Introduction to Recommender System

Approaches of Collaborative Filtering: Nearest Neighborhood and Matrix Factorization

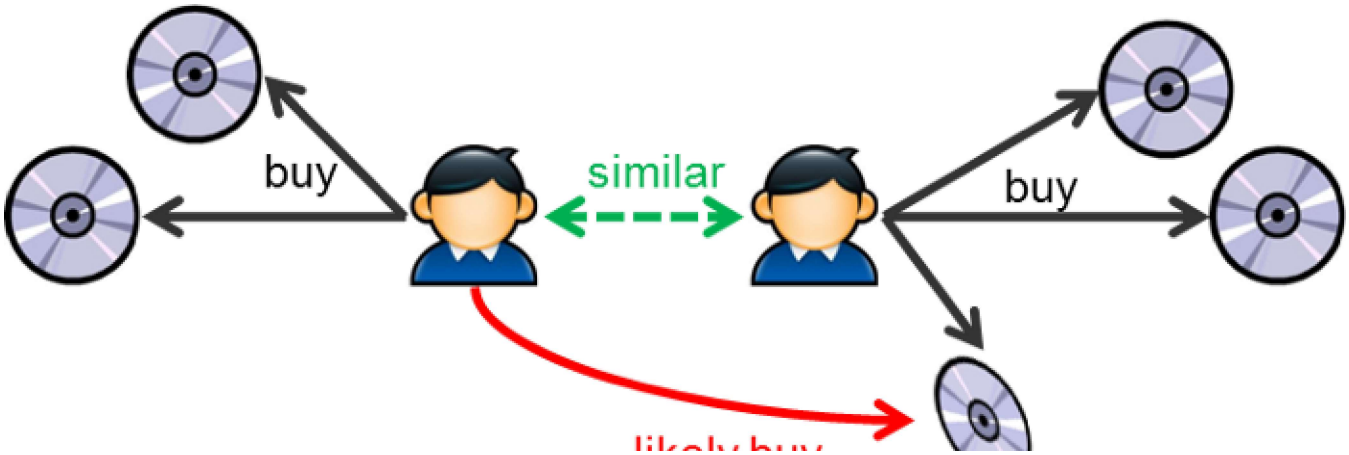
Shuyu Luo Dec 10, 2018 · 8 min read

Like many machine learning techniques, a recommender system makes prediction based on users' historical behaviors. Specifically, it's to predict user preference for a set of items based on past experience. To build a recommender system, the most two popular approaches are Content-based and Collaborative Filtering.

Content-based approach requires a good amount of information of items' own features, rather than using users' interactions and feedbacks. For example, it can be movie attributes such as genre, year, director, actor etc., or textual content of articles that can extracted by applying Natural Language Processing. **Collaborative Filtering**, on the other hand, doesn't need anything else except users' historical preference on a set of items. Because it's based on historical data, the core assumption here is that the users who have agreed in the past tend to also agree in the future. In terms of user preference, it usually expressed by two categories. **Explicit Rating**, is a rate given by a user to an item on a sliding scale, like 5 stars for Titanic. This is the most direct feedback from users to show how much they like an item. **Implicit Rating**, suggests users preference indirectly, such as page views, clicks, purchase records, whether or not listen to a music track, and so on. In this article, I will take a close look at collaborative filtering that is a traditional and powerful tool for recommender systems.

Nearest Neighborhood

The standard method of Collaborative Filtering is known as **Nearest Neighborhood** algorithm. There are user-based CF and item-based CF. Let's first look at **User-based CF**. We have an $n \times m$ matrix of ratings, with user u_i, $i = 1, \ldots n$ and item p_j, $j=1, \ldots m$. Now we want to predict the rating r_{ij} if target user i did not watch/rate an item j. The process is to calculate the similarities 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.



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

While different people may have different baselines when giving ratings, some people tend to give high scores generally, some are pretty strict even though they are satisfied with items. To avoid this bias, we can subtract each user's average rating of all items when computing weighted average, and add it back for target user, shown as below.

$$r_{ij} = \bar{r}_i + \frac{\sum_k Similaries(u_i, u_k)(r_{kj} - \bar{r}_k)}{number\ of\ ratings}$$

