An Effective FP-Tree-Based Movie Recommender System

Sam Quoc Tuan, Nguyen Thi Thanh Sang(✉), and Dao Tran Hoang Chau

School of Computer Science and Engineering, International University – Vietnam National University, Ho Chi Minh City, Vietnam
itiu09030@gmail.com, {nttsang,dthchau}@hcmiu.edu.vn

Abstract. Movie recommender systems play an important role in introducing users to the most interesting movies efficiently. It is useful for users to find what they want in a large numerous of various movies on the Web quickly. The performance of movie recommendation is influenced by many factors, such as user behavior, user ratings. Therefore, the aim of this study is to mine datasets of user ratings and user behaviors in order to recommend the most suitable movies to active users. User behaviors are sequences of users' movie viewing activities which can be discovered by a frequent-pattern tree (FP-Tree). The FP-tree is then modified with rating data and an effective recommendation strategy can improve the recommendation performance of the FP-tree. A MovieLens dataset which is public and popular for evaluating movie recommender systems is observed and examined for assessing the proposed method.

1 Introduction

Due to the information explosion on the Internet, users are facing with an enormous number of choices. A pool of spam data and inaccurate information may require a lot of time for searching relevant information. Recommender systems (RSs) have become indispensable tools to filter available data and provide the user with the most relevant information. Most RSs use a hybrid approach, which is a combination of content-based and collaborative approaches. Collaborative filtering algorithms [1] assume that users with similar tastes will rate items similarly. A content-based recommendation system [2] uses the user's history to recommend new items.

In this study, the data of user's viewing history and ratings are considered. Datasets are collected from MovieLens site (https://www.movielens.org). We use FP-tree [3] to present a novel clustering method which generates frequent patterns of movies and recommend appropriate movies. The FP-tree was built based on users' movies watching activities. The rating data is discovered and then modified into the FP-tree in order to remove low rated movies and improve recommendation performance. The system will be evaluated by various experiments.

The remaining of this article is organized as follows. Related works are considered in Sect. 2, and a new movie recommendation framework is introduced in Sect. 3. Section 3 presents the methodology of using FP-Tree, and Sect. 4 shows its experimental results and discussions. Finally, Sect. 5 concludes and discusses future work.

V. Bhateja et al. (eds.), *Information Systems Design and Intelligent Applications*, Advances in Intelligent Systems and Computing 672,
https://doi.org/10.1007/978-981-10-7512-4_17

2 Related Work

In the field of recommender systems, many approaches have been developed and some recent ones are listed below.

Hybrid Multigroup Co-Clustering (HMCoC) [4]
A Hybrid Multigroup Co-Clustering recommendation framework has been proposed to achieve meaningful user-item groups by extracting user-item rating records, user social networks, and item features from DBpedia knowledge base. This framework is composed of three main modules: information fusion, hybrid multigroup co-clustering, and the top-n recommendation module. Information fusion integrates data from the rating matrix, user social networks, and item's topic. After that, it utilizes some publicly available knowledge base. Then, it uses a uniform graph model to represent the integrated information. HMCoC co-clusters users and items into multiple groups simultaneously. It combines the one-sided and two-sided clustering techniques and presents a fuzzy c-means-based clustering method to discover user-item clusters with different information sources. As a result, it merges the predictions from each cluster, and then makes top-n recommendations to the target users.

User-specific Feature-based Similarity Models (UFSMs) [5]
User-specific Feature-based Similarity Models take historical user preferences into account for building a personalized user model. To build this model, global similarity functions are learned by combining linearly user-independent similarity functions. As a result, we will have item similarity functions learned. Furthermore, these global similarity functions are combined linearly and personalized for users. It is proved to outperform both regression-based latent factor modeling (RLFM) and attribute-to-feature mapping (AFM) methods in cold-start top-n item recommendations.

Recommender Systems using Category Correlations based on WordNet Similarity [6]
In these recommender systems, genres of movies in the database are considered to draw genre correlations among movies. Each movie will be then assigned a new score by computing its average rating and genre correlations. Through the users' inputs, user's preferred genres are gained, and from the ratings of each movie, average movie ratings are estimated. From that, the scores are sorted in descending order and the high position movies can be recommended to active users.

