Fraud Credit Card Transaction Detection

DSCI 303 Final Project

(Apple) Xuanchen Li
xl102@rice.edu
Rice University

(Derek) Tiancheng Fu
tf18@rice.edu
Rice University

ABSTRACT

We aim to build a machine learning pipeline with a model that best distinguishes between legitimate and fraudulent transactions specifically on credit cards. We performed data cleaning and feature engineering on our dataset, then compared the effect of different classifiers including Logistic Regression, Support Vector Machine, Random Forest, and AdaBoost. Results and corresponding plots, along with the interpretations, are presented in the end.

1 INTRODUCTION

In 2021, the Federal Trade Commission (FTC) fielded nearly 390,000 reports of credit card fraud, making it one of the most common kinds of fraud in the U.S. [2]. Hence, fraud detection has played a crucial part in building secure products for companies that offer financial services. By clearly, efficiently, and accurately identifying fraud transactions, companies are able to prevent clients from losing money. In this project, we aim to build a machine learning pipeline with a model that distinguishes between legitimate and fraudulent transactions specifically on credit cards. Such models can be used by banks and online payment companies to block malicious transactions and protect their customers' properties.

Our hypothesis is that fraudulent credit card transactions are associated with unusual patterns, and can be detected by identifying differences from a cardholder's normal behavior and transaction characteristics.

2 RELATED WORK

Financial institutions tend to use a combination of techniques and algorithms to detect fraudulent transactions, depending on the specific characteristics of the transactions. Specific client/customer data is highly private in the real world, hence we do not know the exact modeling process behind each fraud detection department of those companies. Some research papers are available, such as Credit Card Fraud Detection using Machine Learning Algorithms by V. Dornadula and S. Geetha, where they used SMOTE (an oversampling technique) and SVM (Support Vector Machine) for their model [1].

We aim to use this project as a practice to build a robust model with proper data cleaning and feature engineering process that will serve well when we get to work with industry-level datasets in the near future.

3 METHODS

We performed data cleaning and feature engineering on our dataset, then compared the effect of logistic regression (baseline model), Support Vector Machine, random forest, and AdaBoost.

Table 1: Correlation between Numeric Variables and the Target Variable "is_fraud"

Variable Name	Correlation
is_fraud	1.000000
amt	0.219404
city_pop	0.002136
lat	0.001894
merch_lat	0.001741
merch_long	0.001721
long	0.001721
cc_num	-0.000981
zip	-0.002162
unix_time	-0.005078

3.1 Exploratory Data Analysis

We use the dataset "Credit Card Transactions Fraud Detection Dataset" [3] found on Kaggle. Since real transaction data are naturally kept private,the dataset we chose is created using a simulator. Nonetheless, this dataset should still be highly similar to a real-world dataset on credit card transactions. The dataset contains 1,296,675 rows of data in the training set and 555,719 rows of data in the testing set, including both legitimate and fraudulent credit card transactions from 1/1/2019 to 12/31/2020. For each row, the dataset includes 10 numeric variables and 12 categorical variables. Because the entire dataset is simulated, there are no missing values, hence there is no need to drop any particular rows or columns.

As shown in **Table 1**, most numeric variables do not have a strong correlation with the target variable "is_fraud" other than the variable "transaction amount", which has a seemingly moderate relationship with the target variable "is_fraud". This indicates that standard linear models might not be able to produce the best results for this dataset. We could potentially look into tree algorithms for a better model. In addition, we also observe that some numeric variables are suitable for further feature engineering, such as variables "cc_num", "zip", and "unix_time".

We then looked at the categorical variables, but not much clear pattern is found from any individual variable. Therefore we plan to perform feature engineering on some of those variables, such as "category" (transaction category), "job" (jobs of the cardholders), and "gender", so that we could hopefully extract some useful information after the transformations. We also found some categorical variables non-essential to our model and should be removed before we build the model. For example, categorical variables "first" and "last" record the first and last name of the cardholders and thus provide overlapping information with the customers' credit card numbers recorded in variable "cc_num".

3.2 Feature Engineering

In our exploratory data analysis, few variables exhibited strong correlation with the target variable. Therefore, we decided to use feature engineering to create new variables based on existing dataset, tailoring them to better suit the purpose of our project.

