Analysis of Anomalous Transactions Using Automated Predictive Techniques

Mouhssine Rifaki[1]

[1] ***PARIS DAUPHINE UNIVERSITY - PSL***

mouhssine.rifaki@psl.eu

Abstract

Research over the years, including the work by (Kou et al. 2004), has shown that automated methods for fraud detection hold a lot of potential. Inspired by this, I worked on creating a machine learning model to identify fraudulent transactions as they happen, with the goal of allowing immediate responses. I tried to analyze a dataset that contained hundreds of thousands of transaction records, and included diverse details such as timestamps, monetary amounts, and geographic information. It was quite detailed, but also somewhat overwhelming to work with at first. One of the more difficult aspects I encountered was the imbalance in the dataset, fraudulent transactions were only a tiny fraction of the total.

To address the underrepresentation of fraudulent cases in the dataset, I applied fraud oversampling techniques to give these cases greater prominence during training. Alongside this, I incorporated a focal loss function, which helped the model concentrate more effectively on the challenging, less frequent examples by penalizing uncertain predictions more strongly. This combination proved to be useful in improving the model's ability to detect fraud. But managing the class imbalance required significant effort, as it added complexity to the training process. The good news is that the model achieved a test accuracy of 83% overall and an accuracy of 88% specifically for detecting fraudulent transactions. While these results were promising, I feel there is still room for improvement, and further fine-tuning could enhance the model's performance even more in future iterations.

1 Introduction

Under PwC's Global Economic Crime Survey, 36% of organizations reported experiencing economic crime in 2016 (White, Anderson, and Lavion 2016). As fraud tactics continue to become more sophisticated and evolve, many instances of economic crime likely go unnoticed, indicating that the actual financial impact could be much greater than reported. Compounding this issue, the average profit generated from these crimes has been steadily rising (White, Anderson, and Lavion 2016).

Significant events such as the Equifax data breach and cases of international corporate espionage reveal how inconsistent and insufficient many organizations' fraud prevention strategies are. Alarmingly, most companies still uncover fraud by chance rather than through structured detection methods. Some research has shown that automated fraud detection techniques have considerable potential to address these gaps (Kou et al. 2004).

I tried to implement some machine learning methods to design and train a model capable of detecting fraudulent transactions in real time, allowing for immediate responses. For this purpose, I worked with a dataset which included several hundred thousand transactions with temporal, financial, and geographical attributes. The model was trained using a single-layer neural network, and to address the issue of imbalanced data where fraudulent transactions were far less common, I incorporated a focal loss function. This method ensured that the model is more focused on the underrepresented fraudulent cases.

2 Dataset and Features

The dataset I worked with includes 15 features drawn from 290,382 examples of financial transactions. A detailed explanation of these features can be found in Table 2.1. Since the dataset contains both discrete and continuous parameters, I used an encoder from the SciKit Python library to convert the discrete features into continuous ones. This step was important to ensure that all features in the dataset were in a consistent format for analysis (Pedregosa et al. 2011).

Table 2.1: *Features included in the dataset*

Feature	Description
bookingdate	Timestamp when the chargeback was reported
issuercountrycode	Country where the card was issued
txvariantcode	Card type used (e.g., subbrand of VISA or MasterCard)
bin	Card issuer identifier
amount	Transaction amount
currencycode	Currency code
shoppercountrycode	Country of shopper's IP address
shopperinteraction	Indicates whether the transaction was online or a subscription
cardverificationresponsesupplied	Indicates if the shopper provided a CVC/CVV code
cvcresponsecode	Validation result of the provided CVC/CVV code
creationdate	Date of the transaction
accountcode	Merchant's webshop identifier
mail_id	Shopper's email address
ip_id	Shopper's IP address
card_id	Shopper's card number

The temporal variables like transaction timestamps, are broken down into six separate components: year, month, date, hour, minute, and second. I really wanted to ensure consistency across all features, so I made sure that each variable is independently scaled using a Python library. This actually prevents features with larger scales or arbitrary magnitudes, like encoded variables, from disproportionately influencing the objective function.

The feature 'bookingdate', which indicates the time when a fraudulent transaction is reimbursed, is treated as a proxy for the dependent variable. And since this information is only available after the fact, it is removed from the dataset to allow the model to function in real-time fraud detection scenarios.

