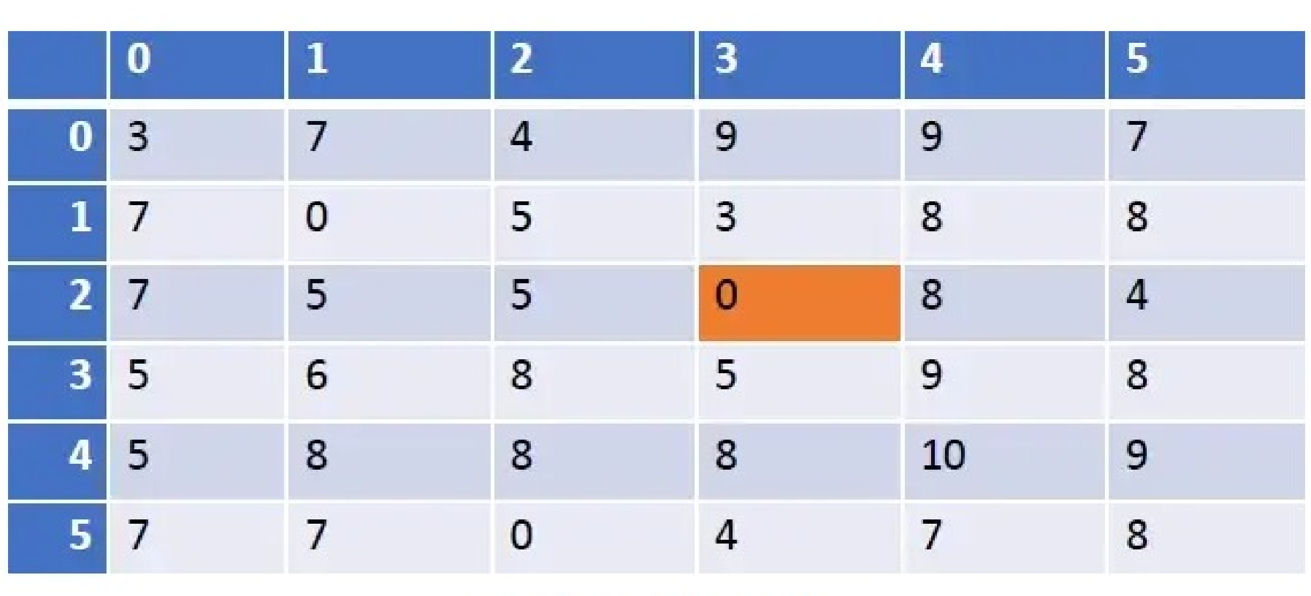
Chhavi Saluja
Mar 6, 2018 · 5 min read · Listen

Collaborative Filtering based Recommendation Systems exemplified..

In my last post, I've given a simple explanation of Recommendation Systems illustrating various types of recommendation systems. In this post, I shall be realizing simple examples for some of these types of recommendation systems using **Python**.

Given a user-item ratings matrix M below, where 6 users (rows) have rated 6 items (columns). Ratings can take up integer values from 1–10 and 0 indicates absence of rating. (Note that we are using zero-based indexing for rows and columns as used in Python. However, for user input, user_id will take up numbers from 1–6 and item_id from 1–6). Suppose, we've to find if user 3 will like item 4 or not. Hence user 3 becomes our target user or active user and item 4 is the target item.



	0	1	2	3	4	5
0	3	7	4	9	9	7
1	7	0	5	3	8	8
2	7	5	5	0	8	4
3	5	6	8	5	9	8
4	5	8	8	8	10	9
5	7	7	0	4	7	8

sample user-ratings matrix

User-Based Collaborative Filtering

Firstly, we will have to predict the rating that user 3 will give to item 4. In user-based CF, we will find say k=3 users who are most similar to user 3. Commonly used similarity measures are cosine, Pearson, Euclidean etc. We will use cosine similarity here which is defined as below:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

And, pearson correlation, defined as:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

In sklearn, NearestNeighbors method can be used to search for k nearest neighbors based on various similarity metrics.

