Detection of Circular Trade using Node2Vec

Anirudh Raghav CH19BTECH11033
Anushka Khare CH19BTECH11029
Tanmay Goyal AI20BTECH11021
Tanay Yadav AI20BTECH11026
Vedika Verma CS19BTECH11057

Abstract

Circular Trade is a common fraudulent scheme that aims to manipulate share trade volume. It involves a set of traders that have very similar buy and sell orders with the same amount of money involved. This causes no difference to the price of the shares but brings about a rise in the trade volume of the shares, thus, indirectly manipulating the price of shares. Through this work, given the transactions of a set of traders, we attempt to detect circular trade using the Node2Vec algorithm.

1. Problem Statement

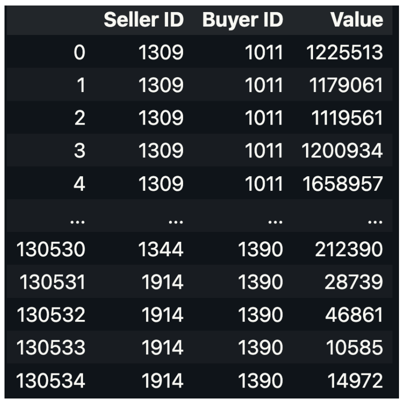
Circular Trade is a common fraudulent scheme that aims to manipulate share trade volume. It involves a set of traders that have very similar buy and sell orders with the same amount of money involved. This causes no difference to the price of the shares but brings about a rise in the trade volume of the shares, thus, indirectly manipulating the price of shares. Through this work, given the transactions of a set of traders, we attempt to detect the set of dealers that may be involved in circular trade using the Node2Vec algorithm.

2. Dataset

The given dataset consists of Iron dealers, where each row consists of one invoice. Each row consists of the Seller ID, Buyer ID, and the value involved in the transaction. Together, the dataset consists of 130535 rows of invoices and involves a set of 703 unique Sellers and 371 unique Buyers. The number of Buyers being almost half the number of Sellers raises suspicion, which is further aggravated by the number of invoices. However, we wish to detect any kind of fraudulent activity among this set of dealers, which is why we make use of the Node2Vec Algorithm which has been explained in the subsequent sections.

3. Algorithm and Methodology

To implement the Node2Vec Algorithm, we need to model all the transactions as a graph. In light of this, we take the following steps:



	Seller ID	Buyer ID	Value
0	1309	1011	1225513
1	1309	1011	1179061
2	1309	1011	1119561
3	1309	1011	1200934
4	1309	1011	1658957
...	...	...	...
130530	1344	1390	212390
130531	1914	1390	28739
130532	1914	1390	46861
130533	1914	1390	10585
130534	1914	1390	14972

Figure 1. A snapshot of the dataset

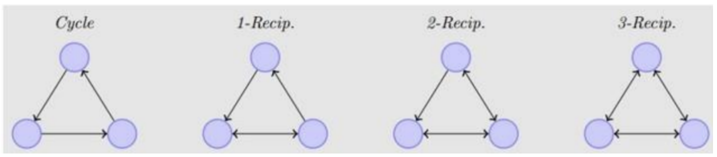
1. We make two dictionaries: *sell_buy* to hold the unique seller-buyer pairs for each transaction and *transactions* for the transaction value for each invoice. Thus, we have two graphs here, one to model the seller-buyer relationship regardless of the number of transactions or the amount of the transactions. And one to model all the information about the transactions.

2. We now convert the graph *transactions* from a multi-graph to a weighted directed graph. For this, we sum up all the edges between two nodes, i.e the final edge $e(i \to j)$ can be written as:

$$e(i \to j) = \sum_{i} e_n(i \to j)$$

where e_n is the n^{th} edge between i and j.

