Fraud Detection in Financial Transactions using Machine Learning Techniques in Spark

Manneyaa Jayasanker
Department of Computer Science
University of Texas at Dallas
Dallas, US
mxj180040@utdallas.edu

Venkatesh Sankar
Department of Computer Science
University of Texas at Dallas
Dallas, US
vxs200014@utdallas.edu

Manasa Bhat
Department of Computer Science
University of Texas at Dallas
Dallas, US
mmb190005@utdallas.edu

Swathi Poseti
Department of Computer Science
University of Texas at Dallas
Dallas, US
sxp190117@utdallas.edu

***Abstract* - The influence of Machine learning in Fraud Detection has been ever growing especially in the past decade or so. As the world is quickly progressing towards digitalization and transactions are becoming cashless, the usage of credit and debit cards has increased. Parallelly, the fraud activities associated with it have also been on the rise which results in a huge loss to the financial industry Therefore, we need to filter out the fraudulent transaction from the non-fraudulent ones. In this paper, we present a review of methods used to detect credit card fraud. The methodology mainly analyzed and experimented with is the Decision Tree Algorithm. The main aim of the paper is to design and develop a novel fraud detection method by modifying existing algorithm. Later the classifier is trained over the buckets separately. And then the attribute with better rating score can be chosen to be one of the best methods to predict frauds. In this paper, we worked with Synthetic datasets generated by the PaySim mobile money simulator.**

I. Introduction

What can credit card organizations do to detect the fraudulent patterns that lead their system to abort the abnormal transactions? This paper explores the dataset containing the credit card transactions, information whether the customers are fraud or not. Our ultimate goal is to address this situation by developing classification models to classify and differentiate fraud transactions.

II. DATA

PaySim simulates mobile money transactions based on real transactions retrieved from the financial logs of an African company [1]. The original logs were provided by a multinational corporation that provides the mobile financial service, which is currently available in over 14 countries around the world [2].

This dataset contains approximately 6 million records, which correspond to a month of data collection. Each record represents a single transaction, including information from the accounts of origin and destination, as well as their respective balances. Every record is labeled as a fraud or a legitimate transaction. Furthermore, these transactions can take place between two types of roles in the system: customers and merchants. Customers are ordinary people who use the platform, and merchants are ATMs that allow them to deposit or withdraw money.

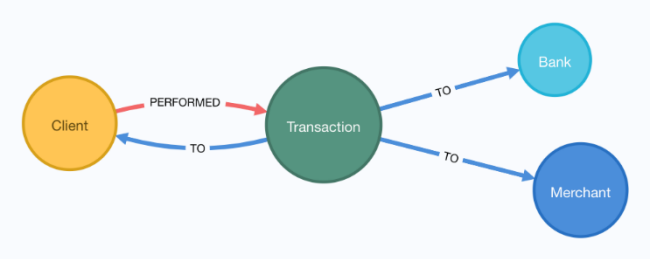


Figure 1: Graphical representation of the PaySim data model [3]

In the datafile we will have the below fields [4]:

The hour of the month is described by *step*, which starts at 1 and goes up to 720, which corresponds to 30 days.

CASH OUT, CASH IN, DEBIT, PAYMENT, and TRANSFER are the five *types* of transactions that occur on the mobile platform.

The *amount* is the amount of money transferred in the transaction; the amount is transferred from an origin account to a destination account.

The *nameOrig* identifies the account that initiated the transaction. Customers and merchants have IDs that begin with a C or an M.

The balance of the origin account prior to the transaction is represented by *oldbalanceOrg*.

The balance of the origin account after the transaction is represented by *newbalanceOrig*.

isFraud is an indicator of illegal activity, with a value of 1 indicating a fraudulent transaction and a value of 0 indicating a legitimate transaction. Only 0.12 percent, or 1 in 833 transactions, are identified as fraudulent, adding to the case's complexity.

isFlaggedFraud indicates whether a transaction attempts to move an amount greater than 200.000, which is a client-implemented policy designed to detect and discourage fraud. However, there is no guarantee that such transaction flagging will work.

The data also includes the fields namely *nameDest*, *oldbalanceDest*, and *newbalanceDest*. In this paper, we are using this data to evaluate the classification model implemented using Decision Trees for detecting fraudulent transactions.

III. DECISION TREE ALGORITHM

Credit card fraud detection is a supervised learning problem, which means that each row or sample in the dataset has a target value. There are two kinds of supervised machine learning algorithms:

- Classification Algorithms
- Regression Algorithms

Credit card fraud detection is a classification problem. Classification problems have integer (0,1) or categorical values(fraud, non-fraud) as the target variable values.

