CS482/682 Final Project Report Group 8

Fraud Detection with Deep Learning

Irfan Jamil, Jianda Chen, Gabriel Givelber, Sungwon Kim
ijamil1@jhu.edu, jchen242@jhu.edu, ggivelb1@jhu.edu, skim434@jhu.edu

1 Introduction

Background and Related Work Credit/debit card fraud is when someone somehow obtains credit/debit card information and makes unauthorized purchases. Often, the card holder is not held reliable for such transactions. However, card institutions and merchants pay the price, having to reimburse customer accounts with usually no way to recall the already paid good/service. We seek to explore how deep learning architectures can perform in this domain with the goal of achieving not only high total accuracy but also high recall accuracy.

Previous work has been done related to neural networks and credit/debit card fraud. In the paper, "Using Neural Networks for Credit Card Fraud Detection" [1], the authors evaluate the neural network model's uses for detecting fraudulent transactions. Our models were particularly inspired by their methodology of balancing/resampling the data since fraud detection datasets are typically extremely imbalanced, the majority beging genuine transactions. Additionally, the paper discusses the architecture of the model which we used as a starting point for our own model (28 input neurons, 1 hidden layer with 13 neurons, and 2 output neurons). Incorporating these takeaways provided a solid foundation for which to add modifications and develop more specialized models.

2 Methods

Dataset The data was contained in two separate data sets which we joined on the attribute: Transaction ID. Once joined, we noticed there were many columns which had missing data. After some research into how to deal with attributes with missing data and data imputation [2], we decided to remove any attributes in which over 25 percent of the data was missing. Of the remaining columns, we determined which columns were numerical and which were categorical. The mean of each of the numerical attributes was used to replace null values for that attribute. The mode was used for categorical attributes. Then, for each categorical attribute, if the attribute took on only two values, this attribute was converted into a numerical attribute by simply assigning -1 and 1s [2]. For the remaining few categorical columns that we had, we decided to drop them from the dataset to make model construction easier and because the absolute value of their correlation to the binary 'is or is not fraud' category was small. For the final data set, we ended up with roughly 590,000 datapoints, each with 284 features.

A prominent issue we ran into over the course of the project was the major imbalance present in the dataset. Fraudulent transactions accounted for less than 1 percent of the dataset. Naturally, this is an issue when training neural networks as the tiny number of examples that the network sees that are fraudulent will contribute essentially nothing to the total loss of the network and thus the network can achieve high overall accuracy by predicting every transaction to be a valid, non-fraudulent transaction. The problem in this would be that of the frauds that were presented to the network, only a few would be detected and the false negative rate would be large. Thus, to resolve this problem, we researched techniques to handle imbalanced datasets and we exper-

imented with various re-sampling techniques including random under-sampling, random oversampling, and the use of the SMOTE technique [3].

Setup, Training and Evaluation One model we chose to detect fraud was a simple neural network with fully connected layers that took inspiration from [1]. Hyperparameter tuning included tweaking dropout ratio, using batch normalization, testing different number of layers, different optimizers, and different loss functions. The best preforming model had 5 fully connected layers each with ReLU and batch normalization with last layer also having dropout ratio of 0.5. Random over-sampling was used to construct the training batch.

Another model we chose to help us detect frauds was an auto-encoder feed forward neural network [4] [5]. To use auto-encoders in anomaly detection, the auto-encoder neural network is trained ONLY on non-anomaly instances which in our case is the valid transactions. Naturally, there will be reconstruction loss between the reproduced input vector that the auto-encoder outputs and the original input vector; however, as training progresses, this reconstruction loss will decline. After training, the auto-encoder is then trained on a random batch. Since the network has not seen any fraudulent transactions, the reconstruction loss for the auto-encoder on these examples will be unusually high. So, we set a threshold and classify a transaction as being fraud if the reconstruction error exceeds the threshold.

