AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering

Development Workshop

Graph Embeddings

Project

Authors: *Filip Jachura*

Klaudia Klepacka

Cracow, 2022

1. Introduction

The aim of this project was to analyze different types of graph embedding algorithms and make the application for finding the most similar cities to the given ones based on graph embedding algorithms. The project was made using Python and Neo4j.

2. Graph embeddings - theoretical description

Graph embeddingd are data structures used for fast-comparison of similar data structures. They determine a fixed length vector representation for each entity (usually nodes) in our graph.
Node embedding techniques usually consist of the following functions:

- **Similarity function** - calculates the similarity between nodes
- **Encoder function** - generates the node embedding
- **Decoder function** - reconstructs pairwise similarity
- **Loss function** - checks the quality of the reconstruction

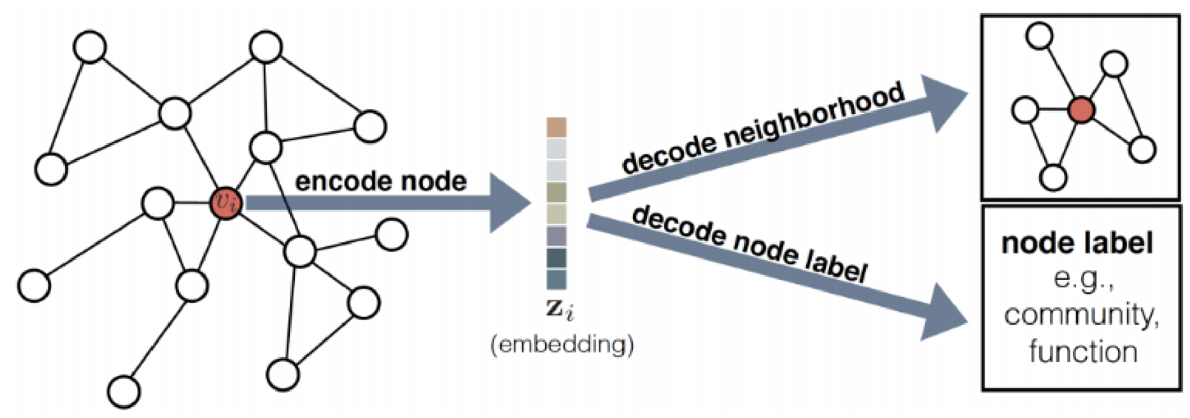


Fig. 1. Functions used in graph embedding algorithms https://arxiv.org/pdf/1709.05584.pdf

Distance calculations in an embedding work in similar ways to calculating distance between two points on a map. The key is that instead of just two or three dimensions, we may have 200 or 300 dimensions. The only difference is the addition of one distance term for each new dimension.

2.1. Fast RP

Fast Random Projection, or FastRP for short, is a node embedding algorithm in the family of random projection algorithms. These algorithms are theoretically backed by the Johnsson-Lindenstrauss lemma according to which one can project n vectors of arbitrary dimension into O(log(n)) dimensions and still approximately preserve pairwise distances among the points. In fact, a linear projection chosen in a random way satisfies this property.

Two key features of FastRP are:

1) it explicitly constructs a node similarity matrix that captures transitive relationships in a graph and normalizes matrix entries based on node degrees

2) it utilizes very sparse random projection, which is a scalable optimization-free method for dimension reduction.

An extra benefit from combining these two design choices is that it allows the iterative computation of node embeddings so that the similarity matrix need not be explicitly constructed, which further speeds up FastRP.

2.2. Node2Vec

Node2Vec is a node embedding algorithm that computes a vector representation of a node based on random walks in the graph. The neighborhood nodes of the graph is also sampled through deep random walks. This algorithm performs a biased random walk procedure in order to efficiently explore diverse neighborhoods. Each node in a graph is treated like an individual word, and a random walk is treated as a sentence. By feeding these "sentences" into a skip-gram, or by using the continuous bag of words model paths found by random walks can be treated as sentences, and traditional data-mining techniques for documents can be used. The algorithm generalizes prior work which is based on rigid notions of network neighborhoods, and argues that the added flexibility in exploring neighborhoods is the key to learning richer representations of nodes in graphs.

2.3. GraphSage

GraphSage is an iterative algorithm that learns graph embeddings for every node in a certain graph. The novelty of GraphSage is that it was the first work to create inductive node embeddings in an unsupervised manner. Just like in NLP, creating embeddings are very useful for downstream tasks. GNNs can use node embeddings for various tasks including node classification, link prediction, community detection, network analysis, etc.

3. Implementation

The project was made using Python with Jupyter Notebook and Neo4j database.

