Collaborative filtering

Collaborative filtering is a family of algorithms where there are multiple ways to find similar users or items and multiple ways to calculate rating based on ratings of similar users. Depending on the choices you make, you end up with a type of collaborative filtering approach.

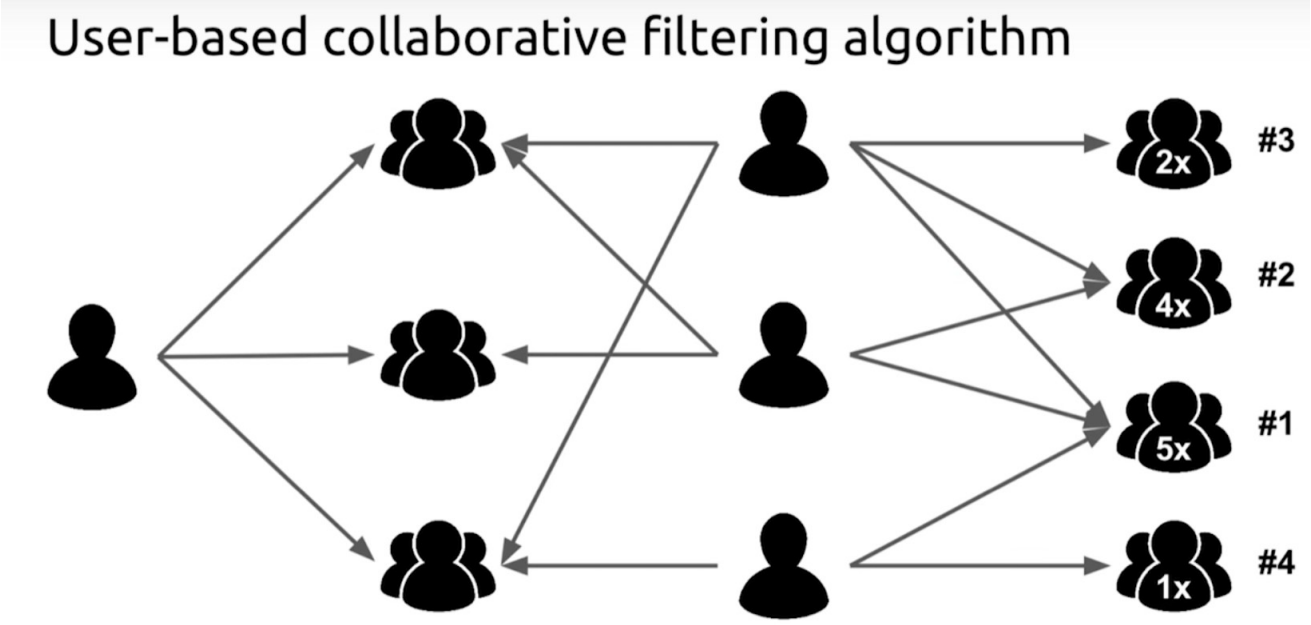
One important thing to keep in mind is that in an approach based purely on collaborative filtering, the similarity is not calculated using factors like the age of users, genre of the movie, or any other data about users or items. It is calculated only on the basis of the rating (explicit or implicit) a user gives to an item. For example, two users can be considered similar if they give the same ratings to ten movies despite there being a big difference in their age.

The third question for how to measure the accuracy of your predictions also has multiple answers, which include error calculation techniques that can be used in many places and not just recommenders based on collaborative filtering.

One of the approaches to measure the accuracy of your result is the Root Mean Square Error (RMSE), in which you predict ratings for a test dataset of user-item pairs whose rating values are already known. The difference between the known value and the predicted value would be the error. Square all the error values for the test set, find the average (or mean), and then take the square root of that average to get the RMSE.

Another metric to measure the accuracy is Mean Absolute Error (MAE), in which you find the magnitude of error by finding its absolute value and then taking the average of all error values.

You don’t need to worry about the details of RMSE or MAE at this point as they are readily available as part of various packages in Python, and you will see them later in the article.



Now let’s look at the different types of algorithms in the family of collaborative filtering.

How to Calculate the Ratings

After you have determined a list of users similar to a user U, you need to calculate the rating R that U would give to a certain item I. Again, just like similarity, you can do this in multiple ways.

You can predict that a user's rating R for an item I will be close to the average of the ratings given to I by the top 5 or top 10 users most similar to U. The mathematical formula for the average rating given by n users would look like this:

Formula for average rating

$$R_U = (\sum_{u=1}^{n} R_u)/n$$

This formula shows that the average rating given by the n similar users is equal to the sum of the ratings given by them divided by the number of similar users, which is n.

There will be situations where the n similar users that you found are not equally similar to the target user U. The top 3 of them might be very similar, and the rest might not be as similar to U as the top 3. In that case, you could consider an approach where the rating of the most similar user matters more than the second most similar user and so on. The weighted average can help us achieve that.

In the weighted average approach, you multiply each rating by a similarity factor(which tells how similar the users are). By multiplying with the similarity factor, you add weights to the ratings. The heavier the weight, the more the rating would matter.

The similarity factor, which would act as weights, should be the inverse of the distance discussed above because less distance implies higher similarity. For example, you can subtract the cosine distance from 1 to get cosine similarity.

With the similarity factor S for each user similar to the target user U, you can calculate the weighted average using this formula:

Formula for weighted average rating

$$R_U = (\sum_{u=1}^{n} R_u * S_u)/(\sum_{u=1}^{n} S_u)$$

In the above formula, every rating is multiplied by the similarity factor of the user who gave the rating. The final predicted rating by user U will be equal to the sum of the weighted ratings divided by the sum of the weights.

