A Probabilistic Graph-Based Method to Improve Recommender System Accuracy

Nima Joorabloo[1](✉), Mahdi Jalili[1], and Yongli Ren[2]

[1] School of Engineering, RMIT University, Melbourne, VIC, Australia
S3624411@student.rmit.edu.au,
mahdi.jalili@rmit.edu.au
[2] School of Science, RMIT University, Melbourne, VIC, Australia
yongli.ren@rmit.edu.au

Abstract. The last two decades have seen a surge of data on the Web which causes overwhelming users with huge amount of information. Recommender systems (RSs) help users to efficiently find desirable items among a pool of items. RSs often rely on collaborating filtering (CF), where history of transactions are analyzed in order to recommend items. High accuracy, and low time and implementation complexity are most important factors for evaluating the performance algorithms which current methods have the shortage of all or some of them. In this paper, a probabilistic graph-based recommender system (PGB) is proposed based on graph theory and Markov chain with improved accuracy and low complexity. In the proposed method, selecting each item for recommendation is conditioned by considering recommended items in the previous steps. This approach uses a probabilistic model to consider the items which are likely to be preferred by users in the future. Experimental results performed on two real-world datasets including Movielens and Jester, demonstrate that the proposed method significantly outperforms several traditional and state-of-the-art recommender systems.

Keywords: Recommender system · Collaborative filtering · Markov chain

1 Introduction

Today, one of the major problems of the online shops is that users are often confused when deciding what to choose among a huge number of items. RSs have been proposed in order to help users to find the most suitable item according to their preferences [1]. Generally, RSs are classified into content-based methods (CB), collaborative filtering (CF), and hybrid methods. In CB approaches, the system recommends items based on available content information on the users and items [2]. CF is a widely used approach in RSs which focuses on similarity values between the users (or items). CF uses an information filtering technique based on the user's previous rating/purchase history to offer items which are aligned with the taste of the target user [3]. Hybrid RSs combine CF and CB approach to obtain improved performance [4]. The existing research papers in the field of RSs have mainly considered movie recommendation topic [5, 6]. There is also a rich literature on other topics, such as e-commerce [7], books [8], documents [9],

J. Macintyre et al. (Eds.): EANN 2019, CCIS 1000, pp. 151–163, 2019.
https://doi.org/10.1007/978-3-030-20257-6_13

music [10], television programs [11], applications in markets [12], e-learning [13], and Web search [14]. There are various metrics in the literature for evaluating the performance of RS algorithms [15].

Accuracy is one of the most important evaluation metrics, and most of studies in RSs evaluation criteria have focused on the accuracy [16]. Evaluation of RSs can be performed in an offline or online manner [17]. In an offline analysis, part of ratings in the dataset are hidden from the recommender algorithm as a test set and the RS algorithm uses the rest of data (training set) to predict new ratings or rank for unseen items. Offline evaluation methods are fast, but we cannot elicit the real taste of users regarding the recommended items. Likewise, online evaluations are conducted in a live environment to observe users' behavior and tracking their acts [18]. In addition, in comparison with offline methods, conducting an online evaluation, if possible, is more costly and time consuming, and this leads most of the research works to offline evaluation. Over the last decade, many research studies have focused on proposing new approaches to improve performance of recommenders. Although paying attention to existing evaluation measures are important to have a good RS, to implement RS in real world, we have to take into the account some considerations, such as simplicity of implementation and reasonable run time.

In this paper we propose an accurate probabilistic recommendation system based on graph theory to generate accurate recommendations with reasonable run-time in comparison with some traditional and state-of-the-art algorithms. PGB algorithm transforms ratings in the first step to *like* and *dislike*, and by so doing, it helps us to apply ratings on traditional Markov model and makes calculations simpler. Also, PGB lets us to model history of the ratings in a comprehensive and compact graph (system-state graph) using Markov model idea that helps us to recommend items without referring to raw ratings anymore; PGB generates system-state graph only one time at the start of algorithm, and then use it to recommend items. In the next step, we make decision based on system-state graph and travers through it for each user to find most probable items which will be liked by him/her in the future. One of the most significant advantages of PGB is its flexibility against updating dataset. In case of adding new ratings, system-state graph can be modified only by updating respective weights and it doesn't need to make it from scratch; Thus, proposed method is suitable for systems that have a lot of change in a short period of time.

