Detecting Fraud in Financial Payments

Noël Dutrée and Dirk Hofland

STANFORD UNIVERSITY

in collaboration with Maikel Lobbezoo *

December 15, 2017

ABSTRACT

Recent studies have shown the potential of automatised fraud detection techniques [1]. Therefore, we employed machine learning methods in order to develop and train a model for real-time detection of fraudulent transactions so that these may be acted upon instantaneously. Using a dataset containing various features, including temporal, financial, and geographical attributes of several hundred thousand transactions provided by Adyen, a global payments firm, we train a single-layer neural network using fraud oversampling and a focal loss-function. The latter penalizes uncertain predictions in order to accommodate the dataset's relative sparsity of fraudulent transactions. Despite this sparsity, we manage to achieve an overall test accuracy of 83% and a fraud detection test accuracy of 88%.

*Head of Growth at Adyen

1. Introduction

Under PwC's Global Economic Crime Survey, 36% of the organizations surveyed reported being victimized by economic crime in 2016 [2]. As approaches to fraud become increasingly sophisticated, more incidents of economic crime go unnoticed so that the actual financial impact of economic crime most likely is far greater. To make matters worse, the average profit obtained from such crimes is also on the rise [2]. One only has to follow recent events at Equifax or any of the ubiquitous instances of alleged international corporate espionage to realize how haphazard and insufficient many companies' fraud prevention strategies are. Most still detect fraud predominantly by accident rather than as a result of comprehensive and systematic fraud prevention policies.

Recent studies have shown the potential of automatised fraud detection techniques [1]. Therefore, we employed machine learning methods in order to develop and train a model for real-time detection of fraudulent transactions so that these may be acted upon instantaneously. Using a dataset containing various features, including temporal, financial, and geographical attributes of several hundred thousand transactions provided by Adyen, a global payments firm, we train a single-layer neural network using a focal loss-function.

2. Dataset and Features

Our chosen dataset was provided by Adyen, a global payments firm to which we have a prior affiliation. It contains fifteen features for 290,382 examples of financial payments. Each feature and its meaning is described in Table 2.1.

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 (subbrand of VISA or MasterCard)
bin	Card issuer identifier
amount	Transaction amount
currencycode	Currency code
shoppercountrycode	Country of shopper IP address
shopperinteraction	Online transaction or monthly subscription
cardverificationresponsesupplied	Did the shopper provide CVC/CVV code?
cvcresponsecode	Validation result of provided CVC/CVV code
creationdate	Date of transaction
accountcode	Merchant's webshop
mail_id	Email address
ip_id	IP address
card_id	Card number

Since the features consist of both discrete and continuous parameters we employ an encoder from the SciKit Python library to map the discrete features to continuous variables in order to have a set of exclusively continuous features [3]. Temporal variables such as transaction timestamps have moreover been split up into six separate variables representing that timestamp's year, month, date, hour, minute, and second.

We independently scale each feature using the same Python library so as to avoid the arbitrary scales and magnitudes of certain encoded variables from dominating the objective function.

As the bookingdate (representing the moment at which a fraudulent transaction is reimbursed to an account) is only known in hindsight and thereby functions as a direct proxy for our dependent variable, we remove it from our dataset since we aim for a model that can be used to detect fraud in real-time. The dataset moreover contains three class labels: 'Chargeback' (fraud), 'Settled' (non-fraud), and 'Refused' (could be fraud, but could also indicate insufficient funds). Due to time limitations we focus solely on the fraud (1) and non-fraud (0) classes and consequently remove all 'Refused' transactions from our dataset. Finally, using a combination of histogram analyses and forward-search based on results obtained from simple classifiers such as Naive Bayes and Logistic Regression, we select an

optimal set of four features with the most explanatory power: 'bin', 'amount', 'cvcresponsecode', and 'mail_id'.

It should be noted that the heavily skewed nature of our dataset presents a significant obstacle. Of the 290,382 transactions, 53,346 are labeled as 'Refused' and therefore omitted from this study. Only 345 transactions are unambiguously known to be fraudulent, implying that of our effective dataset (excluding 'Refused' transactions) only 0.15% of transactions is fraudulent with the remaining 236,691 marked as 'Settled'.

