**Collaborative Filtering based Recommender System using K Nearest Neighbor**

Collaborative filtering is probably the most commonly used recommendation algorithm, there are two main types of methods:

* **User-based** collaborative filtering is based on the user similarity or neighborhood
* **Item-based** collaborative filtering is based on similarity among items

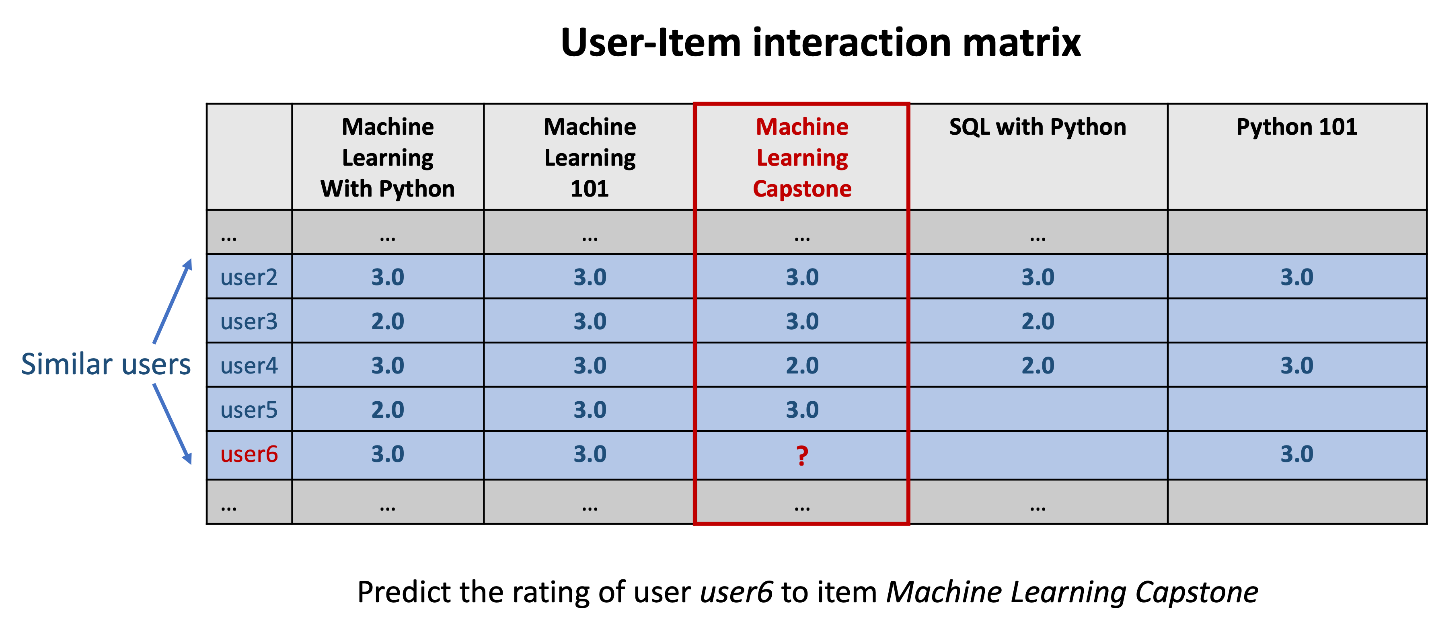
They both work similarly, let's briefly explain how user-based collaborative filtering works.

User-based collaborative filtering looks for users who are similar. This is very similar to the user clustering method done previously; where we employed explicit user profiles to calculate user similarity. However, the user profiles may not be available, so how can we determine if two users are similar?

**User-item interaction matrix**

For most collaborative filtering-based recommender systems, the main dataset format is a 2-D matrix called the user-item interaction matrix. In the matrix, its row is labeled as the user id/index and column labelled to be the item id/index, and the element (i, j) represents the rating of user i to item j.

