


# 推荐系统（2）-协同过滤

 **Alan**  
数据分析、挖掘、机器学习

关注他

10 人赞同了该文章

## 一、协同过滤算法

### 1.1协同过滤的推荐概述

协同过滤（简称：CF）推荐算法是最经典、最常用的推荐算法，该算法通过分析用户的兴趣，在用户群找到指定用户的相似用户，综合这些用户的对某一信息评价，形成系统对该指定用户针对此信息的喜好程度的预测。比如，你想看电影，但不知道选哪一部？你就会得到周围相似爱好朋友

 赞同 10

 2 条评论

 分享

 喜欢

 收藏

 申请转载



**显性反馈：**用户明确表示对物品喜好的行为。这要方式是评分和喜欢/不喜欢。

**隐形反馈：**不能明确反应用户喜好的行为。（购买日志、阅读日志、浏览日志）

实现协同过滤，需要几个步骤：

1. 收集用户偏好；（评分、投票、转发、保存书签、购买、点击、页面停留时间等）
2. 找到相似的用户或物品；
3. 计算并排序

例如，协同过滤的模型一般为 $m$ 个物品， $m$ 个用户的数据，只有部分用户和部分数据之间是有评分数据的，其它部分评分是空白，此时我们要用已有的部分稀疏数据来预测那些空白的物品和数据之间的评分关系，找到最高评分的物品推荐给用户。

## 1.2 用户评分

如何组合用户评分的行为？

- **将不同的行为分组：**一般分为查看和购买，然后基于不同的用户行为，计算不同用户或者物品的相似度（CF）；
- **对不同行为进行加权：**对不同行为产生的用户喜好程度进行加权，然后求出用户对物品的总体喜好程度（LFM），减噪、归一化；

## 1.3 相似度计算

关于相似度的计算，以下几种方法都是基于向量（Vector），距离越近相似度的值越大。

- 欧氏距离（常用）
- 曼哈顿距离
- 切比雪夫距离
- 马氏距离
- 夹角余弦距离（常用）
- 杰卡德相似系数与杰拉德距离
- 相关系数与相关距离

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

```
from numpy import *

# 欧式距离
def EuclideanDistance(a,b):
    return sqrt( (a[0]-b[0])**2 + (a[1]-b[1])**2 )

print('a,b 二维欧式距离为: ',EuclideanDistance((1,1),(2,2)))

def ManhattanDistance(a,b):
    return abs(a[0]-b[0])+abs(a[1]-b[1])

print('a,b 二维曼哈顿距离为: ', ManhattanDistance((1,1),(2,2)))

def ChebyshevDistance(a,b):
    return max( abs(a[0]-b[0]), abs(a[1]-b[1]))

print('a,b二维切比雪夫距离: ', ChebyshevDistance((1,2),(3,4)))

def CosineSimilarity(a,b):
    cos = (a[0]*b[0]+a[1]*b[1]) / (sqrt(a[0]**2 + a[1]**2) * sqrt(b[0]**2 + b[1]**2) )

    return cos
print('a,b 二维夹角余弦距离: ',CosineSimilarity((1,1),(2,2)))

def JaccardSimilarityCoefficient(a,b):
    set_a = set(a)
    set_b = set(b)
    dis = float(len(set_a & set_b) )/ len(set_a | set_b)
    return dis

print('a,b 杰卡德相似系数: ', JaccardSimilarityCoefficient((1,2,3),(2,3,4)))

def JaccardSimilarityDistance(a,b):
    set_a = set(a)
    set_b = set(b)
    dis = float(len( (set_a | set_b) - (set_a & set_b) ) )/ len(set_a | set_b)
    return dis

print('a,b 杰卡德距离: ', JaccardSimilarityDistance((1,2,3),(2,3,4)))
```