In case you're wondering why the sum of weighted ratings is being divided by the sum of the weights and not by n, consider this: in the previous formula of the average, where you divided by n, the value of the weight was 1.

User-Based vs Item-Based Collaborative Filtering

The technique in the examples explained above, where the rating matrix is used to find similar users based on the ratings they give, is called user-based or user-user collaborative filtering. If you use the rating matrix to find similar items based on the ratings given to them by users, then the approach is called item-based or item-item collaborative filtering.

The two approaches are mathematically quite similar, but there is a conceptual difference between the two. Here's how the two compare:

User-based: For a user U, with a set of similar users determined based on rating vectors consisting of given item ratings, the rating for an item I, which hasn't been rated, is found by picking out N users from the similarity list who have rated the item I and calculating the rating based on these N ratings.

Item-based: For an item I, with a set of similar items determined based on rating vectors consisting of received user ratings, the rating by a user U, who hasn't rated it, is found by picking out N items from the similarity list that have been rated by U and calculating the rating based on these N ratings.

Item-based collaborative filtering was developed by Amazon. In a system where there are more users than items, item-based filtering is faster and more stable than user-based. It is effective because usually, the average rating received by an item doesn't change as quickly as the average rating given by a user to different items. It's also known to perform better than the user-based approach when the ratings matrix is sparse.

Although, the item-based approach performs poorly for datasets with browsing or entertainment related items such as MovieLens, where the recommendations it gives out seem very obvious to the target users. Such datasets see better results with matrix factorization techniques, which you'll see in the next section, or with hybrid recommenders that also take into account the content of the data like the genre by using content-based filtering.

You can use the library Surprise to experiment with different recommender algorithms quickly. (You will see more about this later in the article.)

Model Based

The second category covers the Model based approaches, which involve a step to reduce or compress the large but sparse user-item matrix. For understanding this step, a basic understanding of dimensionality reduction can be very helpful.

Dimensionality Reduction
In the user-item matrix, there are two dimensions:

The number of users
The number of items
If the matrix is mostly empty, reducing dimensions can improve the performance of the algorithm in terms of both space and time. You can use various methods like matrix factorization or autoencoders to do this.

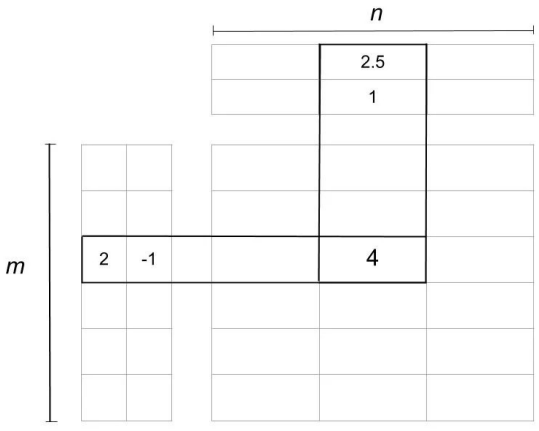
Matrix factorization can be seen as breaking down a large matrix into a product of smaller ones. This is similar to the factorization of integers, where 12 can be written as 6 x 2 or 4 x 3. In the case of matrices, a matrix A with dimensions m x n can be reduced to a product of two matrices X and Y with dimensions m x p and p x n respectively.

Note: In matrix multiplication, a matrix X can be multiplied by Y only if the number of columns in X is equal to the number of rows in Y. Therefore the two reduced matrices have a common dimension p.

Depending on the algorithm used for dimensionality reduction, the number of reduced matrices can be more than two as well.

The reduced matrices actually represent the users and items individually. The m rows in the first matrix represent the m users, and the p columns tell you about the features or characteristics of the users. The same goes for the item matrix with n items and p characteristics. Here’s an example of how matrix factorization looks:

A matrix factorized into two matrices using dimensionality reduction
Matrix Factorization
In the image above, the matrix is reduced into two matrices. The one on the left is the user matrix with m users, and the one on top is the item matrix with n items. The rating 4 is reduced or factorized into:



A user vector (2, -1)
An item vector (2.5, 1)
The two columns in the user matrix and the two rows in the item matrix are called latent factors and are an indication of hidden characteristics about the users or the items. A possible interpretation of the factorization could look like this:

Assume that in a user vector (u, v), u represents how much a user likes the Horror genre, and v represents how much they like the Romance genre.

The user vector (2, -1) thus represents a user who likes horror movies and rates them positively and dislikes movies that have romance and rates them negatively.

Assume that in an item vector (i, j), i represents how much a movie belongs to the Horror genre, and j represents how much that movie belongs to the Romance genre.

The movie (2.5, 1) has a Horror rating of 2.5 and a Romance rating of 1. Multiplying it by the user vector using matrix multiplication rules gives you (2 * 2.5) + (-1 * 1) = 4.

So, the movie belonged to the Horror genre, and the user could have rated it 5, but the slight inclusion of Romance caused the final rating to drop to 4.

The factor matrices can provide such insights about users and items, but in reality they are usually much more complex than the explanation given above. The number of such factors can be anything from one to hundreds or even thousands. This number is one of the things that need to be optimized during the training of the model.

In the example, you had two latent factors for movie genres, but in real scenarios, these latent factors need not be analyzed too much. These are patterns in the data that will play their part automatically whether you decipher their underlying meaning or not.

The number of latent factors affects the recommendations in a manner where the greater the number of factors, the more personalized the recommendations become. But too many factors can lead to overfitting in the model.