Identify Clusters Using Node2Vec, Spectral, and GCN Embeddings

Nikhil Kumar Patel
CS23MTECH11013
Department of CSE
IIT Hyderabad

Ayush Agarwal
AI23MTECH11002
Department of AI
IIT Hyderabad

***Abstract*—In this study, we investigate the application of graph embedding techniques for identifying clusters in a financial transaction network. The dataset comprises sender-receiver pairs and transaction amounts, representing interactions within the network. We explore three graph embedding methods: Node2Vec, Spectral Embedding, and Graph Convolutional Networks (GCNs). These techniques transform the graph structure into low-dimensional vector representations, allowing for efficient analysis and clustering. Our goal is to evaluate the effectiveness of these embedding methods in uncovering meaningful clusters within the financial transaction network.**

I. Problem Statement:

Financial transaction networks are inherently complex, consisting of numerous interconnected entities engaged in monetary exchanges. Identifying clusters within these networks is crucial for various applications, such as fraud detection, risk assessment, and customer segmentation.

In this study, we aim to address the challenge of clustering within a financial transaction network using graph embedding techniques. Specifically, we seek to accomplish the following objectives:

1) Evaluate the performance of Node2Vec, Spectral Embedding, and Graph Convolutional Networks (GCNs) in capturing the structural information of the transaction network.
2) Compare the effectiveness of these embedding methods in uncovering clusters within the network.

II. Description Of Dataset

A. Introduction

The Payments dataset consists of a collection of financial transactions between various entities. Each transaction record contains information about the sender, receiver, and the amount transferred.

B. Dataset Details

The dataset is provided in a CSV file named *payments.csv*. It contains the following columns:

- **Sender**: The identifier of the entity making the payment.
- **Receiver**: The identifier of the entity receiving the payment.
- **Amount**: The monetary value of the transaction, indicating the amount transferred from the sender to the receiver.

C. Graph Representation

The dataset can be represented as directed and un-directed graph, where nodes represent entities (senders and receivers) and edges represent transactions between them. The direction of the edges signifies the flow of money from the sender to the receiver.

D. Graph Statistics

- Total number of transactions: 130,535
- Total number of unique seller-buyer pairs: 799

III. Algorithms

A. Node2Vec

The Node2Vec Algorithm processes the data through the following steps. Please refer to Appendix to see the Algorithm.

- **Input and Output**: The algorithm starts by specifying the input and output. It takes a graph and the number of clusters as input and outputs clusters of nodes.
- **Initialization**: Loads data into a DataFrame and constructs a graph G.
- **Node2Vec Training**: Initializes and trains a Node2Vec model with specified parameters.
- **Embedding Extraction**: Extracts embeddings for each node from the trained model.
- **Clustering**: Applies k-means clustering to the embeddings.
- **Assignment**: Assigns nodes to clusters based on the results from the k-means algorithm.
- **Visualization**: Visualizes the clusters within the graph using a layout algorithm.
- **t-SNE Embedding**: Applies t-SNE to the embeddings for dimensionality reduction and visualization.

B. Spectral Embedding

The spectral clustering algorithm processes the data through the following steps. Please refer to Appendix to see the Algorithm.

1) **Graph Construction:** Construct an undirected graph $G = (V, E)$ where V is a set of vertices representing entities, and E is a set of edges representing connections between these entities, with weights corresponding to the interaction strength (e.g., transaction amounts).

