Transaction Fraud Detection through Machine Learning

Sam Johnson, Hunter Young, Kelsye Anderson

1 INTRODUCTION

Credit card fraud is one of the leading forms of identity theft in the United States today accounting for around 45% of all identity theft. Credit card fraud is defined as transactions that occur without the consent or knowledge of the account holder. This fraud can happen when credit cards are stolen, counterfeited, or when account information is leaked.

Currently, there are many trained models used to predict whether any given transaction is fraudulent or not. When a model determines that a transaction is fraudulent, the account holder's card may decline, and the holder may be notified. These precautions, although at times inconvenient, save millions of dollars a year from being stolen from consumers.

The problem at hand is to improve the efficacy of detecting fraudulent transactions and in turn save the time and resources of account holders and businesses. We do this by training multiple models using various methods on a large dataset. The dataset consists of a transaction table and an identity table. The transaction table contains information related to the transaction including the transaction amount, the product purchased, card information, and address of transaction. The identity table contains information related to online transaction information. This includes network connection information (IP address), browser, operating system, and device type.

This problem was chosen from a Kaggle competition, IEEE-CIS Fraud Detection. The two training datasets were provided to us as well as a testing dataset to evaluate our results. The results were evaluated based on the area under the curve (AUC). The results were submitted to Kaggle with the transaction ID along with the probability that the transaction was fraudulent.

This Kaggle competition ended around 2 years ago, but the public leaderboard from the competition remains. If we were placed on the leaderboard today, our ranking using our highest performing model would rank around 4000 of 6350. Our highest performing model was the Random Forest Classifier with an AUC of 0.904.

Many models were trained throughout this project including several classical machine learning models and one deep learning method using neural networks. The PyTorch framework in Python was used to implement this deep learning method. We were fairly successful in this method but failed to improve on the simple Random Forest Classifier.

2 EVALUATION METRIC

The metric used for the Kaggle competition was the AUC. This metric is used to evaluate the performance of each of our models. ROC is the probability curve while AUC is the area under that curve. The larger the AUC, the more accurate the model is at identifying fraudulent transactions.

The AUC ranges from 0 to 1, 1 meaning the model is 100% capable of detecting fraud while 0 means that the model predicts exactly opposite of the correct result. In theory, an AUC of 0.5 would occur if the probability of fraud was randomly predicted. This would mean that the model is completely incapable of distinguishing between fraudulent and non-fraudulent transactions.

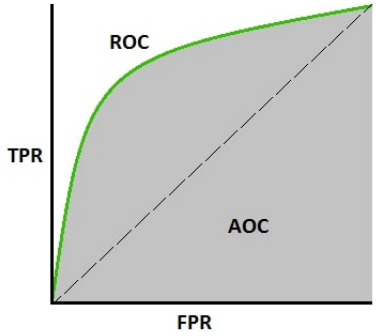

Figure 1: AUC-ROC curve

The ROC curve is plotted using TPR (True Positive Rate) along the y-axis and FPR (False Positive Rate) along the x-axis. The TPR is calculated by:

$$\frac{TP}{TP + FN}$$

while the FPR is calcuated by:

$$\frac{FP}{TN + FP}$$

where TP and TN represent the number of true positive and true negative results found in the generated data and FP and FN represent the number of false positive and false negative results in the data. The values change overtime as we change our threshold for predicting fraud.

3 QUANTITATIVE VARIABLE PREPROCESSING

The dataset consisted of quantitative variables and categorical (nominal) variables. Below we detail how we processed quantitative variables.

3.1 Quantitative Feature Selection

We ran an ANOVA test for each quantitative variable compared to our labeled feature, *isFraud*. We calculated an F-statistic and a p-value for each feature. Then, we selected all of the quantitative variables that had a p-value below 0.01. After performing feature selection, 61 quantitative variables were dropped from the dataset.

3.2 Standardizing Quantitative Variables

We standardized the quantitative variables by subtracting the mean from each value and then dividing by the standard deviation. The below formula was used where x is an observation, μ is the mean, and σ is the standard deviation of a given feature.

$$\frac{x - \mu}{\sigma}$$

3.3 Dealing With Missing Values

Many of the quantitative variables in the dataset had a large number of missing values. To get a better idea of how many values were missing for each feature, we created a *Null Ratio*. The null ratio was calculated by dividing the number of null values in a feature by the total number of values in that feature.

