# Recommender system using active learning and recursive algorithm in collaborative filtering

**Haithem BEN DRISSI**
Université Paris Dauphine-PSL

## Abstract

In this work we present the state of art of collaborative filtering, active learning, and recursive algorithm. Finally, we introduce the implementation of an approach based on active learning used in collaborative filtering to suggest movies to a new user. We proceed with two layers, the first will use the active learning approach to select the first films to be evaluated by the user when registering, and in the second layer we will use the recursive prediction algorithm applied to nearest neighbor based collaborative filtering to predict the rating of the movies by the active user (new user) and suggest the highest ones. The data set used is from [MovieLens], a movie recommendation service.

## 1 Introduction

As part of my MODO master's degree at Paris Dauphine-PSL University, in the IA and Decision workshop, I am required to implement a recommendation system. a bibliographic study on active learning, collaborative filtering, and the recursive prediction algorithm for collaborative filtering recommender systems. In recommender systems with collaborative filtering, user preferences are represented as ratings for items, and each additional annotation increases the system's knowledge and affects the system's recommendation accuracy. In general, the more annotations you receive from users, the more effective the recommendation will be. However, the usefulness of each symbol can vary widely. Different ratings can provide different amounts and types of information about the user's taste. Thus, specific techniques defined as "learning strategies" can be used to selectively select what is presented to the user for evaluation. In fact, active learning strategies identify and employ criteria to obtain data that better reflect user preferences and generate better recommendations.

The key idea is as follows: if the nearest neighbor user has not yet rated a given item, we first recursively estimate his/her rating value based on his/her own nearest neighbors, and then use the estimated rating value to participate in the prediction of joining the last active user process.

## 2 Collaborative filtering

Collaborative filtering (CF) recommender systems use item ratings provided by a collection of users. It recommends items that the target user has not considered but may like [Koren and Bell, 2015]. Scores are stored in an (m × n) matrix, where m is the number of users and n is the number of items. The rows of this matrix store user ratings and the columns store item ratings. If a user rates an article, the corresponding entry will be assigned to the submitted rating. This is the input data for the collaborative filtering system. When a new user logs into the system, a new blank row is added to this matrix. Likewise, adding a new item to the catalog will add a new empty column.

Collaborative filtering systems compute recommendations by exploiting relationships and similarities between users and items. These relationships result from user interactions with elements managed by the system.

### 2.1 Neighborhood-based models

Neighborhood-based techniques are either user-based or object-based [Desrosiers and Karypis, 2011]. User-based methods use two sets of data to compute rating predictions for target users.

Ratings of target users and other like-minded users, ei., Users with similar scoring patterns. The target user's rating of the item is then predicted based on how the item is rated by users such as the target user. In user-based methods, there are several ways to compute rating predictions. A popular formula is as follows, where $r_{ui} \in \{1, \ldots, 5\}$ is the known (five-star) rating for item $i$, and $\hat{r}_{ui}$ is the system's prediction for item $i$ evaluation by $u$:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N_i(u)} sim(u,v)(r_{vi} - \bar{r}_v)}{\sum_{v \in N_i(u)} |sim(u,v)|} \quad (1)$$

Here $\bar{r}_u$ denotes the average rating of user $u$, $sim(u,v)$ is the similarity of the users $u$ and $v$, and $N_i(u)$ is a set of users like user $u$ (neighbours) who rated item i. The user-to-user similarity can be computed using different approaches.

A popular one is Pearson correlation [Adomavicius and Tuzhilin, 2005]:

$$sim(u,v) = \bar{r}_u + \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}} \quad (2)$$

where $I_{uv}$ is the set of items co-rated by users $u$ and $v$.

### 2.2 Cold start problem in collaborative filtering

The cold start, or when the system does not yet have enough ratings to create accurate recommendations, is a major issue with collaborative filtering. The most common scenarios are when the system cannot accurately propose existing items to a new user (new user problem) or when the system cannot accurately recommend a new item to current users (new item problem) [Bishop, 2006].
While these are the most prevalent cold-start scenarios, another issue to consider is the sparsity of the rating data.
The inverse of the density of the available ratings is measured by sparsity, which is defined as:

$$1 - \frac{\#\ of\ available\ ratings}{\#\ of\ all\ possible\ ratings} \quad (3)$$

In real recommender systems, sparsity is often very close to 1 (percent 100) because users only rate a small percentage of the accessible items. As a result, the system has a hard time making reliable recommendations.

