Big Data Solution for Credit Card Fraud Detection

Sourav Gupta
MSc Computer Science
Univeristy of Nottingham
Nottingham, UK
20498840
Psxsg12@nottingham.ac.uk

Arnav Saxena
MSc Computer Science
Univeristy of Nottingham
Nottingham, UK
20444748
Psxas23@nottingham.ac.uk

Aphichek Nagore
MSc Computer Science
Univeristy of Nottingham
Nottingham, UK
20494598
Psxan2@nottingham.ac.uk

Jirat Manpadungkit
BSc Computer Science with AI
Univeristy of Nottingham
Nottingham, UK
20320731
Scyjm5@nottingham.ac.uk

Detecting fraudulent credit card transaction is of the utmost importance is today's digital world. In this paper we employ two machine learning algorithms – one tree based (Gradient Boosted Trees) and one non tree based (Logistic Regression) to detect whether a transaction is fraudulent or not. We compare the AUC ROC score of these two models after every step of the ML process, i.e., before pre-processing, after feature selection, after feature engineering and after hyperparameter tuning. The steps are designed with the intent to improve the dataset for better fraud detection accuracy. Big data libraries Pyspark and MLlib are used exclusively to build this project as the dataset is quite large. Our results show that while both models give a high AUC ROC score, GBT has an edge over Logistic Regression on imbalanced datasets.

I. INTRODUCTION & BACKGROUND

According to [9], credit card fraud is defined as, "when an individual uses another individuals credit card for personal use while the owner of the card as well as the card issuer are not aware of the thing that the card is being used." An accurate fraud detector is extremely important in modern society as most of our transactions happen through a medium such as a bank. A banking system holds great responsibility when it comes to maintaining safe transactions, failure to do so may result in tarnish to reputation, financial loss and even pending lawsuits.

Traditionally, banks used manual monitoring to detect fraud, which was extremely time consuming and expensive. In the status-quo, machine learning techniques are utilised to detect such fraudulent activities and as there are over a billion transactions happening every single day, the amount and velocity of data becomes so large that a big data solution is required.

A lot of work is done in this field using non distributed ML solutions. For example, [10] looks at comparison between naïve bayes, logistic regression and KNN classifier to tackle this issue. [11] uses Random Forest to find fraudulent transactions. Even GBT is implemented using Python's scikit-learn library here [12].

The key to effective fraud detection is quick identification of suspicious transactions to minimise damage. With banks receiving large amounts of data in a single day, a big data solution is required to process all the data as quickly as possible to take actions. That's why our work provides a distributed big data ML solution approach to this problem. The implementation of our solution consists of leveraging Apache Spark's Pyspark, MLlib and Databricks to detect whether a transaction is suspicious or not. These technologies are utilised for the machine learning operations because of their ability to use distributed computing to do large scale big data processing. Furthermore, the goal of our project is to investigate the change of fraud detection accuracy throughout different machine learning steps in our process.

The data used to train our models is from Kaggle's IEEE-CIS Fraud Detection competition. The data contains numerous features. This competition had countless submissions, so we had very good references on how to approach this topic. Many articles described exactly how they cleaned their data and their hyperparameters for each ML technique. Others were very insightful in showcasing interesting correlations they found in the dataset while exploring it and what each feature means. Though these solutions are extremely valuable, they were all implemented using scikit-learn and not using big data methods.

The next section looks at the methods used to explore, clean, and pre-process the dataset. Section III looks at the experimental setup designed to acquire the scores after each step in the ML process. In section IV we discuss the results in depth. In section V, the limitation of our approach is discussed and finally we conclude our findings in section VI.

II. PROPOSED METHODOLOGY

This section presents the methodology employed to implement the model, and the rationale behind selected actions. The datasets are first loaded into Databricks from the Azure blob storage container using spark.read() command.

A. *Dataset*

The data provided on Kaggle consists of four csv files: train_transaction.csv, train_identity.csv, test_transaction.csv, test_identitfy.csv. The description of each dataset was minimal, leaving many columns unknown. Thankfully, the categorical columns were given. This made it easy to separate between numerical and categorical features. The column we are predicting is called isFraud, which only has 2 unique elements making this a binary classification problem. Since the datasets were connected by ***TransacionID,*** the first obvious step was to merge the train datasets together as well as the test datasets. This was done so that we just have one training dataset and one test dataset. The shape of the training dataset is (590540, 432) and the shape of the test dataset is (506691, 432). The important columns and their description as per Kaggle are as follows [5]:

- TransactionDT: timedelta from a given reference datetime (not an actual timestamp)
- TransactionAMT: transaction payment amount in USD
- ProductCD: product code, the product for each transaction
- card1 - card6: payment card information, such as card type, card category, issue bank, country, etc.
- addr: address
- dist: distance
- P_ and (R__) emaildomain: purchaser and recipient email domain
- C1-C14: counting, such as how many addresses are found to be associated with the payment card, etc. The actual meaning is masked.
- D1-D15: timedelta, such as days between previous transaction, etc.
- M1-M9: match, such as names on card and address, etc.
- id_1 – id_38: identity information parameters.
- Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations.

