Credit Card Fraud Detection using Machine Learning

Rohit Awate, Shreyas Nikam

https://github.com/ZeroNP/MLProjectCreditCardFraud

Abstract

The binary classification task of identifying fraudulent transactions from non-fraudulent ones involves identifying abnormal activities from past data. In this project, we plan to study and compare various machine learning techniques to analyze and identify these patterns. Only a minority of transactions are fraudulent, which poses a challenge of highly imbalanced data. We attempt to solve this problem and build robust models that can identify fraudulent transactions using logistic regression, decision trees and deep neural networks. All the files for this project can be found at [1]

1 Introduction

Detecting credit card fraud pattern in payment card transactions is a challenging problem because of the colossal amount of data generated by these transactions. It is impossible for a human analyst to detect fraudulent patterns owing to the large number of dimensions, real time updates and the large number of transactions. The integration of machine learning techniques in detecting credit card frauds has helped to efficiently detect frauds. Some events of credit card frauds include:

- The most common type of fraud belongs to theft or lost cards.
- A fake or a counterfeit card produced by a fraudster by imprinting the information on a card without the card holder's knowledge.
- Card interception during delivery.
- Online or telephonic fraud which is the absence of the card.

Due to the ever-increasing amount of data, the use of machine learning techniques is now widespread in the field of fraud detection. The ability of ML techniques to address the challenges and the real life impact makes it interesting for us to explore.

2 Problem Statement

Credit card fraud detection falls under the umbrella of classification ergo, classification refers to predictive modeling problems where a class label is predicted for a given example of input data. Here, the labels correspond to fraud and non-fraud transactions. This problem branches off from the anomaly detection problem, wherein anomaly detection is a step in data mining that identifies data points, events, and/or observations that deviate from a dataset's normal behavior. Anomalous data can indicate critical incidents, such as a technical glitch, or potential

[1] Project Repository: https://github.com/ZeroNP/MLProjectCreditCardFraud/

opportunities, for instance a change in consumer behavior. Here, the fraudulent transactions can be said to be anomalous, since they deviate from expected normal behaviour. A more concrete plan of our approach to the problem involves the following:

1. Data exploration and visualization, hypothesis testing and drawing observations from the visualizations.
2. Data preparation involving data cleaning, relabeling, categorization, scaling and transformations.
3. Encoding of feature vectors from categorical to numerical and/or ordinal vectors.
4. Handling class imbalance using resampling, undersampling and oversampling.

3 Technical Approach

3.1 Dataset

We plan to use a simulated credit card transaction dataset [1] containing legitimate and fraud transactions from the duration 1st Jan 2019 - 31st Dec 2020. It covers credit cards of 1000 customers making transactions with a pool of 800 merchants in various spending categories.

3.2 Class Imbalance

By far the biggest challenge we encountered was the imbalance between the number of fraudulent and non-fraudulent transactions. Out of the approximately 1.2M samples in our dataset, only 7506 are fraudulent, which is roughly only 0.57%. This makes for a few interesting problems. Firstly, there are not enough samples of fraud for any model to identify patterns and learn from. Secondly, the model's accuracy comes out strong given that it is able to classify the majority class i.e. non-fraudulent well. However, such models exhibited poor performance when classifying fraudulent transactions which is evident from the low precision, recall and F1 scores. An example can be seen below:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	542403
1	0.83	0.13	0.23	13316
accuracy			0.98	555719
macro avg	0.90	0.57	0.61	555719
weighted avg	0.98	0.98	0.97	555719

Figure 1: Example of good accuracy but poor recall and F1 scores

To tackle this problem, we experimented with the following approaches:

3.2.1 Assigning Class Weights

Our first solution was to assign class weights such that class 1 i.e. fraudulent would have far more weight than class 0. This allows us to artificially increase the influence of a certain class during the training process. `scikit-learn` makes this very easy to implement by letting us pass a dictionary mapping classes to their weights. We tried multiple combinations ranging from 1:2 to 1:100. While this resulted in an improvement in overall accuracy and related metrics such as precision, recall and F1, the model still performed poorly on test data.

