Credit Card Fraud Detection

Wen Sun

under the direction of Dr. Dan Li
Eastern Washington University

03/20/2024

1 Introduction

This project is to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. This project contains two parts:

- Predict if a transaction is fraudulent or not
- Use data streaming techniques to see how many fraudulent transactions happen in the last k seconds

2 Dataset

This dataset [1] contains transactions made by credit cards in September 2013 by European cardholders. It presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. This means it is very imbalanced, since the positive class (frauds) count for 0.17% of all transactions.

[1] Andrea, Machine Learning Group-ULB. 2018. Credit Card Fraud Detection, Version 1. Retrieved January 20,2024 from https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data

This dataset contains only numerical input (V1,V2,...,V28) based on a PCA transformation. Due to confidentially issues, we do not know the meaning of these features. "Time" and "Amount" variables are not transformed by the PCA. "Time" contains the seconds elapsed between transaction and the first transaction in the dataset. And the "Amount" is the transaction amount. The target variable is "Class", and it takes value 1 as fraud and 0 otherwise.

3 Data processing

Because the "Amount" features are not transformed while others do. We scaled the "Amount" variable by applying standscaler() provided by Spark API. Standscaler is a feature transformer that standardlizes features by removing the mean and scaling to unit standard deviation. It performs a specific type of normalization known as Z-score normalization or standardlization. By doing standardscaler, the model is not biased towards features with larger values.

Then we dropped the "Time" and "Amount" columns and used "AmountScaled-Double" and all other remaining columns to train the model.

We split the dataset into 70% of training and 30% of testing by using stratified sampling. By doing stratified sampling, the proportion of different classes are preserved in both the training and testing datasets,which helps to prevent biaes and ensure that the performance of the model is evaluated fairly across both classes.

4 Data Prediction

Sailusha[1] stated that the measurements we usually use to detect the credit card fraud problems are precision, recall and the F1 score. And the Random Forest algorithm has the highest values than the Adaboost algorithm. Thus we chose to use the random forest method to build the model.

To assess whether the model exhibits signs of overfitting or underfitting, we evaluated its performance on both the training and testing datasets. Overfitting occurs when the model is overly complex, performing exceptionally well on the training data but poorly on unseen testing data. Conversely,

underfitting indicates that the model is too simple, performing poorly on both training and testing datasets.

Upon training the model, we obtained the following metrics for the training dataset: 99% accuracy, 88% precision, 75% recall, and 82% F1 score. Similarly, for the testing dataset, the model achieved 99% accuracy, 88% precision, 75% recall, and 80% F1 score. The consistency in performance metrics across both datasets suggests that the random forest model neither overfits nor underfits the data.

5 Imbalanced dataset

Even though the model has very high accuracy score, its recall score is rather low (0.75). The model has a bias towards the majority class because this dataset is very imbalanced, as the positive class (frauds) only count for 0.17% of all transactions. In other words, the ratio between the fraud and non-fraud is 1: 500.

So, our prediction will have a bias to predict the class as non-fraud because it is the majority class. In this project, we explored three methods to deal with imbalanced datasets - oversampling, undersampling, and SMOTE.

5.1 Methods

Oversampling duplicates samples from the minority class (fraud).If we do oversampling first, and then split into train and test datasets, the model will see a lot of testing data in training as well. As a result, we first separated out the test dataset, and did oversampling in the training dataset.

Similar to oversampling, undersampling removes samples from the majority class (non-fraud).

SMOTE[2] stands for Synthetic Minority Oversampling. It oversamples the minorirty class by generating synthetic minority examples in the neighborhood of observed ones. This idea is to form new minority examples by interpolating between examples of the same class. This has the effect of creating clusters around each minority observation. For this project, we could not find anything implemented for SMOTE using pySpark, so I just explored the idea by using Pandas.

RF	*Original*	*tuning*	*oversample*	*undersample*	$over - undersample$	*SMOTE*
Accuracy	99.9%	99.9%	99.9%	99.9%	99.9%	99.9%
Precision	89.6%	96.3%	97.3%	91.0%	89.3%	90.8%
Recall	72.2%	72.2%	75.7%	77.1%	75%	75.7%
F1	79.9%	82.5%	85.2%	83.5%	81.5%	82.6%

In conclusion, oversampling performs better than other methods, with the accuracy, precision, recall and f1 score being the highest. Other methods also improve the model a bit.

