Exploring Real-World Applications of Property Graphs through Data Analysis

Semantic Data Management Project

Abstract - This paper provides a comprehensive overview of graph embeddings, delving deep into their definition, structure, technique, and application with meticulous detail. We explored how to cluster property graphs using the K-means algorithm, showing how graph embedding techniques can be applied in real-world scenarios.
Keywords: *Graph embedding, property graph clustering, Python, Neo4j, Cypher, K-means algorithm.*

Research on Graph Embeddings

Introduction

Definition 1 *A **graph** G is a structure based on a set $V(G)$ of vertices and a set $E(G)$ of edges, which are non-ordered pairs of vertices.*

Definition 2 *Let X, Y be topological spaces and $f : X \to Y$ be a continuous function. We say that f is a **topological embedding** if f yields a homeomorphism between X and $f(X)$ together with the subspace topology inherited from Y. [1]*

Graph embedding converts a graph G, which is the *graph embedding input*, into a low dimensional space in which the graph information is preserved, the output is a set of continuous vectors where each vector represents a certain graph structure (node, edge,...) i.e:

$$\underset{\substack{\text{graph}\\\text{embedding}\\\text{input}}}{G} \xrightarrow{\text{embedding}} \underset{\substack{\text{graph}\\\text{embedding}\\\text{output}}}{\text{embedding}(G) \subset \mathbb{R}^d}$$

We categorize 4 types of **graph embedding input**: *homogeneous graph, heterogeneous graph, graph with auxiliary information* and *graph constructed from non-relational data.*

We categorize 4 types of **graph embedding output**: *node embedding, edge embedding, hybrid embedding and whole-graph embedding.*

1 Problem formalization

Definition 3 *Let $G = (V, E)$ be a graph, $v_i, v_j \in V$ nodes, $e_{ij} \in E$ edge, A_{ij} is the weight of the edge e_{ij}. We denote the **first-order proximity** between v_i and v_j as $s_{ij}^{(1)} := A_{i,j}$.*

$s_1^{(1)}$ records the weights of edges connecting v_1 with all the other nodes in the graph.

Two pair of nodes v_i, v_j are (first-order) more similar than v_i, v_k if $s_{ij}^{(1)} > s_{ik}^{(1)}$.

Definition 4 *The **second-order proximity** $s_{ij}^{(2)}$ between v_i and v_j is computed by*

$$s_{ij}^{(2)} = \cos\left(\frac{s_i^{(1)} \cdot s_j^{(1)}}{\|s_i^{(1)}\|\|s_j^{(1)}\|}\right) \tag{1}$$

Example. Let G be the following graph.

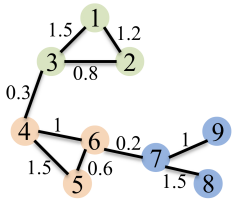


Figure 1: Input graph G

We want to compute the second-order proximity $s_{ij}^{(2)}$ between 1 and 2. First,

$$s_1^{(1)} = [0, 1.2, 1.5, 0, 0, 0, 0, 0, 0]$$
$$s_2^{(1)} = [1.2, 0, 0.8, 0, 0, 0, 0, 0, 0]$$

$$s_{ij}^{(2)} \overset{1}{=} \cos\left(\frac{1.5 \cdot 0.8}{\sqrt{1.2^2 + 1.5^2}\sqrt{1.2^2 + 0.8^2}}\right) = 0.43$$

The **higher-order proximity** can be defined likewise.

Graph embedding problem. Given $G = (V, E)$ graph and d predefined dimensionality of the embedding, the graph embedding aims to convert G into a d-dimensional space in which first and higher-order proximities are preserved.