Figure 2 shows the null ratio of each of the features in sorted order. The labels aren't given. We can see that a large number of features are missing at least 50% of their values. Because a feature with so many missing values has little predictive ability, we opted to remove many of the features. We experimented with using various thresholds for feature removal and found that the results didn't change too much.

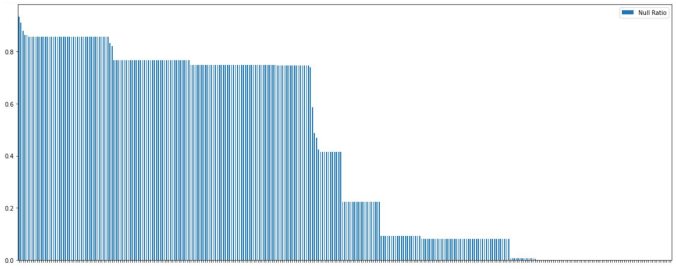

Figure 2: Feature Null Ratios

After removing these features, 177 quantitative variables remained in the dataset. The missing values in the remaining variables were simply filled with the average value of the given feature. The equation below where x is an observation and μ is the mean for a given feature is given below:

$$x = \mu$$

4 QUALITATIVE VARIABLE PREPROCESSING

As previously noted, the dataset contained numerous qualitative variables, each of which were nominal. Below we describe how we processed the data.

4.1 Dummy Variable Creation

The categorical variables were processed using one-hot-encoding. An additional feature was added for the missing values in each feature. One of the original features, *card*1, had 17091 different values and would have created that many new features. Because of computing and memory limitations, we opted to simply remove this variable from the dataset.

4.2 Dummy Variable Feature Selection

After converting the categorical variables into dummy variables through one-hot-encoding, there were nearly 8,000 different features in the dataset. To reduce the dimensionality of the data and select the most relevant features, we performed feature selection on the dummy variables.

We ran the chi-squared test on our dummy variables and calculated an f-statistic and p-value for each of them. Then, we selected only the features with a p-value below 0.01. After running feature selection, we reduced the number of dummy variables to 618.

4.3 Train - Validation Split

After preprocessing the data, we randomly split the training data by putting 60% of it into a training set and the remaining 40% into a validation set. We then fit each model using the training data and evaluated it on the validation data. This gave us a preliminary estimate of the model AUC and helped us fine-tune our parameters. Afterward, we submitted our predictions to Kaggle for the final AUC result.

5 SIMPLE & RANDOM BASELINE MODELS

5.1 Random Model

We generated a random model by guessing random probabilities between 0 and 1. The Kaggle submission resulted in an AUC of 0.494. This was to be expected, as mathematically we would expect to be correct 50% of the time. We then fit other models to see if we could beat this performance baseline.

5.2 Decision Tree

The first model we tried was the decision tree classifier using cross entropy as the loss function. This algorithm works by strategically choosing features and values to split on to most effectively increase the purity in the leaf nodes with as few splits as possible (KDnuggets)

To use a fitted model, you begin at the root node. Then, you simply follow the tree down a branch according to the split criteria. Finally, when you reach a leaf node, you have your predicted observation.

In addition, decision trees are able to predict probabilities. They do this by looking at the observations in the final leaf node that the relevant observation falls into. For example, if 80% of the observations in a leaf node were predicted correctly, then the decision tree would output a probability of 80% for all observations that fall in that node. In this manner we were able to obtain a predicted

probability that a given transaction was fraudulent.

Decision trees also allow you to set a maximum tree depth in order to avoid overfitting the data. We opted not to set a maximum depth for our analysis. We tested our model on the validation set. Figure 3 shows the resulting ROC curve.