6 Data Streaming

As previously noted, this dataset includes a "Time" column, representing the time elapsed in seconds between each transaction and the first transaction. The dataset spans transactions occurring over two days. Our objective is to analyze the frequency of fraudulent card transactions on an hourly basis. Specifically, we aim to identify whether these occurrences exhibit any noticeable patterns of increased frequency during specific hours.

To facilitate this analysis, we divided the entire dataset into 48 separate CSV files, each containing transactions that occurred within a single hour. Subsequently, we created a streaming DataFrame to process all files within the designated folder. Upon plotting the data, no discernible pattern emerges with respect to particular hours. However, it's possible that patterns may become evident as more data becomes available.

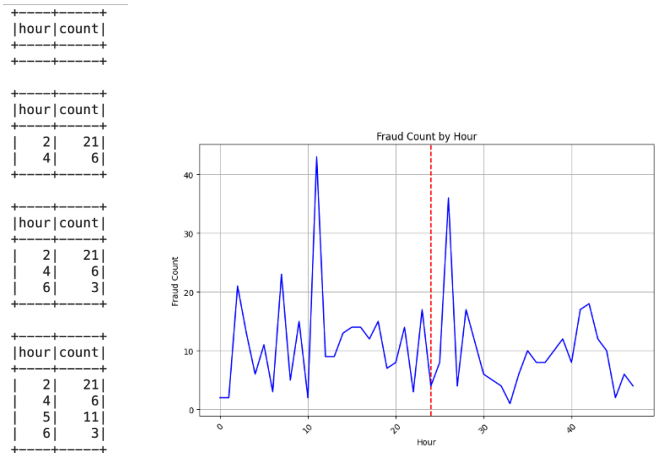

Figure 1: Fraud count within each hour

7 DGIM algorithm

DGIM [3] algorithm uses O($\log^2$N) bits to represents a window of N bits, and allows us to estimate the number of 1's in the window with an error of no more than 50%. Three properties of buckets that are maintained:

- Either one or two buckets with the same power-of-2 number of 1s
- Buckets do not overlap in timestamps
- Buckets are sorted by size

We implemented the DGIM algorithm from scratch, which estimates the number of 1's(fraud count) in the last N seconds.

Dolle [2] has already implemented a opensource API to estimate the number of True in the last N elements of a boolean stream using DGIM algorithm. We used Dolle's implementaion to compare results with ours.

7.1 DGIM result

We applied DGIM algorithm to this credit card fraud dataset to estimate how many fraudulent transactions in the last 3600 seconds (1 hour). When the

[2]Dolle,Simon. dgim. 2014. Retrieved January 20,2024. from https://github.com/simondolle/dgim

new transaction arrive, we continuously update the buckets and estimated count, allowing us to compare them with actual fraud count within that window size. Our evaluation, as depicted in Figure 4, demonstrates that both implementations perform comparably well, with an average error rate between the estimated and actual counts of approximately 9%.

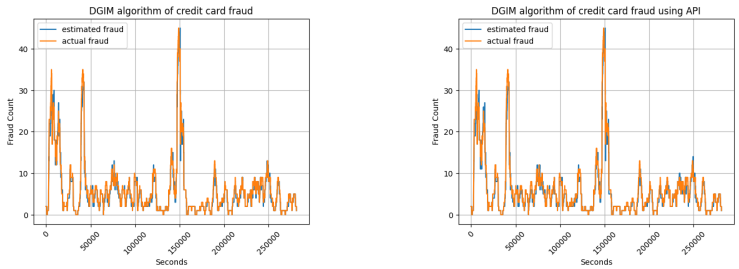

Figure 2: DGIM algorithm accuracy

8 Future work

Due to the time limit, there are some potential work we wish we would have explored further if additional time had been available. For instance, building some applications, like a real-time alarm system for credit card fraud detection. Additionally, expanding the bucket size to see if it would improve the accuracy of the DGIM algorithm that we implemented. Lastly, experimenting different machine learning models (logistic regression...) and other methods to deal with imbalanced dataset (Oversampling the minorirty class and undersampling the majority class at the same time) as this paper discussed [4].

References

[1] Ruttala Sailusha, V. Gnaneswar, R. Ramesh, and G. Ramakoteswara Rao. Credit card fraud detection using machine learning. In *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 1264–1270, 2020.

[2] Jason Brownlee. Smote for imbalanced classification with python, 2021.

[3] Madhura Joshi. Dgim algorithm, 2019.

[4] John O. Awoyemi, Adebayo O. Adetunmbi, and Samuel A. Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. In *2017 International Conference on Computing Networking and Informatics (ICCNI)*, pages 1–9, 2017.