Movie Recommendation using Graph Embedding with DeepWalk and Node2Vec

Xizhi Wu
School of Computing and
Information, University of Pittsburgh
320 N Neville st
4128053747, incl. +1
xiw183@pitt.edu

ABSTRACT

In the past year, random walk methods such as DeepWalk and Node2Vec have proven to be effective in retrieving node features representations in the network. The graph embedding learned from the user-item recommendation graph is an ideal input for deep neural networks. In this paper, we show how deepwalk and node2vec can be used to generate movie recommendations by learning user-movie graph embedding. We apply deepwalk and node2vec on a user-item bipartite graph built from the netflix prize data to generate recommendation in both implicit and explicit way. The result shows that deepwalk outperforms a set of collaborative filtering baselines with implicit feedback while node2vec only outperform deepwalk with explicit feedback and did not outperform baseline with explicit feedback.

Keywords

Recommender System; Feature Learning; Node Embedding; Graph Representations.

1. INTRODUCTION

With the growing demand for personalization and recommendations in commerce and business, recommender systems have been an indispensable tool to meet those needs. A good recommender system alleviates the user from the information overload in online resources, such as movie recommendation and product recommendation in online shopping. User's preferences are captured by examining their past interactions, including explicit feedback such as ratings, and implicit feedback such as clicks, reads and watches.

In the development of the recommender system, the recommendation system has evolved from neighborhood methods[1,2] to representation-learning-based frameworks[]. Neighborhood methods recommend items that are similar to historically interacted items while representation-learning-based approaches encode both users and items as continuous vectors in the same space. Representation methods have been proven to be superior to neighborhood methods since the Netflix Prize competition[3]. A lot of methods have been proposed to learn the representation of items and users, such as matrix factorization methods[2,4] and deep learning methods[5]. Nowadays, deep learning methods are the most widely used method because of its outstanding performance in both research and industry.

One of the deep learning methods in the recommender system is graph-learning-based methods, since the recommendation system has a graph structure of bipartite network with edges between user and item vertices. In this paper, we present a deep learning method that takes random walk embedding as input and output user preference under implicit and explicit settings.

2. Related Work

Recommender system recommends items based on the user's preference. There are two widely used approaches to tackle this problem, neighborhood methods and latent factors methods. There is a visualization of neighborhood methods and latent factor models in Figure 1. The former method is based on comparison of item similarity and the latter method is based on learning user/item interaction through implicit feedback or explicit feedback.

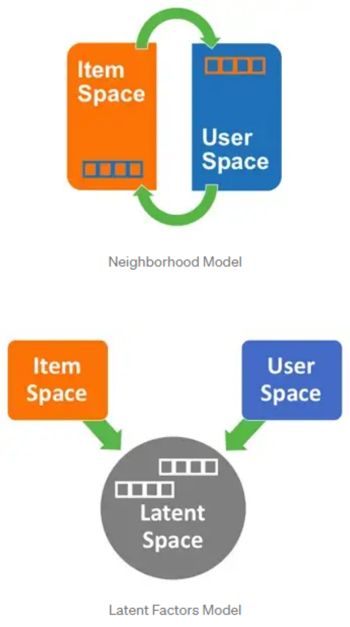

Figure 1. A visualization of Neighborhood Model and Latent Factor Model.

2.1 Neighborhood Methods

Neighborhood methods are concentrating on calculating the relationships between items or users.[6] Based on a certain user's rating on a precious item, the method aims to evaluate the preference of the user for a new item by computing the similarity between the new item and the previous item. In this way, users are regarded as a container of rated items. Item-to-item models are more popular and convenient in several aspects.

First, it is easier to capture the relationships between items than users. Because it is hard to explain the difference between users because a user is more complicated than an item and can be changeable. While item is a static variable and can be much easier to explain.

Secondly, usually in a recommending system, there are more users than items, and it is more expensive to model between users than it is to model between items.

2.2 Latent Factors Methods

Latent features capture a low-dimensional representation of users and items. Matrix factorization is widely used in latent factors models to find weighted low-rank approximations to the user-item matrix. SVD and SVD++[7] are typical matrix factorization methods that decompose the matrix into a user-factor matrix containing vectors p_u and an item-factors matrix containing vectors q_i. When applying SVD on a matrix, the matrix needs to be dense and the missing value needs to be imputed. The value of p_u and q_i is initialized as random numbers and then updated by minimizing the cost function using gradient descent.

