Predictive Model for Fraudulent Credit Card Transactions

DSC 178 Final Project Report

Professor Jingbo Shang Jeffery Lai

Introduction

Credit card transactions make up the majority of online and in person transactions today, requiring only a tap of a card or click of a button. Owing to the advancements in technology today, hackers and criminals have formulated methods to steal customer information on a massive scale. The consequences of credit card fraud are detrimental, ranging from negative impacts on individuals' credit scores and causing great inconvenience, to significant financial losses for institutions. In this project, I will be building a model to predict whether a credit card transaction is made by its rightful owner.

1 Dataset

1.1 Identify Dataset

The dataset we are using (card_transactions.csv) is a collection of simulated credit card transactions from the beginning of the year 2019 to the end of 2020. The training dataset contains 23 features as columns and approximately 1.3 million rows, with the column 'is_fraud' as the label (having 1 represent an instance of fraud and 0 no fraud).

Unfortunately, there is no public dataset of real credit card transactions to protect the personal information of consumers. All datasets containing real information have undergone the process of PCA dimension reduction, making all of their features unrecognizable. While using such data could be effective in practice, it would not be suitable for this project since we are trying to analyze how the underlying features encourage specific design choices for our model. Thus, we are using data downloaded from Kaggle, generated courtesy of Sparkov Data Generation by Brandon Harris.

1.2 EDA

Basic Statistics

The columns of this dataset represent data from transactions simulated from a predefined group of customers and merchants, and each row is a separate recorded transaction. It includes basic information about the customers such as their name, address, and credit card number. It also includes the merchant's name and location. It contains data about the transaction itself, such as the category, time, and amount. Finally, there are no missing values, which makes cleaning the data easier.

We immediately notice that there is a major class imbalance between the positive and negative instances of fraud, with there being a significantly fewer number of positive cases (Figure 1). This is consistent with real world data since there are by far more legitimate transactions than fraudulent ones. We must take this into account when creating and evaluating our model.

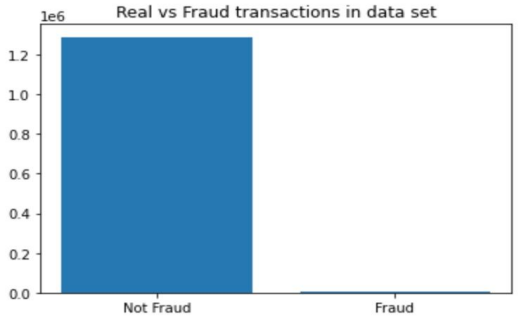


Figure 1: Dataset fraud class imbalance

Inessential Attributes

We can start off by dropping the 'street,' 'first,' and 'last' name columns, since they coincide perfectly with their credit card number, which already uniquely identifies each person. Theoretically, the name or street address attributes alone should not have any influence on the outcome of the label. We can also drop the 'trans_num' column since each transaction is a unique random string, which would not help in the prediction at all. Finally we can simplify the date of birth column by changing it to birth_year, since the specific day would not be important to the model.

Interesting Findings

We begin to analyze our data by creating a heatmap from the dataframe containing all of the quantitative variables (Figure 2). Since there are no features that immediately have a high correlation with the fraud label, we dive deeper into the other columns. We will analyze the most notable columns below.

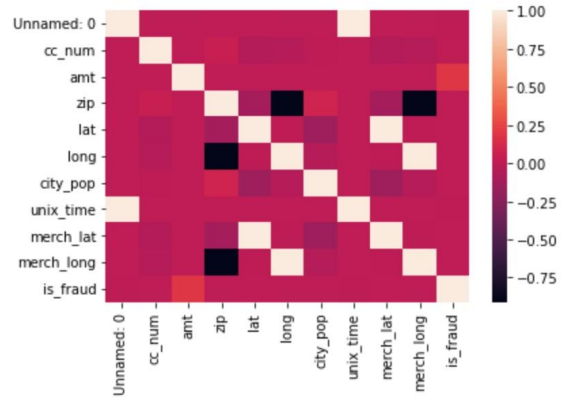
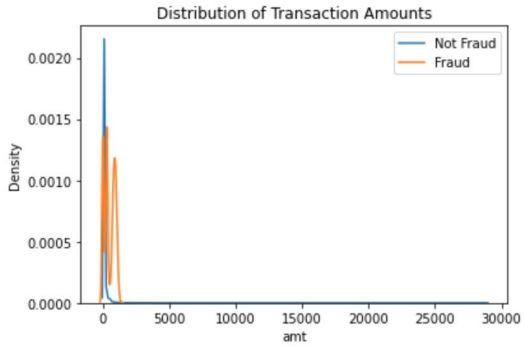


Figure 2: Correlation heatmap of fraudulent transactions dataframe

amt

We can start with the 'amt' column, as it has the highest correlation with the label we are predicting. We plot the density distribution of transaction amounts for all positive and negative instances of fraud separately (Figure 3). We immediately notice that the distribution of fraudulent transactions is on average greater compared to that of the legitimate transactions. However, we should remember that there are much fewer fraud transactions than not fraud, so if we were to plot the counts instead of relative densities, the peak of the fraud plot would not be significantly higher than that of the non fraud plot.