## 2.1 UserCF算法原理

UserCF算法核心主要分两步：

1. 找到与目标用户A兴趣相似的用户；
2. 得到用户A相似用户评分物品中用户A没有进行评分或是浏览购买过的Item，并推荐给用户A；

**第一步：计算两个用户的兴趣相似度。**这里，协同过滤算法主要利用行为的相似度计算兴趣的相似度。给定用户u和用户v，令N(u)表示用户u曾经有过正反馈的物品集合，令N(v)为用户v曾经有过正反馈的物品集合。那么，我们可以通过如下的Jaccard公式简单地计算u和v的兴趣相似度或者通过余弦公式：

余弦公式：

$$w_{uv} = \frac{\sum_{i \in N(u) \cap N(v)} \frac{1}{\log 1 + |N(i)|}}{\sqrt{|N(u)| |N(v)|}}$$

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

这个一个行为记录 我们可以根据余弦公式计算如下

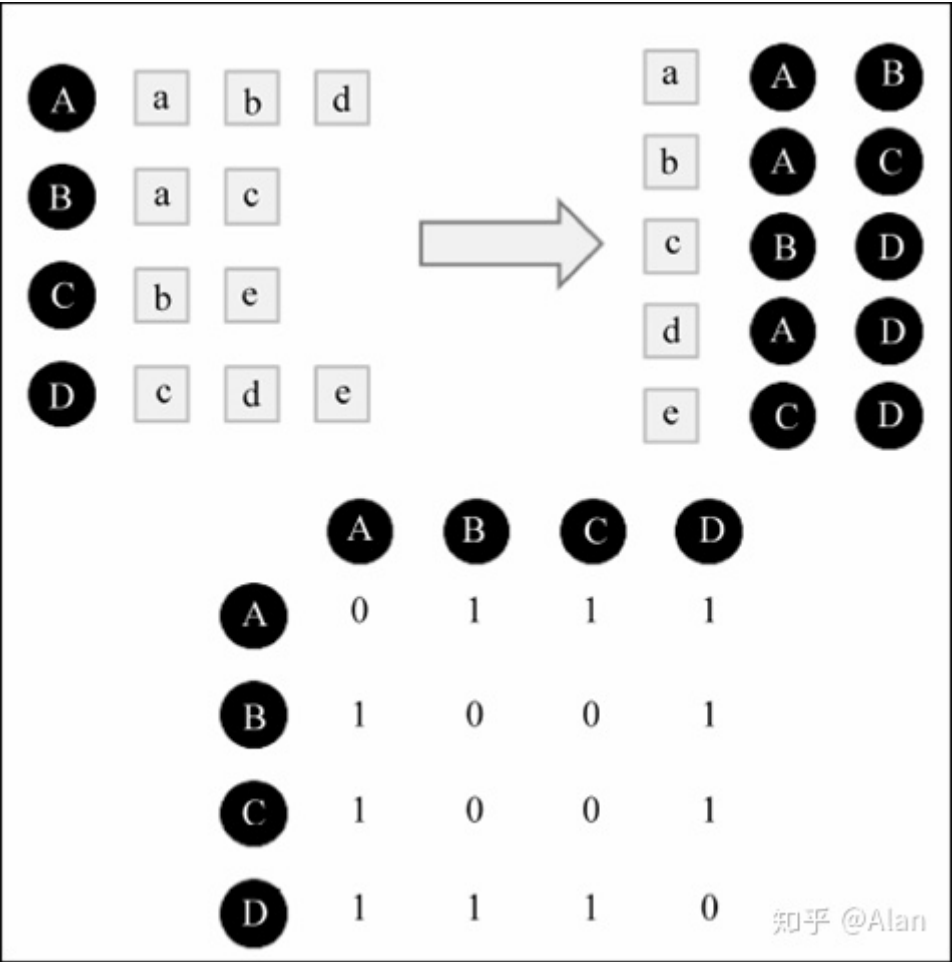
A	a	b	d
B	a	c	
C	b	e	
D	c	d	e

$$w_{AB} = \frac{|\{a,b,d\} \cap \{a,c\}|}{\sqrt{|\{a,b,d\}| \cdot |\{a,c\}|}} = \frac{1}{\sqrt{6}}$$
$$w_{AC} = \frac{|\{a,b,d\} \cap \{b,e\}|}{\sqrt{|\{a,b,d\}| \cdot |\{b,e\}|}} = \frac{1}{\sqrt{6}}$$
$$w_{AD} = \frac{|\{a,b,d\} \cap \{c,d,e\}|}{\sqrt{|\{a,b,d\}| \cdot |\{c,d,e\}|}} = \frac{1}{3}$$

