Node2Vec Clustering for Transaction Network Analysis

Fraud Analytics Assignment

May 2, 2025

Team Members

- **C Paavaneeswar Reddy** – CS22BTECH11014
- **Mohammed Gufran Ali** – CS22BTECH11040
- **Muppalla Bhavesh Chowdary** – CS22BTECH11041

Abstract

This report presents a custom implementation of Node2Vec algorithm for clustering in a transaction network. Following the assignment requirements, we created a graph from payment transactions, converted a multi-directed graph into a simple directed graph, implemented Node2Vec from scratch, and performed clustering analysis on the node embeddings. The implementation demonstrates the application of random walks, skip-gram model, and KMeans clustering to identify patterns in transaction networks.

1 Introduction

Node2Vec is a graph embedding technique that maps nodes to a low-dimensional feature space while preserving the network structure. In this assignment, we implemented Node2Vec from scratch to analyze transaction patterns between users. This approach allows us to identify clusters of users with similar transaction behaviors, which can be valuable for fraud detection, market segmentation, and understanding user transaction patterns.

2 Methodology

2.1 Graph Construction

The first step involved constructing a graph from the transaction data where:

- Nodes represent individual users

- Edges represent payment transactions between users
- Edge weights represent transaction amounts

Since multiple transactions may occur between the same pair of users, the data initially forms a multi-directed graph. Following the requirements, we converted this to a simple directed graph by aggregating multiple edges between the same node pairs, summing their weights.

2.2 Node2Vec Implementation

The core of our solution is a custom implementation of Node2Vec that includes:

2.2.1 Biased Random Walks

We implemented biased random walks controlled by parameters p and q:

- p controls the likelihood of immediately revisiting a node
- q controls the exploration vs. exploitation trade-off

The implementation uses an alias sampling method for efficient sampling from the transition probability distributions.

2.2.2 Skip-Gram Model with Negative Sampling

We implemented a neural network-based skip-gram model using PyTorch that:

- Takes node sequences from random walks as input
- Generates center-context node pairs within a specified window
- Creates negative samples for efficient training
- Learns node embeddings that preserve graph structure

The model contains two embedding layers: one for center nodes and one for context nodes.

2.3 Clustering and Visualization

Once we obtained node embeddings:

- We used the silhouette method to determine the optimal number of clusters
- Applied KMeans clustering to group similar nodes
- Used PCA for dimensionality reduction to visualize the embeddings in 2D and 3D

3 Implementation Details

3.1 Data Loading and Graph Construction

The code first loads transaction data from an Excel or CSV file, creating a multi-directed graph where each transaction is a separate edge. It then aggregates these edges to create a simple directed graph by summing the weights of parallel edges.

3.2 Node2Vec Algorithm

The custom Node2Vec implementation follows these key steps:

3.2.1 Transition Probability Precomputation

Before generating random walks, the algorithm precomputes transition probabilities for each node and edge to enable efficient sampling:

Algorithm 1 Precompute Transition Probabilities

```
1: for each node in graph do
2:     Compute normalized probabilities based on edge weights
3:     Create alias tables for efficient sampling
4: end for
5: for each edge in graph do
6:     Compute transition probabilities based on $p$ and $q$
7:     Create alias tables for these probabilities
8: end for
```

3.2.2 Biased Random Walks

The algorithm generates walks starting from each node, where the next step depends on the previous step according to Node2Vec's bias parameters:

3.3 Skip-Gram Model Training

The code implements a SkipGram neural network model in PyTorch:

- Creates training pairs from walks: (center node, context node, is_positive)
- Uses negative sampling to improve training efficiency
- Trains the model using BCE loss and Adam optimizer
- Extracts node embeddings from the trained model

Algorithm 2 Node2Vec Walk

```
procedure Node2VecWalk(start_node)
    walk = [start_node]
    while length(walk) < walk_length do
        cur = walk[-1]
        cur_neighbors = successors(cur)
        if length(walk) == 1 then
            next_node = random choice from cur_neighbors
        else
            prev = walk[-2]
            Use precomputed probabilities for (prev, cur) edge
            Sample next_node based on these probabilities
        end if
        walk.append(next_node)
    end while
    return walk
end procedure
```

3.4 Clustering Analysis

The embeddings are then analyzed to:

- Find the optimal number of clusters using silhouette scores
- Apply KMeans with the optimal K
- Visualize clusters in 2D and 3D using PCA

4 Results and Visualizations

4.1 Optimal Number of Clusters

The code determines the optimal number of clusters by analyzing silhouette scores for different K values.