Empirical Study of User Preferences Based on Rating Data of Movies [7]
This study represents a hyper-network of users and movies. In this network, a node of a user connects to many movies, and a node of a movie connects to many users. Rating data was used to calculate distances between movies. If a user rated movie *a*, then it is predicted that the user will rate movie *b*, when movies *a* and *b* are first-order h-neighbors. The idea was that if movie *a* was the first-order h-neighbor of movie *b,* the opinion of a user about movies *a* and *b* are almost the same. A user preference model with two tunable parameters has been introduced after many analysis results.

Grouping Like-Minded Users for Ratings' Prediction (GLER) [8]
Principal components analysis (PCA) and K-Means were used to group like-minded users. A recommender system was trained using those user groups, and for each group we build a specific model. GLER algorithm will give ratings predictions of user-movie based on the previous ratings of that user and others in the same group using this group's model. The model of closest group will be used for a new user. MovieLens-100K data set and SVD++ (Singular Value Decomposition) were used to evaluate this algorithm. Root mean squared error (RMSE) and mean absolute error (MAE) were also used as the two evaluation metrics.

3 Methodology

3.1 Framework

Figure 1 illustrates the framework of the proposed recommender system including three main process units: (1) preprocessing, (2) building FP-tree, and (3) recommendation engine.

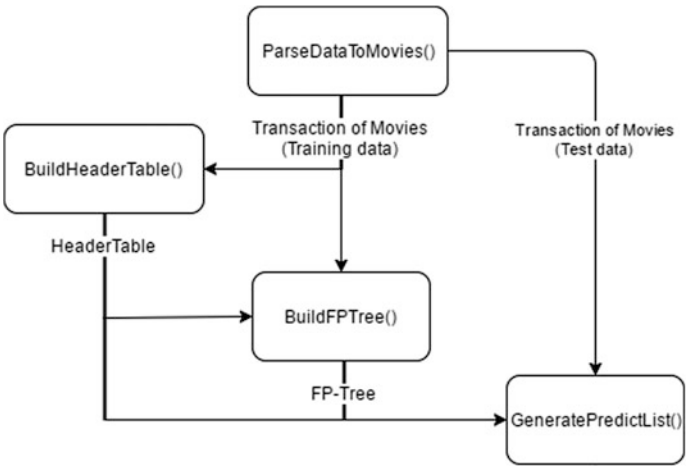

Fig. 1. Recommender system framework

(1) Preprocessing (ParseDataToMovies() in Fig. 1)
There are two ways to achieve movie transactions from MovieLens dataset in this phase: (1) Low rating movies are retained and (2) low rating movies are removed. We are going to have an experimental comparison about accuracy and runtime between retaining low rating movies and removing them.

The following presents the two algorithms of processing data.

Parameters:

- Timestamp represents seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. It is the time when a movie was viewed.

- Constant period represents time interval between two transactions, that is, movies are viewed continuously.

Method 1: With Low Rating Movie

```
Input: users' movie viewing data
Output: Processed data
Data processing:
FOR each line of data
    FOR each movie of a user
        IF (timestamp of current movie) – (timestamp of
        previous movie) < period THEN
            Add current movie into current transaction.
        ELSE
            Add current movie into a new transaction.
```

Method 2: Without Low Rating Movie

```
Input: users' movie viewing data, and rating data
Output: Processed data
Data processing:
FOR each line of data
    FOR each movie of a user
        IF (timestamp of current movie) – (timestamp of
        previous movie) < period THEN
            Add current movie into current transaction.
        ELSE
            Add current movie into a new transaction.
REMOVE all movies whose rating is less than 2.5
```

(2) FP-Tree

FP-tree is a compact structure compressing a large database of event sequences into a tree for complete frequent pattern mining. It also avoids scanning data repeatedly, that is very costly. Generally, an FP-tree structure is defined as follows: "one root labeled as "null," a set of item prefix sub-trees as children of the root, and a frequent-item header table." Each node in the item prefix sub-trees is constituted by three fields: (1) item-name registering which item this node represents, (2) count being the number of transactions, i.e., the portion of the path reaching this node, and (3) node-link linking to the next node carrying the same item-name. Each entry in the frequent-item header table is constituted by two fields: item-name and head of node-link that points to the first node carrying the item-name.