3.2.2 Undersampling

Undersampling is the process of picking a few samples out of a large pool. We used a Python package called `imbalanced-learn` which provided a class called `RandomUnderSampler`. We used its default strategy of undersampling which is to reduce the number of samples of the majority class such that they are equal to the number of samples of the minority class. This meant that our dataset was reduced down to 7506 samples of **each** class for a total of roughly 15K samples. This is a significant reduction of the dataset from 1.2M samples. After training, the results were moderately favorable, but still performed poorly on the test set. We attributed this to the massive reduction in input data.

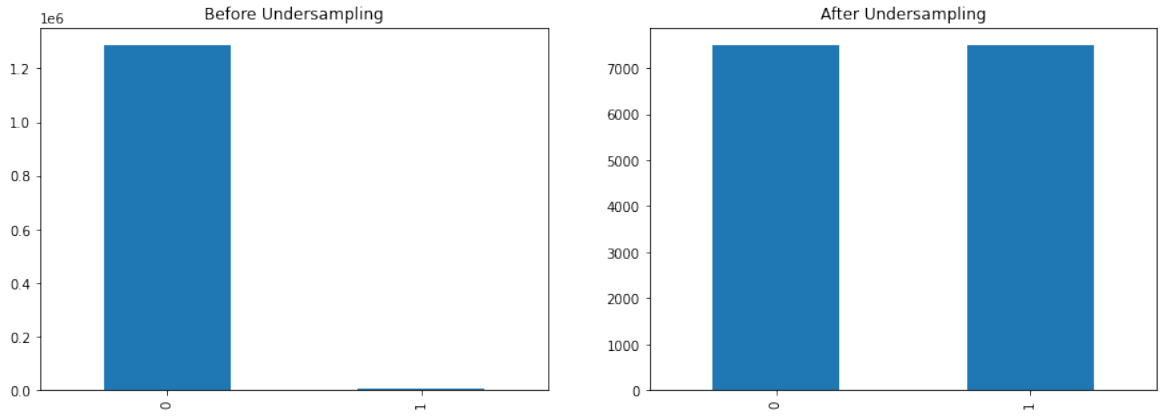


Figure 2: Undersampling on the dataset

3.2.3 Oversampling

Oversampling involves the artificial generation of samples of the minority class to achieve parity in the number of samples of each class. A variety of strategies are employed to achieve this. The simplest one is to randomly duplicate samples. More sophisticated methods such as SMOTE (Synthetic Minority Oversampling Technique) take a learning-based approach. Unsupervised techniques such as k-nearest neighbors are used to *generate* synthetic samples. We used `imbalanced-learn`'s implementation of SMOTE to oversample the fraudulent class. This produced the strongest results, both during training and testing.

