Credit Card Fraud Detection

Machine Learning Project - Phase 5

Malak Gaballa
900201683
Data Science
The American University in Cairo
malakhgaballa@aucegypt.edu

Masa Tantawy
900201312
Data Science
The American University in Cairo
masatantawy@aucegypt.edu

Introduction

In today's world, the use of technology has become an inextricable part of daily life. This is particularly visible in the financial sector due to the rise and prevalence of multiple non-cash payment methods and the most popular one being via the use of credit cards. In order to ensure the effectiveness and usefulness of this technology, its safety must be extremely high with very minimal risks such as fraud. Hence, the aim of this project was to develop a credit card fraud detection system based on the best-performing machine learning algorithm that can be applied at financial institutions that issue cards, such as banks, or any payment methods of the same nature to monitor transactions and flag fraudulent ones.

The dataset used, obtained from Kaggle, consists of simulated transaction from 1-Jan-2019 to 31-Dec-2020 based on real transaction of 1000 customers with 800 merchants. It intitially divided into a training and a testing set, which was combined together for use, of 1,852,394 instances in total. It also initially contained 22 features, 12 categorical and 9 numeric, and a binary label for each transaction indicating if it is fraudulent or not. The dimension of the prepared dataset is 1,851,959 instances and 85 features. This change is due to preprocessing which included identifying outliers, eliminating redundant or irrelevant columns, encoding categorical variables or mapping ones with a large number of classes to fewer ones and other steps.

To isolate the chosen best-performing machine learning algorithm, several models such as linear and logistic regression, kNN, multi-layer neural networks and many others were tried on the dataset, which was approximately split into 75%-25% train-test data. These models were evaluated based on the specificity and the recall of the fraudulent class due to the extremely unbalanced nature of the data. Among them, the top 3 models were Random Forest, Neural Networks, and Decision Trees. When checking for overfitting using 5-fold and 10-fold cross validation, it appeared that the best model was Random Forest.

In this phase, this chosen model was adjusted, and hypertuning of the parameters took place to refine the model. A detailed description of all the adjustments made will be explained in the following section. In addition, a GUI utility application was designed to facilitate the process of classifying new transactions entered by the user. The details of this application will also be described below.

1. Model Design and Architecture

After undergoing cross validation and choosing random forest using the entropy criterion as the best model in the previous phase, this model has to be refined in order to optimize its performance in detecting fraudulent transactions. Even though the model will be evaluated mainly on the recall and specificity due to the high imbalance in the data, it is important to monitor other metrics such as the precision of the non-fraudulent class throughout the hypertuning process in order to examine how the parameter adjustments will affect these other measures. If there was a negative impact such that the precision, for example, significantly decreased for the legitimate transactions, then the model should be refined once again in order to maintain the accuracy in detecting both classes of transactions. This will be done by comparing the classification reports of the initial model and the adjusted model at the end to see the alteration of the performance measures.

All the tables mentioned can be found at the appendix at the end of this document.

1.1 Initial Model

Before hypertuning the parameters, the initial model's performance was set as the standard performance benchmark that the refined models will be compared against. The initial model consists of only 2 parameters set to the not default values, which are the criteria that measures the quality of the split (*criterion)* and a parameter called *random_state* which sets the seed in order to control the randomness, producing the same results and model performance across different calls; these parameter values were set to *entropy* and *1* , respectively. This model yielded a specificity value of 0.9999 and a recall value of 0.7407. Thus, these metrics will be the standard benchmark for comparison and since the specificity value is already high, the main focus will be to increase the recall measure for fraudulent transactions as much as possible.

1.2 Parameter Adjustments

Random State

In previous phases, the performance varied throughout the calls even though the command was fixed. Thus, to fix the results yielded by the model, the parameter *random_state* had to be utilized in the function to make the model easier to replicate for future purposes. A for loop was constructed from 0 to 99 in order to find the seed that yields the highest recall value for the model and start improving the model afterwards. There was no specific pattern in the recall and it was fluctuating as the seed increased. When the seed was set to 50, the recall reached its highest value of 0.746. Therefore, the value of 50 was fixed in this parameter and will be implemented when the rest of the parameters are adjusted.