We first dropped columns that are unnecessary, namely "first", "last", "street" (street address), "city", "state", and "trans_num" (identifier for each transaction). We removed these variables because they either contain overlapping information with another variable, or that the content they are storing is meaningless in the context of transaction fraud detection—for example, the unique ID related to each transaction stored in "trans_num" does not provide any helpful messages for our model at all.

We then created a few new variables. The "dob" variable (date of birth of cardholder) was transformed into a numeric variable "age" indicating the cardholder's age. Such transformation extracts useful information from a seemingly useless categorical variable, as people in certain range of age (for example, very young adults or the elders) could be more frequent victims of credit card fraud. We split "unix_time" (time of the transaction) into "trans_weekday" and "trans_hour", representing the day of the week and the hour of the day of the transaction, respectively. As shown in **Figure 1**, we start to see a trend among the fraudulent transactions—most of those transactions happen during midnight, thanks to the "trans_hour" variable. We also noticed from the distribution of "trans_weekday" that more fraud transactions happen at weekend and on Monday. These variables can be very important predictors to indicate credit card fraud during modeling, which we will show in the next section.

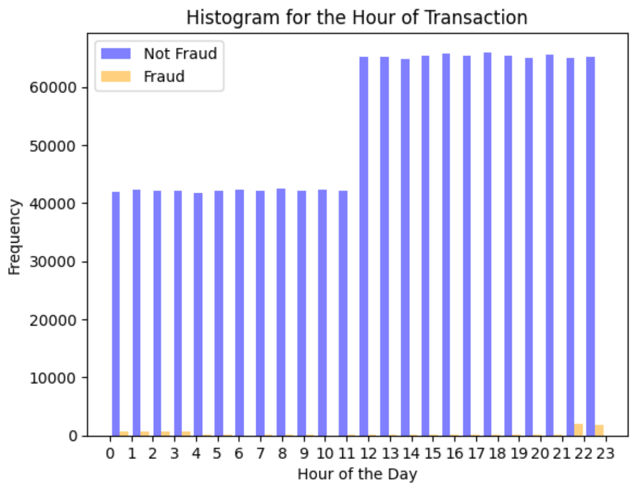

Figure 1: Histogram of Amount of Fraud and Non-Fraud Transactions, by Hour

After that, we used one-hot encoding on the "gender" variable, creating two new variables "gender_F" and "gender_M" out of the two possible inputs for the variable "gender". Applying one-hot encoding breaks this categorical variable into two numerical variables, thus helping the model better understand and utilize the information more easily. Besides, using one-hot encoding also makes sure that we avoid ordinal assumption on a variable that previously lacks such idea.

Additionally, we applied count encoding to categorical variables "category" (type/use of transaction, such as entertainment), "merchant", "job" (job of the cardholder), "cc_num" and "zip". This process involved creating a new variable for each of these variables, populated with the number of occurrences of their respective categorical values. We used count encoding for the latter variables because they have higher cardinality than the variable "gender"; besides, the frequency of occurrences might also play a role in our model—if an attacker choose to over-flood the transaction system with malicious requests on one specific card he has obtained, the frequency of his transactions would likely to look a lot different than the other regular users.

Finally, we concluded the feature engineering phase by oversampling our target variable "is_fraud". We used SMOTE (Synthetic Minority Oversampling TEchnique) to address the imbalanced distribution between fraudulent and non-fraudulent transaction data points. This means that once the minority class in the dataset is identified—in our case, the fraudulent transaction data points—then for each instance in the minority class, SMOTE considers its k-nearest neighbors and generates synthetic instances that are combinations of these neighbors. This not only prevented our models from over-fitting to the predominant non-fraudulent transaction class but also enabled the balanced dataset to explore the feature space of the minority class, thereby enhancing both the performance and robustness of our models. **Figure 2** shows that fraud transactions only accounted for 0.6% of the target variable which was severely imbalanced, but after synthetic minority oversampling, the positive and negative classes (fraud and non-fraud, respectively) were balanced.

