Adrian Gonzalez
Han Duan
Sarah El Shenawy
Professor Smita
11 June 2023
CSCI 184

Detecting Credit Card Fraud Using Machine Learning

Background

Introduction

Credit card fraud has become a major concern for financial institutions and cardholders around the world. According to the Federal Trade Commission, there were a reported 389,737 cases of credit card fraud in the US in 2021, with about $181 million stolen. In response to the rapid growth of digital networks, traditional rule-based fraud detection systems have proven inadequate in combating fraudulent activities. With the rise of machine learning, we believe that we can use it as a powerful solution to combat credit card fraud. In this project we shed light on the challenges associated with credit card fraud, we describe the key features of machine learning fraud detection systems, and we highlight the value in reducing fraudulent activity.

Challenges in Credit Card Fraud Detection:

Credit card fraud is a constantly evolving threat, posing financial risks for cardholders and losses for financial institutions. Detecting fraudulent transactions in real-time is crucial to minimize damages. However, traditional methods that rely on predefined rules and thresholds struggle to keep pace with rapidly changing fraud patterns. Fraudsters continuously develop new tactics, such as identity theft, and account takeover, making it difficult for rule-based systems to adapt. Additionally, with the rise of digital transactions, fraud has become more of an issue, where fraudulent transactions can happen without access to the physical copy of the credit card. The traditional systems have a hard time detecting many of these issues, as they were built for face to face transactions.

The Role of Machine Learning:

Machine learning offers a promising approach to improve credit card fraud detection by using data analysis and pattern recognition. By utilizing transaction data, machine learning algorithms can learn patterns and behaviors associated with fraudulent activities. This enables the creation of predictive models that can detect anomalies and suspicious patterns, improve fraud detection accuracy and reduce false positives.

Project Idea

In this project we will train a system that can track the pattern of all the transactions and if any pattern is abnormal then the transaction should be aborted. We aim to build a machine learning model that can predict whether a credit card transaction is fraudulent or not.

We used this dataset that contains the real bank transactions made by European cardholders in the year 2013.https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.
We will have these major steps in our project

1. Data collection, Preprocessing and understanding
2. Model building
3. Evaluation
4. Conclusion

Data Collection and Preprocessing:

A comprehensive fraud detection system starts with collecting relevant data, including transaction records, user information, and external data sources. Preprocessing involves data cleaning, normalization, and feature engineering, ensuring that the data is suitable for training machine learning models. Feature selection, which is when you identify the most relevant features from the collected data, is crucial for effective fraud detection. During this step, We noticed that the dataset is imbalanced towards some features, which makes sense because many banks probably have vulnerabilities in the same area due to how security is set up. In this data set 0.17% of transactions are fraudulent. We then scaled the data, and dropped the time feature and duplicate transactions to ensure a better model building process.

```
Total_transactions = len(data)
normal = len(data[data.Class == 0])
fraudulent = len(data[data.Class == 1])
fraud_percentage = round(fraudulent/normal*100, 2)
print(cl('Total number of Trnsactions are {}'.format(Total_transactions), attrs = ['bold']))
print(cl('Number of Normal Transactions are {}'.format(normal), attrs = ['bold']))
print(cl('Number of fraudulent Transactions are {}'.format(fraudulent), attrs = ['bold']))
print(cl('Percentage of fraud Transactions is {}'.format(fraud_percentage), attrs = ['bold']))
```

```
Total number of Trnsactions are 284807
Number of Normal Transactions are 284315
Number of fraudulent Transactions are 492
Percentage of fraud Transactions is 0.17
```

```
In [8]: sc = StandardScaler()
        amount = data['Amount'].values
        data['Amount'] = sc.fit_transform(amount.reshape(-1, 1))

In [9]: data.drop(['Time'], axis=1, inplace=True)

In [10]: data.drop_duplicates(inplace=True)

In [11]: data.shape
Out[11]: (275663, 30)
```

Model Building:

Several machine learning algorithms, such as logistic regression, decision trees, random forests, and neural networks, can be used to create predictive models. These models are trained on labeled datasets, where fraudulent and legitimate transactions are appropriately labeled. The models learn the underlying patterns in the data and generate predictions for new, unseen transactions. The machine learning models learn to recognize and capture relationships between different transactional attributes, like the amount of money, the location, the time, the merchant category, and other cardholder information. This information is then all used to learn the knowledge and be able to predict for new transactions if the transaction is likely to be fraudulent or legitimate. It is important to note that the training data sets will be very important for the success of machine learning model predictions. For efficiency, we want to ensure that there is a large data set that accurately covers a large amount of information and has good representation and classification.

In the end, we decided to use decision trees, random forest, and KNN neighbors algorithms as our machine learning models because these models are best suited for classification, which is what we are trying to do. We split our data in training and testing data. Training data is the subset of original data that is used to train the machine learning model, whereas testing data is used to check the accuracy of the model.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

Background for Evaluation and Performance Metrics:

Once a system mastering based fraud detection device is applied, it is crucial to analyze the overall performance and make certain that fraudulent transactions are appropriately detected. Many overall performance metrics may be used to degree the effectiveness of the machine, e.G.

Confusion Matrix: The confusion matrix is a great tool to evaluate the performance of a fraud detection system. It provides a detailed breakdown of the predicted and actual labels, categorizing transactions into true positives, true negatives, false positives, and false negatives. From the confusion matrix, we can derive other metrics such as accuracy, precision, recall, and F1 score.

