推荐系统（一）：基于用户的协同过滤算法

#一、协同过滤算法简介

关于协同过滤的一个最经典的例子就是看电影，有时候不知道哪一部电影是我们喜欢的或者评分比较高的，那么通常的做法就是问问周围的朋友，看看最近有什么好的电影推荐。在问的时候，都习惯于问跟自己口味差不 多的朋友，这就是协同过滤的核心思想。

协同过滤是在海量数据中挖掘出小部分与你品味类似的用户，在协同过滤中，这些用户成为邻居，然后根据他们喜欢的东西组织成一个排序的目录推荐给你。所以就有如下两个核心问题

（1）如何确定一个用户是否与你有相似的品味？

（2）如何将邻居们的喜好组织成一个排序目录？

协同过滤算法的出现标志着推荐系统的产生，协同过滤算法包括基于用户和基于物品的协同过滤算法。

#二、协同过滤算法的核心

要实现协同过滤，需要进行如下几个步骤

（1）收集用户偏好

（2）找到相似的用户或者物品

（3）计算并推荐

#三、协同过滤算法的应用方式--基于用户的协同过滤算法

基于用户的协同过滤通过不同用户对物品的评分来评测用户之间的相似性，基于用户的相似性做推荐，简单的讲：给用户推荐和他兴趣相投的其他用户喜欢的物品

算法实现流程分析：

步骤1： 计算用户的相似度

这里采用的是余弦相似度。以下图为例



这里A、B、C、D为用户，a、b、c、d、e为商品。

则用户A和B，用户A和C，用户A和D之间的相似度分别为

得到用户的相似度后，便可以进行下一步了

步骤2：给用户推荐兴趣最相近的k个用户所喜欢的物品

公式如下：

其中，p(u,i)表示用户u对物品i的感兴趣程度，S(u,k)表示和用户u兴趣最接近的K个用户，N(i)表示对物品i有过行为的用户集合，Wuv表示用户u和用户v的兴趣相似度，Rvi表示用户v对物品i的兴趣（这里简化，所有的Rvi都等于1）。

根据UserCF算法，可以算出，用户A对物品c、e的兴趣是：

p(A, c) = wAB + wAD = 0.7416

p(A, e) = wAC + wAD = 0.7416

#四、基于用户的协同过滤算法实现

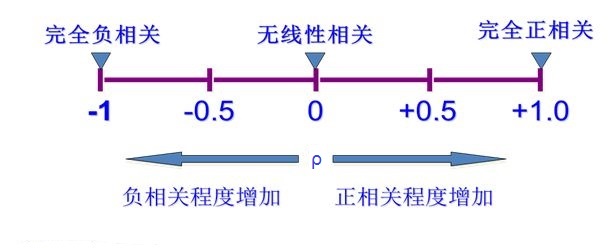
##（一）皮尔逊相关系数

计算用户的相似度，方法有很多。既可以使用余弦相似度，也可以使用皮尔逊相关系数。

皮尔逊相关系数计算公式：



ρ的取值范围和意义：

编程实现公式：

```

sum\_xy = 0

sum\_x = 0

sum\_y = 0

sum\_xx = 0

sum\_yy = 0

n = 0

for (x, y) in some\_condition:

n += 1

sum\_xy += x \* y

sum\_x += x

sum\_y += y

sum\_xx += pow(x, 2)

sum\_yy += pow(y, 2)

if n == 0:

return 0

#皮尔逊相关系数计算公式

denominator = sqrt(sum\_xx - pow(sum\_x, 2) / n) \* sqrt(sum\_yy - pow(sum\_y, 2) / n)

if denominator == 0:

return 0

else:

numerator = sum\_xy - (sum\_x \* sum\_y) / n

return numerator / denominator

