Collaborative Filtering based Recommender System using K Nearest Neighbor

Estimated time needed: **60** minutes

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:

User-Item interaction matrix

	Machine Learning With Python	Machine Learning 101	Machine Learning Capstone	SQL with Python	Python 101
...	...	...	...	...	
user2	3.0	3.0	3.0	3.0	3.0
user3	2.0	3.0	3.0	2.0	
user4	3.0	3.0	2.0	2.0	3.0
user5	2.0	3.0	3.0		
user6	3.0	3.0	?		3.0
...	...	...	...	...	

Similar users

Predict the rating of user *user6* to item *Machine Learning Capstone*

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 u to item i, $\hat{r}_{ui}$ is given by:

User-based collaborative filtering:

$$\hat{r}_{ui} = \frac{ \sum\limits_{v \in N^k_i(u)} \text{similarity}(u, v) \cdot r_{vi}} {\sum\limits_{v \in N^k_i(u)} \text{similarity}(u, v)}$$

Item-based collaborative filtering:

$$\hat{r}_{ui} = \frac{ \sum\limits_{j \in N^k_u(i)} \text{similarity}(i, j) \cdot r_{uj}} {\sum\limits_{j \in N^k_u(i)} \text{similarity}(i, j)}$$

Here $N^k_i(u)$ notates the nearest k neighbors of u.

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`:

In [1]:
```
import numpy as np
import math
```

In [2]:
```
# An example similarity array stores the similarity of user2, user3, user4, and
knn_sims = np.array([0.8, 0.92, 0.75, 0.83])
```

Also their rating on the `Machine Learning Capstone` course are:

In [3]:
```
# 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:

In [4]:
```
r_u6_ml =  np.dot(knn_sims, knn_ratings)/ sum(knn_sims)
r_u6_ml
```

Out[4]:
```
2.7727272727272725
```

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

In [5]:
```
true_rating = 3.0
rmse = math.sqrt(true_rating - r_u6_ml) ** 2
rmse
```

Out[5]:
```
0.22727272727272751
```

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 lab 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

In [6]:
```
import pandas as pd
```

In [7]:
```
rating_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud
rating_df = pd.read_csv(rating_url)
```

In [8]:
```
rating_df.head()
```

Out[8]:

	user	item	rating
0	1889878	CC0101EN	5
1	1342067	CL0101EN	3
2	1990814	ML0120ENv3	5
3	380098	BD0211EN	5
4	779563	DS0101EN	3

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` :

In [9]:
```
rating_sparse_df = rating_df.pivot(index='user', columns='item', values='rating'
rating_sparse_df.head()
```

Out[9]:

	user	AI0111EN	BC0101EN	BC0201EN	BC0202EN	BD0101EN	BD0111EN	BD0115E
0	2	0.0	4.0	0.0	0.0	5.0	4.0	(
1	4	0.0	0.0	0.0	0.0	5.0	3.0	4
2	5	3.0	5.0	5.0	0.0	4.0	0.0	(
3	7	0.0	0.0	0.0	0.0	0.0	0.0	(
4	8	0.0	0.0	0.0	0.0	0.0	3.0	(

5 rows × 127 columns

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 may choose one of the two following implementation 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 (recommended)

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:

In [10]:
```
!pip install scikit-surprise
```

```
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.1.tar.gz (11.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 76.3 MB/s eta 0:00:000
0:0100:01
  Preparing metadata (setup.py) ... done
Collecting joblib>=0.11 (from scikit-surprise)
  Downloading joblib-1.3.2-py3-none-any.whl (302 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.2/302.2 kB 20.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.11.2 in /home/jupyterlab/conda/envs/pytho
n/lib/python3.7/site-packages (from scikit-surprise) (1.21.6)
Requirement already satisfied: scipy>=1.0.0 in /home/jupyterlab/conda/envs/pytho
n/lib/python3.7/site-packages (from scikit-surprise) (1.7.3)
Requirement already satisfied: six>=1.10.0 in /home/jupyterlab/conda/envs/python/
lib/python3.7/site-packages (from scikit-surprise) (1.16.0)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.1-cp37-cp37m-li
nux_x86_64.whl size=781807 sha256=4685f181a99f9cb336a33e01150cf0e4185a2fdcc3d0c86
afd8706ffab1571c4
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/76/44/74/b498c42be47b24
06bd27994e16c5188e337c657025ab400c1c
Successfully built scikit-surprise
Installing collected packages: joblib, scikit-surprise
Successfully installed joblib-1.3.2 scikit-surprise-1.1.1
```

Now we import required classes and methods

In [11]:
```
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.

In [12]:
```
# 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)
```

```
Trying to download dataset from http://files.grouplens.org/datasets/movielens/ml-
100k.zip...
Done! Dataset ml-100k has been saved to /home/jupyterlab/.surprise_data/ml-100k
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9866
```

Out[12]:
```
0.9866482148897945
```

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:

In [13]:
```
# Save the rating dataframe to a CSV file
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))

# Load the dataset from the CSV file
course_dataset = Dataset.load_from_file("course_ratings.csv", reader=reader)
```

We split it into trainset and testset:

In [14]:
```
trainset, testset = train_test_split(course_dataset, test_size=.3)
```

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

In [15]:
```
print(f"Total {trainset.n_users} users and {trainset.n_items} items in the train
```

```
Total 31243 users and 125 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:

In [16]:
```
# Define similarity options
# item-based collaborative filtering using cosine similarity
sim_options = {
    'name': 'cosine',
    'user_based': False  # False → item-based CF, True → user-based CF
}

# Define KNN model
algo = KNNBasic(
    k=40,              # number of neighbors
    min_k=1,
    sim_options=sim_options
)

# Train the model
algo.fit(trainset)

# Predict ratings for test set
predictions = algo.test(testset)
```

```
# Compute RMSE
rmse = accuracy.rmse(predictions)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 1.2872
```

▶ Click here for Hints

To learn more detailed usages about *Surprise* library, visit its website from here

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

If you do not prefer the one-stop Suprise solution and want more hardcore coding practices, you may implement the KNN model using `numpy`, `pandas`, and possibly `sklearn`:

In [17]:
```
import pandas as pd
import numpy as np

# Load ratings
rating_df = pd.read_csv(rating_url)

# User-item interaction matrix
user_item_matrix = rating_df.pivot_table(
    index='user',
    columns='item',
    values='rating'
).fillna(0)

user_item_matrix.head()
```

Out[17]:

item	AI0111EN	BC0101EN	BC0201EN	BC0202EN	BD0101EN	BD0111EN	BD0115EN
user							
2	0.0	4.0	0.0	0.0	5.0	4.0	0.0
4	0.0	0.0	0.0	0.0	5.0	3.0	4.0
5	3.0	5.0	5.0	0.0	4.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	3.0	0.0

5 rows × 126 columns

Summary

In this lab, you have learned and implemented KNN-based collaborative filtering. It is probably the simplest but very effective and intuitive collaborative filtering algorithm.

Since it is based on KNN, it inherits the main characteristics of KNN such as memory-intensive because you need to maintain a huge similarity matrix among users or items. In the future labs, we will learn other types of collaborative filtering which do not rely on such a huge similarity matrix to make rating predictions.

Authors

Yan Luo

Other Contributors

```
toggle##
```

```
toggle|Date
```

```
toggle|-|-|-|-|
```

```
toggle|2021-10-25|1.0|Yan|Created
```