3. We now wish to convert this weighted directed graph into a weighted undirected graph. For this, we first calculate the number of 2-cycles and 3-cycles. Amongst 3-cycles, we can have 4 different types of cycles: We



assign the following weights: 4 for 3-Recip cycle, 3 for 2-Recip cycle, 2 for 1-Recip cycle and 1 for the normal cycle. We assign a weight of 5 for 2-cycles. Thus, for every edge $e(i->j)$, we give it some weight dependent on the number of cycles it is part of and the kind of cycles it is part of.

4. Another requirement of circular trading is the values of transactions have to be similar. Thus, we assign certain weight if within a cycle, edges have *similar* transaction values. Since the term *similar* is ambiguous, we define it to be within 5% of the other edge, i.e we say two edge weights m and n have similar value if

$$m \in [0.95n, 1.05n]$$

5. Finally, we define the undirected edge weight $e(i, j)$ to be the maximum of the two directed edges, i.e

$$e(i, j) = \max\{e(i \rightarrow j), e(j \rightarrow i)\}$$

Now that we have the undirected weighted graph, we shall convert it into an edge list and feed it to the Node2Vec implementation developed by Aditya Grover and Jure Leskovec and can be found on the following page: `https://github.com/aditya-grover/node2vec`. We have to make certain changes to some of the files due to the version errors and deprecations, all of which are listed in the README.

We shall now explain the Node2Vec implementation briefly. It is based on the Word2Vec implementation. The Word2Vec implementation consists of a text source, and produces vector embeddings for each word depending upon it's frequency, it's associated neighbours and meaning in the sentence. It is a two layer neural network (with one hidden layer) that takes as input a one-hot encoded vector of the target word, and learns the probability of seeing all the other words in the corpus given the target word. The training data is selected as follows: a target word is selected over some rolling window. The training data then consists of pairwise combinations of that target word and all other words in the window. This is the resulting training data for the neural network.

The Node2Vec algorithm runs on the same principles, however, it produces embeddings for each node in graph based on its connectivity and neighbours. It generates random walks on the graph to produce the training data and uses it as input to a two-layer neural network. According to the original paper, we set the hyper-parameter $q = 0.5$ and choose the number of dimensions to be equal to the square root of the number of nodes, which is roughly 29.

Once the embeddings have been produced (We use a 29-dimension embedding), we use a Principal Component

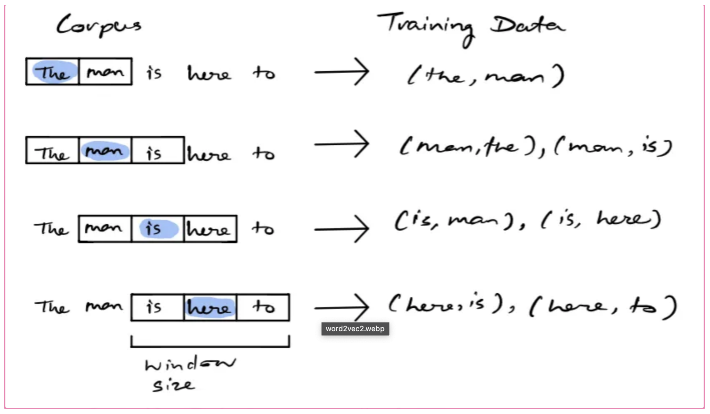


Figure 2. Generating data for the Word2Vec model

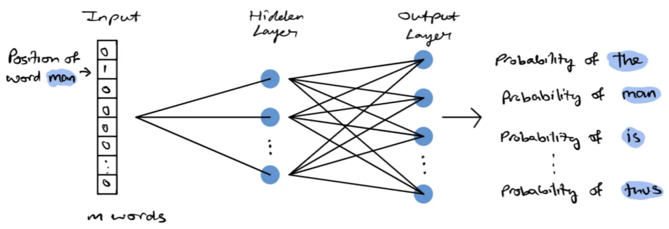


Figure 3. The Word2Vec Model

Analysis to reduce it to 2 dimensions for better visualization and to avoid the curse of dimensionality. We then use the DBSCAN algorithm.

