Activity 1: *Collaborative Filtering*

- Give a high-level explanation of Collaborative Filtering.
    - What does it do, and how?
    - What are the inputs and outputs?
    - Can you give an example?
- What is the underlying assumption of Collaborative Filtering?
- What are the main steps of collaborative filtering?
- To find similar users, we need a metric to measure the similarity between users:
    - What is the Jaccard similarity, and why its usefulness is limited in this application?
    - How does the cosine similarity treat the missing ratings? As positive? As negative?
    - What is the Centered Cosine Similarity?
- After finding the similar users, there are two options to make the final rating prediction. What are they?

Activity 2: *User-User Collaborative Filtering vs item-item Collaborative Filtering*

- What are the key differences between user-user collaborative filtering and item-item collaborative filtering?
- In practice, does the item-item method perform similarly to user-user method? If not, why?

Please revisit page 22 of Week 11's Lecture slides.

Practical exercise 1: Centered Cosine Similarity-Based Collaborative Filtering

This week, let's first explore the centered cosine similarity in collaborative filtering:

```
In [1]: import numpy as np
In [2]: import pandas as pd
```

```
In [3]: cd ml-100k/
/Users/yongli/A-Teaching/PDS/code/recsys/ml-100k

In [4]: names = ['user_id', 'item_id', 'rating', 'timestamp']
In [5]: df = pd.read_csv('u.data', sep='\t', names=names)
In [6]: df.head()
Out [6]:
```

	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

```
In [7]: n_users = df.user_id.unique().shape[0]
In [8]: n_items = df.item_id.unique().shape[0]

In [9]: print(str(n_users) + ' users')
Out [9]: 943 users

In [10]: print(str(n_items) + ' items')
Out [10]: 1682 items

In [11]: i_cols = ['movie id', 'movie title' ,'release date','video release
date', 'IMDb URL', 'unknown', 'Action', 'Adventure', 'Animation',
'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller',
'War', 'Western']
In [12]: items = pd.read_csv('u.item', sep='|', names=i_cols,
encoding='latin-1')
In [13]: items.head()
Out [29]:
```

	movie id	movie title	release date	video release date	IMDb URL	unknown	Action	Adventure	Animation	Children's	...	Fantasy	Film-Noir	Horror	Mus
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1	1	...	0	0	0	0
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0	0	...	0	0	0	0
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0	0	0	...	0	0	0	0
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	1	0	0	0	...	0	0	0	0
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0	0	0	...	0	0	0	0

5 rows × 24 columns

```
# create ratings to store the original ratings
In [14]: ratings = np.zeros((n_users, n_items))
# create ratingsBinary to store a binary value, indicating whether a user
# rated an item.
In [15]: ratingsBinary = np.zeros((n_users, n_items))

In [16]: for row in df.itertuples():
             ratings[row[1]-1, row[2]-1] = row[3]
             ratingsBinary[row[1]-1, row[2]-1] = 1

In [17]: ratings_df = pd.DataFrame(ratings)

In [18]: ratings_userRateNum = ratingsBinary.sum(axis = 1)
In [19]: ratings_sum = ratings.sum(axis = 1)

# calculate the average rating for each user
In [20]: ratings_userAvg = ratings_sum/ratings_userRateNum

# subtract the average rating for each rating.
In [21]: ratings_subtractMean = np.zeros((n_users, n_items))
In [22]: for row in df.itertuples():
             ratings_subtractMean[row[1]-1, row[2]-1] = row[3] -
ratings_userAvg[row[1]-1]

In [23]: ratings_subtractMean_df = pd.DataFrame(ratings_subtractMean)
In [24]: ratings_subtractMean_df.head()
Out [24]:
```