Obtaining training data in this manner can be costly and time intensive, and data that isn't necessarily necessary for improving system performance may be processed.

The number of data points used to train a model based on active learning is often fewer than the number used by a passive learner due to the interaction with the user. It has demonstrated promising effectiveness in the treatment of cold starts.

## 3 Active learning

Machine learning [Mitchell, 1997] is a set of computational approaches that allow a system to enhance its performance by analyzing data. Several strategies have been suggested and implemented [Tong, 2001]. However, employing these strategies frequently necessitates a large volume of high-quality data. Such information can be received in two ways: passively and actively.

The complete data set is provided to the passive learner, and the model (or classifier) is developed using that data. That's where active techniques come in, attempting to pick and choose the data to obtain. "Active Learning" [Chang and al., 2015] refers to the act of steering the sampling process by inquiring for certain categories of instances based on the data the system has seen so far. In general, active learning is well-motivated in a variety of machine learning tasks where data is not sufficient or costly to get. The quantity of data points required to train an active learning model is often less than that required by a passive learner. To reduce the amount of training labels, active learning can be used [Kohrs and Merialdo].

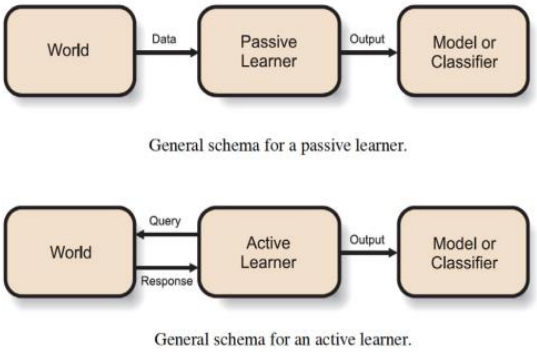


**Figure 1 - The general schema for active learning vs. passive [Chang and al., 2015].**

Information extraction can also result from active learning. In this situation, the system will need to access properly labeled documents to extract data such as relationships between entities (e.g., a person works in a company). This takes time and sometimes necessitates a high level of knowledge. To reduce the number of marked documents required for training, active learning can be used.
Given a training set of $N$ input–output pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $x_i \in X$ is an instance, and $y_i \in Y$ is a label, the model is such that : output M : X → Y, and consider a function Loss(M) which measures the error of the model (the smaller the better). At every iteration j of the active learning process, a query is selected
$q_j \in potentialQueries \subset X$, requesting the label $y_j$ of it.
when the system selects the instance $q_j$, its label is unknown, so to score that $q_j$ is to use the expected loss of M' [Chang and al., 2015] which is the model M having the lowest loss while re-trained by adding to the set of previously labelled instances the new pair $(q_j, y_j)$:

$$Loss(q_j) = E(Loss(M')) \quad (4)$$

**Algorithm 1** General algorithm for Active Learning [Chang and al., 2015]

**Input**: $N = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
**Output**: Model M
1: **for** $j := 1$ to totalIterations **do**
2: **foreach** $q_j$ in potentialQueries **do**
3: Evaluate $Loss(q_j)$;
4: **end**
5: **Ask** query $q_j$ for which $Loss(q_j)$ is the lowest;
6: **Update** M with query $q_j$ and response $(q_j, y_j)$;
7: **end**
8: **return** model M;

## 4 Active learning in collaborative filtering

Active learning for recommender systems has been initially motivated by the need to implement a more effective sign-up process [Desrosiers and Karypis, 2011]. The system actively picks and suggests to users a group of items to rate during the sign-up step [Rashid and al., 2008]. The system does this by evaluating the complete set of things items and selecting the ones that are thought to be the most beneficial. The ratings assigned to these items are intended to improve the system's accuracy the greatest.

In the next sections we will describe in detail how the items are evaluated, scored, and selected by different active learning strategies. We classify these strategies into two main categories, i.e., non-personalized, and personalized.

