Predicting Fraudulent Credit Card Usage with Random Forests

Tristan Lee
April 1, 2024

1 Introduction

I detail the use of random forests to classify various credit card transactions as legitimate or fraudulent. I discuss features in the data, from considered features to generated ones used in the final model, and determining how "good" they are. Then I discuss the various models considered for predictions, settling on random forests in the end. The workflow can be split into 2 general phases, an exploration phase and a model/prediction phase. I'll discuss the features and models used in terms of these phases.

2 Exploration

This phase consisted of identifying useful features in the data, that is, features that may help distinguish between legitimate and fraudulent transactions. Though I still found some additional features in the Model/Prediction phase, most of my insights of the data occurred here.

- I discovered pretty quickly that certain categories, namely `grocery_pos`, `misc_net`, `shopping_net`, and `shopping_pos`, have higher rates of fraud. That is, for each of these categories, the percentage of fraudulent transactions they account for is greater than the percentage of total transactions they account for.
- I wasn't able to make use of the user's and merchants' lat/long coordinates. I tried considering the subset of the data defined by a single user and examining their merchant lat's/long's, but I couldn't see any patterns (an example can be seen in Figure 2.1).
- A quite useful feature came from considering both the merchant and amount. For some merchant, the vast majority of legitimate transactions with that merchant fell

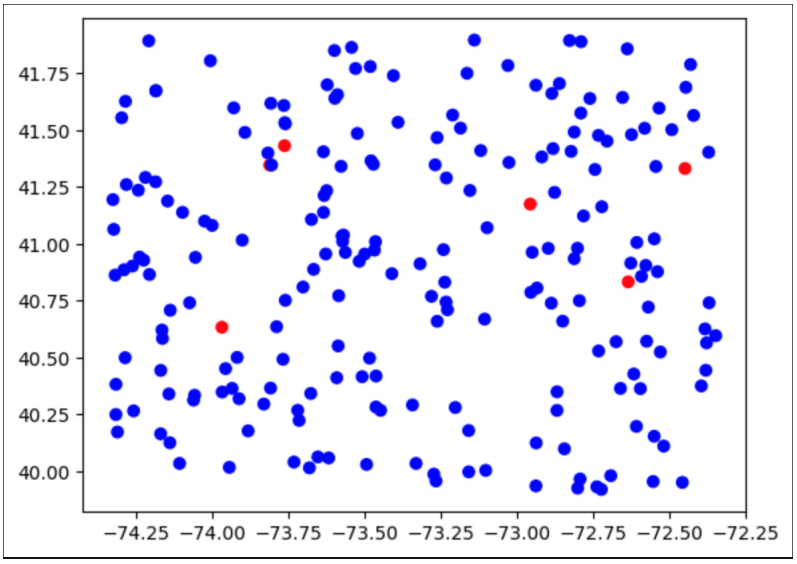

Figure 2.1: Example merchant lat's/long's for a random user.

within some mean and standard deviation amount. At the same time, most fraudulent transactions had amounts that greatly exceeded the mean amount for the corresponding merchant. Thus I created a feature `merc_outlier` that was the number of standard deviations away from the mean amount for that merchant. I applied a similar idea but for each user, and for each user-merchant pair, to get features `user_outlier` and `user_merchant_outlier`.

- I saw that most fraudulent transactions occurred at night, between the hours of 10pm and 3am. So I extracted the hour of each transaction as a feature. It's worth noting that I used the `unix_time` column to do this, rather than the `trans_date/trans_time` columns, as these appeared to be the same value for all entries.
- Finally, I figured I could use the dates and user's dates-of-birth for each transaction to get an `age` feature, which may be important, though I didn't try to observe this.

3 Model/Prediction

With these features, I moved onto training and predicting with a model. I tried KNN, logistic regression, and decision trees/random forests, and the latter produced the best F1 scores locally, so I used these to evaluate the quality of features.
My workflow generally went as follows: identify some set of features to try, then train and predict using a decision tree on a split of the given training data. Then I examined the tree's `feature_importances_` attribute to see if all used features had at least some noticeable importance, which I arbitrarily set to 0.01. Once I was satisfied enough with the importances and F1 score, I switched to using a random forest using the same features to produce a

submission for Kaggle.

- With this method, I found that the `user_outlier` and `user_merchant_outlier` features had little importance and negatively impacted the F1 score, but the `merc_outlier` feature was quite important and improved the F1 score greatly.
- In this phase, I also starting using a new feature `zip_fraud_rate`, which is the ratio of fraudulent to total transactions for the corresponding zip code. In turns out that certain zip codes only have fraudulent transactions (corresponding to rates of 1.0). This feature's importance was also quite high and improved the F1 score slightly. It also caused the importance of `merc_outlier` to fall, perhaps because each merchant has a zip code, so the information between the two is redundant. In any case, I also included `merc_fraud_rate`, which is the same thing but for each merchant.
- I also experimented with outlier detection using SVD, using various subsets of the features, to see if it could capture some relationship between the features that the decision tree/random forest could not. This consistently had low importance when using the other features, though, so in the end I excluded it.

In the end, after trying various sets of features, my final set of features and their importances as determined by a random forest was the following:

```
[('zip_fraud_rate', 0.6489403201753564),
 ('amt', 0.16620472763201152),
 ('merc_outlier', 0.08454432060608562),
 ('hour', 0.03657349616878997),
 ('merc_fraud_rate', 0.03312727318599082),
 ('category', 0.030609862231765622)]
```

The `zip_fraud_rate` had by far the most influence, followed by `amt` and `merc_outlier`. The `category` feature used to have higher importance, but the merchant and zip code related statistics seems to have captured most of its information.
In terms of implementation, for both the intermediary decision trees and final random forest, I set the `class_weight` argument to `'balanced'` to account for the imbalance in each class distribution. While evaluating the model, I used a test-train split of 80-20, but for the final submission I changed the split to 95-5, just to use more of the data. I tried preventing overfitting by changing the `max_depth` and various leaf arguments, but these only gave similar or lower F1 scores. I suppose the nature of random forests took care of overfitting issues by taking random subsets of the data.