Below is a simple example of a user-item interaction matrix:



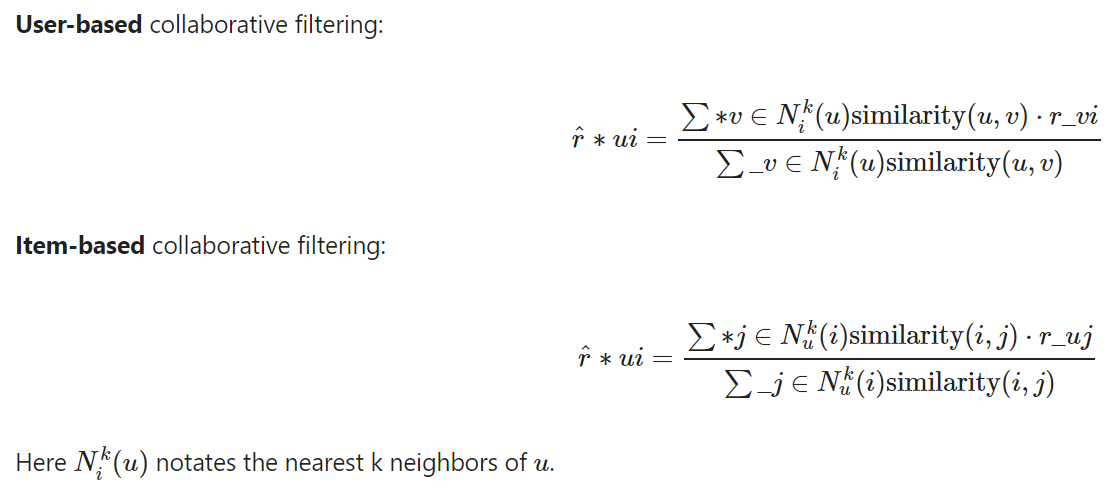
**KNN-based collaborative filtering**

As we can see from above, each row vector represents the rating history of a user and each column vector represents the users who rated the item. A user-item interaction matrix is usually very sparse as you can imagine one user very likely only interacts with a very small subset of items and one item is very likely to be interacted by a small subset of users.

Now to determine if two users are similar, we can simply calculate the similarities between their row vectors in the interaction matrix. Then based on the similarity measurements, we can find the k nearest neighbor as the similar users.

Item-based collaborative filtering works similarly, we just need to look at the user-item matrix vertically. Instead of finding similar users, we are trying to find similar items (courses). If two courses are enrolled by two groups of similar users, then we could consider the two items are similar and use the known ratings from the other users to predict the unknown ratings.

If we formulate the KNN based collaborative filtering, the predicted rating of user uu to item ii, ^r\_uir^\_ui is given by:

****Let's illustrate how the equation works using a simple example. From the above figure, suppose we want to predict the rating of user6 to item Machine Learning Capstone course. After some similarity measurements, we found that k = 4 nearest neighbors: user2, user3, user4, user5 with similarities in array knn\_sims:

**import** numpy **as** np

**import** math

*# An example similarity array stores the similarity of user2, user3, user4, and user5 to user6*

knn\_sims **=** np**.**array([0.8, 0.92, 0.75, 0.83])

Also their rating on the Machine Learning Capstone course are:

*# 2.0 means audit and 3.0 means complete the course*

knn\_ratings **=** np**.**array([3.0, 3.0, 2.0, 3.0])

So the predicted rating of user6 to item Machine Learning Capstone course can be calculated as:

r\_u6\_ml **=** np**.**dot(knn\_sims, knn\_ratings)**/** sum(knn\_sims)

r\_u6\_ml

If we already know the true rating to be 3.0, then we get a prediction error RMSE (Rooted Mean Squared Error) as:

true\_rating **=** 3.0

rmse **=** math**.**sqrt(true\_rating **-** r\_u6\_ml) **\*\*** 2

rmse

The predicted rating is around 2.7 (close to 3.0 with RMSE 0.22), which indicates that user6 is also likely to complete the course Machine Learning Capstone. As such, we may recommend it to user6 with high confidence.

**Objectives**

After completing this mini capstone you will be able to:

* Perform KNN-based collaborative filtering on the user-item interaction matrix

**Load and exploring dataset**

Let's first load our dataset, i.e., a user-item (learn-course) interaction matrix

**import** pandas **as** pd

rating\_url **=** "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-ML321EN-SkillsNetwork/labs/datasets/ratings.csv"

rating\_df **=** pd**.**read\_csv(rating\_url)

rating\_df**.**head()

The dataset contains three columns, user id (learner), item id(course), and rating(enrollment mode).

Note that this matrix is presented as the dense or vertical form, and you may convert it to a sparse matrix using pivot :

rating\_sparse\_df **=** rating\_df**.**pivot(index**=**'user', columns**=**'item', values**=**'rating')**.**fillna(0)**.**reset\_index()**.**rename\_axis(index**=None**, columns**=None**)

rating\_sparse\_df**.**head()

Usually, the dense format is more preferred as it saves a lot of storage and memory space. While the benefit of the sparse matrix is it is in the nature matrix format and you could apply computations such as cosine similarity directly.