2 Related Works

The last two decades witnessed much attention in network science, where graph theory and data mining meet [19]. A number of applications have been proposed to model behaviour of users with graph theory or related theories [20, 21]. Several works have applied Markov models in the context of RSs, where a Markov chain model makes recommendations based on previous actions. Rendle et al. proposed Factorized Personalized Markov Chains (FPMC) to combines Matrix Factorization and Markov Chain to model personalized sequential behavior [22]. His method improved by Cheng et al. by changing factorizing transition matrix into two latent and low-rank submatrices [23]. Shani et al. modelled the RS using Markov decision processes

(MDP) [24]. In [25] a context-aware approach to query suggestion were proposed by He et al. Sahoo et al. proposed a new collaborative filtering algorithm based on Hidden Markov Model that outperformed traditional CF techniques, especially when consumers' preferences are changing [26]. A graph-based algorithm was introduced by Yang et al. to first discover the topics of interest for each user, and then make a topic-aware Markov model to learn the navigation patterns for each user [27]. Although Markov model has been used in many real-world applications, it has some restrictions in RS field, namely it is affected by sparsity and neglecting users' ratings on items. Most of the proposed algorithms only consider the sequence of purchase/rating [6]. Ratings can be so important for getting more accurate result. On the other hand, some of Markov-based algorithms consider the probability of transition between states based on users' history. In this work, we propose a method that considers the probability of selecting items by the target user in future.

3 Probabilistic Graph-Based Recommendation Method

In this section, we first explain Markov model as the base idea for the proposed method, and then discuss details of the proposed method. The proposed method includes two main steps: (*i*) transforming the ratings and creating system-state graph using Markov chain to model users' ratings history, and (*ii*) applying a probabilistic model on the generated graph to recommend items.

3.1 Markov Models

Traditional recommenders like CF constructs the recommendation list based on the preferences of a group of users that are similar to the active user, but Markov chain considers information about ratings' sequence. Let's consider we have a dataset of ratings with a set of users U and set of items I. $S_u = <i_1, i_2, \ldots, i_m>$ represents state of use u, which denotes that target user u has rated m items in a sequential manner. Our goal is to predict $i_{m+1}, \ldots, i_{m+k}$ that are likely to be preferred by the user in the future, where k is the number of the recommended items. System-state graph is created by Markov model where each item can be assumed as a node of a graph and the sequence of ratings can be modelled as edges between them. To predict the probability of purchasing item i_{m+1} by a user who have already purchased $i_1, i_2, \ldots, i_m$ in the past, we need to define a transition function. In fact, this function counts the frequency of $<i_1, i_2, \ldots, i_m>$ and $<i_1, i_2, \ldots, i_m, i_{m+1}>$ sequences in the dataset to obtain the probability of changing $<i_1, i_2, \ldots, i_m>$ to $<i_1, i_2, \ldots, i_m, i_{m+1}>$ sequence which is obtained from following equation.

$$TF(<i_1, i_2, \ldots, i_m>, <i_1, i_2, \ldots, i_m, i_{m+1}>) = \frac{N(<i_1, i_2, \ldots, i_m, i_{m+1}>)}{N(<i_1, i_2, \ldots, i_m>)}, \quad (1)$$

where $N(<i_1, i_2, \ldots, i_m, i_{m+1}>)$ is the number of users who have this sequence in their ratings' history. We use this idea to propose our aggregated system-state graph.

