推荐系统实验

基本协同算法

基于物品的协同算法

基于物品的协同算法首先计算物品之间的相似度，计算相似度的方法有以下几种。

1.基于共同喜欢物品的用户列表计算

例如”经常一起购买“

计算公式是

$$W_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)| * |N(j)|}}$$

分母中 $N(i)$ 是购买物品i 的用户数，$N(j)$ 是购买物品j 的用户数，而分子 $N(i) \cap N(j)$ 是同时购买物品i 和物品j 的用户数。可见上述的公式核心是计算同时购买这两种物品的人数比例。当同时购买这两个物品的人数越多，他们的相似度也就越高。另外值得注意的是，在分母中我们用了物品总购买人数做惩罚，也就是说在某个物品可能很热门，导致它经常会被和其他物品一起购买，所以除以它的总购买人数，来降低它和其他物品的相似分数。

构造一个$N \times N$ 的矩阵 C ，存储物品两两同时被购买的次数。遍历每个用户的购买历史，当i 和j 两个物品同时被购买时，则在矩阵C 中$(i\ ,\ j)$ 的位置上加1。当遍历完成时，则可得到共现次数矩阵C 。

其中，$C[i][j]$ 记录了同时喜欢物品i和物品j的用户数，这样我们就可以得到物品之间的相似度矩阵W。举个例子，$(i1,i2)$，（$i2,i4$）这两个相似物品分别被2个用户同时购买过，即共线次数为2。

```
##ItemCF算法
import math
def ItemSimilarity(train):
    C = dict()    ##同时被购买的次数
    N = dict()    ##购买用户数
    for u,items in train.items():
        for i in items.keys():
            if i not in N.keys():
                N[i] = 0
            N[i] += 1
            for j in items.keys():
                if i == j:
                    continue
                if i not in C.keys():
                    C[i] = dict()
                if j not in C[i].keys():
                    C[i][j] = 0
                ##当用户同时购买了i和j，则加1
                C[i][j] += 1
    W = dict()    ##相似分数
    for i, related_items in C.items():
        if i not in W.keys():
            W[i] = dict()
        for j,cij in related_items.items():
            W[i][j] = cij / math.sqrt(N[i]*N[j])
    return W

if __name__ == '__main__':
    Train_Data = {
        'A':{
            '苹果':1,'香蕉':1,'西瓜':1
        },
        'B':{'苹果':1,'西瓜':1},
        'C':{'苹果':1,'香蕉':1,'菠萝':1},
        'D':{'香蕉':1,'葡萄':1},
        'E':{'葡萄':1,'菠萝':1},
```

```
        'F':{'香蕉':1,'西瓜':1}
    }
    W = ItemSimilarity(Train_Data)
    print(W)

##{'苹果': {'香蕉': 0.5773502691896258, '西瓜':
0.6666666666666666, '菠萝': 0.4082482904638631}, '香蕉':
{'苹果': 0.5773502691896258, '西瓜': 0.5773502691896258,
'菠萝': 0.35355339059327373, '葡萄': 0.35355339059327373},
'西瓜': {'苹果': 0.6666666666666666, '香蕉':
0.5773502691896258}, '菠萝': {'苹果': 0.4082482904638631,
'香蕉': 0.35355339059327373, '葡萄': 0.5}, '葡萄': {'香蕉':
0.35355339059327373, '菠萝': 0.5}}
```

将运行结果填充进表格得到以下相似度矩阵：

	苹果	香蕉	葡萄	西瓜	菠萝
苹果		0.5773502692		0.6666666667	0.4082482905
香蕉	0.5773502692		0.3535533906	0.5773502692	0.3535533906
葡萄		0.3535533906			0.5
西瓜	0.6666666667	0.5773502692			
菠萝	0.4082482905	0.3535533906	0.5		

2. 引入评分：基于余弦(Cosine-based)的相似度计算

上面的方法计算物品相似度是直接使用同时购买这两个物品的人数。但是也有可能存在用户购买了但是不喜欢的情况。所以如果数据集包括了具体的评分数据，我们可以进一步把用户评分引入到相似度计算中。可利用上节提到的余弦公式计算任意两个物品的相似度，公式如下：

$$w_{ij} = cos\theta = \frac{N_i \cdot N_j}{||N_i||||N_j||} = \frac{\sum_{k=1}^{len}(n_{ki} \times n_{kj})}{\sqrt{\sum_{k=1}^{len} n_{ki}^2} \times \sqrt{\sum_{k=1}^{len} n_{kj}^2}}$$