```

##（二）数据准备

在实际的应用中，数据量通常很大。这里为了便于理解程序，只取12条数据。

|  |  |  |
| --- | --- | --- |
| username | score | bookid |
| Liu Yi | 3 | 1001 |
| Chen Er | 4 | 1001 |
| Zhang San | 3 | 1001 |
| Li Si | 3 | 1001 |
| Liu Yi | 3 | 1002 |
| Li Si | 4 | 1002 |
| Liu Yi | 4 | 1003 |
| Zhang San | 5 | 1003 |
| Li Si | 5 | 1003 |
| Liu Yi | 4 | 1004 |
| Zhang San | 3 | 1004 |
| Liu Yi | 5 | 1005 |

##（三）完整的算法代码

```

import pdb

import csv

from math import sqrt

rows = []

csvFile = open('user\_book.csv', 'r')

reader = csv.reader(csvFile)

for row in reader:

rows.append(row)

rows.remove(rows[0]) #remove 1st row

print("rows:\n%s\n" % rows)

csvFile.close()

users = {}

for row in rows:

if row[0] not in users:

users[row[0]] = {}

users[row[0]][row[2]] = float(row[1])

print("users:\n%s\n" % users)

class recommender:

#k：the nearest k neighbors

#cnt：recommend count

def \_\_init\_\_(self, dataset, k=3, cnt=2):

self.k = k

self.cnt = cnt

if type(dataset).\_\_name\_\_ == 'dict':

self.dataset = dataset

# pearson correlation coefficient

def pearson(self, touser, dataset):

sum\_xy = 0

sum\_x = 0

sum\_y = 0

sum\_xx = 0

sum\_yy = 0

n = 0

for key in touser:

if key in dataset:

n += 1

x = touser[key]

y = dataset[key]

sum\_x += x

sum\_y += y

sum\_xy += x \* y

sum\_xx += pow(x, 2)

sum\_yy += pow(y, 2)

if n == 0:

return 0

denominator = sqrt(sum\_xx - pow(sum\_x, 2) / n) \* sqrt(sum\_yy - pow(sum\_y, 2) / n)

if denominator == 0:

return 0

else:

numerator = sum\_xy - (sum\_x \* sum\_y) / n

return numerator / denominator

def neighbors(self, username):

distances = []

for key in self.dataset:

if key != username:

distance = self.pearson(self.dataset[username],self.dataset[key])

distances.append((key, distance))

distances.sort(key=lambda artistTuple: artistTuple[1],reverse=True)

return distances

def recommend\_to\_user(self, user):

# store recommended bookid and weight

recommendations = {}

neighborlist = self.neighbors(user)

user\_dict = self.dataset[user]

totalDistance = 0.0

# total distance of the nearest k neighbors

for i in range(self.k):

totalDistance += neighborlist[i][1]

if totalDistance==0.0:

totalDistance=1.0

#recommend books to to\_user who never read

for i in range(self.k):

weight = neighborlist[i][1] / totalDistance

neighbor\_name = neighborlist[i][0]

#book and score of user i

neighbor\_books = self.dataset[neighbor\_name]

for bookid in neighbor\_books:

if not bookid in user\_dict:

if bookid not in recommendations:

recommendations[bookid] = neighbor\_books[bookid] \* weight

else:

recommendations[bookid] += neighbor\_books[bookid] \* weight

# convert dict to list

print("recomend bookid and score weight:\n%s\n" % recommendations)

recommendations = list(recommendations.items())

# sort descending

recommendations.sort(key=lambda artistTuple: artistTuple[1], reverse = True)

return recommendations[:self.cnt]

def recommend\_bookid\_to\_user(username):

bookid\_list = []

r = recommender(users)

bookid\_and\_weight\_list = r.recommend\_to\_user(username)

print ("Recommend bookid and weight:",bookid\_and\_weight\_list)

for i in range(len(bookid\_and\_weight\_list)):

bookid\_list.append(bookid\_and\_weight\_list[i][0])

print ("Recommended bookid: ", bookid\_list)

if \_\_name\_\_ == '\_\_main\_\_':

recommend\_bookid\_to\_user("Li Si")

```

##（四）运行结果

```

Recommended bookid: [‘1004’, ‘1005’]