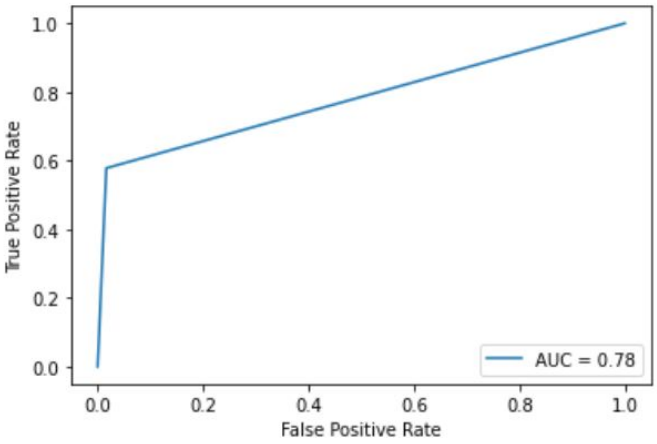

Figure 3: Decision Tree ROC Curve on Validation Set

In the end our decision tree classifier achieved an AUC performance of 0.704 on the Kaggle's private leaderboard, outperforming our random model.

5.3 Random Forest

Because the decision tree model didn't perform so well, we opted to try out the random forest classifier. Random forests work by fitting many different uncorrelated decision trees. When it's time to make a prediction, the predictions are aggregated and the classification with the most votes is selected. This technique generally results in more accurate predictions because of the so-called "wisdom of crowds" (TowardsDataScience).

So how do random forests ensure that the individual trees are uncorrelated? They do so through two techniques. The first is bagging wherein each tree is fitted using a random sample with replacement of the original data. Additionally, when it is time to split a node, only a random sampling of all features are available to make that split. Both of these strategies ensure that the trees are somewhat uncorrelated so that they don't all make the same prediction on every observation.

We first evaluated our model on the validation set. Figure 4 shows the ROC curve and AUC metric after evaluation.

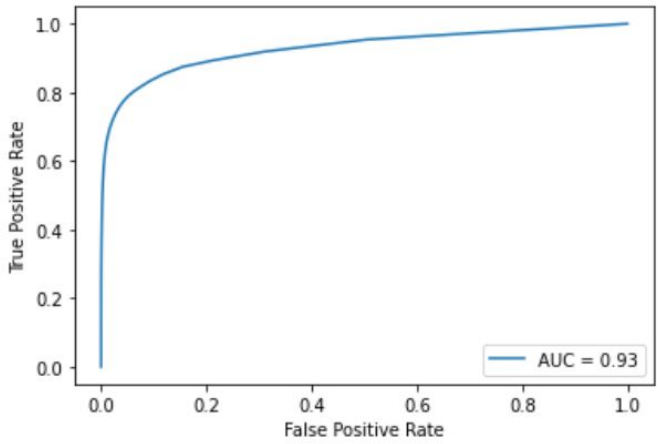

Figure 4: Random Forest ROC Curve on Validation Set

We then fit our random forest model with 100 different decision trees. The algorithm aggregated the results of each tree and spit out a probability for each observation in our test set. The random forest model performed quite well with an AUC of 0.9035 on Kaggle.

5.4 Logistic Regression

The next model we tried was Logistic Regression. Logistic regression works by first fitting an ordinary linear model. Then, it transforms the resulting output with the sigmoid function, which squashes it between 0 and 1 (TowardsDataScience). In our analysis, we interpret the resulting value as the estimated probability that a given transaction is fraudulent. The sigmoid function is given below:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

We ran the trained model on our validation set and it returned an AUC of 0.85. The results can be seen in figure **??**.

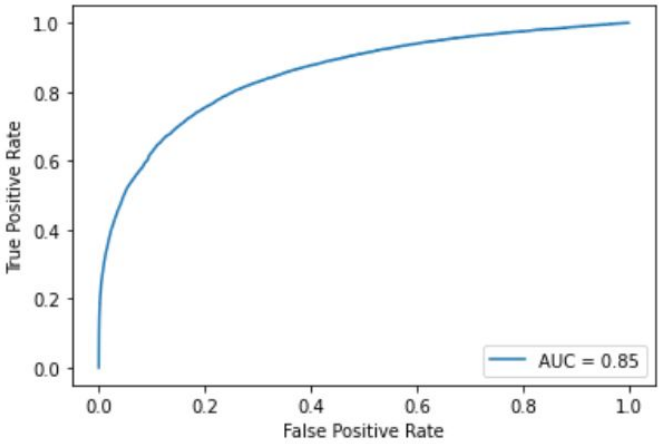

Figure 5: Logistic Regression ROC Curve on Validation Set

Our logistic regression model resulted in an AUC value of 0.875 on Kaggle, which outperformed the Decision Tree Model but underperformed the Random Forest Model.