B. *EDA*

Merging is followed by an exploratory data analysis to gain a better understanding of the data. Using visualizations and statistical analysis, certain observations were made which are given below. After preliminary exploration, logistic regression and gradient-boosted tree algorithms were fitted on the train dataset. The score on the transformed dataset was recorded. This was repeated three more times - after dropping unimportant features, after feature engineering and finally after hyperparameter tuning. The performance of these models was evaluated using the area under the Receiver Operating Characteristic curve. The observations/changes made from data exploration are as followed:

- The dataset is extremely imbalanced, 96.5% of the rows were non-fraudulent transactions (isFraud = 0).
- The feature ***addr2*** is the same for almost 99% of rows so the hypothesis is that this feature must correspond to the country code.
- There is a group of columns in the test and train dataset that are supposed to have the same name but were named incorrectly. Specifically, in the test dataset all this group of columns were named “id-x” while in the training set, they were named “id_x”, therefore we standardised all the columns to ***id_x***.
- There were large amounts missing values, a lot of preparation must be done before actually using the data to train our models.
- All the columns beside categorical columns were casted into float, so that they are in the format that is suitable for training.
- Certain V-features columns had the same number of missing values, indicating that they are highly correlated.
- Fraudulent transactions were done more frequently during specific times of the day.
- All rows are distinct in the train and test dataset.
- TransactionDT column contains the time elapsed in seconds from a reference date.

Figure 1 Shows how imbalanced the dataset is.

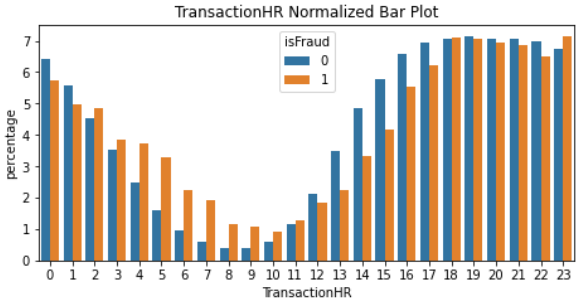

Figure 2 Shows the interesting relationship between the time of transaction and fraud.

Graph is taken from [1]

C. *Dropping Features*

Firstly, columns with 90% missing values were dropped, the reason for why such a high threshold was chosen is because our goal is to detect anomalies. If we drop

features with 60% or more missing value like normal, a great amount of information will be lost. Hence to prevent loss of information a high threshold is used. Secondly, columns with only one unique value are dropped because they do not provide any additional value to the dataset. Lastly, C columns, ID columns, D columns and V columns were checked for correlation, this is because the meanings of each column were not stated. Therefore, we need to find which columns are highly correlated so that they can be dropped as they don't provide more value to the dataset.

The C, ID and D columns turned out to not have any correlation amongst themselves, thus their columns were not dropped. [1] But the V-features which have 339 columns in total contain many features that are highly correlated. Certain V columns had similar numbers of missing values, so we decided to group them together. A correlation test is done to each group of V-features to find the column that is the more than 0.95 correlated to the other columns in its group. A subgroup is constructed with these features. In this subgroup, the feature which has the greatest number of distinct values is chosen as the representative of that group and the rest are dropped. The rationale is that the more distinct values a column has, the more value it adds to the dataset. After features were dropped, missing values in the numerical columns were imputed with -999 and categorical columns were imputed with 'missing'.

D. Feature Engineering

Two major features were added in the dataset.[1] Firstly, Transaction Hour was calculated using the Transaction DT column. The justification for doing this is that fraudulent transactions occur more often during certain times of the day. To have specific columns indicating the minutes, hours and days will make our algorithm learn these patterns more effectively. The second feature engineering done is adding a unique identifier (UID) column. This is done because, once a credit card is marked as fraud, it will not be used again, i.e., once a card is involved in a fraudulent transaction, all the transactions that were made by that card will be marked as fraudulent. Hence, the objective is to predict fraudulent clients, not fraudulent transactions, and the way to do that is to identify unique cards in the dataset. Columns card1, card2, card3, card5, card6, addr1 and P_emaildomain are combined into a single feature as it acts as an identifier for a particular card-client combination.

