A graph-based recommendation system

Individual project

Chiara Guizzaro
Identity number: 2019293

Learning From Networks, Unipd, a.a. 2021-22

Abstract

Recommender systems are really well spread nowadays in web services such as Youtube, Amazon, Netflix and others. They are used for suggesting relatives items to users, thus leading industries to have more incomes and in general to stand out from the competitors when the algorithms applied are efficient. Here the project is based on using graph features and a GNN to implement a recommender system.

1 Introduction

A recommendation system is in general a way to guess what likely will be the preference of an user for a certain item. The possible approaches for determining what suggestions to make are many: given an user, we can exploit what others akin people have chosen, we can look at the past user's interactions, which are the most similar items with respect to the one we are interested in and so on; the important thing is that the final goal is always to help the user. Therefore when a system is effective not only is easier for a person to find something of their interest, but this also results in major incomes for the company that uses such technologies. Some examples of famous applications are for recommending movies, books, songs, restaurants and items in general for e-commerce services.

The usage of recommendation systems has became well spread thanks the modern concept of big data, hence nowadays several information regarding almost everything are available online. While all of these data can be surely helpful, on the other hand is not always so easy to find an efficient way to elaborate them. In addition there is also another consideration to make: the ”cold start problem”. It happens when a system is not able to produce an output or is not a good one, when the user or item interactions are very few. This is especially true when an user or item newly enters in the system.

Here, the proposed method for realizing a recommendation system is to apply some centrality metrics to solve the cold start problem and then use a Graph Neural Network (GNN) for suggesting items to users that have reached a minimum number of interactions.

Therefore in the next section I will introduce what are the most used approaches, then which is the dataset used in this case and a more detailed description of my method. Afterwards there will be an analysis of the obtained results and finally a conclusion.

2 Related Work

In general traditional approaches for having a recommender system can be divided in: collaborative filtering methods, content based methods and hybrid solutions.

Collaborative methods are based entirely on the past users-items interactions matrix in order to produce new recommendations. An advantage is that they do not require information about users or items and are really flexible. Moreover, the more interactions are added the more new recommendations become accurate. Collaborative methods can also be categorized into two more subset: Memory based and model based methods. The first ones, use only information from the user-item interaction matrix and they assume no model to produce new recommendations. Thus, assuming to have a matrix of (n users)x(m items) it is possible to identify users by its rows and then using k-nearest neighbors (K-NN) to find other users with similar interests. Instead, using columns to represents items, there is also the possibility to find the most preferred item for a given user and afterwards applying k-nearest neighbors to get similar items. But a drawback is that when the matrix is huge, K-NN can be infeasible, so the need to find other solutions such as exploit the matrix to be potentially sparse or to use approximate nearest neighbours methods (ANN). On the other hand, the model based methods assume the existence of a latent model supposed to explain the user-item interactions.

Content based methods, instead, in addition to the user-item matrix use also information about users and/or items. For example, the age, the sex, the job or any other personal information for users as well as the category, the main actors, the duration or other characteristics for items like movies. In this cases, the problem can be treated as a classification or regression task, where a variety of algorithms can be applied, such as Bayesian classifier for example.

Then there are also hybrid approaches that combine collaborative filtering and content based methods.

Now, focusing on the usage of GNNs for solving the recommendation problem, we can have GNNs using user-item interaction information, enhanced with social network information or enhanced with knowledge graphs.

