Recommender Systems: Collaborative Filtering(CF): Memory-Based CosineSimilarity

Data: MovieLens dataset It contains 100k movie ratings from 943 users and a selection of 1682 movies.

You can download the dataset here (http://files.grouplens.org/datasets/movielens/ml-100k.zip).

u.data file: contains the full dataset. description of the dataset here (http://files.grouplens.org/datasets/movielens/ml-100k-README.txt).

u.data -- The full u data set, 100000 ratings by 943 users on 1682 items. Each user has rated at least 20 movies. Users and items are numbered consecutively from 1. The data is randomly ordered.

```
 Note: This is a **tab** separated list of
**user id | item id | rating | timestamp**.
```

Import libs

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

Load Data

In [2]:
```
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
data_org = pd.read_csv('data/u.data', sep='\t', names=column_names)
```

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

Out[3]:

	user_id	item_id	rating	timestamp
0	0	50	5	881250949
1	0	172	5	881250949
2	0	133	1	881250949
3	196	242	3	881250949
4	186	302	3	891717742

We have item_id, which is not the movie name.

Use the Movie_ID_Titles csv file to grab the movie names and merge it with this dataframe:

In [4]:
```
movie_titles = pd.read_csv("data/Movie_Id_Titles")
movie_titles.head()
```

Out[4]:

	item_id	title
0	1	Toy Story (1995)
1	2	GoldenEye (1995)
2	3	Four Rooms (1995)
3	4	Get Shorty (1995)
4	5	Copycat (1995)

Both data_org anf movie_titles have 'item_id' in common, merge by that.

In [5]:
```
data_org.shape
```

Out[5]: (100003, 4)

In [6]:
```
movie_titles.shape
```

Out[6]: (1682, 2)

In [7]:
```
data= pd.merge(data_org,movie_titles,on='item_id')
data.head()
```

Out[7]:

	user_id	item_id	rating	timestamp	title
0	0	50	5	881250949	Star Wars (1977)
1	290	50	5	880473582	Star Wars (1977)
2	79	50	4	891271545	Star Wars (1977)
3	2	50	5	888552084	Star Wars (1977)
4	8	50	5	879362124	Star Wars (1977)

In [8]:
```
data.shape
```

Out[8]: (100003, 5)

Tarin/Test Split

In [9]:
```
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(data, test_size=0.25)
```

In [10]:
```
print("df dimension={}".format(data.shape))
print("train_data dimension={}".format(train_data.shape))
print("test_data dimension={}".format(test_data.shape))
```

```
df dimension=(100003, 5)
train_data dimension=(75002, 5)
test_data dimension=(25001, 5)
```

In [11]:
```
train_data.head(3)
```

Out[11]:

	user_id	item_id	rating	timestamp	title
71365	70	380	3	884066399	Star Trek: Generations (1994)
94128	298	946	3	884182868	Fox and the Hound, The (1981)
95666	833	940	2	875134411	Airheads (1994)

Create two user_id - item_id matrices, one for training and another for testing

Each element is the rating

In [12]:
```
n_users = data.user_id.nunique()
n_items = data.item_id.nunique()

print('Num. of Users: '+ str(n_users))
print('Num of Movies: '+str(n_items))
```

```
Num. of Users: 944
Num of Movies: 1682
```

In [13]:
```
train_data_matrix = np.zeros((n_users, n_items))
for line in train_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1] = line[3]

test_data_matrix = np.zeros((n_users, n_items))
for line in test_data.itertuples():
    test_data_matrix[line[1]-1, line[2]-1] = line[3]
```

Calculate the cosine similarity

cosine similarity, where the ratings are seen as vectors in n -dimensional space and the similarity is calculated based on the angle between these vectors. Cosine similiarity for users *a* and *m* can be calculated using the formula below, where you take dot product of the user vector u_k and the user vector u_a and divide it by multiplication of the Euclidean lengths of the vectors.

$$s_u^{cos}(u_k, u_a) = \frac{u_k \cdot u_a}{||u_k||\,||u_a||} = \frac{\sum x_{k,m}x_{a,m}}{\sqrt{\sum x_{k,m}^2 \sum x_{a,m}^2}}$$

To calculate similarity between items *m* and *b* you use the formula:

$$s_u^{cos}(i_m, i_b) = \frac{i_m \cdot i_b}{||i_m||\,||i_b||} = \frac{\sum x_{a,m}x_{a,b}}{\sqrt{\sum x_{a,m}^2 \sum x_{a,b}^2}}$$

*For user and also for items.