其中n_{ki} 是用户k对物品i 的评分，如果没有评分则为0。实现的代码如下：

```
## ItemCF-余弦算法
import math
def ItemSimilarity_cos(train):
    C = dict()    ##同时购买的次数
    N = dict()    ##购买用户数
```

```
    for u,items in train.items():
        for i in items.keys():
            if i not in N.keys():
                N[i]=0
            N[i] += items[i]* items[i]
            for j in items.keys():
                if i == j:
                    continue
                if i not in C.keys():
                    C[i]=dict()
                if j not in C[i].keys():
                    C[i][j]=0
                ##当用户同时购买了i和j，则加评分乘积
                C[i][j] += items[i]*items[j]
    W = dict()  ##相似分数
    for i,related_items in C.items():
        if i not in W.keys():
            W[i]=dict()
        for j,cij in related_items.items():
            W[i][j] = cij / (math.sqrt( N[i])
*math.sqrt( N[j]) )
    return W

if __name__ == '__main__':
    Train_Data = {
        'A':{
            '苹果':2,'香蕉':2,'西瓜':2
        },
        'B':{'苹果':2,'西瓜':2},
        'C':{'苹果':2,'香蕉':2,'菠萝':2},
        'D':{'香蕉':2,'葡萄':2},
        'E':{'葡萄':2,'菠萝':2},
        'F':{'香蕉':2,'西瓜':2}
    }
    W= ItemSimilarity_cos (Train_Data)
    print(W)
```

```
##{'苹果': {'香蕉': 0.5773502691896258, '西瓜':
0.6666666666666667, '菠萝': 0.40824829046386296}, '香蕉':
{'苹果': 0.5773502691896258, '西瓜': 0.5773502691896258,
'菠萝': 0.35355339059327373, '葡萄': 0.35355339059327373},
'西瓜': {'苹果': 0.6666666666666667, '香蕉':
0.5773502691896258}, '菠萝': {'苹果': 0.40824829046386296,
'香蕉': 0.35355339059327373, '葡萄': 0.4999999999999999},
'葡萄': {'香蕉': 0.35355339059327373, '菠萝':
0.4999999999999999}}
```

运行以上代码，可以得到物品间的相似度矩阵，当评分数据都为1的时候，该方法与基于共同喜欢物品的用户列表计算结果一致。

3. 热门物品的惩罚

从相似度计算公式中，我们可以发现当物品i被更多的人购买时，分子中的$N(i) \cap N(j)$和分母中的$N(i)$都会增长。对于热门物品，分子$N(i) \cap N(j)$的增长速度往往高于$N(i)$，这就会使得物品i和很多其他物品相似度都偏高，这就是ItemCF中的物品热门问题。推荐结果过于热门，会使得个性化感知降低。以歌曲的相似为例，大部分用户都会收藏《小苹果》这些热门歌曲，从而导致《小苹果》出现在很多的相似歌曲中。为了解决这个问题，我们对热门物品i进行惩罚，例如下式，当$\alpha \in (0, 0.5)$时，$N(i)$越小，惩罚的越厉害，从而会使热门物品相关性分数下降。

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|^{\alpha} * |N(j)|^{1-\alpha}}$$

计算代码参考如下：

```
##改进算法
import math
def ItemSimilarity_alpha(train,alpha=0.3):
    C = dict()    ##同时被购买的次数
    N = dict()    ##被购买用户数
    for u,items in train.items():
        for i in items.keys():
            if i not in N.keys():
                N[i]=0
            N[i] += 1
            for j in items.keys():
                if i == j:
                    continue
                if i not in C.keys():
```

```
                    C[i]=dict()
                if j not in C[i].keys():
                    C[i][j]=0
                ##当用户同时购买了i和j，则加1
                C[i][j] += 1
    W = dict()  ##相似分数
    for i,related_items in C.items():
        if i not in W.keys():
            W[i]=dict()
        for j,cij in related_items.items():
            W[i][j] = cij /
(math.pow(N[i],alpha)*math.pow(N[j],1-alpha) )
    return W

if __name__ == '__main__':
    Train_Data = {
        'A':{
            '苹果':1,'香蕉':1,'西瓜':1
        },
        'B':{'苹果':1,'西瓜':1},
        'C':{'苹果':1,'香蕉':1,'菠萝':1},
        'D':{'香蕉':1,'葡萄':1},
        'E':{'葡萄':1,'菠萝':1},
        'F':{'香蕉':1,'西瓜':1}
    }
    W= ItemSimilarity_alpha (Train_Data)
```

运行上述代码，可以得到物品间的相似度矩阵，可以观察到香蕉因为比较热门，被降权惩罚，与其他物品的相似分数显著降低。

	苹果	香蕉	葡萄	西瓜	菠萝
苹果		0.5450691788		0.6666666667	0.4427337466
香蕉	0.6115431698		0.4061261982	0.6115431698	0.4061261982
葡萄		0.3077861033			0.5
西瓜	0.6666666667	0.5450691788			
菠萝	0.3764489785	0.3077861033	0.5		