Firstly, the data was imported to the Neo4j database. There are 895 nodes (cities) and 1250 edges (roads). Each node has id, countryCode and name property.

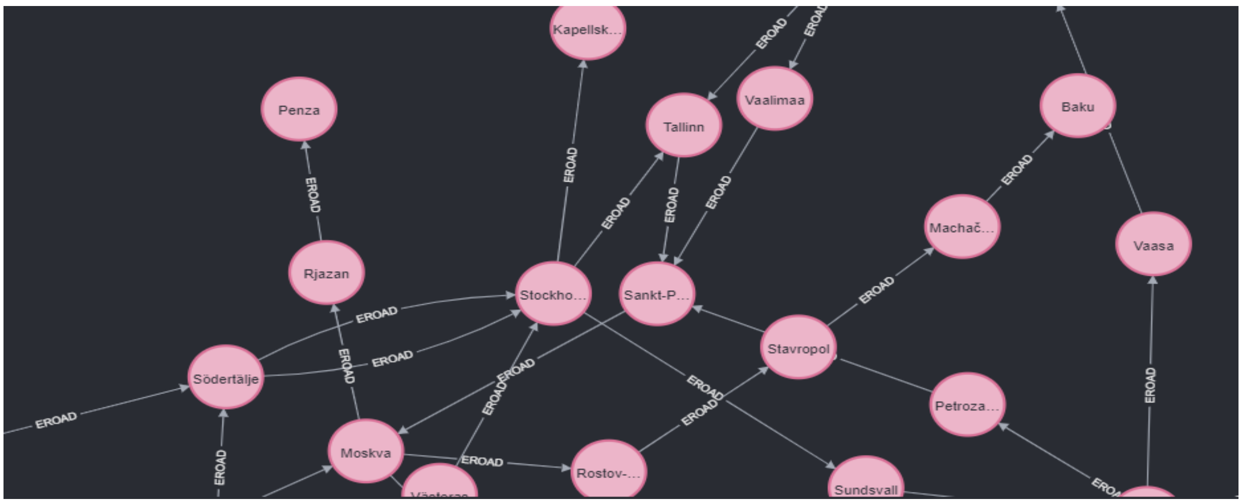


Fig. 2. Database structure in Neo4j

Then, graph embedding algorithms from the Graph Data Science Library were run on the dataset. As a result, there are dataframes with following properies: node id, name, country code and calculated graph embeddings - x, y. Additionally, there are charts, which show how the countries are located on the 2D coordinate system. The results are shown for Fast RP in 2 dimensions, Fast RP in 10 dimensions, Node2Vec in 2 dimensions, Node2Vec in 10 dimensions, GraphSage in 2 dimensions and GraphSage in 10 dimesions. Moreover, each of the above algorithms are tested for finding three the most similar cities for Warsaw. The best results was achieved for Node2Vec algorithm in 10 dimensions.

