Skills Network

Collaborative Filtering based Recommender System using Non-negative Matrix Factorization

Estimated time needed: **60** minutes

In the previous lab, we have performed KNN on user-item interaction matrix to estimate the rating of unknown items based on the aggregation of the user's K nearest neighbor's ratings. Finding nearest neighbors are based on similarity measurements among users or items with big similarity matrices.

The KNN algorithm is memory-based which means we need to keep all instances for prediction and maintain a big similarity matrix. These can be infeasible if our user/item scale is large, for example, 1 million users will require a 1 million by 1 million similarity matrix, which is very hard to load into RAM for most computation environments.

Non-negative matrix factorization

In the machine learning course, you have learned a dimensionality reduction algorithm called Non-negative matrix factorization (NMF), which decomposes a big sparse matrix into two smaller and dense matrices.

Non-negative matrix factorization can be one solution to big matrix issues. The main idea is to decompose the big and sparse user-interaction into two smaller dense matrices, one represents the transformed user features and another represents the transformed item features.

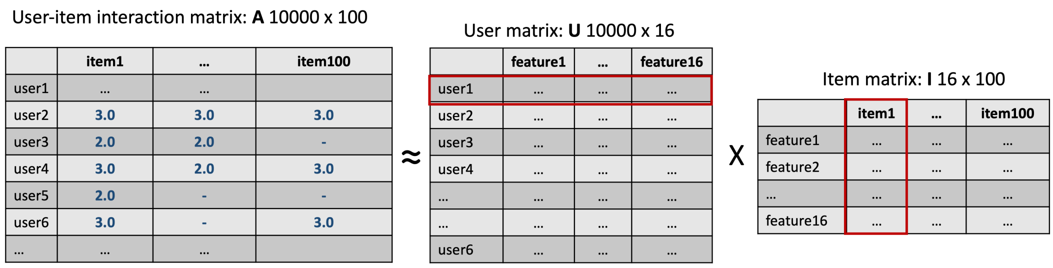
An example is shown below, suppose we have a user-item interaction matrix A with 10000 users and 100 items (10000 x 100), and its element `(j, k)` represents the rating of item `k` from user `j`. Then we could decompose A into two smaller and dense matrices U (10000 x 16) and I (16 x 100). for user matrix U, each row vector is a transformed latent feature vector of a user, and for the item matrix I, each column is a transformed latent feature vector of an item.

Here the dimension 16 is a hyperparameter defines the size of the hidden user and item features, which means now the shape of transposed user feature vector and item feature vector is now 16 x 1.

The magic here is when we multiply the row `j` of U and column `k` of matrix I, we can get an estimation to the original rating $\hat{r}_{jk}$.

For example, if we preform the dot product user ones row vector in U and item ones column vector in I, we can get the rating estimation of user one to item one, which is the element (1, 1) in the original interaction matrix I.

Non-negative Matrix Factorization



Note I is short for Items, and it is not an identity matrix.

Then how do we figure out the values in U and I exactly? Like many other machine learning processes, we could start by initializing the values of U and I, then define the following distance or cost function to be minimized:

$$\sum_{r_{jk} \in {train}} \left(r_{jk} - \hat{r}_{jk} \right)^2,$$

where $\hat{r}_{ij}$ is the dot product of u_j^T and i_k:

$$\hat{r}_{jk} = u_j^Ti_k$$

The cost function can be optimized using stochastic gradient descent (SGD) or other optimization algorithms, just like in training the weights in a logistic regression model (there are several additional steps so the matrices have no negative elements) .