The most famous GNNs using user-item interaction information are Graph Convolutional Network (GCN) [1] and its variant Neural Graph Collaborative Filtering (NGCF) [2]. They use the same concept of convolutional neural networks but operate directly on graphs. On the other hand GraphSAGE [3] and its variant PinSAGE [4] for handling web scale graphs, sample fixed-sized neighborhoods instead of using all the neighbors of each node for the aggregation process. It also provides min, max, or sum pooling as options for aggregating and uses concatenation operation to update node/edge/graph representations. While Graph Attention Networks (GAT) [5] uses an attention mechanism to learn the influence of neighbors to determine their contribution during the aggregation step and Multi-Component Graph Convolutional Collaborative Filtering (MCCF) [6] also combine the attention mechanism with features from explicit user-item interactions. Instead for GNNs enhanced with social network information, the most popular networks are DiffNet++ [7] which learns better user embedding from two separate graphs: the user-item graph and the user-user graph; and GraphRec [8] where the model uses the user-item interactions and user opinions from the same user-item heterogeneous (non-bipartite) graph. Finally, for GNNs enhanced with knowledge graphs, one of the most used method is Knowledge Graph Attention Network (KGAT) [9]. It's based on GAT and constructs a heterogeneous graph composed of users, items, and attributes nodes.

3 Dataset

Assuming that the user-item interactions can be represented as a graph, this last one is created by the data available at [10]. The dataset contains information regarding the preferences of users about some anime. The anime information are: unique id identifying the item, full name, genre (separated by comas for more than one), type, episodes, rating and members (number of community members that are in this anime's "group"). For the ratings the information are: user id (non-identifiable, randomly generated user id), the anime id (the anime that this user has rated) and a rating score in a range from 0 to 10, also if the user watched it but didn't assign a rating then the score is -1. Looking at the anime and rating tables, the initial number of nodes is 84715 (users + items), while the initial number of edges is 7813737 (interactions user-item). These values then decrease with a pre-processing phase which eliminates all data with Nan values, removes all rating values related to anime that don't have a corresponding description and finally drops all user-item interactions when a rate has not been given (rating = -1). Thus the final number of nodes is 79526 and the final number of edges is 6337239.
While for centrality metrics the whole graph is used, on the other hand for the GNN application, due to the high time complexity, a subset of $m = 15000$ nodes are extracted and thus only the relative 204188 edges are considered. Since the recommendation problem is treated as a link regression task, for training the GNN the edges are divided in 107198 elements for the training set, 35733 for the validation set and 61257 for the test set. Thus the labels are the corresponding ratings. Note that the data have no indication about the date of the rating, thus the subsets subdivision is done randomly. On the contrary in an industrial application the data would have been partitioned according to the date.

4 Method

Assuming that the user-item interactions can be represented as a graph, to solve the "cold start problem" the strategies implemented for this project are:

- when a new user enters the system, the top K items to propose are found thanks closeness-centrality (where the graph is considered both with and without weights) or degree-centrality metric;
- when a new item enters the system it is proposed to the most active users. Those are found using closeness-centrality metric (where the graph is considered both with and without weights) or degree-centrality metric.

When the user has reached a minimum number of user-item interactions, then the system can use these information to suggest the most likely anime to be watched (or a set of possible proposals). This is done thanks a Graph Neural Network (GNN), whose performances are compared with a base-line method and also with Node2Vec algorithm.

4.1 Centrality metrics

We assume for a graph $G = (V, E)$, $n = |V|$ is the number of vertices and $m = |E|$ is the number of edges.

4.1.1 Degree centrality

The degree centrality for a given node **u**, $degree(\mathbf{u})$ is equal to the number of adjacent vertices in an undirected graph or the sum of the outdegree and indegree in a directed graph.

4.1.2 Closeness centrality

The closeness centrality $c(\mathbf{v})$ for a node **v** is

$$c(\mathbf{v}) = \frac{n-1}{\sum_{\mathbf{u}\in V} d(\mathbf{u}, \mathbf{v})} \tag{1}$$

where $d(\mathbf{u}, \mathbf{v})$ is a distance measure from **u** to **v**.

