Report on
Graph Node Clustering Algorithms

Arnab Ghosh, Kritik Agarwal, Raghavendra Kulkarni, Shagun Sharma, and Trishita Saha

Instructor: Dr. Sobhan Babu Chintapalli
Fraud Analytics using Predictive and Social Network Techniques (CS6890)
Dept. of Computer Science And Engineering
Indian Institute of Technology, Hyderabad

ABSTRACT

Clustering is a fundamental problem in Machine Learning that falls under the category of Unsupervised Learning. Clustering takes an unlabeled dataset and tries to group the data points into clusters based on their features. The identification of these features is, in general, not supervised by any labels that determine the output. Although many clustering algorithms like K-Means, DBSCAN, and others work well on a wide range of datasets, some applications need to be modeled as a graph network to capture the valuable features of the data points. Traditional Clustering algorithms fail to capture the node features when applied to a graph network. The graph-based applications need an innovative approach to clustering like Node2Vec Clustering, Spectral Clustering, and Graph Convolutional Network Clustering. In the below-presented work, we use a Payment dataset to model the data points as a graph network and apply three different Graph Clustering algorithms: Node2Vec Clustering, Spectral Clustering, and Graph Convolutional Network Clustering. The results compare the clustering output of the three algorithms and discuss their advantages and disadvantages.

Keywords: Graph Convolution Networks, Node2Vec, Spectral Clustering

PROBLEM STATEMENT

Given a Payment Network as a Directional Multi-graph with senders and receivers as nodes, payments as edges, cluster the nodes of the network into two groups using Node2Vec, Spectral and GCN embedding techniques.

DATASET DESCRIPTION

The dataset consists of one .csv files: Payments.csv

Payments.csv

The Payments.csv contains 130,535 entries with each entry in the form of three comma separated integers. These integers denote the sender ID, receiver ID and the amount transferred in the payment respectively. Table 1 below shows the first 10 entries of the Payments.csv file.

Sender	Receiver	Amount
1309	1011	123051
1309	1011	118406
1309	1011	112456
1309	1011	120593
1309	1011	166396
1309	1011	177817
1309	1011	169351
1309	1011	181038
1309	1011	133216
1309	1011	133254

Table 1. The first 10 entries of Payments.csv

ALGORITHMS USED

Node2Vec Clustering

Algorithm 1 Node2Vec Embedding

Require: A directed multi-graph G denoting the payments
A Positive integer d denoting the dimensions of the node embeddings
Ensure: A mapping M containing each graph node mapped to its embedding

```
number_of_walks ← 10
walk_length ← 80
walks ← generateRandomWalks(G, number_of_walks, walk_length)
for each node in G.nodes():
    embeddings[node] ← uniform(d)
window_size ← 10
epochs ← 10
learning_rate ← 0.25
for epoch in range(epochs):
    for each walk in walks:
        for i, center_node in enumerate(walks):
            for j, context_node in enumerate(walk[max(i - window_size, 0):i + window_size]):
                if context_node ≠ center_node:
                    center_embedding = embeddings[center_node]
                    context_embedding = embeddings[context_node]
                    score = center_embedding · context_embedding
                    predicted = sigmoid(score)
                    error = predicted - 1.0
                    embeddings[center_node] -= learning_rate × error × context_embedding
                    embeddings[context_node] -= learning_rate × error × center_embedding
return embeddings
```

Explanation

- We first generate *num_walks* number of random walks on the graph nodes. For each random walk, we traverse through every node of the graph and append a neighboring node of the latest node in the walk chosen at random to construct the walk of length *walk_length*.
- Then we assign an initial embedding of dimension d to every node. These values are chosen from a uniform distribution.
- We run the Node2Vec embedding learning algorithm for 10 epochs and in each epoch, we calculate a *score* by taking the $sigmoid()$ of the difference between the embeddings of a target node and its context nodes. We then use this score adjust the embeddings of the nodes.
- Finally, we return the mapping of these node embeddings. These node embeddings from the Node2Vec algorithm now become an input to the K-Means clustering algorithm.
- The K-Means clustering algorithm is used to group the node embeddings, each of dimension d, into two clusters. These clusters can then be taken as the clusters of Fraudulent and Legitimate transactions respectively.
- Since the dimension of these embeddings is very high, it becomes hard to visualize these nodes in a pictorial representation. Hence we make use of the Principle Component Analysis (PCA).
- Using PCA, we extract the two most prominent components of the vectors and project them on a 2 dimensional plane for visualization.