Decision Tree algorithm is a type of supervised learning algorithms [5]. In contrast to other supervised learning algorithms, the decision tree algorithm can also be used to solve classification and regression problems. Decision Tree algorithm can be used to build a training model that predicts the class by learning decision rules from training data [6].

To solve a problem, the decision tree algorithm employs tree representation. An attribute is represented by an internal node of the tree, while a class is represented by a leaf node.

A. Pseudocode

1. Place the dataset's best attribute at the root.
2. Split the training data such that each subsection contains same value for an attribute.
3. Repeat the above steps for each subset until all the leaf nodes are found.

The biggest challenge in implementing the decision tree algorithm is deciding the attributes which should be considered as the root node at each level. This is manageable through attribute selection.

To identify the attribute that can be considered the root node at each level, we use various attribute selection measures such as information gain, gini index, and so on [6]. The above attribute selection measures calculate values for each attribute. Then, the attributes are placed in the tree accordingly based on their sorted value [7].

Information Gain assumes attributes as categorical, whereas the Gini Index assumes attributes as continuous. In our paper, we used Information Gain as the attributes selection criteria where the attribute with the highest value is placed at the root node.

IV. IMPLEMENTATION

The following experimentations were performed on Dataset:

1. Varying the hyper parameters: buckets for continuous variables, depth and number of nodes and find the impact on decision tree creation
2. Split train and test data into different ratios and find the metrics: accuracy, precision, recall, F1score, confusion matrix and area under ROC.
3. Comparison between custom decision tree implementation and MLLib classifiers using accuracy and F1score.
4. Feature ranking based on information gain.

V. RESULTS AND ANALYSIS

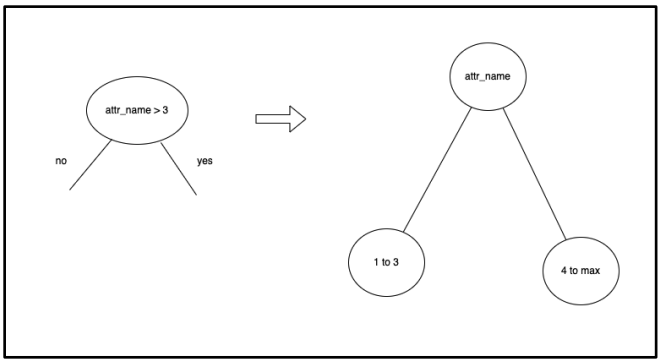
A. Experiments 1 and 2:

Graphical representation of Decision tree explanation

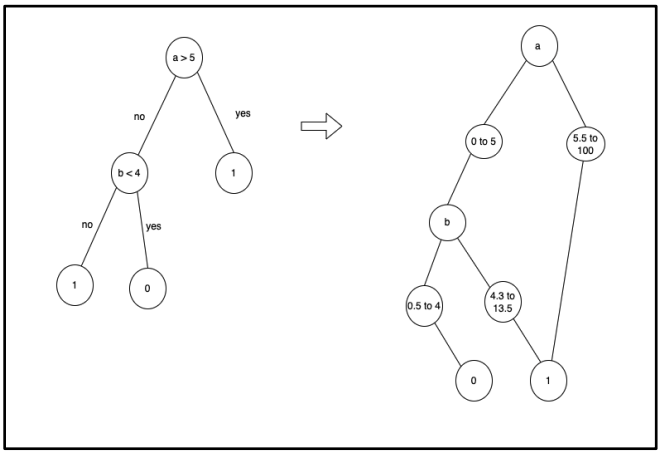
We have chosen graphical representation of the decision tree using the networkx framework.

Since we are using the directed graph, instead of adding new leaf nodes of 0s and 1s when a leaf is reached. There is a single 0 node and single 1 node. And when a leaf condition is reached an edge is created from the respective node to the 0 or 1 node.
So, in the below figures all nodes finally end in either 0 or 1.

The condition over which the traversal needs to take place is also stored as a node. For example, If the node condition is "attr_name > 3 " then the following nodes and edges are created:
Attribute name : attr_name becomes a node, range 0 to 3 becomes a node, and range 4 to higher becomes another node with edges between attr_name to each range.



Hence the graphical representation has nodes for all the attribute names as well as for attribute values. And the overall graphical representation will look like the below transformed decision tree in graphical form.