	0	1	2	3	4	5	6	7	8	9	...	1672	1673	1674	1675	1676	1677
0	1.389706	-0.610294	0.389706	-0.610294	-0.610294	1.389706	0.389706	-2.610294	1.389706	-0.610294	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.290323	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-1.709677	...	0.0	0.0	0.0	0.0	0.0	0.0
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0
4	1.125714	0.125714	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0

```
# calculate the cosine similarity between users.
In [25]: from sklearn.metrics.pairwise import cosine_similarity
In [26]: userSimilarity = cosine_similarity(ratings_subtractMean,
ratings_subtractMean)

In [27]: data_ubs_fast = pd.DataFrame(userSimilarity)

# create the data_pre dataframe to store the rating predictions
In [28]: data_pre = pd.DataFrame(index=range(0,n_users),columns=items['movie
title'])

# a very small value to avoid the error of division by zero in line [28]
In [29]: epsilon=1e-9

# a nested loop statement to make predictions for those missing values in the
user-item rating matrix.
# To prevent line ([30]) runs for a long time, we only try the first three
user here: while.
# If you want to run it for all users, you can change line [30] to
#      for i in range(0,len(data_pre.index))::
# This will make the rating predictions on the first 3 users only.
In [30]: for i in range(0,3):
In [31]:     for j in range(0,len(data_pre.columns)):
                    if ratings_df.iloc[i][j] > 0:
                           data_pre.iloc[i][j] = 0
                    else:
                           mask_rated_users = ratings_df[j] > 0
                           top_neighbours =
data_ubs_fast.loc[mask_rated_users,i].sort_values(ascending=False)[0:10].index
                           top_neighbours_sim =
data_ubs_fast.loc[mask_rated_users,i].sort_values(ascending=False)[0:10].value
s
                    neighbours_ratings = ratings_df.iloc[top_neighbours,j]

                    data_pre.iloc[i][j] =
sum(neighbours_ratings*top_neighbours_sim)/(sum(top_neighbours_sim) + epsilon)
```

```
In [32]: data_pre.head(3)
Out [32]:
```

movie title	Toy Story (1995)	GoldenEye (1995)	Four Rooms (1995)	Get Shorty (1995)	Copycat (1995)	Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)	Twelve Monkeys (1995)	Babe (1995)	Dead Man Walking (1995)	Richard III (1995)	...	Mirage (1995)	Mamma Roma (1962)	Sunchaser, The (1996)	War at Home, The (1996)	Sv Nc (19
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	3.11376	2.32628	3.10641	2.895	3.76188	3.42424	3.94469	4.37541	0	...	0	0	0	0	0
2	3.67194	3.36721	3.30875	3.7235	2.49628	3.00134	3.59595	3.7042	3.50485	3.38278	...	0	0	0	0	0

3 rows × 1682 columns

```
In [33]: data_recommend = pd.DataFrame(index=data_pre.index,
columns=['1','2','3','4','5','6'])

In [34]: for i in range(0,len(data_pre.index)):
             data_recommend.iloc[i,0:] =
data_pre.iloc[i,:].sort_values(ascending=False)[0:6].index

In [35]: data_recommend.head(3)
Out [35]:
```

	1	2	3	4	5	6
0	Sliding Doors (1998)	Boys, Les (1997)	Twisted (1996)	Lay of the Land, The (1997)	Gold Diggers: The Secret of Bear Mountain (1995)	Bitter Sugar (Azucar Amargo) (1996)
1	Wooden Man's Bride, The (Wu Kui) (1994)	Flirt (1995)	Denise Calls Up (1995)	Gold Diggers: The Secret of Bear Mountain (1995)	Hurricane Streets (1998)	Twisted (1996)
2	Awfully Big Adventure, An (1995)	Mercury Rising (1998)	Bad Moon (1996)	Trial by Jury (1994)	I'll Do Anything (1994)	Whole Wide World, The (1996)

Practical exercise 2: Comparing with Cosine Similarity

Next, we investigate the difference between the cosine similarity and the centered cosine similarity.

Specifically, please do the following:

1. Build the cosine similarity-based collaborative filtering. (You can find the example code on page 23 - 29 in Week 10's Slides)
2. Compare the recommendation lists for users 0, 1 and 2 with the Centred Cosine Similarity-based method (as shown in Practical exercise 1)
    a. Are they the same?
    b. Why are they different?
    c. Which one is more appropriate?

Practical exercise 3: Exploring Another Metric to Measure the Similarity Between Users

We explore another metric, Euclidean distance to measure the similarity (distance) between users for recommendation purposes.

1. Replace the inline `[25]` and `[26]` with the following two lines.

    ```
    In [36]: from sklearn.metrics.pairwise import euclidean_distances
    In [37]: userSimilarity = euclidean_distances(ratings_subtractMean,
    ratings_subtractMean)
    ```

2. Replace the following two lines (in `[31]`)

    ```
    top_neighbours =
    data_ubs_fast.loc[mask_rated_users,i].sort_values(ascending=False)[0:10].index
    top_neighbours_sim =
    data_ubs_fast.loc[mask_rated_users,i].sort_values(ascending=False)[0:10].value
    s
    ```

    With

    ```
    top_neighbours =
    data_ubs_fast.loc[mask_rated_users,i].sort_values(ascending=True)[0:10].index
    top_neighbours_sim =
    data_ubs_fast.loc[mask_rated_users,i].sort_values(ascending=True)[0:10].values
    ```

    (Please think why "`ascending=False`" are changed to "`ascending=True`".)
3. Then, run the whole code, and explore the recommendation list for the active user.

Can you try other measurement metrics to measure the similarity between users/items?

The following link shows the available Pairwise metrics in sklearn.

(http://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.pairwise)