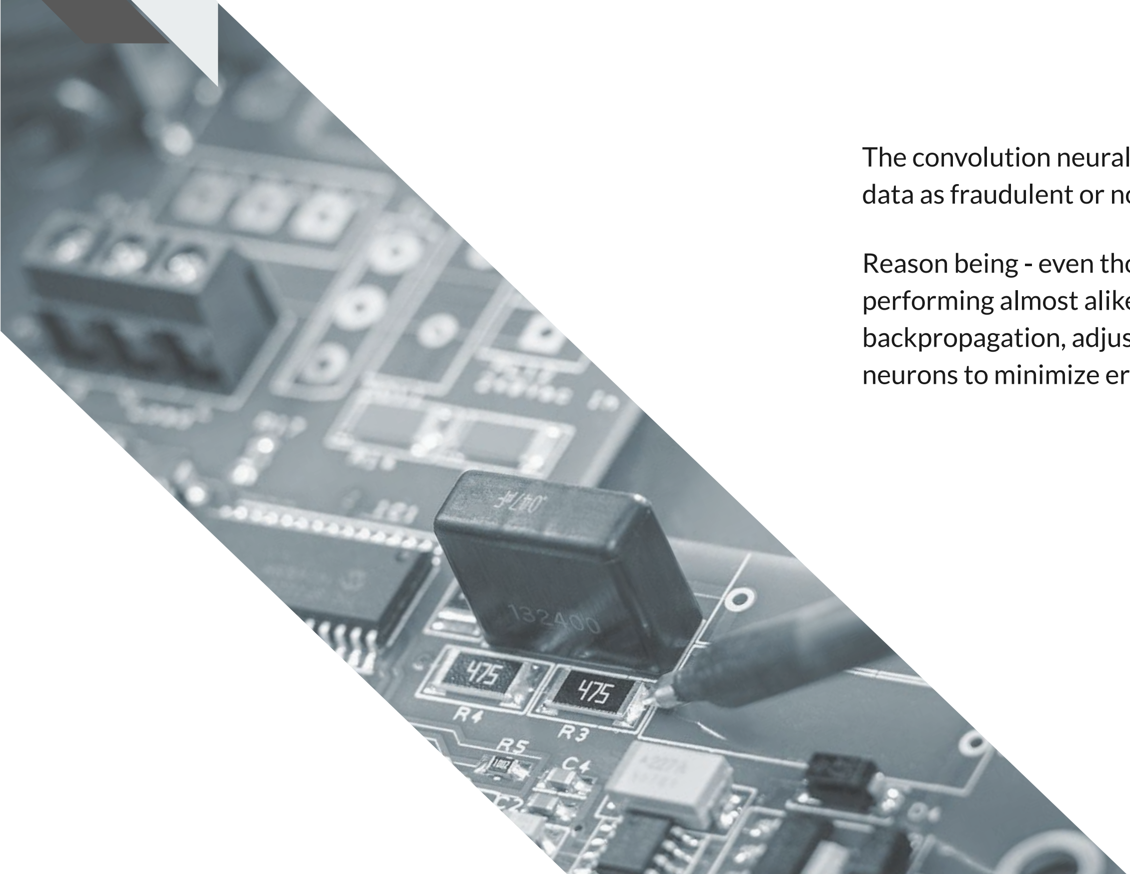
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,confusion_matrix
```

```
## Evaluating the network
def evaluate(y_test, y_pred):
    accuracy = accuracy_score(y_true=y_test, y_pred=y_pred)
    precision = precision_score(y_true=y_test, y_pred=y_pred)
    recall = recall_score(y_true=y_test, y_pred=y_pred)
    f1 = f1_score(y_true=y_test, y_pred=y_pred)
    cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
    print("Accuracy: ", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("Confusion Matrix:\n", cm)
```

```
evaluate(y_test, np.round(cnn_predictions_1))
```

```
Accuracy:  0.9664820192017635
Precision: 0.9659720888223856
Recall: 0.966966014668277
F1 Score: 0.9664687962046635
Confusion Matrix:
 [[286864  10097]
 [  9792 286630]]
```

Final Model



The convolution neural network is selected to be the final model to classify data as fraudulent or not.

Reason being - even though the logistic regression and the NN is performing almost alike - Neural Network’ ability to undergo training with backpropagation, adjusting the weights and biases in the interconnected neurons to minimize errors and optimize performance.