Identifying Clusters using Node2Vec, Spectral and GCN Embeddings

Rahul Verma
AI23MTECH11008
Email: ai23mtech11008@iith.ac.in

Pratik Yawalkar
AI23MTECH11006
Email: ai23mtech11006@iith.ac.in

Sarang Kukade
EM23MTECH11008
Email: em23mtech11008@iith.ac.in

Mahesh Deshmukhe
CC23MTECH11003
Email: cc23mtech11003@iith.ac.in

Aniruddha Paradkar
AI23MTECH13001
Email: ai23mtech13001@iith.ac.in

Code can be found here here.

Abstract

Clustering is a common unsupervised learning method that groups data points into clusters, aiming to maximize similarity among points within the same cluster while minimizing similarity with points from different clusters. This paper presents an investigation of different methods for identifying clusters within a payment dataset. The approach involves constructing a graph network from the dataset and employing three methods—Node2Vec, Graph Convolutional Network (GCN), and Spectral Clustering—to obtain node embeddings that represent the features of the nodes as low-dimensional vectors. By applying various clustering methods to the node embeddings, we effectively partition the payment data into meaningful clusters. Our results demonstrate that the proposed algorithms can successfully discern patterns and relationships within the data, providing insights into the underlying structure of the payment network.

1. Problem Statement

Financial transactions are an integral part of everyday life, with an enormous volume of online transactions being processed daily. Analyzing these transactions to uncover patterns and detect fraudulent activity presents a significant challenge due to the increasing complexity and volume of payment data. Traditional methods for fraud detection are proving insufficient in addressing these challenges.

In recent years, machine learning algorithms have shown significant promise in fraud detection by identifying patterns and anomalies in financial data. One effective approach utilizes clustering techniques applied to payment data. Clustering segments data points into groups of similar transactions, enabling the identification of potential fraud through the recognition of unusual behavior.

To create effective clusters, it is essential to gain a precise understanding of the data and its attributes. Given that payment data is often complex and high-dimensional, transforming it into low-dimensional representations using node embedding methods is beneficial. This preprocessing step enhances the clustering process, leading to more optimal results and improving the overall efficacy of fraud detection.

2. Description of the dataset

The dataset contains one file, ‘payments.csv‘, with 130,535 rows and three columns. Each row represents a specific transaction. Details of each column are listed below:

- ‘Sender‘ (Integer): ID of the sender initiating the payment. The column value ranges from 1001 to 2190.
- ‘Receiver‘ (Integer): ID of the receiver accepting the payment. The column value ranges from 1001 to 1887.
- ‘Amount‘ (Integer): Total amount of the payment made between the sender and receiver. The col-

umn value ranges from \$1,501 to \$2,124,500, with a mean (μ) of \$69,809.6 and a standard deviation (σ) of \$56,966.7. From **Figure 1**, it can be observed that most payments are below \$500,000.

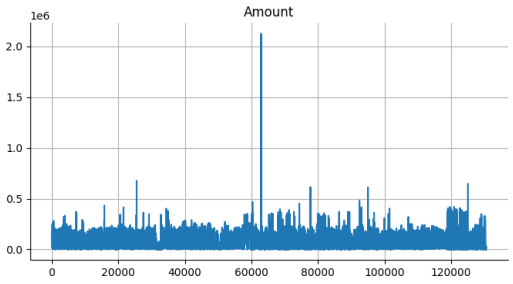

Figure 1. Plot of ‘Amount‘ column

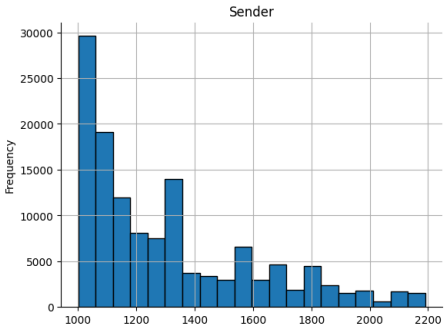

Figure 2. Frequency of transactions by senders

3. Representing Data as Directed Graph

Before implementing a clustering algorithm to identify distinct clusters, the data must first be represented as a graph network. This can be achieved using a directed, weighted graph where the weight of each edge reflects the total amount transferred from the sender to the receiver. **Figure 3** illustrates the graph G obtained from the payment data.