The dataset contains three class labels: 'Chargeback' (fraud), 'Settled' (non-fraud), and 'Refused' (which could indicate either fraud or insufficient funds). In my analysis, I concentrated on the 'Chargeback' and 'Settled' classes, leaving out the Refused transactions to make the process more manageable. To identify which features were most relevant for detecting fraud, I used histogram analyses and experimented with forward feature selection methods. For this, I relied on simple classifiers like Naive Bayes and Logistic Regression to test the effectiveness of different features. Four features stood out as important: 'bin', 'amount', cvcresponsecode, and 'mail_id'.

The difficult aspects of working with the dataset is its imbalance. Of the 290,382 transactions included in the data, 53,346 were labeled as Refused and were removed from this part of the analysis. This left 236,691 transactions marked as Settled and just 345 confirmed as fraudulent. With a fraud rate of only 0.15%, the data was heavily skewed, which added complexity to the modeling process and highlighted the importance of addressing this imbalance.

3 Method

At the start of my analysis, I tested several simple classifiers using the SciKit-learn Python library to figure out which type of model might be most effective for this task (Pedregosa et al. 2011). When I ran the Naive Bayes classifier, it became clear that its assumption of conditional independence among features did not align well with my dataset. Similarly, Support Vector Machines with a linear kernel struggled to deal with the dataset's imbalance because they lacked a way to apply weighted predictions. An unweighted Logistic Classifier also had trouble with this challenge, but by reweighting fraud cases more heavily and reducing the impact of non-fraud cases, I was able to get some moderately encouraging results.

Following my initial experiments, I decided to continue with a single-layer Neural Network. For the hidden layer, I used ReLU as the activation function, while the output layer applied a sigmoid function. Among the approaches I had tested up to that point, this setup appeared to hold the most promise for addressing the problem. It was clear to me that the Neural Network implementation in sklearn had limitations regarding the customization of the architecture and loss function. Consequently, I have switched to TensorFlow, which provided more flexibility I needed to create a more tailored solution (Abadi et al. 2015). I built on what I had learned from the weighted Logistic Classifier so I established a baseline model. For this, I implemented a single-layer Neural Network that utilized a weighted cross-entropy loss function, which was designed to handle the dataset's class imbalance.

$$WCE(y, \hat{y}) = -w_f y \log(\hat{y}) - w_{nf}(1-y)\log(1-\hat{y}), \tag{3.1}$$

$$\mathscr{L} = \sum_{i=1}^{n} WCE(\hat{y}^{(i)}, y^{(i)}), \tag{3.2}$$

where $\hat{y} \in [0, 1]$ represents the model's prediction, and $y \in \{0, 1\}$ is the label. The weights w_f and w_{nf} for fraud and non-fraud cases, respectively, are defined as:

$$w_f = \frac{|\{y \in Y : y = \text{non-fraud}\}|}{|Y|}, \tag{3.3}$$

$$w_{nf} = \frac{|\{y \in Y : y = \text{fraud}\}|}{|Y|}. \tag{3.4}$$

The ReLU and sigmoid activation functions are defined as:

$$g(z) = \max(z, 0) \quad \text{(ReLU)}, \tag{3.5}$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \text{(sigmoid)}. \tag{3.6}$$

I used mini-batch gradient descent to train the model. To initialize the weights, I randomly drew values from a standard normal distribution and set all the biases to zero. Before running the training process, I shuffled the dataset and broke it into smaller groups, or batches, each containing B examples. These batches were then passed through the network to generate predictions. After processing each batch, I updated the parameters θ, which include the weights and biases, by applying the gradient descent rule:

$$\mathscr{L}_{\text{MB}} = \frac{1}{B} \sum_{i=1}^{B} WCE(y^{(i)}, \hat{y}^{(i)}), \tag{3.7}$$

$$\theta := \theta - \alpha \nabla_\theta \mathscr{L}_{\text{MB}}. \tag{3.8}$$

Challenges with Class Imbalance

The baseline model didn't perform as well as expected, mainly because of the heavily skewed dataset. With only 0.15% of transactions being fraudulent, many of the mini-batches ended up containing few or even no examples of fraud at all. This created a noticeable bias toward predicting non-fraudulent transactions, and the weighted loss function wasn't enough to fully compensate for it. To tackle this issue, I experimented with alternative sampling strategies to improve the model's balance.

