Indian Institute of Technology Hyderabad

Fraud Analytics (CS6890)

Assignment: 1 | Detection of circular trade

Name	Roll Number
Manan Darji	CS22MTECH14004
Dhwani Jakhaniya	CS22MTECH14011
Ankit Sharma	CS22MTECH12003
Vishesh Kothari	CS22MTECH12004
Jayanti Mudliar	CS22MTECH14001

Contents

1 | Problem statement

- Identify sets of dealers doing circular trading using **Node2Vec.**
- Each row n the data set is one invoice (seller id, buyer id, value). We have to identify the fraudulent dealers based on transaction cycles and amount of transactions.

2 | Description of the data set

There is files provided named Iron_dealers_data.csv

Iron Dealers data file contains 1,30,535 rows and three columns: Seller ID, Buyer ID, and Value. Each of the 1,30,535 rows in Iron Dealers data is a transaction of some Value(column 3) between the given Seller(column 1) and Buyer(column 2).

Out of the 1,30,535 transactions, only 5358 of them involve a unique pair of dealers who are dealing with each other.These 5358 transactions represent unique transactions between pairs of dealers who have engaged in business with each other at least once.

The graph depicted in Figure 2.1 consists of 799 nodes and 5358 edges, and it is evident that not all nodes are interconnected. Specifically, some nodes do not have directed outgoing edges, resulting in them being disconnected from other nodes.

From analysis of data, we can see a maximum transaction of 2,12,40,000 and a minimum transaction of 10,006.

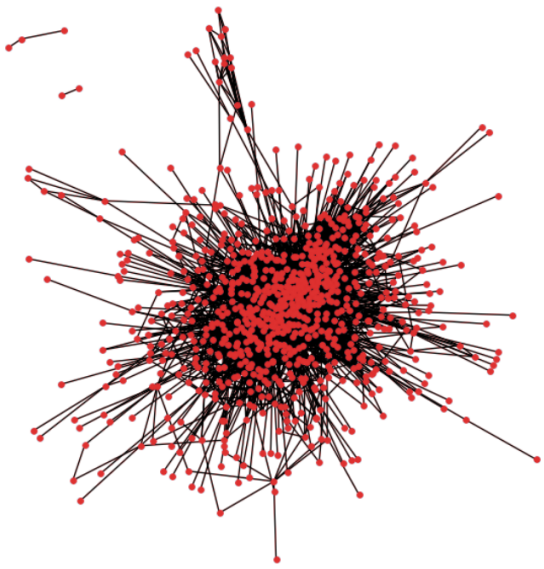


Figure 2.1: Visualization of Data set

3 | Algorithm Used

3.1 | Related Topics

3.1.1 | Node2Vec

Node2Vec is an algorithm that maps nodes in a graph G to an embedding space. Embedding spaces are nothing but vectors corresponding to each node in the graph that are commonly used as inputs for the Machine Learning problems such as community detection and classification. Dealing with extensive data sets as a graph takes much work to visualize. The right solution is to convert the nodes in the embeddings and then visualize these features in lower dimensions.The number of ”dimensions” indicates the vector size on which the graph is to be embedded. The parameter ”walk-length” indicates the length of the random walks that need to be generated from the source node while traversing its neighbors/successors. The parameter ”num-walks” indicates how many such random walks are to be taken from each node under consideration.

3.1.2 | DBSCAN

DBSCAN Density-Based Spatial Clustering of Applications with Noise is an unsupervised clustering method that converts the clusters with high density to those with low density. DBSCAN works in three steps: 1) Divide the data set into n dimensions. 2) DBSCAN forms an n-dimensional shape around each node in the data set and then checks how many points fall within the shape. 3) It counts this shape as a cluster

and expands clusters by iterating through each point.We begin to find that value of "eps" which suites our generated node embedding, from node2vec's output, by iterating from 0.1 to 5, in steps of 0.1. The value 1.7 gives us the maximum number of clusters. At the end, we want to maximize our number of cluster findings, that could help us to identify different sections of traders involved in various methods of circular trading.