Spectral Clustering

Algorithm 2 Spectral Embedding

Require: A directed multi-graph G denoting the payments
Ensure: A mapping M containing each node mapped to its embedding
$A \leftarrow Adjacency\_Matrix(G)$
$D \leftarrow Diagonal\_Matrix(G)$
$L \leftarrow D - A$
$\lambda \leftarrow eigenValues(L)$
$v \leftarrow \textbf{abs}(eigenVectors(L, \lambda))$
Sort the eigenVectors v in increasing order of eigenValues λ
embeddings $\leftarrow$ First 32 dimensions of each eigenVector v
return *embeddings*

Explanation

- We first create an adjacency matrix A and Degree matrix D for the graph G to calculate the normalized Laplacian matrix L as $D - A$
- We then take the eigenvalues and the corresponding eigenvectors of this normalized Laplacian matrix. Since the eigenvectors can be complex, we reduce them to their absolute value.
- We sort the eigenvectors in increasing order of their corresponding eigenvalues and take the top 32 dimensions of these eigenvectors as the node embeddings and return the mapping. These node embeddings from the Node2Vec algorithm now become an input to the K-Means clustering algorithm.
- Now, just like the previous Node2Vec clustering, the K-Means algorithm groups the data points into two clusters and then we use PCA to project the embeddings on a 2-dimensional plane for visualization.

GCN Clustering

Algorithm 3 GCN Embedding

Require: A directed multi-graph G denoting the payments
Ensure: A mapping M containing each node mapped to its embedding
$G \leftarrow$ Merge the multi-edges between two nodes into a single edge
$G \leftarrow$ Relabel nodes by mapping them to node IDs starting from 0
edge_index_tensor $\leftarrow$ Tensor of edge indices
edge_weight_tensor $\leftarrow$ Float Tensor of edge weights
conv $\leftarrow$ Convolutional Network with $(256 - 128 - 32)$ encoding architecture
Train the model *conv* with *adam* optimizer for 2000 epochs
embeddings $\leftarrow$ Encoded 32 dimensional feature embeddings of the nodes
return *embeddings*

Explanation

- We first convert the graph into a simple graph by merging all multi-edges between a pair of nodes into a single edge. Then, we map the node labels to IDs starting from 0.
- Now, we extract the edge indices and weights into separate torch tensors.
- We build a convolutional neural network with an input layer, 2 hidden layers with 256 and 128 units respectively and an output layer of 32 units.
- Finally, we train this model by fitting the torch tensor data and get the 32-dimensional embeddings for each node.
- Now, just like the previous two clusterings, the K-Means algorithm groups the data points into two clusters and then we use PCA to project the embeddings on a 2-dimensional plane for visualization.

IMPLEMENTATION

We make use of various Python packages like ***NetworkX***, ***Scikit-learn***, ***Torch*** and ***Torch_geometric*** for implementing the three algorithms discussed above. This section briefly discusses the usage of these packages.

NetworkX

We use the following functionalities of ***NetworkX***:

- **networkx.MultiDigraph():** We use this function to initialize a Directional Multigraph where we store the senders and receivers as nodes and the payments as edges. The amount of payment done is taken as the edge weight. This function returns an empty Directional Multigraph.
- **networkx.from_pandas_edgelist():** We use this function to construct a graph, taking the edges from a pandas dataframe columns. The pandas dataframe contains three columns: Sender, Receiver and Amount. This function returns the constructed graph *G*.
- **networkx.adjacency_matrix():** We use this function to get the adjacency matrix of the graph. This function returns a sparse square matrix that is convertible to a dense numpy array.
- **networkx.draw_networkx_nodes():** We use this function to represent the graph pictorically excluding the edges. This function returns a scatter plot of the nodes.

Scikit-learn

We use the following functionalities of ***Scikit-learn***:

- **sklearn.cluster.kmeans():** We use this constructor to initialize a KMeans model. The ***fit()*** method of this object runs the standard KMeans algorithm on a list of cluster embeddings. The ***labels_*** attribute of this object returns a list of cluster IDs to which each node belongs.
- **sklearn.decomposition.pca():** We use this constructor to initialize a PCA model. The ***fit_transform()*** method of this object runs the standard PCA algorithm on a list of cluster embeddings and returns a list of 2-dimensional vectors containing the two principle components of each embedding.
- **sklearn.manifold.TSNE():** We use this constructor to initialize a TSNE model. The ***fit_transform()*** method of this object runs the standard TSNE algorithm on a list of cluster embeddings and returns a list of 2-dimensional vectors containing the two principle components of each embedding. We use TSNE to project embeddings in GCN Clustering algorithm.