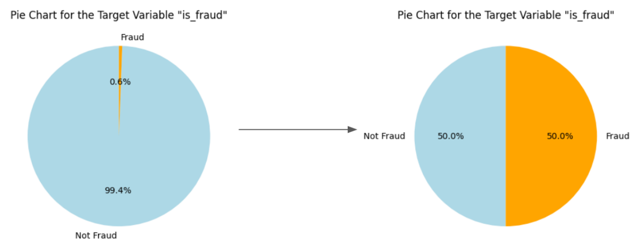

Figure 2: Using SMOTE to address data imbalance

3.3 Modeling

We trained 4 different models using the famous Python package called Scikit-Learn and tried to compare their performances. After oversampling, the training data became even larger in its size with 2,578,338 rows. Running models with high complexity would require computing power beyond our ability. To speed up the modeling process (ensuring that the time used for fitting the data for each model is within 30 minutes), we reduced the size of the training data by randomly sampling **5%** of the original size without replacement. To make sure that the testing data did not contain a lot more rows than the training one, we also reduced the size of the testing data to **10%** of its original size—thus, the sample training data had 128917 rows, and the sample testing data had 55572 rows. These were still large datasets and should be sufficient for modeling.

We made logistic regression as our baseline model. Logistic regression estimates the probability of a binary outcome based on one or more predictor variables. It is one of the most classic and fundamental statistical models for modeling binary classification problems. Since we are using the default parameters, this means that our logistic regression model includes a L2 penalty term, a stopping criteria of 0.0001, and a moderate inverse of regularization strength (C) of 1.0. We also set the maximal number of iteration to 1000, which allows us to stop the model even if it failed to converge within 1000 iterations.

We built a support vector machine (SVM) model. SVM is a model that's been widely used in prior works related to fraud transaction detection. To be specific, we used a radial basis function kernel (RBF) with a stopping criteria of 0.001, and a moderate inverse of regularization strength (C) of 1.0. If SVM turns out to have good performance on the dataset, that means that there exists a clear "margin" between fraudulent and non-fraudulent transactions, even through the "margin" itself may lie in a higher dimension and thus not visually interpretable.

We used 2 tree-based algorithms, random forest and Adaptive Boosting (AdaBoost). A random forest model uses a series of decision trees to keep dividing the overall dataset into smaller subsets where each tree would make predictions based on only a subset of all the features in the training data. Our random forest model contained 100 estimators, which means that 100 trees would be generated in the "forest", and we used Gini impurity score to compare the importance across variables, which is also the default choice of Scikit-Learn's Random Forest Classifier.

Meanwhile, AdaBoost is a machine learning model that improves the performance of models by combining multiple weak learning models sequentially, where each successive model attempts to correct the errors of its predecessor. In this case, we used 50 estimators and a learning rate of 1.0 to construct our AdaBoost model. Our exploratory data analysis showed that most variables do not have a strong linear correlation with the target variable, so we hope to see if we can capture nonlinear relationships between variables (especially those count-encoded variables transformed from categorical ones) through training tree-based models.

4 RESULTS

We first trained the four models with the parameters specified above. The confusion matrices of the models are shown in **Table 2**. It is noticeable that the confusion matrices for all four models are not too far away from each other. Besides, due to the limitation of our dataset (the imbalance between positive and negative classes of our target variable), we do have only 216 data points that represents real fraudulent transactions accounting only 0.389% of the 55572 transactions in the testing set.

Our baseline model, the logistic regression model, achieved an accuracy of 94.9% with a precision of 5.71% and a recall of 77.3%. Support Vector Machine performed the best without tuning as it achieved a 97.7% accuracy—2.77 percentage point higher than the baseline model, and it also recorded the highest precision score among all four models we used (**11.4%**) along with a recall of 73.61%.

