Circular Trading Detection using Node2Vec

Anirudh Srinivasan
CS20BTECH11059

Haritha R
AI20BTECH11010

Nelakuditi Rahul Naga
AI20BTECH11029

Rahul V
AI20BTECH11030

Saketh Vaddamani
CS20BTECH11054

1. Problem Statement

Multiple traders working together to produce fictitious activity or price movement in a financial market is known as **circular trading**. This can be accomplished in a number of ways, such as by trading the same security back and forth between them at escalating prices to fake demand for that security.

Circular trading has the drawback that it might be challenging to spot because to an outside observer, the activity might seem to be normal trade. However, market surveillance teams and financial regulators have a variety of tools at their disposal to spot this kind of dishonest behaviour.

Analyzing trading data is one method for finding trends that point to collusion. Circular trading may be evident, for instance, if numerous traders often trade the same securities with one another and exhibit strongly connected trading patterns. The same goes for traders who buy and sell the same security quickly after each other.

Another approach identifies groups of traders who engage in circular trading by employing network analysis tools. It could be able to spot trends that point to collusion by looking at the relationships between traders and the flow of deals between them.

Regulators and market surveillance teams frequently utilise cutting-edge technology, like machine learning algorithms, to analyse massive amounts of trading data in real-time in order to find circular trading. These algorithms are able to spot questionable trade patterns and flag them for additional examination.

Overall, identifying circular trade is a difficult and complex subject that calls for a blend of traditional detective work and cutting-edge technology. Regulators may contribute to ensuring that financial markets continue to be fair and transparent for all participants by spotting and rooting out fraudulent behaviour.

2. Description of the dataset

The dataset given consists of Iron dealers data in csv format. It has a Seller ID, representing a seller, Buyer ID representing a Buyer, and the Value of transaction represented by Value. An example of the format is shown below:

S.No.	Seller ID	Buyer ID	Value
1	1309	1011	1225513
2	1259	1011	1710778
3	1259	1011	1694993
4	1090	1079	538375

Table 1. Basic Format of the Data

2.1. Basic Statistics

	Seller ID	Buyer ID	Value
count	130535.000000	130535.000000	1.305350e+05
mean	1309.358287	1182.851258	6.930965e+05
std	294.435026	169.809657	5.696676e+05
min	1001.000000	1001.000000	1.000600e+04
25%	1078.000000	1060.000000	2.361085e+05
50%	1214.000000	1112.000000	5.571960e+05
75%	1488.000000	1276.000000	1.074405e+06
max	2190.000000	1887.000000	2.124000e+07

Table 2. Basic Statistics of the Data

The basic statistics of the data are given in the table 2.

2.2. Missing Values

There were no missing values in the dataset.

Other details about the dataset have been provided in the code that we have submitted.

3. Algorithms used

The main algorithm that is involved in the detection of circular trading is **Node2Vec**. But the Node2Vec algorithm is significantly inspired from word2vec skip-gram model. Therefore, we first provide a brief summary about word2vec skip-gram model as a precursor to the Node2Vec algorithm.

3.1. Word2Vec SkipGram Model

Given a text corpus, word2vec model is used to produce word representations as vectors in a embedded space, such that words which share common contexts are located relatively closer to one another. The context here is defined as the adjacent words to the input term. Window size is defined to be the maximum distance between the words in the context window, having input word at center.

In the Skipgram model, we use neural networks with the focus word as single input vector and the target output words as context layer. In Word2Vec, our goal is not to find the output layer, but to learn the weights of the hidden layer. The word embedding is determined by the weights of the hidden layer and number of neurons in hidden layer will determine the embedding dimension, i.e., size of the vector representing each word in vocabulary.

Generally to train a neural network, a training sample is used to adjust all weights in neural networks. But there maybe millions or billions of input-context pairs, and updating weights for every one of them may turn out to be computationally expensive. We solve this performance issue by using negative sampling i.e; by having each training sample to modify only a small subset of the weights rather than all of them.

3.2. Node2Vec

Node2Vec algorithm is implemented using skip-gram with negative sampling. For every node in the algorithm, by choosing a particular node as starting node, a series of random walks, create a number of sentences. For the random walks, the walk length and walks per node can be defined. The walk length decides the length of the sentence. After the sentences are generated, they are passed into the Skip gram negative sampling model as input and weights of the hidden layer are retrieved as node embedding.

