Real use cases of property/knowledge graphs and their exploitation via data analysis

Camila Pérez Millar, Meritxell Arbiol

June 2022

Abstract

In this paper we focused on getting a deeper understanding of Graph Embeddings. We begin explaining embeddings, understanding how graph embeddings work, distinguishing the different types and techniques used to perform them, reviewing advantages and limitations, to continue with the main applications of graph embeddings in today's world. To conclude this project we'll put into practice what we have learned, implementing an embedding strategy, and running a matching algorithm on a real graph.

1 Embeddings

Embeddings is a low-dimensional space into which we can translate high dimensional vectors. Embeddings simplify the work when working in the field of machine learning with high-dimensional data inputs, such as sparse vectors representing words. In other words, there is data that at first glance do not seem to have any similarity and therefore do not present any relationship, but it depends on what measures can be represented closely. Using Embeddings is useful to create more efficient machine learning models. To understand it better, we can think of embedding as transforming some data (e.g. words) into vectors (creating mappings between these words) so that vectors are generated that can be used to model and analyze.[7]

2 Graph embedding

Once the embedding concept is understood, we can introduce another new concept: Graph Embedding. Graph embedding converts a graph into a low dimensional space in which the graph information is preserved. There are several types of graph embedding, so the input varies, and the same happens with the generated output, and depending on the type of graph we are interested in generating, we will obtain the output with the part of the graph we are interested in (edges, nodes, ...) or even the entire graph. Graph embedding can be used

for several purposes, such as extracting useful information, as well as to obtain data representations that facilitate the extraction of information to later generate models. Depending on the problem, we are going to have one type of embedding input type or another: [3]

1. Homogeneous Graph: All nodes and/or edges are of the same type. The most basic graph embedding is an undirected and unweighted homogeneous graph input setting. Adding weights and directions to the graph gives more information and accuracy. Some graphs have only one of the two characteristics: direction or weight. Depending on what we are interested in representing, it will make sense to consider a graph with several weights and/or with directions.

2. Heterogeneous Graph: There are multiple types of nodes and/or edges. This type of input is necessary when the data comes from sites that it is impossible that all I have the same type. An example could be social networks, where we have text and images, and also have to create different types of links between these data. In the Knowledge Graphs the same thing happens, we have nodes of different types, such as *Authors*, *Papers*, *Conferences* and different types of relationships such as *written by* or *published in*.

3. Graph with Auxiliary Information: In this input graph we have auxiliary information like the different labels that the nodes can have, also the attributes (it can be continuous or discrete), node features, information propagation and Knowledge Base.

4. Graph Constructed from Non-relational Data: As the name itself indicates, this network has been constructed from non-relational data.

The output embedding graph is based on the input one. It can be of four different types: Node Embedding, Edge Embedding, Hybrid Embedding and Whole Graph Embedding.[8]

- Node Embedding: This is the most common output graph embedding. Each node is represented as a vector. If the vector representation is similar it means that they are close in the graph. Each graph defines the proximity between nodes in different ways.

- Edge Embedding: Each edge is represented as a low dimensional vector. If we want to focus more on the relationships that exist within the graph, such as predicting connections, then this is the type of output we need.

- Edge Hybrid: This output type is a combination of different types of graph components, as it could be edge and node.

- Whole-Graph Embedding: Embedding of the whole graph as the name itself indicates. It is used for small graphs.

3 Graph embedding techniques

Graph embedding methods can be classified into three general categories; matrix factorization-based methods, random walk and deep learning[3].

1. Matrix factorisation-based methods construct a matrix taking into account the properties of the graph and factorize it to obtain node embeddings. This can be treated as a dimensionality reduction of the graphs while preserving the structure of the original data.

   Two examples of this methodology are:

   (a) Graph Laplacian Eigenmaps: The idea is to represent the network as a Laplacian matrix, which allows us to assign a value to each node; this value changes little for nodes that are closely related to each other, but among less related nodes the value varies more. From the following objective function the optimal embedding can be derived:

   $$y^* = argmin \sum_{i \neq j} \left(Y_i - Y_j^2 W_{ij} \right) = argmin Y^T L_y \tag{1}$$

   This function is used to find the embedding that minimizes the error with respect to the graph's Laplacian matrix. Finally, the optimal embedding y's are the eigenvectors of the Laplacian matrix, which can be calculated as the maximum eigenvalue Wy= αDy.