3.2 Generating System-State Graph

We define the system-state graph different from classic Markov model. First, we define a threshold T to transform users' ratings to *like* if the rating is higher than or equal to T, and *dislike* if the rating is lower than T. This transformation helps us to apply the effects of the ratings in Markov model, simplify the problem and decrease the amount of calculations. With this assumption, items can have only two states: *like* and *dislike*, denoted by $s_{i,l}$ and $s_{i,d}$, respectively. We model the ratings as a graph with $s_{i,l}$ and $s_{i,d}$ being the nodes and co-occurrence of items in the users' states S_u the edges. Let's denote this graph by G, which is an undirected graph. Assume that the ratings dataset contains N items, then G has $2N$ nodes due to having a *like* node and a *dislike* node for each item. The weight of connection between two nodes $s_{i,l}$ and $s_{j,d}$ is defined as the number of users who *like* item i and at the same time *dislike* item j. Figure 1 shows an example how the ratings are transformed.

User	Item	Rate
u_1	i_1	1
	i_2	2
	i_3	5
	i_6	5
u_2	i_1	3
	i_2	3
	i_6	5
	i_8	1

Transforming with T=2.5

User	Item	Rate
u_1	i_1	dislike
	i_2	dislike
	i_3	like
	i_6	like
u_2	i_1	like
	i_2	like
	i_6	like
	i_8	dislike

Fig. 1. Transforming user rates to *like* and *dislike*

In the Next step, we extract two subgraphs, G_l and G_d from G that only contain relation between *like* and *dislike* nodes, respectively. Both G_l and G_d graphs show the correlation of items from different aspects; G_L shows the similarity of two items based on the number of likes they received together while G_d show the similarity of items based on dislike they received together. For example, in Fig. 1, user u_1 dislikes i_1 and i_2, and user u_2 likes i_1 and i_2. It shows i_1 and i_2 have a similar behaviour and get like or get dislike at same time in users' ratings history.

Since G_l and G_d have similar concepts, we merge them and make a new graph that shows the correlation between all items based on users' rating history. users' rating history; this graph is denoted by G_{ld}. To merge these graphs, we unify $s_{i,l}$ and $s_{i,d}$ as a single node and aggregate their edges along with their weights. The resulted graph is the system-state graph, which items in dataset and their correlations make its nodes and weights, respectively. Ultimately, G_{ld} is used for the recommendation purpose. In the next step, we traverse in G_{ld} to find items that are likely to be rated as *like* for the target user in the future. Figure 2 shows the process of generating G_l, G_d and G_{ld} according to

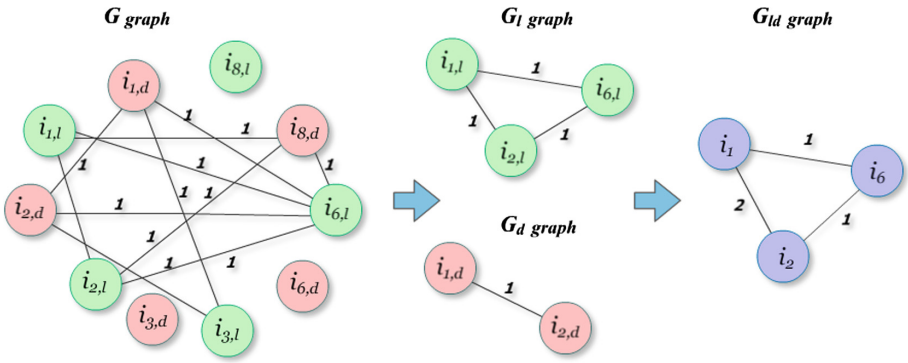

Fig. 2. Generating G graph for users in Fig. 1

dataset in Fig. 1. There are some differences between our proposed method and classic Markov model in creating system-state graph: (*i*) we consider ratings in system-state graph, (*ii*) we don't consider the sequence of ratings (*iii*) we ignore part of unimportant relations when extract G_l and G_d from G to make graph compact.