Next, you need to perform KNN-based collaborative filtering on the user-item interaction matrix. **You must implement the two following options of KNN-based collaborative filtering**.

* The first one is to use scikit-surprise which is a popular and easy-to-use Python recommendation system library.
* The second way is to implement it with standard numpy, pandas, and sklearn. You may need to write a lot of low-level implementation code along the way.

**Implementation Option 1: Use Surprise library**

*Surprise* is a Python sci-kit library for recommender systems. It is simple and comprehensive to build and test different recommendation algorithms.

First, let's install it:

**!**pip install scikit-surprise**==**1.1.1

Now we import required classes and methods

**from** surprise **import** KNNBasic

**from** surprise **import** Dataset, Reader

**from** surprise.model\_selection **import** train\_test\_split

**from** surprise **import** accuracy

Then, let's take a look at a code example how easily to perform KNN collaborative filtering on a sample movie review dataset, which contains about 100k movie ratings from users.

*# Load the movielens-100k dataset (download it if needed),*

data **=** Dataset**.**load\_builtin('ml-100k', prompt**=False**)

*# sample random trainset and testset*

*# test set is made of 25% of the ratings.*

trainset, testset **=** train\_test\_split(data, test\_size**=**.25)

*# We'll use the famous KNNBasic algorithm.*

algo **=** KNNBasic()

*# Train the algorithm on the trainset, and predict ratings for the testset*

algo**.**fit(trainset)

predictions **=** algo**.**test(testset)

*# Then compute RMSE*

accuracy**.**rmse(predictions)

As you can see, just a couple of lines and you can apply KNN collaborative filtering on the sample movie lens dataset. The main evaluation metric is Root Mean Square Error (RMSE) which is a very popular rating estimation error metric used in recommender systems as well as many regression model evaluations.

Now, let's load our own course rating dataset:

rating\_df**.**to\_csv("course\_ratings.csv", index**=False**)

*# Read the course rating dataset with columns user item rating*

reader **=** Reader(

line\_format**=**'user item rating', sep**=**',', skip\_lines**=**1, rating\_scale**=**(2, 3))

coruse\_dataset **=** Dataset**.**load\_from\_file("course\_ratings.csv", reader**=**reader)

We split it into trainset and testset:

trainset, testset **=** train\_test\_split(coruse\_dataset, test\_size**=**.3)

then check how many users and items we can use to fit a KNN model:

print(f"Total {trainset**.**n\_users} users and {trainset**.**n\_items} items in the trainingset")

**TASK: Perform KNN-based collaborative filtering on the user-item interaction matrix**

*TODO: Fit the KNN-based collaborative filtering model using the trainset and evaluate the results using the testset:*

*## WRITE YOUR CODE HERE:*

*# - Define a KNNBasic() model*

*# Note there are some arguments such as:*

*# max\_k and min\_k, representing the max and min number of neighors for rating estimations*

*# sim\_option, representing similarity measurement such as cosine and whether you want it to be user\_based or items\_based*

*# e.g., sim\_option = {*

*# 'name': 'cosine', 'user\_based': False,*

*# }*

*#*

*# more KNN model hyperparamets can be found here:*

*# https://surprise.readthedocs.io/en/stable/knn\_inspired.html*

*#*

*# You may try different hyperparamet combinations to see which one has the best performance*

*# - Train the KNNBasic model on the trainset, and predict ratings for the testset*

*# - Then compute RMSE*

To learn more detailed usages about *Surprise* library, visit its website from [here](https://surprise.readthedocs.io/en/stable/getting_started.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMML321ENSkillsNetwork32585014-2022-01-01)

**Implementation Option 2: Use numpy, pandas, and sklearn**

If you want more hardcore coding practices, you may implement the KNN model using numpy, pandas, and possibly sklearn:

*## WRITE YOUR CODE HERE:*

*## One solution could be:*

*## - Calculate the similarity between two users using their rating history (the row vectors of interaction matrix)*

*## - Build a similarity matrix for each pair of users with the training dataset*

*## - For each user, find its k nearest neighbors in the sim matrix*

*## - For each rating in the test dataset, estimate its rating using the KNN collaborative filtering equations shown before*

*## - Calculate RMSE for the entire test dataset*