Table 2: Confusion Matrix Across Models, before Tuning

Model	TN	FP	FN	TP
Logistic Regression	52597	2759	49	167
SVM	54116	1240	57	159
Random Forest	51004	4352	19	197
Adaboost	51862	3494	27	189

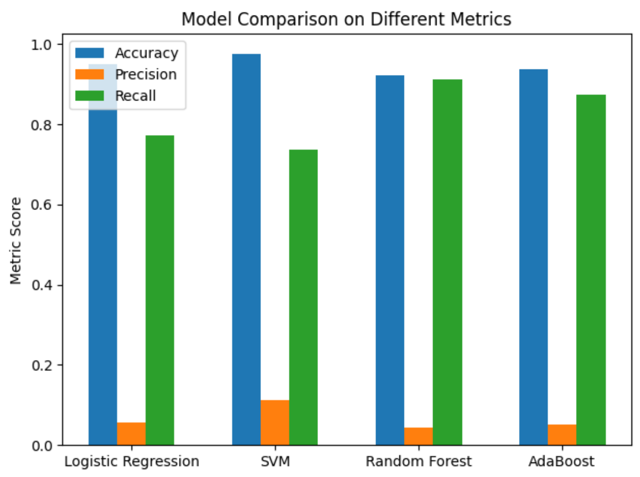

Figure 3: Comparison in Model Accuracy, Precision and Recall

Random Forest performed worse than the baseline on accuracy (92.1%), but also recorded the highest recall among all four models (**91.2%**) along with a precision of 4.33%. AdaBoost's performance was very mediocre, achieving an accuracy of 93.7% with a precision of 5.13% and a recall of 87.5%. A bar plot highlighting the differences in performance between these models is shown in **Figure 3**.

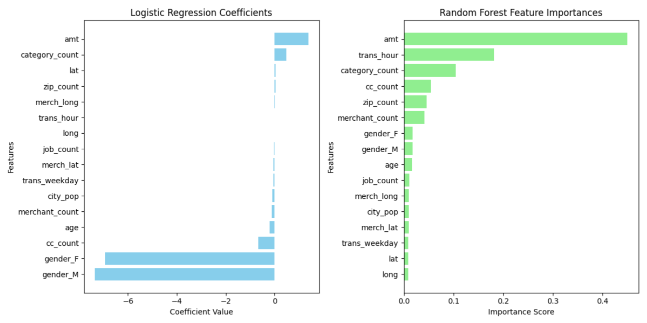

Figure 4: Feature weights in Logistic Regression and feature importance in Random Forest Model

We also extracted a bar plot **(Figure 4)** to compare the weights of each feature in our baseline logistic regression model and feature importance in the Random Forest model. A few variables such as "amt" (transaction amount), "category_count", "zip_count" showed

Table 3: Grid Search Parameters for Random Forest

Random Forest Parameters			
n_estimators	50	100	200
max_depth	None	10	
min_samples_split	2	5	
min_samples_leaf	1	2	

relatively high importance across both models. Additionally, as expected, Random Forest was able to better utilize the count-encoded variables, such as “category_count”, “cc_count”, “zip_count”, and “merchant_count” due to the nature of the tree-structured algorithm.

In short, SVM and Random Forest stood out during the modeling phase by using mostly default parameters in Scikit-Learn. We noticed that with such a large training set and the nature of high complexity of SVM’s algorithm (especially with the RBF kernel), it was almost impossible to do any parameter tuning without spending hours and hours training each model. Hence, we decided to only tune the model on Random Forest, which was a little more efficient in computing time compared to SVM. We sampled the training data again without replacement, leaving the training set only 1% of its original size (around 120,000 rows), then used grid search and 5-fold cross validation to help us find ideal parameters for the Random Forest Model. A list of possible parameters that was passed in to the grid search function can be found in **(Table 3)**. “n_estimators" specified the number of trees we need; “max_depth" specified the maximum depth of each tree—by default, it was set to “None", which might cause very deep trees leading to over-fitting. “min_samples_split" tells the model the minimum samples it needs to split a new node, while “min_samples_leaf" constrains the minimum number of samples a leaf node needs to have.

After using grid search, the best list of parameters we found was: “n_estimators" = 100, “max_depth" = None, “min_samples_split" = 2, “min_samples_leaf" = 1, which was exactly the same as the default parameters provided by Scikit-Learn! However, the model trained on less data resulted in an increase in all three metrics—the new Random Forest model with the best parameters recorded an accuracy of 94.9%, a precision of 6.60%, and a recall of 92.1%. A new plot comparing the other three models with this tuned Random Forest model is shown in **(Figure 5)**. SVM still has the highest overall accuracy and precision, but Random Forest has a close accuracy and much higher recall.