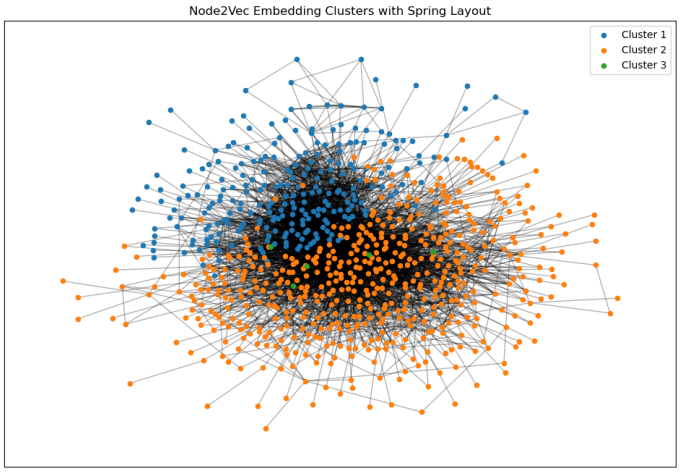

Fig. 1. Node2vec embedding clusters

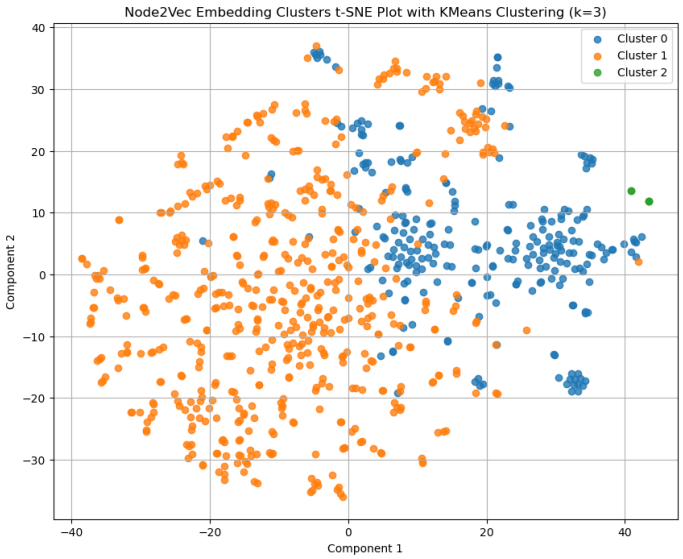

Fig. 2. Node2vec embedding clusters t-SNE plot

2) **Adjacency Matrix Construction:** Generate the adjacency matrix A of the graph, where A_{ij} represents the weight of the edge between vertices i and j. If there is no edge between i and j, then $A_{ij} = 0$.
3) **Degree Matrix Construction:** Compute the degree matrix D which is diagonal where each element D_{ii} is the sum of the weights of the edges connected to vertex i.
4) **Laplacian Matrix Construction:** Compute the Laplacian matrix L defined by $L = D - A$.
5) **Normalized Laplacian Matrix:** Compute the normalized Laplacian matrix $\mathcal{L}$ as:

$$\mathcal{L} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

where I is the identity matrix.
6) **Eigenvalue Decomposition:** Perform eigenvalue decomposition on the normalized Laplacian matrix $\mathcal{L}$ to obtain its eigenvalues and eigenvectors. Select the k smallest eigenvalues and their corresponding eigenvectors.
7) **Matrix of Eigenvectors:** Form matrix $X \in \mathbb{R}^{n \times k}$ from the k eigenvectors.
8) **Row Normalization:** Normalize each row of the matrix X to have unit length, resulting in matrix Y.
9) **Clustering:** Treat each row of Y as a point in $\mathbb{R}^k$, and cluster them into k clusters via k-means clustering.
10) **Cluster Assignment:** Assign the original points (vertices in the graph) to clusters based on the clustering of the rows in Y.
11) **Visualization:** Visualize the clusters using different techniques such as Spectral Embedding, t-SNE, and plotting clusters in the graph layout.

C. GCN

The GCN Algorithm processes the data through the following steps. Please refer to Appendix to see the Algorithm.

- **Data Loading**: Data is loaded from a CSV file into a DataFrame.
- **Graph Construction**: A graph G is constructed from the DataFrame, where 'Sender' and 'Receiver' columns define the nodes, and 'Amount' defines the edge weights.
- **GCN Model Definition**: A Graph Convolutional Network (GCN) model is defined and initialized with two convolutional layers, aimed at transforming node features into a lower-dimensional space capturing the graph's topology.
- **Model Transfer**: The model is transferred to an appropriate computational device (GPU or CPU) for processing.
- **Feature Preparation**: An identity matrix of size equal to the number of nodes plus one is created to serve as featureless input features. Edges of the graph are converted into a tensor format suitable for processing by the model.
- **Model Execution**: The GCN model is run in evaluation mode to generate embeddings for each node without further modifying the network weights.
- **Clustering**: K-means clustering is applied to the node embeddings to identify clusters within the graph.
- **Graph Visualization**: The network graph is visualized using a layout algorithm, with nodes colored according to their cluster membership.
- **t-SNE Embedding**: t-SNE is used to reduce the dimensionality of the embeddings for visualization purposes, aiding in the interpretation of clustering results in a 2D space.

IV. Results

In this section we will see the results of each pf the embedding algorithm that we found.

