Graph Embeddings for Recommender Systems

In this project, we will revisit the problem central to recommender systems: predicting a user's preference for some item they have not yet rated. Like the Spark recommender from the first project, we will use a collaborative filtering model to explore this problem. Recall that in this model, the goal is to find the sentiment of a user about a particular item Unlike the the first project that used the ALS method, however, we will perform this task using a graph-based technique called DeepWalk. The main steps that you will complete are to:

1. Create a heterogeneous information network with nodes consisting of users, item-ratings, items, and other entities related to those items
2. Use DeepWalk to generate random walks over this graph
3. Based on these random walks, embed the graph in a low dimensional space using word2vec.

The goal of this project is to evaluate and compare *preference propagation algorithms* in heterogeneous information networks generated from user-item relationships. Specifically, you will implement and evaluate a word2vec-based method.

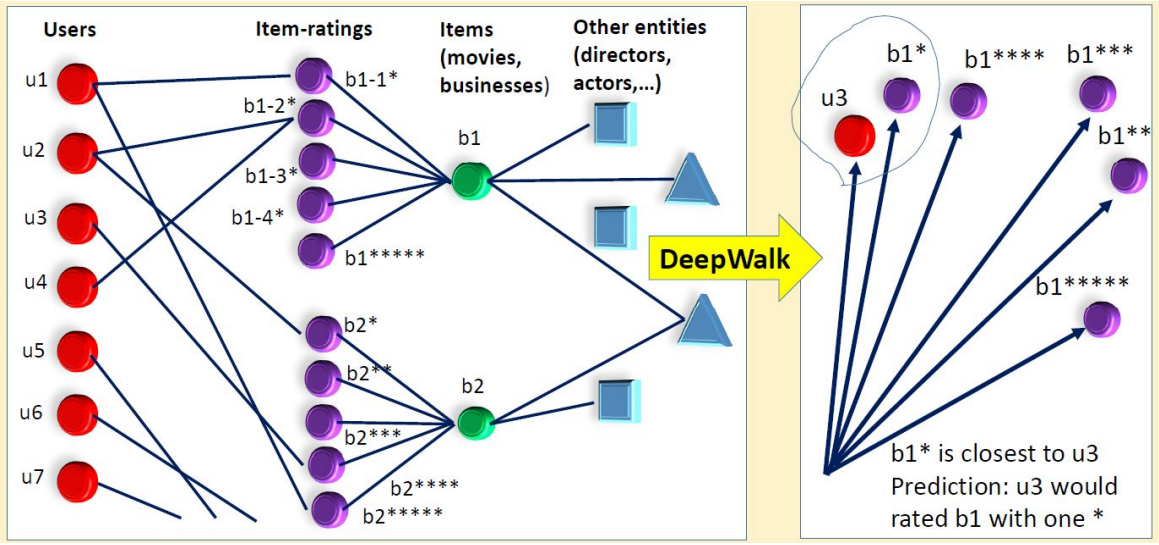

Fig 1: Heterogeneous network (left) embedded into low dimensional space (right). Unknown item ratings are selected based on the item-rated node most similar to the given user.

Background

User-item relationships can be represented as heterogeneous information network. In such a network, entities of different categories are placed as nodes in the same network (e.g., users, businesses, actors, directors). The edges between nodes represent some form of a relationship. The network-based entity recommendation systems, which utilize user-item relationship information, require a method for preference propagation over the network [1].

Data embedding is used in machine learning applications to create low-dimensional feature representations, which preserves the structure of data points in their original space. Embedding means low dimensional vector representation of entities. In particular, information network embedding is representation of a network in low dimensional vector space which preserves information about the structure of the network. The main goal of the heterogeneous embedding task is to learn mapping functions to project data from different modalities to a common space so that similarities between objects can be directly measured.

One of the recent efficient to obtain a network embedding is DeepWalk [4]. DeepWalk produces the embeddings of nodes of the information network by first creating corpus of text, where words are node ids and sentences are random walks originated at each node of predefined length. After generation of the random-walk corpus, DeepWalk feeds it to word2vec, which produces the embeddings. word2vec, a shallow neural network-based set of algorithms [2] has been applied outside of the natural language processing (NLP) domain to, effectively map latent features hidden in the paths over the information network into a vector space.

