source: altkom software & consulting

DESCRIPTION AND A COMPARISON OF TWO RECOMMENDATION ENGINES

engines: collaborative filtering

Jonathan Leban May 11, 2020 · 7 min read ★

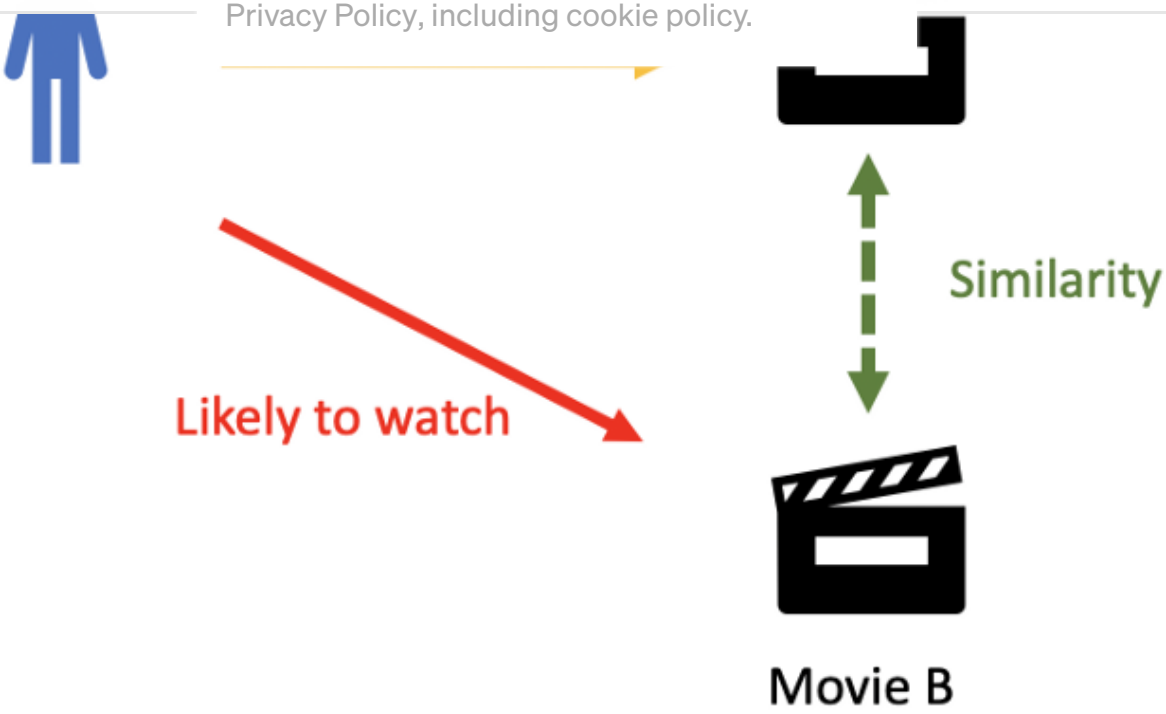
Have you ever wondered why the ads on Facebook are so relevant to what you're interested in or how the "movie-match" on Netflix works? Is it magic? No. In both cases, a recommendation engine or system makes predictions based on your historical behavior.

If you are a fan of science fiction movies and have watched Star Wars, the recommendation engine may suggest that you watch Avatar. This method is known as **content-based filtering** because it analyzes the content of each item and finds similar items. While very useful, it requires a thorough knowledge of each item in order to find similar items.

We can also think of another situation where one of your friends who shares many similarities with you tells you "Yesterday, I went to the cinema and saw the new George Lucas movie, it was great! You should really go see it". Having common points with other users can also be a good way to recommend movies. Here is the second type of recommendation engine: **collaborative filtering**.

In this article, I will review the principles behind content-based filtering and collaborative filtering before comparing them.

Content-Based filtering



> *The idea here is to recommend similar items to the ones you liked before. The system first finds the similarity between all pairs of articles and then uses the articles most similar to the articles already evaluated by a user to generate a list of recommendations.*
>
> *But a new question arises:* ***how can you find similarities between items?***

To compare two items, we need to transform them into mathematical objects such as vectors, on which we can compute metrics such as *Euclidean distance, Pearson's coefficient* or *cosine similarity*.
Below are expressed the formulas of the different metrics exposed above:

$$Euclidean\ distance(u,p) = \sum_i (u_i - p_i)^2$$

$$\sum_i (u_i - \bar{u})(p_i - \bar{p})$$

$$Cosine\ similarity(u,p) = \frac{\sum_i u_i p_i}{\sqrt{\sum_i u_i^2}\sqrt{\sum_i p_i^2}}$$