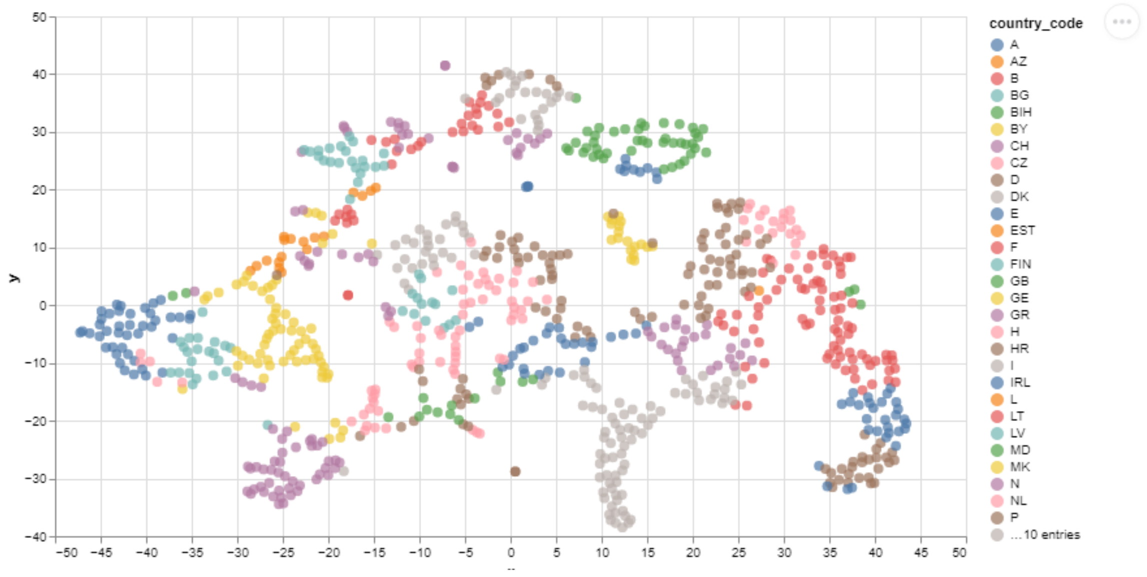


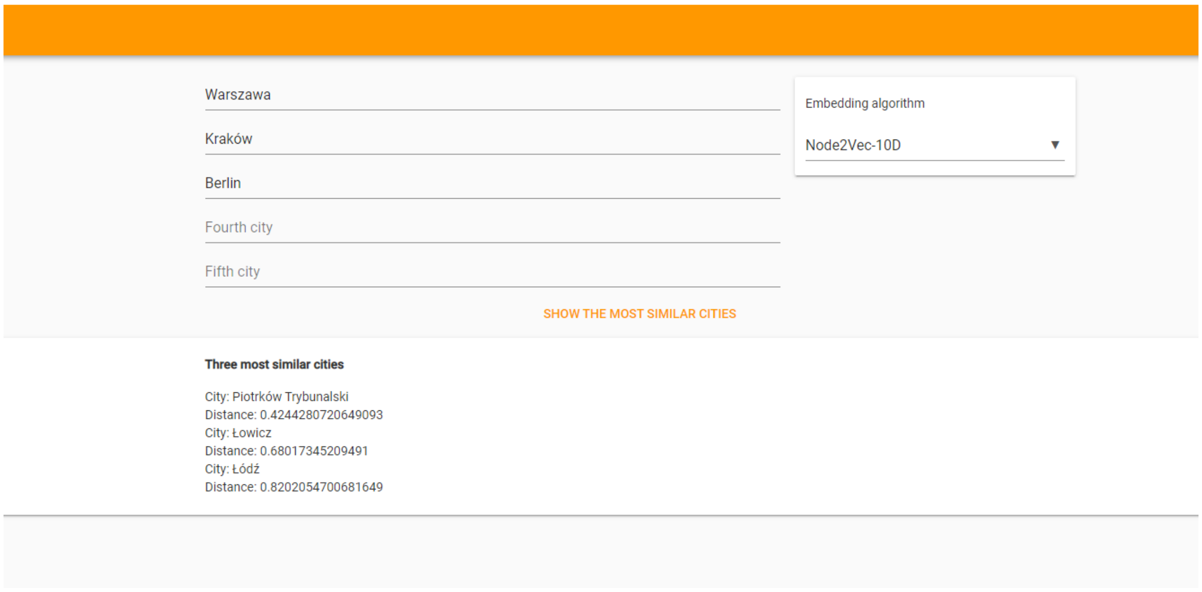
Fig. 3. Chart for Node2Vec in 10D algorithm displaying counties embeddings

The whole code is available at the following link:
https://github.com/KlaudiaK1/GraphEmbeddingsAlgorithms

4. User Interface

Anvil was used to create a web app. It is a framework for building full-stack web apps with nothing but Python. Firstly, Jupyter Notebook should be connect to the Anvil server using `anvil.server.connect(key)` command. Then, there is a possibility to define functions available in Anvil by using `@anvil.server.callable` before implementing them.

In the application user can enter up to 5 different countries and select an embedding algorithm. When user enters more than one country, the centroid is calculated, and then based on the embeddings the nearest points on the coordinate systems are returned, so user receive three the most similar countires to the given ones.



Warszawa
Kraków
Berlin

Embedding algorithm
Node2Vec-10D ▼

Three most similar cities

City: Piotrków Trybunalski
Distance: 0.4244280720649093
City: Łowicz
Distance: 0.68017345209491
City: Łódź
Distance: 0.8202054700681649

Fig. 4. User interface for the graph embeddings application

5. Summary

As a result of this project, we created an application which allows user to put up to 5 different countries, select an embedding algorithm and received three the most similar countires to the given ones as a result.

During working on this project we encountered the following difficulties:

- The documentation for embedding algorithms applies to the old version of neo4j
- It's not easy to obtain satisfying results for each algorithm

6. Bibliography

- https://www.geeksforgeeks.org/node2vec-algorithm/
- https://neo4j.com/developer/graph-data-science/graph-embeddings/
- https://www.tigergraph.com/blog/understanding-graph-embeddings/
- https://arxiv.org/pdf/1709.05584.pdf
- https://medium.com/analytics-vidhya/ohmygraphs-graphsage-and-inductive-representation-learning-ea26d2835331