Identify clusters using (Node2Vec Embedding, Spectral, and GCN) embeddings

Morey Piyush Prabhakar
AI23MTECH14003

Dindorkar Mayuresh Rajesh
CS23MTECH14007

Sanyam Kaul
CS23MTECH14011

Shrenik Ganguli
CS23MTECH14014

1. ABSTRACT

The aim of this assignment is to identify clusters by using Node2Vec, Spectral and GCN node embeddings.

2. Problem Statement

We are provided with a CSV data file - Payments.csv. The file contains three columns. The first column Provides the label of the sender initiating the transaction. The second column provides information about the receiver, which is receiving the amount sent by the sender. The third column informs us about the total amount sent from sender to receiver. Each row in the CSV file, therefore, depicts one transaction. The goal of the assignment is as follows: -

- Formulate the data provided in the CSV file into a graph data structure.
- Perform Node2Vec, Spectral and GCN embeddings on the nodes.
- Cluster the nodes into groups by applying a clustering algorithm on the calculated node embeddings.

3. Description of the dataset

The dataset provided comprises two distinct files: 'Payments.csv' and 'bad_sender.csv'

'Payments.csv': Consists of 130,535 rows and three columns - 'Sender', 'Receiver' and 'Amount', where each row denotes a transaction between a particular buyer and seller, whereas the columns denote the Buyer ID, Seller ID and the Transaction Amounts. There are 799 unique nodes in the dataset.

4. Methodology of construction of graph

We have used the Networkx library to build graph G based on the data provided. Graph G is undirected. Each node depicts a sender/receiver. The edge between 2 nodes is weighted by the amount details provided in the data. In case there are multiple transactions between 2 nodes the amounts of each transaction are added to calculate the final single total amount. That value is used as the weight of the edge between the respective nodes. Graph G, therefore, is simple and undirected. The code implementation provides an excellent visualisation of the graph constructed.

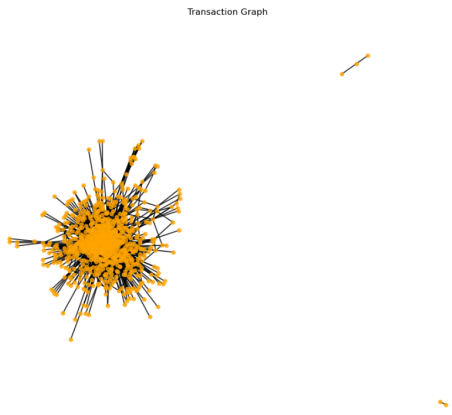

Figure 1. Visualization of graph

5. Methodology of Embeddings

As per the problem statement, we have to calculate three separate embeddings for the nodes of G. We are happy to inform you that we have successfully implemented all three embedding techniques from scratch and are not using any in-build/ 3rd party library functions to get the node embeddings. Following are the details of the procedure adopted by us to achieve node embeddings

5.1. Node2Vec implementation

For our scenario, we have taken the number of walks to be performed to 100 and the maximum walk length to be 4. We have implemented a simulation function which sim-

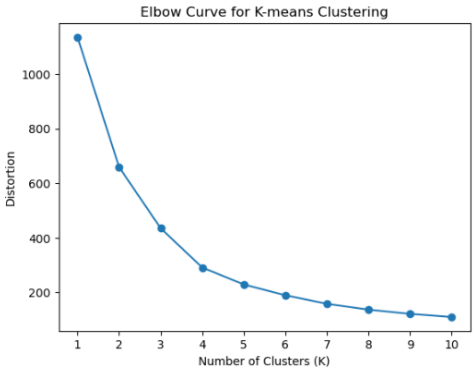

Figure 2. Elbow curve in case of Node2Vec embedding

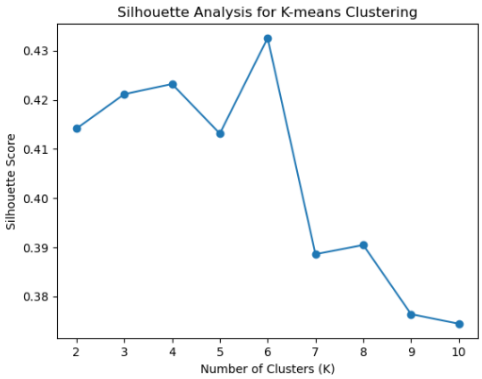

Figure 3. Silhouette analysis graph in case of Node2Vec embedding

ulates the walk performed on the graph and uses the list of 100 walks performed on the graph. Each walk consists of 4 nodes visited in the walk. The Node2Vec function implemented by us first calculates the walks on the graph. It then calculates the Word2Vec embeddings. Eventually, it calculates Node2Vec embeddings using the word2Vec embeddings and the walk calculated earlier. The function eventually returns the nodes' embeddings, each calculated in 64 dimensions.