知乎 @Alan

上述算法很简单但是计算量较大，因为需要所有用户之前的复杂度  $n(n-1)/2$ 。下面这种计算用户相似度算法通过空间换时间。

首先建立物品到用户的倒排表，然后统计每两个用户的公共物品数量（如下图所示）。



得到用户之间的兴趣相似度后，UserCF算法会给用户推荐和他兴趣最相似的K个用户喜欢的物品。上面右边公式度量了UserCF算法中用户u对物品i的感兴趣程度：其中， $S(u, K)$ 包含和用户u兴趣最接近的K个用户， $N(i)$ 是对物品i有过行为的用户集合， $W_{uv}$ 是用户u和用户v的兴趣相似度， $R_{vi}$ 代表用户v对物品i的兴趣，因为使用的是单一行为的隐反馈数据，所以所有的 $R_{vi}=1$ 。

上述推荐算法缺陷：

如果两个用户都曾经买过《新华字典》，这丝毫不能说明他们兴趣相似，因为绝大多数中国人小时候都买过《新华字典》。但如果两个用户都买过《数据挖掘导论》，那可以认为他们的兴趣比较相似，因为只有研究数据挖掘的人才会买这本书。换句话说，两个用户对冷门物品采取过同样的行为更能说明他们兴趣的相似度。因此，John S. Breese在论文中提出了如下公式，根据用户行为计算用户的兴趣相似度：

计算用户相似度的改进算法：

$$w_{uv} = \frac{\sum_{i \in N(u) \cap N(v)} \frac{1}{\log 1 + |N(i)|}}{\sqrt{|N(u)| |N(v)|}}$$

知乎 @Alan

分子中的倒数惩罚了用户u和用户v共同兴趣列表中热门物品对他们相似度的影响。 $N(i)$ 是对物品i有过行为的用户集合，越热门， $N(i)$ 越大。

## 2.2案例：基于UserCF算法的电影推荐系统

数据地址：

MovieLens 1M Dataset

[grouplens.org](http://grouplens.org)

```
# -*- coding: utf-8 -*-
```

```
"""
```

Author: Alan

```
import math
import json
import os

class UserCFRec:
    def __init__(self,datafile):
        self.datafile = datafile
        self.data = self.loadData()

        self.trainData,self.testData = self.splitData(3,47) # 训练集与数据集
        self.users_sim = self.UserSimilarityBest()

# 加载评分数据到data
def loadData(self):
    print("加载数据...")
    data=[]
    for line in open(self.datafile):
        userid,itemid,record,_ = line.split("::")
        data.append((userid,itemid,int(record)))
    return data

"""
拆分数数据集为训练集和测试集
k: 参数
seed: 生成随机数的种子
M: 随机数上限
"""
def splitData(self,k,seed,M=8):
    print("训练数据集与测试数据集切分...")
    train,test = {},{}
    random.seed(seed)
    for user,item,record in self.data:
        if random.randint(0,M) == k:
            test.setdefault(user,{})
            test[user][item] = record
        else:
            train.setdefault(user,{})
            train[user][item] = record
    return train,test

# 计算用户之间的相似度，采用惩罚热门商品和优化算法复杂度的算法
```

```

userSim = json.load(open("data/user_sim.json", "r"))
else:
    # 得到每个item被哪些user评价过
    item_users = dict()
    for u, items in self.trainData.items():
        for i in items.keys():
            item_users.setdefault(i, set())
            if self.trainData[u][i] > 0:
                item_users[i].add(u)
    # 构建倒排表
    count = dict()
    user_item_count = dict()
    for i, users in item_users.items():
        for u in users:
            user_item_count.setdefault(u, 0)
            user_item_count[u] += 1
            count.setdefault(u, {})
            for v in users:
                count[u].setdefault(v, 0)
                if u == v:
                    continue
                count[u][v] += 1 / math.log(1+len(users))
    # 构建相似度矩阵
    userSim = dict()
    for u, related_users in count.items():
        userSim.setdefault(u, {})
        for v, cuv in related_users.items():
            if u==v:
                continue
            userSim[u].setdefault(v, 0.0)
            userSim[u][v] = cuv / math.sqrt(user_item_count[u] * user_item_cou
    json.dump(userSim, open('data/user_sim.json', 'w'))
return userSim

"""
为用户user进行物品推荐
user: 为用户user进行推荐
k: 选取k个近邻用户
nitems: 取nitems个物品
"""

def recommend(self, user, k=8, nitems=40):