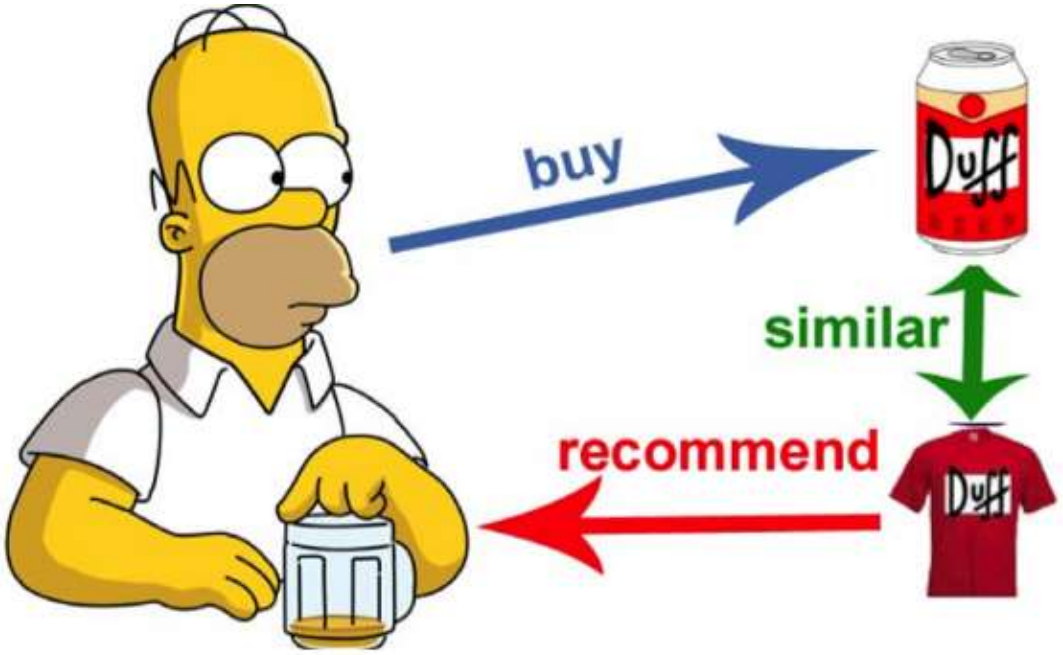
Two ways to calculate similarity are **Pearson Correlation** and **Cosine Similarity**.

$$Pearson\ Correlation : Sim(u_i, u_k) = \frac{\sum_j (r_{ij} - r_i)(r_{kj} - r_k)}{\sqrt{\sum_j (r_{ij} - r_i)^2 \sum_j (r_{kj} - r_k)^2}}$$

$$Cosine\ Similarity : Sim(u_i, u_k) = \frac{r_i \cdot r_k}{|r_i||r_k|} = \frac{\sum_{j=1}^{m} r_{ij} r_{kj}}{\sqrt{\sum_{j=1}^{m} r_{ij}^2 \sum_{j=1}^{m} r_{kj}^2}}$$

item for target user.

Without knowing anything about items and users themselves, we think two users are similar when they give the same item similar ratings . Analogously, for **Item-based CF**, we say two items are similar when they received similar ratings from a same user. Then, we will make prediction for a target user on an item by calculating weighted average of ratings on most X similar items from this user. One key advantage of Item-based CF is the stability which is that the ratings on a given item will not change significantly overtime, unlike the tastes of human beings.



Source: https://medium.com/tiket-com-dev-team/build-recommendation-engine-using-graph-cbd6d8732e46

There are quite a few limitations of this method. It doesn't handle sparsity well when no one in the neighborhood rated an item that is what you are trying to predict for target user. Also, it's not computational efficient as the growth of the number of users and products.

Matrix Factorization

Since sparsity and scalability are the two biggest challenges for standard CF method, it comes a more advanced method that decompose the original sparse matrix to low-

Beside solving the issues of sparsity and scalability, there's an intuitive explanation of why we need low-dimensional matrices to represent users' preference. A user gave good ratings to movie Avatar, Gravity, and Inception. They are not necessarily 3 separate opinions but showing that this users might be in favor of Sci-Fi movies and there may be many more Sci-Fi movies that this user would like. Unlike specific movies, latent features is expressed by higher-level attributes, and Sci-Fi category is one of latent features in this case. What matrix factorization eventually gives us is how much a user is aligned with a set of latent features, and how much a movie fits into this set of latent features. The advantage of it over standard nearest neighborhood is that even though two users haven't rated any same movies, it's still possible to find the similarity between them if they share the similar underlying tastes, again latent features.

To see how a matrix being factorized, first thing to understand is **Singular Value Decomposition(SVD)**. Based on Linear Algebra, any real matrix R can be decomposed into 3 matrices U, Σ, and V. Continuing using movie example, U is an n × r user-latent feature matrix, V is an m × r movie-latent feature matrix. Σ is an r × r diagonal matrix containing the singular values of original matrix, simply representing how important a specific feature is to predict user preference.

