

User Modeling and Personalization

6: Collaborative Filtering Recommender Systems

Eelco Herder

L3S Research Center / Leibniz University of Hanover
Hannover, Germany

9 May 2016

Outline I

Collaborative Filtering Recommendation Systems

Recommender Systems

- Collaborative Filtering and Personalization

- Applications of Collaborative Filtering

- Functionality of CF systems

- Assumptions behind collaborative filtering

Acquiring User Ratings

- Explicit and implicit ratings

- Rating Scales

- Cold Start Issues

Collaborative Filtering Algorithms

User-Based Neighbor Algorithms

- Preparation of the input data

Outline II

- Measuring similarity between users
- Selecting the most similar users
- Generating recommendations
- Discussion

Item-Based Neighbor Algorithms

- Calculating the similarity between items
- Generating Recommendations
- Discussion

Recommender Systems

Recommender systems work from a specific type of information filtering system technique that attempts to recommend items (movies, TV program/show/episode, video on demand, music, books, news, images, web pages, scientific literature such as research papers etc.) that are likely to be of interest to the user.

Collaborative Filtering (CF)

Collaborative Filtering is the process of filtering or evaluating items using the opinions of other people. Collaborative filtering systems produce predictions or recommendations for a given user and one or more items.

Items can consist of anything for which a human can provide a rating:

- ▶ art
- ▶ books
- ▶ CDs
- ▶ journal articles
- ▶ vacation destinations
- ▶ ...

Motivation

For years, people have stood over the back fence or in the office break room and discussed books they have read, restaurants they have tried, and movies they have seen. And they used these discussions to form opinions.

At some point, you might observe that among your friends:

- ▶ Matt recommends the types of films that you like
- ▶ Paul typically recommends films that you despise
- ▶ And Margaret simply recommends everything.

Over time, you learn whose opinions you should listen to.

Schafer, J.B., Frankowski, D., Herlocker, J. and Sen, S. (2007) Collaborative Filtering Recommender Systems. The Adaptive Web, 291-324

Collaborative Filtering and Personalization

Early collaborative filtering systems were designed to explicitly provide users with recommendations for items (users visited the system only to receive these recommendations).

Later, websites began to use CF systems behind the scenes to adapt their content to users

- ▶ which news items to display prominently
- ▶ what information are users likely to want to see

When is recommendation useful

Collaborative filtering is a popular personalization mechanism in websites with many items, such as online shops and online libraries, where:

- ▶ the number of items is too high to be covered by (hand-crafted) adaptation rules
- ▶ the set of items may be very dynamic (e.g. news articles)
- ▶ personalization is needed for the user in order to maintain an overview

“If I have 3 million customers on the Web, I should have 3 million stores on the Web”. Jeff Bezos, amazon.com

User Tasks

Tasks for which people use collaborative filtering include:

- ▶ *Help me find new items I might like.*

Most common application area is consumer items (music, books, movies), but also for research papers, web pages or other ratable items.

- ▶ *Advice me on a particular item.*

Does the community know whether it is good or bad?

- ▶ *Help me find users I might like.*

Forming discussion groups, matchmaking or connecting users so that they can exchange recommendations socially.

- ▶ *Help me find a mixture of new and old items*

For example, restaurant recommendations that include ones in which I have eaten previously.

Functionality of CF systems

The following main families of common CF system functionality can be distinguished:

Recommend items

Show a list of items to a user, in order of how useful they might be.

In some systems, the rating associated with the item is the predicted user rating. Other systems, such as Amazon, show the average customer rating instead.

Picking the top few items well is crucial; the predicted rating is of secondary importance.

Predict for a given item

To provide predictions for a particular item, a system must be prepared to say something about any requested item, even rarely rated ones.

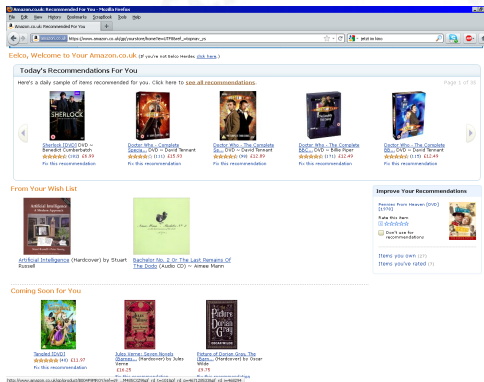
Personalized predictions may be challenging!

