Collaborative Filtering and Why-Not Queries

TIETS43 Recommender Systems, Tuomas Luojus & Toni Kuikka

# Introduction

Recommender systems are known as information filtering platforms that seek to predict the preference that a user would give to an item. In recent years, the application areas of them have significantly increased due to the digitalization and nowadays recommender systems are also widely utilized in giving suggestions in video and music services such as YouTube, Spotify and Netflix.

For instance, one of the highly popular forms of entertainment are movies. According to IMDB, since 1900 on the average 2 500 films have been produced per year around the World and during this decade, the film production has doubled. While new movies are continuously released, the older ones are still available to watch which makes it unachievable for a user to go through all releases manually. Nowadays, increasing number of movies are watched online in streaming medias and to manage this type of information overload, the streaming services usually have implemented several types of recommendation engines.

One of the most widely used and well-understood recommender techniques is collaborative filtering that is based on the idea that users who had similar tastes in the past, will have similar tastes also in the future. In the user-based collaborative filtering, the purpose is to find a set of users that have had similar taste in the past and have rated the items that are not yet seen by the target user. In the end, the algorithm is supposed to output a list of unseen items that have received the best ratings according to, for example, the average of the ratings of the similar users.

However, although recommender systems aim to give the most valid suggestions, sometimes they can fail at locating the best possible items to the specific user. This can be due to several reasons of which the so-called cold start problem, when there is not enough information about a user available, the over-specification on the part of the users, and the misdirection due to ambiguous information on the users are somewhat common. In this paper, our goal is to implement a simple prototype of a user-based collaborative filtering system for movies and based on it, analyses questions concerning why certain items were not recommended. We will focus on the following issues:

* Existence: Why does not certain item/genre exist in the recommendation list?
* Position: Why is not certain item/genre higher in the ranking?
* Grouping: Why there are not more certain items/genre in the recommendations?

# Background

In this chapter, we look at literature used as background for our analysis on recommendation systems and collaborative filtering.

## Recommender systems

Recommendation systems are usually classified into three categories: content based, collaborative, and hybrid recommenders. Content-based methods are based on the process of indexing and calculating similarities in textual or other content of items, assuming items similar in content to be evaluated similarly by users. Collaborative methods or *collaborative filtering* is a method which takes into account reviews of other users and finds similar items based on that information. For example, the system could first find similar users or *neighbors* of the current user, and using the review data of neighbors, the system could calculate estimates for the current user. Hybrid recommendation systems combine the two aforementioned methods to create recommendations to the user (Adomavicius & Tuzhilin, 2005; Melville & Sindhwani, 2011).

Melville and Sindhwani have written a comprehensive review of recommendation systems, in which they look at the three categories of recommender systems and their general implementations. They describe recommendation systems as “a critical component of the natural process of human decision making”. They claim that the need for recommendation systems are rising as consumerism and E-commerce and the number of products rise (Melville & Sindhwani, 2011).

Isinkaye, Folajimi, and Ojokoh claim recommender systems to be essential in navigating present-day life overflowing with information and choices, i.e. *information overload*, especially in the context of the internet. They describe the recent trends in recommender systems by analyzing the implementation of different prediction techniques in recommenders. They organize the recommendation process into *information collection phase*, *learning phase*, and *prediction/recommendation phase* (Isinkaye, Folajimi & Ojokoh, 2015).

## Collaborative filtering

Su and Khoshgoftaar have examined the three models of collaborative filtering (CF): memory-based(a.k.a. *Neighborhood-based* [Melville & Sindhwani, 2011]), *Model-based,* and *hybrid* CF. While neighborhood-based models generate recommendations based on ratings of some number of similar users, model-based CF uses statistical models and different classifiers to generate recommendations. Hybrid CF combines CF with other recommendation techniques (Su & Khoshgoftaar, 2009).

Koren and Bell claim recent CF techniques to be shifting away from purely neighborhood-based techniques and focusing more on latent factor models such as matrix factorization, which generates recommendations automatically by linear algebra; vectorizing items and users using factors automatically inferred from user feedback. In their article, Koren and Bell also present common problems of neighborhood-based techniques and ways to improve neighborhood-based recommendation models (Koren & Bell, 2015).

# Analysis of why-not queries

Sometimes the recommendations given can differ from the expectations of the user, and the user might want to know certain information about the recommendations. Why-not queries are a way for user to ask question about the recommendations given to them by the recommendation system. Essentially the user is able to input a question in natural language, for which the system would give an answer based on the given recommendations and estimated scores of other items.