3.3 Recommendation

If S_{l_u} represents the set of items that are rated as *like* by u, the aim is to find k items with strongest correlation with S_{l_u} in G_{ld}, and recommend them to u. The main idea is that if we find recommendations based on the users' rating history, we can assume that users are likely to be like this recommendation. Then, we add this recommendation to S_{l_u} to consider it as rating history of the target user. In other words, each time we recommend a new item, we update S_{l_u} for target user u with the items recommended until that step. Suppose that *Rec* is a function that generate the recommendation list $R_u = <r_1, \ldots r_k>$ where k is the number of recommendations; the mth recommendation r_{m} for target user u is obtained as follow:

$$r_{\mathrm{m}} = Rec(\overline{S_{l_u}}), \tag{2}$$

where $\overline{S_{l_u}}$ is updated S_{l_u} which is obtained from union of S_{l_u} and previous recommended items as follow:

$$\overline{S_{l_u}} = S_{l_u} \cup R_u^{m-1}, \tag{3}$$

where R_u^{m-1} is the situation of R_u after adding the $(m-1)$th recommendation. The pseudo-code of the above function is given in Algorithm 1.

```
Input: S_{l_u}, G_{ld}, k //user-state, system-state graph, number of recommendations
Output: R_u // final recommendation list
NI_u = {}, NW_u = {}, R_u = {}// Initializing variables
S̄_{l_u} = S_{l_u} ∪ R_u^0
For m = 1:k
For each item j in S̄_{l_u}
    max_j = connected item with highest weight to item j in G_{ld}
    Add max_j to NI_u set
    Add weight between j and max_j to NW_u set
  End for;
remove duplicate item in NI_u and Aggregate related weights in NW_u
 r_m = find item with maximum weight in NI_u
 R_u^m = R_u^{m-1} ∪ r_m
 S̄_{l_u} = S_{l_u} ∪ R_u^m
End for;
```

Algorithm. 1. Pseudo-code of the proposed *Rec* function

To recommend item r_1 to target user u, our *Rec* function finds S_{l_u} items in G_{ld} and then find connected nodes with the highest weights to them; We denote these items and their related weights by NI_u and NW_u, respectively. Since we may have repetitive nodes in NI_u, *Rec* removes the duplicates in NI_u and aggregates the related weights in NW_u. The items present in NI_u have the strongest correlation with S_{l_u} items. In other words, weights in NW_u show the probability of occurring NI_u and S_{l_u} items together in the ratings history. We select the item with the highest weight in NI_u as the first item for the recommendation. In fact, we assume all S_{l_u} items as a single node in graph and select the node in G_{ld} that has the highest correlation with S_{u_l}; Fig. 3 shows how r_1 and r_2 are selected. Figure 3(*a*) shows G_{ld} where purple nodes and the red line around them shows S_{l_u}. In Fig. 3(*b*), the algorithm finds connected nodes to S_{l_u} items with highest weights which are depicted with red colour. In (*c*), i_5 with the highest aggregated weight is selected as a first recommendation and added to S_{l_u} to update it for next round of recommendation. For recommending the next item, we assume that u like r_1, and then find r_2 based on this assumption. This means that we select r_2 only if r_1 is preferred by the target user. To this end, we add r_1 to S_{u_l} and repeat the same process with updated S_{u_l}. Figure 3(*c, d*) shows the process of selecting r_2. The proposed approach reveals hidden correlations between items in system-state graph and helps users to find more neighbours when they have few numbers of items in S_{u_l}, which ultimately leads to have better precision in spars datasets.