3.2 | Approach we followed

3.2.1 | Converting Directed Weighted Multi-Graph To Directed Weighted Simple Graph

To begin the implementation of this algorithm, we first need to load the dataset, focusing on the first two columns containing the seller ID and buyer ID. Upon examination, we can determine that there are 799 unique nodes. There are a total of 1,30,535 transactions in the dataset. To account for the possibility of multiple directed edges between nodes, we can consolidate them and transform the multi-directed graph into a single-directed graph. This consolidation process results in a graph of 5,358 edges, where the weight of each directed edge represents the total sum of directed transactions between any two given nodes.

3.2.2 | Converting Directed Weighted Simple Graph To Undirected Weighted Simple Graph

As discussed in class, the primary objective of this problem is to transform a directed weighted simple graph into an undirected weighted simple graph while preserving some major of circular trade as edge weights. We have explored numerous methods and approaches to accomplish this task and have ultimately derived the solution presented in this section.

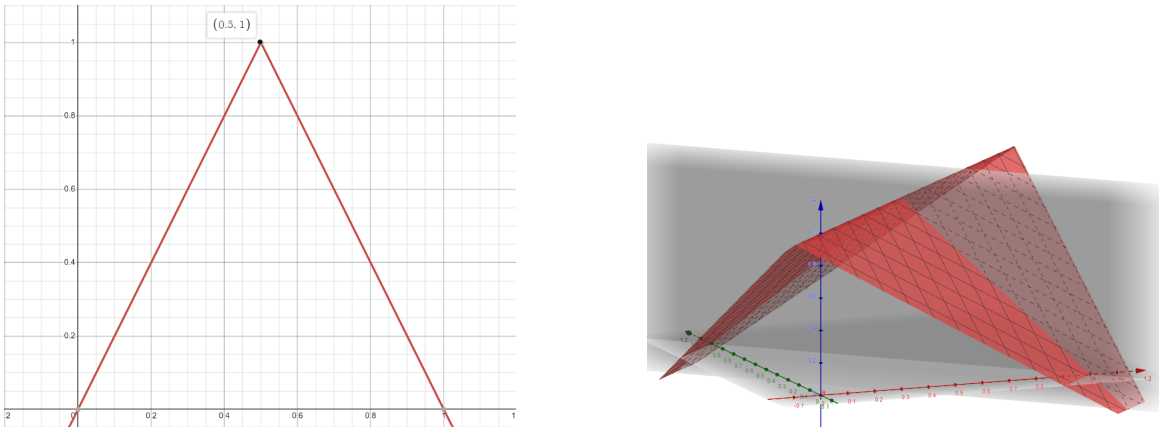
Algorithm 1 Converting Directed Weighted Simple Graph To Undirected Weighted Simple Graph

Input: Edges E, Cycles C
Output: Weighted Adjacency Matrix for Undirected Graph M

Initialize Matrix $M \leftarrow 0$
for all V_1 **and** $V_2 \in Edges$ **do**
  $InvolvedCycles \leftarrow \emptyset$
  for all $c \in Cycles$ **do**
    if V_1 **and** $V_2 \in Cycles$ **then**
      $InvolvedCycles \leftarrow c$
    end if
  end for
  for all $c \in InvolvedCycles$ **do**
    if $Cycles\ Length\ is\ 2$ **then**
      $CycleSum = \sum Cycle\ Edge\ costs$
      $W_i = \frac{Edge\ i\ Weight}{CycleSum}$
      $FraudFactor = 1 - |W_1 - W_2|$
      $WeightByThisCycle = FraudFactor * CycleSum$
      $Update\ Matrix\ M\ with\ WeightByThisCycle$
    end if
    if $Cycles\ Length\ is\ 3$ **then**
      $CycleSum = \sum Cycle\ Edge\ costs$
      $W_i = \frac{Edge\ i\ Weight}{CycleSum}$
      $D_i = 1 - |W_i - W_j|$
      $FraudFactor = \frac{\sum D_i}{3}$
      $WeightByThisCycle = FraudFactor * CycleSum$
      $Update\ Matrix\ M\ with\ WeightByThisCycle$
    end if
  end for
end for
Return S