There are two types of random walks, first-order biased random works and second-order biased random works. In first-order random walks, we only look at current state, i.e., probability of returning to a previous node or any new node is equal. But, if the graph is weighted, probability also depends on weights. In second-order random walks, we also take previous state as well as current state to calculate traversal probabilities.

The return parameter p and in-out parameter q decide how the graph is traversed in case of second-order random walks. If p is higher, then there is a low chance of revisiting immediately previous node and also avoid 2-hop redundancy in sampling. If the value of q is higher , then it is biased to hop to nodes closest to the previous node, i.e., approximately same as breadth-first search. If the value of q is lower, it has a high chance of visiting farther nodes, i.e., similar to depth-first search. However, we use second-order biased random walks for implementing node2vec algorithm.

By using various configurations of p and q in the second-order biased random walks, various similarities can be produced between nodes. By setting a small value of q, Homophily can be achieved. Here, we can define Homophily as nodes that belong to same network community, i.e., which are closer to one another in the network. When we set a high value for q, structural equivalence similarity can be achieved. Here, Structural equivalence refers to the extent to which two nodes are connected to the same nodes, i.e., they share same neighbourhood while not being directly connected.

3.3. TSNE

TSNE (t-distributed stochastic neighbor embedding) is a dimensionality reduction technique for visualizing higher-dimensional data in a lower dimensional space (say 2D). It transforms higher-dimensional data into lower-dimensional data by maintaining consistency in the aspect of the similarity and dissimilarity between data points in higher and lower dimensional spaces.

3.4. DBSCAN

DBSCAN is a density-based algorithm that is used for clustering the vector representation of nodes obtained through the Node2Vec algorithm. The vectors belong to the real space $\mathbb{R}^d$ where d is the dimension of Node2Vec embedding. There are two hyper-parameters that are involved in the implementation of this algorithm - **MinPts** and **Eps**. As a rule of thumb, we can initialize MinPts as follows :

$$\text{MinPts} = 2 \times d = 2d \tag{1}$$

To initialize Eps, we use k-Nearest Neighbours algorithm (with an appropriate k) and plot the distance of k^{th}-nearest neighbour from each data point in an ascending order. Then, we try to locate the **elbow** region in the plot to determine the appropriate value of Eps.

3.5. Main Algortihm

We first convert the given csv file into a dataframe using the pandas library. Then using networkx library, we form a Multi-directed graph (from the dataframe) with edges as transactions, source and destinations of those edges as Seller ID and Buyer ID respectively. We assign values of the transactions as the weights of those edges. Then we need to convert this multidirected graph into simple undirected graph. So, for that we first convert the multidirected graph to simple directed graph. Then, for every edge in this directed graph, we make edge weights proportional to number of two and three cycles this edge is involved in. We also make sure that if the values are close to each other in these cycles, then the weights would be increased. For our sake, we took until 10% difference, they can be considered close to each other. So, in this way we finally get a simple undirected graph. Then we input this networkx graph to the Node2Vec() function imported from node2vec library to generate 2nd order biased random walks. These generated walks along with parameters such as dimension of embedding, window size, min_count and batch_words are then input to the Word2vec model to get the vector embeddings of the nodes. We have used $d = 2$ as the dimension of embedding output by the Word2vec model.

The embeddings generated are of dimension d i.e; they are vectors in the real space $\mathbb{R}^d$ as mentioned earlier. In order to visualize the embeddings, we convert them into two-dimensional vectors i.e; vectors in the real space $\mathbb{R}^2$ using TSNE.

Finally, we cluster the vector representation of nodes obtained through Node2Vec and TSNE algorithms, using DBSCAN so as to identify the clusters of people involved in circular trading. For implementing DBSCAN algorithm, as mentioned earlier, we need to initialize two hyperparamenters, namely MinPts and Eps. As we are clustering in 2D space, we initialized MinPts as $2 \times 2 = 4$. Then we used k-Nearest Neighbours algorithm with $k = 4$ to plot all the sorted 4th nearest neighbour distances in order to determine Eps. We then input the vector embeddings and the obtained values of the hyperparameters to the DBSCAN algorithm to obtain the clusters of nodes. Finally, we display the clusters and also print all the points in a corresponding cluster. We separately print the noise points.