The statistics are listed below the graph, and we can see that although the mean for fraudulent transactions is much higher, normal transactions have a much greater range. This makes sense because most everyday transactions are not a high amount, but criminals want to steal the maximum amount possible without setting off any alarms.




```
Not fraud mean amount: 67.667109812599
Fraud mean amount: 531.3200919264573
Not fraud max amount: 28948.9
Fraud max amount: 1376.04
```

Figure 3: Distribution of transaction amounts between all fraudulent and non fraudulent transactions

category

The first categorical column we will analyze is 'category,' which has 14 distinct values. Splitting the data into fraud and not fraud instances, we plot the proportion of spending by category as a grouped bar chart (Figure 4). It is evident that a higher proportion of fraudulent transactions are from online purchases or grocery stores as compared to the relative proportions for genuine transactions. Consequently, there is a lower relative proportion for fraudulent transactions in every other category.

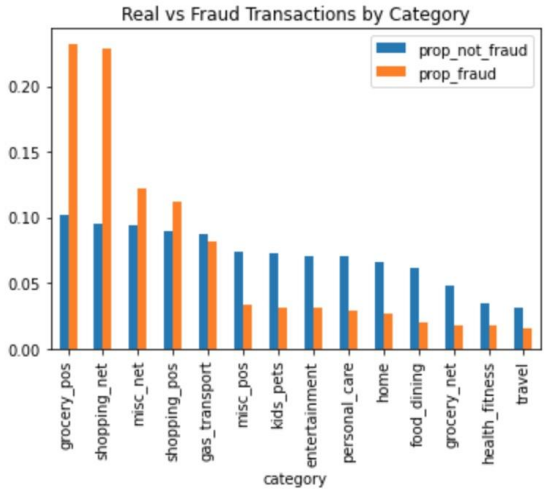


Figure 4: Grouped bar chart showing transactions by category

city_population, birth_year

Using a scatter plot, we can analyze if there are any interactions between city_population and birth_year and fraudulent transactions, either isolated or combined (Figure 5). We want to know if certain age groups are more likely to have their information stolen, or if living in a higher population area makes one more susceptible to credit card theft. By using a scatter plot, we can see if there is a more complex underlying relationship between fraud and both variables together. Though there is no apparent trend, we do not have enough evidence to not include these columns in our model.

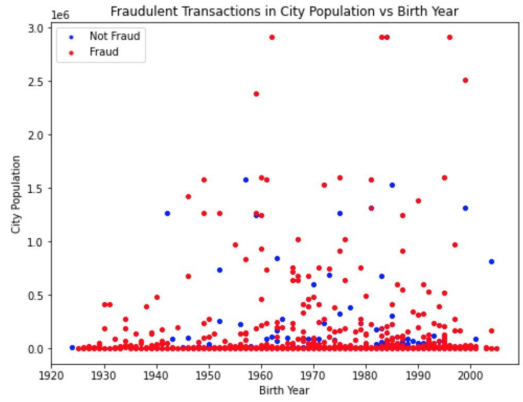
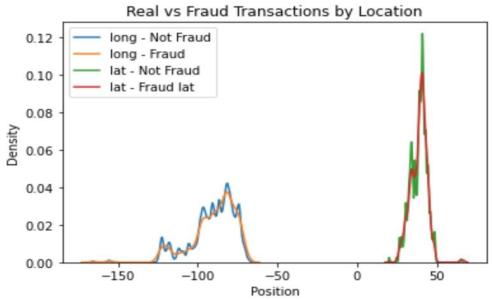


Figure 5: Scatter plot of individuals who are victims of fraudulent transactions based on city population vs birth year

long, lat, merch_long, merch_lat

After examining the location features, unfortunately it does not seem like there is any difference in the distributions for fraud or non fraudulent transactions by location (Figure 6). However, we should not drop these columns yet as there could be more complex interactions happening with other attributes that we can not yet see.