2 Problem settings of graph embedding

2.1 Graph Embedding Input

The input of graph embedding is a graph. In [2] they divide graph embedding input into four categories:

2.1.1 Homogeneous Graph

Definition 5 *A **homogeneous graph** $G = (V, E)$ is a graph such that $|\mathcal{T}^v| = |\mathcal{T}^e| = 1$. Where $\mathcal{T}^v$ and $\mathcal{T}^e$ are the set of node and edge types.*

Homogeneous graphs can be either *directed* or *undirected*, with *weighted* or *unweighted* edges. Edge weights and directions provide crucial information to embedd the graph. For instance, higher edge weights suggest closer embedding in the graph.

Example. An example of a homogeneous graph would be a simple social network, $\mathcal{T}^e = \{\text{friendship}\}$, $\mathcal{T}^v = \{\text{person}\}$. In contrast, a heterogeneous graph includes multiple types of relations, such as $\mathcal{T}^e = \{\text{friendship, following, likes}\}$, representing different interactions within the same network.

2.1.2 Heterogeneous Graph

Definition 6 *A **heterogeneous graph** $G = (V, E)$ is a graph such that $|\mathcal{T}^v| > 1$ and/ or $|\mathcal{T}^e| > 1$.*

Heterogeneous graph mainly exist in the following three scenarios: **Knowledge Graphs**, Community-based Question Answering (cQA) and Multimedia Networks.

2.1.3 Graph with Auxiliary Information

Graphs that provide extra information play a crucial role in generating embedding. For example:

- **Label**: Nodes in the graph can have labels, nodes with different labels should be embedded far away. For instance, in a simple social network, each person could be labeled with labels={, }.
- **Attribute**: Nodes in the graph can have attributes, which may take discrete or continuous values.
- **Node Features**: Nodes in the graph can have various features, including text, documents, and images, constituting unstructured data.
- **Information propagation** and **Knowledge base**.

2.1.4 Graph Constructed from Non-relational Data

When input graphs are not provided, they can be constructed from non-relational data, from **feature matrix** $X \in \mathbb{R}^{|V| \times N}$ or from **similarity matrix** S.

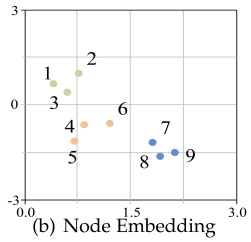
2.2 Graph Embedding Output

The output of graph embedding, embedding(G), is a subset of $\mathbb{R}^d$, representing (part of) the graph. Based on [2] we classify them into the following four categories:

2.2.1 Node Embedding

Node embedding represents each node in the graph as a vector in a continuous space. There are various node embedding functions, depending on how is similarity between nodes calculated, weather we use first-order proximity (see Definition 3), second or higher order proximity (see Definition 4).

Example. Let Figure 8 be the input graph $G = (V, E)$. Let the embedding function $f : V \to \mathbb{R}^2$ map each node in the graph G to a point in the 2D space.



(b) Node Embedding

Figure 2: Node Embedding

2.2.2 Edge Embedding

Edge embedding represents each edge in the graph as a vector in a continuous space.

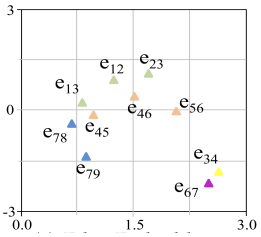


Figure 3: Edge Embedding

2.2.3 Hybrid Embedding

Hybrid embedding occurs when a combination of different graph component types, such as node + edge (substructure) or node + community, is represented as vectors in a continuous space.

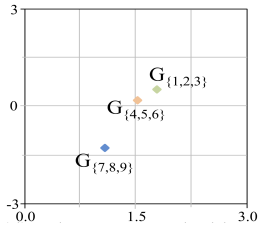
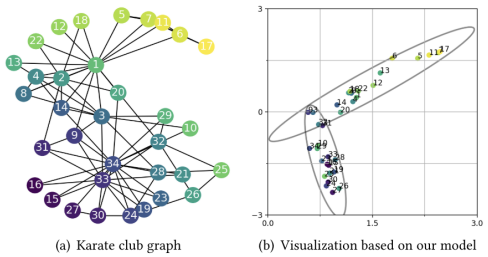


Figure 4: Hybrid Embedding (substructure)

Substructure embedding focuses on capturing the structural relationships between nodes and edges, aiding tasks like semantic proximity search and graph classification. **Community embedding**, main aim is to detect communities, represent communities within graphs, and can also performe node classification.



