

---

# Data Science Assignment#4

Predict the ratings of movies using collaborative algorithms  
+ Python3

Lee Eunah - 2018년 6월 10일

---

Course name: Data Science (ITE 4005)  
Professor: Sang-Wook Kim  
TAs: Jangwan Koo  
Tae-ri Kim

Student name: Lee Eunah(이 은아)  
Student number: 2016025769  
Major: Computer Science Engineering

---

## Implementation environment

- OS: Mac OS 10.12.6
- Language: Python 3.6.3

## Summary of my algorithm

Collaborative Filtering(CF)is one of the prediction method that find out target user's interests from closed neighbors. The process is roughly divided into three step.

Step 1: Finding neighbors whose preferences are similar to target user.

Step 2: Estimating the user's rating for items that have not been given a score using rating for items given by the neighbors.

Step 3: Recommending a few items with the ratings estimated high

Pearson Correlation Coefficient(PCC) is used as a measure of similarity. It find out a value how two datasets are represented in a straight line.

## Detailed description of my codes

### • Module

- Importing sys module for using command line arguments;

```
from math import sqrt
import sys
```

- Importing math module for using sqrt() function.

### • File I/O

```
def readTrainFile(fileName):
    Dtrain = {}
    trainFile = open("./"+fileName, 'r')
    lines = trainFile.readlines()
    for line in lines:
        (user, movie, rating, time) = line[:-1].split('\t')
        Dtrain.setdefault(user, {})
        Dtrain[user][movie] = float(rating)
    trainFile.close()
    return Dtrain
```

- 
- This function is for reading train file. (u1.base, u2.base, u3.base ...)
  - Datas are stored in nested dictionary type so they are managed each user and movie variables.
  - I think time stamp values are not important in this assignment. So I did not store them.

```
def readTestFile(fileName):
    datas = []
    testFile = open("./"+fileName, 'r')
    lines = testFile.readlines()
    for line in lines:
        data = line[:-1].split('\t')
        datas.append(data)

    testFile.close()
    return datas
```

- This function is for reading test file. (u1.test, u2.test, u3.test ...)
- Split datas are stored in list type.

```
def writeFile(result, title):
    word = ""
    for row in result:
        word += "%s\t%s\t%s\n" % (row[0], row[1], row[2])
    outputFile = open(title, 'w')
    outputFile.write(word)
    outputFile.close()
```

- This function is for writing given format file.
- The given format requires to print user\_id, item\_id(movie) and rating.

---

- Pearson Correlation Coefficient(PCC)

```
# Calculate Pearson Correlation Coefficient(PCC) formula
# to find out similiarity between neighbors and target user
def pearson(Dtrain, user1, user2):
    sumX = 0    # sum of user1's rating
    sumY = 0    # sum of user2's rating
    sumPowX = 0    # sum of square of user1's rating
    sumPowY = 0    # sum of square of user2's rating
    sumXY = 0    # sum of multiplying user1's rating and user2's rating
    cnt = 0    # the number of movie

    for movie in Dtrain[user1]:
        if movie in Dtrain[user2]:    # movies that user1 and user2 watched
            sumX += Dtrain[user1][movie]
            sumY += Dtrain[user2][movie]
            sumPowX += pow(Dtrain[user1][movie], 2)
            sumPowY += pow(Dtrain[user2][movie], 2)
            sumXY += Dtrain[user1][movie] * Dtrain[user2][movie]
            cnt += 1

    if cnt == 0:    # handle division by 0 error
        return 0

    square = sqrt((sumPowX - (pow(sumX, 2) / cnt)) * (sumPowY - (pow(sumY, 2) / cnt)))
    if square == 0:    # handle division by 0 error
        return 0

    pearson_formula = (sumXY - ((sumX * sumY) / cnt)) / square
    return pearson_formula
```

- This function is for calculating Pearson Correlation Coefficient to find out similarity between neighbors and target user.
- This formula is used to determine the linear relationship between two variable. So it can be used to calculate similarity.

```
# Store similarity of neighbors to descending order
def neighborSimilarity(Dtrain, user1):
    sim_neighbor = []
    for user in Dtrain:
        if user == user1 : continue    # exclude myself
        sim_neighbor.append((pearson(Dtrain, user1, user), user))    # tuple type

    sim_neighbor.sort()
    sim_neighbor.reverse()    # descending order

# print(sim_neighbor)
return sim_neighbor
```

- 
- This function is for storing similarity of neighbors that is calculated by PCC to descending order.