It's been shown that the resulting low dimensional embeddings, which are produced by word2vec, are equivalent to factors in matrix factorization approaches [3] and [5]. Note, that in this project, word2vec has not been used for its originally intended NLP purpose as, for example, for extraction of sentiment from user reviews. Rather, word2vec, will be used as a low-dimensional embedding tool, capitalizing on the fact that it is agnostic to the type of vocabulary and corpus that is fed to it.

Project Setup

You will be making your edits in the original DeepWalk source code, which can be found here: https://github.com/phanein/deepwalk/tree/master/deepwalk. There are a number of packages that are required, see `rec2vec/graph.py`, but you should have all of these from previous projects. If not, they should be simple to install.

Datasets

You are provided with several movie datasets located in the data folder. This folder includes a user-movie ratings dataset, given in a training and test set, as well as other data files containing information about the movies, such as the actors, directors, and genres. Note that these data files are hardcoded into the code and read into python objects for you.

Project Requirements

You are expected to implement parts of the algorithm as outlined in the introduction. You will be making your edits to the files `rec2vec/graph.py` and `rec2vec/__main__.py`. As usual, your job is to complete the missing parts of the file, look for the tag `# YOUR CODE GOES HERE` and fill in the missing parts to complete the code. Specifically, you must complete the following functions:

`records_to_graph`

Here you will be converting the data into a heterogenous information network. The nodes of the graph are the users, movies, actors, genres, directors, and ratings. Edges a made between these entities if, for example, an actor starred in a movie, the movie is of a certain genre, or a user gave some movie a rating.

Five rating nodes should be created for every movie, as illustrated in Figure 1. For example, suppose there was a movie named “b”, there should be five nodes created: one for each possible rating. The simplest way to name these nodes is probably by append an “_i” for i from 1 to 5. These nodes should be connected to their respective movie nodes. Note: the user nodes should not be connected to the movie nodes, rather they should be connected to movie rating nodes which in turn are connected to a movie node.

The data files needed to make this graph are already read-in and stored in convenient python objects for you. You are welcome to change any of this if you desire.

`process`

The only code you must add to this section is for performing the random walks using DeepWalk and the embedding using word2Vec. For the former, you just need to call the `build_deepwalk_corpus` function (provided in graph.py) with the arguments taken from the argparser variable `args` (e.g., args.walk_length). For the latter, you must call the Word2Vec function. Pass the following optional arguments: size, window, min_count=0, workers. Again, take these from the `args` variable.

`predict_rating`

This function should predict the rating of a user-movie pair. Remember to use `nodedict` to map the user and movie IDs to their corresponding graph IDs (the id field of the node object) as this is what DeepWalk was built from. Also, be aware that the movie input argument is the movie ID not the movie rating IDs, which are the five nodes you want to compare the similarity with.

Submission Instructions

Zip your rec2vec folder and submit this on moodle.

Running the program

The program can be run using the following command from root folder (not inside rec2vec):

```
python -m rec2vec --walk-length 2 --number-walks 2 --workers 4
```

This should produce output that looks something like the following:

```
MSE = 3.397626
```

```
accuracy = 0.214639
[[  3   4   6   8   0]
 [ 11   9  17  16  11]
 [ 42  57  56  28  56]
 [ 79  83 102 100  86]
 [ 52  42  44  50  49]]
```

Note: your results will improve the larger your walk-length is, however it will take longer to run.

References

[1] Yu, X., Ren, X., Sun Y., Gu, Q., Sturt, B., Khandelwal, U., Norick, B., Han, J. (2014) Personalized entity recommendation: A heterogeneous information network approach. in *J Proc. 2014 ACM Int. Conf. on Web Search and Data Mining (WSDM'14),* New York, pp. 283-292.

[2] Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013) Efficient estimation of word representations in vector space. *arXiv:1301.3781v1*

[3] Levy, O., Goldberg, Y., (2014) Neural word embedding as implicit matrix factorization. *Advances in Neural Information Processing Systems 27,* pp. 2177-2185

[4] Perozzi, B., Al-Rfou, R., Skiena, S. (2014) DeepWalk: Online Learning of Social Representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery, (KDD'14),* pp. 701-710

[5] Yang, C., Liu, Z., Zhao, D., Sun, M., Chang, E.Y. (2015) Network Representation Learning with Rich Text Information. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015),* pp. 2111-2117