Constrained recommendations

Given a particular set or a constraint that gives a set of items, recommend from within that set.

For example, personalized search results or movie recommendations in a particular genre, of a particular length and for a particular age group.

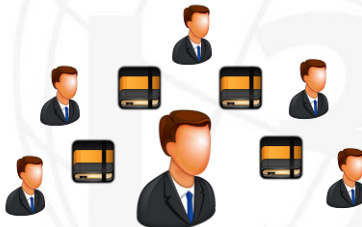
Collaborative filtering recommendations in Amazon



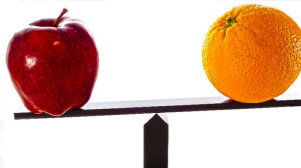
Assumptions behind collaborative filtering

Collaborative filtering recommender systems work only when certain assumptions are sufficiently met:

- ▶ There are other users with common needs or tastes
- ▶ People with similar tastes will rate things similarly
- ▶ Taste persists: CF has been successful for movies, books and electronics. If tastes change frequently, older ratings may be less useful (e.g. clothing)



- ▶ Item evaluation is personal: if objective (content-based) criteria for goodness are more relevant, collaborative filtering may not be very useful
- ▶ Items persist long enough to receive sufficient ratings: news stories are only important for a short time, which hinders CF
- ▶ Items are sufficiently homogeneous: for example music albums. Recommendations such as 'if you buy a hammer, you might also want to buy a refrigerator' are not very useful.



Acquiring User Ratings

Explicit ratings provided by users offer the most accurate description of a user's preference for an item.

- ▶ require additional work from the user
- ▶ in return, users get higher quality recommendations
- ▶ other motivations for rating include goodwill, having one's opinion's voiced and valued, and the ability to store their own likes and dislikes
- ▶ bootstrapping a system needs a relatively small number of "early adopters" who rate frequently and continuously
- ▶ CF systems may use incentives (e.g. presents) to encourage users to provide more explicit ratings

In some domains - most notably hotel booking sites - users are particularly willing to express their opinions.

4.3 OUT OF 5


90% of guests recommend

Room cleanliness 4.6

Service & staff 4.4

Room comfort 4.1

Hotel condition 4.3



VERIFIED REVIEWS
 They paid and stayed. We double-checked.

See reviews for:

☐ Families
 ☐ Couples
 ☐ Business Travelers
 ☐ Students
 ☐ Other

Sort by: **Newest**

4.0

for everyone

by

Mitch

Recommended

Santa Paula, CA

Old-world hotel in a quiet location

Posted June 8, 2013

Location, price, included buffet breakfast.

Real queen or king beds would be nice, instead of two twins shoved together. Not a big issue for us, but others might be offended.

Via Veneto, Villa Borghese, Spanish Steps (about half mile walk)

Implicit ratings are collected with little or no cost to the user

- ▶ may be based on the time spent reading information about a product
- ▶ or based on the products that the user actually bought, bookmarked or added to a wish list
- ▶ if implicit ratings are used, there is more uncertainty in the computation

The more ratings you have, the more uncertainty in the ratings you can handle: multiple implicit ratings can be combined in a single estimated rating by averaging or voting.

Rating scales

Different kinds of rating scales can be found on the Internet (and elsewhere), such as :

- ▶ A simple like-button: a boolean scale with two values (I like it or not)
- ▶ Five-star ratings: very popular in online stores and social networks
- ▶ Slider bars: allow for very fine-grained scales, such as 1-100

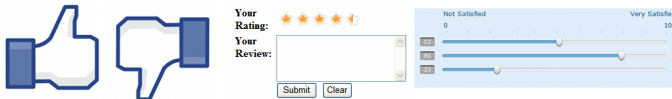


Your Rating: ★ ★ ★ ★ ★
Your Review:

Not Satisfied 0 Very Satisfied 100

Boolean scales do not give the users sufficient possibilities to express their opinions. Boolean scales are also often too coarse-grained for collaborative filtering.

Very fine-grained scales may confuse the user: do I like 'The Hobbit' 44% or rather 47%? Fine-grained scales may make it unlikely to find users that give (exactly) the same rating to a particular item.



Cold Start Issues

Cold start problem

The cold-start problem describes situations in which a recommender is unable to make meaningful recommendations due to an initial lack of ratings.