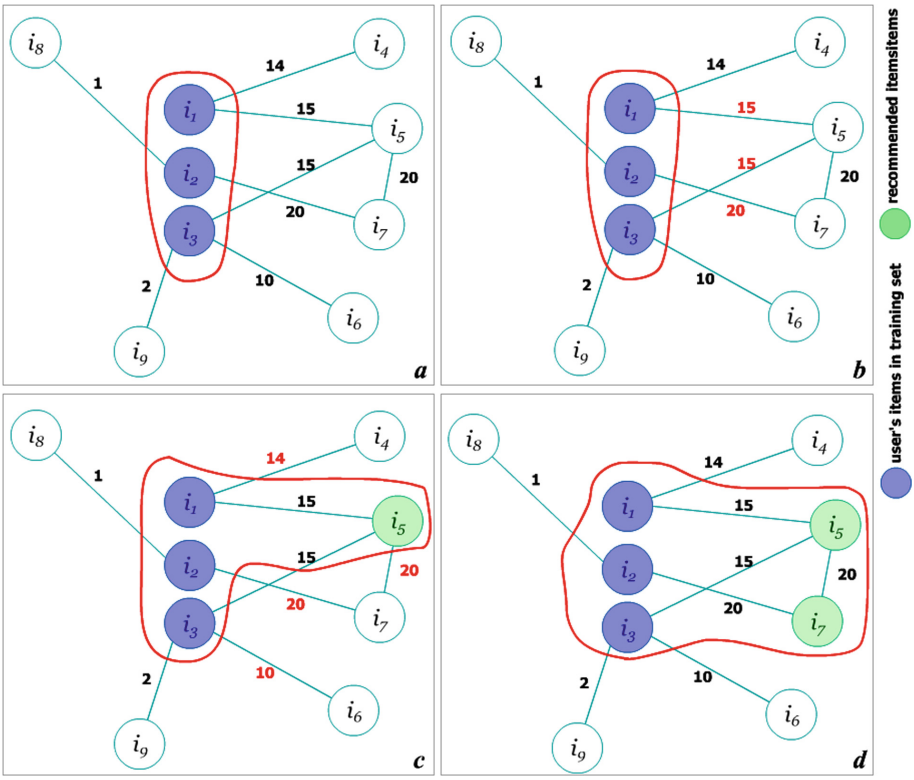

Fig. 3. Steps to obtain recommendation from system-state graph. (*a*) initial state of the target user in ratings dataset. (*b*) finding items with highest connected weight for each item in S_{u_l}. (*c*) select i_5 as first recommended item and add it to initial set. (*d*) repeat *b* and *c* process based on updated S_{u_l} and selecting i_7 as second recommendation. (Color figure online)

To make the approach clearer, imagine that $S_{u_l} = <i_1, i_2, i_3>$ for target user u. G_{ld} and the process of selecting the first and second recommendation items are depicted in Fig. 3. The strongest connection between i_1, i_2 and i_3 with other nodes in G_{ld} are i_5, i_7 and i_5 respectively. Figure 4 shows the process of creating NI_u and NW_u based on G_{ld}. After aggregating weights, the item with the highest weight is selected as the recommendation, which is I_5 in this example.

S_{u_l}	NI_u	NW_u
i_1	i_5	15
i_2	i_7	20
i_3	i_5	15

Aggregating weights

S_{u_l}	NI_u	NW_u
i_1, i_3	i_5	30
i_2	i_7	20

Fig. 4. Crating NI_u and NW_u and get the recommendation after aggregating weights

For top-k recommendation problem, we repeat the above introduced process for k times, which is a simple process. Markov model often struggles with some problems, such as determining the size of state and sparsity of the dataset, which is often the case for many real datasets. Markov model only considers the previously rated items in the recommendation process and ignores valuable users' rating information. Small chain number can make another problem for Markov model, as small chains cannot accurately represent the taste of users. The proposed method aims at solving these problems, and our experiments in the next section reveals its effectiveness to the state-of-the-art recommendation methods.

4 Experimental Results

In this section we compare the proposed algorithm with a number of classical and state-of-the-art algorithms. Since our algorithm is a ranking method, we use the evaluation metrics, which are proper for this type of methods.

4.1 Evaluation Metrics

Precision
$P_u(N)$ is precision for a list of recommended items to user u and is defined as the percentage of relevant items to user u in their list of recommendation. Relevant items to target user are those rated as *like* by the user. Precision of a system with N users, $P(N)$, is calculated as:

$$P(N) = \frac{\sum_{u \in testSet} P_u(N)}{N} \tag{4}$$

Recall
Recall is among the most frequently used metrics of information retrieval field. Recall for a target user u *is denoted by* $Recall_u(N)$ which is the proportion of relevant items to all items and Recall of a system with N users, $Recall(N)$, is calculated as:

$$Recall(N) = \frac{\sum_{u \in testSet} Recall_u(N)}{N} \tag{5}$$

Normalized Discounted Cumulative Gain (NDCG)
Discounted cumulative gain (DCG) measures the ranking quality of the recommended items. DCG increases when relevant items are placed higher in the list. It is obtained as follows:

$$DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2(i+1)} \tag{6}$$

where R is the recommendation list and rel_i shows the relative item in position i, which can be zero or one if a relevant item recommended in *ith* position in the recommended list or an irrelevant item is placed there, respectively. Normalized DCG (NDCG) is determined by calculating DCG and dividing it by the ideal DCG in which the recommended items are perfectly ranked:

$$IDCG = 1 + \sum_{i=2}^{|S_i|} \frac{1}{\log_2(i+1)} \tag{7}$$

F1 SCORE

Since precision and recall are inversely correlated, it is needed to consider both of them when evaluating different algorithms. Since precision and recall are dependent on the number of recommended items, researchers have often used $F1$ score, as a combination of precision and recall. $F1$ is calculated as follow:

$$F1 = \frac{2 * P(N) * Recall(N)}{P(N) + Recall(N)} \tag{8}$$

4.2 Datasets

In this paper, we employ two well-known datasets, including Movielens-100K and Jester, to evaluate performance of our method. The density of Jester dataset is about 10 times more than Movielens dataset that helps us to compare the performance of the proposed algorithm in spars and dense datasets. Movielens-100 K is a movie dataset with 943 users, 1682 items and 100,000 ratings. Jester is ratings of users to set of jokes. In this work we use a sample of the original dataset with 3000 users, 100 jokes and 165,536 ratings. The ratings in Movielens and Jester are on a scale of 1 to 5 and −10 to +10, respectively. For all benchmarks we use the same train and test sets for recommending 10 items. Threshold T to transform data to *like* and *dislike* is set to 2.5 for Movielens dataset and 0 for Jester dataset.

4.3 Results

In order to generate the result for comparison, we have used the Librec library in Java. The proposed method is developed in Matlab and compared with AspectModel [28], BPoissMF [29], EALS [17], ListRankMF [30], RankSGD [31], RankALS [32], WBPR [33], CLIMF [34], UserKNN, BUCM [35], ItemKNN, IMULT [36], and GRAD [37].

The result in Tables 1 and 2 report the performance of the algorithms in terms of different evaluation metrics over Movielens and Jester datasets, respectively. The proposed algorithm performs better than other algorithms in terms of precision, recall, NDCG and F1 evaluation metrics in both datasets. While it is the fastest algorithm in Jester, it has the fourth fastest runtime in Movielens, where AspectModel and ListRankMF are the fastest, and the second fastest algorithms, respectively. The performance of other algorithms differs across the datasets. While UserKNN is the second top-performer in Movielense (after the proposed algorithm), in the other dataset,

Table 1. Performance of algorithms on Movielens dataset. The best result for each metric is shown in boldface, while the second best result is shown in underlined boldface.

	Precision	Recall	NDCG	F1	Time(ms)
GBP	**0.3478**	**0.1475**	**0.3949**	**0.207149**	9500
AspectModel	0.228862	0.091245	0.247767	0.130473	**3148**
BPoissMF	0.019919	0.006309	0.015464	0.009583	16172
EALS	0.174187	0.07951	0.187264	0.109182	64966
ListRankMF	0.10122	0.047671	0.108673	0.064816	**4282**
RankSGD	0.261179	0.11386	0.292407	0.158586	13282
RankALS	0.175203	0.070117	0.189135	0.100153	572904
WBPR	0.14939	0.06305	0.152679	0.088674	104072
CLIMF	0.004065	0.00849	0.003427	0.001465	3696650
UserKNN	**0.305691**	**0.129918**	**0.33227**	**0.182341**	21259
BUCM	0.057927	0.020219	0.05347	0.029976	6368
ItemKNN	0.030081	0.012945	0.029488	0.0181	22388
IMULT	0.1842	0.0657	0.2068	0.096854	3402569
GRAD	0.0774	0.0429	0.0914	0.055203	19302

Table 2. Performance of algorithms on Jester dataset. The best result for each metric is shown in boldface, while the second best result is shown in underlined boldface.