*using pairwise_distances function from sklearn.

Note: since the ratings are all positive, the similarity values will range from 0 to 1

In [14]:
```
from sklearn.metrics.pairwise import pairwise_distances

user_similarity = pairwise_distances(train_data_matrix, metric='cosine')#user
 x movie = (944, 1682)
print("user_similarity dimension:{}".format(user_similarity.shape))

item_similarity = pairwise_distances(train_data_matrix.T, metric='cosine')
print("item_similarity dimension:{}".format(item_similarity.shape))
```

```
user_similarity dimension:(944, 944)
item_similarity dimension:(1682, 1682)
```

Prediction

prediction by applying following formula for user-based CF:

$$\hat{x}_{k,m} = \bar{x}_k + \frac{\sum\limits_{u_a} sim_u(u_k, u_a)(x_{a,m} - \bar{x_{u_a}})}{\sum\limits_{u_a} |sim_u(u_k, u_a)|}$$

You can look at the similarity between users *k* and *a* as weights that are multiplied by the ratings of a similar user *a* (corrected for the average rating of that user). You will need to normalize it so that the ratings stay between 1 and 5 and, as a final step, sum the average ratings for the user that you are trying to predict.

The idea here is that some users may tend always to give high or low ratings to all movies. The relative difference in the ratings that these users give is more important than the absolute values. To give an example: suppose, user *k* gives 4 stars to his favourite movies and 3 stars to all other good movies. Suppose now that another user *t* rates movies that he/she likes with 5 stars, and the movies he/she fell asleep over with 3 stars. These two users could have a very similar taste but treat the rating system differently.

When making a prediction for item-based CF you don't need to correct for users average rating since query user itself is used to do predictions.

$$\hat{x}_{k,m} = \frac{\sum\limits_{i_b} sim_i(i_m, i_b)(x_{k,b})}{\sum\limits_{i_b} |sim_i(i_m, i_b)|}$$

In [15]:
```
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = np.mean(ratings, axis=1, keepdims=True )
        ratings_diff = ratings - mean_user_rating
        pred = mean_user_rating + np.dot(similarity, ratings_diff) / np.sum(np
.abs(user_similarity),axis=1,keepdims=True)
    elif type == 'item':
        pred = np.dot(ratings, similarity) / np.sum(np.abs(similarity), axis=1
, keepdims=True).T
    return pred
```

In [16]:
```
train_data_matrix.shape
```

Out[16]:
```
(944, 1682)
```

In [17]:
```
user_similarity.shape
```

Out[17]:
```
(944, 944)
```

In [18]:
```
item_similarity.shape
```
Out[18]:
```
(1682, 1682)
```
In [19]:
```
user_prediction = predict(train_data_matrix, user_similarity, type='user')
item_prediction = predict(train_data_matrix, item_similarity, type='item')
```
In [20]:
```
user_prediction.shape
```
Out[20]:
```
(944, 1682)
```
In [21]:
```
item_prediction.shape
```
Out[21]:
```
(944, 1682)
```

Evaluation

Root Mean Squared Error (RMSE).

$$RMSE = \sqrt{\frac{1}{N}\sum(x_i - \hat{x}_i)^2}$$

Note that, here we calculate the RMSE on the test data on its non-zero values. Therefore we should find those spots in user_prediction and item_prediction as well.

In [22]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
def rmse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, ground_truth))
```
In [23]:
```
print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
```
```
User-based CF RMSE: 3.128463864065603
Item-based CF RMSE: 3.4558112339709757
```