A. Node2Vec Embedding

As depicted in Plot 1, we present a Spring layout visualization of our clusters obtained through Node2Vec embeddings.

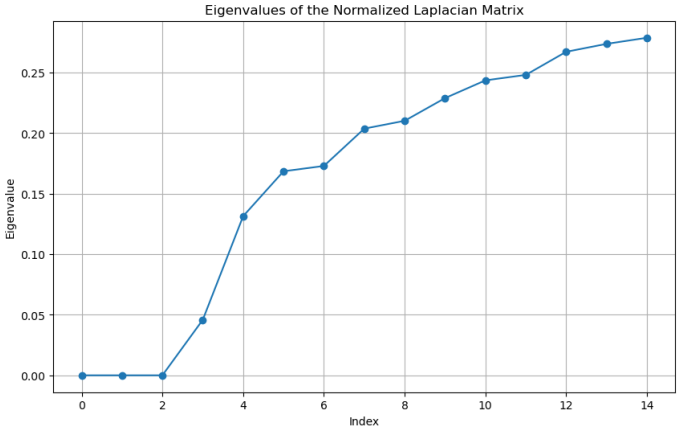

Fig. 3. Eigenvalues of the Normalized Laplacian Matrix

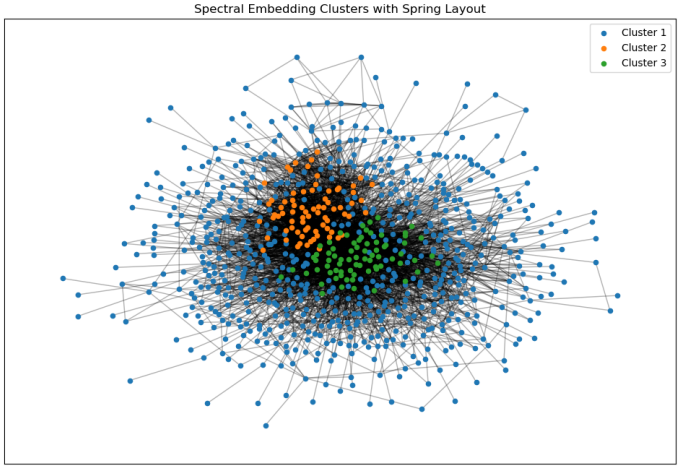

Fig. 4. Spectral embedding clusters plot

Additionally, Plot 2 showcases the t-SNE layout resulting from our Node2Vec algorithm. Employing k-means clustering and leveraging the silhouette score, we determined that utilizing 3 clusters yielded the most optimal results. This choice of k allowed us to generate a visually appealing plot, affirming the efficacy of our clustering approach.

B. Spectral Embedding

After computing the normalized Laplacian matrix, we proceeded to visualize its top 15 eigenvalues, as illustrated in Plot 3. Remarkably, we observed an inflection point, or 'elbow', after the third eigenvalue. This pivotal observation guided our decision to adopt three clusters for our subsequent clustering analysis.

To provide visual validation of our clustering approach, we generated Spring and t-SNE plots, presented in Plot 4 and Plot 6 respectively. These plots offer intuitive insights into the distribution and separation of data points within our dataset, corroborating the appropriateness of selecting three clusters.

Furthermore, to offer a more granular understanding of our clustered data, we created 2D projections of our clusters, depicted in Plot 5. This visualization facilitates a direct examination of the spatial arrangement and inter-cluster relationships, aiding in the interpretation of our clustering results.