- **Predict user's rating**

```
def predictRating(Dtrain, Dtest, target_user):
    score = 0
    sum_movie = {}
    sum_sim = {}
    target_rate = []
    sim_neighbor = neighborSimilarity(Dtrain, target_user)
    for (sim, neighbor) in sim_neighbor:
        if sim <= 0 : continue
        for movie in Dtrain[neighbor]: # if similarity is positive
            if movie not in Dtrain[target_user]:
                score += sim * Dtrain[neighbor][movie]
                sum_movie.setdefault(movie, 0) # initialize
                sum_movie[movie] += score # sum of movie rating of target user

                sum_sim.setdefault(movie, 0)
                sum_sim[movie] += sim
            score = 0

    for key_movie in sum_movie:
        # target rating = total rating / total similarity
        sum_movie[key_movie] = sum_movie[key_movie] / sum_sim[key_movie]
        target_rate.append((key_movie, sum_movie[key_movie]))

    target_rate.sort()
    return target_rate
```

- This function is for predicting user's rating.
- If neighbor's similarity is not positive, it needs to exclude. Because negative value of similarity causes inaccuracy.

---

```
def checkUser(Dtrain, Dtest):
    dup_check = set()
    result = []
    # the movie that user did not watch
    for (target_user, target_movie, target_rating, target_time) in Dtest:
        if target_user not in dup_check:      # check duplication of user
            target_row = predictRating(Dtrain, Dtest, target_user)

            dup_check.add(target_user)
            for (movie, rating) in target_row:
                if movie == target_movie:
                    result.append((target_user, target_movie, rating))

    return result
```

- dup\_check variable checks whether target user is duplicate or not.
- Predicting rating performs when target user does not watch that movie.

#### • Main

```
## main

# Check the command line arguments
if len(sys.argv) != 3:
    print('Please fill in the command form.
    Executable_file minimum_support inputfile outputfile')
    exit()
input_base = sys.argv[1]
input_test = sys.argv[2]












Dtrain = readTrainFile(input_base)
Dtest = readTestFile(input_test)

result = checkUser(Dtrain, Dtest)
title = input_base + '_prediction.txt'
writeFile(result, title)
```

- Main function.
- If length of command line is not three, it occurs error.

---

## Compile my code

 recommender.py  
 u1.base  
 u1.test  
 u2.base  
 u2.test  
 u3.base  
 u3.test  
 u4.base  
 u4.test  
 u5.base  
 u5.test

Put all files in same directory.

```
Eunahui-MacBook-Pro:data leeeunah$ python recommender.py u1.base u1.test
Eunahui-MacBook-Pro:data leeeunah$
```

- Enter the command.

```
Eunahui-MacBook-Pro:Recommender leeeunah$ ls
recommender.py      u2.test             u4.test
u1.base             u3.base             u5.base
u1.base_prediction.txt  u3.base_prediction.txt  u5.base_prediction.txt
u1.test             u3.test             u5.test
u2.base             u4.base
u2.base_prediction.txt  u4.base_prediction.txt
Eunahui-MacBook-Pro:Recommender leeeunah$
```

- Output files are created.

---

## Any other specification about my code

- .setdefault function

It adds key and value in dictionary.

It cannot change the key which already exists.

I wanted to implement nested dictionary. When I input datasets that is same key but different value, the later one is stored. So I found.setdefault function. It helps to store many values in one key.