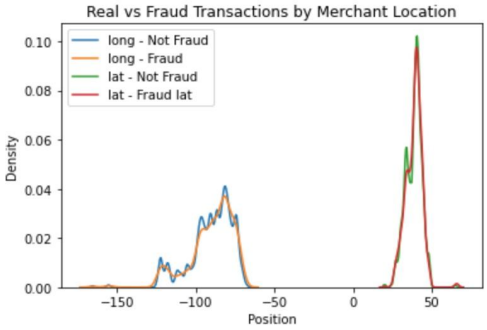


Figure 6: Distribution of transactions by cardholder location and merchant location

2 Predictive Task

2.1 Predictive Task

Given the available features, predict whether or not a credit card transaction is fraudulent. This is a classic binary classification problem, and the target label in this dataset is 'is_fraud.'

2.2 Evaluation

As we discovered earlier, there is a major class imbalance in the fraud instances vs the not fraud instances. Since over 99 percent of the transactions are not fraudulent, accuracy would be a poor measure of our model's performance. Doing so would encourage our model to have a higher bias, since had the model simply predict that every transaction is not fraudulent, it would achieve an accuracy rate of the proportion of non fraudulent transactions.

Thus, we will use the other evaluation metrics for a confusion matrix, which include precision, recall, and f1-score, which is a combination of the precision and recall metrics (Equation 1). Precision is the ratio of instances that are true positives of the ones that the model predicts are positive, so a high precision in this case would mean that the model is accurate in not misidentifying legitimate transactions as fraudulent. Recall is the ratio of true positive instances that the model correctly identifies as positive, in this case a high recall would indicate that the model is catching most of the fraudulent transactions.

Equation 1:

F1 = 2 * (precision * recall) / (precision + recall)

Precision may be more important if the institution is willing to incur losses to improve customer satisfaction, as incorrectly flagging transactions could be frustrating for the card user. However, recall may have the slight edge in this case since the process of dealing with fraud after it has happened is extremely inconvenient for both the consumer and costly for the credit card company. Thus, maximizing recall could be more beneficial, but we will use f1-score as the overall metric to evaluate our models to get an equal balance of both metrics. However, we will still include both precision and recall separately.

Training, Validation, and Test Data

There is a separate train and test set included in the downloaded dataset, so we will train our model on a subset of the training set while tuning it with our validation set, which is the remaining portion of the training set. Once we have tuned all the hyperparameters of our model, we will run it on the test set to evaluate its final performance on unseen data.

2.3 Baseline Model

We will be trying to create a baseline model using 3 different classification models. We will be using the basic categorical features ['cc_num', 'merchant', 'category', 'zip'] and quantitative features ['amt', 'lat', 'long', 'city_pop', 'birth_year', 'unix_time', 'merch_lat', 'merch_long']. All models are set on default hyperparameters except to prevent extreme overfitting or exceptionally long run times.

Logistic regression can be used for classification tasks by calculating the probability that an instance belongs to a class. It works as a good baseline as a linear classifier.

Random forest classifier is a popular classification algorithm which uses an ensemble of decision trees. Each individual tree is trained on a random subset of the features and training data, then the predictions are taken from all of the trees and the majority prediction is returned. It is capable of finding complex interactions between features which would be helpful for working with our given data and is robust to overfitting.

Gradient boosting classifier also builds an ensemble of decision trees, but instead each subsequent tree is specifically trained according to the errors of the previous tree. Though gradient boosting can lead to higher accuracy, this method is prone to overfitting. Limiting the complexity of feature interactions and setting a max depth are some ways to prevent overfitting.

Baseline Results

Random Forest Classifier and Gradient Boosting Classifier performed significantly better than their Logistic Regression counterpart. Random Forest achieved a baseline f1-score of 0.73 and Gradient Boosting 0.68, while Logistic Regression was at a mere 0.15. Therefore, we will continue building our real model with Random Forest Classifier and Gradient Boosting Classifier.

3 Model

3.1 Model Design

We will be using the same training and validation sets as we did previously for our baseline models, and will be moving forward with RandomForestClassifier and GradientBoostingClassifier. We will be using most of the basic features of the baseline model and adding a few ordinal columns such as 'city' and 'state' and also a few more features we will later engineer.

3.2 Optimizing Model

Feature Engineering

We are given a date time attribute in the dataset, but it alone may not be useful. In order to determine any potential cyclical patterns of fraud in the data, we create 2 separate columns, extracting the month for one to see if there is a relationship between fraud and the time of year, and extracting the hour of the day for the other to determine if the time of day is a determining factor. When plotting the transactions against the hour of day, we can see a noticeable difference in the distributions between instances of fraud and genuine transactions (Figure 7). This makes sense since criminals would feel safer conspiring at night than in broad daylight. This relationship may not have been caught had the time been left in unix time, as the overall date added noise to the data (Figure 8).

