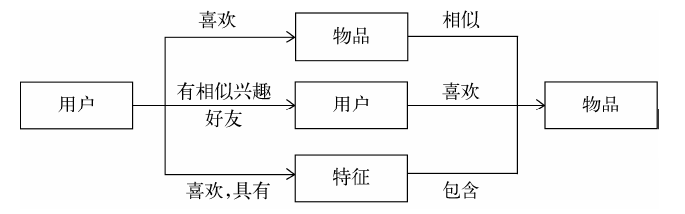
#一、推荐系统的主要方法

推荐系统的目的是联系用户的兴趣和物品，这种联系需要依赖不同的媒介。目前流行的推荐系统基本上通过以下3种方式联系用户兴趣和物品：

（1）利用用户喜欢过的物品，给用户推荐与他喜欢过的物品相似的物品。

（2）利用和用户兴趣相似的其他用户，给用户推荐那些和他们兴趣爱好相似的其它用户喜欢的物品。

（3）通过一些特征联系用户和物品，给用户推荐那些具有用户喜欢的特征的物品。这里的特征有不同的表现方式，可以是物品的属性集合，也可以是隐语义向量。



#二、测试数据

为了便于理解，这里只很了很少的数据量。tagdata.txt中的数据为：

```

刘一 黄飞鸿 movie

刘一 风清扬 music

陈二 黄飞鸿 movie

陈二 黄昏 music

张三 黄飞鸿 movie

张三 新龙门客栈 movie

张三 爱就一个字 music

李四 新龙门客栈 movie

李四 爱就一个字 music

李四 古拉格群岛 book

王五 爱就一个字 music

```

#三、对应关系表

从测试数据，可以计算出：

##（一）用户-物品对应关系

|  |  |  |
| --- | --- | --- |
| 用户 | 物品 | 物品数量 |
| 刘一 | 黄飞鸿 | 1 |
| 风清扬 | 1 |
| 陈二 | 黄飞鸿 | 1 |
| 黄昏 | 1 |
| 张三 | 黄飞鸿 | 1 |
| 新龙门客栈 | 1 |
| 爱就一个字 | 1 |
| 李四 | 新龙门客栈 | 1 |
| 爱就一个字 | 1 |
| 古拉格群岛 | 1 |
| 王五 | 爱就一个字 | 1 |

##（二）用户-标签对应关系

|  |  |  |
| --- | --- | --- |
| 用户 | 标签 | 标签数量 |
| 刘一 | movie | 1 |
| 刘一 | music | 1 |
| 陈二 | movie | 1 |
| 陈二 | music | 1 |
| 张三 | movie | 2 |
| 张三 | music | 1 |
| 李四 | movie | 1 |
| 李四 | music | 1 |
| 李四 | book | 1 |
| 王五 | music | 1 |

##（三）物品-标签对应关系

|  |  |  |
| --- | --- | --- |
| 物品 | 标签 | 标签数量 |
| 黄飞鸿 | movie | 3 |
| 风清扬 | music | 1 |
| 黄昏 | music | 1 |
| 新龙门客栈 | movie | 2 |
| 爱就一个字 | music | 3 |
| 古拉格群岛 | book | 1 |

##（四）标签-物品对应关系表

|  |  |  |
| --- | --- | --- |
| 标签 | 物品 | 物品数量 |
| movie | 黄飞鸿 | 3 |
| movie | 新龙门客栈 | 2 |
| music | 风清扬 | 1 |
| music | 黄昏 | 1 |
| music | 爱就一个字 | 3 |
| book | 古拉格群岛 | 1 |

#四、计算推荐列表

##（一）推荐列表

以用户“刘一”为例

“刘一”打过标签的物品有“黄飞鸿”和“风清扬”，对应的标签为“movie”和“music”

在“标签-物品”对应关系表中，

moive标签对应的物品有“黄飞鸿”和“新龙门客栈”。因为物品“黄飞鸿”“刘一”已经打过标签了，说明“刘一”已经看过了这部电影。不能再向“刘一”推荐这部电影。但是“刘一”没看过“新龙门客栈”，所以要向“刘一”推荐“新龙门客栈”。

music标签对应的物品有“风清扬”、“黄昏”和“爱就一个字”。“风清扬”已经被“刘一”打过标签了，说明已经听过了，不能再推荐了。“黄昏”和“爱就一个字”没被“刘一”打过标签，可以推荐。