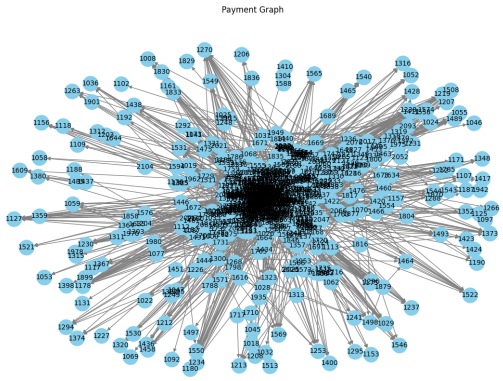

Figure 3. Graph G for the payment data

In addition to the overall graph, subgraphs were also plotted for a selection of randomly chosen nodes. **Figure 4** shows the subgraphs for nodes 1104, 1226, 1493, and 1434.

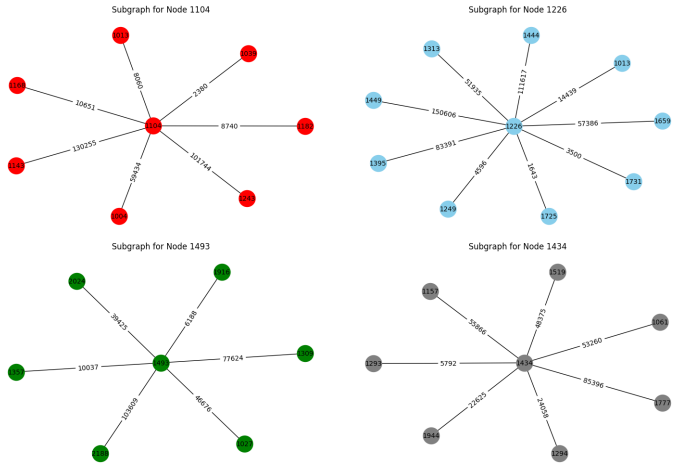

Figure 4. Subgraphs for nodes 1104, 1226, 1493, and 1434

The constructed graph G comprises 799 nodes and 5,358 edges, with a total edge weight sum of 306,671,493. To proceed with clustering the graph, it is important to first reduce the dimensionality of the data while preserving essential information. This can be achieved through node embedding, a technique that encodes nodes as low-dimensional vectors. This representation effectively encapsulates the positional relationships of nodes within the graph as well as the structural characteristics of their local neighborhoods.

4. Node Embedding Methods

In this section, we will explore different methods to perform node embeddings namely, **Node2Vec** and **Graph**

Convolution Network (GCN).

4.1. Node2Vec Algorithm

Node2Vec is a method for generating node embeddings that capture the structural and topological properties of a graph. The technique employs biased random walks through the graph, offering a balance between breadth-first search (BFS) and depth-first search (DFS) patterns. This adaptable approach enables Node2Vec to retain both local and global characteristics of the graph in its low-dimensional node representations.

Biased Random Walk The algorithm uses biased random walks that are guided by two parameters: the return parameter p and the in-out parameter q. These parameters influence the random walk by adjusting the likelihood of returning to the previous node (p) or venturing further from the current node (q).

Given a graph $G = (V, E)$ and a starting node u, the algorithm conducts random walks of length l from u. During each step, the next node v is chosen based on a transition probability:

$$\pi_{uv} = \begin{cases} \frac{\alpha_{pq}(t,v)}{Z} & \text{if } v \neq t, \\ 0 & \text{otherwise,} \end{cases}$$

In this context, t represents the previously visited node, and $\alpha_{pq}(t, v)$ is a function of the shortest path distance between nodes t, u, and v:

$$\alpha_{pq}(t,v) = \begin{cases} \frac{1}{p} & \text{if } d_{tv} = 0, \\ 1 & \text{if } d_{tv} = 1, \\ \frac{1}{q} & \text{if } d_{tv} = 2. \end{cases}$$

The parameter d_{tv} is the shortest path distance between nodes t and v, and Z is a normalization factor.