2.3 Graph Learning Based Methods

Graph learning based methods processing the user-item information in a graph perspective. Compared to previous methods that used user-item interactions as collaborative signals, graph based methods can explicitly capture the graph structures. Such as in random walk methods, we can directly derive embedding for every node based on the appearances of nodes in walk sequences.In GNN methods[7, 8, 9, 10, 11], higher levels of feature can be explored to improve recommender systems. For example, in GCN model[10, 11], the eigendecomposition of the graph Laplacian to iteratively aggregate information from neighbors.

3. Dataset

Netflix Prize data is the dataset used in the Netflix Prize open competition for the best algorithm to predict user rating for movies. The dataset contains movie's information such as movie id, year of release and title, and user information such as user id, rating and date. In our experiment, we only need to make use of the movie id(ranging from 1 to 17770), user id(ranging from 1 to 2649429) and ratings(ranging from 1 to 5).

We created an explicit dataset with each column being index, movie id, user id and rating as the format in Table 1. This is used for explicit feedback. We also created an implicit feedback dataset(Table 2.) just by removing the last column "rating" of the explicit dataset.

Table 1. Explicit Dataset

index	movie id	user id	rating
0	m0	u0	5
1	m1	u1	4
…	…	…	…

Table 2. Explicit Dataset

index	movie id	user id
0	m0	u0
1	m1	u1
…	…	…

4. Method

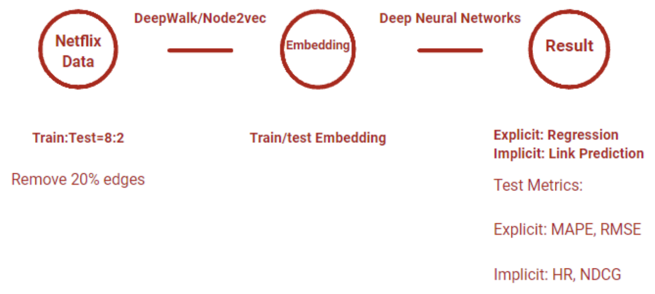

Figure 2. An Overview of the Model

First we need to discover the embedding of the User-Movie network using random walk methods, DeepWalk and Node2vec. Then we pass the generated embedding into our deep neural networks to get our prediction. A regression model is used in both explicit models and implicit models. In explicit models, the output is a rating ranging from 1 to 5. In implicit models, the output is a binary number, with 1 indicating a link exists between user and movie, and 0 is the opposite.

4.1 Train Test Split

We generated two graph embeddings, one is for training and the other is used for testing. The test embedding is generated from the network of users and items with 20% edges removed for testing. This network is referred as test graph. The train embedding is generated from the train graph. Compared to the test graph, it removes another 20% edges as a part of a training example from the test graph. The training examples contain 20% edges randomly selected from the test graph with the same number of generated negative examples. The testing set also contains the equal number of negative examples as the 20% edges removed for testing. All the nodes are still reachable after we remove those edges, so every node will have a distinct embedding vector.

In the training set, we did another 8:2 train test split for choosing computing operators on the movie vectors and user vectors from the train embedding. There are two operating operators, one is the L1 operator which is a L1 error between the two vectors. The other is the concatenation operator which is a concatenation between the user vector and the movie vector. We will pass the outcome of the operator computation on user vector and movie vector into our regression model.

4.2 Baseline Methods

Two baseline models were chosen, NeuMF[5] and SVD[7] with bias, for implicit model comparison and for explicit model comparison.

$$\hat{y}_{ui} = f(u, i|\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u^T \mathbf{q}_i = \sum_{k=1}^{K} p_{uk} q_{ik},$$

Figure 3. Image taken from SVD paper[7]

The Singular Value Decomposition(SVD) is a method commonly used in linear algebra for dimensionality reduction. It is adopted by machine learning for reducing the number of features in a dataset. In a matrix where each column represents an item and each row represents a user. p_u is the latent feature vector of the user and q_i is the latent feature vector of the movie. By calculating the dot product of p_u and q_i, we will get the predicted rating. By calculating the loss between predicted rating and true rating, we will be able to update p_u and q_i using stochastic gradient descent.

NeuMF include Multiple-layer Perceptron module apart from the Generalized Matrix Factorization layer. In the input layer, the user and item are one-hot encoded. Then they were mapped to the hidden space. The output of GMF and MLP modules are concatenated and then connected to the output layer.

4.3 DeepWalk Method[12]

Based on the assumption that adjacent nodes are similar and should have similar embeddings, deepwalk regards nodes appearing together in one walk to be similar. Two parameters of the deepwalk, k and l, correspond to the k number of random walks starting at each node and the length l of one path. Parameter k controls the portion of the network explored and parameter l controls how close the node is considered similar. If l is increasing, then a more distant node is considered similar.