```



```

        if i in have_score_items:
            continue
        result.setdefault(i, 0)
        result[i] += wuv * rvi
    return dict(sorted(result.items(), key=lambda x: x[1], reverse=True)[0:nitems])

"""
    计算准确率
    k: 近邻用户数
    nitems: 推荐的item个数
"""

def precision(self, k=8, nitems=10):
    print("开始计算准确率 ...")
    hit = 0
    precision = 0
    for user in self.trainData.keys():
        tu = self.testData.get(user, {})
        rank = self.recommend(user, k=k, nitems=nitems)
        for item, rate in rank.items():
            if item in tu:
                hit += 1
        precision += nitems
    return hit / (precision * 1.0)

if __name__ == '__main__':
    cf = UserCFRec("../data/ml-1m/ratings.dat")
    result = cf.recommend("1")
    print("user '1' recommend result is {}".format(result))

    precision = cf.precision()
    print("precision is {}".format(precision))

```

## 三、基于物品的协同过滤 (ItemCF)

### 3.1 ItemCF算法原理

ItemCF算法核心主要分两步：

## 第一步：计算物品之间的相似度；

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

其中， $|N(i)|$ 是喜欢物品*i*的用户数， $|N(j)|$ 是喜欢物品*j*的用户数， $|N(i) \cap N(j)|$ 是同时喜欢物品*i*和物品*j*的用户数。

从上面的定义看出，在协同过滤中两个物品产生相似度是因为它们共同被很多用户喜欢，两个物品相似度越高，说明这两个物品共同被很多人喜欢。

这里面蕴含着一个假设：就是假设每个用户的兴趣都局限在某几个方面，因此如果两个物品属于一个用户的兴趣列表，那么这两个物品可能就属于有限的几个领域，而如果两个物品属于很多用户的兴趣列表，那么它们就可能属于同一个领域，因而有很大的相似度。

(用户活跃度大的用户可能喜欢列表中有很多商品，会影响ItemCf算法计算结果准确性，后面优化算法会增加一个用户活跃度分子)

