User-based collaborative filtering

The user-based collaborative filtering algorithm mainly consists of two steps.
1. Find a set of users with similar interests to the target users.
2. Find items in the collection that the user likes and that the target user has not heard of and recommend them to the target user.

Similarity

The key of step 1 is to **calculate the similarity** between two users. The following is a list of two common similarity calculation methods.
Given user U and user V, let N(u) represent the set of items that user U has had positive feedback on, and let N(v) be the set of items that user V has had positive feedback on.

- Jaccard

$$w_{uv} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

- Cosine Similarity

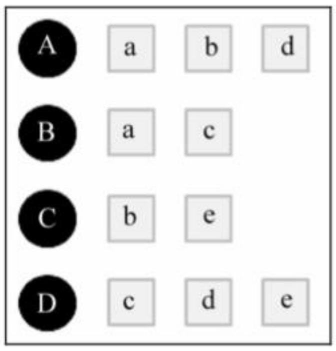
$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)||N(v)|}}$$

- Pearson

$$s_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \overline{r_u})(r_{vi} - \overline{r_v})}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \overline{r_u})^2} \sqrt{\sum_{i \in I_{u,v}} (r_{vi} - \overline{r_v})^2}}$$

Example

Calculate the similarity between two individual users
The following is a brief introduction to the cosine similarity calculation method to get the similarity between two users.



For example, user A has had behavior towards item {A,b,d}, and user B has had behavior towards item {A,c}. The cosine similarity formula is used to calculate the interest similarity between user A and user B as follows:

the similarity between A and B

$$w_{AB} = \frac{|\{a,b,d\} \cap \{a,c\}|}{\sqrt{|\{a,b,d\}|\ |\{a,c\}|}} = \frac{1}{\sqrt{6}}$$

Similarly, we can calculate the similarity between user A and user C and D:

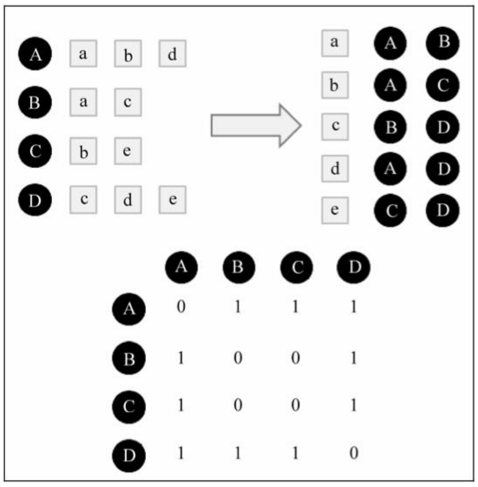
$$w_{AC} = \frac{|\{a,b,d\} \cap \{b,e\}|}{\sqrt{|\{a,b,d\}|\ |\{b,e\}|}} = \frac{1}{\sqrt{6}}$$

$$w_{AD} = \frac{|\{a,b,d\} \cap \{c,d,e\}|}{\sqrt{|\{a,b,d\}|\ |\{c,d,e\}|}} = \frac{1}{3}$$

Calculate the similarity matrix between user pairs

For both users, cosine similarity is used to calculate the similarity. The time complexity of this method is order N^2. It takes a lot of time.
Therefore, we need to establish an inversion list between items and users, so that we can exclude the calculation of similarity between users who have no connection at all.
Then calculate the matrix of the co-rated items from the inversion list.
As shown in the figure below.

Calculate the matrix of co-rating items



Each value in the matrix is the numerator part of the cosine similarity, and then the numerator is divided by the denominator to get the final user interest similarity.
In other words, the matrix of co-rated items in the figure above can be converted into the similarity matrix between users, and only the non-zero part can be calculated.
For example, calculate the user similarity between A and B. The value in the matrix of users A and B is 1, that is, the item they share is 1. A rated 3 items in total, B rated 2 items in total. According to the cosine similarity algorithm, the similarity is.

$$\frac{1}{\sqrt{6}}$$

Select K users who are most similar to the target users

After the similarity between users is obtained, the algorithm will recommend to users the favorite items of K users who are most similar to their interests
The following formula measures user U's interest in item I in the algorithm

$$p(u,i) = \sum_{v \in S(u,K) \cap N(i)} w_{uv} r_{vi}$$

Where, S(u,K) contains K users whose interests are most similar to those of user U, N(*i*) is the set of users who have rated on item *i*, W_{uv} is the interest similarity between user *u* and user *v*, and r_{vi} is 1.