The algorithm of FP-tree construction is described as follows:

a. Scan the set of transactions *S* once; Retrieve *F*, which is the set of frequent items, and compute the support of each item; Sort *F* in support-descending order.
b. Create the root node of an FP-tree, tree, and label it as "null"; For each transaction *T* in *S* do the following:

- Find frequent items in *T*. Let the selected frequent-item list in *T* be [*e* | *E*], where e is the first element and E is the remaining list. Call *insert-tree*([*e* | *E*], *Tree*).
- *insert-tree*([*e* | *E*], *Tree*): If tree has a child *C*: *N.item-name* = *e.item-name*, then increase C's count by 1; else create a new node C, and set its count to 1, link its parent-link to tree, and link its node-link to the nodes carrying the same item-name. If *E* is nonempty, call *insert-tree*(*E*, *C*) recursively.

(3) Recommendation Engine (GeneratePredictList() in Fig. 1)
The data sets are firstly preprocessed and clean. Firstly, all movies that each user watches were added into a transaction. Then we divide this transaction into smaller transactions using *timestamp*. The reason is that in a period of time, a user may like a set of movies but in another period this user may be interested in another set of movies. After having these transactions, we are able to build FP-tree.

A movie recommendation algorithm is proposed as follows: When a testing transaction is input, the algorithm would apply top-n recommendations based on the built FP-tree. That means subsequences of movies in the transaction are matched with patterns in the FP-tree, the longest matching patterns will be considered. Movies in the matching patterns and movies in children nodes will be candidates for recommendation. They are sorted in descending weights (counts) and top-n movies are recommended.

3.2 Evaluation Methods

This study applies two evaluation metrics: precision and satisfaction for assessing the performance of the movie recommendation system. Precision is the proportion of number of correct predictions in the next step to the number of matching times. While, satisfaction is the proportion of the number of correct predictions in next m-steps to the number of matching times. The following describes evaluation algorithms for performance measure.

Performance evaluation:
Testing data is a subset of users' movie viewing sequences, in which, a transaction is a sequence of viewing movies continuously. In other words, in a transaction, time interval between two viewed movies should not be longer than a predefined period.

a. Calculate precision:

```
set positive = 0; and set negative = 0;
FOR each transaction in the testing data.
     FOR i = 1 to length of current transaction.
        Get the list of recommended movies for the active
        sequence of watched movies from 1 to index = i.
        IF movie at index = i+1 in current transaction
        CONTAINS at least one movie in the recommendation list
        THEN INCREMENT positive
        ELSE INCREMENT negative
Precision = positive / (positive + negative) *100%
```

b. Calculate satisfaction with m = 5

```
FOR each transaction in the testing data.
     FOR i = 1 to length of current transaction.
        Get the list of recommendation movies for the active
        sequence of watched movies from 1 to index = i.
        IF movies at index = i+1 to i+5 in current transaction
        CONTAINS at least one movie in the recommendation list
        THEN INCREMENT positive
        ELSE INCREMENT negative
Satisfaction = positive / (positive + negative) *100%
```

Return Precision and Satisfaction

4 Experiments

In order to evaluate the performance, fivefold cross-validation is applied. Besides, a number of values are set for the min_support and the timestamp to examine several different points of view. The min_support is used to filter less interesting movies. Removing low rating movies can return good experimental results. Applying the precision and statistical metrics is very effective to evaluate the system's performance.

4.1 Dataset

A popular MovieLens dataset[1] containing 20 million of movies is used. In this dataset, we focus on ratings.csv (rating data) and tags.csv (users' movie viewing behaviors). Each line after the header row in file ratings.csv represents one movie's rating by each user following this format: userId, movieId, rating, timestamp. The lines within this file are ordered first by userId, then, within a user, by movieId. Ratings are based on a five-star scale, with half-star increments (0.5 stars–5.0 stars). Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

The datasets are firstly preprocessed and cleaned before building FP-Tree. We add all movies within a user into a transaction. Then timestamp is used to divide a transaction into smaller transactions. The period of each smaller transaction is predefined by observing the datasets. After having these transactions, we can build the FP-tree.

