UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Project: Graph Embeddings

Semantic Data Management

Fathollahi, Mohana
Iborra , Paula

June 23, 2022

1 Introduction

Graphs are a universal language for describing and modeling complex systems. Analysing these graphs provides insights to understand the information hidden in graphs. Therefore, graph analytics has become an impactful research area in recent years. Developing effective and efficient graph analytics can greatly help to better understand complex networks that benefits a lot of applications.

Traditional graph analytics when applied to large-scale network analysis may suffer from high computational cost and excessive memory requirements due to the challenging and inevitable high-dimensionality and heterogeneous characteristics of the networks.

Recently, graph embedding techniques have shown remarkable capacity of converting high-dimensional sparse graphs into low-dimensional, dense and continuous vector spaces, where graph structure properties are maximally preserved (Figure **??**. By representing a graph as a (or a set of) low dimensional vector(s), graph algorithms can be computed more efficiently.

2 Embedding graphs definitions

A graph $G = (V, E)$ is defined as a mathematical data structure that contains a node (or vertex) set $V = v1, v2, v3, ..., v_n$ and an edge set E. G is associated with a node type mapping function $f_v : V \rightarrow T^v$ and an edge type mapping function $f_e : V \rightarrow T^e$. T^v and T^e denote the set of node types and edge types, respectively.

From different perspectives (i.e., edge direction, diversity of nodes and edges, edge cost, etc), graphs can be classified into different categories: directed/undirected graphs, homogeneous/heterogeneous graphs or weighted/binary graphs.

Some formal definitions of graph types (see also Figure 2):

- Homogeneous graph: is a graph where all nodes in G belong to a single type and all edges belong to one single type. Formally, $G_{homo} = (V, E)$ in which $|T^v| = |T^e| = 1$.
- Heterogeneous graph: $G_{hete} = (V, E)$ in which $|T^v| > 1$ and /or $|T^e| > 1$.
- Knowledge graph: $G_{know} = (V, E)$ is a directed graph whose nodes are entities and edges are *subject-property-object* triple facts. Each edge of the form *(head entity, relation, tail entity)* (denoted as $< h, r, t >$) indicates a relationship of r from entity h to entity t.

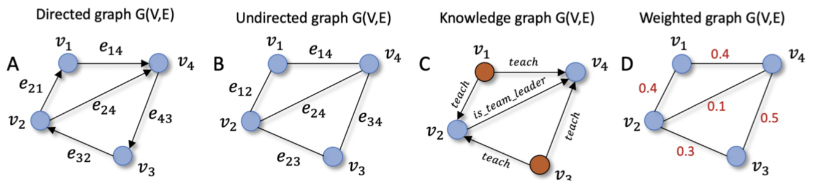

Figure 1: Representation of different types of graphs.

Graph embeddings can be roughly divided into four main groups:

- Node embeddings: each node is encoded with its own vector representation. This technique is used when we aim to perform visualization of prediction on the node level (i.e., visualization of nodes in the 2D plane, or prediction of new connections based on vertex similarities).

- Edge embeddings: aims to represent an edge as a low-dimensional vector. Benefits edge (/node pairs) related graph analysis, such as link prediction, knowledge graph entity/relation prediction, etc.
- Substructure embeddings: focuses on embedding a part of the graph (i.e, embeds the graph structure between two possibly distant nodes). The embedding of substructures can be derived by aggregating the individual node and edge embedding inside it. However, such a kind of "indirect" approach is not optimized to represent the structure. One main challenge in this task is how to generate the embedding target structure.
- Graph embeddings: the graph is represented in its whole with a single vector. These embeddings are commonly used to make predictions on the graph level or when we aim to compare or visualize whole graphs, usually for small graphs.