For example

Recommend target user A. Select K=3. User A has no behavior towards items *c* and *e*, so these two items can be recommended to user A. According to the algorithm, user A's interest in item *c* and *e* is:

$$p(A,c) = w_{AB} + w_{AD} = 0.7416$$
$$p(A,e) = w_{AC} + w_{AD} = 0.7416$$

User-based collaborative filtering-python

1. MovieLens

The MovieLens data set contains multiple movie ratings by multiple users, as well as movie metadata information and user attribute information. This data set is often used as a test data set for recommendation systems, machine learning algorithms. Especially in the area of recommendation systems.

The download of the data set is:http://files.grouplens.org/datasets/movielens/， There are several versions with different amounts of data, and the data used for this article is mL-latest-small-zip. In this paper, two data tables, ratings. CSV and Movies.csv, are mainly used:

Ratings Data
userId: The ID of each user
movieId: The ID of each movie
rating: User rating, on a 5-star scale, increments by the size of half a star (0.5 stars 5 Stars)
timestamp: The number of seconds from zero on January 1, 1970 until the time the user submits the evaluation.

The order in which the data is sorted is userId, movieId.

Movies Data
movieId:The ID of each movie
title:Movie title
genres:Movie type

2. Data processing

First, we define the path to our data and use Pandas to read in:

```
import mathimport numpy as npimport pandas as pdfrom sklearn.model_selec
tion import train_test_split



pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)

pd.set_option('display.width', 1000)

moviesPath = ".\\data\\movies.csv"

ratingsPath = ".\\data\\ratings.csv"

moviesDF = pd.read_csv(moviesPath, index_col=None)

ratingsDF = pd.read_csv(ratingsPath, index_col=None)
```

Here we split the data set in a 4:1 ratio, and print out the total number of users and movies, the number of users and movies in the training set, and the number of users and movies in the test set:

```
trainRatingsDF, testRatingsDF = train_test_split(ratingsDF, test_size=0.
2)

print("total_movie_count:" + str(len(set(ratingsDF['movieId'].values.tol
ist()))))

print("total_user_count:" + str(len(set(ratingsDF['userId'].values.tolis
t()))))

print("train_movie_count:" + str(len(set(trainRatingsDF['movieId'].value
s.tolist()))))

print("test_movie_count:" + str(len(set(testRatingsDF['movieId'].values.
tolist()))))

print("train_user_count:" + str(len(set(trainRatingsDF['userId'].values.
tolist()))))

print("test_user_count:" + str(len(set(testRatingsDF['userId'].values.to
list()))))
```

Results are as follow:

```
total_movie_count:9066total_user_count:671train_movie_count:8381train_u
ser_count:671test_movie_count:4901test_user_count:671
```

Below, the user-film rating matrix is obtained using pivot_table, and 610*8981 is obtained.

```
trainRatingsPivotDF = pd.pivot_table(trainRatingsDF[['userId', 'movieId',
 'rating']], columns=['movieId'],

                              index=['userId'], values='rating', fill_va
lue=0)
```

Gets the mapping of the movie ID, the user ID, and its index:

```
# enumerateNumbers and values# 8981movies

moviesMap = dict(enumerate(list(trainRatingsPivotDF.columns)))# 610user

usersMap = dict(enumerate(list(trainRatingsPivotDF.index)))# The matrix b
ecomes list and each row becomes a value of list

ratingValues = trainRatingsPivotDF.values.tolist()
```