New User. When a user registers to a system, he has no ratings on record, so no personalized predictions can be given. Possible solutions:

- ▶ having the user rate some initial items upon registering
- ▶ displaying non-personalized recommendations (for popular items) until the user has rated enough
- ▶ asking the user to describe their taste in aggregate or for demographic information (stereotyping approach)

New Item. Newly added items have no ratings, so they will not be recommended. In these situations, content-based techniques can be used (generating recommendations based on metadata such as author, genre or production year).

New Community. Bootstrapping a community is the biggest cold-start problem. Apart from initially using non-CF approaches, a common solution is to provide *rating incentives* to initial users before inviting the entire community to use the service.

How does collaborative filtering work?

User-based algorithms require all ratings, items and users be stored in memory.

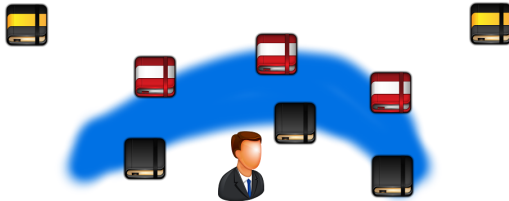
- ▶ Identify the users who bought or liked the same items as you did (the *neighborhood*)
- ▶ Recommend items that the neighborhood bought or liked best

Memory-based algorithms do not scale well. Therefore, almost all practical algorithms use some form of pre-computation.



Item-based algorithms periodically create a summary of rating patterns offline.

- ▶ Identify the items that are liked by all other users the same way as you like them
- ▶ Recommend items that are most similar to the ones that you like best



User-Based Neighbor Algorithms

User-Based Neighbor Algorithms

User-based algorithms generate a prediction for an item i by analyzing ratings for i from users in u 's neighborhood. The neighborhood of u consists of users with similar rating behavior.

User-based recommendations are created with the following four steps:

1. **Preparation of the input data**

- ▶ a User x Item matrix

2. **Measurement of similarity between users**

- ▶ *for example:*
- ▶ quadratic similarity
- ▶ cosine similarity
- ▶ Pearson correlation

3. **Selection of the most similar users**

- ▶ by setting a similarity threshold
- ▶ by setting the minimum or maximum size of the neighborhood

4. **Generation of recommendations**

- ▶ based on the ratings of items in the user's neighborhood

Step 1: Preparation of the input data

- ▶ a simple user-item matrix
- ▶ ratings are on a 1-5 scale
- ▶ cells corresponding to not-rated items remain empty

User/Item	I_1	I_2	I_3	I_4	I_5	I_6
U_1	5		3		4	
U_2	1	1		1		
U_3	1		3	1		
U_4	5	2	2		5	4
U_5		3		2		

Step 2: Measurement of similarity between users

Let us define the following parameters:

- ▶ u, v are two users to be compared
- ▶ I are the ratable items in the systems
- ▶ $u[i]$ and $v[i]$ are ratings of users u and v of an item i from I
- ▶ \vec{u} and \vec{v} are the vectors of the corresponding user ratings of items that have been rated by both u and v
- ▶ \bar{u} and \bar{v} is the average of the above-mentioned ratings:
$$\bar{u}[i] = \frac{1}{n} \sum_{l=1}^n u[l] \text{ and } \bar{v}[i] = \frac{1}{n} \sum_{l=1}^n v[l], \text{ with } n \text{ the number of items that have been rated by both } u \text{ and } v$$

Quadratic distance: (a very basic distance measure, range $[0:\infty)$)

$$quad(u, v) = \frac{(\vec{u} - \vec{v})^2}{n}$$

- ▶ smaller values indicate more similarity

Cosine similarity: (the angle between the two rating vectors, range $[0:1]$)

$$cosim(u, v) = \frac{\vec{u} * \vec{v}}{|\vec{u}| * |\vec{v}|}$$

- ▶ an established similarity measure in the fields of information retrieval and text mining
- ▶ values closer to 1 indicate more similarity

Pearson correlation: (linearity between the ratings of the two users, range [-1:1])

$$r(u, v) = \frac{(\vec{u} - \bar{\vec{u}}) * (\vec{v} - \bar{\vec{v}})}{|(\vec{u} - \bar{\vec{u}})| * |(\vec{v} - \bar{\vec{v}})|}$$

- ▶ a standard similarity (or actually a correlation) measure from the field of statistics
- ▶ values closer to 1 indicate more similarity, values closer to -1 indicate opposite tastes

Remarks

- ▶ for all similarity measures, one needs a minimum number of items rated by both users
- ▶ theoretically, cosine similarity functions works starting from two items
- ▶ but it would be better to define a minimum number of items
- ▶ for Pearson correlation, significance of the correlation is formally defined as $t = \frac{r}{\sqrt{\frac{1-r^2}{n-2}}}$

Example

Calculation of the similarity of U_1 with all other users U_2, U_3, U_4 and U_5 .