4.2 HinSAGE

HinSAGE is a graph neural network, variant of GraphSAGE [3] where there are separate neighborhood weight matrices to account for heterogeneity. As for its original version, HinSAGE exploits node attributes information to efficiently generate node embedding on previously unseen data.
Therefore this model to solve the recommendation problem, firstly perform link prediction (Figure 1):

1. generate the nodes embedding, thanks the aggregation of one or more neighborhood information;
2. concatenate two nodes embedding to create an edge embedding;
3. for each edge embedding the model performs link prediction on the associated rate;

After a training phase, when the GNN performances are good enough, the recommendation for a given user **u** is done thanks:

I retrieve a list of possible candidate edges for performing link prediction. In details, an oriented tree is constructed thanks breadth-first-search (BFS) starting at source node **u**, then all anime nodes **v** at maximum distance $d = 4$ from the root **u** are considered (Figure 2);

II perform link prediction on the edge list of previous point;

III filter all rates that don’t reach a certain threshold T;

IV filter all suggestions already watched;

V return the top-k items.

4.3 Model evaluation

The evaluation of recommendation system can be done in different ways: one based on well defined metrics and another one mainly based on human judgment and satisfaction estimation. The latter aims to measure properties such as diversity and explainability of recommendations, thus something hard that depends also on subjective factors. On the other hand when a model outputs ratings predictions or matching probabilities, we can measure the performances using regression measurement metrics (Section 4.3.1) or also utilizing a threshold to binarize these values and apply traditional classification metrics.

Another option is also to compare rankings, when we have as output the top-K items. In this case some well known metrics are Kendall’s Tau τ [11] or Spearman’s Rho ρ [12], but even if they are often used, they have some drawbacks: they requires the two ranking lists to be conjoint, in other words same elements in both lists, and secondly, it is unweighted, hence an error at the bottom of the ranking is as important as an error at the top. For these reasons I use instead ”A Similarity Measure for Indefinite Rankings (RBO)” [13] for comparing the output rankings of the different centrality metrics (Section 4.3.2).

4.3.1 Regression metrics

For correctly training a model on the link regression task, a classic regression metric such as the mean squared error (MSE) has been chosen as loss function.

$$MSE(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{2}$$

where n is the total number of samples considered. Respectively y, $\hat{y}$ are the true and the predicted values.

On the other hand for measuring a model’s performances, the root mean squared error (RMSE)

$$RMSE(y,\hat{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2} \tag{3}$$

and mean absolute error (MAE) are used

$$MAE(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}| \tag{4}$$

4.3.2 A Similarity Measure for Indefinite Rankings

In general RBO is a similarity metric which uses weights for each rank position so to have a comparison between rankings that can have different elements in their list. In addition disagreements between items at the top list have higher weights than the ones at the bottom.

Let S and T two given rankings, then

$$RBO(S,T,p) = (1-p)\sum_{d=1} p^{d-1}A_d \tag{5}$$

where $d \in [1,\infty)$ is the ranking depth being examined, X_d is the size of the overlap between the two given rankings up to depth d and $A_d = \frac{X_d}{d}$. Also $p \in (0,1)$ is a tunable parameter that can be used to determine the top d ranks contribution to the final value of the RBO metric (the contribution of a single discordant pair decreases with the increase in the depth of the ranking). Used values are: $p = 0.9$ and $d = 10$. Instead to get a single RBO score, assuming that the agreement seen up to a depth k is continued indefinitely among the two lists, then:

$$RBO(S,T,p,k) = \frac{X_k}{k}\cdot p^k + \frac{1-p}{p}\sum_{d=1}^{k}\frac{X_d}{d}\cdot p^d \tag{6}$$