Then the generated walks are fed into SkipGram model[], which is a technique used to learn word embeddings. Given a word sequence and a window size, SkipGram tries to maximize the conditional probability of words appearing in one window. Since words appear in the same window, SkipGram assumes that their embedding is similar. Similar assumptions can be made with the DeepWalk on graph, where nodes appearing in the same path should have similar embedding. In this way, we can get a deepwalk embedding of our user nodes and item nodes in the user-item graph.

4.4 Node2Vec Method[13]

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

Figure 4. Image taken from the Node2Vec paper[13]

Node2Vec is a biased random walk method. The above formula introduced in the node2vec paper defines the conditional probability of moving from node v to node x. If the graph is weighted, then it is ideal to add bias with weight. If the graph is unweighted, the author of node2vec introduces an unnormalized transition probability as in the Figure 5.

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Figure 5. Image taken from the Node2Vec paper[13]

p indicates the probability of random walk getting back to the visited node and q indicates the probability of random walk getting far away from the starting point. The generated random walks are also fed into the SkipGram model.

5. Evaluation results

5.1 Evaluation steps

After the user embeddings and movie embeddings are generated, they are first computed using the concatenation operator and L1 operator. The outcomes are then fed into two regression models with the same structure except the input layer. Because the outcome dimension of using L1 operator is the same as user and movie embedding, while the outcome dimension of concat operator is twice as the user and movie dimension.

Since we had another 8:2 train test split on the training set, the 20% of training set data is used for choosing L1 or concat operator. After this step, we will run the evaluation on the testing set. There are two stages of evaluations in each model, the first for choosing the operator and the second for evaluation on the testing set. In deepwalk models, concat outperform L1 greatly in prediction accuracy. In node2vec models, concat performance is very close to L1 performance in the first stage of evaluation, so both are kept for the second stage of evaluation.

5.1 Evaluation metrics

For implicit recommendation tasks, link prediction is used to predict user preference. In the testing of the implicit recommendation tasks, every testing example is combined with 99 negative examples. Then we try to find our true testing example's score among the 100 scores. If our testing example is among the top10 scores, then it's a hit and the NDCG value will be calculated and averaged at the end of testing.

For explicit recommendations tasks, we will calculate the MAPE and RMSE between predicted ratings and true ratings.

5.1 Testing results

Two baseline models, SVD with bias, are used to compare with implicit tasks and the NeuMF is used for comparing with explicit tasks. Among the homework2 models, the referenced model 3(R3) is the best performing model. The other model's performance is visualized as follows:

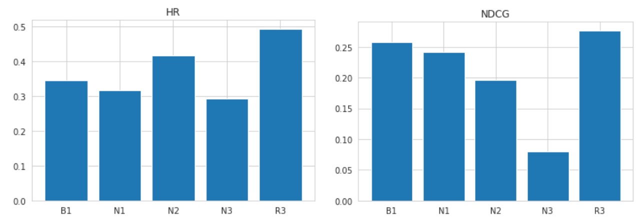

Figure 6. The bar chart on the left is the HR of the models in homework2 and the bar chart on the right is the NDCG value. I corrected the calculation of NDCG according to the evaluations.py[] of our homework2 repository, since my previous calculation of NDCG is higher than it is supposed to be.

The following three tables are the result table for baseline models, deepwalk models and node2vec models.

Table 3. Evaluation results With Baseline Models

	NeuMF		**SVD with Bias**
HR	0.494	MAPE	0.3042
NDCG	0.277	RMSE	0.99

Table 4. Evaluation results With DeepWalk

	Implicit Results		**Explicit Results**
	Concat		**Concat**
HR	0.4831	MAPE	0.4012
NDCG	0.4621	RMSE	1.2024

Table 5. Evaluation results With Node2Vec

	Implicit Results			Explicit Results	
	L1	**Concat**		**L1**	**Concat**
HR	0.4045	0.5146	MAPE	0.3404	0.3468
NDCG	0.2071	0.3337	RMSE	1.0716	1.1944

6. Discussion

Our random walks based method outperforms the best model in HW2 with implicit feedback. The best random walk based result with implicit feedback was the DeepWalk with concat operator with a HR of 0.48 and a NDCG of 0.4621, which outperform the NeuMF model.

However, our best model with explicit feedback did not outperform our baseline. The best random walk based result with explicit feedback is node2vec with L1 operator with a MAPE of 0.3404 and a RMSE of 1.0716, which did not outperform the SVD with bias.