User/Item	I_1	I_2	I_3	I_4	I_5	I_6
U_1	5		3		4	
U_2	1	1		1		
U_3	1		3	1		
U_4	5	2	2		5	4
U_5		3		2		

User/Sim	$quad(U_1, *)$	$cosim(U_1, *)$	$r(U_1, *)$
U_2	16	1	0
U_3	8	0,76	-1
U_4	$\frac{2}{3}$	0,98	0,866
U_5	∞	∞	∞

$$\begin{aligned} quad(U_1, U_4) &= \frac{(\vec{U}_1 - \vec{U}_4)^2}{n} = \frac{((5, 3, 4) - (5, 2, 5))^2}{3} = \\ &= \frac{(5 - 5, 3 - 2, 4 - 5)^2}{3} = \frac{2}{3} \end{aligned}$$

$$\begin{aligned} cosim(U_1, U_4) &= \frac{\vec{U}_1 * \vec{U}_4}{|\vec{U}_1| * |\vec{U}_4|} = \\ &= \frac{(5, 3, 4) * (5, 2, 5)}{\overrightarrow{(5, 3, 4)} * \overrightarrow{(5, 2, 5)}} = \frac{(5 * 5 + 3 * 2 + 4 * 5)}{\sqrt{(25 + 9 + 16)} * \sqrt{(25 + 4 + 25)}} = \\ &= \frac{51}{\sqrt{50} * \sqrt{54}} \approx 0.98 \end{aligned}$$

$$\overline{U_1} = \frac{5 + 3 + 4}{3} = 4, \overline{U_4} = \frac{5 + 2 + 5}{3} = 4$$

$$r(U_1, U_4) = \frac{(\vec{U_1} - \overline{\vec{U_1}}) * (\vec{U_4} - \overline{\vec{U_4}})}{|\vec{U_1} - \overline{\vec{U_1}}| * |\vec{U_4} - \overline{\vec{U_4}}|} =$$

$$\frac{(5 - 4, 3 - 4, 4 - 4) * (5 - 4, 2 - 4, 5 - 4)}{|(5 - 4, 3 - 4, 4 - 4)| * |(5 - 4, 2 - 4, 5 - 4)|} =$$

$$\frac{(1, -1, 0) * (1, -2, 1)}{\sqrt{1 + 1 + 0} * \sqrt{1 + 4 + 1}} = \frac{(1 * 1 + (-1) * (-2) + 0 * 1)}{\sqrt{2} * \sqrt{6}} =$$

$$\frac{3}{\sqrt{2} * \sqrt{6}} \approx 0.866$$

Step 3: Selection of the most similar users

Select the set S of users that are *sufficiently* similar to user U , to be used as a base for recommendations.

There are several ways of doing so:

Similarity threshold

S contains all users with a similarity to user U higher than a predefined threshold t

Predefined number

S contains the top- k users with the highest similarity to user U

Aggregate neighborhood

- ▶ construct a set S using the similarity threshold method
- ▶ if the set does not contain sufficient users, calculate the *centroid* of the set and add users that are sufficiently similar to this centroid
(the centroid of the set is a vector containing the average ratings of all rated items in the set)
- ▶ repeat until the set contains at least k users

Step 4: Generation of recommendations

In this final step we predict the items $i \in I$ that user U probably will be interested in.

The prediction is based on the set S of users most similar to U and the ratings $s[i]$ of item i , provided by s from S . S_i is the number of users $s \in S$ who rated item i .

Each recommender r generates an ordered list of predicted ratings $r(U, i)$ (high to low), of which the top- k of items that have not yet been rated by user U may be presented to the user.