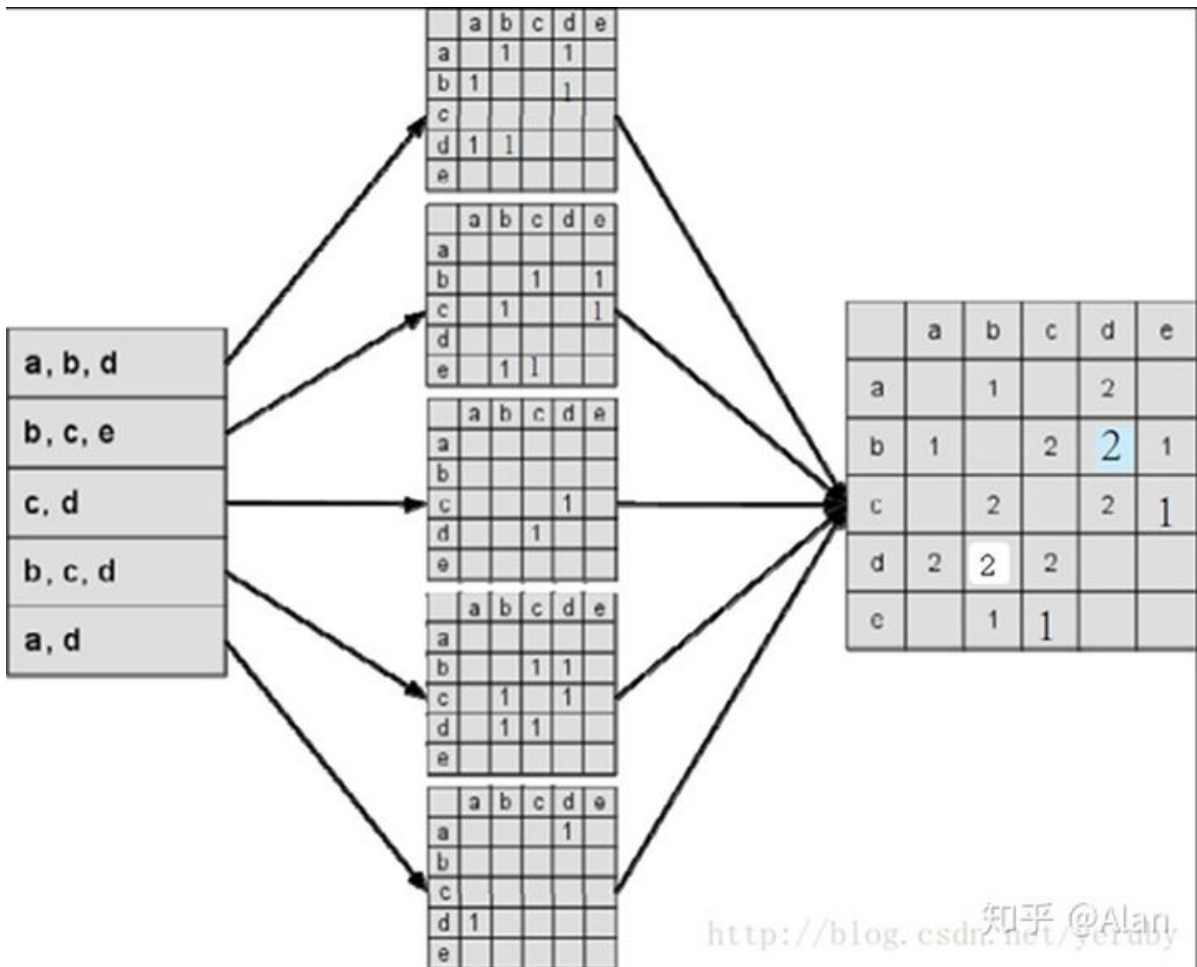
举例，用户A对物品a、b、d有过行为，用户B对物品b、c、e有过行为，等等；

Ⓐ	a	b	d
Ⓑ	b	c	e
Ⓒ	c	d	
Ⓓ	b	c	d
Ⓔ	a	d	

依此构建用户——物品倒排表：物品a被用户A、E有过行为，等等；



建立物品相似度矩阵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。

在得到物品之间的相似度后，进入第二步。

第二步：根据物品的相似度和用户的历史行为给用户生成推荐列表；

$$p_{uj} = \sum_{i \in N(u) \cap S(i, K)} w_{ji} r_{ui}$$

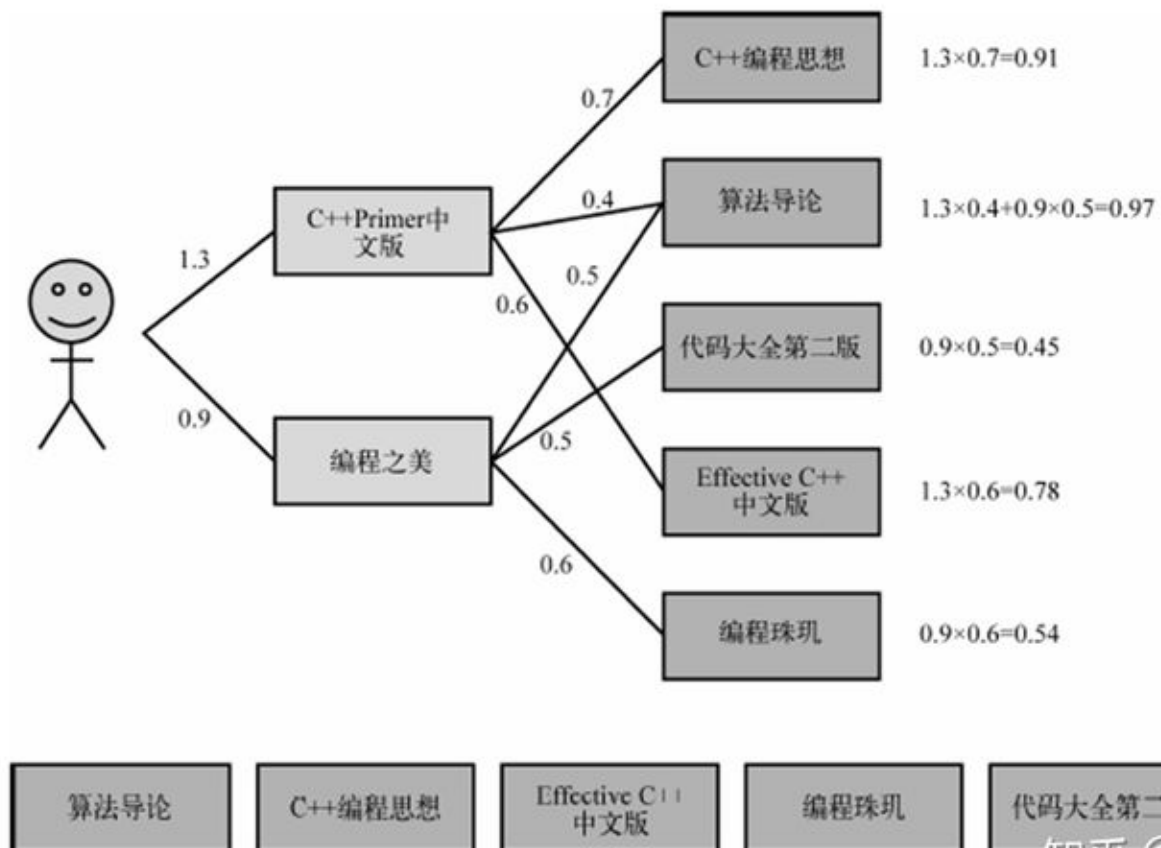
知乎 @Alan

其中， $P_{uj}$ 表示用户 $u$ 对物品 $j$ 的兴趣， $N(u)$ 表示用户喜欢的物品集合（ $i$ 是该用户喜欢的某一个物品）， $S(i, k)$ 表示和物品 $i$ 最相似的 $K$ 个物品集合（ $j$ 是这个集合中的某一个物品）， $w_{ji}$ 表示物品 $j$ 和物品 $i$ 的相似度， $r_{ui}$ 表示用户 $u$ 对物品 $i$ 的兴趣（这里简化 $r_{ui}$ 都等于1）。

该公式的含义是：和用户历史上感兴趣的物品越相似的物品，越有可能在用户的推荐列表中获得比较高的排名。

下面是一个书中的例子，帮助理解ItemCF过程：

图2-12是一个基于物品推荐的简单例子。该例子中，用户喜欢《C++ Primer中文版》和《编程之美》两本书。然后ItemCF会为这两本书分别找出和它们最相似的3本书，然后根据公式的定义计算用户对每本书的感兴趣程度。比如，ItemCF给用户推荐《算法导论》，是因为这本书和《C++ Primer中文版》相似，相似度为0.4，而且这本书也和《编程之美》相似，相似度是0.5。考虑到用户对《C++ Primer中文版》的兴趣度是1.3，对《编程之美》的兴趣度是0.9，那么用户对《算法导论》的兴趣度就是 $1.3 \times 0.4 + 0.9 \times 0.5 = 0.97$ 。



