Detecting potential fraudulent transactions in a bitcoin trading network *(combining Network/Graph theory and Deep Learning)*

Problem statement and the need

Cryptocurrency, one of the basic and direct applications of the block chain systems is available online for one and all for trading. They can be exchanged for other currencies, products, and services. Transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain. One such cryptocurrency is Bitcoin, which is a decentralized digital currency that can be sent between users in a peer-to-peer bitcoin network.[1]. Time and again Bitcoin users have reported their Bitcoin wallets being compromised. The transactions in a peer to peer network is not trustworthy. Using the feature learnings on graphs using methods like: node2vec algorithm or matrix factorization methods, I built a classifier that can distinguish between fraudulent or honest traders in order to estimate if the seller will commit fraud in a future transaction or not. [2]

Bitcoin trading follows a peer to peer (P2P) market place. In addition to providing cryptocurrency-based marketplaces, we have a number of such platforms that we may use in our day to day lives that run on such an architecture. (PayPal, Venmo, Paytm) [7]. In simpler terminology, in such networks, one can trade goods and cryptocurrencies in a platform that has no mediator and offers anonymity to its users. However, just like in all other trading platforms, one can fall victim to a fraudster, losing money and trust in the platform. Some of these bitcoin trading platforms try to prevent fraudulent transactions by letting users rate each other after completing a transaction using **a perception score** or **trust values on a scale of -10 to 10**. Unfortunately, as promising as they can be, these perception scores do not suffice for identifying potential fraudsters who may disguise themselves as trustworthy users (sellers or receivers).

I decided to work on the “transaction” and “node” aspect of the blockchain architecture to strengthen the secure transactions (edges) between the users (nodes) in a peer to peer bitcoin trading market place.

The dataset and exploration:

I am using a mixture of the two bitcoin marketplaces datasets: Bitcoin OTC and Bitcoin Alpha. These are who-trusts-whom network of people who trade using Bitcoin on these aforementioned platforms. I downloaded them from SNAP [3] and combined the datasets for the marketplaces for an analysis later.

We can imagine them as graphs or networks which have the users as the nodes and transactions as the edges.

The problems with current perception/trust scores marking: It is difficult to conclude the confidence of the honest seller where there are a lot of ambiguous. Also, it becomes very difficult to track a fraudster after completing the money transfer as they may manipulate the positive ratings by assigning honest reviews to themselves from different accounts/platforms.

Hence, with the aforementioned problems, we can conclude that in such marketplaces like the Bitcoin Alpha or OTC, only perception scores are not enough to detect fraudsters. We can enhance perception scores using network/graph theory explained in the next section. by using Node2vec algorithms to learn node features and give them better information like.

Files used:

1. original _network: contains the concatenated ratings of the two markets
2. Shorter_network: contains a shortened version of the above file for trainings and feature learning. (contains 70% of the original data)
3. Test_network: used as validation dataset

Feature Learning with Node2vec:

In addition to perception scores, to enhance the feature extraction and learning, we can combine some of the learnt continuous features representations for the nodes in the graph that can downstream machine learning tasks and make it easier to classify. One such algorithm is 'Node2vec' by SNAP [4].
We can characterize this information from the network by mapping the users as nodes and transactions as edges where we can depict very positive ratings as some weights.

Concatenating the Bitcoin OTC and Bitcoin Alpha dataset tables results in a graph with **59,788 transactions** (*edges*) and **9,664** users (*nodes*). For each node, 6 features like: ratings received, positive ratings received, negative ratings received, etc were given. In addition to these features given, I also incorporated node2vec algorithm's python implementation. It therefore creates a 20 dimension-feature vector for each node.
Then, for the normalization of the data/features and converting it to a data frame then later save it into a csv file.

1. Normalize the node features from extracted from the node2vec algorithm and concatenating with the other 6 perception features.
2. Because all the 20 features have different scales, we will normalize the entire (Node_features_matrix[]) matrix using the following formula:

$$X_{normalized} = \frac{X - \mu_X}{\sigma_X}$$

Where μ_X and σ_X are the columnwise mean/STD values of the matrix X. In this way, we ensure that all values in the matrix lie within the $[-3\sigma_X, 3\sigma_X]$ range

Model construction: Training and Evaluation

I first started with supervised classifiers like logistic regression using only perception scores first with labelling fraud transaction class as 1 and the honest transaction class as 0.

```
Accuracy of logistic regression classifier on test set: 0.96
Classification Report for Logistic Regression
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98      9119
         1.0       0.83      0.62      0.71       846

    accuracy                           0.96      9965
   macro avg       0.90      0.81      0.84      9965
weighted avg       0.95      0.96      0.95      9965
```

Then, to be sure of the dataset I plotted a confusion matrix to know more about the percentage of correct classification of honest or fraudulent transactions. Due to the lower availability of the fraudulent ratings, the confusion matrix is not able to learn the fraud ratings too accurately. We can use data augmentation in the future to enhance our dataset.

Then created a neural network of 6 layers with relu and Sigmoid activation functions. For comparison I trained the model with the following:

1. NN with both perception scores and node2vec-- with using bootstrapping

```
#evaluate model 4 with test set
y_pred = M4.predict(X_test)

# Print classfication report
target_names = ['Honest','Fraudulent']
confusion_matrix = cm(y_test, np.round(y_pred))
print("Classification Report for the NN model (test set)")
print(classification_report(y_test, np.round(y_pred)))
```

```
Classification Report for the NN model (test set)
              precision    recall  f1-score   support

         0.0       0.99      0.93      0.96      9119
         1.0       0.56      0.93      0.70       846

    accuracy                           0.93      9965
   macro avg       0.78      0.93      0.83      9965
weighted avg       0.96      0.93      0.94      9965
```

2. Neural network with perception scores only

Finally, I tried to improve the accuracy by using the MLP for binary classifier for the validation dataset.

```
Epoch 1/5
39857/39857 [==============================] - 11s 270us/step - loss: 0.3000 - acc: 0.9143
Epoch 2/5
39857/39857 [==============================] - 10s 243us/step - loss: 0.2583 - acc: 0.9154
Epoch 3/5
39857/39857 [==============================] - 9s 236us/step - loss: 0.2550 - acc: 0.9154
Epoch 4/5
39857/39857 [==============================] - 10s 263us/step - loss: 0.2538 - acc: 0.9154
Epoch 5/5
39857/39857 [==============================] - 9s 236us/step - loss: 0.2540 - acc: 0.9154
```

We need bootstrapping in order to estimate standard errors and confidence intervals (https://papers.nips.cc/paper/659-assessing-and-improving-neural-network-predictions-by-the-bootstrap-algorithm.pdf). *The bootstrap is a widely applicable and extremely powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method.*

Model Evaluation:

Plotting the ROC/AUC curves.
Cross validation is often not used for evaluating deep learning models because of the greater computational expense. For example k-fold cross validation is often used with 5 or 10 folds. As such, 5 or 10 models must be constructed and evaluated, greatly adding to the evaluation time of a model.

References:

1.https://en.wikipedia.org/wiki/Bitcoin
2. https://arxiv.org/abs/1709.05584
3.http://snap.stanford.edu/data/index.html
4.https://snap.stanford.edu/node2vec/
5. http://snap.stanford.edu/data/soc-sign-bitcoin-otc.html
6. https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf)
7.http://snap.stanford.edu/class/cs224w-2013/projects2013/cs224w-030-final.pdf