We have converted a directed weighted simple graph to an undirected weighted simple graph using **Algorith 1**.
Steps for **Algorithm 1.**

- **Input** The input for the algorithm is edges and cycles in a graph.
- **Output** Algorithm will output the weighted Adjacency matrix.
- **STEP 1** Initializing NxN matrix to store the weighted Adjacency matrix
- **STEP 2-26** Iterating over all edges.
- **STEP 3-7** Identify all cycles that include the current edge.
- **STEP 9-24** Now iterate over all the cycles.
- **STEP 10** check for cycles of length 2
- **STEP 11** Compute summation over all edges including both the edges of each bidirectional edge and store it as cyclesum.
- **STEP 11** Like softmax function determining the weight for every edge.
- **STEP 12** Calculate the difference between the weights of two edges. If two edges have nearly equal contributions to the total sum of cycle cost, it is likely that there is fraud or circular trade involved. Based on this observation, we have formulated a cost function, as shown in **??**, in steps 10-14 of the algorithm. This cost function is designed to be at its maximum value when both edge weights are close to 0.5, indicating a potentially fraudulent activity.
- **STEP 13** Finding the Fraud Factor based on the weights.
- **STEP 14** The previously calculated fraction for the current edge is multiplied by the total cost of the cycle to adjust with cycle weights.
- **STEP 15** We proceed to update the Weighted undirected matrix for the current cycle, which is initially set to zero. This operation is carried out for all two cycles that the edge participates in.
- **STEP 17** Now check for cycles of length 3
- **STEP 18-23** In this step, we extend the approach from the previous step to 3 cycles. Now we have 3 softmax weights and find the difference as in **Step 20**.We divide this difference by 3 since 3 edges sum to 3 to calculate FraudFactor.
- **STEP 23** Finally, we update the matrix similar to 2 cycles.



(a) image a **(b)** image b

Figure 3.1: Our undirected edge cost function(Shown for 2 Cycles)

so in Figure 3.2, we can see the adjacency matrix as the output of **Algorithm 1**, which we later used to generate an undirected weighted simple graph, and we used this graph to do further analysis.

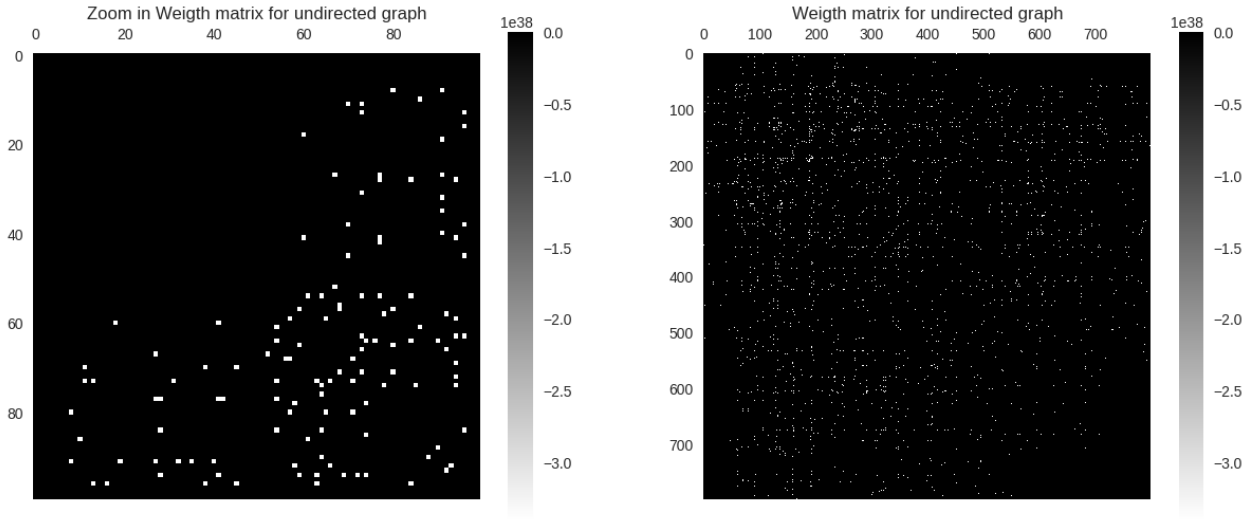


Figure 3.2: Weighted Undirected Adjacency Matrix

3.2.3 | Find embeddings using Node2Vec

We experimented with different values of "walk length", "num-walks," and "dimensions". After all the hits and trials, these were the values that were found to be our best find after considering the output for DBSCAN as well. Those values for each for walk length, num-walks, and dimensions are 10, 500, and 64, respectively.