### 3.2案例：编写一个基于ItemCF算法的电影推荐系统

数据地址：

MovieLens 1M Dataset

[grouplens.org](https://grouplens.org)

```
# -*- coding: utf-8 -*-
"""
    Author: Alan
    Desc:
        实例：编写一个基于ItemCF算法的电影推荐系统
"""
import random
import math
import os
import json

class ItemCFRec:
    def __init__(self,datafile,ratio):
        # 原始数据路径文件
        self.datafile = datafile
        # 测试集与训练集的比例
        self.ratio = ratio

        self.data = self.loadData()
        self.trainData,self.testData = self.splitData(3,47)
        self.items_sim = self.ItemSimilarityBest()

# 加载评分数据到data
def loadData(self):
    print("加载数据...")
    data=[]
    for line in open(self.datafile):
        userid,itemid,record,_ = line.split("::")
        data.append((userid,itemid,int(record)))
    return data
```

**seed**: 生成随机数的种子

**M**: 随机数上限

"""

```
def splitData(self,k,seed,M=9):
    print("训练数据集与测试数据集切分...")
    train,test = {},{}
    random.seed(seed)
    for user,item,record in self.data:
        if random.randint(0,M) == k:
            test.setdefault(user,{})
            test[user][item] = record
        else:
            train.setdefault(user,{})
            train[user][item] = record
    return train,test
```

# 计算物品之间的相似度

```
def ItemSimilarityBest(self):
    print("开始计算物品之间的相似度")
    if os.path.exists("data/item_sim.json"):
        print("物品相似度从文件加载 ...")
        itemSim = json.load(open("data/item_sim.json", "r"))
    else:
        itemSim = dict()
        item_user_count = dict() # 得到每个物品有多少用户产生过行为
        count = dict() # 共现矩阵
        for user, item in self.trainData.items():
            print("user is {}".format(user))
            for i in item.keys():
                item_user_count.setdefault(i, 0)
                if self.trainData[str(user)][i] > 0.0:
                    item_user_count[i] += 1
            for j in item.keys():
                count.setdefault(i, {}).setdefault(j, 0)
                if self.trainData[str(user)][i] > 0.0 and self.trainData[str(u
                    count[i][j] += 1
        # 共现矩阵 -> 相似度矩阵
        for i, related_items in count.items():
            itemSim.setdefault(i, dict())
            for j, cuv in related_items.items():
                itemSim[i].setdefault(j, 0)
```

为用户进行推荐

user: 用户

k: k个临近物品

nitem: 总共返回n个物品

"""

```
def recommend(self, user, k=8, nitems=40):
    result = dict()
    u_items = self.trainData.get(user, {})
    for i, pi in u_items.items():
        for j, wj in sorted(self.items_sim[i].items(), key=lambda x: x[1], reverse=True):
            if j in u_items:
                continue
            result.setdefault(j, 0)
            result[j] += pi * wj

    return dict(sorted(result.items(), key=lambda x: x[1], reverse=True)[0:nitems])
```

# 计算准确率

```
def precision(self, k=8, nitems=10):
    print("开始计算准确率 ...")
    hit = 0
    precision = 0
    for user in self.testData.keys():
        u_items = self.testData.get(user, {})
        result = self.recommend(user, k=k, nitems=nitems)
        for item, rate in result.items():
            if item in u_items:
                hit += 1
        precision += nitems
    return hit / (precision * 1.0)
```

```
if __name__ == "__main__":
    ib = ItemCFRec("../data/ml-1m/ratings.dat", [1, 9])
    print("用户1进行推荐的结果如下: {}".format(ib.recommend("1")))
    print("准确率为: {}".format(ib.precision()))
```



### (1) 用户活跃度对物品相似度的影响

即认为活跃用户对物品相似度的贡献应该小于不活跃的用户，所以增加一个IUF (Inverse User Frequency) 参数来修正物品相似度的计算公式：

$$w_{ij} = \frac{\sum_{u \in N(i) \cap N(j)} \frac{1}{\log 1 + |N(u)|}}{\sqrt{|N(i)| |N(j)|}}$$

用这种相似度计算的ItemCF被记为ItemCF-IUF。

ItemCF-IUF在准确率和召回率两个指标上和ItemCF相近，但它明显提高了推荐结果的覆盖率，降低了推荐结果的流行度，从这个意义上说，ItemCF-IUF确实改进了ItemCF的综合性能。