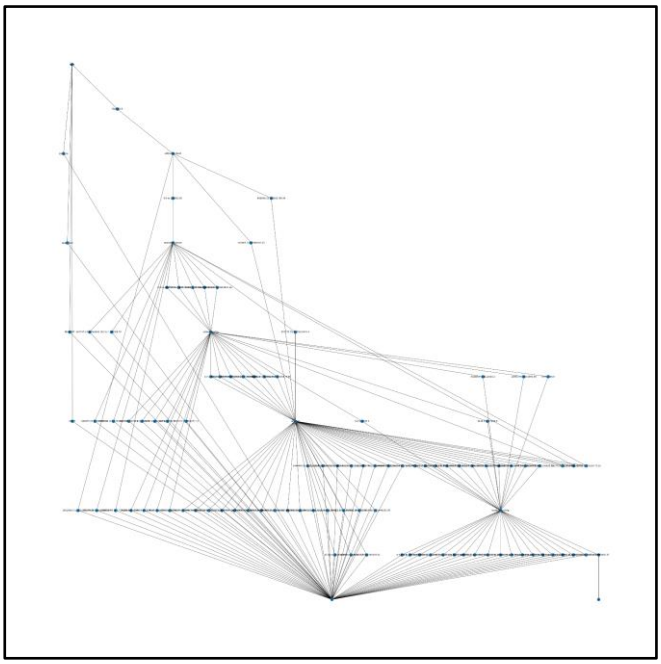
We have introduced three hyper parameters in building decision tree:

1. bucket_count
2. depth_limit
3. nodes_limit

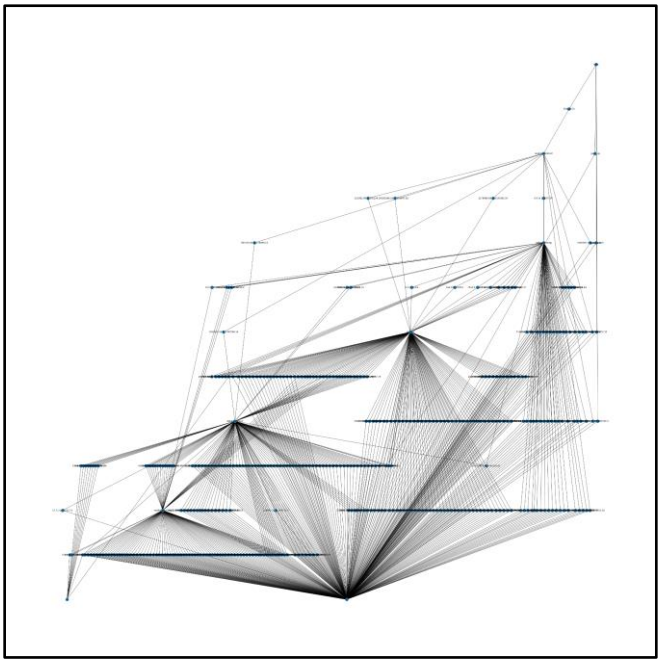
Increasing all the hyper parameters results in a naive decision tree which overfits the training data. And very less values result in underfitting. Hence the hyper parameters are tuned such that the learning is balanced.

However, the fraud detection dataset is highly unbalanced due to rows with fraud data are less than 1%

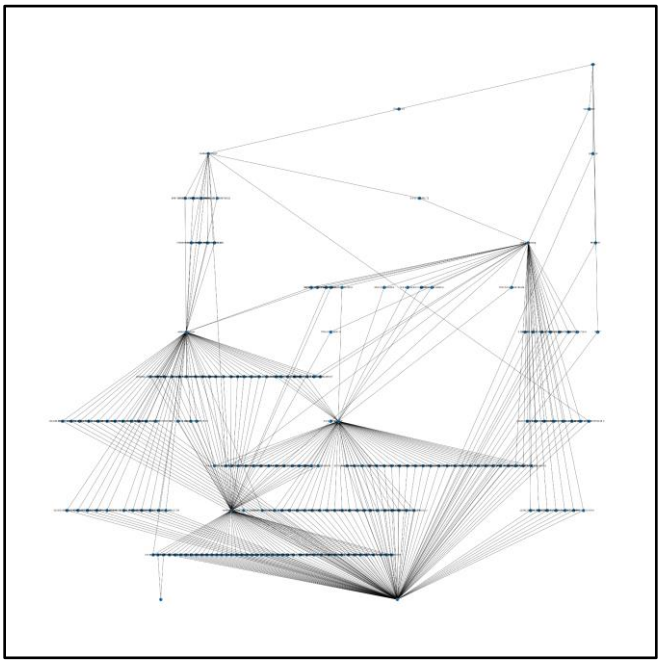
Train Test Split 80:20
bucket_count = 4
depth_limit = 8
nodes_limit = 100
Area under ROC for test data: 0.50335
Precision 1.0
Recall 0.006699147381242387
F1 measure 0.9999629332014474
Accuracy 0.9987176493435324