3.2.4 | Cluster the embeddings using DBSCAN

We have run DBSAN multiple times with multiple values of EPS to determine the best possible value for the EPS plot for which we can see in Figure 3.3.

As the Figure 3.3 indicates, the value of 1.7 is given to the DBSCAN clustering algorithm for the node embeddings generated with values of walk-length, num-walks, and dimensions as 10, 500, and 64, for which we get the maximum number of clusters as 7.

We now have to further check/verify for those vertices which are present in these 7 clusters if they are part of the cycles in the consolidated directed simple graph.

also, there will be a cluster of more than 500 nodes which we have ignored by assuming that it indicates all the nodes which do not have any circular trade going on (Basically disconnected components in the undirected weighted simple graph).

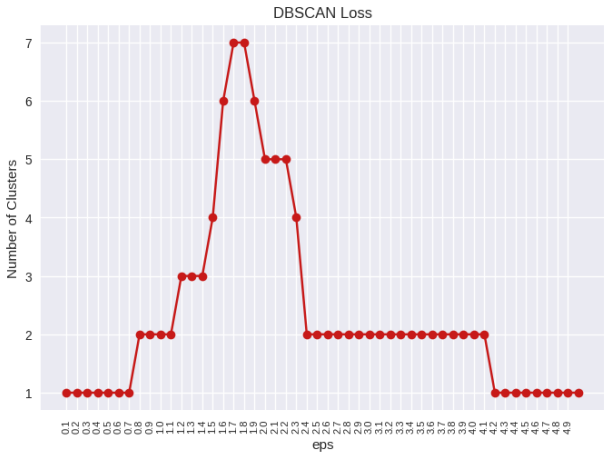


Figure 3.3: Graph Comparisons

4 | Results

4.1 | Cluster plot using DBSCAN

After the clustering by DBSCAN, we plotted the clusters and visualized them by reducing the dimensions using the PCA. As we can see from the plots below, 6 clusters are of different colors. These clusters represent the cycles in the data sets that may be part of circular trading, which we can verify by checking these nodes in the original graph that there are cycles present between these nodes, which we clustered using DBSCAN.