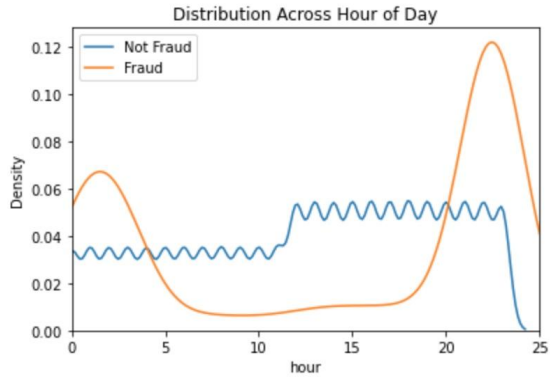


Figure 7: Distribution of transactions over hours of the day

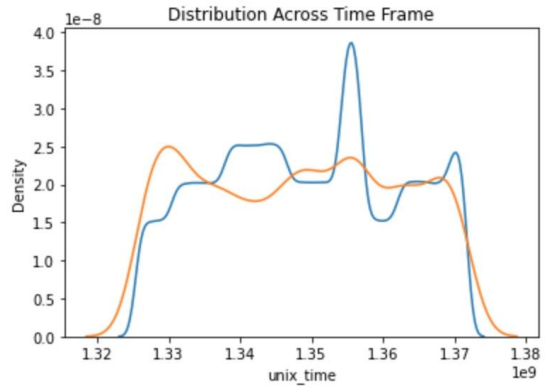


Figure 8: Distribution of transactions over the entire recorded time period

Using the longitude and latitude coordinates of both the credit card billing address and the merchant location, I engineered a distance feature indicating from how far each transaction was made. However, it did not have much of an impact on the evaluation score since the distributions of distances for fraudulent and non fraudulent transactions were virtually identical, so this feature was dropped. This is consistent with our earlier discovery that the latitude and longitude distributions were similar.

Hyperparameter Tuning

Taking into account the size of our dataset, running GridSearchCV with 5 fold cross validation is very computationally expensive, so we will only run grid search using the most influential parameters for each model. We can estimate the ideal hyperparameter range using intuition and our knowledge of our dataset size and class imbalance. We would want to somewhat limit splitting to reduce overfitting, while not setting the thresholds to split the internal nodes of the decision tree too high so that the model is still able to identify the positive instances of fraud. We can limit overfitting by decreasing the maximum depth and by requiring more samples to split an internal node.

3.3 Optimization Results

After running GridSearchCV for both models, we got the following combinations as the most effective. For RandomForestClassifier we have max_depth = 10 and for GradientBoostingClassifier we have min_samples_split = 1500, min_samples_leaf = 50, max_depth = 8, subsample=0.8.

Though the RandomForestClassifier (Figure 9) achieves a very high recall which may be appropriate for certain applications, GradientBoostingClassifier (Figure 10) has an overall higher f1-score by a margin of 0.02, so we will be continuing with this model for our final test.

```
In [72]: f1_score(rfc_pred, y_val)
Out[72]: 0.809617658651951

In [73]: precision_score(rfc_pred, y_val)
Out[73]: 0.688337801608579

In [74]: recall_score(rfc_pred, y_val)
Out[74]: 0.9827751196172249
```

Figure 9: RandomForestClassifier f1, precision, and recall scores

```
In [131]: f1_score(gbc_pred, y_val)
Out[131]: 0.8308931185944364

In [132]: precision_score(gbc_pred, y_val)
Out[132]: 0.7607238605898123

In [133]: recall_score(gbc_pred, y_val)
Out[133]: 0.9153225806451613
```

Figure 10: GradientBoostingClassifier f1, precision, and recall scores

3.4 Model on Final Test Set

We will now read in a separate test set to obtain an independent evaluation of the performance of our model. Using the GradientBoostingClassifier, we get a final performance of f1-score: 0.77, precision: 0.75, recall: 0.80, which is consistent with its performance on the validation sets. It is expected to perform better on our validation set because it came from the same subset of data, and the test set we just used was generated as a separate dataset. We can use a confusion matrix to visualize the accuracy of its predictions (Figure 11). Ultimately, a f1-score of around 0.8 is acceptable, so we can conclude that our model is successful.

