Deep Social Recommendation with Implicit Feedback
Final Report

Qing Wang, Tianqi Wu, Hao Zhu
`{qwang55,twu38,haozhu3}@illinois.edu`
University of Illinois at Urbana-Champaign
Champaign, IL 61820

December 5, 2019

1 Introduction

As the amount of information available on the web grows rapidly in recent decade, recommender system becomes an inevitable approach to avoid providing users with contents that are not of their interest across a variety of web domains. Recommender systems with implicit feedback have attracted many studies recently as it is easier to collect implicit feedbacks from massive users such as "like" or "dislike" rather than specific ratings. However, as there is no rating information, it is harder to capture user's degree of interest over items and requires more advanced strategies. For review-based web platform such as Yelp, social network of users is easily accessible and can serve as an additional source to better capture user preference. For users with little or no feedback, there is a high chance that they will like the item if most of their friends like it. In this work, we explore the possibility of incorporating social network information with implicit feedbacks to enhance performance of existing recommender systems. We use Bayesian Personalized Ranking method for implicit feedback as our baseline model which is only based on user-item interactions. To incorporate social network with recommender system, we add social weights which represents the effect of friends on users. To further improve the model, we use *node2vec* [1] to initialize the weights of the embeddings. Our proposed model improves Recall@50 from 0.1368 to 0.2634 and NDCG@50 from 0.1102 to 0.1935. The results show that our model is able to outperform standard non-social recommender BPR.

2 Related Work

2.1 Bayesian Personalized Ranking

Methods like matrix factorization and adaptive knearest-neighbor are commonly used for item recommendation of implicit feedback. However, none of them are optimized for ranking. Bayesian personalized ranking [2] (BPR) is designed to improve the standard learning techniques. It uses maximum posterior estimator derived from a Bayesian analysis as the optimization criterion and stochastic gradient-descent algorithm based on bootstrap sampling as learning models. Unlike traditional methods, the training data consists of both positive and negative pairs and missing values. Also, the training data is for the actual objective of ranking. Whereas, the model relies on the quality of negative sampler and the uniform negative sampler may be inefficient for data sets with large number of items.

2.2 Socially Enabled Preference Learning from Implicit Feedback Data

This work [3] proposes a method to incorporate social network in recommender system with implicit feedback by adding friends' influence term, a weighted sum of user's friends' preferences, in objective function and optimizing it with block Gauss-Seidel process. The advantage is that such model learns user preference and friend importance simultaneously. The model outperforms recommender systems that use only user-item interaction and illustrates that it is beneficial to incorporate social network with recommender systems based on implicit feedback. However, the model only considers influence of a user's direct friends and sometimes a friend of friend will also share similar taste than a user and has positive influence in predicting a user's rating. A model that utilizes social information to deeper extent should further improve the model.

2.3 Leveraging Social Connections to Improve Personalized Ranking for Collaborative Filtering

The paper [4] proposes a method to estimate users' rankings of products by leveraging their social connections in a Social Bayesian Personalized Ranking model, specifically by adding a social coefficient in the objective function to show the preference difference between users' positive and social feedback, and further implementing stochastic gradient descent to optimize the objective function. In terms of advantages, the work solves the one-class recommendation problem (when we only have data from implicit feedback) by leveraging users' positive feedback and social feedback. Authors also comes up a solution for cold-start problem: using social feedback when we don't have enough information from users' positive preference. However, the model SBPR is limited to sparse data. When it comes to future improvement, rating information (explicit feedback) should be incorporated into the SBPR model.

3 Problem Definition

3.1 Recommendation with Implicit Feedback

A recommendation system with implicit feedback is a system that recommends items to users given existing user-item interactions without explicit rating. User-item interaction information can be encoded by an interaction matrix with dimensions "user" and "item". An entry holds value "1" if corresponding user and item have positive interaction and "0" otherwise. Different from recommendation with explicit feedback where we have both positive and negative ratings from user, implicit feedback only encodes positive information and it can be harder to make accurate recommendations as a result. A common approach for recommendation with implicit feedback is to find best approximation of interaction matrix using matrix factorization of two matrices representing user features and item features respectively. Our final model adopts matrix factorization as basic building block and we will discuss the details in section 4.1.