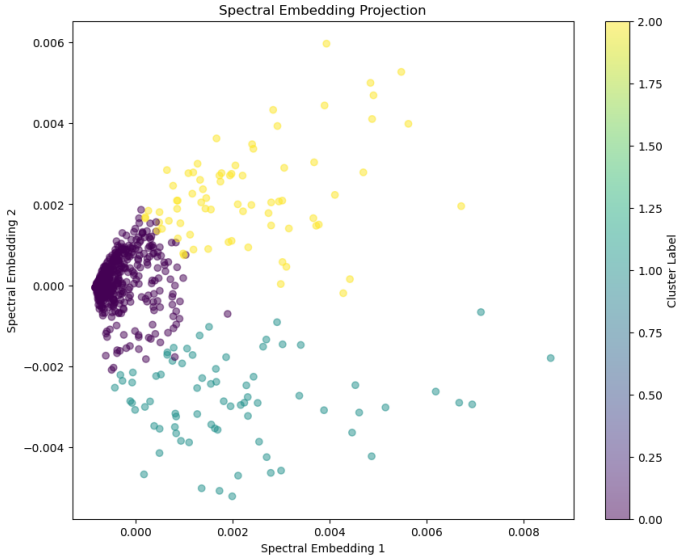

Fig. 5. Node2vec embedding Projection

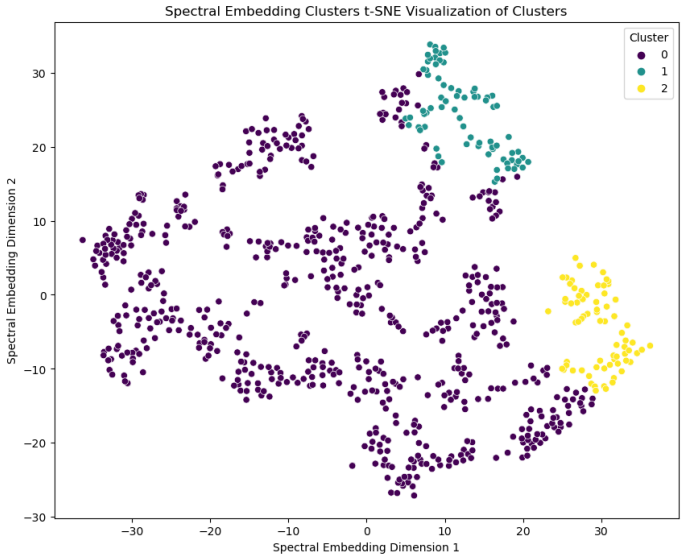

Fig. 6. Spectral embedding clusters t-SNE plot

C. GCN

For the Graph Convolutional Network (GCN) embedding method, we assessed the quality of our clusters by computing the silhouette score using k-means clustering. Remarkably, we discovered that setting the number of clusters, k, to 3 resulted in the highest silhouette score, indicating optimal clustering performance.

To visually corroborate our findings, we generated a t-SNE plot, depicted in Plot 7. This plot provides a compelling visual

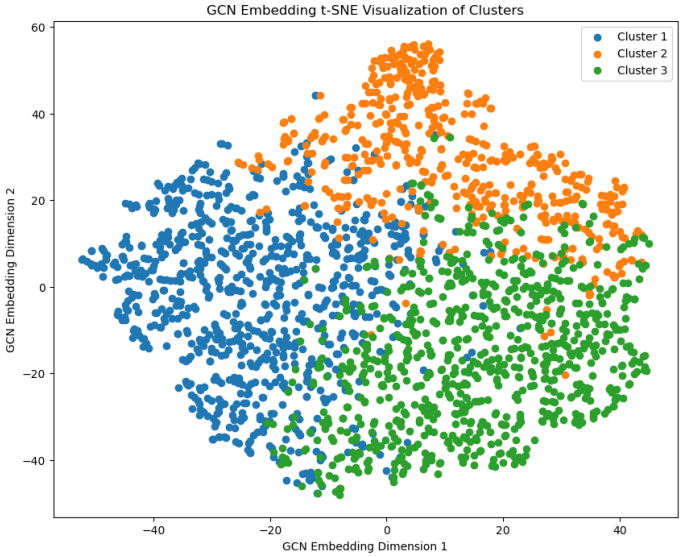

Fig. 7. GCN Embedding t-SNE Visualization of Clusters

representation of our clustered data, offering clear delineation and insight into the distinct groupings within our dataset.

The combination of our silhouette score analysis and the t-SNE plot underscores the efficacy of our clustering approach, affirming the meaningfulness and robustness of the identified clusters.

REFERENCES

[1] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. CoRR, abs/1607.00653, 2016.
[2] https://github.com/eliorc/node2vec.
[3] https://networkx.org/documentation/stable/install.html.
[4] Michael Jordan Andrew Ng and Yair Weiss. On spectral clustering: Analysis and an algorithm. NIPS'01: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, page 849–856, 2001.
[5] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. Feb. 22, 2017.
[6] Fenyu Hu et al. Hierarchical Graph Convolutional Networks for Semi-supervised Node Classification.