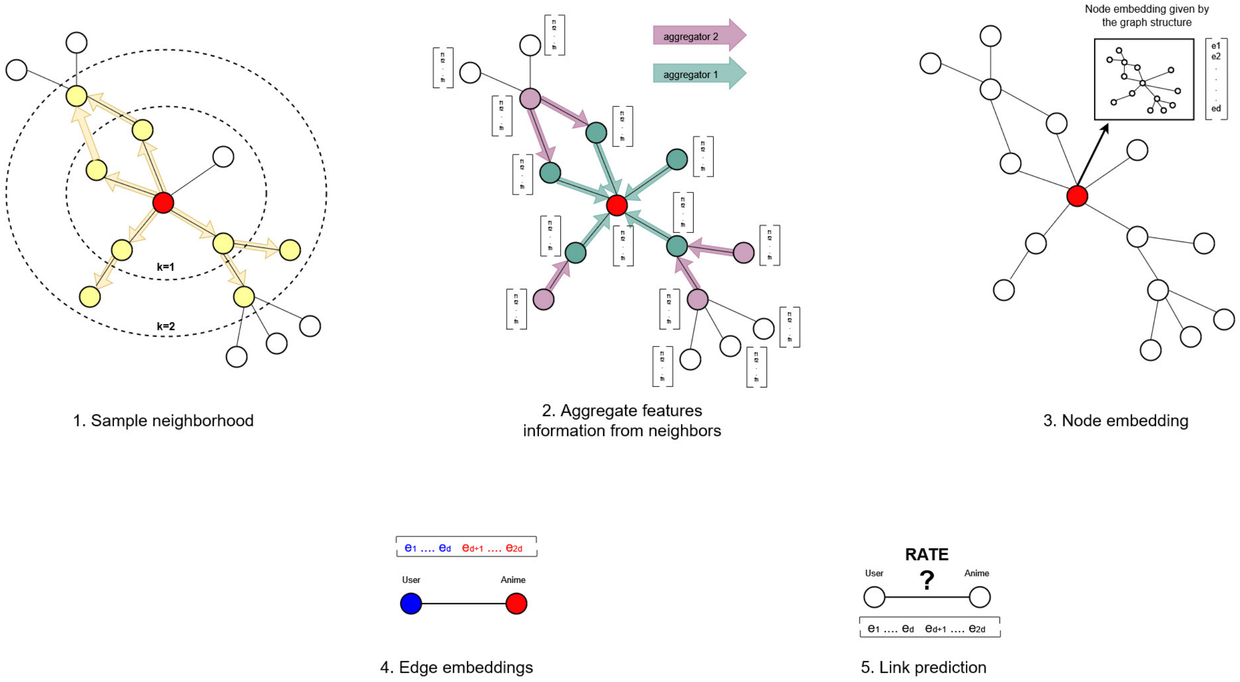
5 Experiments

Firstly I analyze the results on solving the ”cold start problem” by means of two possible centrality metrics: degree centrality and closeness centrality. Afterwards there is the evaluation of the GNN performances compared with a base-line method and Node2Vec [14].

Regarding centrality metrics, Figures 3-4, a first consideration is that degree centrality computed with and without considering the property of the graph to be bipartite (”degree B” and ”degree” respectively) produces the same ranking list, even if the final scores are different. Also closeness centrality with graph without weights has a ranking quite similar to the ones of degree metrics, but the order of some elements are different, while others are some new additions. Instead the closeness centrality with weighted graph (Dijkstra’s algorithm is used), where ratings are weights, is not really effective. This is due the fact that in this application rates are not distances as it is supposed in the computation of the Single-Source Shortest Path (SSSP) problem. Indeed the output ranking is rather different compared with the others. For this reason a function $f(r) = -r^2 + 110$ is used to map ratings into approximate ”distances”. For high rates we get low distances and decreasing the rate, the corresponding ”distance” climbs up.

It’s interesting to observe that results obtained thanks graph analysis are not the same as sorting the anime by global average rating. A possible reason is that while the graph centrality metrics are calculated based on the given user-item interactions, the global average rating is, as the name suggests, only a mean value among ratings. Thus an anime watched by a small community that really likes it and so having a high global average rating, it may happen that it’s not actually one of the most popular items. This assumption is supported by the fact that our metrics compared with the ranking given by sorting the anime by number of members are more similar.