(a) Karate club graph (b) Visualization based on our model

Figure 5: Hybrid Embedding (community) from [3]

2.2.4 Whole-Graph Embedding

Whole-Graph Embedding represents an entire graph as a single vector in continuous space, so similar graphs are embedded closer in $\mathbb{R}^d$.

Applications: Graph classification tasks: such as classifying compounds where each graph represents a molecule (e.g., a protein, amino acids). We embed each molecule to facilitate compound classification.

3 Graph Embedding Techniques

There are different algorithms available for graph embedding, each designed to achieve specific embedding outputs while preserving certain graph properties.

3.1 Matrix Factorization

MF is mostly used to embed a graph constructed from non-relational data (Sec.2.1.4) for node embedding. There are two primary methods.

3.1.1 Graph Laplacian Eigenmaps

Provide a mathematical solution for computing optimal node embeddings. This method uses an objective function, with various versions discussed in the literature, and employs different algorithms to compute the similarity between nodes. Ultimately, the problem reduces to finding the eigenvectors of a matrix. For a deeper mathematical understanding, refer to A.1.

3.1.2 Node Proximity Matrix Factorization

Given the node proximity matrix W, also referred before as the similarity matrix, we aim to directly factorize this matrix and minimize the loss of approximation. There are various node proximity matrix factorization considered in [2] for example

$$w_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases}$$

3.2 Deep Learning

DL techniques embed either paths samples from a graph or the whole graph. There are two categories based on whether random walk is adopted to sample paths from a graph.

Definition 7 *A walk of length k in a graph G is a sequence of vertices $(v_1, \ldots, v_k)$ where $v_i v_{i+1} \in E(G)$ for $i = 1, \ldots, k-1$.*

3.2.1 DL based Graph Embedding with Random Walk

The second-order proximity in a graph can be preserved in the embedded space by maximizing the probability of observing the neighbourhood of a node conditioned on its embedding. A graph is represented as a set of random walk paths sampled from it. The deep learning methods (DeepWalk,SikpGram) are then applied to the sampled paths for graph embedding which preserves graph properties carried by the paths

3.2.2 DL based Graph Embedding without Random Walk

3.3 Edge Reconstruction

There are various optimization techniques and algorithms designed to maximize the similarity between the embeddings and the input graph. We shall consider:

- *Maximizing Edge Reconstruction Probability*: embedding optimization by maximizing the probability of generating the input edges from the input graph.
- *Minimizing Distance-based Loss*: The node proximity calculated based on node embedding should be as close to the node proximity calculated based on the observed edges as possible.
- *Minimizing Margin-based Ranking Loss*: embedding optimization so that a node's embedding is more similar to the embedding of relevant nodes than that of any other irrelevant node.

3.4 Graph Kernel

A graph kernel is designed for whole-graph embedding only as it captures the global properties of a whole graph. The whole graph structure can be represented as a vector containing the counts of elementary substructures that are decomposed from it.

By using different types of substructures such as graphlets, subtree patterns, and random walks—graph kernels effectively compare and analyze the structural similarities between graphs, enabling robust graph classification and similarity assessment.

3.5 Generative Model

A generative model can be defined by specifying the joint distribution

$$P(\text{input features}, \text{class labels} \mid \text{parameters})$$

Generative models can be applied to graph embedding in two main ways:

- Embed Graph Into The Latent Semantic Space
- Incorporate Latent Semantics for Graph Embedding

4 Applications

Graph embedding main strength is that can be processed efficiently in time and space.

4.1 Node related applications

4.1.1 Node classification

Node classification is useful when dealing with labeled nodes within the graph, subsequently transformed into vectors through embedding. Therefore similar nodes have the same labels, therefore the vectors are closer in the embedding space. We train our classification model using these labeled node embeddings and apply a classifier such as SVM, logistic regression, or KNN. Therefore with an unlabeled node, we can predict its label.