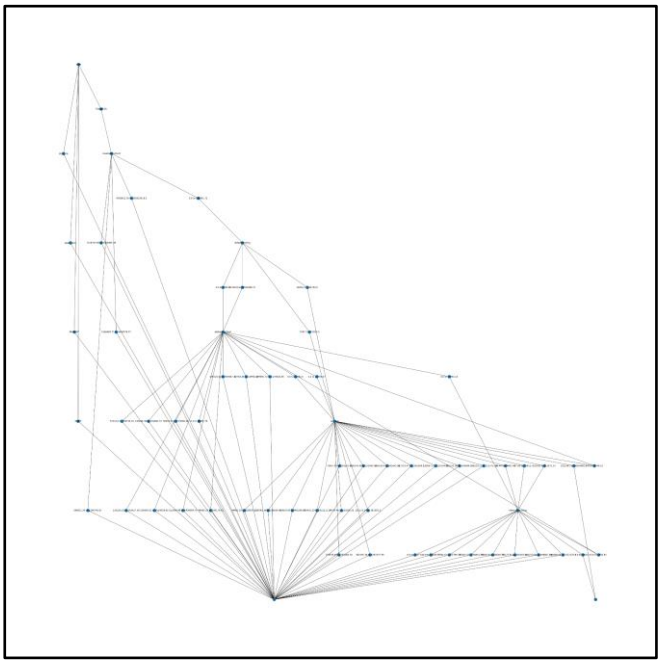
Train: Test Split 80:20
bucket_count = 10
depth_limit = 12
nodes_limit = 400
Area under ROC for test data: 0.53455
Precision 0.027076222980659842
Recall 0.07247259439707673
F1 measure 0.027076222980659842
Accuracy 0.9954406219435106
Took 24.02 minutes to build



bucket_count = 5
depth_limit = 12
nodes_limit = 200
Train Test Split of 90:10
Area under ROC for test data: 0.50307
Precision 1.0
Recall 0.006142506142506142
F1 measure 0.999959551646248
Accuracy 0.9987284152816505
Took 14.41 minutes to build



bucket_count = 5
depth_limit = 6
nodes_limit = 50
Train Test Split of 90:10
Area under ROC for test data: 0.53855
Precision 1.0
Recall 0.07711138310893513
F1 measure 0.9999970079462085
Accuracy 0.9988148828085706
Took 3.26 minutes to build



An increase in all the three hyper parameters shows a decrease in precision, recall and consequently a decrease in F1 measure. This is attributed to overfitting due to increased branching in the decision tree as can be seen in the corresponding figure for bucket_count = 10, depth_limit = 12 and nodes_limit = 400. By executing build tree for various depth values, a depth of around 6-8 gave good results when trained with a data split of 90:10 ratio.

The data split of 90:10 trained the decision tree better compared to 80:20 or 70:30. This is because the data is highly unbalanced with fraud data of less than 1% in our dataset. With a more balanced dataset other data splits for training could be considered.

Continuous variables were divided into a range of values or buckets based on quantiles. And more the buckets, more is the number of child nodes in the graph representation. If buckets were increased the nodes limit also had to be relaxed to avoid underfitting in some branches in the decision trees. Overall, a bucket count of 5 and nodes limit of 50 gave a balanced decision tree.

Note: nodes in our graphical representation not only include the attributes but also each range for continuous variables and unique values for discrete values. Hence a high number of nodes in our representation corresponds to a very low number of nodes in a normal decision tree representation. (For example, 60 nodes correspond to around 8-10 nodes in a normal decision tree representation)

Overall, the precision is very high, and recall is low. This is also the reason for a low area under ROC value around 0.54 which shows a very low discrimination ability. The decision tree could be further improved with pruning techniques for higher ROC values along with obtaining a more balanced dataset.

B. Experiment 3:

Comparison between existing decision tree implementation and Random Forest implementation using built in libraries produced the following:

Classifier	***Test & Train Split (Train: Test) Ratio***	***Metrics***
Decision Tree	*80 : 20*	*Accuracy: 0.9992872433054307 F1 Score: 0.6424911312573907*
Decision Tree	*90 : 10*	*Accuracy: 0.9988825358107195 F1 Score: 0.6511266511266511*
Random Forest	*80 : 20*	*Accuracy: 0.9988951092474484 F1 Score: 0.25053304904051177*
Random Forest	*90 : 10*	*Accuracy: 0.9988825358107195 F1 Score: 0.24442082890541977*

C. Experiment 4:

Feature Ranking based on Info gain
Rank 1 - type
Rank 2 - oldbalanceOrg
Rank 3 - amount
Rank 4 - newbalanceOrig
Rank 5 - oldbalanceDest
Rank 6 - newbalanceDest

Using the information gain it is seen that type of attribute is highly ranked or is best suited to be chosen as the root node of the decision tree. In the below fig1. Out of all the types of transactions: Payment, Cash in, Cash out, Transfer, Debit only Transfer and Cashout have fraudulent transactions (fig2.). From fig 1. Transfer makes for 8% of transactions in the data and Cash out corresponds to 35%

of the transactions. Out of this collective 43% of transactions, 21.5% of the transactions are fraudulent! Hence intuitively, type attributes is a very good measure.