4.2 Experimental Results

In the following, we conduct six experimental cases.

a. Keep low rating movies in the dataset, and the period of each examined transaction is set to 1000000 s. The satisfaction and precision are measured at different min_support values: 7, 8, and 9%. Table 1 shows the results of satisfactory measure with m = 5.

Table 1. Satisfactions examined for three min_supports 7%, 8%, and 9%

Min_support (%)	Satisfaction (m = 5) (%)	Runtime (h)
7	100	40
8	99	12
9	99	3

Table 2. Precisions examined for three min_supports 7, 8, and 9%

Min_support (%)	Precision (%)	Runtime (h)
7	99	40
8	99	12
9	99	3

Table 1 shows that choosing min_support is not easy. We can achieve 100% satisfaction for min_support of 7%, but it costs too much time. For min_support of 9%, the satisfaction decreases 1%, but it takes less time. It is similar when measuring precisions, but we can just obtain 99% of accuracy at min_support = 7% (Table 2).

[1] https://www.movielens.org, https://www.imdb.com.

b. Keep low rating movies in the dataset, and the min_support is set to 8%. The performance is evaluated in two cases: the chosen periods of each examined transaction are 1000000 and 500000s (Tables 3 and 4).

Table 3. Satisfaction and runtime for the two different chosen periods

Timestamp (s)	Satisfaction (m = 5) (%)	Runtime (h)
1000000	99	12
500000	97	2

Table 4. Precision and runtime for the two different chosen periods

Timestamp (s)	Precision (%)	Runtime (h)
1000000	100	12
500000	97	2

The chosen period affects the accuracy (satisfaction and precision) and runtime, as in Tables 3 and 4. Period 100000s gives higher accuracy (99%) but it takes 12 h to complete all testing data. Period 500000s gives a lightly lower accuracy (97%) but it takes only 2 h.

c. Low rating movies can be removed or not from the dataset. It is supposed that low rating movies may be not interested by users, so they should not be taken into account. The chosen period of each examined transaction is 1000000s, the min_support is 8%, as shown in Tables 5 and 6.

Table 5. Satisfaction and runtime for two cases: with and without low rating movies

	Satisfaction (m = 5) (%)	Runtime (h)
Without rating movies	99	12
With low rating movies	99	4

Table 6. Satisfaction and runtime for two cases: with and without low rating movies

	Precision (%)	Runtime (h)
Without rating movies	99	12
With low rating movies	99	4

The results in both Tables 5 and 6 give the same accuracies but the runtimes are significantly different between the two cases with and without low rating movies. Thus, removing low rating movies offers much more benefit.

d. Removing low rating movies and the chosen period of each examined transaction is 500000s. The min_support is set to 8%. Table 7 shows that we receive the same precision and satisfaction (m = 5), i.e., accuracy, with the same runtime. Thus, it proves that the proposed method can achieve significantly high performance.

Table 7. Accuracy and runtime when applying the precision and satisfaction (m = 5) metrics

Metric	Accuracy (%)	Runtime (h)
Precision	96	1
Satisfactory with m = 5	96	1

e. Removing low rating movies and the chosen period of each examined transaction is 500000s. The min_support is set to 8%. In this experiment, we apply fivefold cross-validation to accuracy estimate (Table 8).

Table 8. Accuracy (precision and satisfactory) and runtime when applying fivefold cross-validation

Metric	Accuracy (%)	Runtime (h)
Precision	98	7
Satisfactory with m = 5	98	7

In fivefold cross-validation, the experimental data is divided into fivefolds, onefold is used for testing while the remaining fourfolds are used for training. Each fold has once it becomes testing data and four times it is a part of the training data. Applying fivefold cross-validation in Table 8 will make the accuracy much more reliable. As a result, the performance achieves higher accuracy (98%).

f. Comparing with the experimental results in [5] using the same dataset, our results are remarkable.

The same data set MovieLens-1M was used for our recommendation engine. We also applied the same Recall at n metric (Rec@n) to compare the performance of our method with UFSM. User profile was the input for UFSM while we used user history to recommend movies. Given top-n recommendation movies for a user, Rec@n of the user is computed as: *Rec@n = number of matched movies / n*. Rec@n is computed for each user and then averaged over all test users.