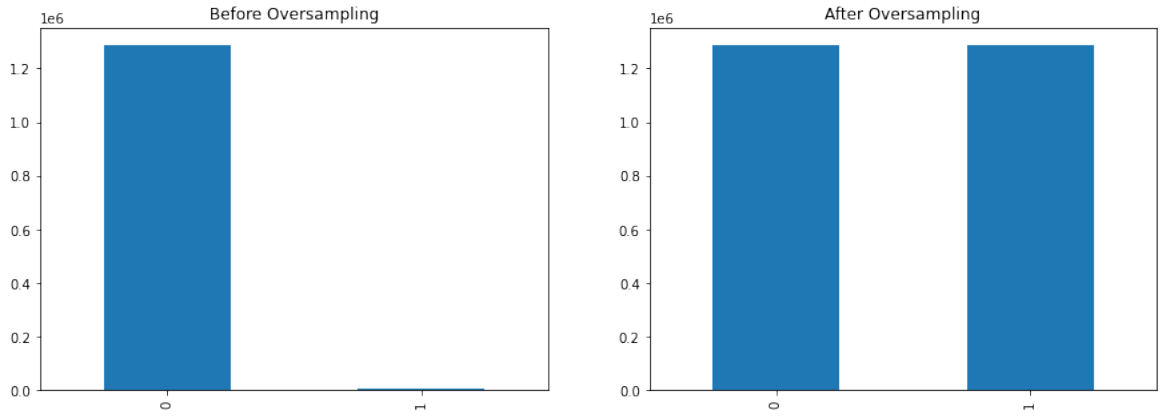


Figure 3: Oversampling on the dataset

3.2.4 Resampling

This is a mix of both oversampling and undersampling where the minority class gets oversampled while the majority class is undersampled to generate a balanced, consistent dataset. We experimented with `scikit-learn`'s implementation of resampling. This produced similarly strong results as oversampling.

Ultimately, we chose **resampling** as our strategy of choice for tackling class imbalance because it produced the strongest and most consistent results.

3.3 Feature Engineering

Initially, we performed the bare minimum data pre-processing required to feed the data into the model and get baseline results. It was quickly apparent that we would have to perform significant amount of feature engineering to get the most out of our data. We hypothesized and experimented with many engineered features and observed their correlation with the output. These are mentioned below:

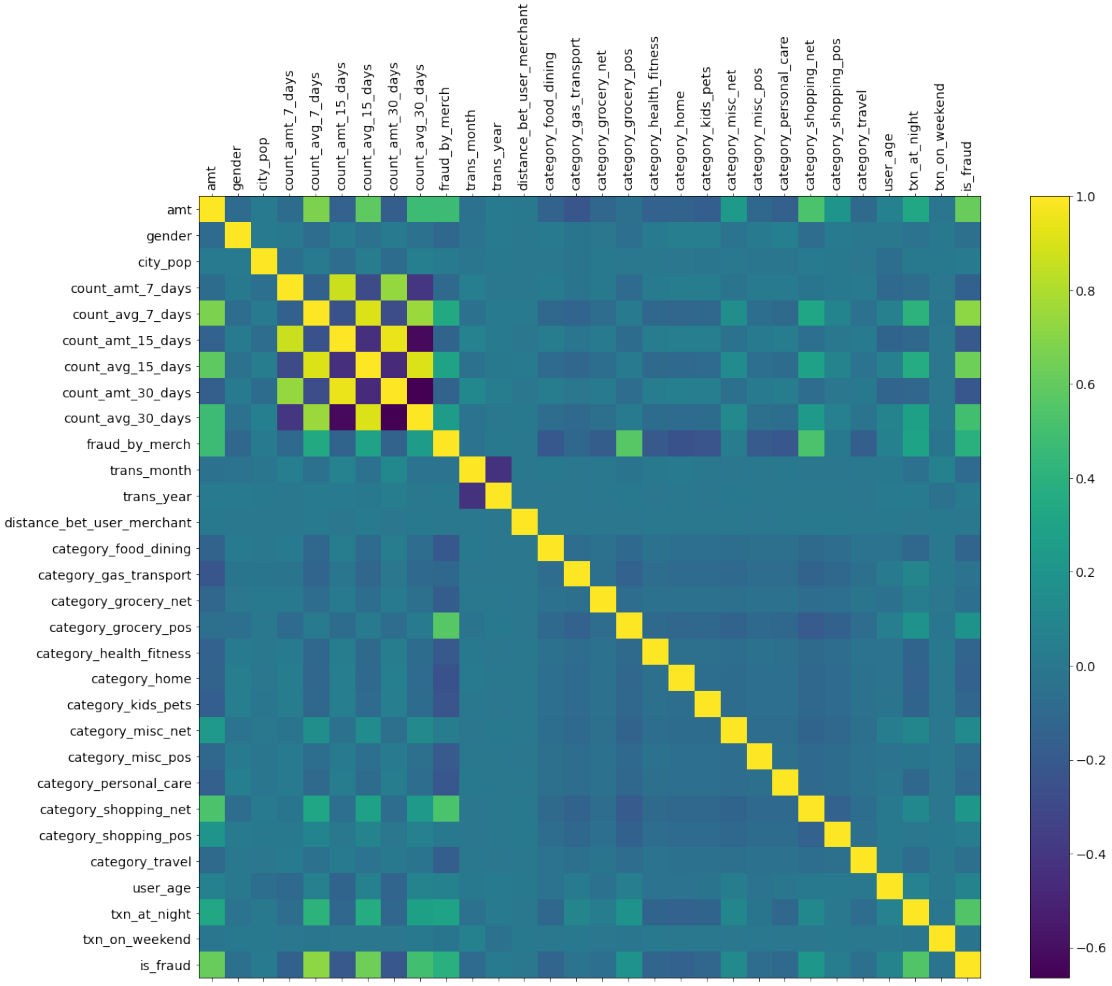


Figure 4: Correlation matrix

- **Transaction History**
  Here, we calculated the frequency of transactions and average amount spent on them over

a given period of time. We experimented with many window sizes such as 7, 15, 30, 60, and 100 days. Finally, we settled for 7, 15, and 30 days as our window sizes as they produced the best results. These metrics produced very strong correlation with output. A strong correlation can be observed in 4 between `count_avg_amt_x_days`, `count_amt_x_days` and `is_fraud`.

- **Transaction Time Classes**
  We found that more frauds happened on weekends and at night and thus we generated binary variables. These indicate if a certain transaction happened over a weekend or at night. A strong correlation can be observed in 4 between `txn_at_night` and `is_fraud`.
- **Fraud by Merchant**
  We observed that there were more frauds at certain merchants and thus we summed up the number of frauds at each merchant and assigned it as a score. A strong correlation can be observed in 4 between `fraud_by_merchant` and `is_fraud`.
- **Distance between user and merchant**
  We calculated the haversine distance between the location of the user and the merchant. Our hypothesis was that fraudulent transactions would take place away from the merchant. While this did not present a particularly strong correlation, we still used it as a feature.

Through our exploratory data analysis, we observed that certain features showcased a strong correlation with the output. `amt`, transaction categories of `category_grocery_net`, `category_shopping_net` and `category_misc_net` are a few examples of this.

4 Experimental Results

4.1 Algorithms Performance

4.1.1 Deep Neural Network

Table 1 gives the overview of the performances of the various models. It is evident that the fine-tuned Deep Neural Network gives the best performance since it is able to capture complex patterns among the data. Following are the details of our network architecture:

```
FraudDetectionDNN(
  (fc1): Linear(in_features=29, out_features=29, bias=True)
  (fc2): Linear(in_features=29, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=512, bias=True)
  (fc4): Linear(in_features=512, out_features=64, bias=True)
  (relu): ReLU()
  (fc5): Linear(in_features=64, out_features=1, bias=True)
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.2, inplace=False)
)
```

- Batch size: 64
- Optimizer: Stochastic Gradient Descent
- Learning Rate: 0.07

The loss of the deep neural network was about 0.067 with 98% overall accuracy along with the highest and most consistent F1-score, precision and recall amongst all models.

4.1.2 Logistic Regression

The default parameters for logistic regression worked unexpectedly well where the model uses L2-regularization. We have also attempted to change the solvers from 'saga', 'liblibear' and 'lbfgs' during the trial phases. Refer table 1 for a detailed breakdown of its performance.

4.1.3 Decision Trees and Random Forest

Decision Trees and Random Forests gave a training accuracy of 100% meaning that it was successfully able to classify all the samples correctly, with high F1-score and precision/recall values. However, it could not perform as well while testing, which suggests that the model overfit on the training set. This was supported by the cross validation.