3.2 Social Recommendation

Besides user-item interaction information, user social information can also be beneficial to generate accurate recommendations as people tend to share similar interests with their friends. Social information can be viewed as a graph with nodes being all users. Two users are connected by an edge if they are friends to each other. Basic idea of social recommendation is to combine predicted user rating with predicted social influence during model training to make better recommendations.

4 Method

4.1 Matrix Factorization

Given user-item interaction matrix, we use matrix factorization with user and item representations and learn to best approximate it during training. We denote interaction matrix by X, user feature matrix by U and item feature matrix by I. We then obtain approximated ratings with:

$$\hat{X} = U \cdot I^T \tag{1}$$

where rows of U are user features and rows of I are item features. They have same number of columns, which is the number of hidden units defined during model training. Then, for user i with feature u_i and item k with feature i_k, the rating that user i will give to item k is approximated by:

$$\hat{x}_{user_{i,k}} = u_i \cdot i_k^T \tag{2}$$

Then, the task of building recommender system for implicit feedback becomes finding best representations of U and I that best approximates X. We denote the approximated user-item score matrix by $\hat{X}_{user}$.

4.2 Deep Social Information Incorporation

Using matrix factorization and appropriate optimization technique can already yield some good results as in the work by Rendle et al [2]. We can further enhance our approximation by incorporating user social information. As different friends have different degree of influence on a user, it is crucial to learn friend's weight as proposed in work by Matuszczyk et al[3]. We adopt their idea of assigning and training weights of friends of a user and further improves it by incorporating deeper social information. Our intuition is that if two users are not direct friends but have many mutual friends, then it is possible that they share similar interests to some extent. Yet previous model did not effectively encode this information.

Our assumption is that close friends have higher influence and distant friends have smaller influence on each other while all users can have an influence on other users to different extent. Thus, our social weight matrix has dimensions (n, n) where n denotes total number of users. Given social network of all users as an undirected graph, we first obtain social graph embedding G of each user from existing graph embedding algorithm *node2vec* so that we can compute social distance between users. Each row of G represents a user's features in social network. We then calculate cosine similarities between all user pairs and denote this information in matrix S where for user i and j:

$$S_{i,j} = CosineSim(G_i, G_j) \tag{3}$$

We then set all diagonal entries of S to 0 so that it does not capture user's rating at the beginning and use this matrix to initialize our social weight parameter for model training.

As different users can have different influence from friends, it is crucial to assign coefficients to user score and friends influence for different users as in the work by Zhao et al[4]. One advantage of our method is that we use diagonal entries of S to capture this information. Since diagonal entries can become non-zero during training, it has same functionality as assigning coefficients to user scores and social influence for different users.

We use following formula to obtain a user's friends' influence score $\hat{X}_{friends}$ where for user i and item k:

$$\hat{x}_{friends_{i,k}} = \sum_{j=0}^{n} \frac{S(i,j)}{n} \cdot u_j \cdot i_k^T \tag{4}$$

and the final score user i gives to item k is calculated as:

$$\hat{x}_{i,k} = \hat{x}_{user_{i,k}} + \hat{x}_{friends_{i,k}} \tag{5}$$

4.3 Optimization

We assume that the user would prefer observed items over all other items that are not observed. Also, we assume that all users act independently and the ordering of each pair of items are independent. Hence, our goal is to recommend user i with items that have higher score $\hat{x}_{i,k}$.

During the forward pass, we can obtain the predictions using equation 5. The loss of BPR [2] is essentially pairwise comparison between (user, observed item) and (user, unobserved item). For each user i, we use a positive item from training data denoted as pos_item to obtain positive prediction. . Then, we obtain a negative item for user i by randomly choosing items from the whole item set excluding the items that have interactions with the user, denoted as neg_item. We denoted predicted score of positive interaction as $\hat{x}_{i,pos\_item}$ and denote that of negative interaction as $\hat{x}_{i,neg\_item}$.

Finally, the BPR loss for each batch is calculated as follows:

$$BPR\_loss = \sum_{i=1}^{I} -log(\sigma(\hat{x}_{i,pos\_item} - \hat{x}_{i,neg\_item}))) \tag{6}$$

where σ is the logistic sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{7}$$

After obtaining loss, we can use Adam optimizer from Pytorch to do back-propagation of parameter weights.