4.1.2 Node clustering

Is an unsupervised learning algorithm which aims to group similar nodes into clusters. Is useful when we do not have labeled nodes.

4.1.3 Node Recommendation/Retrieval/Ranking

Let $G = (V, E)$ be a graph, $n \in V$ node. Node recommendation is a task such that assigns to n a set of the top K nodes of interest based on certain criteria (such as similarity).

4.2 Edge Related Applications

4.2.1 Link Prediction

Embedding the edges of a graph can benefit from predicting missing links of an incomplete graph. We can do it with homogeneous or heterogeneous graphs. A real use case may be real friendship between two users that is missing in a social network.

4.2.2 Triple Classification

Embedding the edges can benefit in knowledge graphs for indicating the relation between the head and tail entities. Therefore we can make a triplet classification whether an unseen triplet is correct.

4.3 Graph Related Applications

As introduced in Section 2.2.4, the embedding of the entire graph may be useful for classifying graphs. For instance, in biology, it can aid in classifying molecules, which are typically represented as graphs, and also in identifying similarities between different graphs, and evaluate similarities between compounds through substructure similarities.

5 Future directions

- Traditional deep learning models optimized struggle with the irregular structure of graphs, leading to inefficiencies.
- One main problem of graph embedding is that it primary focus on static graph embedding, limiting the main strong advantage of graphs, which is that can have a dynamic structure and may evolve over time through the addition and deletion of nodes/edges.
- Some embedding techniques are limited and do not contemplate all the information that is stored in the graph. Making more efficient computationally but less rich in information.
- Graph embedding facilitates the conversion of data instances from various sources into a unified space, enabling direct comparisons.

1 Real use cases of property graph

We'll start by revisiting the **property graph** from Lab 1. To explore a real use case using this graph, we shall think about the **application** of the graph, what analyses we want to extract, and how the embedding of the graph would be useful. Also we need to explore the **technique**, i.e which algorithm use to embed the graph, and what would be our **input graph embedding** and our **output graph embedding**.

Our main idea is make a paper clustering, therefore perform a node clustering. Therefore we need to make node embedding. We used Neo4j to upload the graph and the Data Science Library to embedd the graph. Then we also applied K-means algorithm using N4J and add extra analysis with Python.

1.1 Create the embedding

In order to generate a graph embedding, (in our example) we require a (or multiple) subgraph. Consequently, our initial step involves the creation of said subgraph. Subsequently, we shall proceed to project a part of the property graph from Lab 1.

```
CALL gds.graph.project(
  'fullGraph',
  {
    Author: {
      label: 'Author'
    },
    Papers: {
      label: 'Papers'
    },
    Edition: {
      label: 'Edition'
    },
    Volume: {
      label: 'Volume'
    },
    Conference: {
      label: 'Conference'
    },
    Journal: {
      label: 'Journal'
    },
    Keywords: {
      label: 'Keywords'
    }
  },
  {
    writes: {
      type: 'writes',
      orientation: 'UNDIRECTED'
    },
    cites: {
      type: 'cites',
      orientation: 'UNDIRECTED'
    },
    reviews: {
      type: 'reviews',
      orientation: 'UNDIRECTED'
    },
    published_in: {
```

```
      type: 'published_in',
      orientation: 'UNDIRECTED'
    },
    relates_to: {
      type: 'relates_to',
      orientation: 'UNDIRECTED'
    },
    contained_in: {
      type: 'contained_in',
      orientation: 'UNDIRECTED'
    },
    belongs_to_conference: {
      type: 'belongs_to',
      orientation: 'UNDIRECTED'
    },
    belongs_to_journal: {
      type: 'belongs_to',
      orientation: 'UNDIRECTED'
    }
  }
)
```