Effects of p and q Parameters The parameters p and q play a key role in controlling the random walk's exploration strategy:

- Return Parameter (p): The parameter p controls the probability of returning to the previous node. A higher value of p (e.g., $p > 1$) reduces the likelihood of backtracking and encourages exploration in new directions, similar to a DFS approach. Conversely, a lower value of p (e.g., $p < 1$) increases the likelihood of returning to the previous node, resembling a BFS approach.

- In-Out Parameter (q): The parameter q controls the likelihood of exploring nodes closer to or further from the starting node. A higher value of q (e.g., $q > 1$) discourages moving to nodes further from the starting node and promotes staying closer, similar to a BFS approach. A lower value of q (e.g., $q < 1$) encourages moving to nodes further from the starting node, resembling DFS.

The node2vec embedding was performed with the following parameters: embedding dimensionality set to 64, walk length of 50, number of walks per node set to 100, return parameter $p = 10$, and in-out parameter $q = 1$. The resulting 64-dimensional embeddings were then compressed to two dimensions using Principal Component Analysis (PCA) for visualization purposes. The 2D plot of the embeddings is shown in **Figure 5**.

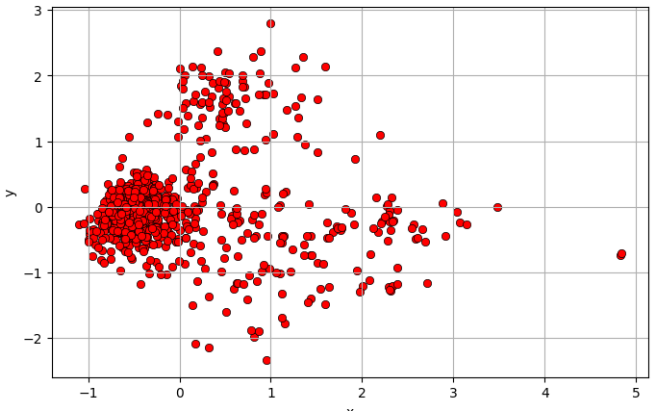

Figure 5. 2D visualization of Node2Vec embeddings

Algorithm used for generating Node2Vec embedding is given below.

Algorithm 1 node2vec Walk Algorithm

Input: Graph $G' = (V, E, \pi)$, Start node u, Length l
Output: A walk of length l starting from node u

Initialize walk to $[u]$
for walk iter = 1 to l **do**
  $curr \leftarrow walk[-1]$
  $V_{curr} \leftarrow$ GetNeighbors$(curr, G')$
  $s \leftarrow$ AliasSample(V_{curr}, π)
  Append s to walk
end for
return walk

4.2. Graph Convolutional Network (GCN)

Graph Convolutional Networks (GCN) are a class of neural network models designed for learning node embeddings from graph-structured data. These models apply convolution-like operations directly on the graph, allowing them to capture both the structural relationships and node features in the data.

Graph Convolution Layer The core of a GCN is the graph convolution layer, which aggregates and transforms information from a node's neighbors. Given an adjacency matrix A, a degree matrix D, and an input feature matrix X, a graph convolution layer at depth k is defined as:

$$H^{(k)} = \sigma\left(D^{-1/2}AD^{-1/2}H^{(k-1)}W^{(k)}\right),$$

where $H^{(k-1)}$ is the output from the previous layer (or the initial feature matrix X for the first layer), $W^{(k)}$ is the weight matrix for layer k, and σ is a non-linear activation function (e.g., ReLU). The matrices $D^{-1/2}$ are used for normalization to mitigate issues with different node degrees.