4.1.4 SVM

SVM took unfeasibly long to train. This can be attributed to the fact that the training time of Support Vector Machines is directly proportional to the cube of the number of samples i.e. $O(n^3)$. Since the number of samples in our training set was of the order of a million, $n \sim 1.2e{+}6$, we were not able to evaluate the performance of support vector machines on the chosen dataset.

4.1.5 Naive Bayes

Naive Bayes classifiers make naive assumptions about the conditional independence between features, and apply Bayes' theorem on them. The performance from Naive Bayes was surprisingly good, given that it is not considered to be a particularly robust algorithm.

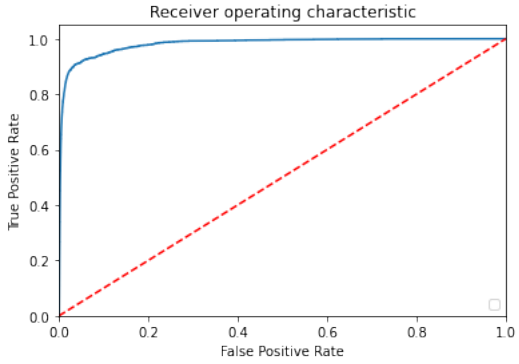
	Precision	**Recall**	**F1-score**	**Accuracy**
Logistic Regression	Fraud: 0.96 Not Fraud: 0.91	Fraud: 0.91 Not Fraud: 0.95	Fraud: 0.93 Not Fraud: 0.93	Training: 94% Testing: 93%
Decision Trees	Fraud: 1.00 Not Fraud: 0.78	Fraud: 0.82 Not Fraud: 1.00	Fraud: 0.90 Not Fraud: 0.87	Training: 100% Testing: 89%
Random Forest	Fraud: 1.00 Not Fraud: 0.79	Fraud: 0.83 Not Fraud: 1.00	Fraud: 0.90 Not Fraud: 0.88	Training: 100% Testing: 89%
Deep Neural Network	Fraud: 0.98 Not Fraud: 0.98	Fraud: 0.98 Not Fraud: 0.98	Fraud: 0.98 Not Fraud: 0.98	Training: 98% Testing: 98%
Naive Bayes	Fraud: 0.95 Not Fraud: 0.80	Fraud: 0.83 Not Fraud: 0.94	Fraud: 0.88 Not Fraud: 0.87	Training: 88% Testing: 87%
Support Vector Machines	SVM training proved to be infeasible since training takes time proportional to cube of the number of samples $O(n^3)$, where n is of the order of 1.2M			

Table 1: Comparative Analysis of the performance of various algorithms.

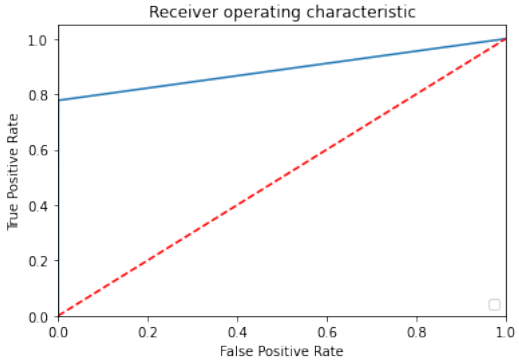
5 Summary & Discussion

This section involves a high-level discussion of takeaways, findings and other observations.

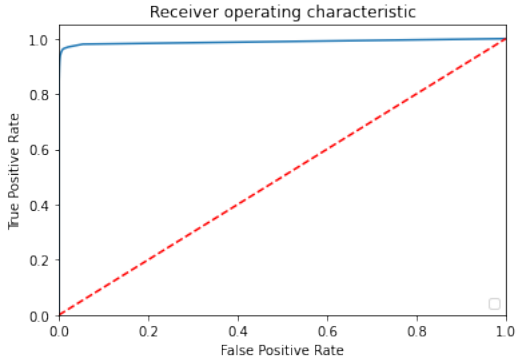
- Oversampling works well for dealing with class imbalance.