To embed the created subgraph, our initial step is to determine the embedding dimension. We will perform 2 different embeddings, one into $\mathbb{R}^2$ and another into $\mathbb{R}^{32}$. The embedding into 2D is only because we wanted to understand graph embedding and visualize it with a plot in the 2D plane, but for the analysis we will embed into 32. The following Cypher query is to embed the subgraph and save the embedding vector as a property in the node, as *embedding2d*.

```
CALL gds.fastRP.stream(
  'fullGraph',
  {
    embeddingDimension: 2,
    randomSeed: 42
  }
)
YIELD nodeId, embedding
WITH gds.util.asNode(nodeId) AS node, embedding
SET node.embedding2d = embedding
```

Then due to Neo4J constrains we needed to project a new graph only with the embedding (otherwise gave an error).

```
CALL gds.graph.project(
  'graph_2d',
  {
    Papers: {
      label: 'Papers',
      properties: ['embedding2d']
    }
  },
  {
    cites: {
      type: 'cites',
      orientation: 'UNDIRECTED'
    }
  }
);
```

1.2 Clustering: Application of node embedding

We chose clustering as an application of node embeddings because embeddings a graph reminds us the idea of transforming nodes into points on an $\mathbb{R}^2$ plane, where closer points would represent more similar nodes. This concept naturally led us to think of classifying and grouping these points as a practical application of embeddings.

We applied the KMEANS algorithm with $k = 3$.

```
CALL gds.kmeans.stream(
  'graph_2d',
  {
    nodeProperty: 'embedding2d',
    k: 3,
    randomSeed: 42
  }
)
YIELD nodeId, communityId


WITH nodeId, communityId
MATCH (node)
WHERE id(node) = nodeId


SET node.cluster2d = communityId

RETURN gds.util.asNode(nodeId).name AS name, communityId
ORDER BY communityId DESC;
```

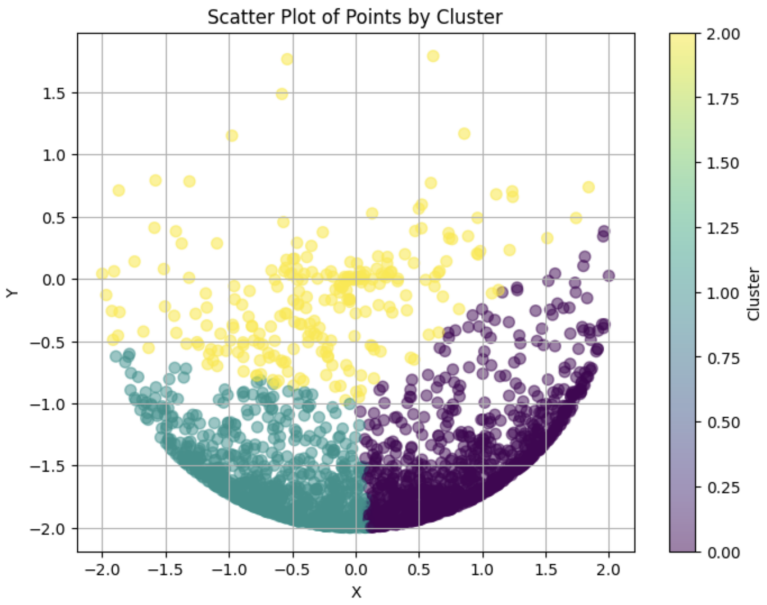


Figure 6: Caption

To embed the nodes instead of into $\mathbb{R}^2$ in $\mathbb{R}^{32}$ we shall see annex A.2 where we performed similar kind of queries.