Also, surprisingly oldbalanceOrg which corresponds to original balance is higher ranked compared to amount of transaction. Intuitively the higher the original balance the higher chances of fraud involved. And by computing the average oldbalanceOrg the average is a high value of $164,9667.6057116778.

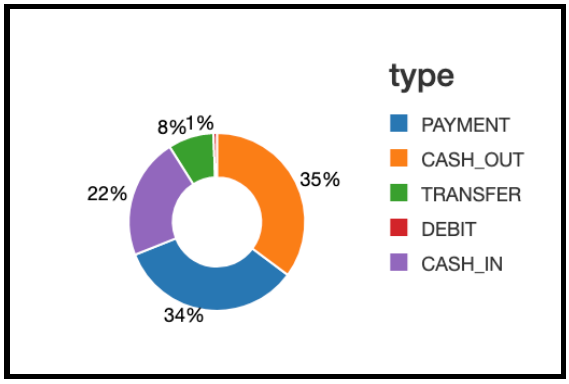


Figure 2: Number of transactions of each type

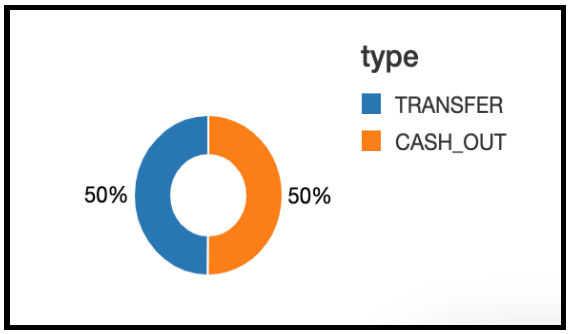


Figure 3: Number of fraudulent transactions

VI. CONCLUSION AND FUTURE WORK

In this paper, we created a framework for distinguishing fraudulent transactions from non-fraudulent transactions in financial data. This framework helps in understanding the intricacies of fraud detection and tuning hyperparameters and choosing the best ranking feature. We experimented with two machine learning algorithms – Decision Tree (using standard built in library and custom algorithm) and Random Forest. The Random Forest algorithm performed worse than the Decision Tree Algorithm, indicating that tree-based algorithms perform well for transactions data with clearly defined classes and attributes. This also emphasizes the importance of conducting rigorous exploratory analysis to thoroughly understand the data before developing machine learning models. We derived the best features that differentiated the classes and attributes through this analysis.

For Future work, we could address Data Imbalance.

There are several techniques to address Data Imbalance. A few examples are as follows –

1. Under sampling: Here, samples taken randomly from the majority class are eliminated so that the class imbalance can be tackled and is made more manageable.

2. Oversampling: Here, observations of the minority class can be sampled with repetition to increase their occurrence in the data, thus handling oversampling.

VII. REFERENCES

[1] E. A. Lopez-Rojas , A. Elmir, and S. Axelsson. "PaySim: A financial mobile money simulator for fraud detection". In: The 28th European Modeling and Simulation Symposium-EMSS, Larnaca, Cyprus. 2016

[2] Paysim, Synthetic Financial Datasets For Fraud Detection URL https://www.kaggle.com/ealaxi/paysim1

[3] Paysim Data Model, URL https://www.sisu.io/posts/paysim/

[4] David Gamba, Data Science Journeys: Credit Card Fraud Detection https://medium.com/@ccontreras.gamba/data-science-journeys-fraud-detection-4b6618b7fb82

[5] Decision Tree Algorithm Explained, URL https://www.thinkdataanalytics.com/decision-tree-algorithm/

[6] Sharmila Polamuri, Credit Card Fraud Detection with Classification algorithms in Python, URL https://dataaspirant.com/credit-card-fraud-detection-classification-algorithms-python/

[7] Sandeep Khurana, Decision Tree using Spark For Layman, URL https://towardsdatascience.com/machine-learning-decision-tree-using-spark-for-layman-8eca054c8843