### 4.1 Non-personalized active learning

Simpler active learning strategies don't consider past user ratings and instead ask everyone to review the identical items. These strategies are referred to as non-personalized. The heuristic employed for item selection in this scenario is independent of the profile of individual users. An example of a non-personalized technique is by "popularity," where the system presents to all users the same popular items to rank.

In this part we present some non-personalized active learning strategies used in a recommendation system based on collaborative filtering. Three are types: uncertainty reduction, error reduction and attention based.

**Uncertainty-reduction**

The system is supposed to be more uncertain about the users' perceptions and asking a user to score such objects can provide helpful (discriminatory) information about the system.

Indeed, if a similar number of people gave the same movie high and poor ratings, determining whether the film is good or not for a certain user is difficult. On the other hand, movies with poor ratings from practically all users are generally awful and should not be suggested to anyone. When the system must choose between the two sorts of movies to ask the user to rate, the first will most likely provide more information about the user's tastes than the second.

There are three strategies in this group: variance, entropy, and entropy0.

**Variance [Golbandi and al., 2010]**

This technique favors items with the biggest variation, therefore favoring items that have been evaluated differently by users, on the basis that the variance indicates the system's uncertainty about the item's ratings.

$$Variance(i) = \frac{1}{|U_i|}\sum_{\mathrm{u}\in \mathrm{U_i}}(r_{ui} - \bar{r_u})^2 \qquad (5)$$

where $U_i$ is the set of users who rated item $i$, and $\bar{r_i}$ is the average rating for $i$.

**Entropy [Golbandi and al., 2010]**

Measures the dispersion of the ratings for an item.

$$Entropy(i) = -\sum_{r=1}^{5} p(r_i = r)\log\left(p(r_i = r)\right) \qquad (6)$$

where $p(r_i = r)$ is the probability that a user rate the item $i$ as $r$. The drawback of this strategy is that it tends to select unpopular obscure items, since in general there are many more items with few ratings, for which the entropy may result to be high, but since it is computed on just a few ratings it tends to be unreliable. Hence, while this strategy can identify informative items, it may also pick up rarely rated ones.

**Entropy0 [He and al., 2011]**

Was proposed for solving the mentioned limitation of entropy: the tendency to select unpopular obscure items. Entropy0 addresses this problem by assigning to all the missing ratings a new value equal to 0, hence modifying the rating scale. In this way items heavily rated as 0, i.e., with few ratings, have a small entropy0 score and therefore are not selected.

**Error-reduction**

While items with a wide range of ratings may appear to be useful, they may not be the items with the best ratings for reducing system error. For example, Napoleon Dynamite is a highly controversial film in the Netflix dataset, with widely disparate scores, but it is also a very unique film (its ratings are weakly correlated to ratings of other items). As a result, while the techniques in the previous group may select this item, it is not very useful for calculating the ratings of other things [69]. The active learning solutions discussed in this section directly seek to increase the recommender's predicted accuracy in order to tackle this challenge.

**Greedy Extend [Carenini and al., 2003]**

Tries to identify the items whose ratings (if elicited by the users and added to the training set) yield the lowest system RMSE. Let us denote with A a prediction algorithm (for instance a Factor Model one), and with F(A(L)) the performance of A (e.g., RMSE), when it is trained on a preexistent training set extended with the ratings for the items in the list L. Then greedy extend requests to the user u to rate the items in Lu, which is defined as follows:

$$L_u = argmin_{L\subset \mathrm{l_u}} F(A(L)) \qquad (7)$$

where $I_u$ is the set of all the items that can be rated by $u$. For each candidate item in $I_u$, this strategy computes the reduction of RMSE, before and after adding the rating of the candidate item to the training set. The RMSE metric over the training set can be computed by adopting leave one-out methodology. Then the item with the largest RMSE reduction is added to the set L. This process is repeated, and items are iteratively added until the size of L reaches the maximum number N.

**Attention-based**

This set of strategies focuses on identifying the products that have gotten the most "attention" from users. Users are likely to be familiar with such movies, allowing them to rate them. As a result, these strategies typically add a large number of ratings [Rashid and al., 2002]. These solutions are straightforward and quick to implement, and they were first suggested in collaborative filtering's earliest attempts to overcome the cold start problem [Elahi and al., 2014]. They are regarded as foundational tactics.