Furthermore it’s to note that while for anime rankings, closeness FW (closeness centrality with a function of ratings as weights) is similar to degree, degree B, closeness and members metrics, the same is not true comparing results for user rankings. A possible reason is that for users


k=1
k=2
aggregator 2
aggregator 1
Node embedding given by the graph structure
e1
e2
ed
e_1 .... e_d e_{d+1} .... e_{2d}
User
Anime
RATE
?
User
Anime
e_1 .... e_d e_{d+1} .... e_{2d}

1. Sample neighborhood

2. Aggregate features information from neighbors

3. Node embedding

4. Edge embeddings

5. Link prediction

Figure 1: Pipeline for computing the embedding of a single node thanks HinSAGE. Then given two node embedding they are concatenated together so to have a single edge embedding for which it is possible to perform link prediction.

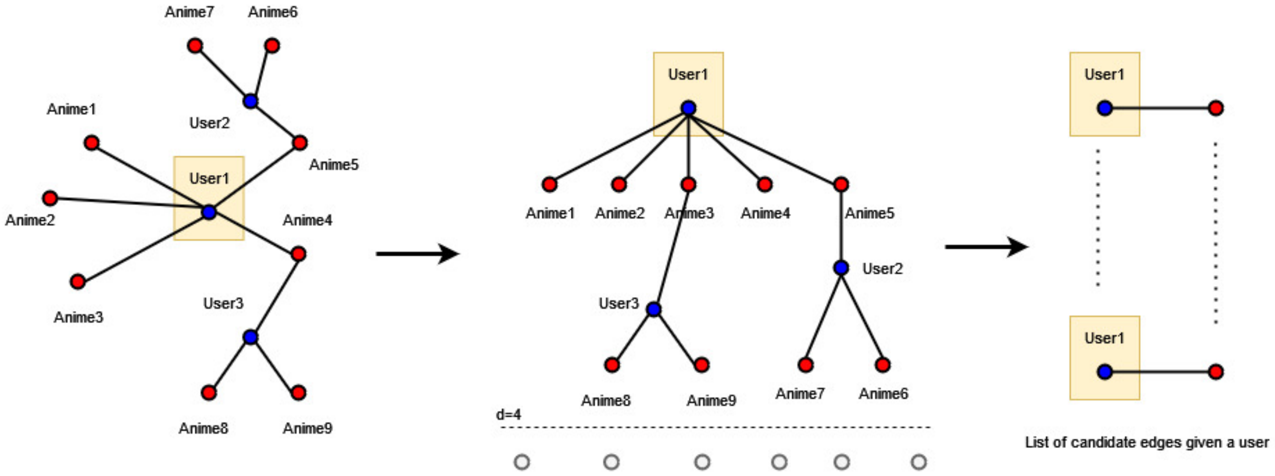


Figure 2: Pipeline for producing a list of possible edge candidates for a given user **u**. The respective recommendations will then be computed from the respective edge embedding and performing link prediction with additional filtering process. Note: the list of edge candidates is from user **u** and all the anime found with BFS tree with maximum depth $d = 4$ avoiding the creation of user-user edges.

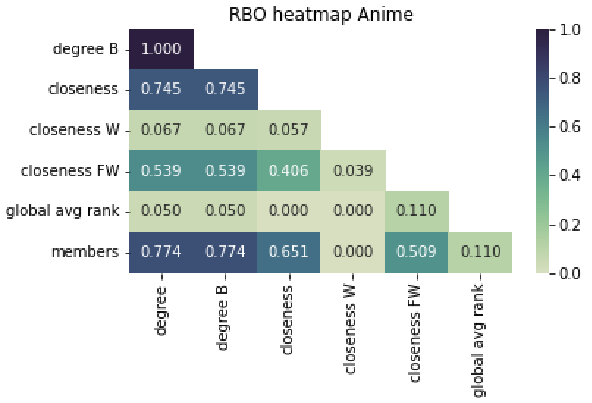


Figure 3: RBO heatmap for anime, where the comparison is between degree centrality without the bipartite property (degree), with the bipartite property (degree B), closeness centrality with unweighted graph (closeness), with weighted graph (closeness B) and also with a function to map weights (closeness FW). Moreover there are the global average rank (global avg rank) and the number of members (members) as dataset given information.