6 DEEP LEARNING MODEL

We looked into several different neural network structures and researched what other people had tried for their Kaggle submissions when choosing what deep learning method we wanted to use for our report. After a bit of research, we decided to use the K-fold Cross Validation technique with a TabNet neural network. We used some code that someone else had already submitted as our base code and also modified it somewhat.

6.1 Pytorch's TabNet

TabNet is a neural network architecture that was designed specifically for learning from tabular data. The network filters the data through a learnable mask and selects a small set of features that most accurately classify it. We can see that process happening in figure 6.

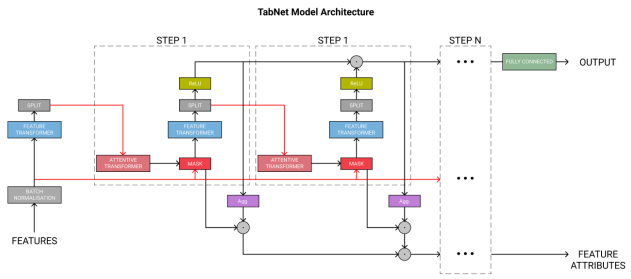

Figure 6: TabNet Model Architecture

First, the network inputs the data into the model and passes it through a normalizing layer. Then it passes the data into n layers of feature transformers or GLU blocks. These blocks consist of a fully connected layer, a batch normalization layer, and a gated linear unit layer where half of those layers are shared and half are independent. This structure has been demonstrated to have a much higher performance on large datasets compared to other tree-based algorithms because of its ability to learn how to reduce error using techniques like Gradient Descent.

6.2 K-fold Cross Validation Introduction

K-fold cross-validation is a data partitioning strategy. A common problem in building a model is over fitting to the training data. We want the model to be able to generalize well to data that it hasn't seen. Using the K-fold technique helps us to not overfit to our training data.

6.3 K-fold Technique

The first thing we do when using K-fold cross-validation is shuffle our dataset. The reason why we try training the model on the split-up data is we're assuming the observations in our dataset are all distributed equally. By shuffling our data set we're hoping to prevent any of our folds from being biased. We then split our data into k independent folds without replacement. The last fold is always used to check the performance while the others are used to train the model. After it has finished running, we have k performance estimates. We then take the mean of those numbers to get our overall performance estimate.

6.4 K-fold Implementation

In this model, we are training our network for 5 epochs a fold for a total of 25 iterations. We can see the results of one of these training episodes in figure 7. If we look at the graph at each of the iterations of 5, we notice the AUC drops a little as our model gets a new dataset. This shows that the model was overfitting to the previous data set and has to adjust its predictions as it's introduced to new information.

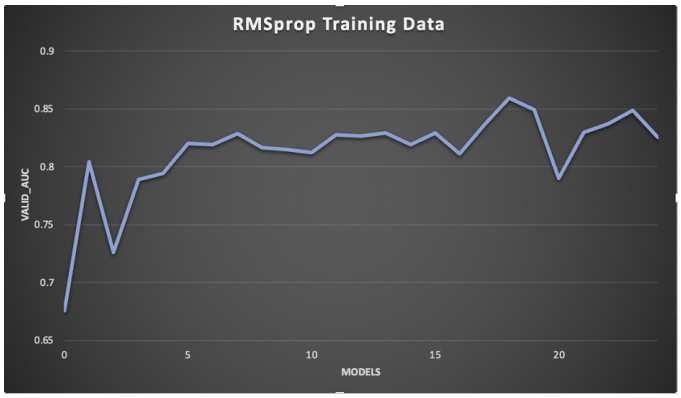

Figure 7: RMSprop Training Data

6.5 RMSprop

We tried several different optimizers with our model including Adam, Adamax, and SGD, but the optimizer that worked best with our k-fold strategy was RMSprop. This optimizer is interesting because it uses gradient-based optimization technique. Instead of keeping the same learning rate throughout the process, RMSprop adapts the learning rate as the gradient becomes larger and smaller. By normalizing the gradient, the optimizer increases the step size for smaller gradients and decreases it for those that are larger. As mentioned earlier, this technique worked well with the neural network structure we chose since TabNet is especially good at identifying changes in the gradient.