**Popularity [Rubens and Sugiyama, 2007]**

Selects the most popular items, i.e., those with the highest number of ratings. It is very likely that the users are able to rate these items and consequently the size of the rating dataset can be increased. However, popular items are typically widely liked by the users. Therefore, their ratings usually bring little information to the system. Moreover, this strategy may cause the prefix bias, i.e., the system trained with ratings for popular items tend also to recommend these popular items, making them even more popular.

**Co-coverage [Carenini and al., 2003]**

Aims at selecting the items that are highly co-rated by the users. Co-coverage is defined as follows:

$$Co - coverage(i) = \sum_{j=1}^{n} m_{ij} \quad (8)$$

where $m_{ij}$ is the number of users who rated both item $i$ and item $j$. By selecting the items with high co-coverage, this strategy tries to identify items that are useful for collaborative filtering based predictions, which are based on patterns of co-rated items. It is important to stress that co-coverage is influenced by the items' popularity, and hence, it can request many popular items.

These strategies can be combined to obtain combined-heuristic strategies.

## 4.2 Personalized active learning

**Acquisition probability based**

The strategies in this group are aimed at boosting the system's performance by increasing the likelihood that the selected items are familiar to the user and hence ratable. These tactics personalize rating requests by recommending various products to different users.

**Item–Item [Rashid and al., 2002]**

The items that are most similar to the user's previously rated items are chosen. As a result, after receiving at least one rating from a user, the similarities between the user rated item(s) and the remaining unrated items are calculated, and the items that are most similar to the rated ones are offered to the user to rate. Like user-to-user similarity, item-to-item similarity is calculated using Pearson Correlation.

$$sim(i,j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)^2 (r_{uj} - \bar{r}_j)^2}} \quad (9)$$

where $U_{ij}$ denotes the subset of users who co-rated items $i$ and $j$, $\bar{r}_i$ denotes the average rating of item $i$. Since this strategy does not consider the informativeness of the items, it can acquire ratings that do not improve the system in terms of prediction accuracy.

**Binary Prediction [Elahi and al., 2014]**

It first transforms the rating matrix into a new matrix with the same number of rows and columns, by mapping null entries to 0, and not null entries to 1. Hence, this new matrix models only whether a user rated an item or not, regardless of its value. Then, using a factor model, a prediction for all the ratings initially set to 0 is computed and for each user, and the items with the highest predicted score are selected. This strategy tries to identify the items that the user is more likely to have experienced, in order to maximize the chances that the user can provide the requested rating. In that sense it is analogous to the popularity strategy, but it tries to make a better (personalized) prediction of what items the user can rate by exploiting the rating knowledge of each user.

**Impact based**

Obtaining ratings for the items about which the system is less certain can help the system better predict the ratings of these particular items. However, this will not always result in a reduction in the rating prediction's uncertainty for other goods. Impact-based techniques choose elements in an attempt to reduce the overall rating prediction uncertainty.

**Influence Based [Rubens and Sugiyama, 2007]**

This strategy estimates the influence of item ratings on the rating prediction of other items and selects the items with the largest influence. First the rating prediction $\hat{r}_{ui}$ for user u and an unrated item $i$, is computed. Then, the rating prediction $\hat{r}_{ui}$ is decreased by 1 unit: $\hat{r}'_{ui} = \hat{r}_{ui} - 1$ (e.g., rating 5 is changed to 4). Finally, two prediction models are generated, one adding $\hat{r}_{ui}$ and another adding $\hat{r}'_{ui}$ to the training set, and the absolute value of the differences of their predictions for the ratings of all the items different from $i$ are computed. The influence of $i$ is estimated by summing up all these differences. Finally, the items with the highest influence are selected for active learning.

# 5 Recursive Prediction Algorithm for Collaborative Filtering Recommender Systems

The conventional prediction process of the user-based CF approach selects neighbor users using two criteria: 1) they must have rated the given item; 2) they must be quite close to the active user (for instance, only the top K nearest neighbor users are selected). However, most users in recommender systems are unlikely to have rated many items before starting the recommendation process, making the training data very sparse. As a result, the first criterion may cause a large proportion of users being filtered out from the prediction process even if they are very close to the active

user. This in turn may aggravate the data sparseness problem.