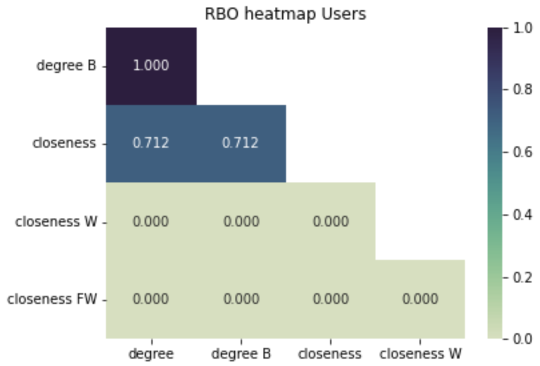


Figure 4: RBO heatmap for users, where the comparison is between degree centrality without the bipartite property (degree), with the bipartite property (degree B), closeness centrality with unweighted graph (closeness), with weighted graph (closeness B) and also with the addition of a function to map weights (closeness FW).

degree metrics and closeness centrality without weights, which are indeed similar in terms of RBO, consider the top users the ones which are more active in terms of anime watched. On the contrary closeness with ratings as weights judges the users on the values of rates given. Same for closeness centrality with a function of ratings as weights, but with different final results obviously given by the application of the function $f(r)$.

Finally regarding the time complexity, on one hand degree centrality (for both versions) is really fast to compute, while closeness centrality requires $9 \sim 10$ minutes for the unweighted graph and 41/52 minutes for the weighted variants. This is a lot of time if we also consider that a 'hand-made' implementation of Eppstein-Wang algorithm [15] for computing approximately the closeness metric is used. Therefore considering that these centrality metrics will be used to solve the cold-start problem, a frequent occurrence, having an operation that requires so much time, would be quite demanding. Thus degree centrality can be helpful when the operation is done with an high frequency, but if we decide to compute the most popular items to suggest and the most active users only from time to time (updating the values after a certain period has passed), then also closeness centrality could be used.

Now, focusing on the HinSAGE network, its application requires a set of features for each node, so handcraft feature vectors[1] are created in order to find suitable neighbors.

Regarding its perfomances, it is compared with a base-line method which gives in output always the mean ranking value, and with Node2Vec. In order to have a fair comparison, both GNN's embedding and the ones created with Node2Vec have afterwards a multi layers perceptron (MLP) network, with only one hidden dense layer and an output layer with one node to compute the final rating predictions. Looking at Table 1 is possible to see that HinSAGE outperforms the other two in terms of Mean Square Error (MSE) and Mean Absolute Error (MAE) for the link prediction sub-task. The same can be said analysing Figure 5 where additionally it is also possible to note that, while the ground truth labels correctly have a discrete distributions on values that go from 1 to 10, on the other hand the GNN and Node2Vec has continuous floating outputs that are somehow centered around the 7 value. It clearly appears that labels such as from 1 to 5 and 10 are really hard to model for the examined algorithms.