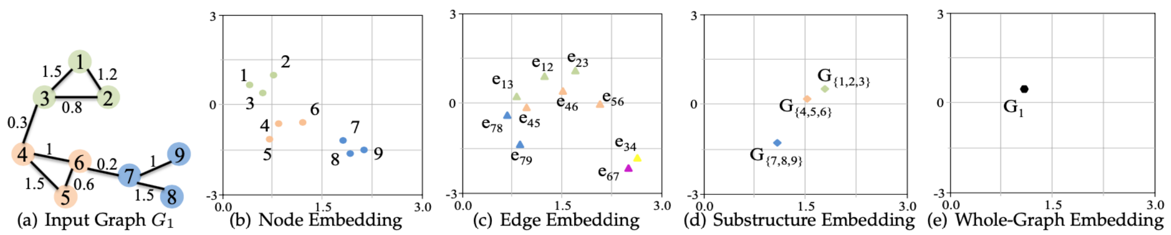

Figure 2: Embedding a graph into 2D space with different granularities. (a) Sample network used as a reference for graph embedding. Different node colors depict different substructures. (b–e) Different static graph embedding outputs. Note that the colors refer to the substructures and connections displayed in panel (a).

3 Comparing traditional ML and new approach

In traditional ML for graphs, we need to do feature engineering for each level that we want, such as; node level, edge level or graph level. For example, features can be defined in node level are; node degree, node centrality, clustering coefficient and graphlet. The problem of traditional ML model on graph is hand-designed features that should be done every single time.

In the figure 3 we can see big picture of applying ml algorithms on graph. The first question can be raised is that, how to automatically learn features instead of hand-design features? In the following we will try to answer this question. Additionally, different level of embedding and application of each of them will be described.

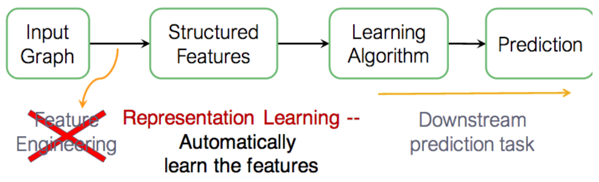
Input Graph
Structured Features
Learning Algorithm
Prediction
~~Feature Engineering~~
Representation Learning -- Automatically learn the features
Downstream prediction task

Figure 3: Procedure of applying ML algorithm on graph

4 Different type of ML tasks in graph

In the figure 4 four types of embedding levels have been shown. In the following some applications of them are provided:

- Node level: Node classification or node prediction for example predict property of a node.
- Edge level: Link prediction: predict if there are missing links between two nodes.
- Community or sub graph level: Classifying communities.
- Graph level: We can categorize it to two group: Graph classification, for example: Molecule property prediction and Graph generation, for example Drug discovery.

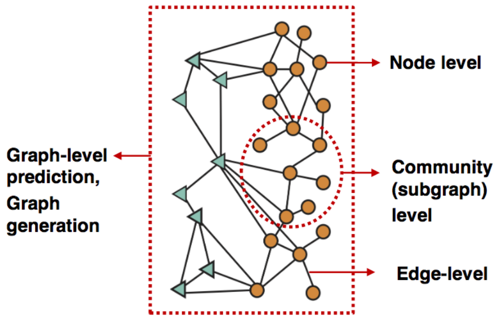
Node level
Graph-level prediction, Graph generation
Community (subgraph) level
Edge-level

Figure 4: Different level of a graph

4.1 Node Embedding

In the node embedding, we want to represent each node as a vector in d dimensional space, like figure 5, it will help us to apply machine learning algorithms on the graph. In this mapping, the function f has been used. This function should help us to provide better embedding vector. How we will understand that our embedding vector is a good representation of a node? To answer this question we should go more into detail of embedding procedure.

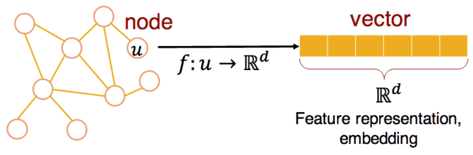

Figure 5

In the learning, node embedding we should know two concepts; encoding and decoding. In the encoding, like figure 6, nodes will be mapped to embedding space but this embedding should keep structure of nodes in graph, for example if two nodes are similar in original network, like u and v they should be similar in embedding space too. We can say that ENC is equal to function f that mentioned above.