To visualize the graph in Neo4j with the assigned cluster labels (in different colors), we needed to add a specific label to each Paper node. We used the embedding in 32d to

```
MATCH (p:Papers) WHERE p.cluster32d IN [0] SET p:Cluster0 RETURN p
```

```
MATCH (p:Papers) WHERE p.cluster32d IN [1] SET p:Cluster1 RETURN p
MATCH (p:Papers) WHERE p.cluster32d IN [2] SET p:Cluster2 RETURN p
```

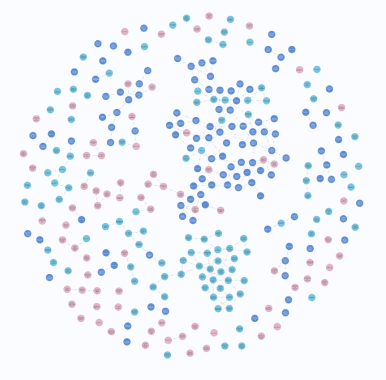


Figure 7: Part of the graph clustering in NEO4J

Now, let's delve into each cluster within the provided Python Notebook named `embedding_neo4j.ipynb`. Within this notebook, we analyze each cluster, examining various statistics and providing comprehensive profiling for each cluster. Some of the queries were:

```
MATCH (p:Papers)
RETURN p.cluster32d AS cluster, COUNT(p) AS num_paper
```

```
MATCH (p:Papers)-[:contained_in]->(:Volume|Edition)-[:belongs_to|belongs_to]->(j:
    Journal|Conference)
RETURN p.cluster32d AS cluster, j.name AS Jorunal_Conference, COUNT(p) AS paperCount
ORDER BY paperCount DESC
```

```
MATCH (p:Papers)-[:relates_to]->(k:Keywords)
RETURN p.cluster32d AS cluster, k.name AS keyword, COUNT(*) AS num_keyword
ORDER BY num_keyword DESC
```

We created a visualization based on the query results as we found the distribution of keywords within each cluster intriguing.

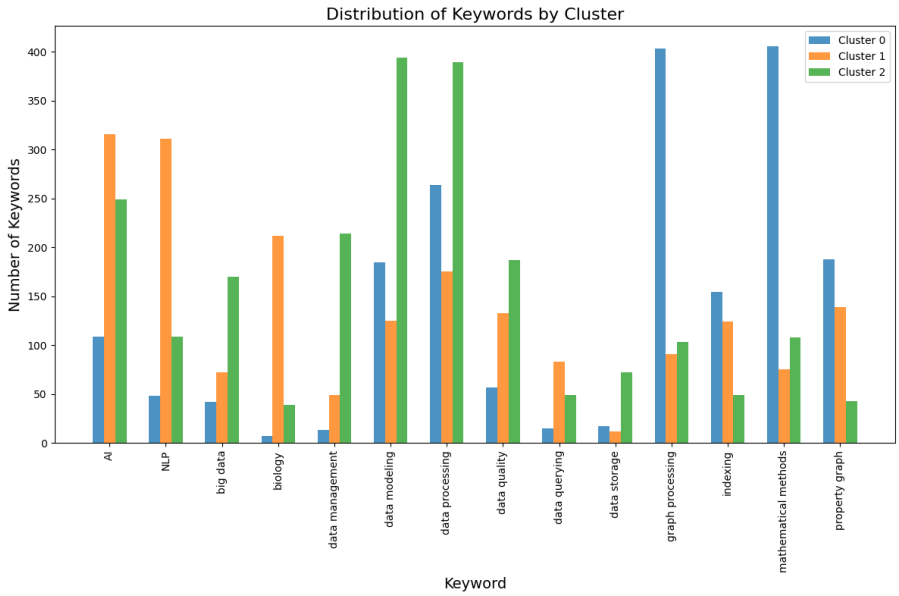


Figure 8: Keyword distribution betwee clusters

Here, we observe a notable prevalence of certain keywords within one cluster compared to another. Thus, we can infer that paper nodes sharing these keywords are likely embedded closer together in the space.

1.3 Profiling

- Cluster 0 primarily comprises papers related to graph processing and mathematical methods. Notably, key conferences and journals within this cluster include "Universität Trier, Mathematik/Informatik" and "Forschungsberichte, TU Munich". Additionally, numerous authors have contributed extensively to this cluster, with a substantial number of published papers. Which made us think that the papers are reaserch university papers related to mathematics, informatics, and graph processing. Additionally, the high publication activity from authors affiliated with these institutions may indicate active research programs or collaborations within academia.

