MACHINE LEARNING IN NETWORK SCIENCE

CENTRALESUPÉLEC

Lab 3: Graph Representation Learning

Instructor: Fragkiskos Malliaros
TA: Rajaa El Hamdani

February 28, 2022

Description

In this lab, we focus on *neighborhood reconstruction methods* relying on random walks to learn node representations (DeepWalk, node2vec). We then evaluate the performance of the derived embeddings on two downstream machine learning tasks: link prediction and node classification.

Part I: Random Walk-Based Approaches

Many node representation learning methods have been inspired by the advancements in the area of *natural language processing* (NLP), borrowing various ideas originally developed for computing *word embeddings*. A prominent example here is the *Skip-Gram* architecture [2], which aims to find latent representations of words by estimating their context within the sentences of a textual corpus. To this direction, many pioneer studies in graph representation learning [3, 1] utilize the idea of random walks to transform graphs into a collection of sentences – as an analogy to the area of natural language – and these sentences or walks are later being used to learn node embeddings.

Exercise 1

In this first part you will be using the Karate graph stored in the file `karate.gml`. Each node of the graph has a label stored in the `community` attribute. The function `get_node2community` from `helpers.py` is used to get the nodes labels. The labels indicate to which community a node belongs.

1. In the first exercise, we will implement the `generate_random_walk(graph, root, L)` method in order to generate a random walk over a given network from a starting node `root`. The uniform random walking is a very natural strategy which is adopted in the DeepWalk model [3].

   You will then implement `deep_walk(graph, N, L)` method to generate N short random walks of length L over the graph G. You can find the detailed required steps of both functions in Algorithm 1 and Algorithm 2.

2. You can now learn node representations by running the SKIPGRAM model [2] over the node sequences that you have generated. Please extract the embedding vectors by running the `Word2Vec` method of the *Gensim*[1] library. You will need to change the default parameters of `Word2Vec` method in order to run the SKIPGRAM version with hierarchical softmax similarly to the DeepWalk model [3].

[1]https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec

Algorithm 1 RandomWalk

```
Require: A graph $G = (\mathcal{V}, \mathcal{E})$, the starting node $root$, the walk length $L$.
Ensure: : The list of nodes visited by the random walk $walk$
1: $walk \leftarrow [root]$
2: $current \leftarrow root$
3: for $l = 2, \dots, L$ do
4:     $next \leftarrow$ Uniformly sample a node from the set of neighbors of $current$ node at random.
5:     $walk \leftarrow walk + [next]$
6:     $current \leftarrow next$
7: end for
```

Algorithm 2 DeepWalk

```
Require: A graph $G = (\mathcal{V}, \mathcal{E})$, the number of walks $N$, the walk length $L$.
Ensure: The list of random walks $walks$
1: $walks \leftarrow []$
2: for $n = 1, \dots, N$ do
3:     $\mathcal{O} = Shuffle(\mathcal{V})$
4:     for each node $v \in \mathcal{O}$ do
5:         $current\_walk = RandomWalk(G, v, L)$
6:         $walks \leftarrow walks + [current\_walk]$
7:     end for
8: end for
```

3. We can visualize the embeddings by mapping them into 2D space with t-SNE package. You can simply use `visualize` method to plot the learned representations.

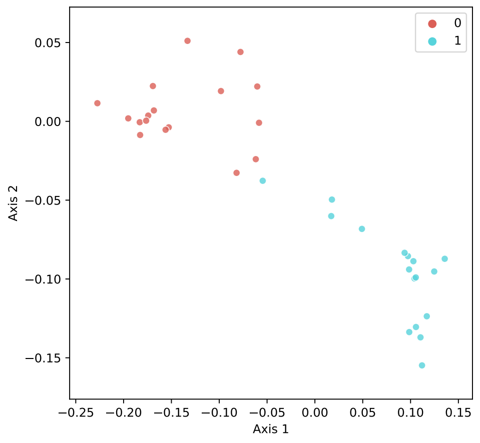

Figure 1: An example visualization of embedding vectors

Part II: Link Prediction

Reminder on link prediction:

- We remove a certain fraction of edges chosen randomly from the network in order to construct *positive* samples for the testing set. (The network remains connected during the process). The remaining edges in the residual network are used to generate the positive samples of training set.
- The same number of distinct node pairs which are not linked in the initial network, are randomly sampled and added to the testing and training sets as *negative* samples.

- We then learn node embeddings on the residual network via a carefully chosen representation learning method.
- The feature vector of an edge (v, u) is formed with a defined operation on the embedding vectors x and y corresponding to nodes v and u (e.g. $|x_i^v - y_i^u|$).

Exercise 2

1. Perform link prediction using `generate_sample` and `edge_prediction`, as well as the embeddings generated above by the `Word2Vec` model.
2. How does the AUC score behave depending on the walk length and the number of walks ? Does the window size of the model have any impact on the performance? If yes, how does it affect the scores?
3. How can you extend the uniform walking strategy in Algorithm 2 ?
4. Perform again the link prediction task, but using this time `Node2Vec`, available at `https://github.com/eliorc/node2vec`. Do not forget to download the package.
5. What are the main limits of these approaches? How could you improve them? (Do not skip this question, really try to think about it).

Part III: Node Classification

In the node classification task, we have access to the labels of a certain fraction of nodes in the network (training set), and our goal will be to predict the labels of the remaining nodes (test set).

Exercise 3

1. Perform a classification experiment for a training set ratio of 60% using the Karate graph. You will need to complete the implementation of `node_classif` which takes the embeddings of the nodes and their labels.
2. Repeat this task for another, more complex dataset: *musae_facebook_edges.csv* + *musae_facebook_target.csv*, already downloaded. This is a page-page facebook network, where nodes stand for facebook official pages, and edges exist if two pages mutually like each other. These pages belong to four different categories (politicians, governmental organizations, television shows and companies) and we would like to train a model to predict them. To do so, you will first need to import correctly the data, including the graph and node labels. You then pre-process them before computing node embeddings. Finally, you will construct your model and evaluate it, similarly to what was done in the previous section. You can re-use `node_classif`.
3. If you have tried to train a Node2Vec or DeepWalk or Word2Vec model on this dataset, what do you notice? (Compare to previous dataset).

References

[1] 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*, pages 855–864, 2016.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[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*, pages 701–710, 2014.