We have also calculated the Node2Vec embeddings by using sklearn's function to compare our embedding with sklearn's embedding. We are happy to inform you that upon performing dimensionality reduction and applying K-means clustering, our embedding and sklearn's embedding give identical results. This shows the accuracy of our implementation.

Dimensionality reduction is performed to reduce the dimension of the embedding from 64 to 2. We utilise PCA to do the reduction. It, therefore, reduces the dimension to 2 by keeping all the critical information from the 64-D embedding. The two dimensions of final embedding also make it applicable for plotting purposes.

We have plotted the Elbow curve and Silhouette Analysis graph to get to the optimal value of K for the K-means clustering algorithm. Multiple executions of the code show that the elbow curve is gradual without any clear bend. However, the Silhouette analysis gives the highest score to k equal to 5, 6 (sometimes five and sometimes 6). We have also provided the cluster vizualization in code implementation.

Despite having a good Silhouette score on K = 5 or 6, we have observed that the clusters tend to overlap each other when we use Node2Vec embedding. This result is the same when we use the sklearn's Node2Vec implementation.

5.2. Spectral Clustering

The overall methodology of our work is the same as we have followed in the case of Node2Vec. We have implemented the function from scratch to calculate the spectral embedding of each node. We first calculate the adjacency matrix in the graph, which is used to calculate the Laplacian matrix. Then, we calculate the eighen values on the Laplacian matrix and sort them to eventually get the spectral embeddings.

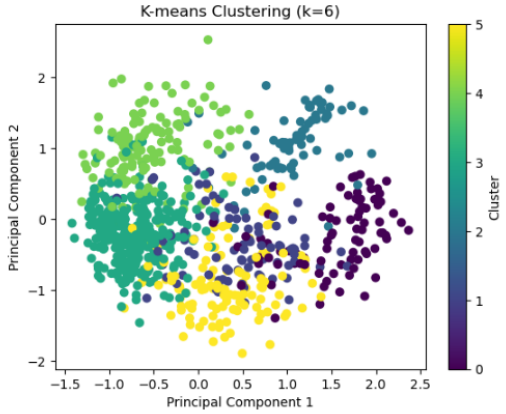

Figure 4. K-Means clustering in case of Node2Vec embedding

The calculated spectral embeddings for each node are of 64 dimensions, the same as in the case of Node2Vec. We perform PCA on the embeddings for dimensionality reduction and get the final 2-dimensional embeddings. We plot the elbow and Silhouette analysis curves and get the optimal K values of 5,6.

The key observation in this case is that, unlike Node2Vec, the final clusters do not overlap and depict clear boundaries. We have also done the same experiment by calculating the spectral embeddings from sklearn's in-built function, and the results are identical. Again, this shows the accuracy of our implementation of spectral embedding.

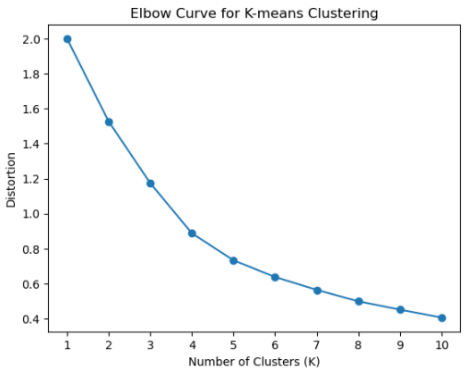

Figure 5. Elbow curve in case of Spectral embedding

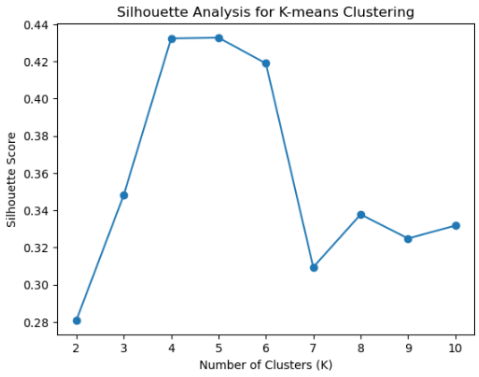

Figure 6. Silhouette analysis graph in case of Spectral embedding

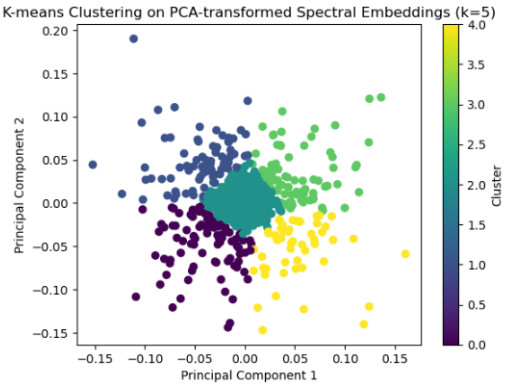

Figure 7. K-Means clustering in case of Spectral embedding

5.3. Graph Convolution Network

In Graph Convolution Networks, the first step is to develop smart features for our nodes, which are constructed using the graph. In our case, we have constructed seven features which include: -