3. METHODS

We start by exploring a variety of simple classifiers using the SciKit-learn Python library in order to determine which type of model might prove successful [3]. The results produced by a Naive Bayes Classifier show that the assumption that the input features are conditionally independent is clearly unwarranted for our dataset. Given its inability to weight predictions, the highly skewed nature of our dataset also disqualifies using a Support Vector Machine with a linear kernel. We experienced similar problems with an unweighted Logistic Classifier, although overweighting and underweighting fraud and non-fraud examples, respectively, did result in a moderately promising outcome. Finally, we implement a simple single-layer Neural Network using a ReLU and sigmoid activation function for the hidden and output layer respectively, the results of which suggested such a model might prove most successful.

Given that the SciKit-learn Neural Network implementation does not allow users to customise the network's architecture and loss-function, we employ the Tensorflow library in order to further develop our own customised implementation [4]. As the weighted Logistic Classifier achieved moderately good results, a natural baseline model is a single-layer Neural Network using the weighted cross-entropy loss-function, which is defined as

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

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

with $\hat{y} \in [0,1]$ as the prediction of our model and $y \in \{0,1\}$ as the label. Here, w_{nf} and w_f, the weights for non-fraud and fraud cases, respectively, are defined as:

$$w_f = \frac{|\{y \in Y : y = \text{non-fraud}\}|}{|Y|} = \frac{|\{y \in Y : y = 0\}|}{|Y|} \tag{3.3}$$

$$w_{nf} = \frac{|\{y \in Y : y = \text{fraud}\}|}{|Y|} = \frac{|\{y \in Y : y = 1\}|}{|Y|} \tag{3.4}$$

We again used the ReLU and sigmoid activation functions for the hidden and output layers respectively, which are defined as:

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

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

We train our model using mini-batch gradient descent. That is, we initialise our weights by sampling from the standard normal distribution and initialise all biases to zero. We shuffle our training set, split it in approximately equally sized batches of size B, and then feed each mini-batch to the network in order to compute the predictions of the network. For each batch, we then update all parameters θ (the weights and biases of the model's hidden and output layers) using the mini-batch gradient descent update function:

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

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

Problematically, the performance of this baseline model is somewhat disappointing with a dataset as skewed as ours. A natural next step, as corroborated by the literature, is therefore to re-examine our sampling procedure [5, 6, 7]. Given the minute proportion of fraud examples in our dataset (0.15%),

picking a large batch size, let alone a relatively small one as is common when implementing mini-batch gradient descent, can result in batches having none or very few fraud examples. This in turn gives rise to a significant bias towards non-fraud examples that even the skewed weights defined in equations (3.3) and (3.4) cannot remedy. We consequently explore two alternative methods of generating batches from the training set.

Under our first strategy, we generate batches of a fixed size by randomly and independently sampling complete batches from the training set. However, the probability of an example being sampled is not uniform, but is instead defined as the unit-normalised weight of the corresponding label. That is, if we assign to every example the weight corresponding to its label, then we obtain an array which we can subsequently normalise to a unit-vector in order to obtain the weight of an example (or label). Note how this approach ensures that in expectation the share of fraud and non-fraud examples in each batch is 50%, so that the resulting model is expected to perform best under unweighted cross-entropy loss, which is equivalent to using the weighted cross-entropy loss-function as defined in equation (3.1) with $w_f = w_{nf} = 1$.

An alternative, preferable strategy is to sort the training set by label and generate batches by sampling a fixed proportion of examples from either set. Letting f be the proportion of fraud examples in each batch, we accordingly set $w_f = f$ and $w_{nf} = 1 - f$. Note that when we set $f = 0.5$ this second strategy is in expectation equivalent with the former, yet when $f \neq 0.5$ it is not. We therefore find that this latter approach is much more convenient as the meta-parameter f provides us with more control over the model's performance and can prove to be very useful in calibrating our model.

It is evident, however, that by artificially increasing the number of fraud examples in our training set this model inevitably overfits to these examples. This can result in a large performance gap in fraud classification accuracy between the training and dev set. It is therefore imperative to also include a regularization term in our loss-function:

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

where W_1 and W_2 are the weights of the hidden and output layers, respectively, and λ is the regularization parameter. We modify the gradient descent update functions for W_1 and W_2 as

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

but leave the update functions for the biases unchanged.