$$ENC(u) = Z_u$$

Moreover, we need to define decoder too. In the decoder we calculate similarity of two vectors in embedding space and find a score for similarity between them. Similarity in the embedding space is inner product between two vectors, like below equation:

$$similarity(u, v) \approx Z_v^T.Z_u$$

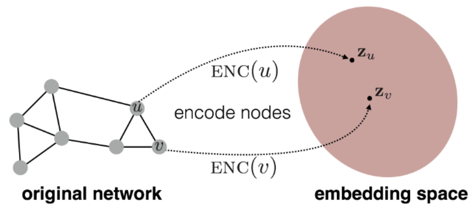

Figure 6: Encoding nodes

Our objective function is maximizing $Z_v^T.Z_u$ for nodes u and v that are similar in original network. How do we know that two nodes are similar? There are several options that we can say two nodes are similar:

- If there is a link between two nodes.
- If they have common neighbors.
- If they have similar structural roles.

In the following two strategies has been described that help us to find an embedding vector that keeps similarity between two nodes.

4.1.1 Random walk

This method is unsupervised or self-supervised way of learning node embedding. Our goal is estimating embedding of each node that preserve some aspect of network structure. In a random walk we randomly select a neighbor of a node, and move to this neighbor; then we select a neighbor of this point at random again. We repeat this procedure until reaching the maximum length.
In the random walk, similarity between two nodes defined as if these two nodes co-occurred in a random walk. To find co-occurrence between two nodes, conditional probability has been used. It will help us to find probability of visiting node v when we started from node u; $p(v|z_u)$.
If we go further and consider neighbor of node u instead of just one node, v, that co-occurred

in random walk, we should revise formula like below, in this formula $N_R(u)$ is neighborhoods of node u by strategy R.

$$p(N_R(u)|Z_u)$$

As mentioned before our objective function is maximizing similarity to find a good representation of a node in embedding space. For this method we have following objective function.

$$\max_f \sum_{u \in v} logp(N_R(u)|Z_u)$$

To parameterize $p(N_R(u)|Z_u)$, softmax has been used because we want to consider similarity between nodes out of all nodes. Therefore, final objective function will be like below:

$$L = \sum_{u \in V} \sum_{v \in N_R(u)} -log(\frac{exp(Z_u^T.Z_v)}{\sum_{n \in V} exp(Z_u^T.Z_n)})$$

We should minimize L to find Z_u that can preserve structure of node u in embedding space. Using random walk has some advantage and disadvantage. For example, random walk incorporates both local and higher-order neighborhood information. Higher order neighborhood means that nodes which are connected with more than one edge. But its computation complexity is $O(|v|^2)$ because of nested sum that exist in L.
To solve this problem negative sampling will help us to rather using all nodes, just considering k random nodes, negative samples. Additionally, to distinguish between target and noise nodes logistic regression has been used. With negative sampling complexity reduced to $O(k.|v|)$, that k is number of samples that we are considering.

4.1.2 Node2vec

In this technique we have more flexibility, it means that there is a trade off between local and global views of the network. Local view or micro-view of neighbourhood is BFS, Breadth First Search, and global view or macro-view of neighbourhood is DFS, Depth First Search.
To reach these views, two parameters have been defined, p and q. In the figure 7, trade of between these parameters has been shown.

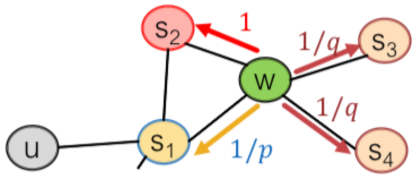

Figure 7: Trade off between p and q

If we are in node w and our previous node was S_1, we should decide where should we go in next step. Parameters p and q will help us to make this decision instead of randomly go to another node like what random walk do in each step. P is "Return" parameter and q is "walk away" parameter. If we assign low value for p, there is high provability to return to previous nodes, it is related to local view or BFS. On the other side if we assign low value for q, there is high provability to go further of starting point that is related to global view or DFS.