	Precision	Recall	NDCG	F1	Time(ms)
GBP	**0.7331**	**0.7443**	**0.823**	**0.738658**	**11880**
AspectModel	**0.553032**	**0.527585**	0.628423	**0.540009**	24522
BPoissMF	0.186442	0.174734	0.214315	0.180398	32608
EALS	0.238501	0.217747	0.265346	0.227652	47041
ListRankMF	0.36413	0.341146	0.433668	0.352264	15783
RankSGD	0.381922	0.366335	0.445621	0.373967	**13709**
RankALS	0.323398	0.309093	0.384412	0.316084	103244
WBPR	0.406522	0.375251	0.43215	0.390261	71296
CLIMF	0.115103	0.103977	0.085998	0.109257	1602445
UserKNN	0.446568	0.41683	0.482625	0.431187	26096
BUCM	0.307723	0.288875	0.343151	0.298002	14850
ItemKNN	0.136041	0.123463	0.127869	0.129447	36769
IMULT	0.562	0.4438	0.6301	0.495955	985245
GRAD	0.5133	0.4081	**0.6318**	0.454694	24834

AspectModel is the second top-performer in terms of precision, recall and F1 and GRAD has the second best performance for NDGC. The result shows that proposed algorithm is not so sensitive about increasing number of users and by increasing 300% in number of users, time of recommendation increases only 20%. In addition, while most of algorithms have a big change in evaluation ranking by changing the dataset, GBP almost shows dataset-independent behaviour in comparison with other algorithms.

5 Conclusion

In this paper, we introduced a probabilistic graph-based method to obtain accurate recommender systems. The proposed method that called PGB, uses classic Markov model idea to makes a system-state graph based on users' ratings history, and then traverses the graph to predict items which are likely to be preferred by uses in the future. Selecting each item for recommendation is conditioned by considering recommended items in the previous steps. This approach uses a probabilistic model to consider the items which are likely to be preferred by users in the future. Experimental results performed on two real-world datasets including Movielens and Jester, demonstrate that the proposed method significantly outperforms several traditional and state-of-the-art recommender systems in terms of precision, recall, NDCG and F1.

References

1. Quan, T.K., Fuyuki, I., Shinichi, H.: Improving accuracy of recommender system by clustering items based on stability of user similarity. In: CIMCA 2006. IEEE (2006)
2. Pazzani, M.J., Billsus, D.: Content-based recommendation systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web. LNCS, vol. 4321, pp. 325–341. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72079-9_10
3. Bobadilla, J., et al.: Recommender systems survey. Knowl.-Based Syst. **46**, 109–132 (2013)
4. Burke, R.: Hybrid recommender systems: survey and experiments. User Model. User-Adap. Inter. **12**(4), 331–370 (2002)
5. Winoto, P., Tang, T.Y.: The role of user mood in movie recommendations. Expert Syst. Appl. **37**(8), 6086–6092 (2010)
6. Javari, A., Jalili, M.: A probabilistic model to resolve diversity–accuracy challenge of recommendation systems. Knowl. Inf. Syst. **44**(3), 609–627 (2015)
7. Castro-Schez, J.J., et al.: A highly adaptive recommender system based on fuzzy logic for B2C e-commerce portals. Expert Syst. Appl. **38**(3), 2441–2454 (2011)
8. Núñez-Valdéz, E.R., et al.: Implicit feedback techniques on recommender systems applied to electronic books. Comput. Hum. Behav. **28**(4), 1186–1193 (2012)
9. Porcel, C., et al.: A hybrid recommender system for the selective dissemination of research resources in a technology transfer office. Inf. Sci. **184**(1), 1–19 (2012)
10. Tan, S., et al.: Using rich social media information for music recommendation via hypergraph model. ACM Trans. Multimedia Comput. Commun. Appl. (TOMM) **7**(1), 22 (2011)
11. Barragáns-Martínez, A.B., et al.: A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition. Inf. Sci. **180**(22), 4290–4311 (2010)
12. Costa-Montenegro, E., Barragáns-Martínez, A.B., Rey-López, M.: Which App? A recommender system of applications in markets: implementation of the service for monitoring users' interaction. Expert Syst. Appl. **39**(10), 9367–9375 (2012)
13. Bobadilla, J., Serradilla, F., Hernando, A.: Collaborative filtering adapted to recommender systems of e-learning. Knowl.-Based Syst. **22**(4), 261–265 (2009)
14. McNally, K., et al.: A case study of collaboration and reputation in social web search. ACM Trans. Intell. Syst. Technol. (TIST) **3**(1), 4 (2011)