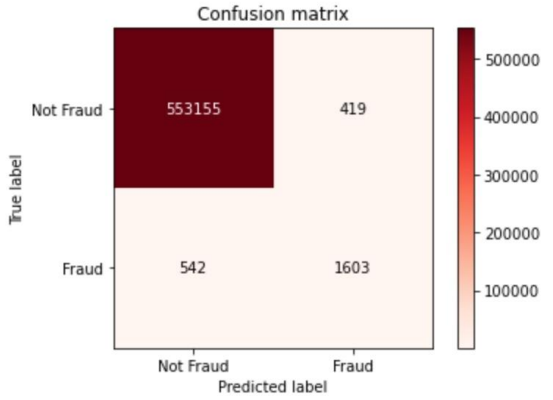


Figure 11: Confusion matrix showing performance of model on final test set

4 Literature

4.1 Previous Models

https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive-models

Gabriel Preda takes on the same task as described in this project, but he uses a PCA dimensionality reduced dataset instead. Preda does a more complete analysis of the data in the EDA process, but the dataset he is using contains more features. We both start off on a similar train of thought by looking at the differences between the distributions of fraudulent and genuine transactions for the amount and time categories. However, he goes further into depth in the other attributes his dataset contains, plotting multidimensional graphs for more detailed analysis.

He also proposed to use different evaluation metrics for his model, GINI and AUC. GINI measures the true positive rate and false positive rate, ensuring that the model identifies most true positives while not inaccurately classifying negative instances. AUC is a more holistic measure, an indicator of how effectively and confidently a model can distinguish between classes. I used f1-score, precision, and recall as my evaluation metrics, which essentially accomplish the same thing. Precision measures how well a model predicts true positives without misidentifying the negative instances, while recall measures how well a model predicts true positives without missing many other true positives. Ultimately, f1-score acts as a composite measure of both metrics, accomplishing the same goal of the GINI metric.

Preda experiments with a more diverse collection of models than I did, including many which are not available in the scikit-learn library. His best performing models were XGBoost and LightGBM, both of which are gradient boosting algorithms. I used scikit-learn's GradientBoostingClassifier, which is less powerful, but accomplishes a similar task.

4.2 Similar Projects

This work has likely been studied since the beginning of the creation of credit cards, both by amateur data scientists on Kaggle and by professionals in financial sectors. While my model may be designed slightly differently than many other existing models, there are many models that perform at a much higher accuracy. Banks and financial institutions have almost certainly invested heavily into developing accurate models, and they have access to classified data that is not made available to the public.

5 Results

5.1 Model Performance

Final vs Baseline Model

Fortunately, our final model was able to outperform our baseline models by a sizable margin. The final GradientBoostingClassifier had a 14% improvement in f1-score compared to the baseline RandomForestClassifier, and a remarkable 22% improvement over its own baseline model. By adding more features and tuning hyperparameters of models that were inherently more effective, I was able to end with a final model with a more optimal performance.

Effective Features
I tried designing multiple features, but the ones that proved to be the most effective were 'month' and 'hour' (of the day). Decomposing 'unix_time' into their respective months and times revealed more underlying patterns in the data that the baseline models did not pick up from 'unix_time'.

There were also features that were not very effective, such as the distance feature I tried deriving from the longitude and latitude coordinates of the address and merchant location. Since the distributions of this feature were identical for all fraud and legitimate instances, it was not at all surprising that it had no impact on the effectiveness of our model.

Challenges
One of the biggest challenges in optimizing the data was working with CVGridSearch, which is increasingly computationally expensive when a large combination of features are included. To counteract this, I ran the grid search for only the hyperparameters with the greatest impact, then manually adjusted the remaining less important parameters. Another challenge was designing relevant features from the available columns, since there were not many features with high correlations with fraud to begin with. The only useful features that I was able to derive were from the date/time column.

5.2 Major Takeaways
It is important to take into consideration the structure of the dataset I am working with when designing a model and selecting an evaluation metric. In this case, there is a major class imbalance, which must be accounted for when tuning the model hyper-parameters and evaluating its performance.

Sometimes, intuitive features are not always helpful, like the distance feature I attempted to engineer. Though it may seem like it would be a good indicator in predicting fraud, it does not actually improve model performance.

When building this model, I didn't follow the exact order of steps taken in this proposal. There were multiple instances when I went back to clean a feature column or make a new feature to improve the model's performance. I even had to go back to do additional EDA to determine if a column was worth including in the model.

Finally, I limited my choice of classification models to the ones that were already built into the sklearn library. I skimmed through similar projects that have created more robust models using third party libraries such as XGBoost and LightGBM. In my future projects, I would consider including those models as well in the baseline models.

REFERENCES

[1] https://scikit-learn.org/stable/
[2] How to create a meaningful EDA https://www.kaggle.com/code/marcinrutecki/how-to-create-a-meaningful-eda
[3] Complete Machine Learning Guide to Parameter Tuning in Gradient Boosting (GBM) in Python https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/