Recommender 1: Simple Average

For each item i , the predicted value is the average of all neighbors' ratings for this item.

$$r_1(U, i) = \frac{1}{|S_i|} \sum_{s \in S_i} s[i]$$

This method does not take into account that some members of S are more similar to U than other members. The following recommender take this into account:

Recommender 2: Weighted Average

For each item i , the predicted value is the average of all neighbors' ratings for this item, weighted according to the similarity of each user s to user U (and normalized by the sum of the neighbors' similarities)

$$r_2(U, i) = \frac{1}{\sum_{s \in S_i} \text{sim}(U, s)} \sum_{s \in S_i} \text{sim}(U, s) * (s[i])$$

This method does not take into account that some members of S may consistently provide higher (more optimistic) ratings than other members. The next recommender also takes that into account:

Recommender 3: Weighted Normalized Average

For each item i , the predicted value is the average of all neighbors' ratings for this item, normalized with respect to the average rating \bar{s} of s and the similarity of each user s to user U

$$r_3(U, i) = \bar{U} + \frac{1}{\sum_{s \in S_i} \text{sim}(U, s)} \sum_{s \in S_i} \text{sim}(U, s) * (s[i] - \bar{s})$$

Example (continuation of the previous example)

Generate recommendations for user U_1 . The neighborhood S of similar users is given by $S = \{U_3, U_4\}$. We use the cosine similarity for determining similarity between users.

User/Item	I_1	I_2	I_3	I_4	I_5	I_6	$\text{cosim}(U_1, U_i)$	$\overline{U_i}$
U_1	5		3		4		–	4
U_2	1	1		1			1	1
U_3	1		3	1			0,76	$\frac{5}{3}$
U_4	5	2	2		5	4	0,98	$\frac{18}{5}$
U_5		3		2			∞	1.5
$r_1(U_1, i)$	-	2	-	1	-	4		
$r_2(U_1, i)$	-	2	-	1	-	4		
$r_3(U_1, i)$	-	2.4	-	3.33	-	4.4		

- ▶ Recommender 1 will recommend items: I_6, I_2 and I_4 (in this particular order)
- ▶ Recommender 3 will recommend items: I_6, I_4 and I_2

Similarity calculations

$$r_2(U_1, I_2) = \frac{1}{0.98} * (0.98 * 2) = 2$$

$r_2(U_1, I_4)$ and $r_2(U_1, I_6)$ are just as trivial, in this case.

$$r_3(U_1, I_2) = 4 + \frac{1}{0.98} * (0.98 * (2 - \frac{18}{5})) = 4 + (2 - 3.6) = 2.4$$

$$r_3(U_1, I_4) = 4 + \frac{1}{0.76} * (0.76 * (1 - \frac{5}{3})) = 4 + (1 - \frac{5}{3}) \approx 3.33$$

$$r_3(U_1, I_6) = 4 + \frac{1}{0.98} * (0.98 * (4 - \frac{18}{5})) = 4.4$$

Discussion

The user-based neighbor algorithm captures how word-of-mouth recommendation sharing works and it can detect complex patterns given enough users.

The original implementation is *memory-based*, as it is hard to calculate all user similarities offline (if new ratings are provided, all similarities need to be recalculated)

The algorithm does not incorporate *agreement* about an item: if two users agree about a universally loved movie, this is much less important than agreement for a controversial movie.

Rating data is often *sparse*: similarity between users may be based on only a small number of co-ratings. It is not uncommon that these users with a high similarity based on only 1-3 items dominate the user's neighborhood.

The algorithm does not *scale* well for large numbers of items and/or users. For example, Amazon.com has tens of millions of customers. It would be immensely resource-intensive to scan the ratings of millions of customers to return recommendations in less than a fraction of a second.

Possible solutions:

- ▶ *Subsampling*: a subset of users is selected prior to prediction computation. Neighborhood computation time remains fixed.
- ▶ *Clustering*: well-known clustering methods (e.g. k-means clustering) can quickly discover a set of users similar to the current user, which can be used for neighborhood computation.

Item-Based Neighbor Algorithms

Item-Based Neighbor Algorithms

Item-based algorithms generate a prediction for an item i based on its similarity (in terms of ratings) to items already rated by user u .

The basic idea is that the user-item matrix is transformed into an item-item matrix.

An item-item matrix is a $|I| \times |I|$ matrix, in which each cell x, y represents the similarity between two items x and y .

Given sufficient ratings, the similarities between items are more stable than the similarities between users. Therefore, the similarities can be calculated offline (on a regular basis).