3. Calculating user similarity

The cosine similarity is used to calculate the similarity between users：

```
def calCosineSimilarity(list1, list2):

    res = 0

    denominator1 = 0

    denominator2 = 0

    for (val1, val2) in zip(list1, list2):

        res += (val1 * val2)

        denominator1 += val1 ** 2

        denominator2 += val2 ** 2

    return res / (math.sqrt(denominator1 * denominator2))
```

Calculate the similarity matrix between users（610*610）：

```
# The similarity between each user is determined by the user's rating of the movie
```

```
userSimMatrix = np.zeros((len(ratingValues), len(ratingValues)), dtype=np.float32)for i in range(len(ratingValues) - 1):

    for j in range(i + 1, len(ratingValues)):

        userSimMatrix[i, j] = calCosineSimilarity(ratingValues[i], ratingValues[j])

        userSimMatrix[j, i] = userSimMatrix[i, j]
```

Next, we will find the K users who are most similar to each user, and use the preferences of these K users to recommend items to the target users, where K=10. The following code is used to calculate the 10 users who are most similar to each user:

```
# Find the top K users that are most similar to each user

userMostSimDict = dict()for i in range(len(ratingValues)):

    userMostSimDict[i] = sorted(enumerate(list(userSimMatrix[i])), key=lambda x: x[1], reverse=True)[:10]
```

4. recommender movie

After getting the 10 users with the most similar interests for each user, we calculated the user's interest in each movie that he or she had not seen according to the following formula:

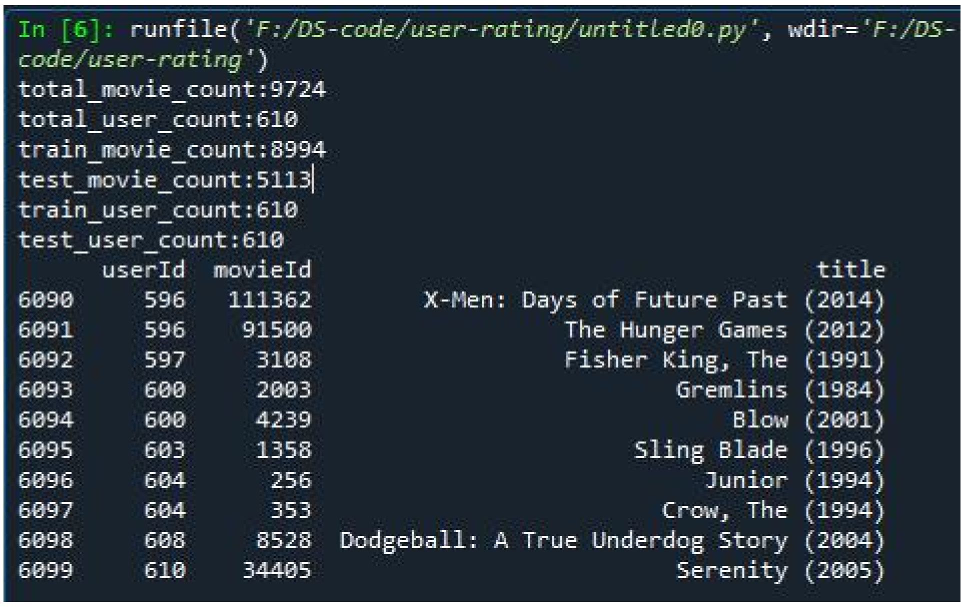
$$p(u,i) = \sum_{v \in S(u,K) \cap N(i)} w_{uv} r_{vi}$$