7 MODEL RESULTS

Overall none of our models performed exceptionally well on the Kaggle leaderboard. To give you a better idea of the range of submissions in Kaggle, the top score is 0.945884. The first AUC score to drop below .9 is in the $4,333^{th}$ spot.

Out of all the models we tried, the Random Forest Classifier performed best with a score of 0.903590.

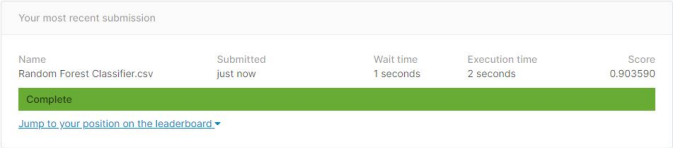

Figure 8: Random Forest Kaggle Results

Due to limitations in RAM, we were only able to run our folds with five epochs each. With the resources we had, our model was only able to score a 0.868008 in the Kaggle competition.

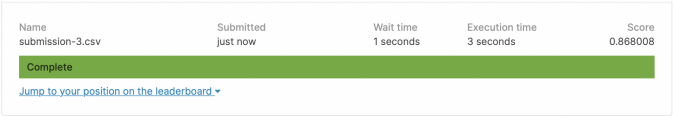

Figure 9: RMSprop Kaggle Results

8 CONCLUSION

This competition offers enough data and enough options that you could continue modifying things and resubmitting models for years. If we had more time and more resources, we would probably continue to fine tune hyper-parameters and test out different neural network structures to see if we could get a higher score.

Overall, this project was definitely enlightening about how real world data science problems are approached. By working with large data sets, we were able to learn how to preprocess data when you have hundreds of columns of data that may or may not be relevant to your results. We were then able to see how the performance of simple models can compare to deep learning methods and were able to learn and research about different ways of implementing neural networks. While it's impossible to train a program to be able to catch every instance of credit card fraud, it's interesting to see how we can use machine learning to detect it. We can then provide a faster and more accurate response to fraud for account holders everywhere.

9 REFERENCES

[1] Pramoditha, R., 2021. k-fold cross-validation explained in plain English. [online] Medium. Available at: <https://towards datascience.com/k-fold-cross-validation-explained-in-plain-english-659e33c0bc0>.
[2] Sanghvi, R., 2021. A Complete Guide to Adam and RMSprop Optimizer. [online] Medium. Available at: <https://medi um.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be>.
[3] Shafi, A., 2021. TabNet: The End of Gradient Boosting?. [online] Medium. Available at: <https://towardsdatascience.co m/tabnet-e1b979907694>.
[4] Silva, T., 2021. Machine Learning 101: An Intuitive Introduction to Gradient Descent. [online] Medium. Available at: <https://towards datascience.com/machine-learning-101-an-intuitive-introduction-to-gradient-descent-366b77b52645>.
[5] 25 Credit Card Fraud Statistics To Know in 2021 + 5 Steps for Reporting Fraud. [online] Medium. Available at: <https://mint.intuit.com/blog/planning/credit-card-fraud-statistics/>.
[6] IEEE-CIS Fraud Detection [online] Medium. Available at: <https://www.kaggle.com/c/ieee-fraud-detection>.
[7] Lim, Yoong Kang., 2019. Beginner's Random Forest example [online] Medium. Available at: <https://www.kaggle.com/yoongkang/beginner-s-random-forest-example>
[8] 2020. TabNet_fraud_detection [online] Medium. Available at: <https://www.kaggle.com/divyareddyyeruva/tabnet-fraud-detection>
[9] Chauhan, Nagesh Singh., 2020. Algorithms, Decision Trees, Explained [online] Medium. Available at: <https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html>
[10] Yiu, Tony., 2019. Understanding Random Forest [online] Medium. Available at: <https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html>
[11] Swaminathan, Saishruthi., 2018. Logistic Regression - Detailed Overview [online] Medium. Available at: <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>