```

##（五）程序的关键步骤分析

1 读取user\_book.csv中的数据，放到list中

```

[

[‘Liu Yi’, ‘3’, ‘1001’]

[‘Chen Er’, ‘4’, ‘1001’]

[‘Zhang San’, ‘3, ‘1001’]

[‘Li Si’, ‘3’, ‘1001’]

[‘Liu Yi’, ‘3’, ‘1002’]

[‘Li Si’, ‘4’, ‘1002’]

[‘Liu Yi’, ‘4’, ‘1003’]

[‘Zhang San’, ‘5, ‘1003’]

[‘Li Si’, ‘5’, ‘1003’]

[‘Liu Yi’, ‘4’, ‘1004’]

[‘Zhang San’, ‘3’, ‘1004’]

[‘Liu Yi’, ‘5’, ‘1005’]]

]

```

2 把list中的数据，转换成dict形式

```

注意，外面的dict是以用户为key，以bookid和评分构成的字典为value；里面的字典是以bookid为key，以评分为value

{

‘Liu Yi’ : {‘1001’:3.0, ‘1002’:3.0, ‘1003’:4.0, ‘1004’:4.0 ,‘1005’ 5.0},

‘Chen Er’ : {‘1001’ : 4.0},

‘Zhang San’ : {‘1001’ : 3.0, ‘1003’ : 5.0, ‘1004’ : 3.0},

‘Li Si’ : {‘1001’ : 3.0, ‘1002’ : 4.0, ‘1003’ : 5.0}

}

```

3 计算皮尔逊相关系数（distance）

计算Li Si与Liu Yi的距离：

```

sum\_x = 12,

sum\_y = 10,

sum\_xx = 50,

sum\_yy = 34,

sum\_xy = 3\*3 + 4\*3 + 5\*4 = 41

denominator = sqrt[(50 - 144/3)(34 - 100/3)] = sqrt(12/9)

numerator = 41 - (12 \* 10)/3 = 1

distance = 0.866

```

计算Li Si与Chen Er的距离：

```

sum\_x = 3

sum\_y = 4

sum\_xx = 9

sum\_yy = 116

sum\_xy =12

denominator = sqrt[(9 - 9/1)(16 - 15/1)] = 0

distance = 0

```

计算Li Si与Zhang San的距离：

```

sum\_x = 3 + 5 = 8

sum\_y = 3 + 5 = 8

sum\_xx = 9 + 25 = 34

sum\_yy = 9 + 25 = 34

sum\_xy = 9 + 25 = 34

denominator = sqrt[(34 - 64/2)（34 - 64/2）] = 2

numerator = 34 - 64/2 = 2

distance = 1

```

按距离大小排序，得到的distances:

```

[(‘Zhang San’, 1),

(‘Liu Yi’, 0.866),

(‘Chen Er’, 0)

]

```

4 根据distance计算权重

|  |  |
| --- | --- |
| Name | Weight |
| Zhang San | 1 / (1 + 0.866) = 0.536 |
| Liu Yi | 0.866 / (1 + 0.866) = 0.464 |
| Chen Er | 0 / (1 + 0.866) = 0 |

5 根据权重和评分计算推荐值（recommendation）

|  |  |  |
| --- | --- | --- |
| Name | BookId | recommendation |
| Zhang San | 1004 | 3 \* 0.536 = 1.608 |
| Liu Yi | 1004 | 1.608 + 4 \* 0.464 = 3.464 |
| Liu Yi | 1005 | 5 \* 0.464 = 2.32 |
| Cher Er | - | - |

这里可以看到，编号为1004这本书的推荐指数为3.464，编号为1005的这本书的推荐指数为2.32。

6 最后，推荐前三个近邻目标，也就是给目标用户推荐三本书。因为这里总共只有两本书，所以最终只能推荐两本书：

Recommended bookid: [‘1004’, ‘1005’]

#五、源码下载地址

[源码下载](https://github.com/zhenghaishu/MachineLearning/tree/master/Recommend)