Another model we chose to use was an attempt at a neural decision tree model as discussed in [6]. In our research on Kaggle before starting, we found that this dataset was learned well by decision trees, so we thought a neural decision tree would naturally perform well on this data. Like a decision tree, we set up a binary tree, but now traversal decisions are made probabilistically by a deep neural network at each node. Each node is fed the data and makes a determination, and then the probability of arriving at each leaf is calculated to make a classification. To simplify our model as compared to that in [6], we decided to have leaf nodes only make a 1 or 0 classification. This keep the divide and conquer aspect of decision trees while also providing a differentiable and deep model.

Evaluation was based on the metrics that constitute a confusion matrix. This was pretty much the standard evaluation metric in the majority of the papers and references cited in this paper including in [4]. In anomaly detection, as touched upon above, accuracy can be misleading because of the imbalanced dataset. Thus, the primary metrics used to evaluate the models were true positive and true negative rates and from which is naturally computed: false positive and false negative rates.

3 Results

The model with the fully connected layer achieved 21 percent type 1 error and 23 percent type 2 error in the case when the training batch was comprised of 65% frauds. Table 1 in appendix has comprehensive data.

The best result that the auto-encoder achieved was about 70 percent true positive rate and about 70 percent true negative rate

We tried many different configurations of the Neural Decision Tree to try to optimize the accuracy and got results slightly better than the auto-encoder. The performance of the network was highly dependent on the sampling rate of frauds in the data, and we found that as the model trained, the recall rate would decrease, perhaps due to overfitting. The best we could get was 75% accuracy with a 75% recall rate, with a neural decision tree of depth 5.

4 Discussion

For each of the models that we implemented, we noticed a common trend. Often we could improve recall at the expense of having a higher false positive rate as can be seen in table 1 of the appendix. For both the neural network and the neural decision tree, the more aggressive our oversampling to make larger the proportion of the training data that were fraudulent transactions, we noticed that we gained higher and higher recall rate but the cost was that the mod-

els were predicting a higher proportion of non-fraud transactions to be fraudulent. Discussion continued in the appendix.

5 Appendix

Perhaps, a different approach to handling the imbalanced dataset would be to oversample less aggressively and instead compensate for the remaining imbalance by using the pytorch option of weighting the losses on the positive fraud more heavily than the losses of the non-fraud class. This would push the network to adjust the weights more substantially for every fraud that it encounters. Potentially, this would reduced the false positive rate and enable the same recall rate. But, this is hard to say without actually running the data through this altered model.

Additionally, it may have been better to combine both undersampling and oversampling. SMOTE produces artificial examples of fraud by identifying the k nearest neighbors of a fraud example and averaging the weights of these neighbors to produced a new fraud example. So, something that could have impacted our results may have been the use of SMOTE when training possibly produced numerous examples that fall near the decision boundary between the classes and thus pushed the model to have a higher false positive rate as a result. The decision boundary is likely very convoluted and not easily discerned; moreover, one can expect the data, in whatever dimension space that it exists, to be concentrated relatively close to each other otherwise fraud detection wouldn't be much of an issue in the first place! Thus, it is conceivable that during oversampling using SMOTE, some of the k nearest neighbors that are averaged in each iteration of creating a new fraud example may actually be non-fraud examples. Thus, not only is the decision boundary likely to be very jagged and convoluted as it is, but also the use of SMOTE may actually have created examples that resemble non-fraud transactions just as much as they do fraud examples. This would, then, lead to a higher false positive rate. A possible solution could be to combine oversampling and undersampling as mentioned above. It's conceivable that under-sampling would obviously aid in balancing the data set but it's also likely to have completely avoided the possible pitfall of SMOTE above.