A commonly used technique is the **TF-IDF** (frequency term — inverted document frequency). It is a statistical measure that evaluates the relevance of a word to a document in a set of documents. This is done by multiplying two measures: the number of times a word appears in a document (term frequency) and the inverse of the number of documents in which the word appears (inverse document frequency). If a film is a science fiction film like Star Wars is, the science fiction word may appear a lot in the film description, so the term frequency will be high, and if there are not many science fiction films in the film corpus, the inverse of the term frequency will also be high. Finally, after normalization, the value of TF-IDF will be close to one.

If we apply this technique to a set of films, after normalization, we can obtain the following table. The numbers in this table are random plausible values.

Movies	Science-fiction	Action	Romantic
Star Wars	0.92	0.87	0.35
Avatar	0.93	0.7	0.5
Titanic	0.05	0.2	0.9

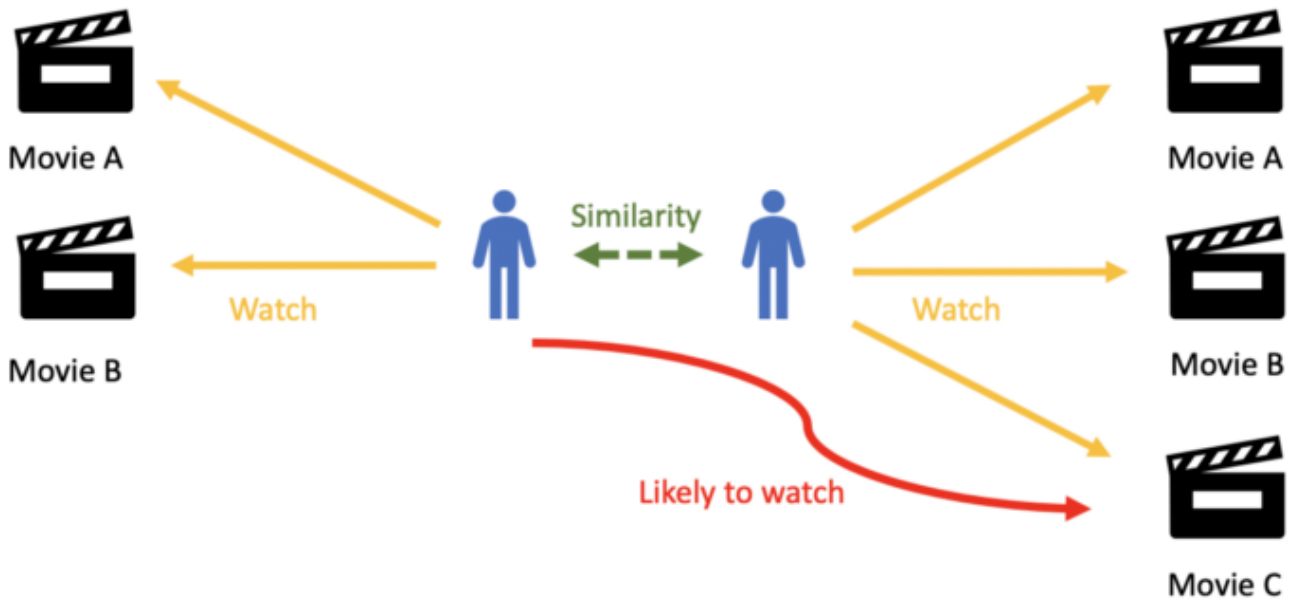
Now that we have vectors that describe a movie, we can calculate the *Euclidean distance, the Pearson coefficient, or the Cosine similarity* between two of these vectors. For example, the Euclidean distance

more similar the how close Star Wars and Avatar are according to the characteristics we have chosen (Science Fiction, Action, Romantic) and how far apart Star Wars and Titanic are.

As you can see, it is essential to know the content of each element for this method. In the next part, we will delve into the collaborative method as a method based on the similarities between users and objects simultaneously.