   (b) Node Proximity Matrix Factorization: Using direct matrix factorisation, this method allows to approximate the closeness of the nodes in a low dimensional space.[1] This is to minimize proximity loss, by minimizing the following objective function:

   $$min \left| W - YYcT \right| \tag{2}$$

2. Random walk methods: Sample a graph with a large number of paths by starting the walks from random initial nodes, allowing to traverse the graph and capture global and local structural information. To learn the representation of the nodes, probabilistic models such as skip-gram and bag-of-words can be used on the randomly sampled paths.

   - DeepWalk: The purpose of the most popular random walk-based graph embedding methods is to obtain the probability of observing node Vi from the nodes already visited in the random walk.

   $$Pr\left(v_i | \left(\phi\left(v_1\right), \phi\left(v_2\right), ..., \phi\left(v_{i-1}\right)\right)\right) \tag{3}$$

   ϕ is a mapping function that shows the latent representation associated to each node v in the network.[1]

3. Deep learning architectures: are mainly based on neural networks, and they learn a function that maps a graph in numerical form to a low-dimensional embedding by optimizing over a wide group of expressive neural network functions. The convolutional network of graphs allows end-to-end learning of the graph with random size and shape. This model uses the convolution operator on the graph and iteratively adds neighbour embedding for nodes. The propagation rule propagation rule used is:

$$f\left(H^l, A\right) = \sigma\left(D^{-(1/2)}\hat{A}D^{-(1/2)}H^{(l)}W^{(l)}\right) \tag{4}$$

Where A is the adjacency matrix, I is the identity matrix. D is the diagonal node degree matrix of $\hat{A}$.

Deep learning models allow embeddings to be extracted more efficiently, as they bring forward new ways of approximating convolutions and kernels of classical graphs.

4 Some advantages of graph embedding

Creating a good graph embedding gives us a good representation of the nodes in the form of vectors.

We can reduce the dimensions of the data we are going to work with, so we are simplifying the complexity of the problem, preserving the structure of the graph and the links between nodes. We can find the optimal dimension for the representation of the graph according to the application.

It provides scalability, it is a important thing since the networks are really complex and embedding them provides a scalable property using which we can process large graphs.

5 Applications

The number of applications for graph embedding has grown and continues to grow, among the wide range of applications that this powerful modeling tool offers are the following[2]:

- Social Networks: Considering that the information available has increased enormously from online social networks, and that information is continuously created and updated, there are quite complicated social network mining tasks in which graph embedding techniques can provide a graph representation learning tool to solve these challenging problems. Large scale attributed social networks dataset from different sources such as Facebook, Twitter, Reddit or Github can be analyzed.

- Brain Networks: Graph embedding makes it possible to obtain a meaningful low-dimensional representation from a complete graphical instance of a brain network; this can be used to analyze disease mechanisms and describe therapeutic interventions. It also makes it possible to explain information from multiple neuroimaging views at the same time.

- Citation Networks: Given the large number of publications, having an efficient way to represent the complex relationships between citations, publications, and their derivatives is of great relevance. Embedding graphs provides an efficient representation of these relationships, allowing quantitative analysis, identifying groups and the possibility of recommendations.

6 Graph Embeddings in a real Graph

6.1 Description of the Data

We use Cora, a well-known dataset for working with graphs. Each node contains a scientific publication related to data science, and each edge shows how one article cites another. We found 2708 nodes and 5429 edges. The dataset is divided into 7 sub-branch classes of machine learning with which each publication was labelled; Rule Learning, Reinforcement Learning, Theory, Neural Networks, Case-Based, Genetic Algorithms and Probabilistic Methods.

Each scientific publication has a short description with an average length of 130 words. These are described by means of a binary word vector (0 and 1) indicating whether it contains a dictionary word; consisting of 1433 words.[4]

6.2 Embedding proposal and implementation

From the vectors representations (word embeddings) already incorporated in our dataset for each scientific publication we compared the performance of models applying graph embedding against regular natural language processing models. We used multiclass classification models performed by two classifiers:

- Random Forest : Using the default parameters of Random Forest Classifier we fit the model on the training data, and the classifier will build decision trees.

- Naive Bayes : Widely used in natural language processing, calculate the probability of each label from a given text (or vector) and choose the one with the highest probability.

We then performed network embeddings using Fast Random Projection (FastRP), which was created by H. Chen et. al, and it is a node embedding algorithm that allows a large dimensionality reduction and preserves most of the distance information; using the in-memory graph created. [5]

It generates low-dimensional embeddings by randomly placing random adjacency matrix projections of the graph to a low-dimensional matrix, thus reducing

the computational power required for data processing. This algorithm is both simple and fast.

