Indian Institute of Technology Hyderabad

Fraud Analytics (CS6890)

Assignment - 5

Finding Clusters using Node2Vec embedding

Name	Roll Number
Saumay Lunawat	CS24MTECH14005
Shivendra Deshpande	CS24MTECH12017
Chinmay Rajesh Ingle	CS23MTECH12002

Contents

1 Problem Statement 2
2 Dataset Description 2
3 Algorithm 3
4 Approach 3
5 Results & Observations 4

List of Figures

1 Sample rows from the `payments.xlsx` dataset. 2
2 Node2Vec Embeddings Clustered 5

1 Problem Statement

The primary objective of this assignment is to identify meaningful clusters or communities within a network derived from financial transaction data. Given a dataset of transactions, where each transaction involves a sender, a receiver, and an amount, the goal is to represent the entities (senders/receivers) as nodes in a graph and the transactions as weighted directed edges. Subsequently, we aim to leverage the Node2Vec algorithm to learn low-dimensional vector representations (embeddings) for each node that capture the network structure and neighborhood information. Finally, using these embeddings, we apply the K-Means clustering algorithm, with the number of clusters (K) determined by the Elbow method, to group nodes with similar structural roles or connectivity patterns. The expected outcome is a partition of the nodes into distinct clusters, visualized using PCA, which could potentially reveal underlying communities, hubs, or anomalous groups within the transaction network.

2 Dataset Description

The dataset used is contained within an Excel file named `payments.xlsx`. It consists of transactional records.

- **Format:** Tabular data, with each row representing a single transaction.
- **Initial Records:** The file contains 130,535 transaction records initially loaded.
- **Key Columns:** The essential columns used for graph construction are:
  - `Sender`: The identifier of the entity initiating the transaction. Treated as a string.
  - `Receiver`: The identifier of the entity receiving the transaction. Treated as a string.
  - `Amount`: The numerical value associated with the transaction. Treated as a float.
- **Sample Data:** A snapshot of the data structure is shown in Figure Figure 1.

	A	B	C
1	Sender	Receiver	Amount
2	1309	1011	123051
3	1309	1011	118406
4	1309	1011	112456
5	1309	1011	120593
6	1309	1011	166396
7	1309	1011	177817
8	1309	1011	169351
9	1309	1011	181038
10	1309	1011	133216
11	1309	1011	133254

Figure 1: Sample rows from the `payments.xlsx` dataset.

- **Preprocessing and Graph Construction:**
  - Data Loading: The data was loaded using pandas in approximately 2.92 seconds.
  - Cleaning: Sender and Receiver IDs were explicitly converted to strings. The Amount column was converted to a numeric (float) type, handling potential errors.
  - Aggregation: A directed graph (`networkx.DiGraph`) was constructed. Multiple transactions between the same Sender-Receiver pair were aggregated by summing their 'Amount' values. This sum became the weight of the single directed edge between the corresponding nodes.
  - Filtering: Self-loops (transactions where Sender == Receiver) were removed. Nodes that became isolated (having neither incoming nor outgoing edges after aggregation and filtering) were also removed.

  - Graph Processing Time: The graph building and aggregation step took approximately 0.04 seconds.

- **Final Graph Statistics:** After preprocessing and aggregation, the resulting graph used for embedding and clustering consists of **799 nodes** and **5358 weighted directed edges**.

3 Algorithm

The core algorithms employed in this assignment are Node2Vec for graph embedding (specifically configured like DeepWalk in this run), K-Means for clustering, the Elbow method for determining the optimal number of clusters, and PCA for visualization.

1. **Node2Vec:** A semi-supervised algorithm for learning continuous feature representations (embeddings) for nodes in networks.
   - **Biased Random Walks:** Node2Vec generates sequences of nodes by simulating random walks. The simulation uses two parameters, p and q, to bias the walks. In this execution, parameters $p = 1.0$ and $q = 1.0$ were used, which effectively makes the walk strategy equivalent to that of DeepWalk (unbiased walks based on transition probabilities proportional to edge weights). The Alias method was used for efficient weighted sampling during walk generation.
   - **Skip-gram with Negative Sampling:** The generated walks (10,545 valid walks in this case) were used to train a Skip-gram model. The model learns node embeddings by predicting context nodes within a window, optimizing using negative sampling via a PyTorch neural network (input embedding layer, output embedding layer).
2. **K-Means Clustering:** An unsupervised iterative algorithm partitioning the node embeddings into a pre-defined number (K) of clusters by minimizing the within-cluster sum of squares (WCSS or inertia).
3. **Elbow Method:** A heuristic used to estimate a suitable K for K-Means. It involves running K-Means for a range of K values, plotting the inertia, and identifying the "elbow" point where the decrease in inertia significantly slows down.
4. **Principal Component Analysis (PCA):** A dimensionality reduction technique used to project the high-dimensional node embeddings onto a 2D plane for visualization, capturing the directions of maximum variance.