A recursive prediction algorithm that relaxes the first condition listed above to overcome the data sparseness problem and allow additional people to participate in the prediction process is proposed. If a nearest-neighbor user hasn't rated the given item yet, we'll first recursively estimate the rating value for him/her based on his/her own nearest-neighbors, and then use the estimated rating value to join the prediction process for the last active user. As a result, we will have more data to contribute to the prediction process, and the prediction accuracy of collaborative filtering recommender systems should improve.

## 5.1 The Strategies for Selecting Neighbors

In summary, we propose the following five strategies for selecting the active user's nearest neighbors:

**Baseline strategy (BS)**

Selects the top K nearest-neighbors who have rated the given item. This is the conventional strategy for selecting neighbors as illustrated in [Resnick and al., 1994].

**Baseline strategy with overlap threshold (BS+)**

Selects the top K nearest-neighbors who have rated the given item and have rated at least ϕ items that have also been rated by the active user (overlapped with the active user).

**Similarity strategy (SS)**

Selects the top K' nearest neighbors purely according to their similarity with the active user.

**Combination strategy (CS)**

Combines top K nearest neighbors selected by the baseline strategy (BS) and top K' nearest neighbors selected by the similarity strategy (SS).

**Combination strategy with overlap threshold (CS+)**

combines the top K nearest-neighbors selected by baseline Strategy (BS) and top K' nearest-neighbors selected by the similarity strategy (SS). Also, each user must have rated at least ϕ items overlapped with the active user.

## 5.2 Prediction Computation

Once the user similarity and the subset of neighbor users are determined, we need to aggregate their rating information to generate the prediction value [Resnick and al., 1994]. Formally, for the active user x to the given item $i$, the predicted rating $\hat{r}_{x,i}$ can be calculated as following:

$$\hat{r}_{x,i} = \bar{r}_x + \frac{\sum_{y \in U_x}(r_{y,i} - \bar{r}_y) sim(x,y)}{\sum_{y \in U_x} |sim(x,y)|} \quad (10)$$

where $U_x$ represents the subset of neighbor users selected for the active user x. The similarity value $sim(x, y)$ can be calculated according to Equation 9 and it acts as a weight value on the normalized rating value $r_{y,i} - \bar{r}_y$. In the conventional CF approach, only those neighbor users who have rated the given item explicitly are selected, so the value $r_{y,i}$ can be fetched directly from the training dataset.

## 5.3 The Recursive Prediction Algorithm

The goal of the recursive prediction algorithm is to include nearest neighbors who haven't rated the given item in the prediction process. When the process requires a rating value that doesn't exists in the dataset, we can estimate it recursively on the fly, and then use it in the prediction process. The estimated rating values may not be as accurate as those ratings explicitly given by the users. In the algorithm a weight value is specified to distinguish the different contribution of these two types of ratings. Formally, the recursive prediction algorithm can be represented as the following:

**Algorithm 2** The recursive prediction algorithm [Zhang and Pu, 2007]

**Configuration Values**:
ζ : the threshold of recursive level
λ : the combination weight threshold

**Input**:
$x$ : the active user
$i$ : the given item to be predicted
$level$ : the current recursive level

**Output**: the predicted rating value

```
1:  function RecursivePrediction (x, i, level)
2:      if level ≥ ζ then
3:          return BaselinePrediction (x, i);
4:      endif
5:      U ← SelectNeighbors (x, i);
6:      α = 0.0;
7:      β = 0.0;
8:      for each y in U do
9:          if r_{y,i} is given then
10:             α + = (r_{y,i} − r̄_y) sim(x, y);
11:             β + = |sim(x, y)|;
12:         else
13:             r̂_{x,i} = RecursivePrediction (y, i, level + 1);
14:             α + = λ(r̂_{y,i} − r̄_y) sim(x, y);
15:             β + = λ|sim(x, y)|;
16:         endif
17:     endfor
18:     return r̄_x + α/β
```

where $U_x$ is the set of neighbor users for the active user $x$ determined by the respective strategy we have mentioned earlier. If $r_{y,i}$ is not given explicitly from the training dataset, the recursively predicted value $\hat{r}_{y,i}$ instead is applied. The weight threshold λ is a value between [0, 1].