5 Practical Implementation

5.1 Dataset

The dataset used for this assigment is the CareerVillage. It can be downloaded from Data Insights Career Village — Kaggle. CareerVillage.org has provided several years of anonymized data and each file comes from a table in their database. The dataset is quite huge and consist of 15 table's: `‘answers.csv’`, `‘answer_scores.csv’`, `‘comments.csv’`, `‘emails.csv’`, `‘groups.csv’`, `‘group_memberships.csv’`, `‘matches.csv’`, `‘professionals.csv’`, `‘professionals_info.csv’`, `‘questions.csv’`, `‘question_info.csv’`, `‘question_scores.csv’`, `‘school_memberships.csv’`, `‘students.csv’`, `‘tags.csv’`.

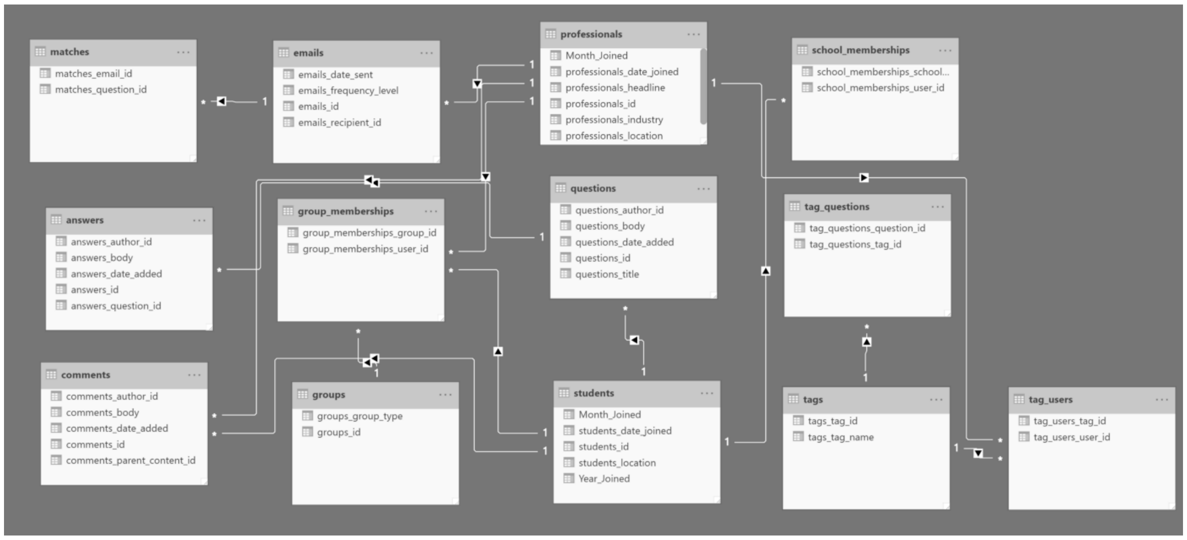

Figure 8: Datset class diagram overview for used tables.

5.2 ML method

Our goal in this study is classifying professional nodes from student nodes. First, Some preprocessing steps have been done on the graph and then two strategies for embedding nodes have been applied.
In next step, data set splitted to training and test set, 80% of data considered as train set and remaining as test set. Then, SVM model is trained on train set for each strategy and based on its performance on test set, we could find which embedding strategy is good for our goal in the graph.
At the end based on best strategy that gave us better embedding for nodes, we applied t-SNE for dimensionality reduction to visualize the results.

5.3 Embedding strategies

Two strategies have been used for node embedding in this study and we used packages that have been defined in Embedding graph— github.

- Deep walk: graph embedding techniques that uses walks, which are a concept in graph theory that enables the traversal of a graph by moving from one node to another, as long as they are connected to a common edge.
  In this strategy, concept of random walk has been used and our graph converts to sequence of nodes. As we described in the last section, in random walk, the model generate a random path of nodes that are connected.
  After applying this strategy with embedding size equal to 10 we could reach to 87.05% accuracy.