Encoder-Decoder Architecture The GCN encoder-decoder architecture consists of two main components:

- **Encoder**: The encoder uses multiple graph convolution layers to aggregate information from a node's neighbors and transform it based on the graph structure and node features. Given an adjacency matrix A, a degree matrix D, and an input feature matrix X, the output of the final convolution layer provides the learned node embeddings.

- **Decoder**: The decoder leverages the node embeddings to generate node-level outputs depending on the specific task. Common decoders include linear transformations or deconvolution layers, which can be used for tasks such as graph reconstruction, node classification, or link prediction.

Algorithm 2 Graph Convolutional Network (GCN)

Input: Graph $G = (V, E)$ with adjacency matrix A, degree matrix D, feature matrix X, depth K, non-linearity σ, learnable weight matrices $W^{(k)}$ for $k = 1, \ldots, K$
Output: Node-level embeddings Z

$H^{(0)} \leftarrow X$
for $k = 1$ to K **do**
  $H^{(k)} \leftarrow \sigma\left(D^{-1/2}AD^{-1/2}H^{(k-1)}W^{(k)}\right)$
end for
$Z \leftarrow H^{(K)}$
return Z

For generating node embeddings using Graph Convolutional Networks (GCN), we used the adjacency matrix A of the graph and an embedding dimensionality of 64. The embeddings were learned using an encoder-decoder architecture.

In **Figure 6**, a small portion of the adjacency matrix A of the graph is shown. This provides insight into the connections between nodes in the graph and serves as the input for the GCN.

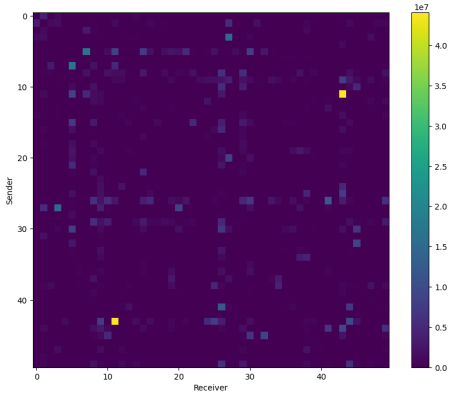

Figure 6. A small portion of the adjacency matrix A of the graph G

After training the GCN, the learned node embeddings were projected into a 2D space for visualization using dimensionality reduction techniques. The 2D plot of the GCN embeddings is shown in **Figure 7**.

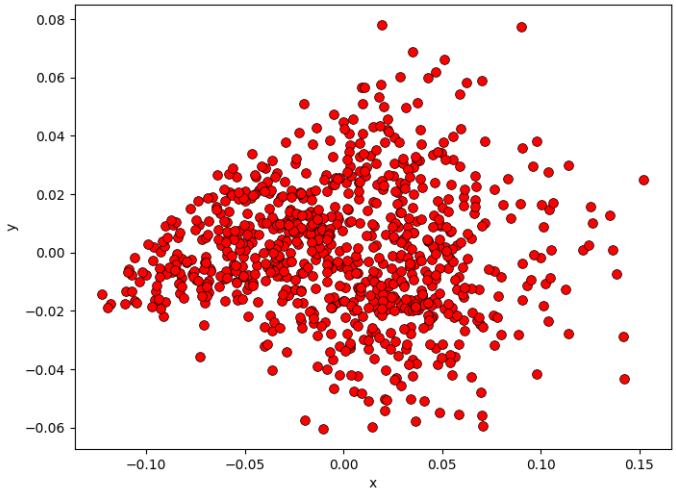

Figure 7. 2D plot of the GCN embeddings

5. Clustering Algorithms

Given the node embeddings of graph G, various clustering methods can be applied to identify distinct clusters within the graph. In this study, we employed two primary clustering algorithms: Spectral Clustering and K-means Clustering given below.