E. Machine Learning Algorithms

Our project wants to compare a tree-based and a non-tree-based algorithm therefore Logistic Regression and Gradient Boosted tree algorithm were chosen. Logistic Regression was chosen because of its simplicity to implement, ability to resist overfitting and training quickness. The algorithm is also robust against missing values and outliers, in which this dataset has a myriad of. It is important to prioritise preventing overfitting and deal with missing values effectively because the bigger the dataset, the more these issues cause problems **[7]**. With the quickness in training, more time can go into tuning the hyperparameters. Gradient-Boosted tree algorithm is chosen as the tree algorithm because it has a special property that works well with imbalanced datasets. It also produces more accurate results than Logistic Regression because it is more capable of capturing complex patterns. However, they are prone to noisy data, thus hyperparameter tuning and cleaning the data properly is imperative **[8]**.

F. Evaluation Technique

The chosen performance metric used in this project will be ROC-AUC or the area under Receiver Operating Characteristic curve. This metric is preferred over accuracy when it comes to highly imbalanced datasets. This is because a model trained on such a dataset can wrongly predict all the transactions as non-fraudulent (majority label) but still get a high accuracy score.

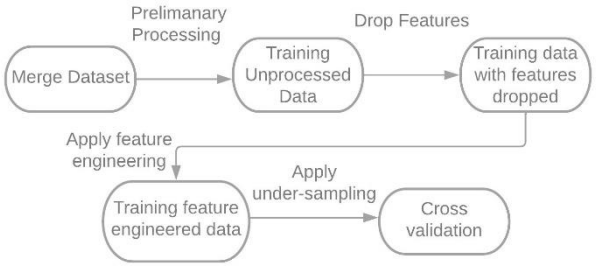

Figure 3 This image displays the flow of actions in the investigation.

III. EXPERIMENTAL SET-UP

The experiment is broken down into four parts, firstly training the models on unprocessed data, secondly training on data after dropping its features, thirdly training on data after feature engineering and finally performing cross validation to determine the best hyperparameters. Gradient Boosted Trees and Logistic Regression are the models chosen for each of these stages. At each part, after training on the train dataset, the model is used to transform the test dataset. This produces a TransactionID and a probability column. The former represents a particular test transaction row while the latter is the probability for whether this transaction is fraudulent or not. To find the AUC ROC score we downloaded this dataframe of two columns using the display () function. The csv file is submitted on Kaggle, and the scores are noted.

A. ML Part 1 – Training Unprocessed Data

In this section the partially clean data is used to train the model. An ML pipeline model is created which includes StringIndexer, OneHotEncoder and VectorAssembler. The first two functions are used to encode categorical columns to numerical values. The vector assembler is finally used to create the features vector column. The two machine learning algorithms then train and transform the data and the corresponding csv files are produced.

B. ML Part 2 – Training Data after Dropping Features

In this section, several columns are dropped from the train and test dataset. The names of categorical and numerical columns are updated to reflect this change. Another ML pipeline model is created which consists of the same stages as in ML Part 1. The data then goes through the pipeline and is trained and transformed using the two algorithms. The csv files are downloaded and saved.

C. ML Part 3 – Training Data after Feature Engineering

In this section, two important features are added. One is named "New_Transaction_Hour" and contains the hour of the day (a number between 0 and 23) for each transaction row entry. The second is named "uid" and contains the unique identifier for each transaction row entry. The names of the categorical and numerical features are updated and the above steps in ML Part 1 and Part 2 are repeated.

D. ML Part 4 – Hyperparameter Tuning

Up till now we were using the two algorithms with an initial set of hyperparameters. Once all the processing and cleaning is done, we should tune the parameters to increase our AUC ROC score on Kaggle. This is done by first creating a train and validation dataset. The original train dataset is under sampled first and then Pyspark's TrainValidationSplit is applied to both the models to find their best parameters. This model is then used to transform the test dataset and the corresponding csv files are downloaded and saved.

IV. RESULT AND DISCUSSION

The csv files that were downloaded in the experimental step were uploaded on Kaggle and the AUC ROC score obtained were noted down. Table 1 depicts these results.

TABLE 1	Logistic Regression	Gradient Boosted Tree
ML part 1	0.825302	0.859121
ML part 2	0.817832	0.866290
ML part 3	0.836921	0.866546
ML part 4	0.848193	0.875234

Figure 4 This table shows the results of the 2 models, throughout different stages.