We now use DBSCAN to find any closely associated clusters that may appear suspicious.

The DBSCAN algorithm is explained below: it classifies points into 3 categories: Core point, Border point and a noise point. It takes two hyper-parameters as inputs, *minPoints* and *eps*. *eps* is the radius around each point. If a point has *minPoints* samples in a neighbourhood of radius *eps*, then it is a Core point. If a point is not a Core point, however, it is in the *eps*-neighbourhood of a core point, then it is Border Point. Finally, if a point is neither a Core Point or a Border Point, it is classified as a Noise Point.

The method to find the optimal *eps* is as follows: We fix *minPoints* and then find the *minPoints* nearest neighbour of each point. The idea is the same indexed nearest neighbour for each point in a cluster shall be at a similar distance from the point. Thus, we sort the distances of the nearest neighbours and find the bend in the curve.

Finally, we choose $eps = 0.0020$, and the minimum sample number is 10. This results in 9 clusters. We then pick out the clusters based on the number of points in the cluster. If the number of traders in a cluster is close to 10, then we shall say it may be involved in circular trading and is suspicious. This way, we get cluster numbers 2, 4,6, and

7 with 10,12,10 and 7 traders respectively, to be the suspicious groups of traders that may be involved in circular trading.

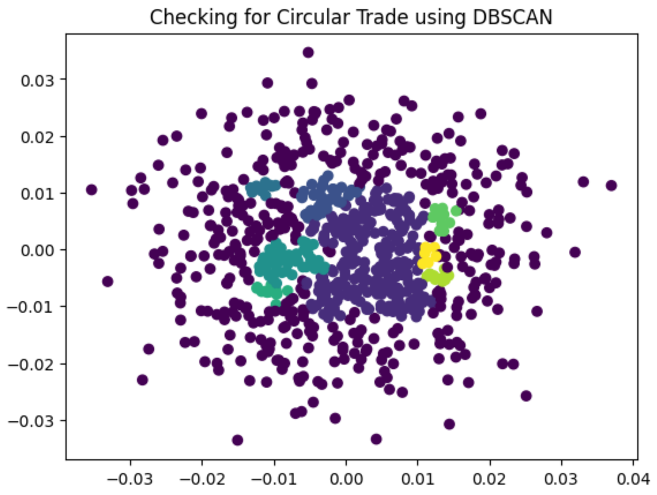
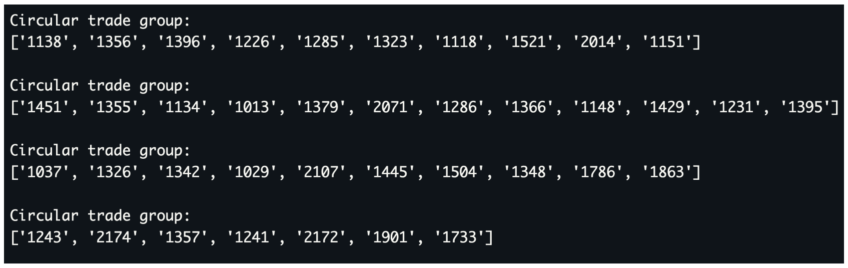


Figure 4. Outlier detection using DBSCAN

Finally, we can say that these densely clustered groups of traders can be labeled as suspicious and might be involved in circular trading.

4. Results

The list of potential circular trading groups are listed in the output file *output.txt*. They are listed here too:



```
Circular trade group:
['1138', '1356', '1396', '1226', '1285', '1323', '1118', '1521', '2014', '1151']

Circular trade group:
['1451', '1355', '1134', '1013', '1379', '2071', '1286', '1366', '1148', '1429', '1231', '1395']

Circular trade group:
['1037', '1326', '1342', '1029', '2107', '1445', '1504', '1348', '1786', '1863']

Circular trade group:
['1243', '2174', '1357', '1241', '2172', '1901', '1733']
```

Figure 5. Outlier detection using DBSCAN