4 Approach

The implemented approach followed a sequential pipeline as detailed in the Python script:

1. **Data Loading and Preparation:** Loaded 130,535 transactions from `payments.xlsx`. Cleaned 'Sender', 'Receiver', and 'Amount' columns. (Time: 2.92s)
2. **Graph Construction and Aggregation:** Built a weighted directed graph using `networkx`, aggregating edge weights and removing self-loops and isolates. Resulted in a graph with 799 nodes and 5358 edges. (Time: 0.04s)
3. **Node2Vec Embedding Generation:**
   - **Simulate Walks:** Generated biased random walks with parameters:
     - Walk Length: `WALK_LENGTH = 40`
     - Walks per Node: `NUM_WALKS = 15`

- Return Parameter: `P = 1.0`
- In-Out Parameter: `Q = 1.0` (DeepWalk-like behavior)

10,545 valid walks were generated. (Walk simulation time: 0.6s)

- **Train Skip-gram Model:** Trained on a CPU using PyTorch with hyperparameters:
  - Embedding Dimension: `EMBEDDING_DIM = 64`
  - Window Size: `WINDOW_SIZE = 5`
  - Negative Samples: `NUM_NEGATIVE = 5`
  - Batch Size: `BATCH_SIZE = 128`
  - Learning Rate: `LEARNING_RATE = 0.01`
  - Epochs: `EPOCHS = 5`

  The final average loss after 5 epochs was approximately 0.5078. Embeddings of shape (799, 64) were generated. (Training time: 34.96s)

4. **Optimal Cluster Number Estimation (Elbow Method):**
   - Tested K values in `range(2, 7)` (i.e., K = 2, 3, 4, 5, 6).
   - Calculated inertia for each K using K-Means (`n_init=10`).
   - Identified the elbow point using a heuristic, determining `optimal_k = 4`.
   - Plotted the Elbow curve. (Analysis time: 0.67s)
5. **Final Clustering:**
   - Performed K-Means with the optimal $K = 4$ on the 64D embeddings (`n_init=10`).
   - Mapped cluster assignments back to original node IDs. (Clustering time: 0.30s)
6. **Visualization:**
   - Applied PCA to reduce embeddings to 2 dimensions. The 2 components explained 11.65% of the variance.
   - Created a scatter plot, coloring nodes by cluster.
   - Saved the plot as `cluster_visualization.png` and displayed it. (Visualization time: 6.62s)
   - Saved the Elbow plot as `elbow_method_plot.png`.

5 Results & Observations

The execution of the pipeline yielded the following results and observations, consistent with the console output:

- **Graph Structure:** The final network analyzed comprised 799 nodes and 5358 weighted directed edges, derived from 130,535 initial transactions.
- **Node Embeddings:** Node2Vec (configured as DeepWalk with P=1, Q=1) successfully generated 64-dimensional embeddings for each of the 799 nodes after 5 epochs of training on the CPU, taking approximately 35 seconds.
- **Optimal Cluster Estimation:** The Elbow method analysis for K from 2 to 6 yielded the following inertia values:
  - K=2: 42272.62
  - K=3: 41325.34
  - K=4: 40559.52
  - K=5: 39996.62

– K=6: 39531.39

The heuristic identified $K = 4$ as the optimal number of clusters, representing the "elbow" point where the reduction in inertia diminishes. This process was efficient, taking about 0.67 seconds. The corresponding Elbow plot (saved as `elbow_method_plot.png`) would visually confirm this bend.

- **Clustering and Visualization:** Final K-Means clustering with $K = 4$ was performed quickly (0.30s). The clusters were visualized using PCA, projecting the 64D embeddings into 2D space. This 2D representation captured **11.65%** of the total variance. The resulting scatter plot (Figure 2) displays the four clusters.