Number of Estimators

After controlling the randomness of the model, the number of decision trees in the forest had to be specified. The initial model had 100 trees in the forest by default. A loop was constructed over the numbers from 1-100 in order to see if a lower number of trees would return a higher recall for fraudulent transactions. The recall fluctuated over the numbers, but 65 trees yielded the highest recall of 0.754. Thus, the parameter *n_estimators* was set to 65 trees in the forest.

Maximum Depth

The maximum depth for the 65 trees had to be set since the depth is unlimited by default which would make each tree contain an unlimited number of leaf nodes, possibly overfitting. A loop over the first 100 numbers was generated, and the recall started to increase across the first 20 values until it started to fluctuate between 0.71 and 0.755. The recall reached 0.7552 when the depth or number of splits was set to 57.

Number of Jobs to run in parallel

This parameter tells the engine how many processors it is allowed to use. A value of -1 means there is no restriction whereas a value of 1 means it can only use one processor. It exploits all the CPUs available on the local computer in order to speed up the training process. A loop was constructed from -2 to 3 and the recall was 0.7552 for all of them which shows that the number of processors run in parallel doesn't affect the performance of this model. Hence, the parameter will not be refined and will be set to its default value.

Minimum Samples Split

When tuning the value of the minimum number of samples required to split an internal node, the value of the parameter was looped over from 2, the minimum, to 100. As the minimum number of samples increased, recall decreased drastically from 0.7552 to 0.66. Therefore, the parameter value was set to its default value where a minimum of 2 samples are required to split an internal node.

Minimum Samples at Leaf Node

The parameter that selects the minimum samples required to be at a leaf node was also looped over (1-100) in order to see whether or not increasing the minimum number of samples would increase the recall along with it. Similar to the *min_samples_split* parameter, the recall decreased from 0.7552 to approximately 0.47 as the parameter value increased. That being said, the value of this parameter was left as its default value which dictates that 1 sample is the minimum number of samples required to be at a leaf node (*min_samples_leaf=1*).

Maximum Number of Features considered for split

Since there are 83 features in the dataset, limiting them to be considered for each split would create a better performing model. So a loop was constructed from 1 to 83 to determine whether or not limiting the amount of features would affect the model recall and overall performance. When the parameter *max_features* was set to 64 features, recall jumped from 0.7552 to 0.8325 which shows that this parameter in particular affected the performance of the model the most since it had the highest impact on the recall and the detection of fraudulent credit card transactions.

Class Weights ({1:3})

Because the data is extremely unbalanced and the number of non-fraudulent transactions is almost 190 times the fraudulent transactions, weights had to be assigned for the classes. Multiple combinations were studied in the form of *{non fraudulent label (0) : weight,fraudulent label (1) : weight}* and the recall was highest when either both class labels had weights equal to one or when the weight of the fraudulent transactions was 3 times the weight of the legitimate or non fraudulent. Even though both weight combinations yielded a recall value of 0.8325, it is more reasonable to assign greater weight to the fraudulent than to the non-fraudulent due to the data imbalance as previously mentioned. Thus, the weights were assigned as follows {0:1, 1:3}.

1.3 Numeric Columns Binning

Another attempt to enhance the model by increasing the recall of the fraudulent class is by checking the binning of the numeric features, which are 9 features. In *sklearn*, this is done automatically in the model; the values of the bins created were tried to be extracted however due to the extremely large dataset size and number of features, a single tree appeared to be very complicated. In another attempt to study the numerical features initial distributions in order to determine the binning, the features were denormalized to revert them to their initial distribution. Next, the index plots were constructed as well as their histograms. However, these plots did not show any particular or unique characteristic. For the scatter plots, all graphs showed a random scatter of points with no clear pattern; the histograms displayed that most distributions almost followed a normal distribution, which was sometimes skewed. This supports the central limit theorem due to the large number of instances in the data. It is important to note that the autobinning done by the model internally already yielded very high recall and specificity values. Thus, given the model's performance and the randomness of the index plots, the autobinning done by the model will be used for the features.