Calculating the similarity between items

Let us define the following parameters:

- ▶ i, j are two items
- ▶ $N = \{U_1, \dots, U_n\}$ is the set of users who rated both i and j
- ▶ $r_i[U]$ is the rating of user $U \in N$ for item i
- ▶ \vec{r}_i and \vec{r}_j are the vectors of all user ratings for i and j
- ▶ $\bar{\vec{r}}_i$ and $\bar{\vec{r}}_j$ are the average values of the above-mentioned vectors:

Similar to the user-based approach, we define the following similarity measures:

Cosine similarity: (the angle between the two item vectors, range [0:1])

$$\text{cosim}(i, j) = \frac{\vec{r}_i * \vec{r}_j}{|\vec{r}_i| * |\vec{r}_j|}$$

Pearson correlation: (linearity between the ratings of the two items, range [-1:1])

$$r(i, j) = \frac{(\vec{r}_i - \bar{\vec{r}}_i) * (\vec{r}_j - \bar{\vec{r}}_j)}{|(\vec{r}_i - \bar{\vec{r}}_i)| * |(\vec{r}_j - \bar{\vec{r}}_j)|}$$

Adjusted Cosine Similarity

The regular cosine similarity measure does not take into account that some users rate items more positive (optimistic) than others.

Instead of taking each user's rating $r_i[U]$ as the basis for building the item rating vector \vec{r}_i , it uses the difference between the individual user's rating for this item with their average rating value.

Adjusted-cosine similarity is the most popular (and believed to be the most accurate) item-item similarity metric.

Adjusted Cosine Similarity - more formally

- ▶ we define $U_{k,avg}$ as the average of all ratings of user $U_k \in N$
- ▶ $\vec{r_U}$ consists of $\vec{r_U}[k] = U_{k,avg} \quad \forall k \in [1 : n]$
in other words: this is the vector of the average ratings of all users.

The adjusted cosine is defined as follows:

$$adj_cosim(i, j) = \frac{(\vec{r_i} - \vec{r_U}) * (\vec{r_j} - \vec{r_U})}{|(\vec{r_i} - \vec{r_U})| * |(\vec{r_j} - \vec{r_U})|}$$

Example: Constructing the Item x Item Matrix.

We use cosine similarity.

User/Item	I_1	I_2	I_3	I_4	I_5	I_6
U_1	5		3		4	
U_2	1	1		1		
U_3	1		3	1		
U_4	5	2	2		5	4
U_5		3		2		

Item/Item	I_1	I_2	I_3	I_4	I_5	I_6
I_1	-	0,96476	0,83591	1	0,99388	1(*)
I_2	0,96476	-	1(*)	0,98995	1(*)	1(*)
I_3	0,83591	1(*)	-	1(*)	0,95293	1(*)
I_4	1	0,98995	1(*)	-	n.a.	n.a.
I_5	0,99388	1(*)	0,95293	n.a.	-	1(*)
I_6	1(*)	1(*)	1(*)	n.a.	1(*)	-

(*): in these cases, the cosine similarity is 1 because it is based on the ratings of one individual user. *n.a.*: there is no user who rated both items

Generating Recommendations

As the similarity matrix already takes the user ratings into account, the procedure for generating recommendations is pretty straightforward:

1. Build item-item-similarity vectors from all rows J from the item-item matrix corresponding to items rated by the user
2. Build the similarity vector by taking the (weighted) average of item-item similarities $itemsim_{i,j}$ in each row j from J
3. Remove the predicted ratings for the items that the user has already rated before
4. Sort all remaining predictions in descending order and recommend the top- k to the user

Example (continued)

We generate recommendations for U_1 , who rated items I_1, I_3 and I_5 .

Item/Item	I_1	I_2	I_3	I_4	I_5	I_6
I_1	-	0,96476	0,83591	1	0,99388	1(*)
I_2	0,96476	-	1(*)	0,98995	1(*)	1(*)
I_3	0,83591	1(*)	-	1(*)	0,95293	1(*)
I_4	1	0,98995	1(*)	-	n.a.	n.a.
I_5	0,99388	1(*)	0,95293	n.a.	-	1(*)
I_6	1(*)	1(*)	1(*)	n.a.	1(*)	-
Similarity Vector:	0,91489	0,98825	0,89442	1	0,97340	1

U_1 will receive the following recommendations: I_6, I_4 and I_2 .

Discussion

Item-based neighbor algorithms typically recommend items that are *very similar* to items that the user already knows (rated before).

But there is evidence that item-based algorithms are *more accurate* in predicting ratings than their user-based counterparts.

The main advantage of item-based algorithms is that similarities can be *computed offline*

Similar to user-based algorithms, item pairs with few co-ratings can lead to skewed correlations and care must be taken not to let skewed correlations dominate a prediction.