Batch Sampling Strategies

- **Weighted Sampling**: Batch Sampling Strategies
  The imbalance in my dataset required a different approach, so I tried weighted sampling first. For this, I assigned weights to each transaction depending on whether it was labeled as fraud or not. I then normalized these weights so that both fraud and non-fraud examples would have about the same representation in each batch. By doing this, I could rely on a standard cross-entropy loss function without needing additional class weight adjustments. Effectively, I set the weights for both fraud and non-fraud classes to 1 ($w_f = w_{nf} = 1$).

- **Proportional Sampling**:
  I also tested proportional sampling as another option. In this case, I sorted the data by labels and built batches with a fixed fraction f of fraud cases. Adjusting f gave me more control over how much attention the model paid to fraud examples in each batch, which seemed helpful for fine-tuning the model's performance during training.

Regularization and Overfitting

I want to point out that I encountered an issue with overfitting while trying to balance the dataset. The model seemed to perform noticeably better on the training data compared to the validation set. This made it clear that the model was picking up patterns unique to the training data rather than learning to generalize effectively. This was especially problematic because I had artificially increased the number of fraud examples. To address this, I added a regularization term to help the model avoid depending too much on specific patterns in the data. While this reduced the overfitting to some extent, it wasn't a perfect solution, and there's likely room for further improvement:

$$\mathscr{L}_{\text{reg}} = \mathscr{L}_{\text{MB}} + \lambda \left(||W_1||_2^2 + ||W_2||_2^2 \right), \tag{3.9}$$

where W_1 and W_2 are the weights of the hidden and output layers, respectively, and λ is the regularization parameter. Gradient descent updates for W_1 and W_2 were modified as:

$$W_{1,2} := (1 - 2\alpha\lambda) W_{1,2} - \alpha \nabla_{W_{1,2}} \mathscr{L}_{\text{reg}}, \tag{3.10}$$

with bias updates remaining unchanged.

Focal Loss for Ambiguous Cases

Many non-fraud examples were easily classified, rendering them less useful for training. Inspired by (Lin et al. 2017), we incorporated focal loss, which emphasizes ambiguous examples:

$$FL(y, \hat{y}) = -(1-\hat{y})^\gamma f y \log(\hat{y}) - \hat{y}^\gamma (1-f)(1-y) \log(1-\hat{y}), \tag{3.11}$$

where $\gamma \geq 0$ controls the focus on uncertain predictions. As $\hat{y}$ approaches 0 or 1, the loss decreases, while it peaks near $\hat{y} = 0.5$, where model confidence is lowest.

Takeaway

Trying to manage the imbalance in the dataset turned out to be one of the toughest challenges I faced. Although strategies like focal loss and balanced sampling showed improvements, applying them wasn't as simple as it seemed. I had to experiment a lot and carefully adjust the parameters to prevent overfitting while still ensuring that the model could handle well new and unseen data.

4 Results and Discussion

4.1 Model Evaluation and Optimization

I focused on three key metrics to evaluate how well the model performed: overall accuracy, fraud accuracy, and non-fraud accuracy. I think these were particularly important because mistakes in fraud detection, like false positives or false negatives, could lead to serious financial losses. I split the dataset into three parts: 70% for training, 15% for development, and 15% for testing. I used cross-validation to calculate the metrics, and Table 4.1 presents a summary of the results for each tested model.

Table 4.1: *Training and dev performance per model.*

	Train accuracy			Dev accuracy		
Model	**Overall**	**Fraud**	**Non Fraud**	**Overall**	**Fraud**	**Non Fraud**
Naive Bayes	90	62	90	69	90	69
Weighted Logistic Classifier	82	89	82	82	80	82
SVM	100	0	100	100	0	100
NN with unweighted cross-entropy	60	100	60	63	92	63
Baseline NN (weighted cross-entropy)	82	90	82	83	71	83
Baseline with f-sampling	83	89	83	81	70	81
Baseline with regularization	71	95	71	70	96	70
Baseline with feature selection	82	94	82	80	78	80
Baseline with focal loss, f-sampling, regularization and feature selection	83	88	83	86	83	86

During testing, I noticed that the Naive Bayes classifier was inconsistent. Its performance on the training set was significantly different from its performance on the development set, which made it clear that the assumption of conditional independence among features didn't work well for this dataset. The Weighted Logistic Classifier was more consistent overall, but I still saw some overfitting, as shown by differences in fraud accuracy between the training and development sets.