1.4 Final Model Random State

After refining the parameters, setting them to the values that contributed to yielding the highest recall value for fraudulent transactions, and examining the binning process of numeric columns, the random state parameter was looped over once again (0-99) to study if other seeds would yield higher recall values or not. Consistently, the parameter value 50, as stated earlier, still yielded the highest recall. Hence, the random state parameter will be fixed to 50.

1.5 Model Comparison (Initial vs Refined)

Now that the parameters have been hypertuned, it is essential to compare the tuned model to the initial model implemented in order to understand how the modifications impacted the other performance measures such as the precision of non-fraudulent transactions. The initial model and the refined model were reproduced along with their classification reports, displaying the effect of the modified parameters on all metrics. The precision of the non-fraudulent class is expected to decrease, for the weights of the fraudulent transactions is triple that of the legitimate; however, the decrease should be minor since the majority of transactions are legitimate.

For the initial model, the precision, recall and specificity were 0.98, 0.74 and 0.99993 for the fraudulent transactions, respectively. This indicated that the model was able to classify the legitimate transactions but struggled to classify the fraudulent since the recall was less than 0.75.

After adjusting the parameters, the changes in the classification report metrics had to be observed. The precision, recall and specificity for the fraudulent transactions were 0.96, 0.83 and 0.99997 respectively; the precision slightly decreased, by 0.02, while the specificity experienced a minor increase from 0.99993 to 0.9997. The Recall, on the other hand, significantly jumped by 0.0925 (from 0.74 to 0.8325). These metrics indicate that the

refined model was able to classify both fraudulent and non-fraudulent transactions correctly compared to the initial model. Therefore, it is safe to say that the refined model will be utilized in the application as the final model.

1.6 Final Model

After numerous steps to reach the the random forest model with the optimal performance, the parameters were fixed to values that yielded the highest specificity, 0.9997, and recall, 0.8325, for the fraudulent transactions. The final parameters in the *RandomForestClassifier()* command are shown in the table below.

Parameter	Value
criterion	entropy
n_estimators	65
random_state	50
max_depth	57
n_jobs	None (Default)
min_samples_split	2 (Default)
min_samples_leaf	1 (Default)
max_features	64
class_weight	{0:1,1:3}