5.1. Spectral Clustering Algorithm

Spectral clustering is a clustering algorithm that uses the spectrum (eigenvalues and eigenvectors) of the graph Laplacian to identify clusters in the graph. Given an adjacency matrix A, a degree matrix D, and the graph Laplacian L,

the first k eigenvectors of L are computed to form a matrix U, representing the graph in a lower-dimensional space.

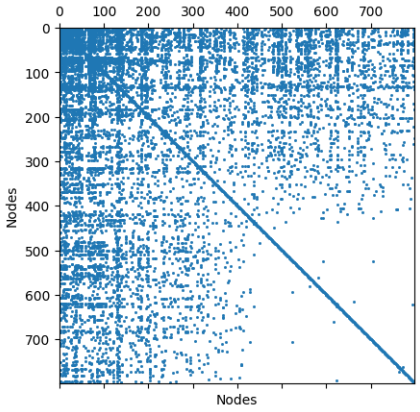

Figure 8. Graph Laplacian matrix L

In **Figure 8**, the graph Laplacian matrix L is shown, which captures the structure and connectivity of the graph. Spectral clustering uses L to compute node embeddings in a lower-dimensional space.

5.2. K-means Clustering

K-means clustering divides data points into k clusters according to their similarities. The algorithm iteratively assigns each data point to the nearest cluster centroid and then updates the cluster centroids based on the mean of the assigned data points.

Effects of Parameters

- **Number of clusters (k)**: Dictates the level of detail in clustering; a higher k leads to more, smaller clusters, while a lower k yields fewer, larger clusters.
- **Initialization**: Various initialization techniques (e.g., K-means++) provide different starting points for the algorithm, impacting its stability and efficiency.
- **Distance Metric**: The chosen distance measure (e.g., Euclidean, Manhattan) affects the assignment of data points to clusters.
- **Convergence Criteria**: Convergence is reached when cluster assignments remain constant or a predefined maximum number of iterations is achieved. Different criteria influence the iteration count and the stability of the outcomes.

Algorithm 3 SpectralClustering(D, s, k)

Input: $\{x_1, \ldots, x_n\}$: set of data points, $s(x, x')$: similarity function, k: number of desired clusters
Output: Clusters $A_1, \ldots, A_k$

Construct a similarity graph G from the dataset using function s.
Form the weighted adjacency matrix W from the graph.
Compute the graph Laplacian matrix L.
Compute the first k eigenvectors of L and store them as columns of matrix $U \in \mathbb{R}^{n \times k}$.
for $i = 1$ to n **do**
    Extract the i-th row of matrix U and store it as vector $z_i \in \mathbb{R}^k$.
end for
Apply k-means clustering to vectors z_i and obtain clusters $C_1, \ldots, C_k$.
for $i = 1$ to k **do**
    Define cluster A_i as the set $\{x_j \mid z_j \in C_i\}$.
end for
return $A_1, \ldots, A_k$

6. Results

6.1. Results from Node2Vec

We conducted K-means clustering on the node embeddings derived from the Node2Vec algorithm, using a range of cluster numbers k from 2 to 9. Our analysis revealed that the inertia curve exhibits a characteristic 'knee' near $k = 3$, suggesting an optimal value of $k = 3$. This is illustrated in **Figure 9**, which presents the three clusters identified through K-means clustering.

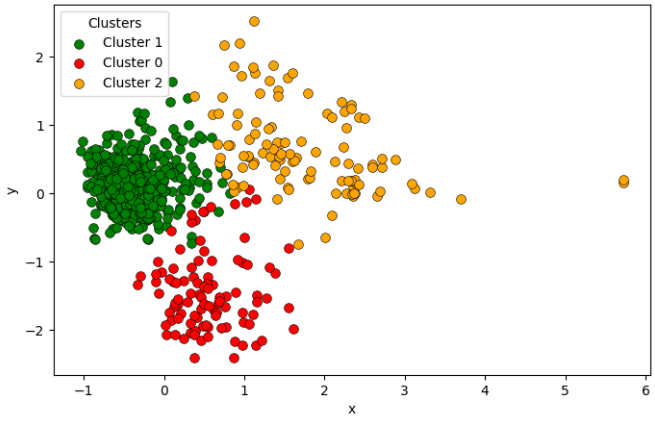

Figure 9. Visualization of clusters obtained using Node2Vec

6.2. Results from GCN

To get Node embeddings using GCN, we used encoder-decoder architecture consisting of 2 layers of GCNConv.