	MSE	MAE
base line	1.58319800527454	1.24777182518899
HinSAGE	**1.32179027273786**	**1.0094630544332**
Node2Vec	1.45437096555196	1.12369392193626

Table 1: Models perfomances

Therefore established that HinSAGE is the best model so far, just to give an example of what are the possible predictions for a random user **u**, who has watched this list of anime

```
Neon Genesis Evangelion:  The End of Evangelion
Code Geass:  Hangyaku no Lelouch
Bakemonogatari
K
Hataraku Maou-sama!
Noragami
Shokugeki no Souma:  Ni no Sara
```

the corresponding predictions are (from higher to lower):

```
Shin Megami Tensei Devil Children
Dansai Bunri no Crime Edge
Tanaka-kun wa Kyou mo Kedaruge
Kataribe Shoujo Honoka
M3:  Sono Kuroki Hagane Recap
Mahou Shoujo SonicoMagica
Tsubasa to Hotaru (2015)
Kekkai Sensen
Lovedol:  Lovely Idol
Eyeshield 21:  Jump Festa 2005 Special
```

[1] user features: [average rate given to anime]; anime features: [global average rating, number of members]

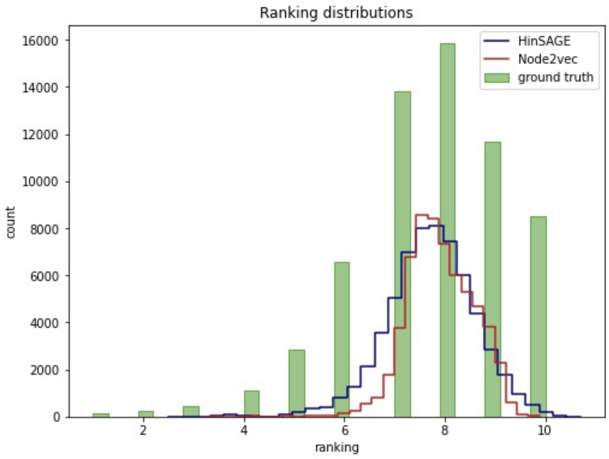


Figure 5: Final ranking distribution of HinSAGE model and Node2Vec compared with the ground truth.

5.1 Technical details

For computing the centrality metrics Networkx will be used. Instead StellarGraph (HinSAGE) is chosen for the GNN implementation. For Node2Vec, StellarGraph is used for biased random walk and then Word2Vec implementation by Gensim is applied.
For all the code used for this project, look at https://github.com/chguizz/graph_based_recommendation_system
Instead for checking all hyper-parameter tried for HinSAGE network, https://github.com/chguizz/graph_based_recommendation_system/blob/main/data/hyper-parameters.xlsx
Machine for experiments: Colab free available resources.

6 Conclusion

From the numerical results we can conclude that the proposed system seems to achieve a sufficient good level of performances. It is better than a simple base-line model that output always the mean rate and also more importantly it creates better embedding than Node2Vec. Nevertheless it is fundamental to note that HinSAGE as well as GraphSAGE, is especially useful for graphs that have rich node attribute information, while here there are only few attributes. Therefore a possible further experimentation will be to test this model when more information are available. Moreover for the centrality metrics computation also having more hardware resources can help in getting better results, indeed more powerful machines permit to apply the metrics mentioned before without approximation algorithms.

References

[1] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[2] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[3] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[4] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[5] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[6] Xiao Wang, Ruijia Wang, Chuan Shi, Guojie Song, and Qingyong Li. Multi-component graph convolutional collaborative filtering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6267–6274, 2020.

[7] Le Wu, Junwei Li, Peijie Sun, Richang Hong, Yong Ge, and Meng Wang. Diffnet++: A neural influence and interest diffusion network for social recommendation, 2021.

[8] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426, 2019.

[9] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 950–958, 2019.

[10] *Anime recommendations database*, 2021. https://www.kaggle.com/CooperUnion/anime-recommendations-database.

[11] Maurice George Kendall. Rank correlation methods. 1948.

[12] Ronald Fagin, Ravi Kumar, and Dakshinamurthi Sivakumar. Comparing top k lists. *SIAM Journal on discrete mathematics*, 17(1):134–160, 2003.

[13] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems (TOIS)*, 28(4):1–38, 2010.

[14] 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.

[15] David Eppstein Joseph Wang. Fast approximation of centrality. *Graph algorithms and applications*, 5(5):39, 2006.