4.2 Elbow Method

The elbow method provides another perspective on the optimal number of clusters.

4.3 Embedding Visualizations

The node embeddings are visualized in 2D and 3D spaces using PCA.

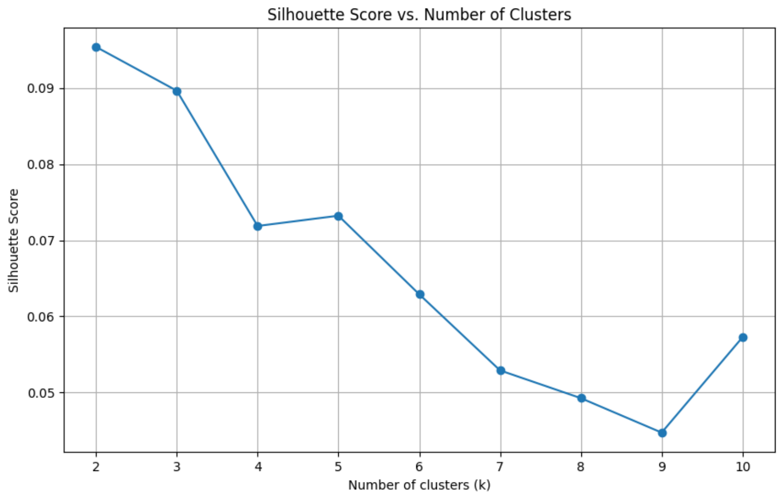

Figure 1: Silhouette Score vs. Number of Clusters

5 Discussion

5.1 Implementation Challenges

Several implementation challenges were addressed:

- Efficient sampling using alias method for large graphs
- Creating a robust skip-gram dataset with negative sampling
- Ensuring numerical stability during probability calculations
- Handling edge cases such as nodes with no outgoing edges

5.2 Parameter Selection

The code allows tuning of several parameters to optimize the Node2Vec algorithm:

- Embedding dimensions: Controls the richness of the feature space
- Walk length and number of walks: Controls the amount of context captured
- p and q parameters: Controls the nature of exploration
- Window size: Controls the size of the context window in skip-gram
- Training parameters: Batch size, epochs, and learning rate

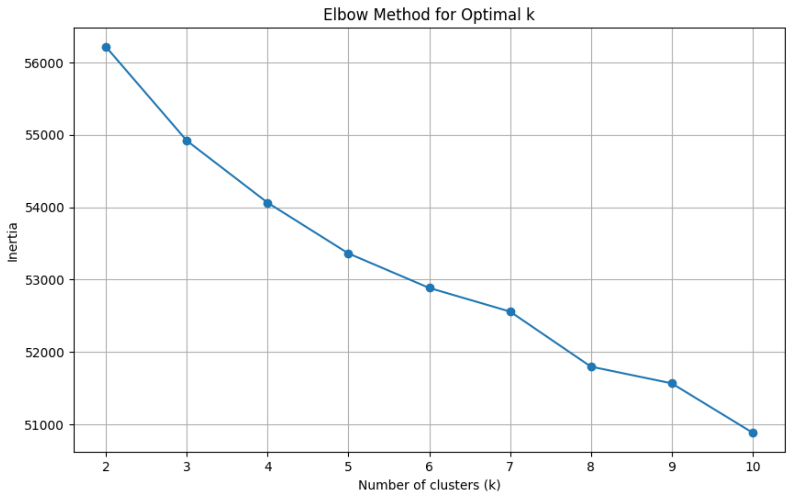

Figure 2: Elbow Method for Optimal K

5.3 Advantages of Custom Implementation

By implementing Node2Vec from scratch rather than using existing packages:

- We gained deeper understanding of the algorithm mechanics
- Had more control over each step of the process
- Could make modifications specific to transaction network analysis
- Integrated seamlessly with custom clustering and visualization

6 Conclusion

This implementation successfully addresses the assignment requirements by:

- Creating a graph from payment transactions
- Converting a multi-directed graph to a simple directed graph
- Implementing Node2Vec from scratch
- Generating meaningful node embeddings
- Identifying clusters of users with similar transaction patterns

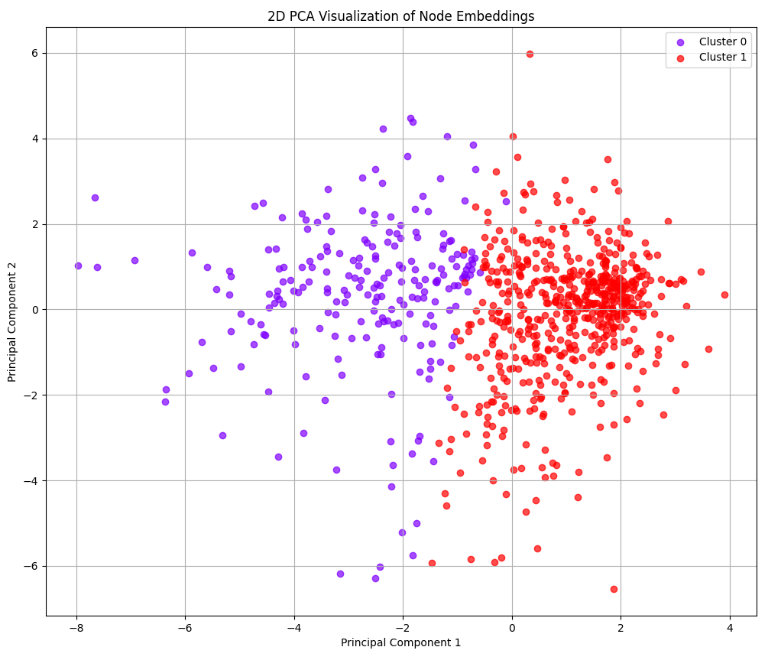

Figure 3: 2D PCA Visualization of Node Embeddings

The approach demonstrates how graph embedding techniques can reveal hidden patterns in transaction networks. The clustering results could be further analyzed to identify specific transaction behaviors within each cluster, potentially revealing useful insights for fraud detection or user segmentation.

Future work could include comparing the custom Node2Vec implementation with other graph embedding techniques, incorporating temporal aspects of transactions, and applying the learned embeddings to downstream tasks such as fraud classification.

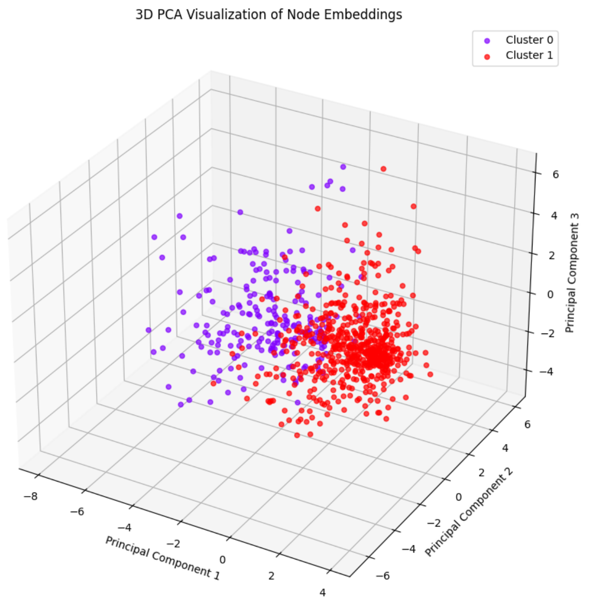

Figure 4: 3D PCA Visualization of Node Embeddings