在得到物品之间相似度后，进入第二步。按如下公式计算用户u对一个物品i的预测分数：

$$P_{ui} = \sum_{N(u) \cap S(j,k)} w_{ji} score_{ui}$$

其中$S(j,k)$是 物品j相似物品的集合，一般来说j的相似物品集合是相似分数最高的k个，参照上面计算得出的相似分数。$score_{ui}$是用户对已购买的物品i的评分，如果没有评分数据，则取1。如果待打分的物品和用户购买过的多个物品相似，则将相似分数相加，相加后的得分越高，则用户购买的可能性越大。我们以书举例，假设用户购买过《明朝那些事儿》（评分0.8）和《品三国》（评分0.6），而《鱼羊野史和》和《明朝那些事儿》相似分是0.2分，《鱼羊野史》和《品三国》的相似分数是0.1分，则用户在《鱼羊野史》上的分数则为0.22分（0.8 * 0.2 +0.6 * 0.1）。这时候找出与用户喜欢物品相似度高的top N个，也就是分数最高的N个作为推荐的候选。

```
import math
def ItemSimilarity_alpha(train,alpha=0.3):
    C = dict()   ##被购买的次数
    N = dict()   ##被购买用户数
    for u,items in train.items():
        for i in items.keys():
            if i not in N.keys():
                N[i]=0
            N[i] += 1
            for j in items.keys():
                if i == j:
                    continue
                if i not in C.keys():
                    C[i]=dict()
                if j not in C[i].keys():
                    C[i][j]=0
                ##当用户同时购买了i和j，则加1
                C[i][j] += 1
    W = dict()  ##相似分数
    for i,related_items in C.items():
        if i not in W.keys():
            W[i]=dict()
        for j,cij in related_items.items():
            W[i][j] = cij /
(math.pow(N[i],alpha)*math.pow(N[j],1-alpha) )
    return W

def Recommend(train,user_id,W,K):
    rank = dict()
    ru = train[user_id]
    for i,pi in ru.items():
        tmp=W[i]
```

```
        for j,wj in sorted(tmp.items(),key=lambda d:
d[1],reverse=True)[0:K]:
            if j not in rank.keys():
                rank[j]=0
            ##r如果用户已经购买过，则不再推荐
            if j in ru:
                continue
            ##待推荐的物品j与用户已购买的物品i相似，则累加上相似
分数
            rank[j] += pi*wj
    return rank

if __name__ == '__main__':
    Train_Data = {
        'A':{
            '苹果':1,'香蕉':1,'西瓜':1
        },
        'B':{'苹果':1,'西瓜':1},
        'C':{'苹果':1,'香蕉':1,'菠萝':1},
        'D':{'香蕉':1,'葡萄':1},
        'E':{'葡萄':1,'菠萝':1},
        'F':{'香蕉':1,'西瓜':1}
    }
    W=ItemSimilarity_alpha(Train_Data)
    print(Recommend(Train_Data,'C',W,3) )
## {'西瓜': 1.278209836428268, '香蕉': 0, '菠萝': 0, '苹
果': 0, '葡萄': 0.5}
```

在上面的计算过程中，我们发现还有一个为定义参数K，即对物品相似物品中top K个物品进行召回，过大的K，会召回很多相关性不强的物品，导致准确率下降；过小的K会使召回的物品过少，使得准确率也不高。一般来说，算法工作人员需要尝试不同的K值对比算法准确率和召回率，以便选择最佳的K值。

基于用户的协同算法

基于用户的的协同过滤(User CF) 的原理其实是和基于物品的协同过滤类似的。所不同的是，基于物品的协同过滤的原理是用户U购买了A物品，推荐给用户U和A相似的物品B，C，D。而基于用户的协同过滤，是先计算用户U和其他用户的相似度，然后取和U最相似的几个用户，把他们购买过的物品推荐给用户U。

为了计算用户相似度，我们首先要把用户购买过物品的索引转化为物品被用户购买过的索引数据，即物品的倒排索引：

```
def defItemIndex(DictUser):
    DictItem=defaultdict(defaultdict)
    ##遍历每个用户
    for key in DictUser:
        ##遍历用户k的购买记录
        for i in DictUser[key]:
            DictItem[i[0]][key]=i[1]
    return DictItem
```

