IBCF (item based collaborative filtering) - Recommendation System

We have used the MovieLens dataset consisting of 100K ratings provided by 941 users across 1682 items for implementing IBCF.

In [1]:

```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
```

```
Bad key "text.kerning_factor" on line 4 in
C:\Users\91920\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_cla
ssic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
 (https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.templat
e)
or from the matplotlib source distribution
```

Read the dataset using Pandas library

In [2]:

```
import pandas as pd
header = ['user_id','item_id','rating','timestamp']
dataset = pd.read_csv('C:/Users/91920/Downloads/Compressed/ml-100k/ml-100k/u.data',sep = '\
print(dataset.head())
```

```
   user_id  item_id  rating  timestamp
0      196      242       3  881250949
1      186      302       3  891717742
2       22      377       1  878887116
3      244       51       2  880606923
4      166      346       1  886397596
```

We transform the dataset into a matrix where each row represents the user and column represents the item.

In [4]:

```
A = dataset.pivot_table(values='rating',index=['user_id'],columns=['item_id'])
A
```

Out[4]:

item_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676
user_id															
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	NaN	NaN	NaN	NaN
2	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5	4.0	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN	...	NaN	NaN	NaN	NaN
940	NaN	NaN	NaN	2.0	NaN	NaN	4.0	5.0	3.0	NaN	...	NaN	NaN	NaN	NaN
941	5.0	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
942	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
943	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	...	NaN	NaN	NaN	NaN

943 rows × 1682 columns

The MovieLens dataset consists of ratings on a scale of 1-5 where 1 represents the lowest rating while 5 represents the highest rating. However, different ratings could have different meanings to users. For instance, a rating of 3 might be good for one user while average for another user.

To solve this ambiguity, big giants such as Netflix or YouTube have moved to binary ratings. Therefore, in this blog, we will work on binary ratings instead of continuous ratings to keep ourselves in sync with the latest research.

The below code converts the MovieLens dataset into the binary MovieLens dataset. We have considered items whose ratings are greater or equal to 3 being liked by the user and others being disliked by the user. As we are only considerate about the liking of users, making ratings less than 3 as 0 would not impact the recommendation process.

In [5]:

```
A.fillna(0 , inplace=True)
A
```

Out[5]:

item_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678
user_id																	
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0
2	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
940	0.0	0.0	0.0	2.0	0.0	0.0	4.0	5.0	3.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
941	5.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
942	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
943	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0

943 rows × 1682 columns

In [6]:

```
for coloums in A:
    A[coloums].mask(A[coloums] <3 ,0, inplace=True)
    A[coloums].mask(A[coloums] >=3 ,1, inplace=True)

print(A)
```

```
item_id  1     2     3     4     5     6     7     8     9     10    ...  \
user_id                                                              ...
1         1.0   1.0   1.0   1.0   1.0   1.0   1.0   0.0   1.0   1.0  ...
2         1.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
3         0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
4         0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
5         1.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
...       ...   ...   ...   ...   ...   ...   ...   ...   ...   ...  ...
939       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0   0.0  ...
940       0.0   0.0   0.0   0.0   0.0   0.0   1.0   1.0   1.0   0.0  ...
941       1.0   0.0   0.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0  ...
942       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
943       0.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0   0.0  ...

item_id  1673  1674  1675  1676  1677  1678  1679  1680  1681  1682
user_id
1         0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
2         0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
3         0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
4         0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
5         0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
...       ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
939       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
940       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
941       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
942       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
943       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

[943 rows x 1682 columns]
```

how many movie like by user 1 (for prediction)

```
data= A.iloc[1]
data.value_counts()
```

To save the memory, we convert the dense rating matrix into a sparse matrix using the csr_matrix() function.

In [7]:

```
csr_sample = csr_matrix(A)
print(csr_sample)
```

```
  (0, 0)        1.0
  (0, 1)        1.0
  (0, 2)        1.0
  (0, 3)        1.0
  (0, 4)        1.0
  (0, 5)        1.0
  (0, 6)        1.0
  (0, 8)        1.0
  (0, 9)        1.0
  (0, 11)       1.0
  (0, 12)       1.0
  (0, 13)       1.0
  (0, 14)       1.0
  (0, 15)       1.0
  (0, 16)       1.0
  (0, 17)       1.0
  (0, 18)       1.0
  (0, 19)       1.0
  (0, 21)       1.0
  (0, 22)       1.0
  (0, 23)       1.0
  (0, 24)       1.0
  (0, 25)       1.0
  (0, 27)       1.0
  (0, 29)       1.0
  :     :
  (942, 624)    1.0
  (942, 654)    1.0
  (942, 671)    1.0
  (942, 684)    1.0
  (942, 716)    1.0
  (942, 720)    1.0
  (942, 721)    1.0
  (942, 731)    1.0
  (942, 738)    1.0
  (942, 762)    1.0
  (942, 764)    1.0
  (942, 793)    1.0
  (942, 795)    1.0
  (942, 807)    1.0
  (942, 815)    1.0
  (942, 823)    1.0
  (942, 824)    1.0
  (942, 839)    1.0
  (942, 927)    1.0
  (942, 942)    1.0
  (942, 1043)   1.0
  (942, 1073)   1.0
  (942, 1187)   1.0
  (942, 1227)   1.0
  (942, 1329)   1.0
```

Compute similarity between items of csr_sample using cosine similarity for simple task of finding the nearest neighbors

In [8]:

```
knn = NearestNeighbors(metric='cosine', n_neighbors=3, n_jobs=-1)
knn.fit(csr_sample)
```

Out[8]:

```
NearestNeighbors(metric='cosine', n_jobs=-1, n_neighbors=3)
```

Generate Recommendations

Here, we are generating recommendations for the user_id: 1.

We generate recommendations for user_id:1 based on 20 items being liked by him. So, we first get the 20 items being liked/consumed by the user as shown below:

In [9]:

```
dataset_sort_des = dataset.sort_values(['user_id', 'timestamp'], ascending=[True, False])
filter1 = dataset_sort_des[dataset_sort_des['user_id'] == 1].item_id
filter1 = filter1.tolist()
filter1 = filter1[:20]
print("Items liked by user: ",filter1) #these 20 movies has been watch by user recently.
```

```
Items liked by user:  [74, 102, 256, 5, 171, 111, 242, 189, 32, 209, 270, 1
8, 6, 244, 221, 129, 20, 271, 272, 255]
```

Next, for each item being liked by the user1, we recommend 1 similar items. The number of similar items to be recommended can vary depending on the need of the system.

In [10]:

```
distances1=[]
indices1=[]
for i in filter1:
    distances , indices = knn.kneighbors(csr_sample[i],n_neighbors=2) # 1 movie for each mo
    indices = indices.flatten()
    indices= indices[1:]
    indices1.extend(indices)
print("Items to be recommended: ",indices1)
```

```
Items to be recommended:  [356, 758, 883, 473, 311, 771, 614, 904, 510, 642,
473, 578, 312, 688, 681, 275, 365, 685, 719, 373]
```

In [11]:

```
# to show only
distances , indices = knn.kneighbors(csr_sample[74],n_neighbors=2)
indices.flatten()
```

Out[11]:

```
array([ 74, 356], dtype=int64)
```

Ques: Why we are using extend not append ?

In [ ]: