FRAUD DETECTION IN THE FINANCIAL DOMAIN

Prashanthi R
Rathi Kashi
Shivam Agarwal

Abstract

This project aims to detect fraudulent transactions based on previously seen fraudulent points while also detecting new frauds which have not been seen before.

Introduction

Banks and companies lose a lot of money to credit card fraud. It has been estimated that by 2020, online credit card fraud would amount to $32 Billion. Being able to recognize these frauds as and when they happen would be very helpful in reducing the money lost to online credit card fraud and that is what our model aims to do.

We have used a dataset that simulates actual transactions while classifying them as fraudulent or non-fraudulent (https://www.kaggle.com/ntnu-testimon/paysim1). Since the data simulates real life transactions, naturally very few of these transactions are actually fraudulent. Also, there could always be new types of fraud that are not present in the current dataset available to us.

Through this project, we propose a model that combines classification and clustering in order to determine whether a given transaction is fraudulent or not.

The Dataset

The data points represent financial transactions. Each data point has 10 features. A sample data point looks like this:

```
1,PAYMENT,1060.31,C429214117,1089.0,28.69,M1591654462,0.0,0.0,0,0
```

The 10 features are explained below:

1. step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).

2. type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.

3. amount - amount of the transaction in local currency.

4. nameOrig - customer who started the transaction

5. oldbalanceOrg - initial balance before the transaction

6. newbalanceOrig - new balance after the transaction

7. nameDest - customer who is the recipient of the transaction

8. oldbalanceDest - initial balance recipient before the transaction.

9. newbalanceDest - new balance recipient after the transaction.

10. isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control of customer accounts and try to empty the funds by transferring to another account and then cashing out of the system.

11. isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

Fraudulent data points have the value 1 as their corresponding isFraud value, while non-fraudulent data points have the value 0. In the given dataset, there were `6354407` non-fraudulent points and `8213` fraudulent points, i.e., non-fraudulent points and fraudulent points are 99.87% and 0.13% of the total dataset respectively.

Cleaning and working with the data

For the purposes of this project, we have ignored the `step`, `nameOrig`, `nameDest`, and `isFlaggedFraud` columns, because they are not relevant to this classification. We have One-Hot encoded the `type` column which is categorical, i.e., CASH_IN: [1,0,0,0,0], CASH_OUT: [0,1,0,0,0], DEBIT: [0,0,1,0,0], PAYMENT: [0,0,0,1,0], TRANSFER: [0,0,0,0,1]. For building our model, the first five features considered were this encoding of categorical data `type`. Why we used one-hot encoding: one-hot encoding ensures that there is no order or hierarchy imposed on the dataset because of the way the category is represented.

From the analysis of the given dataset, we know that there is a clear class imbalance problem (non-fraudulent points and fraudulent points are 99.87% and 0.13% of the total dataset respectively). This means that, if we train a model using this dataset, it is highly likely to classify a given data point as non-fraudulent because there is very little representation of the minority class in the dataset in order for the model to analyse the pattern well.

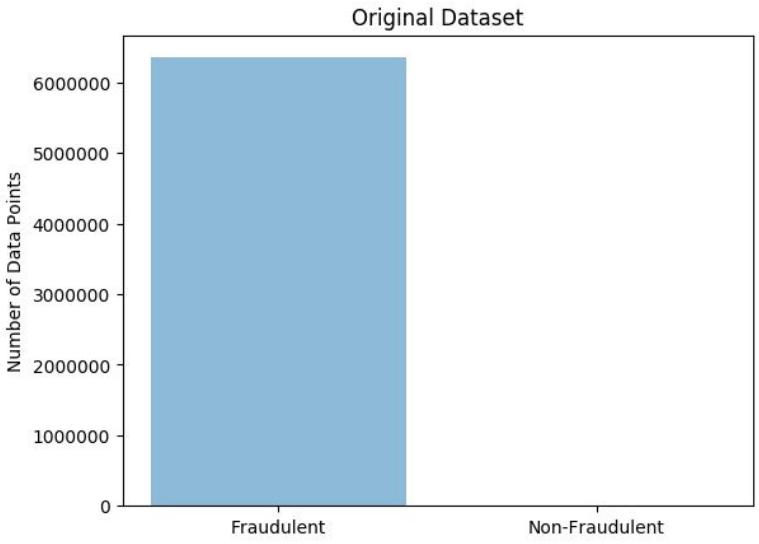
Given the task of fraud detection in financial transactions, we cannot afford a large number of false negatives. For this reason, we need to somehow balance classes. Two approaches for doing this can either be increasing the representation of the minority class or decreasing the representation of the majority class. One approach might result in overfitting, and the other might result in underfitting.

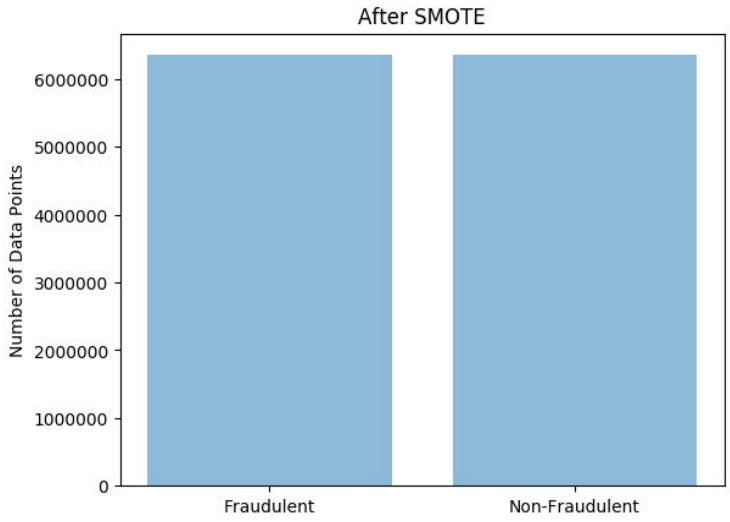
More specifically, one can decrease the representation of the majority class by performing undersampling on the data, and increase the representation of the minority class by performing oversampling.

Undersampling refers to randomly eliminating majority class data points to decrease the representation of the majority class in the dataset, thereby relatively increasing the representation of the minority class. This might result in underfitting, since there is a potential loss of important information in building a classifier. Oversampling refers to randomly replicating minority class data points in order to increase the representation of the minority class. However, this might result in overfitting of the minority class data points into the model.

Keeping this in mind, we used SMOTE (Synthetic Minority Oversampling Technique) algorithm. SMOTE makes sure overfitting does not happen due to minority oversampling by taking a random subset of the minority samples and synthesising new samples from them. This way we were able to get a class ratio of 1:1 in the dataset.

Class ratio visualisation before and after SMOTE:





Classification

Classification is an instance of supervised learning which, for a certain observation, identifies which one of the given categories/sub-populations it belongs to. In our problem, there are two possible categories of data -- non-fraudulent and fraudulent. Thus, to detect fraud in a given transaction, we decided to use a classifier. The output would be a 1, indicating a fraud, or a 0, indicating not-fraud. Although there are many types of classifiers, for the purposes of this project, we have used a random forest classifier.

Before deciding to use a random forest classifier, we tried using other classifiers, k-nearest neighbours and Naive Bayes. While Naive Bayes gave extremely poor accuracy -- owing to the fact that the features of this dataset are not independent of each other -- K-nearest neighbours classifier gave about 91% accuracy for k = 190. As we increased the value of k, the runtime and consumption of RAM kept increasing. After a certain point, it became infeasible to increase the value of k to improve results. Moreover, the confusion matrix had a very skewed distribution, implying that this was not a very practical classifier to use given the conditions.

Random forests, on the other hand, was much faster and efficient. It is known to work well with data of high dimensionality. Most of all, it is a strong classifier built using many uncorrelated weak classifiers (decision trees).

Before taking a look at the results of the random forest classifier on the training data, it is important to discuss the metrics we use to assess the performance of the classifier.

Metrics used to assess classification:

In this section, we first briefly define the metrics and then explain why we used these metrics.

- Accuracy: This metric quantifies the number of correctly classified observations out of all classified observations.
- Precision: This is a metric that quantifies what proportion of the number of examples classified as a certain class are correctly classified.
- Recall: This metric quantifies what proportion of examples from a class were correctly classified out of all examples from the class.
- F1- score: It is also a measure of a test's accuracy. This metric takes into account both precision and recall.
- Confusion matrix: This is a matrix that represents the number of correct and incorrect predictions with count values corresponding to each class. It looks something like this for a binary classifier:

-

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

where, in our case,
TN -- True Negatives: The examples that are classified are non-fraudulent and are actually non-fraudulent.
FP -- False Positives: The examples that are classified as fraudulent but are actually non-fraudulent.
FN -- False Negatives: The examples that are classified as non-fraudulent but are actually fraudulent.
TP -- True Positives: The examples that are classified as fraudulent and are actually fraudulent.

For a binary classifier, a good confusion matrix is one which has most of its density on its diagonal.

This project aims to propose a solution that identifies almost all fraudulent examples correctly. Given the seriousness of this problem, we cannot afford to have a large number of false negatives. Although accuracy and precision give us a measure of how much of the test data the model correctly classifies, they do not say much about the actual measure of false negatives or true positives with respect to all fraudulent examples. However, recall is an interesting metric that gives us an idea of "how many relevant items are selected." Since we are required to minimise the number of false negatives as much as possible, we should ideally obtain a recall (for the fraudulent class) close to 1.

The below is the best result - confusion matrix and classification report - for random forests with 50 trees and 80% of the dataset as training data we obtained:

```
[[1270907      42]
 [    341    1234]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270949
           1       0.97      0.78      0.87      1575

    accuracy                           1.00   1272524
   macro avg       0.98      0.89      0.93   1272524
weighted avg       1.00      1.00      1.00   1272524
```

The best recall for the fraudulent class we got was 0.78%.

New Frauds

One of the main challenges to this problem is the fact that the fraudulent transactions given in the training dataset might represent only certain types of frauds. Any given transaction might be a new type of fraud for which data is not present in sufficient quantities in the current dataset. Given also that fraudulent or invalid transactions are absolutely unacceptable, we have to make sure that all (or at least, most) of those cases are detected correctly. More precisely, we should ensure negligible false negatives.

The classifier has a recall of 0.78. This means that it classifies a fraudulent transaction as fraudulent 78% of the time. This means that there is a significant number of false negatives that can also be reduced.

Since all the non-fraudulent data points represent regular transactions, it is imperative that a fraudulent data point would differ from a non-fraudulent data point when plotted on the feature set. To use this difference to our benefit, we decided to use clustering which is the unsupervised analog of classification. This means that it tries to figure out the different classes present in the given data on its own and clusters them together. We have clustered all the nan-fraudulent data points expecting any fraudulent data point to appear as noise on the output of the clustering algorithm. We run anything classified as non-fraudulent by the classifier through the clustering algorithm.

Clustering

Clustering is a technique of unsupervised learning in which data is distributed into various clusters wherein data in a cluster has similar properties while data in different clusters are as different as possible. We tested out two clustering techniques in our model: K-means and DBSCAN.

The k-means algorithm takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intra cluster similarity is high but the inter cluster similarity is low. Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster's centroid or center of gravity.

While k-means is one of the most popular algorithms to use for clustering, we decided to use DBSCAN instead which is a density based model that works better with a large dataset with high dimensionality (the dataset we are using has 10 dimensions). The clusters created by DBSCAN need not be spherical unlike k-means.

DBSCAN involves the following two parameters:

1. Epsilon: Radius of a vicinity from the observing degree.
2. Minimum points (MinPts): Minimum variety of information degrees contained in such a vicinity.

DBSCAN classifies data into clusters and noise. Clusters consist of points present inside the cluster or core points and points present on the border of the cluster or border points. Noise contains data points that

cannot be fit into any cluster with the given parameters of Epsilon and MinPts and have different properties compared to the other clusters and thus cannot be fit into any.