4. Final Results and Conclusions

The results obtained after clustering of the Node2Vec embedded vectors for different values of hyperparameters p, q, l (p - Return parameter, q - In-out parameter, l - Walk-length) are as follows :

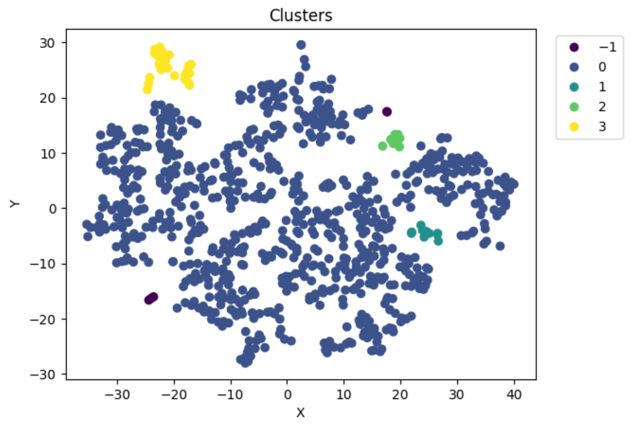

Figure 1. $p = 1, q = 1, l = 20$

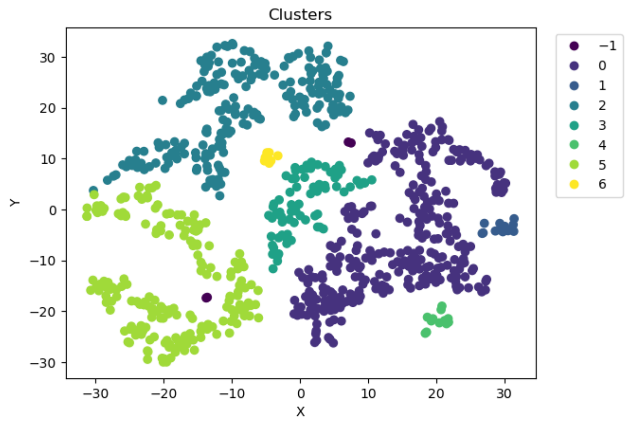

Figure 2. $p = 1, q = 1, l = 200$

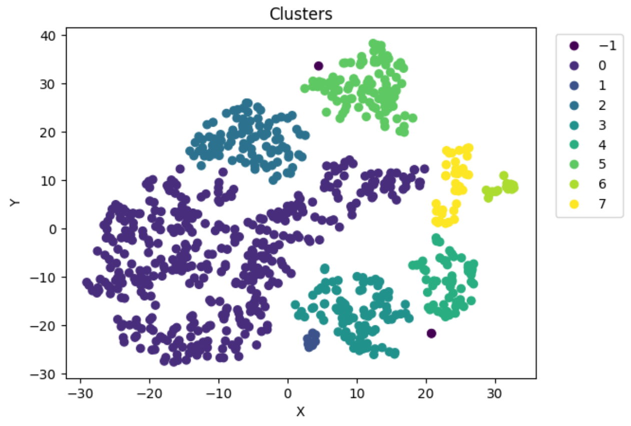

Figure 3. $p = 1, q = 10, l = 80$

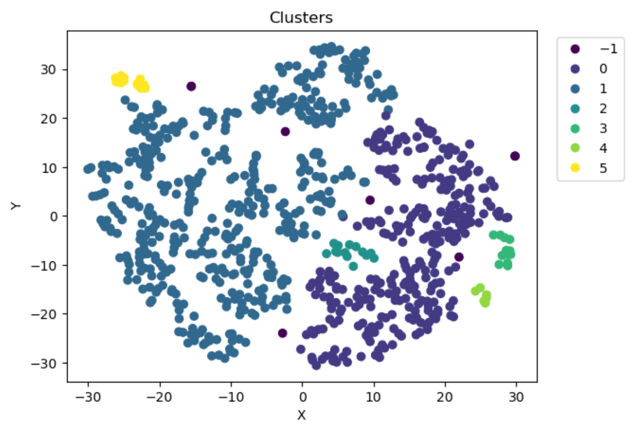

Figure 4. $p = 10, q = 1, l = 80$

References

[1] Paper on circular trading - Link

[2] Blog on Node2Vec -Link

[3] Blog on Node2Vec Implementation - Link

[4] TSNE Wikipedia - Link

[5] Tutorial on DBSCAN - Link