##（二）排序

还可以对推荐的物品，根据物品/标签数量进行排序。

从“物品-标签”对应关系表或者“标签-物品”对应关系表可以看出，给“刘一”推荐的“新龙门客栈”、“黄昏”和“爱就一个字”，其物品/标签数量分别为2、1、3。所以排序后的推荐列表为{“爱就一个字”，“新龙门客栈”，“黄昏”}。

##（三）代码

```

def Recommend(user):

recommend\_list = dict()

tagged\_item = user\_items[user]

for tag, wu in user\_tags[user].items():

for item, wi in tag\_items[tag].items():

if item not in tagged\_item:

if item not in recommend\_list:

recommend\_list[item] = wu \* wi

else:

recommend\_list[item] += wu \* wi

return sorted(recommend\_list.items(), key = lambda a:a[1], reverse = True)

```

#五、计算标签流行度

##（一）标签流行度

标签流行度可以用标签出现的次数来表示。

从“用户-标签”对应关系表中可以计算出，各标签的流行度分别为

movie : 4, music : 5, book : 1

排序后，标签流行度为：

{music : 5, movie : 4, book : 1}

##（二）代码

```

def TagPopularity():

tagFreq = {}

for user in user\_tags.keys():

for tag in user\_tags[user].keys():

if tag not in tagFreq:

tagFreq[tag] = 1

else:

tagFreq[tag] += 1

return sorted(tagFreq.items(), key = lambda a:a[1], reverse = True)

```

#六、计算余弦相似度

##（一）定义和公式

物品之间的相似度可以用物品标签向量的余弦相似度来衡量。

余弦相似度，又称为余弦相似性，是通过计算两个向量的夹角余弦值来评估他们的相似度。

假设向量a、b的坐标分别为(x1,y1)、(x2,y2) 。则：

C:\Users\yilinrun\Pictures\cos1.jpg

设向量 A = (A1,A2,...,An)，B = (B1,B2,...,Bn) 。推广到多维:

C:\Users\yilinrun\Pictures\cos2.jpg

##（二）例子

以下表所示的推荐列表为例

|  |  |  |
| --- | --- | --- |
| 物品 | 标签 | 出现次数 |
| 爱就一个字 | music | 3 |
| 新龙门客栈 | movie | 2 |
| 黄昏 | music | 1 |

例1：求“爱就一个字”和“新龙门客栈”之间的余弦相似度

分析：因为“爱就一个字”和“新龙门客栈”的标签不一样 ，所以余弦相似度为0

例2：求“爱就一个字”和“黄昏”的余弦相似度

分析：因为二者的标签都是“music”，所以

余弦相似度 = (3 \* 1 ) / sqrt( 3 ^ 2 \* 1 ^ 2) = 3 / 3 = 1

##（三）代码

```

def CosineSim(item\_tags,i,j):

pdb.set\_trace()

ret = 0

for tag,w in item\_tags[i].items(): #求物品i,j的标签交集数目

if tag in item\_tags[j]:

ret += w \* item\_tags[j][tag]

if ret == 0:

return 0

ni = 0

nj = 0

for tag, wi in item\_tags[i].items(): #统计 i 的标签数目

ni += wi \* wi

for tag, wj in item\_tags[j].items(): #统计 j 的标签数目

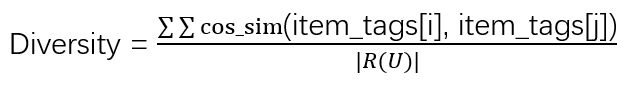
nj += wj \* wj

return ret/math.sqrt(ni \* nj) #返回余弦值

```

#七、计算推荐列表多样性

假充R(u)为给用户u的长度为N的推荐列表。在得到物品之间的相似度度量后，我们可以用如下公式计算一个推荐列表的多样性：



以下表所示的推荐列表为例

|  |  |  |
| --- | --- | --- |
| 物品 | 标签 | 出现次数 |
| 爱就一个字 | music | 3 |
| 新龙门客栈 | movie | 2 |
| 黄昏 | music | 1 |