- Node2vec: Different values for p, q and embedding size have been used. Between different values in p and q, we could reach higher accuracy when $p = 1$ and $q = 2$ for three embedding size that have been tested.
  When embedding size was 10, we reached to 70.33% accuracy, with assigning different values for p and q, the accuracy did not have considerable changes. Maybe because of small size of embedding, model could not encode enough features for nodes.
  On the other side, when we assign high value for embedding size for example 30, we could reach to 72.23% accuracy and when we used 128 for embedding size our accuracy increased to 74.56%.
  Additionally, when we had larger value for embedding size, changes in p and q could give us wider range of changes in accuracy. Based on optimum values for p and q, we can conclude that local view or BFS is more important in classifying student and professional nodes than global view or DFS.

With respect to results that we get form these two strategies, best strategy is deep walk in embedding student and professional nodes. In the figure 9 we can see confusion matrix for SVM after applying deep walk embedding.

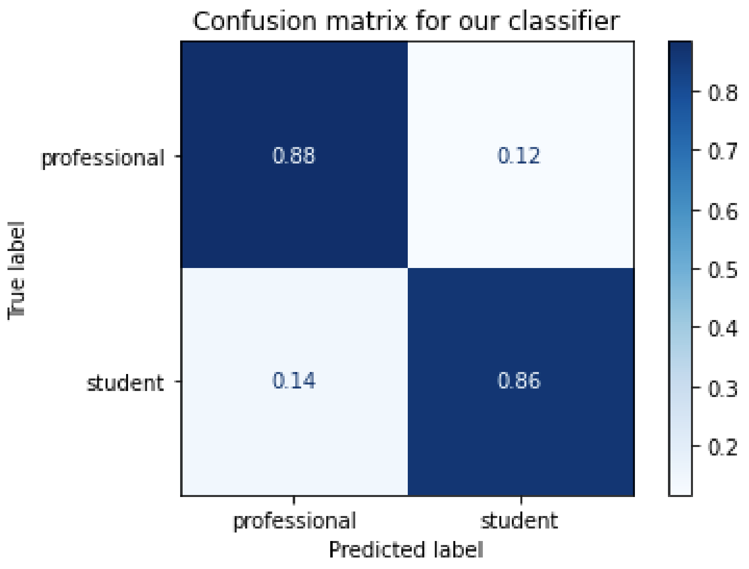

Figure 9: Confusion matrix

Based on the result of confusion matrix, our model behaves well, especially in detecting professionals. Additionally, in the figure 10 first 3000 train items considered and their dimension reduced from 10 to 2 with applying t_SNE.

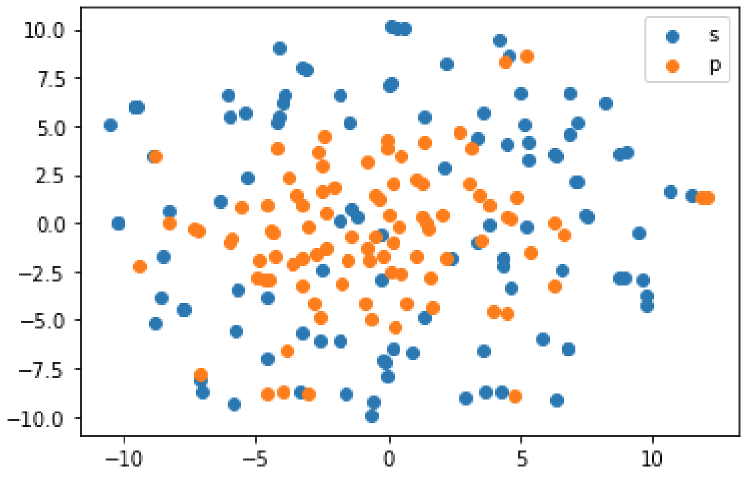

Figure 10: Dimension reduction

References

[1] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. "A comprehensive survey of graph embedding: Problems, techniques, and applications". In: *IEEE Transactions on Knowledge and Data Engineering* 30.9 (2018), pp. 1616–1637.

[2] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[3] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[4] Flawnson Tong. *Graph Embedding for Deep Learning — Towards Science*. Online; accessed June 2022. URL: https://towardsdatascience.com/overview-of-deep-learning-on-graph-embeddings-4305c10ad4a4.

[5] Mengjia Xu. "Understanding graph embedding methods and their applications". In: *SIAM Review* 63.4 (2021), pp. 825–853.