The possible explanation would be that for implicit feedback, random walk based methods can better capture the graph structure compared to NeuMF, since NeuMF is essentially a Matrix Factorization method that first randomizes its p_u and q_i. The NeuMF then updates the p_u and q_i using stochastic gradient descent with a neural network. While in random walk based methods, user-vector and item-vector are already a low-level feature representation instead of a randomly generated sequence of numbers.

However, following this explanation, the explicit feedback result should also be better than the baseline. One reason could be that the regression model is not ideal for handling explicit ratings data. And if a classification model is used to predict the ratings from 1 to 5, it could have performed better. Another reason could be that I did not make use of the ratings as edge weights, instead I forced the explicit node2vec with preset hyperparameters p and q. However, for a network with edge weight, it is better to take a weighted random walk on the network, and the ratings are the perfect edge weights to better distinguish edges with different ratings and the nodes on the two ends of that edge. This could explain why deepwalk is performing better than node2vec in implicit feedback and why baseline SVD with bias is performing better than random walk based methods. Because under the current settings of the node2vec method, an edge with a rating of 1 is handled the same as an edge with a rating of 5.

Another limitation of the random walk based method is that every node has to be traversed in order to generate an embedding of the graph. So adding users or items to the recommender system will be very expensive and the entire embedding needs to be recalculated. Therefore, deepwalk and node2vec are not suitable for online learning of embedding and online recommendation.

Future work would be to generate a weighted random walk with explicit feedback using rating as edge weights. In this way we might be able to improve results for explicit feedback. More combinations of hyperparameter p and q should also be tested out in the future to better our models. And if that is being achieved, we would try to propose a way to resolve the online recommendation issues for deepwalk and node2vec methods. Since every walk has a fixed length so the farest reachable node can be estimated. And if we can update our walks for only a part of the whole network, then random walk methods can be used for online recommendation. But as far as one can see, this is still very expensive and might not outperform other SOTA methods such as GCN[10] and GraphSage[14].

7. Conclusion

In this paper, we generated embeddings for users and movies in our bipartite network and learned high level features by using a regression deep neural network. There are two types of data used, implicit and explicit feedback data. Then we compared our result to the two best models from homework1 and homework2. Our random walk based method outperforms our baseline implicit feedback model. For explicit feedback models, the baseline model is still better because we might not achieve the full potential of our node2vec method. But possible explanations are discussed for this scenario and possible future works are proposed.

8. Acknowledgments

First I would like to thank Dr. Lin for having me in her class, giving me instructions and inspiration on our projects and for opening this epic course. I did not have a comprehensive knowledge of graph data mining until I took this course. Now I can search for the most up to date research papers I'm interested in and be able to comprehend them and provide feedback on them! I will keep up with this and try to read an interesting paper or grab some interesting material every week in my future studies and research.

Then I would like to thank my friends in this class, Auru and Yijun, for answering my questions with patience and leaving interesting comments below my reflections.

I also would like to thank Ahana for sharing interesting thoughts and answering my questions. Even though we didn't meet her in class, I believe we would also be friends if we met her.

9. REFERENCES

[1] Bell, Robert M., and Yehuda Koren. "Scalable collaborative filtering with jointly derived neighborhood interpolation weights." Seventh IEEE international conference on data mining (ICDM 2007). IEEE, 2007.

[2] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009): 30-37.

[3] https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data

[4] Koren, Yehuda, Steffen Rendle, and Robert Bell. "Advances in collaborative filtering." Recommender systems handbook (2022): 91-142.

[5] He, Xiangnan, et al. "Neural collaborative filtering." Proceedings of the 26th international conference on world wide web. 2017.

[6] Linden, Greg, Brent Smith, and Jeremy York. "Amazon. com recommendations: Item-to-item collaborative filtering." IEEE Internet computing 7.1 (2003): 76-80.

[7] Koren, Yehuda. "Factorization meets the neighborhood: a multifaceted collaborative filtering model." Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. 2008.

[8] Kabbur, Santosh, Xia Ning, and George Karypis. "Fism: factored item similarity models for top-n recommender systems." Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. 2013.

[9] Wang, Xiang, et al. "Neural graph collaborative filtering." Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval. 2019.

[10] Ying, Rex, et al. "Graph convolutional neural networks for web-scale recommender systems." Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. 2018.

[11] He, Xiangnan, et al. "Lightgcn: Simplifying and powering graph convolution network for recommendation." Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval. 2020.

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

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

[14] Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in neural information processing systems 30 (2017).