5. Code

```
#-*- coding: utf-8 -*-
import math
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

moviesPath = ".\\movies.csv"
ratingsPath = ".\\ratings.csv"
moviesDF = pd.read_csv(moviesPath, index_col=None)
ratingsDF = pd.read_csv(ratingsPath, index_col=None)



trainRatingsDF, testRatingsDF = train_test_split(ratingsDF, test_size=0.2)
print("total_movie_count:" + str(len(set(ratingsDF['movieId'].values.tolist()))))
print("total_user_count:" + str(len(set(ratingsDF['userId'].values.tolist()))))
print("train_movie_count:" + str(len(set(trainRatingsDF['movieId'].values.tolist()))))
print("test_movie_count:" + str(len(set(testRatingsDF['movieId'].values.tolist()))))
print("train_user_count:" + str(len(set(trainRatingsDF['userId'].values.tolist()))))
print("test_user_count:" + str(len(set(testRatingsDF['userId'].values.tolist()))))

trainRatingsPivotDF = pd.pivot_table(trainRatingsDF[['userId', 'movieId', 'rating']],
columns=['movieId'],
                                          index=['userId'], values='rating',
fill_value=0)


# enumerate numbers and values
# 8981 movies
moviesMap = dict(enumerate(list(trainRatingsPivotDF.columns)))
# 610 users
usersMap = dict(enumerate(list(trainRatingsPivotDF.index)))
# The matrix becomes list and each row becomes a value of list
ratingValues = trainRatingsPivotDF.values.tolist()


def calCosineSimilarity(list1, list2):
    res = 0
    denominator1 = 0
    denominator2 = 0
    for (val1, val2) in zip(list1, list2):
        res += (val1 * val2)
        denominator1 += val1 ** 2
        denominator2 += val2 ** 2
    return res / (math.sqrt(denominator1 * denominator2))


# The similarity between each user is determined by the user's rating of the movie
```

```
userSimMatrix = np.zeros((len(ratingValues), len(ratingValues)), dtype=np.float32)
for i in range(len(ratingValues) - 1):
    for j in range(i + 1, len(ratingValues)):
        userSimMatrix[i, j] = calCosineSimilarity(ratingValues[i], ratingValues[j])
        userSimMatrix[j, i] = userSimMatrix[i, j]

# Finding the top K users that are most similar to each user
userMostSimDict = dict()
for i in range(len(ratingValues)):
    userMostSimDict[i] = sorted(enumerate(list(userSimMatrix[i])), key=lambda x:
x[1], reverse=True)[:10]


# Use these K preferences to recommend movies that the target audience has not seen
userRecommendValues = np.zeros((len(ratingValues), len(ratingValues[0])),
dtype=np.float32)   # 610*8981

for i in range(len(ratingValues)):
    for j in range(len(ratingValues[i])):
        if ratingValues[i][j] == 0:
            val = 0
            for (user, sim) in userMostSimDict[i]:
                val += (ratingValues[user][j] * sim)
            userRecommendValues[i, j] = val


userRecommendDict = dict()
for i in range(len(ratingValues)):
    userRecommendDict[i] = sorted(enumerate(list(userRecommendValues[i])),
key=lambda x: x[1], reverse=True)[:10]


# Converts the initial index to the original user ID and movie ID
userRecommendList = []
for key, value in userRecommendDict.items():
    user = usersMap[key]
    for (movieId, val) in value:
        userRecommendList.append([user, moviesMap[movieId]])

# Converts the movie ID of the recommended result to the corresponding movie title
recommendDF = pd.DataFrame(userRecommendList, columns=['userId', 'movieId'])
recommendDF = pd.merge(recommendDF, moviesDF[['movieId', 'title']],
on='movieId', how='inner')
print(recommendDF.tail(10))
```

About training time: 10 min



```
In [6]: runfile('F:/DS-code/user-rating/untitled0.py', wdir='F:/DS-
code/user-rating')
total_movie_count:9724
total_user_count:610
train_movie_count:8994
test_movie_count:5113
train_user_count:610
test_user_count:610
      userId  movieId                                    title
6090     596   111362         X-Men: Days of Future Past (2014)
6091     596    91500                   The Hunger Games (2012)
6092     597     3108                   Fisher King, The (1991)
6093     600     2003                           Gremlins (1984)
6094     600     4239                               Blow (2001)
6095     603     1358                        Sling Blade (1996)
6096     604      256                             Junior (1994)
6097     604      353                         Crow, The (1994)
6098     608     8528  Dodgeball: A True Underdog Story (2004)
6099     610    34405                           Serenity (2005)
```