Figure 2 shows that the performance of top-n recommendation of USFM and our method for values of n = 5, 10, 20. The accuracy of our method are much higher than USFM. About the training time, if we run on the MovieLens-20M dataset, it will take 68.5 minutes. While, USFM takes about eight times of training the MovieLens-1M dataset, which is 20 times smaller than the one used in our experiments. Table 9 shows how efficient our method is.

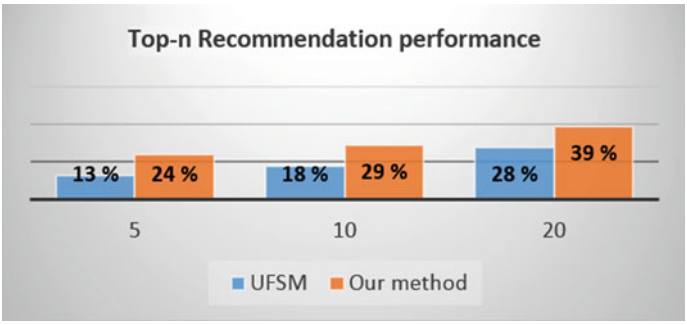

Fig. 2. Performance of top-n recommendation

Table 9. Training time in minutes

Method	Training time in minutes
USFM	566.31
Our method	68.5

5 Conclusions

In this study, we have proposed a recommendation system with an advanced sequence mining method using FP-tree and the effective recommendation strategy. This methodology compresses a large database into a compact FP-tree. Based on that, movies are recommended efficiently. In particular, the largest dataset (20M) in MovieLens is used to test its performance.

The experimental results have showed that if we remove low rating movies from the dataset, we can save a lot of time with a slightly decreased accuracy rate (about 1%). The proposed methodology is proved to achieve high performance, and the precision is almost acceptable.

In the future, we will consider the levels of rating movies which helps rank the movies and recognize which movie a user likes most. The count of a node in FP-tree will be replaced by the weight. Ratings will be used to calculate the weight. Since ratings are based on five-star scale, we choose the average of rating 2.5 to be equal to 1 unit of weight. The function *insert-tree*([$e \mid E$], *Tree*) will have some change according to the weight.

Acknowledgements. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this article.

References

1. Mcleod, D. & Chen, A. Y.-A., Collaborative Filtering for Information Recommendation Systems. Non-published Research Reports (2009).
2. Pazzani, M. J. & Billsus, D., Content-Based Recommendation Systems. In: Brusilovsky, p., Kobsa, a. & Nejdl, W. (eds.) The Adaptive Web: Methods and Strategies of Web Personalization. Berlin, Heidelberg: Springer Berlin Heidelberg (2007).
3. Han, J., Pei, J., Yin, Y. & Mao, R., Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery, 8 (2004) 53–87.
4. Huang, S., Ma, J., Cheng, P. & Wang, S., A Hybrid Multigroup Coclustering Recommendation Framework Based on Information Fusion. ACM Trans. Intell. Syst. Technol., 6 (2015) 1–22.
5. Asmaa Elbadrawy and George Karypis, User-Specific Feature-Based Similarity Models for Top-n Recommendation of New Items, ACM Transactions on Intelligent Systems and Technology, Vol. 6, No. 3 (2015).
6. Choi, S.-M., Cho, D.-J., Han, Y.-S., Man, K. L. & Sun, Y. Recommender Systems Using Category Correlations Based on WordNet Similarity, International Conference on Platform Technology and Service (PlatCon), 26–28 Jan. (2015) 5-6.
7. YingSi Zhao, Bo Shen, Empirical Study of User Preferences Based on Rating Data of Movies, PLoS ONE 11(1): e0146541. https://doi.org/10.1371/journal.pone.0146541, January 6 (2016).
8. Jaffali S., Jamoussi S., Hamadou A.B., Smaili K., Grouping Like-Minded Users for Ratings' Prediction. In: Czarnowski I., Caballero A., Howlett R., Jain L. (eds) Intelligent Decision Technologies 2016. Smart Innovation, Systems and Technologies, vol 56. Springer, Cham (2016).