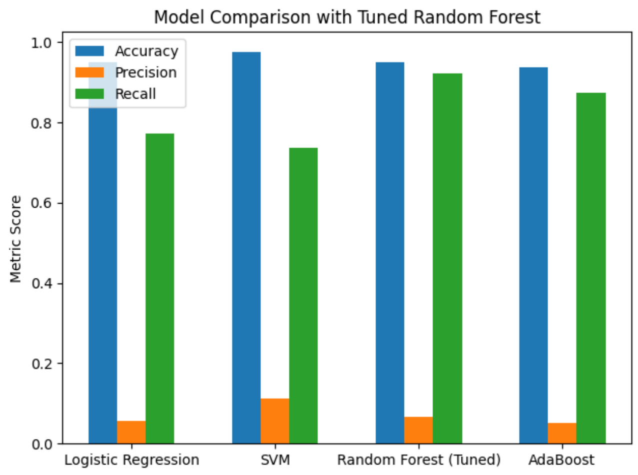

Figure 5: Comparison in Model Accuracy, Precision and Recall with Tuned Random Forest

5 DISCUSSION

Before we invested our time into tuning the hyper-parameters, SVM and Random Forest performed the best when taking all three metrics into consideration. The SVM model had the highest accuracy on the testing dataset among all four models we used. However, training the model took more time than the others, and the result can be hard to interpret and visualize the model in higher dimensions. Meanwhile, the Random Forest model performed well in that it built a good balance between metrics. Besides, as shown in **Figure 4**, it was able to capture information contained by the variables created via count-encoding. Another takeaway is that after 1 hour of tuning using grid search, the default parameters were still the best, and they performed even better using a smaller training set.

From the results above, we can argue that different models may have uses towards different goals. SVM had the highest precision score, meaning that it could avoid the situation where a person does a normal purchase but is identified as fraud somehow and the purchase is thus cancelled. On the other hand, Random Forest had the highest recall score, indicating that it is the best at avoiding False Negatives, i.e. the situation where the fraud action happens but is regarded as a normal transaction. One can argue that the latter is more important because it will actually cause people to lose a lot of money without being properly notified and thus Random Forest is the better solution here, but in the end it all depends the purpose of building such models—one can also use SVM as the main model if their priority is to protect customer experience and the convenience of the cardholders.

6 CONCLUSION

In conclusion, in this project we utilized a simulated dataset to demonstrate the entire machine learning pipeline from data cleaning and pre-processing, exploratory analysis, feature engineering, modeling, and visualization of the results. Throughout the project we have shown different techniques to transform different types of variables related to credit card transaction, and built different models aiming to identify fraud transactions. To summarize, classic machine learning models like Support Vector Machine and Random Forest are still very powerful tools to help detect unusual patterns in credit card transactions, and each of them have their own advantages. Therefore, it is wise for any fraud detection department inside a bank or other financial institutions to always try different approaches to solve this problem based on different business needs so that cardholders can feel safe using their money.

7 CONTRIBUTION

Most parts of the work, from data collection to modeling, were completed by both members in the team. A specific breakdown of our contributions to this project is listed below:

* Apple: Data Cleaning, Exploratory Data Analysis, Feature Engineering;

* Derek: Feature Engineering, Modeling & Parameter Tuning, Visualizations.

Again, this does not mean that the other member did not participate in any part of the project that is not listed in the breakdown. It simply shows the parts we each focused on.

8 GITHUB LINK

Below is the GitHub link to our repository, where we store the Python code and notebook used to generate all the results above:

https://github.com/Applelxc/dsci-303-final-project.git

REFERENCES

[1] Vaishnavi Nath Dornadula and S Geetha. 2019. Credit Card Fraud Detection using Machine Learning Algorithms. *Procedia Computer Science* (2019).
[2] John Egan. 2023. Credit card fraud statistics. *Bank Rate* (2023).
[3] Kartik Shenoy. 2023. *Credit Card Transactions Fraud Detection Dataset.* Retrieved November 28, 2023 from https://www.kaggle.com/datasets/kartik2112/fraud-detection/data