In order to further boost the performance, we explored a number of other changes to our model, such as adding extra hidden layers to our neural network, varying the available meta-parameters, applying polynomial expansion to our input features and using different optimisers, all to little effect as most, if not all, of these changes do not address the underlying problem of the strong imbalance in our dataset. In using this regularised model, however, it becomes evident that the vast majority of non-fraud examples are easily classified as such, and are therefore effectively superfluous to training our model. Crucially, this observation is not incorporated into our choice of weights, which remain constant for each example [8]. Yet in order to improve our model's performance, we are incentivised to overweight examples that are to some extent ambiguous to categorise, as it is precisely these examples that provide the most insight into the defining characteristics of either class. Following [8] we consequently modify our model to incorporate this by implementing a modified cross-entropy loss-function known as 'focal loss':

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

Here, $\gamma \geq 0$, and $(1 - \hat{y})^{\gamma}$ and $\hat{y}^{\gamma}$ are known as 'focusing' and 'modulating' parameters, respectively. Observe how as $\hat{y}$ approaches 1 or 0 —indicating a model's greater confidence in its predictions— the modulating factors approach 0, implying that the marginal loss generated by an example about which the model is relatively certain is relatively small. Conversely, $FL(y, \hat{y})$ is maximised at $\hat{y} = 0.5$ when the model's confidence is lowest.

4. Results and discussion

In evaluating our model's performance, we utilise three metrics: overall accuracy, fraud accuracy, and non-fraud accuracy. The latter two are used to gain insights into false positives and negatives. These are particularly relevant to transaction fraud detection given the significant financial cost associated

with undetected fraudulent transactions. Given the size of our dataset, metrics are calculated with simple cross validation using training, dev and test sets containing 70%, 15% and 15% of examples, respectively. The results obtained for all tested models are shown in Table 4.1.

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

It is observed that the performance of Naive Bayes is unstable, yet consistently produces significant discrepancies between train and dev performance. As noted, this strongly suggests the conditional independence assumption does not hold for the given data. The performance of the Weighted Logistic Classifier is promising and remains fairly consistent. It is evident from the difference in fraud accuracy between the train and dev sets, however, that the model somewhat overfits to the training set. Like the SVM, the unweighted logistic classifier labels all examples as non-fraud, and thereby confirms the importance of weights to counter the imbalanced nature of the dataset.

As noted, the performance of our baseline Neural Network model with cross-entropy loss is rather disappointing, although the consistency between the train and dev sets gives reason for hope. It is evident that under this model the extreme sparsity of fraud examples in the data, coupled with highly divergent values for w_f and w_{nf}, causes an unacceptably large discrepancy between fraud and non-fraud accuracy. Presumably, this is in part because relevant and informative non-fraud examples are almost ignored while the model fits mostly a minute fraction of fraudulent transactions. This hypothesis is corroborated by the performance of our model when we use f-sampling. Less extreme weight-discrepancies ensure that the model does not altogether disregard non-fraud examples, thereby greatly increasing the model's overall accuracy.

Unsurprisingly, the discrepancy in fraud accuracy between the train and dev sets in this model has increased dramatically in comparison with our baseline, suggesting the model overfits to fraud examples in the former. This is most likely a result of the limited amount of fraud examples available in combination with the potential feature similarity between frauds and non-frauds, making it difficult to detect general trends. Fortunately, this problem can be remedied somewhat by the introduction of a regularization term to the loss-function. Observe how the discrepancy between fraud accuracy in the train and dev sets under our baseline model is significant, whereas it is virtually non-existent when we regularize.

Also, observe the significant impact our feature selection has on model performance. When we combine these three additions to our baseline model and moreover implement the focal loss-function, we accordingly see much higher performance overall, with significant reductions to discrepancies both between fraud and non-fraud accuracies within the train and dev sets as well as between the train and dev sets themselves. The focal loss-function has likely also contributed strongly to this due to it penalizing false positives/negatives more heavily and by minimizing the contribution of high confidence predictions to the loss.

There are a number of parameters to be optimized for this final model, all of which are shown in Table 4.2, including their final, optimized values. This was accomplished by creating a wrapper that varies one parameter at a time whilst keeping the other parameters at their baseline values. By averaging over multiple runs, 'optimal' values were determined for every parameter. The metrics used for this optimization process are the same as the ones tracked in Table 4.1, although a greater emphasis was placed on pursing improvements in fraud accuracy. Example optimization plots for γ and f are shown in Fig. 4.1 and 4.2, respectively. Available time and computational power necessitated this somewhat rudimentary approach, although better results might be achieved by using more comprehensive optimisation strategies.