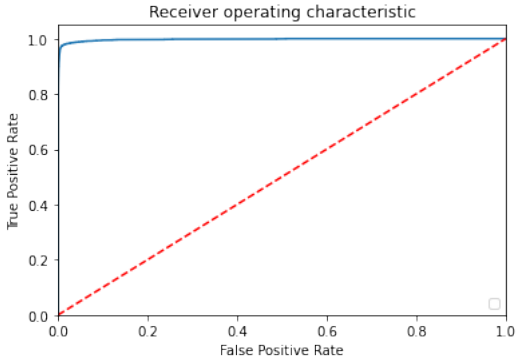
(a) Logistic Regression



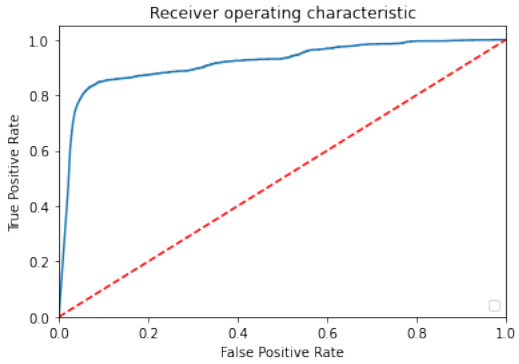
(b) Decision Tree Classifier



(c) Random Forest Classifier



(d) Deep Neural Network



(e) Naive Bayes Classifier

Figure 5: ROC curves of the models

- Scaling the data through normalization and standardization gave satisfactorily better results than the original features. Thus, scaling is an indispensable part of any machine learning problem where the range is large.
- Precision, recall and F1-score are far better metrics for determining the quality of a model built on imbalanced data as compared to accuracy and loss.
- Feature engineering is pivotal to properly derive and select features that actually influence the outcome and to get the most out of the data. This requires extensive experimentation, hypothesizing and domain knowledge.
- Hyper-parameter tuning, cross-validation and input scaling are critical to achieve the best possible performance from the model.

- Support vector machines may be practically infeasible to train with extremely large datasets such as the one used here.

6 Contributions

The work for the project was mostly evenly divided amongst the two of us. We performed exploratory data analysis together. Rohit was primarily responsible for Feature Engineering, building and training the Logistic Regression, Decision Trees, Random Forest Classifier and Support Vector Machines. Shreyas was responsible for dealing with class imbalances, building the Deep Neural Model and the Naive Bayes Classifier. The work for the proposal, presentation and the report was also evenly distributed between the both of the members.

References

[1] https://www.kaggle.com/datasets/kartik2112/fraud-detection

[2] Haseeb Ali, Mohd Najib Mohd Salleh, Kashif Hussain, Arshad Ahmad, Ayaz Ullah, Arshad Muhammad, Rashid Naseem, and Muzammil Khan. A review on data preprocessing methods for class imbalance problem. International Journal of Engineering Technology, 8:390–397, 2019.

[3] Christopher M Bishop. Pattern recognition and machine learning. Springer, 2006.

[4] Fabrizio Carcillo. Beyond Supervised Learning in Credit Card Fraud Detection: A Dive into Semi-supervised and Distributed Learning. Université libre de Bruxelles, 2018.

[5] Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.

[6] Fabrizio Carcillo, Yann-Aël Le Borgne, Olivier Caelen, Yacine Kessaci, Frédéric Oblé, and Gianluca Bontempi. Combining unsupervised and supervised learning in credit card fraud detection. Information Sciences, 2019.

[7] Akhilesh Gupta, Nesime Tatbul, Ryan Marcus, Shengtian Zhou, Insup Lee, and Justin Gottschlich. Class-weighted evaluation metrics for imbalanced data classification. arXiv preprint arXiv:2010.05995, 2020.