Collaborative Filtering

> *Collaborative filtering doesn't need anything else but users' historical preference on a set of items.*



The standard method of Collaborative Filtering is known as the **Nearest Neighborhood** algorithm. We have an n × m matrix of ratings, that we will call R, the user matrix is denoted as U and the item matrix as P. The user i is represented by the vector u_i and they are n users so i = 1, …n.

watch/rate an it ilarities between target user i and all other users, select the top X similar users, and take the weighted average of ratings from these X users with similarities as weights. The rating $r_{i\,j}$ is defined as:

$$r_{ij} = \frac{\sum_k Similarities(u_k, u_i) * r_{kj}}{number\ of\ ratings}$$

However, people do not have the same rating scales and some people tend to give generally higher scores than others. This bias can be avoided by subtracting each user's average score for all items when calculating the weighted average, and adding it for the target user, as shown below.

$$r_{ij} = \bar{r_i} + \frac{\sum_k Similarities(u_k, u_i) * (r_{kj} - \bar{r_k})}{number\ of\ ratings}$$

Then we use the *Pearson Correlation, the Cosine similarity* or *the Euclidean distance* to compute the similarities.

But there are some limitations to this method. It doesn't handle sparsity well, e.g. when no one in the neighborhood rated the item you are trying to predict for a target user.

Since **sparsity and scalability** are the two biggest challenges for the standard CF method, here comes a more advanced method that

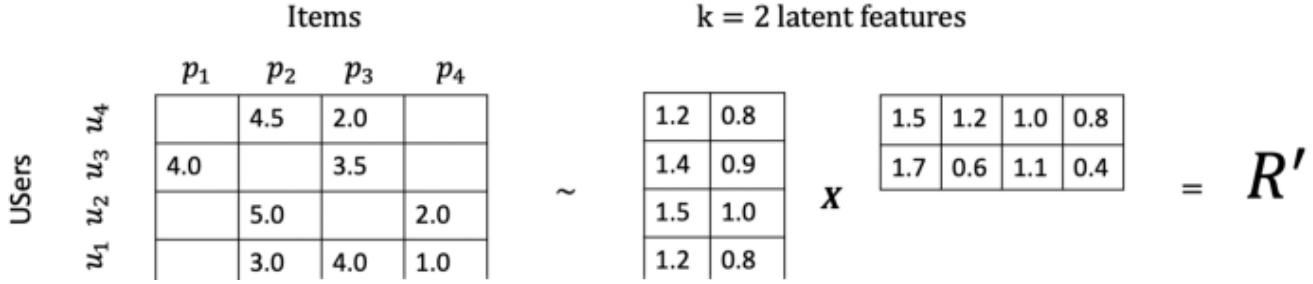
Matrix Factoriz, we want to write R as a product of two matrices: U the matrix of users and P the matrix of items. As the decomposition cannot be perfectly equal (R is a bit different than the product of U and P), we introduce another matrix called R' which represent the predicted rating. It is defined as the result of the product U by P.

The number of rows of the matrix U are equal to the number of users. The number of columns of the matrix P is equal to the number of items. The number of columns of the matrix U is equal to the number of rows of the matrix P which is equal to the number of the latent vectors.

In the example below, I choose the number of latent vectors equal to 2. To understand this choice, we need to look into the SVD decomposition where the number of latent vectors is the number of vectors ensuring R' to be able to capture the most of variance within the original matrix R, so that R' is the approximation of R, R' ≈ R. We note r_{ij}' the approximate rating of item j by user i. Thus, we have:

$$r'_{ij} = u_i^T p_j$$

The idea is to minimize the difference between r_{ij} and r_{ij}'. The problem is exposed below:



> *But, how do we find the optimal vector u_i and p_j ?*

Recall that these vectors represent the user i and the item j. If you are familiar with machine learning, to make a prediction, we always use a loss function that we want to minimize:

$$Loss = min_{u,p} \sum (r_{ij} - u_i^T p_j)^2 + \lambda(\| u_i \|^2 + \| p_j \|^2)$$

The purpose of the optimization process is to find the best U and P that will minimize the loss function. Also, L2 regularization has been added to prevent overfitting of user and item vectors.

Comparison of the two techniques

Although in content-based filtering, the model does not need data on other users since the recommendations are specific to that user, it is at the heart of the collaborative filtering algorithm. However, a thorough knowledge of the elements is essential for the content-based algorithm, whereas only element evaluations are required in the collaborative filtering method.

Closely related to this point is the issue of introducing new elements, also known as the “cold start problem”. If an element is not seen during the training, the system cannot create an embedding for it and cannot query the model with that element. While some techniques exist as projection in WALS or heuristics to generate new embeddings, the addition to a new feature will strengthen the content-based filtering method.

Recommendation System Recommendation Engine Collaborative Filtering

Content Based Filtering