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

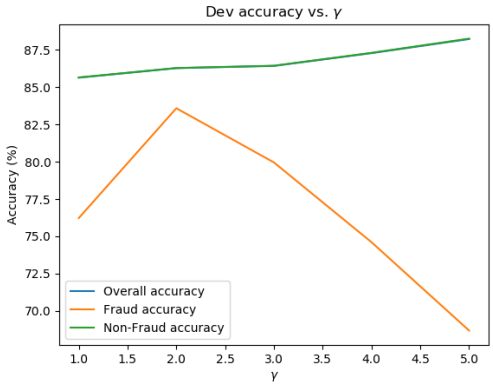

Figure 4.1: Optimization plot for γ.

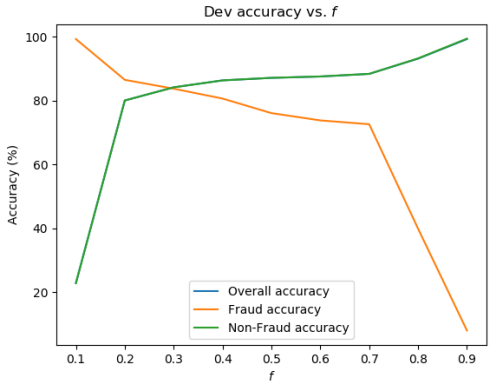

Figure 4.2: Optimization plot for f.

It should be noted, however, that our approach did corroborate many of our findings. In particular, access to the f meta-parameter provided us with insightful control over the model's fraud accuracy, albeit often at significant cost to overall performance. Larger batch sizes under our earlier models also clearly indicated that the sparsity of fraud examples was the main challenge for our model, hence directing us towards the use of our focal loss-function.

5. Conclusions

In conclusion, we find that a neural network with focal loss, regularization, and oversampling of the fraud class yields the most robust and successful method for overcoming the significant imbalance in our dataset and resulting danger of overfitting. We accordingly conclude that some method for correcting these imbalances, such as the use of f-sampling, as well as loss functions that incorporate weights and distinguish between high/low confidence predictions, and regularization are essential for adequate performance.

However, if we are to implement a system that performs real-time fraud detection, it is imperative we reach accuracies of at least 95% for all metrics, as the number of false positives and negatives is otherwise too high for commercial application. Future work should consequently be directed at exploring other pertinent features to add, performing a thorough optimization process, and performing real-time tests at Adyen in order to further improve accuracy. Another possibility —unfortunately not pursued for lack of time— is to refine our metrics in the form of a financial loss decision system, whereby the purpose of a model is not to maximise the number of transactions that are accurately classified, but instead minimise the costs associated with following up on fraudulent transactions based on the confidence of the model and the associated financial loss. Finally, approaches for dealing with the 'refused' examples are to be explored.

REFERENCES

[1] Kou, Y., Lu, C.-T., Sinvongwattana, S. & Huang, Y.-P. Survey of Fraud Detection Techniques. In *International Conference on Networking, Sensing, and Control*, 749–754 (IEEE, Taipei, 2004).

[2] White, T., Anderson, M. & Lavion, D. Global Economic Crime Survey 2016. Tech. Rep., Price Waterhouse Coopers (2016).

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

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

[5] Dal Pozzolo, A. & Bontempi, G. *Adaptive Machine Learning for Credit Card Fraud Detection.* Ph.D. thesis, Université Libre de Bruxelles (2015).

[6] Chan, P. K. & Stolfo, S. J. Toward Scalable Learning with Non-uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection. In *Fourth International Conference on Knowledge Discovery and Data Mining*, 164–168 (The AAAI Press, New York, 1998).

[7] Paasch, C. A. *Credit Card Fraud Detection Using Artificial Neural Networks Tuned by Genetic Algorithms.* Ph.D. thesis, Hong Kong University of Science and Technology (2008).

[8] Lin, T.-Y., Goyal, P., Girshick, R., He, K. & Dollár, P. Focal Loss for Dense Object Detection. *ICCV* 3 (2017).

A. Contributions

Both team members were involved in the decision making process as to how to approach the problem at hand and all encountered issues. In addition, report writing and model selection processes proceeded with input from both team members as well. By way of a more detailed division of labor:

Dirk contributed:

- Preliminary model selection
- Model implementations
- Tensorflow adaption
- Sampling strategies

Noël contributed:

- Preliminary dataset analysis
- Feature selection algorithm
- Parameter optimization
- Poster design