Objectives

After completing this lab you will be able to:

- Perform NMF-based collaborative filtering on the user-item matrix

---

Load and exploring dataset

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

In [1]:
```
import pandas as pd
```

In [2]:
```
rating_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud
rating_df = pd.read_csv(rating_url)
```

In [3]:
```
rating_df.head()
```

Out[3]:

	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`, `item id`, and `the rating`. Note that this matrix is presented as the dense or vertical form, you may convert it using `pivot` to the original sparse matrix:

In [4]:
```
rating_sparse_df = rating_df.pivot(index='user', columns='item', values='rating'
rating_sparse_df.head()
```

Out[4]:

	user	AI0111EN	BC0101EN	BC0201EN	BC0202EN	BD0101EN	BD0111EN	BD0115I
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

Next, you need to implement NMF-based collaborative filtering, and you may choose one of the two following implementation options:

- The first one is to use `Surprise` which is a popular and easy-to-use Python recommendation system library.
- The second way is to implement it with `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 scikit library for recommender systems. It is simple and comprehensive to build and test different recommendation algorithms. First let's install it:

In [5]:
```
!pip install scikit-surprise
```

```
Requirement already satisfied: scikit-surprise in /home/jupyterlab/conda/envs/pyt
hon/lib/python3.7/site-packages (1.1.1)
Requirement already satisfied: joblib>=0.11 in /home/jupyterlab/conda/envs/pytho
n/lib/python3.7/site-packages (from scikit-surprise) (1.3.2)
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)
```

We import required classes and methods

In [6]:
```
from surprise import NMF
from surprise import Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import accuracy
```

In [7]:
```
# 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_sc

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

Now we split the data into a train-set and test-set:

In [8]:
```
trainset, testset = train_test_split(course_dataset, test_size=.3)
```

Then check how many users and items we can use to fit the KNN model:

In [9]:
```
print(f"Total {trainset.n_users} users and {trainset.n_items} items in the train
```

```
Total 31393 users and 125 items in the trainingset
```

TASK: Perform NMF-based collaborative filtering on the course-interaction matrix

TODO: Fit a NMF model using the trainset and evaluate the results using the testset The code will be very similar to the KNN-based collaborative filtering, you just need to use the `NMF()` model.

In [10]:
```
# 1. Define the NMF model
nmf_model = NMF(
    n_factors=16,          # latent factors
    n_epochs=50,           # training iterations
    random_state=123,
    verbose=True
)

# 2. Train the model on the trainset
nmf_model.fit(trainset)

# 3. Predict ratings for the testset
```

```python
predictions = nmf_model.test(testset)

# 4. Evaluate using RMSE
rmse = accuracy.rmse(predictions)
```

```
Processing epoch 0
Processing epoch 1
Processing epoch 2
Processing epoch 3
Processing epoch 4
Processing epoch 5
Processing epoch 6
Processing epoch 7
Processing epoch 8
Processing epoch 9
Processing epoch 10
Processing epoch 11
Processing epoch 12
Processing epoch 13
Processing epoch 14
Processing epoch 15
Processing epoch 16
Processing epoch 17
Processing epoch 18
Processing epoch 19
Processing epoch 20
Processing epoch 21
Processing epoch 22
Processing epoch 23
Processing epoch 24
Processing epoch 25
Processing epoch 26
Processing epoch 27
Processing epoch 28
Processing epoch 29
Processing epoch 30
Processing epoch 31
Processing epoch 32
Processing epoch 33
Processing epoch 34
Processing epoch 35
Processing epoch 36
Processing epoch 37
Processing epoch 38
Processing epoch 39
Processing epoch 40
Processing epoch 41
Processing epoch 42
Processing epoch 43
Processing epoch 44
Processing epoch 45
Processing epoch 46
Processing epoch 47
Processing epoch 48
Processing epoch 49
RMSE: 1.3018
```

▶ 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, you may implement the KNN model using `numpy`, `pandas`, and possibly `sklearn`:

In [11]:
```
# Pivot to user-item matrix
rating_matrix = rating_df.pivot(
    index='user',
    columns='item',
    values='rating'
).fillna(0)

rating_matrix.head()
```

Out[11]:

item user	AI0111EN	BC0101EN	BC0201EN	BC0202EN	BD0101EN	BD0111EN	BD0115EN
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 practiced NMF-based collaborative filtering. The basic idea is to decompose the original user-item interaction matrix into two smaller and dense user and item matrices. Then, we have built the two matrices, we can easily estimate the unknown ratings via the dot product of specific row in user matrix and specific column in item matrix.

Authors

Yan Luo

Other Contributors

```
Toggle##

Toggle|Date

Toggle|-|-|-|-|

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