15. Jalili, M., et al.: Evaluating collaborative filtering recommender algorithms: a survey. IEEE Access **6**, 74003–74024 (2018)
16. Li, X., Wang, H., Yan, X.: Accurate recommendation based on opinion mining. In: Sun, H., Yang, C.-Y., Lin, C.-W., Pan, J.-S., Snasel, V., Abraham, A. (eds.) Genetic and Evolutionary Computing. AISC, vol. 329, pp. 399–408. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-12286-1_41
17. He, X., et al.: Fast matrix factorization for online recommendation with implicit feedback. In: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. ACM (2016)
18. Santos, B.S., et al.: Integrating user studies into computer graphics-related courses. IEEE Comput. Graph. Appl. **31**(5), 14–17 (2011)
19. Deo, N.: Graph Theory with Applications to Engineering and Computer Science. Courier Dover Publications, Mineola (2017)
20. Augustyniak, P., Ślusarczyk, G.: Graph-based representation of behavior in detection and prediction of daily living activities. Comput. Biol. Med. **95**, 261–270 (2018)
21. Mobasher, B.: Data mining for web personalization. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web. LNCS, vol. 4321, pp. 90–135. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72079-9_3
22. Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Factorizing personalized Markov chains for next-basket recommendation. In: Proceedings of the 19th International Conference on World Wide Web. ACM (2010)
23. Cheng, C., et al.: Where you like to go next: successive point-of-interest recommendation. In: Twenty-Third International Joint Conference on Artificial Intelligence (2013)
24. Shani, G., Heckerman, D., Brafman, R.I.: An MDP-based recommender system. J. Mach. Learn. Res. **6**(Sep), 1265–1295 (2005)
25. He, Q., et al.: Web query recommendation via sequential query prediction. In: 2009 IEEE 25th International Conference on Data Engineering. IEEE (2009)
26. Sahoo, N., Singh, P.V., Mukhopadhyay, T.: A hidden Markov model for collaborative filtering. In: Management Information Systems Quarterly, Forthcoming (2010)
27. Yang, Q., et al.: Personalizing web page recommendation via collaborative filtering and topic-aware markov model. In: 2010 IEEE International Conference on Data Mining. IEEE (2010)
28. Hofmann, T., Puzicha, J.: Latent class models for collaborative filtering. In: IJCAI (1999)
29. Salakhutdinov, R., Mnih, A.: Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In: Proceedings of the 25th International Conference on Machine Learning. ACM (2008)
30. Shi, Y., Larson, M., Hanjalic, A.: List-wise learning to rank with matrix factorization for collaborative filtering. In: Proceedings of the Fourth ACM Conference on Recommender Systems. ACM (2010)
31. Töscher, A., Jahrer, M.: Collaborative filtering ensemble for ranking. J. Mach. Learn. Res. W&CP **18**, 61–74 (2012)
32. Takács, G., Tikk, D.: Alternating least squares for personalized ranking. In: Proceedings of the Sixth ACM Conference on Recommender Systems. ACM (2012)
33. Gantner, Z., et al.: Personalized ranking for non-uniformly sampled items. In: Proceedings of KDD Cup 2011 (2012)
34. Shi, Y., et al.: CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In: Proceedings of the Sixth ACM Conference on Recommender Systems. ACM (2012)

35. Barbieri, N., et al.: Modeling item selection and relevance for accurate recommendations: a Bayesian approach. In: Proceedings of the Fifth ACM Conference on Recommender Systems. ACM (2011)
36. Ranjbar, M., et al.: An imputation-based matrix factorization method for improving accuracy of collaborative filtering systems. Eng. Appl. Artif. Intell. **46**, 58–66 (2015)
37. Lin, C.-J.: Projected gradient methods for nonnegative matrix factorization. Neural Comput. **19**(10), 2756–2779 (2007)