The function "SelectNeighbors $(x, i)$" is the implementation of one of the 5 different selection strategies.

Note that if the recursive algorithm has reached the predefined maximal recursive level, it will stop the recursive procedure and call the "BaselinePrediction" function instead.

The BaselinePrediction function can be implemented by various strategies – we use the conventional baseline strategy (BS).

# 6 Recommender system using active learning and recursive algorithm Implementation

The implementation of our recommender system is based on a fusion between active learning to choose the items or movies to be presented to our active user i.e., the new user having no ratings to be based on for item recommendation, and recursive algorithm to overcome the data sparseness problem and allow additional people to participate in the prediction process so the nearest neighbors can give us more information on their taste of items and help for better predicted ratings of our active user. So, this implementation will present a collaborative filtering recommender system based on two layers to get more accurate recommendations for new users that the system still not acquired enough data about.

Our experiments are executed on Google colab on 0,90GB memory and one CPU of 2.30GHZ under the Linux operating system.

## 6.1 Preprocessing data

Data preprocessing in Machine Learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for a building and training Machine Learning models. The data set used is from [MovieLens], a movie recommendation service. In this first part, we download the dataset, then try to use the different data frames to get a matrix or a new data frame containing the information we need to run our algorithm.

In our case we need a matrix containing users as rows and movies or items to rate as column presenting the available rating et NAN i.e., Not a Number if the rating for item $i$ by user $u$ is not available.

## 6.2 Implementation of an active learning strategy

In our presented code we used the variance strategy. Indeed, this strategy selects the items with the highest variance, hence, it favors the items that have been rated diversely by the users on the assumption that the variance gives an indication of the uncertainty of the system about that item's ratings.

We set a function to calculate the variance for each item and then we get the list of items sorted by variance. The use will be asked for the number of movies he wants to rate in the subscribing process and the algorithm will propose him the first $k$ movies from that list and add his rating to the built matrix.

## 6.3 Nearest neighbor based collaborative filtering implementation

To predict the rating value of a given item for an active user, a subset of neighbor users are chosen based on their similarity to the active user – called nearest-neighbor users – and their ratings of the given item are aggregated to generate the prediction value for it.

**User similarity**

We choose the Pearson correlation as the metric for user similarity. First, we set a function taking as input two user returns a list of the items rated by both passed in. Then we set another function taking a user as input et returning the average rating of that user. And finally, we can set a similarity function that returns the similarity between two users using the Pearson correlation as the metric.

**Selecting Neighbors**

We will use the Baseline Selecting Neighbors strategy "BS". To implement this strategy, we need a function taking as input an item $i$ and returning a list of users who have rated that same item $i$, and another function that returns the top K nearest-neighbors to the active user $x$ have rated the item i sorted by similarity with $x$.

## 6.4 Implementation of the recursive algorithm

We start by setting up a function returning the prediction on item $i$ for the active user x based on baseline strategy. This function will be used as mentioned in the algorithm 2 in the recursive prediction function.

Then, based on the literature [paper 2] we set up the configuration parameters as following: Set of neighbors Size: k=10, threshold = 2, and weight Threshold: lambda = 0.5. The function input would be the active user $x$, the item $i$, and the $level$.

Finally, we will set our main program by looping over the set of movies selected by variance to get the to predict the ratings of the non-rated items by our new user based on the nearest neighbors actual and predicted ratings.

The whole implementation of this approach for recommender system is presented in this GitHub repository: https://github.com/haith-gi/Recommender-system-atelier-IA- .

# 7 Conclusion

In this work we present the state of art relative to collaborative filtering, active learning and recursive algorithm used in recommender systems. We also introduce an approach based on two layers to get more accurate recommendations of movies or items for new users while the system did not acquire enough data on them. An active learning strategy was used to select the most helpful items to ask the new user to rate in the subscribing phase, and the recursive algorithm was used to allow additional people to participate in the prediction process based on the nearest neighbors collaborative filtering approach.

Multiple executions by changing the different parameters with an analysis of the results found each time could be subject of a future study.