Other models, like SVM and unweighted Logistic Classifiers, performed poorly. They often classified almost all transactions as non-fraud, which highlighted just how important it was to address the dataset's imbalance. The baseline Neural Network, which used cross-entropy loss, had similar issues. It focused mostly on non-fraud cases because fraud examples were so rare in the dataset. However, f-sampling proved helpful in reducing these imbalances and improving the model's accuracy across the board.

Regularization was also an important factor in reducing overfitting. It helped bring the fraud accuracy for the training and development sets closer together. Feature selection further enhanced the model's performance. By combining it with focal loss, I was able to achieve better results, including higher accuracy across all datasets. In my case, the focal loss function worked by penalizing false positives and false negatives more heavily, while also limiting the influence of overly confident predictions on the loss calculation.

Table 4.2: *Final values for model parameters.*

Parameter	**Value**
Batch size	1000
Tolerance	0.01
Learning rate	5.0
λ	0.001
γ	2.0
f	0.45
Number of hidden units	500

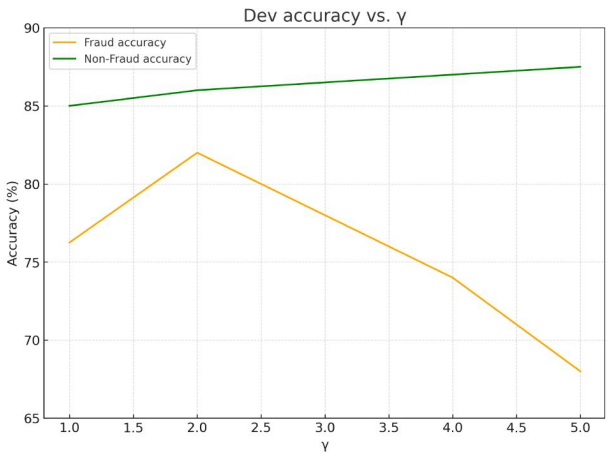

Figure 1: *Optimization plot for γ.*

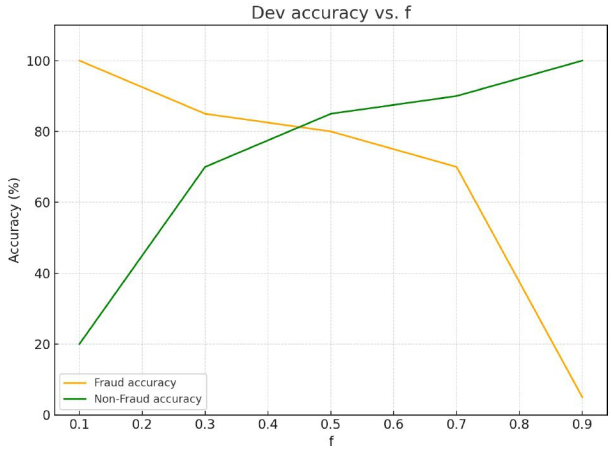

Figure 2: *Optimization plot for f.*

5 Conclusion

The process of tuning the model parameters, detailed in Table 4.1, involved varying one parameter at a time while keeping other parameters fixed. To make sure we have reliable results, I averaged performance across several runs for each parameter configuration.

Despite this approach being relatively simple, I think it provided useful insights into the influence of individual parameters. For instance, Figures 1 and 2 illustrate how γ and f affected the model's performance. f proved to be particularly an important parameter for balancing fraud and non-fraud predictions, which had a non-negligible impact on the model's overall effectiveness.

References

Abadi, Martin et al. (2015). *Tensorflow: Large-Scale Learning on Heterogeneous Systems*. URL: `https://www.tensorflow.org/`.

Kou, Y. et al. (2004). "Survey of Fraud Detection Techniques". In: *International Conference on Networking, Sensing, and Control*. Taipei: IEEE, pp. 749–754.

Lin, T.-Y. et al. (2017). "Focal Loss for Dense Object Detection". In: *ICCV*. Vol. 3.

Pedregosa, F. et al. (2011). "SciKit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

White, T., M. Anderson, and D. Lavion (2016). *Global Economic Crime Survey 2016*. Tech. rep. Price Waterhouse Coopers.