Torch and Torch_geometric

We use the following functionalities of ***Torch*** and ***Torch_geometric***:

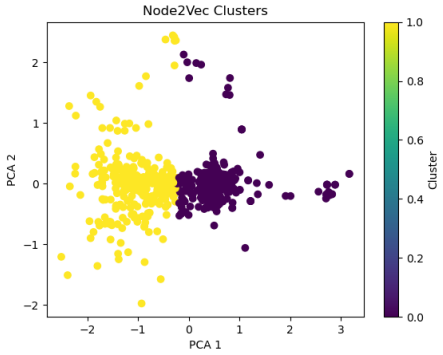
- **torch.eye():** We use this function to one-hot encode the features of a node before feeding them to the Convolutional Network in GCN clustering algorithm. This returns a binary vector of dimension equal to number of features of a node, with a 1 and the rest all 0s.
- **torch.optim.adam():** We use this function to set the Adam optimizer as the optimizing function of the Convolutional network in the GCN clustering algortithm. This function returns an instance of Adam Optimizer.
- **torch.tensor():** We use this function to convert the edge indices and weights of the directed graph to tensors in the GCN clustering algorithm. This function returns a torch tensor.
- **torch_geometric.GCNConv():** We use this function to build the Graph Convolutional Network architecture in the GCN Clustering algorithm. We also use this in training the convolutional network.
- **torch.nn.functional.relu():** We use this function to set the relu function as the activation function for each node of the Convolutional network in the GCN Clustering algorithm. This function returns an instance of relu activation function.

Other Light Weight Packages

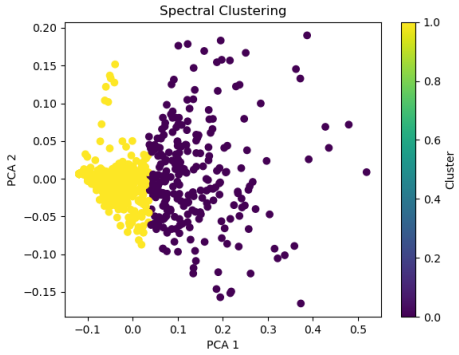
- ***Numpy*** and ***Pandas*** for data storing and manipulation
- ***Matplotlib*** for visulization of the results

RESULTS

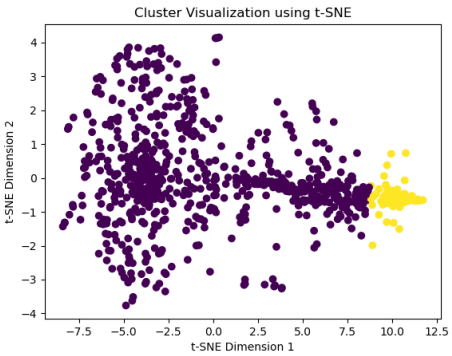
By running the above three algorithms, we are able to cluster the transactions into two groups. The Fig. 1 below shows the Node2Vec clustering, Spectral clustering and GCN clustering results in that order.



(a) Node2Vec Clustering



(b) Spectral Clustering



(c) GCN Clustering

Figure 1. Clustering Results

Inference

- Fig. 1a shows the Node2Vec clustering of the data. The cluster sizes look almost equal in this clustering.
- Fig. 1b shows the Spectral clustering of the data. The cluster sizes here slightly differ from the Node2Vec clustering.
- Fig. 1c shows the GCN clustering of the data. This clustering is showing a varied result.

CONCLUSION

In this work, we used a Payment Dataset containing 130,535 transactions and modeled it into a Directional Multi-graph. On this multi-graph, we applied three different graph clustering algorithms: Node2Vec, Spectral and GCN Clustering. We see that the Node2Vec and Spectral Clustering are producing almost similar outputs. The GCN clustering is producing a different kind of clustering output. This is due to the convolutional nature of the clustering algorithm.

REFERENCES

[1] D. de Roux, B. Perez, A. Moreno, M. d. P. Villamil, and C. Figueroa, "Tax fraud detection for under-reporting declarations using an unsupervised machine learning approach," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, (New York, NY, USA), p. 215–222, Association for Computing Machinery, 2018.