## References


[Adomavicius and Tuzhilin, 2005] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 734–749. http://dx.doi.org/10.1109/TKDE.2005.99.

[Bishop, 2006] C.M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Carenini and al., 2003] G. Carenini, J. Smith, and D. Poole. Towards more conversational and collaborative recommender systems, in: Proceedings of the 8th International Conference on Intelligent User Interfaces, IUI'03, ACM, New York, NY, USA, 2003, pp. 12–18. http://dx.doi.org/10.1145/604045.604052.

[Chang and al., 2015] S. Chang, F.M. Harper, and L. Terveen. Using groups of items for preference elicitation in recommender systems, in: Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, CSCW'15, ACM, New York, NY, USA, 2015, pp. 1258–1269. http://dx.doi.org/10.1145/2675133.2675210

[Desrosiers and Karypis, 2011] C. Desrosiers, G. Karypis. A comprehensive survey of neighborhood-based recommendation methods, in: F. Ricci, L. Rokach, B. Shapira, P.B. Kantor (Eds.), Recommender Systems Handbook, Springer, 2011, pp. 107–144. http://dx.doi.org/10.1007/978-0-387-85820-3_4.

[Elahi and al., 2014] M. Elahi, F. Ricci, and N. Rubens. Active learning strategies for rating elicitation in collaborative filtering: A system-wide perspective, ACM Trans. Intell. Syst. Technol. 5 (1) (2014) 13:1–13:33. http://dx.doi.org/10.1145/2542182.2542195.

[Elahia and al., 2015] Mehdi Elahia, Francesco Riccib and Neil Rubensc, A survey of active learning in collaborative filtering recommender systems.

[Golbandi and al., 2010] N. Golbandi, Y. Koren, and R. Lempel. On bootstrapping recommender systems, in: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, CIKM'10, ACM, New York, NY, USA, 2010, pp.1805–1808. http://dx.doi.org/10.1145/1871437.1871734

[He and al., 2011] L. He, N.N. Liu, and Q. Yang. Active dual collaborative filtering with both item and attribute feedback, in: W. Burgard, D. Roth (Eds.), AAAI, AAAI Press, 2011, pp. 1186–1191. URL: http://dblp.uni trier.de/db/conf/aaai/aaai2011.html#HeLY11.

[Koren and Bell, 2015] Y. Koren, R. Bell. Advances in collaborative filtering, in: Recommender Systems Handbook, Springer, 2015, pp. 77–118. http://dx.doi.org/10.1007/978-1-4899-7637-6_3.

[Kohrs and Merialdo] A. Kohrs, B. Merialdo Improving collaborative filtering for new users by smart object selection, in: Proceedings of International Conference on Media Features, ICMF, 24.

[Mitchell, 1997] T.M. Mitchell. Machine Learning, first ed., McGraw-Hill, Inc., New York, NY, USA, 1997.

[Rashid and al., 2002] A.M. Rashid, I. Albert, D. Cosley, S.K. Lam, S.M. McNee, J.A. Konstan, and J. Riedl. Getting to know you: Learning new user preferences in recommender systems, in: Proceedings of the 7th International Conference on Intelligent User Interfaces, IUI'02, ACM, New York, NY, USA, 2002, pp. 127–134. http://dx.doi.org/10.1145/502716.502737.

[Rashid and al., 2008] A.M. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems: an information theoretic approach, SIGKDD Explor. Newsl. 10 (2008)90–100. http://dx.doi.org/10.1145/1540276.1540302.

[Resnick and al., 1994] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[Rubens and Sugiyama, 2007] N. Rubens, M. Sugiyama. Influence-based collaborative active learning, in: Proceedings of the 2007 ACM Conference on Recommender Systems, RecSys'07, ACM, New York, NY, USA, 2007, pp. 145–148. http://dx.doi.org/10.1145/1297231.1297257.

[Tong, 2001] S. Tong. Active learning: Theory and applications (Ph.D. thesis), The Department of Computer Science, 2001.

[Zhang and Pu, 2007] Jiyong Zhang and Pearl Pu. A Recursive Prediction Algorithm for Collaborative Filtering Recommender Systems. Conference Paper · January 2007.