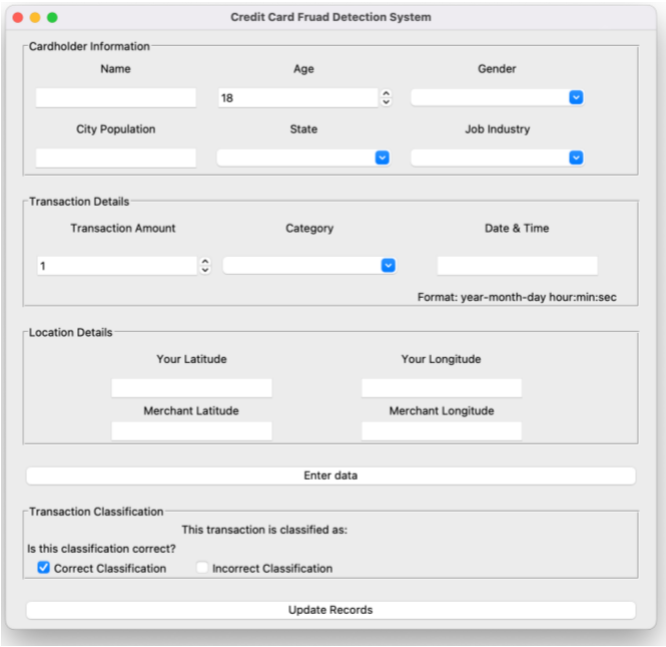
2. Utility Application

As mentioned in the previous phase, utility application was implemented to allow the user to enter the details of the transaction to be classified as either fraudulent or legitimate. This simple GUI (graphical user interface) was created using the *tkinter* python package. The application's user interface consists of a single window that is divided into 4 section shown below. The first collects the cardholder's information: name, age, gender, city population, state, and job industry; the second focuses on the transaction details: amount, category, and date and time; the third allows the user to enter the location details which are the latitude and longitude of the cardholder and the merchant. This is followed by an 'Enter data' button which displays the transaction's classification after being clicked on. The classification is printed on the screen either as fraudulent or not fraudlent in the last section transaction classification. To ensure that the classification was correct, the user checks one of 2 boxes: correct classification, which is checked by default, or incorrect classification; these 2 checkboxes cannot be ticked together, so when one is ticked the other is automatically unchecked. Finally, 'Update Records' button stores the transactionif it was incorrectly classified and retrains the model on the new data as well as displays a thank you message for the user.

The first step is to preprocess the entered data in order to match the format of the rest of the data on which the model trains and predicts. The function *preprocess* is called inside the app after the user presses on 'Enter Data'. Primarily, a single-row dataframe of zeros is created with the same columns as the prepared dataset (training and testing of previous phases) except for the *is_fraud* feature named *new_trans*. This is followed by all the preprocessing steps done during the data preparation phase. The date and time entered in datetime format obtained as a string is converted a datetime object, split in year, month, day, hour, and minute then stored in the dataframe. As for the latitudes and longitudes, they are used to compute the distance between the cardholder and the merchant via the *haversine* package. The computed distance, the 4 features of the transaction time and date and all the numerical features, amount, age and city population, are normalized using the mean and standard deviation of all the initial dataset (*New CCtransactions)* and strored in *new_trans*. The categorical features, gender, state, job industry, and transaction category, are one-hot encoded based on the values chosen from the dropdown menues and stored. It is worth noting that the name is not used part of the used data and is added to the app only for user-friendliness as well as that the user directly chooses their job industry unlike in the acquired data where each job was mapped to an industry.

Now, the data has been preprocessed, so a function *classify* is called to yield the model's classification. Initially when the app is first opened, the model is already trained on all

the existing data so when the function classify is called, it directly returns the predicted class of the transaction based on the features. According to the user's feedback, the model might be retrained. Specifically, a function *update* is called when its repective button is clicked. This function keeps the classification as it is if the user checks correct classification, or it changes the class and displays the new one if it was incorrect. Furthermore, in the case of incorrect classification, this new transaction with its correct class is added to the full initial dataset csv file (*New CCtransactions.csv)* and the model is retrained for improvement. This active learning process ensures that the model's prediction is more likely to be correct in the future as it increases the trainining data.



Conclusion

To conclude, the aim of this project was to design a credit card fraud detection system based on the best-performing machine learning algorithm that can be applied at financial institutions that issue cards, such as banks, or any payment methods of the same nature to monitor transactions and flag fraudulent ones. This project were divided into 5 phases allowing previous research on this topic to be studied, the data to be prepared and analyzed, multiple models to be tested, the final model to be designed and refined as well as a user-friendly utility app. In this study, the best model that can classify whether or not a credit card transaction is fraudulent is the random model as it optimizes the model performance evaluated based on the specificity and the recall of the fraudulent class due to the extremely unbalanced nature of the data. It appeared in this phase that the maximum number of features considered for split had the most promiment effect on the model's performance due to increasing the recall the highest from all parameters, allowing better flagging of fraudulent transactions. As for the utility app, it allows the user to enter all the require fields, either by typing or choosing from one of the options in the dropdown menus, then displays the classficiation of the entered transaction according to the model's output. This is done after all the inputs are preprocessed and adjusted. The app also allows the user to provide feedback on the output, and if the classification was incorrect, the transaction is added to the instances and the model is retrained for better performance.

References

[1]
Credit Card Transactions Fraud Detection Dataset. *Kaggle*. Retrieved February 13, 2022 from https://www.kaggle.com/datasets/kartik2112/fraud-detection?resource=download