Accuracy: The overall accuracy of the machine learning-based fraud detection system in classifying transactions as fraudulent or legitimate correctly.

Precision: (TP/(TP+FP))The precision metric measures the proportion of correctly identified fraudulent transactions out of all transactions classified as fraudulent. It will show the system's ability to avoid false positives.

F1 Score: The F1 score combines precision and recall, providing a balanced measure of the system's performance. It gives equal weight to precision and recall.

False Positive Rate (FPR): FPR measures the proportion of legitimate transactions that are incorrectly classified as fraudulent. Minimizing the number of false positives is crucial to avoid annoying genuine cardholders with unnecessary security measures.

False Negative Rate (FNR): FNR measures the proportion of fraudulent transactions that are incorrectly classified as legitimate. Minimizing false negatives is critical to preventing fraudulent activities from going undetected.

By evaluating system performance using these metrics, financial institutions can evaluate the effectiveness of a machine learning-based fraud detection system and identify areas for improvement. Regular monitoring and evaluation allows adjustments to be made, ensuring the accuracy and effectiveness of the system to combat credit card fraud.

Evaluation:

```
#Decision Trees
DT = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
DT.fit(X_train, y_train)
dt_yhat = DT.predict(X_test)
```

```
print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(y_test, dt_yhat)))
```

```
Accuracy score of the Decision Tree model is 0.9991438853096524
```

```
print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, dt_yhat)))
```

```
F1 score of the Decision Tree model is 0.7467811158798283
```

```
confusion_matrix(y_test, dt_yhat, labels = [0, 1])
```

```
array([[68770,    18],
       [   41,    87]])
```

Pros for using decision trees:
Interpretability: Decision Trees offer transparent and interpretable models, allowing easy understanding of the decision rules and feature importance.
Handling non-linear relationships: Decision Trees can capture nonlinear relationships between features and the target variable, making them suitable for complex datasets.
The cones are that Decision Trees are prone to overfitting, particularly when the tree becomes too deep or complex, leading to poor generalization on unseen data.
In our confusion matrix, the first row represents positive and the second row represents negative. We have 68770 as true positives and 18 are false positives. That says, out of 68770+18=68788 samples, we have 68770 that are successfully classified as a normal transaction and 18 were falsely classified as normal, but were in fact fraudulent transactions.

```
#Random forest
rf = RandomForestClassifier(max_depth = 4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
```

```
print('Accuracy score of the Random Forest model is {}'.format(accuracy_score(y_test, rf_yhat)))
```

```
Accuracy score of the Random Forest model is 0.9991148644726914
```

```
print('F1 score of the Random Forest model is {}'.format(f1_score(y_test, rf_yhat)))
```

```
F1 score of the Random Forest model is 0.7162790697674419
```

We chose Random forest due to its high predictive accuracy due to the ensemble of decision trees and feature randomness. It can handle both linear and non-linear relationships in the data.
Random Forest also mitigates overfitting by aggregating predictions from multiple trees, reducing the risk of capturing noise or irrelevant patterns.It can also handle class imbalance issues effectively by balancing the class distribution during training, reducing the bias towards the majority class. One big downside we realized was its complexity. It is computationally

expensive and time-consuming, especially for large datasets and complex feature spaces. In our case, the random forest model took a very long time to compute.

```
#KNN
from sklearn.neighbors import KNeighborsClassifier

KNN = KNeighborsClassifier(n_neighbors = 7, n_jobs=-1)

KNN.fit(X_train, y_train)
knn_yhat = KNN.predict(X_test)
```

```
print('Accuracy score of the K-Nearest Neighbors model is {}'.format(accuracy_score(y_test, knn_yhat)))
print('F1 score of the K-Nearest Neighbors model is {}'.format(f1_score(y_test, knn_yhat)))
```

KNN is a straightforward algorithm that relies on the proximity of data points, making it easy to understand and implement. It does not assume any specific data distribution, making it applicable to a wide range of datasets. It can also adapt to changes in the dataset by retraining the model with new data points, making it suitable for dynamic environments.
However, KNN requires computing distances between the query point and all training samples, making it computationally expensive for large datasets. Our sample took a very long time to compute as well. It is also sensitive to the scale and range of features, as distances are calculated based on the individual feature values. Proper feature scaling is crucial for accurate predictions but in our case, we scaled our data in the preprocessing step.

Conclusion

After evaluating the accuracy and F1 scores of each model, it is clear that KNN performs exceptionally well in this case. The KNN model achieved an accuracy score of 99.95% and an F1 score of 0.8365. These results indicate that KNN is highly effective in classifying credit card transactions as either normal or fraudulent.

It is worth noting that the data used for training the models is balanced, which contributes to the high accuracy scores obtained across different models. The transformed version of PCA features used in the dataset might not perfectly represent the actual features. Therefore, further investigation and testing with the original features could provide more insights into the model's performance.

Future Project Idea: Real-Time Monitoring and Feature Engineering

The dataset used for training the models utilizes transformed features obtained through Principal Component Analysis (PCA). Although PCA can help reduce dimensionality, it might result in a loss of interpretability. It would be beneficial to explore other feature engineering techniques or experiment with the original features to capture more nuanced information and improve the model's performance.

In the future, we can also implement real time monitoring. Once the models are trained, they can be deployed in real-time systems to monitor incoming transactions. As new transactions occur, the models evaluate the features and generate predictions in real-time. Transactions flagged as potentially fraudulent are subjected to further investigation or additional security measures.