“爱就一个字”与“新龙门客栈”的余弦相似度为0，与“黄昏”的余弦相似度为1

“新龙门客栈”与“爱就一个字”的余弦相似度为0，与“黄昏”的余弦相似度为0

“黄昏”与“爱就一个字”的余弦相似度为1，与“新龙门客栈”的余弦相似度为0

所以，Diversity = = 1/3

#八、完整的程序

```

#-\*-coding:utf-8-\*-

import pdb

import math

user\_items = dict()

user\_tags = dict()

item\_tags = dict()

tag\_items = dict()

def addValueToMat(mat, key, value):

if key not in mat:

mat[key] = dict()

mat[key][value] = 1

else:

if value not in mat[key]:

mat[key][value] = 1

else:

mat[key][value] += 1

def InitStat():

data\_file = open('tagdata.txt')

line = data\_file.readline()

while line:

terms = line.split("\t")

user = terms[0]

item = terms[1]

tag = terms[2]

addValueToMat(user\_items, user, item)

addValueToMat(user\_tags, user, tag)

addValueToMat(item\_tags, item, tag)

addValueToMat(tag\_items, tag, item)

line = data\_file.readline()

data\_file.close()

# 计算推荐列表

def Recommend(user):

recommend\_list = dict()

tagged\_item = user\_items[user]

for tag, wu in user\_tags[user].items():

for item, wi in tag\_items[tag].items():

if item not in tagged\_item:

if item not in recommend\_list:

recommend\_list[item] = wu \* wi

else:

recommend\_list[item] += wu \* wi

return sorted(recommend\_list.items(), key = lambda a:a[1], reverse = True)

# 统计标签流行度

def TagPopularity():

tagFreq = {}

for user in user\_tags.keys():

for tag in user\_tags[user].keys():

if tag not in tagFreq:

tagFreq[tag] = 1

else:

tagFreq[tag] += 1

return sorted(tagFreq.items(), key = lambda a:a[1], reverse = True)

#计算余弦相似度

def CosineSim(item\_tags,i,j):

ret = 0

for tag,w in item\_tags[i].items(): #求物品i,j的标签交集数目

if tag in item\_tags[j]:

ret += w \* item\_tags[j][tag]

if ret == 0:

return 0

ni = 0

nj = 0

for tag, wi in item\_tags[i].items(): #统计 i 的标签数目

ni += wi \* wi

for tag, wj in item\_tags[j].items(): #统计 j 的标签数目

nj += wj \* wj

return ret/math.sqrt(ni \* nj) #返回余弦值

def Diversity(item\_tags, recommend\_items):

ret = 0

n = 0

for i in dict(recommend\_items).keys():

for j in dict(recommend\_items).keys():

if i == j:

continue

ret += CosineSim(item\_tags, i ,j)

n += 1

return ret / n

InitStat()

recommend\_list = Recommend("刘一")

print("推荐列表： %s" % recommend\_list)

tagFreq = TagPopularity()

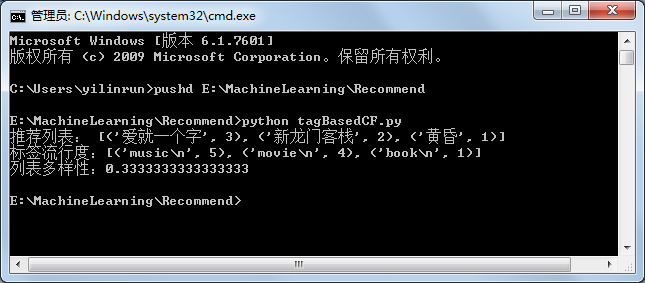
print("标签流行度：%s" % tagFreq)

diversity = Diversity(item\_tags, recommend\_list)

print("列表多样性：%s" % diversity)

```

运行结果



#九、Github源码下载

[基于标签的推荐源代码] (https://github.com/zhenghaishu/MachineLearning/tree/master/Recommend)

#十、参考资料

<http://blog.csdn.net/gamer_gyt/article/details/51684716>