The configuration of the GCN model was as follows:

- Hidden Space Size: 64
- Optimizer: Adam
- Learning Rate: 0.01
- Epochs: 200

Various learning rates were tested in the range of 0.0001 to 0.1, with the best results obtained using a learning rate of 0.01.

After training the model, the learned node embeddings were clustered using the k-means algorithm with $k = 3$ clusters. Clusters formed using GCN embedding are shown in **Figure 10** below.

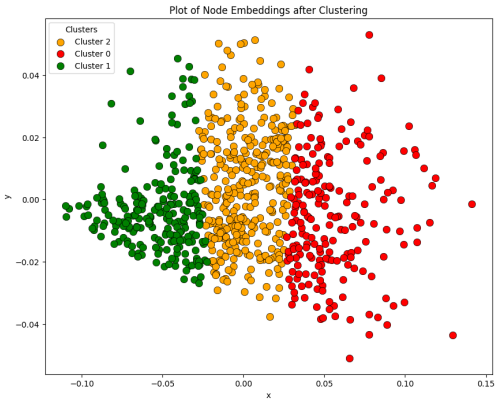

Figure 10. Visualization of clusters obtained using GCN

6.3. Results from Spectral Clustering

First, we computed the first 16 eigenvalues of the normalized Laplacian matrix. **Figure 11** presents a plot of the eigenvalues against the corresponding k-values, indicating that a choice of $k = 3$ clusters is suitable for the data.

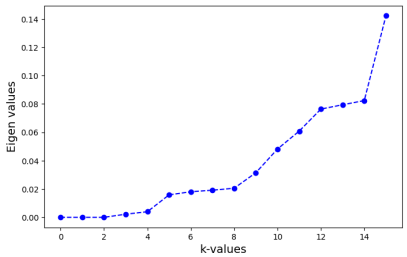

Figure 11. Eigenvalues vs. k-values plot indicating the choice of $k = 3$

The results of clustering using spectral clustering are visualized in **Figure 12**.

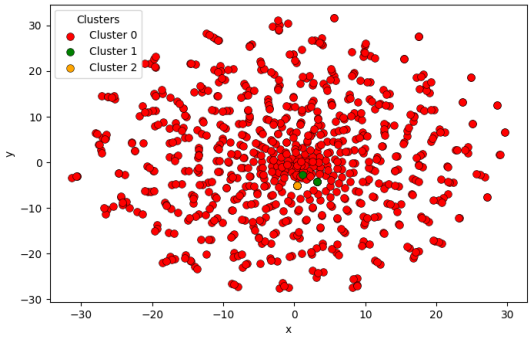

Figure 12. Clusters obtained using spectral clustering

7. Conclusion

This paper investigated Node2Vec, Graph Convolutional Networks (GCN), and Spectral Clustering methods for clustering nodes in a payment dataset. Our findings demonstrated that these approaches effectively identified meaningful clusters, offering insights into the payment network.

References

[1] Hamilton, William L. et al. “Representation Learning on Graphs: Methods and Applications.” IEEE Data Eng. Bull. 40 (2017).

[2] Roux, Daniel Pérez, Boris Moreno, Andres Villamil, Pilar Figueroa, César. (2018). Tax Fraud Detection for Under-Reporting Declarations Using an Unsupervised Machine Learning Approach. 215-222. 10.1145/3219819.3219878.

[3] A Gentle Introduction to Graph Neural Networks https://distill.pub/2021/gnn-intro/