- DegreeCentrality
- TotalTransactionAmount Sender

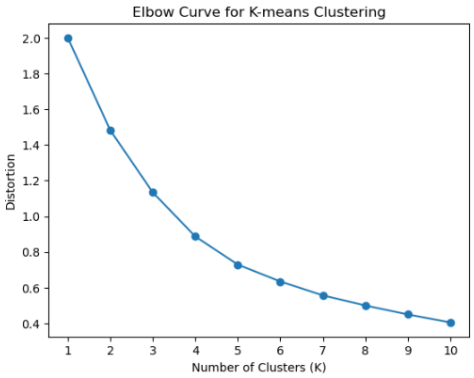

Figure 8. Elbow curve in case of GCN embedding

- TotalTransactionAmount Receiver
- AverageTransactionAmount Sender
- AverageTransactionAmount Receive
- TransactionFrequency Sender
- TransactionFrequency Receiver

These constructed features incorporate key and relevant details from the graph for each node. A data frame is constructed for our nodes; each row depicts the feature information. There are seven columns in our data frame. We implement a Graph Convolutional Network (GCN) as a PyTorch module for graph-based data processing tasks. Named GCN, the class encapsulates the network architecture and functionalities. In its constructor, the GCN class expects three parameters: the dimensionality of input features (input_layer_dimension), the dimensionality of the hidden layer (hidden_layer_dimension), and the dimensionality of the output layer (output_layer_dimension). Inside the constructor, the initialize_layers() method sets up the GCN layers using a custom graph convolutional layer (GCNConv).

This method initializes two layers, conv1 and conv2, representing the first and second graph convolutional layers, respectively. The forward() method implements the forward pass of the GCN. It takes input features (x) and graph connectivity information (edge_index) as inputs. During the forward pass, the input passes through the first convolutional layer followed by a rectified linear unit (ReLU) activation function. Dropout regularization is then applied to mitigate overfitting. Finally, the output is generated by passing the processed features through the second convolutional layer, and the result is returned. This GCN architecture is a simple yet effective approach for learning representations from graph-structured data, commonly used in various applications such as node classification and link prediction.

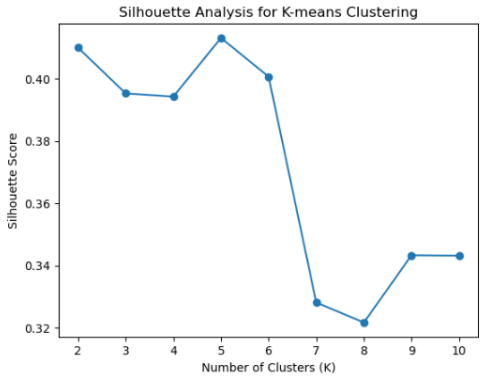

Figure 9. Silhouette analysis graph in case of GCN embedding

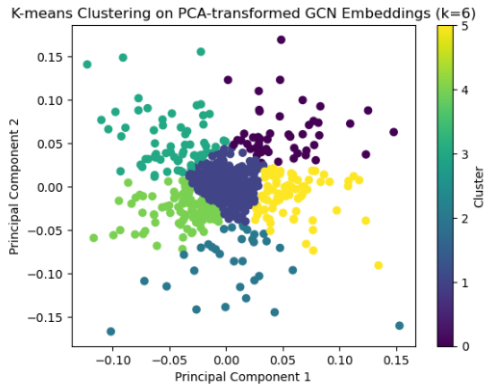

Figure 10. K-Means clustering in case of GCN embedding

Training is done on this convolution network by feeding it the dataframe of node features, Finally, during the evaluation phase, we extract the embeddings for each node. Each node embedding is of 32 dimensions in this case, as the final output layer in our convolution network is of 32 dimensions.

Finally, we perform the same steps as in the above two approaches. We perform PCA to get the final dimensions in 2 dimensions. The curves plotted provided us with the K value of 5 and 6 (in different executions)

In this case, the key observation is that the learnt clusters do not overlap and give distinct boundaries.

6. Results

Following is a summary of the results obtained by us: -

- Successfully implemented Node2Vec, Spectral and GCN embeddings from scratch
- We can see that the clusters overlap in the case of Node2Vec embedding but get clear boundaries in the case of Spectral and GCN embeddings
- We have seen, after multiple executions, that Elbow curves for all three techniques are gradual with no sharp bend
- The Silhouette Analysis gives the highest score to 5,6 clusters in all three techniques
- Our results come out to be identical when we use in-built functions for embedding which depict that our own implementations are working as expected