Another way to interpret the results we have achieved and attempt to improve them would be to implement more regularization techniques to curb possible overfitting and acheive higher recall rate. As discussed in the first paragraph of the setup section above, the best performing neural network model was trained on a set that was randomly oversampled. With random oversampling (not SMOTE but standard random oversampling), it is possible that the model overlearned the fraud examples it was trained on and was thus susceptible to classify as frauds only those examples which very closely resembled the frauds it was trained on. Since random oversampling just replicates frauds in the training set, it is very possible that the model learns the peculiarities of these examples and thus struggles to learn a generalization of fraudulent examples. So, it could have been useful and beneficial to the recall rates to implement regularization techniques such as drop out and weight decay (l2 regularization) such as to minimize the possible overfitting that would occur when randomly oversampling and containing multiple copies of identical frauds.

Continuing with the above discussion, in the domain of fraud detection, false negatives are far more costly than false positives so this fact may lend itself to allowing a higher false positive rate that would otherwise be deemed too high in other domains. However, ideally, we'd want both a high true negative rate and a low false positive rate as to maximize accuracy and recall.

Looking back, something we could have done differently was spend more time on using auto-encoders for the task at hand as these architectures are better suited to learn only the representation of the non-fraud transactions which may lend itself to being able to accomplish both high true negative rate and low false positive rate.

We would have liked to further experiment with different auto-encoder architectures. The auto-encoder is, in essence, a feed-forward neural network so altering the number of hidden layers and the number of neurons in each hidden layer would have been

something we'd have like to experiment with and see the effect upon the results. Another two hyperparameters that would have been interesting to analyze and tweak would be the the number of training epochs that the auto-encoder trained on the non-fraudulent transactions and the measurement used to calculate reconstruction loss. In regards to the number of training epochs, it would be expected that the auto-encoder would learn better and better the key features and lower dimension representation of the non-fraud transactions as the training epochs increased. It'd be interesting to see if the reconstruction error as training epochs increased converged and, if so, how this would affect the results on the testing set. If it did converge, I would expect that the results would improve although the model would run the risk of overfitting. In regards to the measurement used to calculate reconstruction loss, it could be insightful to experiment with different ways to calculate loss such as mean squared error, cosine similarity, the median of the absolute value of the difference between each element of the output and input vector. Each of these essentially measure closeness but performing ablation tables by swapping the use of them could be insightful to improving results. Additionally, a natural hyperparameter to experiment with would have been the threshold value above that defined the fraud/non-fraud decision boundary. Of course raising the threshold could lead to higher false negative rate. And doing the opposite could lead to a higher false positive rate. However, experimenting with the threshold value could yield a threshold that could have low values of both FNR and FPR.

Below is a table that shows how the neural network performs as the proportion of frauds in the training batch is altered.

Fraud Probability	Type 1 Error	Type 2 Error
0.0	0.00000	1.00000
0.1	0.03190	0.48106
0.2	0.05031	0.39404
0.3	0.09382	0.33924
0.4	0.11740	0.32608
0.5	0.18207	0.23959
0.6	0.19824	0.26135
0.65	0.21206	0.23180
0.7	0.33411	0.17647
0.8	0.41844	0.13323
0.9	0.46279	0.11926
1.0	1.00000	0.00000

Table 1: Fraud Ratio in Training Data of Neural Network to Error Rate

Tree Depth	Accuracy	Recall
3	68.92%	81.43%
4	77.78	72.08
5	74.67%	74.51%
6	76.39%	71.47%
7	75.64%	71.14%

Table 2: Tree depth's relationship to accuracy and recall

References

[1] Sevdalina Georgieva, Maya Markova, and Velizar Pavlov. Using neural network for credit card fraud detection. *AIP Conference Proceedings*, 2159(1):030013, 2019.

[2] James McCaffrey. Visual studio magazine, Jul 2013.

[3] Jason Brownlee. Machine learning mastery, Jan 2020.

[4] Junyi Zou, Jinliang Zhang, and Ping Jiang. Credit card fraud detection using autoencoder neural network, 2019.

[5] Abel G. Gebresilassie. Neural networks for anomaly (outliers) detection, Jun 2018.

[6] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.