建立好物品的倒排索引后，就可以根据相似度公式计算用户之间的相似度

$$w_{ab} = \frac{|N(a) \cap N(b)|}{\sqrt{|N(a)| * |N(b)|}}$$

其中 $N(a)$ 表示用户 a 购买物品的数量，$N(b)$ 表示用户 b 购买物品的数量，$N(a) \cap N(b)$表示用户 a 和 b 购买相同物品的数量。

```
def defUserSimilarity(DictItem):
    N=dict()   #用户购买的数量
    C=defaultdict(defaultdict)
    W=defaultdict(defaultdict)
    ##遍历每个物品
    for key in DictItem:
        ##遍历用户k购买过的物品
        for i in DictItem[key]:
            #i[0]表示用户的id ，如果未计算过，则初始化为0
            if i[0] not in N.keys():
                N[i[0]]=0
            N[i[0]]+=1
            ## (i,j)是物品k同时被购买的用户两两匹配对

            for j in DictItem[key]:
                if i(0)==j(0):
                    continue
                if j[0] not in C[i[0]].keys():
                    C[i[0]][j[0]]=0
                #C[i[0]][j[0]]表示用户i和j购买同样物品的数量

                C[i[0]][j[0]]+=1
```

```
    for i,related_user in C.items():
        for j,cij in related_user.items():
            W[i][j]=cij/math.sqrt(N[i]*N[j])
    return W
```

有了用户的相似数据，针对用户 U 挑选 K 个最相似的用户，把他们购买过的物品中，U 未购买过的物品推荐给用户 U 即可。如果有评分数据，可以针对这些物品进一步打分，打分原理与基于物品的推荐原理类似，公式如下：

$$p_{ui} = \sum_{N(i) \cap S(u,k)} w_{vu} score_{vu}$$

其中$N(i)$是物品i被购买的用户集合，$S(u,k)$是用户u的相似用户集合，挑选最相似的用户k个，将重合的用户v在物品i上的得分乘以用户u和v的相似度，累加后得到用户u对于物品i的得分。

```
from collections import defaultdict
import math
def defItemIndex(DictUser):
    DictItem=defaultdict(defaultdict)
    ##遍历每个用户
    for key in DictUser:
        ##遍历用户k的购买记录
        for i in DictUser[key]:
            DictItem[i][key]=[key,DictUser[key][i]]
    return DictItem

def defUserSimilarity(DictItem):
    N=dict()   #用户购买的数量
    C=defaultdict(defaultdict)
    W=defaultdict(defaultdict)
    ##遍历每个物品
    for key in DictItem:
        ##遍历用户k购买过的物品
        #print(key,":")
        for x in DictItem[key]:
            i = DictItem[key][x]
            #i[0]表示用户的id ，如果未计算过，则初始化为0
            if i[0] not in N.keys():
                N[i[0]]=0
            N[i[0]]+=1
```

```
            ## (i,j)是物品k同时被购买的用户两两匹配对

            for j in DictItem[key]:
                if i[0]==j[0]:
                    continue
                if j[0] not in C[i[0]].keys():
                    C[i[0]][j[0]]=0
                #C[i[0]][j[0]]表示用户i和j购买同样物品的数量

                C[i[0]][j[0]]+=1
    for i,related_user in C.items():
        for j,cij in related_user.items():
            W[i][j]=cij/math.sqrt(N[i]*N[j])
    return W

if __name__ == '__main__':
    Train_Data = {
        'A':{
            '苹果':1,'香蕉':1,'西瓜':1
        },
        'B':{'苹果':1,'西瓜':1},
        'C':{'苹果':1,'香蕉':1,'菠萝':1},
        'D':{'香蕉':1,'葡萄':1},
        'E':{'葡萄':1,'菠萝':1},
        'F':{'香蕉':1,'西瓜':1}
    }
    W=defItemIndex(Train_Data)
    print(defUserSimilarity(W))

#defaultdict(<class 'collections.defaultdict'>,
#{
#    'A': defaultdict(None,
#        {'B': 0.8164965809277261, 'C':
0.6666666666666666, 'D': 0.4082482904638631, 'F':
#0.8164965809277261}),
#    'B': defaultdict(None,
#        {'A': 0.8164965809277261, 'C':
0.4082482904638631, 'F': 0.5}),
#    'C': defaultdict(None,{'A': 0.6666666666666666,
'B': 0.4082482904638631, 'D': 0.4082482904638631, 'F':
0.4082482904638631, 'E': 0.4082482904638631}),
#    'D': defaultdict(None,{'A': 0.4082482904638631,
'C': 0.4082482904638631, 'F': 0.5, 'E': 0.5}),
```

```
#    'F': defaultdict(None,{'A': 0.8164965809277261,
'C': 0.4082482904638631, 'D': 0.5, 'B': 0.5}),
#    'E': defaultdict(None,{'C': 0.4082482904638631,
'D': 0.5})
#})
```