Check in my **Jupyter Notebook embed** below,

csaluja/JupyterNotebooks-Medium
Contribute to JupyterNotebooks-Medium development by creating an account on GitHub.
github.com

findksimilarusers function uses this method to return similarities and indices of k-nearest neighbors for active user. The function ***predict_userbased*** further predicts rating that user 3 will give to item 4, using user-based CF approach. Predictions are computed as weighted average of deviations from neighbor's mean and adding it to active user's mean rating. Deviations are used to adjust for the user associated biases. User biases occur as certain users may tend to always give high or low ratings to all items.

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in K}(r_{u,i} - \bar{r}_u) \times w_{a,u}}{\sum_{u \in K} w_{a,u}}$$

Where p(a,i) is the prediction for target or active user a for item i, w(a,u) is the similarity between users a and u, and K is the neighborhood of most similar users.

Item-Based Collaborative Filtering

In this approach, similarities between **pair of items** are computed using cosine similarity metric. The rating for target item *i* for active user *a* can be predicted by using a simple weighted average as:

$$p_{a,i} = \frac{\sum_{j \in K} r_{a,j} w_{i,j}}{\sum_{j \in K} |w_{i,j}|}$$

where K is the neighborhood of most similar items rated by active user a, and w(i,j) is the similarity between items *i* and *j*.

Check in Jupyter Notebook embed, the function ***findksimilaritems*** uses NearestNeighbors method employing cosine similarity to find k items similar to item *i*. The function ***predict_itembased*** further predicts rating that user 3 will give to item 4, using item-based CF approach (above formula).

Adjusted Cosine Similarity

Using cosine similarity metric for item-based CF approach does not consider difference in ratings of users. Adjusted cosine similarity offsets this drawback by subtracting respective user's average rating from each co-rated pair, and is defined as below-

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R_u})(R_{u,j} - \bar{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R_u})^2}}.$$

To realize Adjusted Cosine similarity in Python, I've defined a simple function named *computeAdjCosSim,* which returns adjusted cosine similarity matrix, given the ratings matrix. The functions ***findksimilaritems_adjcos*** and ***predict_itembased_adjcos*** exploit adjusted cosine similarity to find k similar items and calculate the predicted ratings.

The function ***recommendItem*** prompts the user to select the recommendation approach (user-based (cosine), user-based (correlation), item-based (cosine), item-based (adjusted-cosine). Based on the selected approach and similarity metric, this function predicts the rating for specified user and item and also suggests if the item could be recommended to the user. If the item is not rated by the user already and if the predicted rating is greater than 6, the item is recommended to the user. If the rating is less than 6, the item is not recommended to the user.

Selection of similarity metric

Some helpful cues while selecting similarity metric are-

·Use Pearson when your data is subject to user-bias/ different ratings scales of users

· Use Cosine, if data is sparse (many ratings are undefined)

· Use Euclidean, if your data is not sparse and the magnitude of the attribute values is significant

· Use adjusted cosine for Item-based approach to adjust for user-bias

Evaluation of recommendation system

There are many evaluation metrics for evaluating recommendation systems. However, the most popular and most commonly used is RMSE (root mean squared error). The function ***evaluateRS*** uses mean_squared_error function from sklearn to calculate RMSE between predicted ratings and actual ratings, and displays RMSE values for selected approach. (For simplicity of explanation, small dataset is used and hence it has not been split into train and test sets; cross-validation is also not considered in this post).

In applications where user-base is large, user-based approaches face scalability issues, as their complexity grows linearly with number of users. Item-based approaches address these scalability concerns to recommend items based on item similarities. Hybrid techniques take advantages of various kinds of such approaches and combine them in several ways to achieve better performance.

Thanks for reading. This post is to share my understanding of recommendation systems and any feedback for improvement of this implementation is most welcome.