A. Logistic Regression

Logistic Regression is run with all default settings except the iteration's parameter (maxIter is set as 10). We see a slight dip in the score after dropping features. This could be due to information loss or parameter mismatch. Dropping unimportant features resulted in reducing the size of the dataset by half and hence this benefit greatly outweighs the slight decrease in the score. We then see an increase after feature engineering and hyperparameter tuning. The tuned parameters are: maxIter = 10, regParam = 0.01 and elasticNetParam = 0.0.

B. Gradient Boosted Trees

Gradient Boosted Trees is run with all default setting except the iteration's parameter (maxIter is set as 5). As seen in the table, we see increases in the score after each part. GBT outperforms Logistic Regression throughout the experiment. It clearly works better with imbalanced datasets. After hyperparameter tuning we reach the highest score possible for this model. The tuned parameters are: maxIter = 5, maxDepth = 7, maxBins = 32.

V. LIMITATIONS

Creating an accurate subset of the highly imbalanced data for cross validation was our biggest hurdle. Since MLlib is a relatively newer library than Scikit-learn, it did not have a function to generate stratified k folds of the data. We settled on using the under-sampling approach.

Furthermore, due to time constraints, we couldn't implement a backward feature elimination and a forward feature selection function. This would have helped us to boost our scores.

VI. CONCLUSIONS

In this paper we looked at a popular real-world problem – credit card fraud detection. We implemented two machine learning algorithms using Apache Spark libraries to solve this issue, thereby using distributed computing. We then looked at the AUC ROC score after every step in a ML process for the two algorithms and compared their results. For our future work we would like to implement a backward feature elimination function and a forward feature selection function. We would also try out different imputation methods.

References

1. Mishra, P. (2021). *A realistic approach to IEEE CIS Fraud Detection*. [online] Medium. Available at: https://medium.com/@mr.priyankmishra/a-realistic-approach-to-ieee-cis-fraud-detection-25faea54137 [Accessed 7 May 2023].
2. www.formica.ai. (n.d.). *Big Data Analytics for Fraud Detection and Prevention*. [online] Available at: https://www.formica.ai/blog/big-data-analytics-problems-in-fraud-detection#:~:text=Using%20big%20data%20analytics%20in [Accessed 11 May 2023].
3. Mohan, A. (2021). *IEEE-CIS Fraud Detection - Top 5% Solution*. [online] Medium. Available at: https://medium.com/towards-data-science/ieee-cis-fraud-detection-top-5-solution-5488fc66e95f [Accessed 11 May 2023].
4. Spark by {Examples}. (n.d.). *PySpark Tutorial For Beginners | Python Examples*. [online] Available at: https://sparkbyexamples.com/pyspark-tutorial/.
5. kaggle.com. (n.d.). *IEEE_CIS_Fraud_Detection*. [online] Available at: https://www.kaggle.com/code/yuvannabawa/ieee-cis-fraud-detection [Accessed 11 May 2023].
6. Cretan, P. (2022). *Machine Learning On A Large Scale*. [online] Medium. Available at: https://towardsdatascience.com/machine-learning-on-a-large-scale-2eef3bb749ee.
7. Cross Validated. (n.d.). *machine learning - Can increasing the amount of training data make overfitting worse?* [online] Available at: https://stats.stackexchange.com/questions/436006/can-increasing-the-amount-of-training-data-make-overfitting-worse#:~:text=So%20increasing%20the%20amount%20of [Accessed 11 May 2023].
8. Gradient Boosting Trees vs. Random Forests | Baeldung on Computer Science. (2022). *Baeldung on Computer Science*. [online] 25 Feb. Available at: https://www.baeldung.com/cs/gradient-boosting-trees-vs-random-forests#:~:text=4.3.-.
9. Chaudhary, K., Yadav, J., Mallick, B. (2012) *A review of Fraud Detection Techniques: Credit Card*.
10. J. O. Awoyemi, A. O. Adetunmbi and S. A. Oluwadare, "Credit card fraud detection using machine learning techniques: A comparative analysis," *2017 International Conference on Computing Networking and Informatics (ICCNI)*, Lagos, Nigeria, 2017, pp. 1-9, doi: 10.1109/ICCNI.2017.8123782.
11. M. S. Kumar, V. Soundarya, S. Kavitha, E. S. Keerthika, and E. Aswini, "Credit Card Fraud Detection Using Random Forest Algorithm," *2019 3rd International Conference on Computing and Communications Technologies (ICCCT)*, Chennai, India, 2019, pp. 149-153, doi: 10.1109/ICCCT2.2019.8824930.
12. In (2014). *Artificial intelligence applications and innovations.* Heidelberg: Springer. Page 17.