$$R = U\Sigma V^T$$

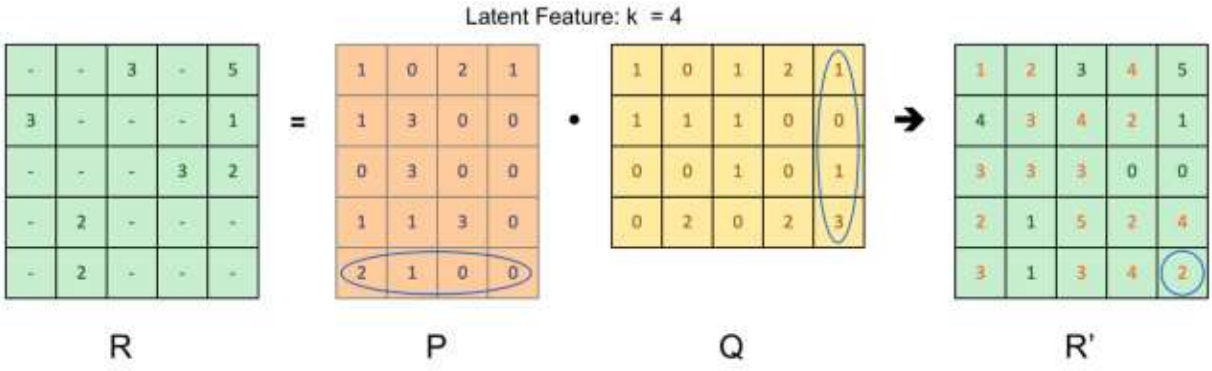
$$U \in IR^{n\times r}, \quad \Sigma \in IR^{r\times r}, \quad V \in IR^{r\times m}$$

To sort the values of Σ by decreasing absolute value and truncate matrix Σ to first k dimensions(k singular values), we can reconstruct the matrix as matrix A. The selection of k should make sure that A is able to capture the most of variance within the original matrix R, so that A is the approximation of R, A ≈ R. The difference between A and R is the error that is expected to be minimized. This is exactly the thought of Principle Component Analysis.

When matrix R is dense, U and V could be easily factorized analytically. However, a matrix of movie ratings is super sparse. Although there are some imputation methods to fill in missing values , we will turn to a programming approach to just live with those missing values and find factor matrices U and V. Instead of factorizing R via SVD, we are trying find U and V directly with the goal that when U and V multiplied back together the output matrix R' is the closest approximation of R and no more a sparse matrix. This numerical approximation is usually achieved with **Non-Negative Matrix Factorization** for recommender systems since there is no negative values in ratings.

See the formula below. Looking at the predicted rating for specific user and item, item i is noted as a vector q_i, and user u is noted as a vector p_u such that the dot product of these two vectors is the predicted rating for user u on item i. This value is presented in the matrix R' at row u and column i.

$$Predicted\ Ratings: \ r'_{ui} = p_u^T q_i$$



How do we find optimal q_i and p_u? Like most of machine learning task, a loss function is defined to minimize the cost of errors.

$$\min \sum (r_{ui} - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

r_{ui} is the true ratings from original user-item matrix. Optimization process is to find the optimal matrix P composed by vector p_u and matrix Q composed by vector q_i in order to minimize the sum square error between predicted ratings r_{ui}' and the true ratings r_{ui}. Also, L2 regularization has been added to prevent overfitting of user and item vectors. It's also quite common to add bias term which usually has 3 major components: average rating of all items μ, average rating of item i minus μ(noted as b_u), average rating given by user u minus u(noted as b_i).

$$\min_{p,q,b_u,b_i} \sum (r_{ui} - (p_u^T q_i + \mu + b_u + b_i))^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

Optimization

A few optimization algorithms have been popular to solve Non-Negative Factorization. **Alternative Least Square** is one of them. Since the loss function is non-convex in this case, there's no way to reach a global minimum, while it still can reach a great approximation by finding local minimums. Alternative Least Square is to hold user factor matrix constant, adjust item factor matrix by taking derivatives of loss function and setting it equal to 0, and then set item factor matrix constant while adjusting user factor matrix. Repeat the process by switching and adjusting matrices back and forth until convergence. If you apply Scikit-learn NMF model, you will see ALS is the default solver to use, which is also called Coordinate Descent. Pyspark also offers pretty neat decomposition packages that provides more tuning flexibility of ALS itself.

Some Thoughts

Collaborative Filtering provides strong predictive power for recommender systems, and requires the least information at the same time. However, it has a few limitations in some particular situations.

First, the underlying tastes expressed by latent features are actually not interpretable because there is no content-related properties of metadata. For movie example, it doesn't necessarily to be genre like Sci-Fi in my example. It can be how motivational the soundtrack is, how good the plot is, and so on. Collaborative Filtering is lack of transparency and explainability of this level of information.

to make any personalized recommendations . Similarly, for items from the tail that didn't get too much data, the model tends to give less weight on them and have popularity bias by recommending more popular items.

It's usually a good idea to have ensemble algorithms to build a more comprehensive machine learning model such as combining content-based filtering by adding some dimensions of keywords that are explainable, but we should always consider the tradeoff between model/computational complexity and the effectiveness of performance improvement.