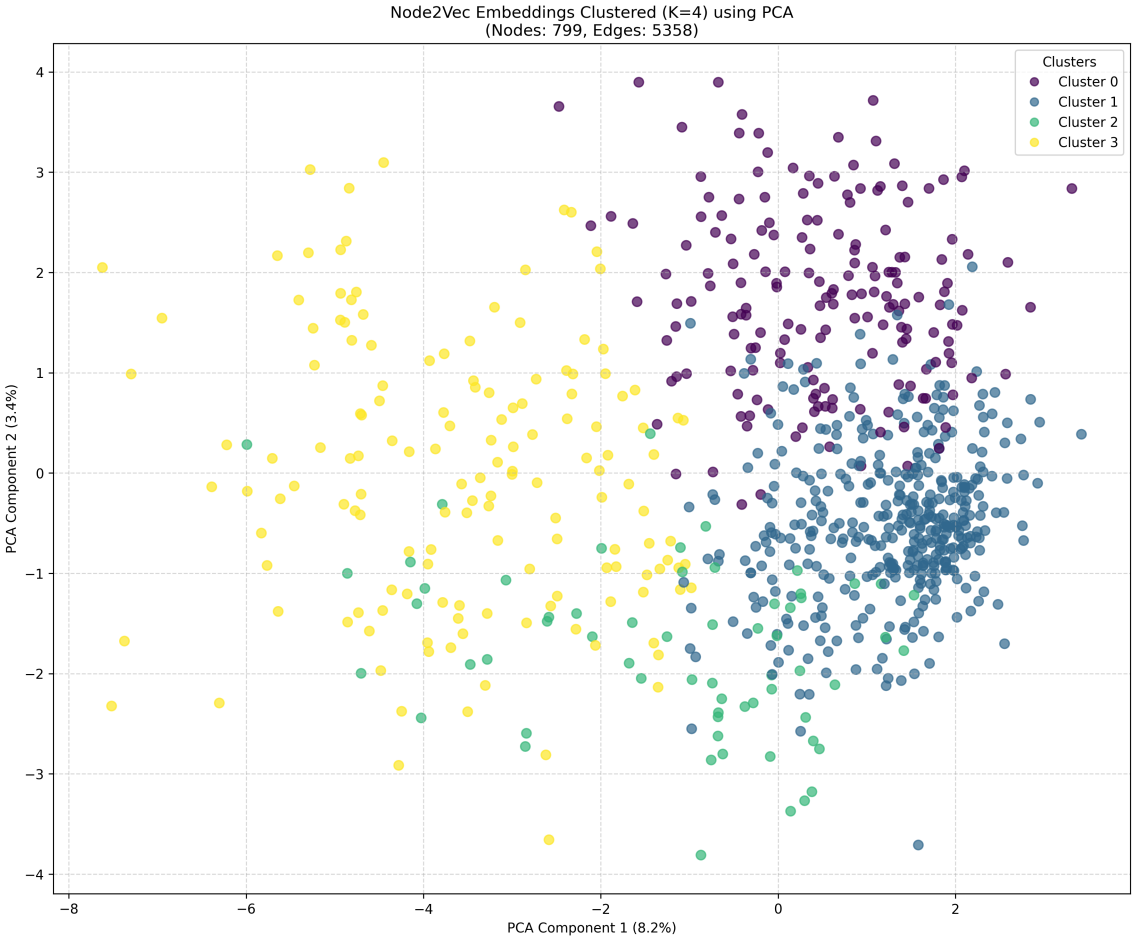

Figure 2: Node2Vec Embeddings Clustered

- **Observations from Visualization:**
  - The PCA visualization shows four distinct groups of nodes, colored according to their cluster assignment (0: purple, 1: blue-green, 2: green, 3: yellow).
  - While there is some overlap in the 2D projection (expected, given the low explained variance of 11.65
  - The relatively low explained variance by the first two principal components suggests that the cluster separation might be more pronounced in the original 64-dimensional space.
  - The successful grouping indicates that the node embeddings captured meaningful structural information from the transaction graph, allowing K-Means to identify communities or groups of nodes with similar connectivity patterns.
  - These clusters could represent different types of actors or communities within the financial network (e.g., high-volume traders, specific merchant groups, potentially fraudulent circles). Further analysis of the nodes within each cluster would be needed to characterize their specific roles.