[6]

6.3 Results obtained

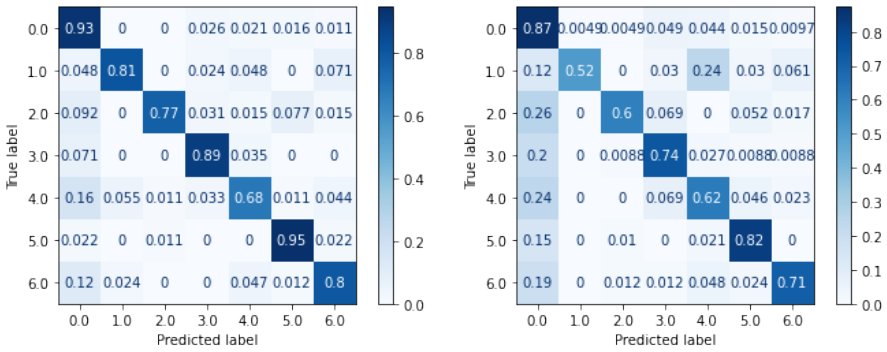
The results obtained with the **Random Forest** and **Naive Bayes** classifiers using Word and Graph embedding are:

[htp]

	Graph Embedding	**Word Embedding**
Accuracy RF	0.853	0.765
Accuracy NB	0.741	0.501

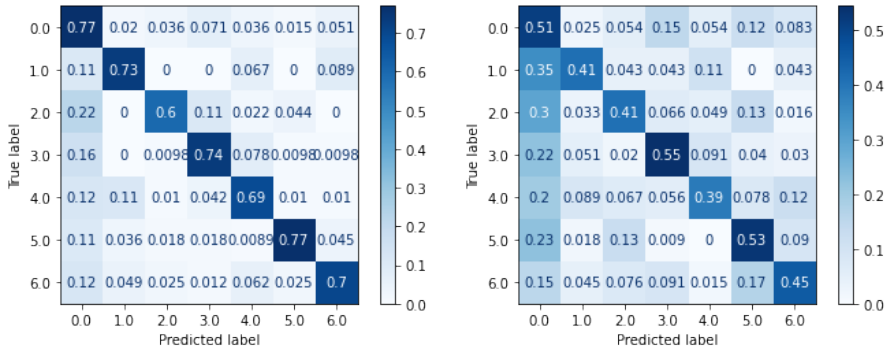
Figure 1 shows the results of the confusion matrix obtained with the Random Forest classifier. In image 1a we have the result with graph embedding and in 1b with word embedding.

Figure 2 shows the results of the confusion matrix obtained with the Naive Bayes classifier. In image 2a we have the result with graph embedding and in 2b with word embedding.



(a) Graph Embedding confusion matrix (b) Word Embedding confusion matrix

Figure 1: Confusion Matrix using Random Forest algorithm



(a) Graph Embedding confusion matrix (b) Word Embedding confusion matrix

Figure 2: Confusion Matrix using Naive Bayes algorithm

As we can see in the previous results we obtain a higher accuracy using Graph Embedding for both classifiers, so we get better results.

7 Summary and Conclusions

Throughout this paper, we have reviewed and explained Graph Embeddings, reviewing how it works, identifying some of its variants, as well as indicating its advantages and disadvantages, and finally implementing it using Neo4j and evaluating its performance with a real graph.

References

[1] Fenxiao C., Yun-cheng W, Kuo J., "Graph representation learning: a survey", 2019.

[2] MENGJIA X., "Understanding Graph Embedding Methods and Their Applications", 2020.

[3] Hongyun C., Vincent W., Kevin Chen-Chuan C., "A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications", 2018.

[4] Makarov I., Makarov M., Kiselev D., "Fusion of text and graph information for machine learning problems on networks", 2021.

[5] "Fast Random Projection", https://neo4j.com/docs/graph-data-science/current/machine-learning/node-embeddings/fastrp, 2021.

[6] "Fast Random Projection (FastRP)", https://docs-legacy.tigergraph.com/v/3.2/graph-algorithm-library/node-embeddings/fast-random-projection-fastrp, 2022.

[7] Yugesh V., "All you need to know about Graph Embeddings", https://analyticsindiamag.com/allyouneedtoknowaboutgraphembeddings, 2022.

[8] McAuliffe E., "All you need to know about Graph Embeddings", https://www.tigergraph.com/blog/understandinggraphembeddings/, 2021.