We wanted to examine why-not questions presented in table 1. In our prototype website the recommender system gives the user five recommended movies based on their prior reviews. Then the user can ask why-not questions about those recommendations. Then the python program tries to parse the natural language questions and give the proper answer.

|  |  |  |
| --- | --- | --- |
|  | Independent | Dependent |
| Existence | Why was item X not recommended to me? | Why was item Y recommended whereas item X was not? |
| Position | Why is item X not in a higher rating? | Why is item X below Y? |
| Grouping | Why are there no more genre X movies? | Why so few genre X movies but so many genre Y movies? |

Table 1: Possible why-not questions

In table 2, possible answers to the six why-not questions are shown. In the case of *general answers*, X can be an item or a type of item. For example, in our prototype’s case, Movie names or genres. Scores such as XR are given as estimated scores for that user.

|  |  |
| --- | --- |
| Why-not question | Possible answers |
| General answers (all questions) | AG1: X is already rated.  AG2: X does not exist in database.  AG3: It looks like none of the users that are similar to you have rated item X and therefore we were unable to predict a rating for it. |
| Q1: Why was item X not recommended to me? | A1: It looks like we only predicted a rating of XR for that item based on the rating of similar users to you. The minimum rating to be in recommendations would have been X. |
| Q2: Why was item Y recommended whereas item X was not? | A2: It looks like we predicted a rating of YR for Y based on the ratings of similar users to you whereas X received only a rating of XR. The minimum rating to be in recommendations would have been KR. |
| Q3: Why is item X not in a higher rating? | A3: It looks like that movie got a predicted score of XR and got a placing i. The movies between the placements 1 and k got ratings between 1R and kR. |
| Q4: Why is item X below Y? | A4: It looks like we predicted a rating of YR for Y based on the ratings of similar users to you whereas X received a rating of XR. Due to this, X placed k places lower than Y. |
| Q5: Why are there no more genre X movies? | A5: The highest rated movie(s) of genre X are X1 and X2 which have ratings of X1R and X2R. The minimum rating to be in recommendations would have been KR. |
| Q6: Why so few genre X movies but so many genre Y movies? | A6: You were recommended k X movie(s) and j Y movie(s). X movies such as X1 received a rating X1R at best whereas Y movies such as Y1 a rating Y1R at best. |

Table 2: possible answers for why-not questions

We modeled our prototype to read inputs such as Q1-Q6 and provide the respective answers in natural language to the user. We sought to read the inputs as flexibly as possible. For example, the user does not have to give the full name of a movie and can just give a part of the name (I.e. *New Hope* instead of *Star Wars: Episode IV - A New Hope (1977)).*

# Implementation of the recommender system

In this section, we present our implementation of the prototype of the recommendation system that aims to give responses to the Why-Not queries presented in the Section 3. The code is based on the User-Based Collaborative Filtering system presented in the article *Recommendation Systems: User-based Collaborative Filtering using N Nearest Neighbors* (Pathak, Mandava & Patel, 2019). The process contains four main phases. Firstly, the similarities between the users are measured by the cosine similarities that are calculated from the normalized ratings of the users. Secondly, based on the Cosine similarities, the k most similar peers for each user are defined by the k-Nearest Neighbors algorithm. After that, for each unseen movie of the target user, the predicted scores are given according to the ratings of the most similar users and the list of recommendations is printed. Finally, the user is able to ask for the program questions that are presented in the Section 3 and the system aims to respond to them.

The input data sets of movies from MovieLens-dataset (GroupLens, n.d.) and the normalized ratings given by the users are firstly merged into one large data frame so that each row corresponds each user whereas each column corresponds each movie. Thus, cell [i][j] is equivalent with the normalized rating given by the user at index i to the movie at index j. If the user has not rated the movie, the cell has a value NaN. This is done for simplifying the operations that will be presented in the following subchapters. In practice, the program is implemented in Python and with help of libraries such as Pandas, Numpy and statistics.

## Cosine Similarity

One of the most widely used measures of similarity is the Cosine Similarity Measure. In this method, ratings are seen as vectors of an inner product space. Similarity between two users is calculated based on the angle between their corresponding vectors so that the cosine of 0° is 1 (full similarity) and less than that for any angle in the interval (0, π] radians. The cosine similarity between users *u1* and *u2* is calculated with the formula

,

where *r1i* is equivalent with the rating given by the user *u1* and *r2i*is equivalentwith the rating given by the user *u2* to movie *i*.

However, it is a well-known fact that people tend to rate using different criteria so that some give full marks more often than others, for instance. To obtain more efficient measures, the ratings of the users are normalized before defining the cosine similarities. It is done so that for each user, the average rating is calculated at first. Then, users’ ratings are compared with the averages so that the normalized rating is 0 when it is the same as the average, negative when it is less than the average and positive when it is more than the average. Finally, in this implementation, the cosine similarities from the normalized ratings are simply calculated by the *cosine\_similarity* function provided by the *sklearn.metrics.pairwise* library.