- Cluster 1 predominantly focuses on AI, NLP and biology topics including ethical considerations in science and engineering. Key conferences and journals within this cluster include "Sci. Eng. Ethics" and "IWBS Report." Similar to Cluster 0, numerous authors have contributed extensively to this cluster, as evidenced by their substantial publication records. This observation suggests that the papers within this cluster may primarily revolve around ethical aspects of scientific and engineering practices. Additionally, the high publication activity from authors affiliated with these institutions may indicate active research programs or collaborations within the field of ethics in science and engineering.

- Cluster 2 is centered around various data-related topics such as data modeling, data processing, data management, and data quality. What distinguishes this cluster is the diversity of authors contributing to it. Unlike some clusters dominated by specific authors or institutions, the papers within this cluster come from a wide range of contributors, reflecting a broad and inclusive research community. This diversity suggests a rich landscape of perspectives and expertise contributing to discussions on data-related issues, making this cluster particularly dynamic and interdisciplinary.

1.4 Conclusions

Alongside working on this project, we found the following results and reflections about graph embedding. The main benefit of graph embedding is that it is easier to apply machine learning algorithms and more efficient because its ability to transform complex graph structures into a vector space. However, a significant downside that we found is the loss of explainability and interpretability. When graph data is transformed into numerical vectors, the intuitive understanding of the original graph structure is lost.

A Annex

A.1 Understanding Graph Laplacian Eigenmaps

Recall that we want compute the best embedding output to maximize the similarity between pairs of nodes.

The objective function is defined as follows:

$$y^* = \arg\min \Sigma_{i \neq j}(y_i - y_j^2 w_{ij}) = \arg\min y^T L y$$

where w_{ij} is the similarity between the node v_i and v_j. $L = D - W$ where D is the diagonal matrix such that $d_{ii} = \Sigma_{j \neq i} w_{ij}$. There are different ways to compute the similarity w_{ij} between the node v_i and v_j:

- Let X_i, X_i feature vectors, w_{ij} is the Euclidian distance between the feature vectors, $w_{ij} = d(X_i, X_j)$.
- KNN

The The optimal y depend on the algorithm choice: choice of objective function and way of computing the similarity between nodes. There are several algorithms depict in the paper [2]

A.2 Embedding into $\mathbb{R}^{32}$

```
CALL gds.fastRP.stream(
  'fullGraph',
  {
    embeddingDimension: 32,
    randomSeed: 42
  }
)
YIELD nodeId, embedding
WITH gds.util.asNode(nodeId) AS node, embedding
SET node.embedding32d = embedding
```

```
CALL gds.graph.project(
  'graph_32d',
  {
    Papers: {
      label: 'Papers',
      properties: ['embedding32d']
    }
  },
  {
    cites: {
      type: 'cites',
      orientation: 'UNDIRECTED'
    }
  }
);
```

```
CALL gds.kmeans.stream(
  'graph_32d',
  {
    nodeProperty: 'embedding32d',
    k: 3,
    randomSeed: 42
  }
```

```
)
YIELD nodeId, communityId


WITH nodeId, communityId
MATCH (node)
WHERE id(node) = nodeId


SET node.cluster32d = communityId

RETURN gds.util.asNode(nodeId).name AS name, communityId
ORDER BY communityId DESC;
```

A.3 Notebook

cluster	num_paper
2	725
1	639
0	636

Table 1: Cluster Paper Counts

References

[1] Alicia Chimeno. Topology notes. Mathematics degree course notes, 2021.

[2] Vincent W. Zheng Hongyun Cai and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, XX(XX):1616–1637, 2018.

[3] S. Cavallari, V. W. Zheng, H. Cai, K. C. Chang, and E. Cambria. Learning community embedding with detection and node embedding on graphs. In *Proc. 26th Int. Conf. World Wide Web Companion*, pages 431–439, 2017.