4.4 Initialization

There are multiple ways to initialize user features U and item features I for matrix factorization such as Xavier or SVD. Since we are training with optimizer of gradient descent style, a good initialization prevents model from gradient explosion or stuck at local minimum. During our experiments, we find that validation performance of model with BPR loss and simple initialization like Xavier starts to drop after only 3-4 epochs of training. To solve this issue, our initialization aims to achieve best possible approximation of U and I with existing algorithm using only user-item interaction.

As user-item interactions can also be viewed as a bipartite graph with edges between users and items representing positive interactions, we use graph embedding of user and item nodes with *node2vec* to get a better initialization for our model. Model with such initialization achieves Recall@50 = 0.1054 and NDCG@50 = 0.0804 after first epoch, which is a better start than Xavier initialization with Recall@50 = 0.0239 and NDCG@50 = 0.0259 for first epoch. Model performance with graph embedding initialization also keeps increasing on validation set as number of epochs increase and yields much better result.

5 Result

5.1 Dataset

The baseline BPR model is trained and tested on Yelp dataset. There are 141623 user-restaurant interactions, 6858 unique users and 3317 unique restaurants in the train dataset. In addtion, there are 137977 user-user interactions in the social dataset. The task is to recommend restaurants for given user id. There is no user rating since we are focusing on implicit feedback. The train dataset is split into 80% of training set and 20% of validation set.

5.2 Evaluation Metric

The performance metric used is Recall@K and NDCG@K and the equations are provided:

$$Recall@K(u) = \frac{\sum_{k=1}^{K} rel(k)}{min(K, |I_u|)} \tag{8}$$

$$NDCG@K(u) = \sum_{k=1}^{K} \frac{rel(k)}{log_2(k+1)} \tag{9}$$

where I_u is the set of held-out items for user u and $rel(k)$ is indicator equaling 1 if item at rank k is relevant.

5.3 Experiments

We first train the baseline model with default parameters (embed_dim=64, batch_size=1024). Then, we try our best to tune the parameters (embed_dim=256, batch_size=256). Afterwards, we use *node2vec* initialization without social weight, denoted as **GEI** (graph embedding initialization). Finally, we add social weights to incorporate the social network information, denoted as **GEI+SW** (graph embedding initialization with social weight). The model is trained with around 50 epochs. For Adam optimizer, learning_rate is reduced from 0.0001 to 0.00001 after epoch 15 and weight_decay is reduced from 0.0001 to 0 after epoch 35. The experiments are done on Google Colab with GPU and the approximate training time of final model is 30 minutes. The results are illustrated in Table 1.

Metric	Baseline	Tuned Baseline	GEI	GEI+SW
Recall@50	0.1368	0.1433	0.2431	**0.2634** (+0.1266)
NDCG@50	0.1102	0.1167	0.1796	**0.1935** (+0.0833)

Table 1: Performance Comparison of different models

5.4 Results and Discussion

Recall@50 measures the fraction of relevant items retrieved correctly within top-50 ranks and NDCG@50 is discount factor to emphasize the importance of higher ranks. Initialization of weights improves Recall@50 and NDCG@50 by 0.1063 and 0.0694. It may prevent the model from stuck at local minimum. Tuning parameters, *node2vec* initialization and adding social weights to incorporate social network information altogether improve Recall@50 of 0.1266 and NDCG@50 of 0.0833. Our result show that adding social weights does improve the performance of the model. Social information really helps to make better recommendations in this case. Our proposed model is able to encode the social information by assigning weights which illustrates the user-user interactions. Hence, we can outperform the standard non-social baseline model.

5.5 Future Work

We also attempt to utilize item categories. However, the performance is not improved with our methods. We suspect that information of item categories may not be that useful for this particular case. In future, we may try to make better use of information from item categories. Also, there are other methods to encode the social information that can be explored. And hopefully, it would improve the performance of current model.

References

[1] Grover Aditya and Leskovec Jure. node2vec: Scalable feature learning for networks. *KDD '16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 855-864*, 2016.

[2] Steffen Rendle et al. Bpr: Bayesian personalized ranking from implicit feedback. *Conference on Uncertainty in Artificial Intelligence, pages 452–461*, 2009.

[3] Tomasz Matuszczyk Stephane Canu Julien Delporte, Alexandros Karatzoglou. Socially enabled preference learning from implicit feedback data. *ECML PKDD 2013*, 2013.

[4] Tong Zhao et al. Leveraging social connections to improve personalized ranking for collaborative filtering. *CIKM '14 Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management Pages 261-270*, 2014.