## k-Nearest Neighbors

To find the *k* closest peers for the target user, the application of k-Nearest Neighbor method, which is one of the simplest instance-based learning methods, is utilized. The main idea is to compute the distances between the target user and its every peer in the sample and after that, select a subsample of size *k* that contains the closest peers to the user.

In the program, the method is implemented as a simple algorithm that takes the distance matrix of all users that is based on the cosine distances computed in the previous phase as well as an integer *k* as its parameters. The algorithm sorts each row of the distance matrix in increasing order and returns the *k* first columns from the sorted matrix. The diagonals of the distance matrix are not considered because they indicate the users’ distances to themselves and are supposed to have zero as their values.

In general, the best choice of *k* depends on the data. For example, in a classification task, larger values of *k* reduce effect of the noise while make boundaries between classes less distinct. Nevertheless, in this application, circa 30 are considered as a sufficient number of neighbors.

## Predicting scores

In this phase, for each unseen movie of the target user, the predicted scores are calculated based on the ratings of the nearest neighbors. At first, the movies that will be under consideration are defined by listing the movies that are ranked by the k-nearest neighbors so that the movies already seen by the target user are excluded. Then, for each movie *i* in the list, the weighted rating of the target user *u* is predicted with the following score function:

,

where  is the average of the ratings given by the user *u*, *k* is the number of nearest neighbours, *rji* is the rating given by the neighbour *j* to the movie *i* and j is the average of the ratings given by the neighbour *j* (normalizing). The predicted ratings are weighted by the cosine similarities between users *u* and *j,* marked as ρuj in the formula above.

In practice, the algorithm calculates the score for the movie by listing the normalized ratings of each neighbor that has rated the movie (i.e. has not *NaN* as the rating) as the first column of a data frame at first. Then, the cosine similarities (weights) between neighbors and the target user are listed as well as the second column of the data frame. Finally, the values (normalized rating and weight) in each row are multiplied and then the multiplication results as well as the values of the second column (weights) are summed up. The results of these data frame operations are then used in the score function above to obtain the predicted rating for each movie under consideration.

At the end, all the predicted ratings are listed and then the list is sorted in descending order. The algorithm returns first five values of the sorted list as a top-5 recommendation for the target user.

# Conclusion

We have examined recent findings in the study field of recommendation systems, and especially collaborative filtering techniques. We then looked at six types of Why-not questions and defined possible answers to them. Following this information, we have created a recommendation web application with Python programming language using MovieLens dataset. In that prototype, the user is given five movie recommendations based on nearest neighbor collaborative filtering technique. The user can then ask why-not questions in natural language from the recommender, for which the system will give an answer.

Although basic nearest-neighbor technique of collaborative filtering was satisfactory for the purposes of this study, it seems that other techniques like collaborative filtering-content based hybrid and model-based ones are becoming increasingly prevalent. In future research, the performance differences of these techniques should be looked at more comprehensively, but since our study mostly focused on why-not queries, the technique of providing recommendations to the user was not in our greatest interest, and we found that the recommendations given by nearest-neighbor technique was sufficient in providing answers to why-not questions.

Interpreting the question from the user proved to be perhaps the most challenging part of creating the prototype. In fact, the user must write their question somewhat according to a predetermined formula in order to get a meaningful answer, so there is place for future research for interpreting why-not questions. Another issue that was out of scope for this study was personalization of search results based on user’s why-not questions. Ideally the given search results would automatically or optionally change according to the question input of the user.

# References

* Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, (6), 734-749.
* Melville P., & Sindhwani V. (2011) Recommender Systems. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA
* GroupLens. *MovieLens*. (n.d.). Retrieved from https://grouplens.org/datasets/movielens/
* Isinkaye, F. O., Folajimi, Y. O., & Ojokoh, B. A. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, *16*(3), 261-273.
* Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in artificial intelligence*, *2009*
* Koren, Y., & Bell, R. (2015). Advances in collaborative filtering. In *Recommender systems handbook* (pp. 77-118). Springer, Boston, MA.
* Pathak, A., Mandava, C., Patel, R. (2019).Recommendation Systems: User-based Collaborative Filtering using N Nearest Neighbors. *SFU Big Data Science.* Referenced in 11.12.2019.Retrieved from<https://medium.com/sfu-big-data/recommendation-systems-user-based-collaborative-filtering-using-n-nearest-neighbors-bf7361dc24e0>