APPENDIX

A. Algorithms

Algorithm 1 Graph Construction Algorithm from Payments Dataset

Input: Payments dataset D (e.g., payments.csv)
Output: Graph G
Preprocessing:
$D \leftarrow$ Load payments dataset from input file
Construct Graph:
$G \leftarrow$ Create an empty graph
for each row r in D **do**
    $sender \leftarrow$ Extract sender from r
    $receiver \leftarrow$ Extract receiver from r
    $amount \leftarrow$ Extract amount from r
    $G \leftarrow$ Add edge from $sender$ to $receiver$ with weight $amount$
end for
Return G

Algorithm 2 GCN for Node Clustering

Input: Graph $G = (V, E, w)$ where $w : E \to \mathbb{R}^+$
Output: Clusters $C_1, C_2, \ldots, C_k$
$df \leftarrow$ read_csv('Payments_Q1.csv')
$G \leftarrow$ from_pandas_edgelist(df, 'Sender', 'Receiver', 'Amount')
Define $GCN(\ell, \Theta)$ with layers ℓ and parameters Θ
$\Theta \leftarrow \{\text{GCNConv}(|V| + 1, 16), \text{GCNConv}(16, 8)\}$
$model \leftarrow GCN()$
$X \leftarrow$ torch.eye($|V| + 1$)
$edge\_index \leftarrow$ torch.tensor(E).$t()$.$contiguous()$
$data \leftarrow$ Data($x = X, edge\_index = edge\_index$)
$model.eval()$
$out \leftarrow$ model($data$)
$kmeans \leftarrow$ KMeans(3, random_state $= 42$, n_init $= 10$)
$pos \leftarrow$ nx.kamada_kawai_layout(G)
Plot graph with nodes colored by $labels$
$tsne \leftarrow$ TSNE($n\_components = 2$, random_state $= 42$)
$embeddings \leftarrow tsne$.fit_transform($out$)
Plot t-SNE embeddings colored by $labels$

Algorithm 3 Spectral Clustering Algorithm

Input: Graph $G = (V, E)$ with vertices V and weighted edges E
Output: Clusters of vertices $C_1, C_2, \ldots, C_k$
procedure SPECTRALCLUSTERING(G, k)
    $A \leftarrow$ adjacency_matrix(G)
    $D \leftarrow$ diag($\sum_j A_{ij}$ for each i)
    $L \leftarrow D - A$
    $\mathcal{L} \leftarrow D^{-1/2} L D^{-1/2}$
    [values, vectors] $\leftarrow$ eig($\mathcal{L}$)
    $X \leftarrow$ vectors[: k]
    $Y \leftarrow$ normalize_rows(X)
    $kmeans \leftarrow$ KMeans(k)
    $labels \leftarrow kmeans$.fit_predict(Y)
    for $i \leftarrow 1$ to k **do**
        $C_i \leftarrow \{v \mid labels[v] = i\}$
    end for
    Return $C_1, C_2, \ldots, C_k$
end procedure

Algorithm 4 Node2Vec Clustering Algorithm

Input: Graph $G = (V, E)$, Number of clusters k
Output: Clusters $C_1, C_2, \ldots, C_k$
Initialization:
Load data into DataFrame df
$G \leftarrow$ Construct graph from df using nodes 'Sender', 'Receiver' and edges 'Amount'
Node2Vec Training:
$params \leftarrow$ {'dimensions': 64, 'walk_length': 30, 'num_walks': 200, 'workers': 4}
$node2vec \leftarrow$ Node2Vec($G, params$)
$model \leftarrow node2vec$.fit(window $= 10$, min_count $= 1$)
Embedding Extraction:
$X \leftarrow$ array(list($model.wv[n]$ for n in $G.nodes()$))
Clustering:
$kmeans \leftarrow$ KMeans(k, 'random_state': 42, 'n_init': 100)
$clusters \leftarrow kmeans$.fit_predict($X$)
Assignment:
for $i \leftarrow 1$ to k **do**
    $C_i \leftarrow \{v \mid clusters[v] = i\}$
end for
Visualization:
$pos \leftarrow$ nx.kamada_kawai_layout(G)
Visualize clusters in G using positions pos and clusters C_i
t-SNE Embedding:
$tsne \leftarrow$ TSNE($n\_components = 2$, 'random_state': 42)
$Y \leftarrow tsne$.fit_transform(X)
Return $C_1, C_2, \ldots, C_k$