Our model clusters all the non-fraudulent data from the training set and when a test data is passed through the output of the clustering algorithm, we check to see if it is within epsilon distance of any of the core points of the cluster. If it does not or is found to be present in noise, the point is taken to be fraudulent, if it can be present in a cluster, it is non fraudulent.

Final Model

Keeping everything in mind, we have built a system that is two layered. The first layer is a classifier classifies data points as fraudulent or non-fraudulent. If the first layer classifies a data point as fraudulent, the system labels it as fraudulent and returns. However, if the first layer does not classify a point as fraudulent or invalid, the point has to go through the second layer.

The second layer is based on clustering. The second layer uses a clustering algorithm that is built based on all the non-fraudulent data points given in the data set. Given a data point that enters Layer-2, if it is clustered as noise in comparison to the clusters built on the non-fraudulent transactions, it is labelled as fraudulent. If it is clustered with one of the existing clusters, it is labelled as non-fraudulent.

Final Results

We trained the classifier using 80% of the data points given to us. As expected, our model was able to give about 78% recall on the test data.

Since we did not have data points for new types of fraud, we could not test its performance on them. However, we tried to test our classification based on clustering model by clustering purely based on non-fraudulent points and then testing it on all fraudulent points. Since DBSCAN was run purely on non-fraudulent data points, we expected all these fraudulent data points to be in the “-1” cluster, which meant noise - or in other words, we expected them to not be part of any cluster (each cluster contained purely non-fraudulent points from the dataset).

Due to computational constraints, we could cluster only 6% of all non-fraudulent data, i.e., 5,00,000 non-fraudulent data points. The highest number of fraudulent data points we could run through the model was 100. This gave us about 5% accuracy.

Summary and Conclusion

Fraud detection is one of the biggest challenges faced by several companies in the financial domain. Machine learning tools (as discussed above) can be used to create a powerful solution for the same. The dataset for this problem was highly skewed since we were using real world data where the ratio between non-fraudulent and fraudulent transactions is very high. In order to solve this issue, we used the SMOTE

algorithm to synthesise data points that were fraudulent and trained our Random Forest Classifier to correctly classify transactions with 78% recall. In order to address the issue of false negatives and the possibility of new types of fraud, we ran the data point through a clustering algorithm to check the distance of the point from the non fraudulent clusters and if found to be more than a given threshold, we classified the point as fraudulent, if not, it was classified as non fraudulent. Since we were not able to train the cluster with all the data points and use the ideal value of epsilon due to the unavailability of resources, our accuracy while predicting the classes after clustering came out to be 5%.

Challenges Faced

One of the main challenges we were posed with was the idea of the possibility of new frauds. Our model was expected to be resistant against any type of fraud for which data is not available. At the same time, there was a limitation that classifiers imposed in terms of data required. Other challenges included the lack of enough computational resources. Google Colab offered only limited RAM and DBSCAN clustering on approximately 6 million points was extremely difficult.

Acknowledgement

We would like to thank Prof. Kothari, Chanda, and Dhruv for exposing us to the world of machine learning. It has been a great journey and we are very grateful. We would also like to thank our friends and family.

Bibliography:

1. Xu, Dongkuan, and Yingjie Tian. “A Comprehensive Survey of Clustering Algorithms.” *SpringerLink*, Springer Berlin Heidelberg, 12 Aug. 2015, https://link.springer.com/article/10.1007/s40745-015-0040-1.
2. Nidhi, Km Archana. “An Efficient and Scalable Density-Based Clustering Algorithm for Normalize Data.” *Procedia Computer Science*, Elsevier, 11 Aug. 2016, https://www.sciencedirect.com/science/article/pii/S1877050916315824.
3. Ali Huda and Kadhum Lubna, “K- Means Clustering Algorithm Applications in Data Mining and Pattern Recognition.” *ISSN (Online): 2319-7064.*
4. Ester Martin , Kriegel Hans-Peter ,Sander Jiirg, X Xiaowei. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. *Institute for Computer Science, University of Munich.*