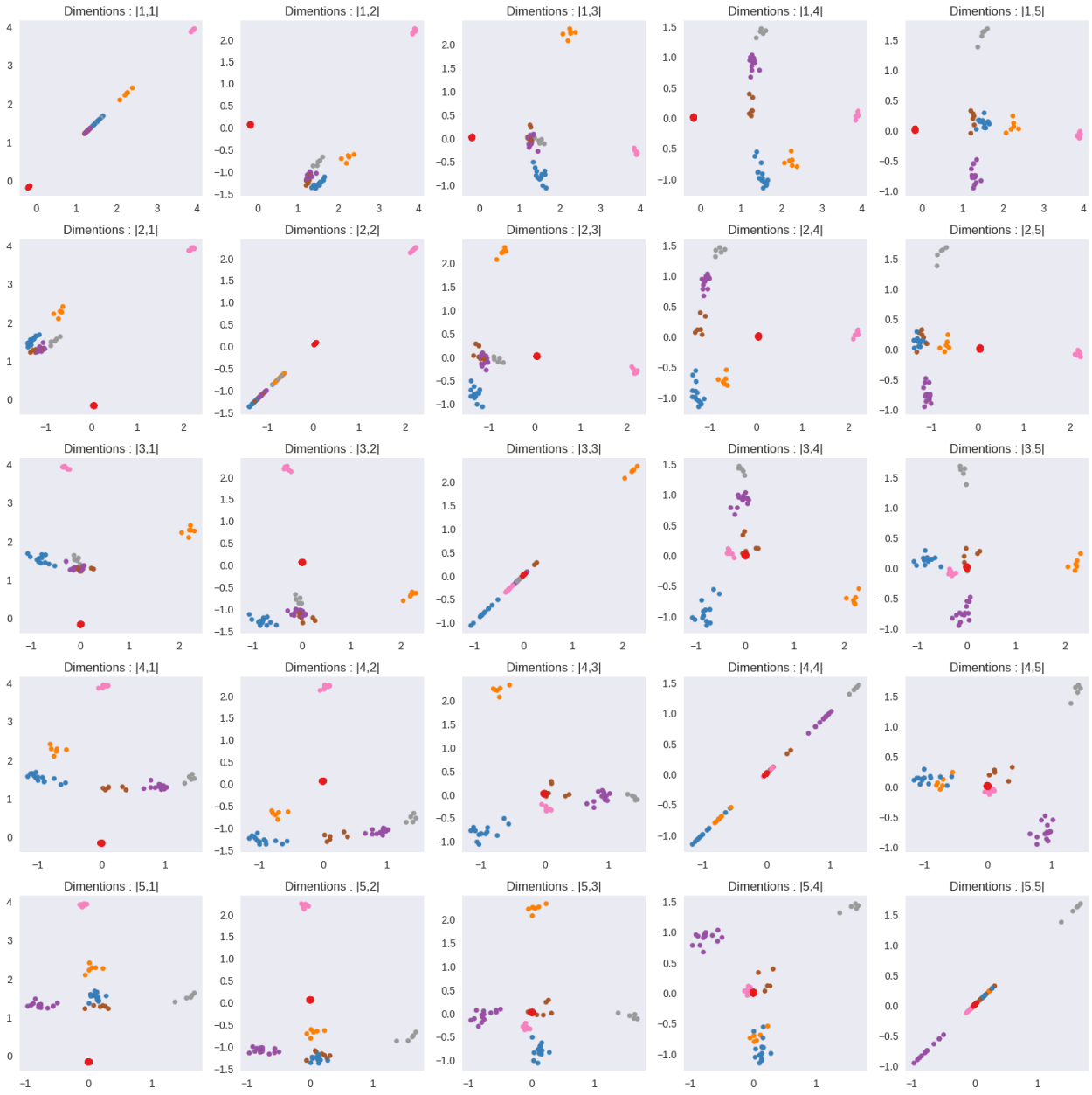


Figure 4.1: Plotting the Clustered Graph Embeddings using PCA

4.2 | Cycle Frequency analysis

We have plotted the clusters in a Figure 4.2 representing the Number of cycles present in the original graph for that cluster. We can infer that there is a cycle between the nodes we got from the DBSCAN clusters. And hence they might involve in circular trading.

For example, we can see that in cluster 1, we have 14 nodes present and that 14 nodes have 65 cycles present between them in an original graph, from which we can infer that our algorithm is detecting the clusters that might be involved in some kind of circular trading activity.

we are not sure how else we can verify this any other way as sir mentioned, this is a subjective question. We have tried to show whatever techniques we applied to come up with these results. and also gave a detailed explanation of the algorithm we came up with, as asked by sir in the class.

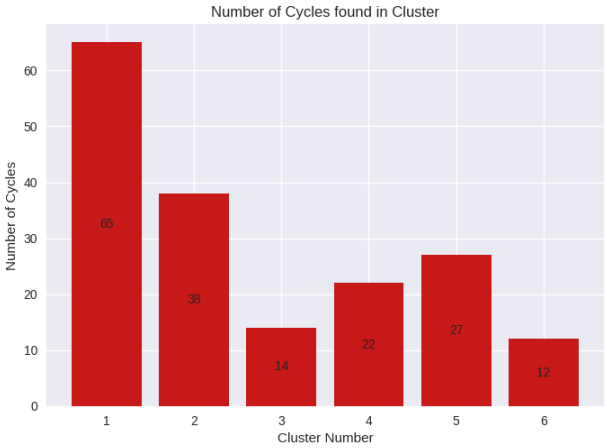


Figure 4.2: Number of cycles present in an original graph for each cluster