### (2) 物品相似度的归一化

Karypis在研究中发现如果将ItemCF的相似度矩阵按最大值归一化，可以提高推荐的准确度。其研究表明，如果已经得到了物品相似度矩阵 $w$ ，那么可用如下公式得到归一化之后的相似度矩阵 $w'$ ：

$$w'_{ij} = \frac{w_{ij}}{\max_j w_{ij}}$$

最终结果表明，归一化的好处不仅仅在于增加推荐的准确度，它还可以提高推荐的覆盖率和多样性。

用这种相似度计算的ItemCF被记为ItemCF-Norm。

## 四、对比分析UserCF与ItemCF算法

- UserCF算法注重用户所在的兴趣小组，给用推荐兴趣小组的热门物品，更注重**社会化**；
- Item算法注重用户的过往行为的历史物品，用户本省兴趣变化和继承，对用户影响更大，因此更注重**个性化**；



UserCF算法：当物品数量远超用户数量时，可以考虑UserCF算法，例如：新闻类、短视频、热点多、社交性质重、需常更新数据的网站或APP；（抖音、微博）

ItemCF算法：当用户数量远远超过物品数量，可以考虑使用Item算法，例如：购物网站、技术博客、文章等不常更新、数据稳定的网站或APP；（小说APP）

## 2、推荐系统多样性

- 单用户多样性：ItemCF算法小于UserCF算法多样性丰富，ItemCF算法覆盖面小、丰富度低
- 多用户多样性：ItemCF算法大于UserCF算法多样性丰富，因为UserCF算法注重社会热点物品

ItemCF算法推荐具有新颖性，容易发现推荐长尾里的物品，ItemCF算法计算精度小于UserCF算法。如果考虑多样性，那么ItemCF算法计算精度大于UserCF算法，UserCF算法推荐长尾物品能力不足。

## 3、用户特点

UserCF算法适用于用户邻居多情况，例如：微信、微博、支付宝等

ItemCF算法适用于用户喜欢他以前购买过物品类型相似的物品，喜欢同类型物品的自相似度要大的情况

## 五、推荐系统的各类算法使用场景（简单了解）

推荐系统问题，用机器学习的思想来建模解决，主流的方法可以分为：用关联算法，聚类算法，分类算法，回归算法，矩阵分解，神经网络，图模型以及隐语义模型等等来解决。

### （1）常用的关联推荐算法有Apriori，FP Tree和PrefixSpan。

一般我们可以找出用户购买的所有物品数据里频繁出现的项集活序列，来做频繁集挖掘，找到满足支持度阈值的关联物品的频繁N项集或者序列。如果用户购买了频繁N项集或者序列里的部分物品，那么我们可以将频繁项集或序列里的其他物品按一定的评分准则推荐给用户，这个评分准则可以包括支持度，置信度和提升度等。

### （2）常用的聚类推荐算法有K-Means, BIRCH, DBSCAN和谱聚类。

用聚类算法做协同过滤就和前面的**基于用户或者项目的协同过滤**有些类似了。我们可以按照用户或

### (3) 常见的分类推荐算法有逻辑回归和朴素贝叶斯，两者的特点是解释性很强。

如果我们根据用户评分的高低，将分数分成几段的话，则这个问题变成分类问题。比如最直接的，设置一份评分阈值，评分高于阈值的就是推荐，评分低于阈值就是不推荐，我们将问题变成了一个二分类问题。虽然分类问题的算法多如牛毛，但是目前使用最广泛的是逻辑回归。为啥是逻辑回归而不是看起来更加高大上的比如支持向量机呢？因为逻辑回归的解释性比较强，每个物品是否推荐我们都有一个明确的概率放在这，同时可以对数据的特征做工程化，得到调优的目的。目前**逻辑回归做协同过滤**在BAT等大厂已经非常成熟了。

### (4) 常用的回归推荐算法有Ridge回归，回归树和支持向量回归。

用回归算法做协同过滤比分类算法看起来更加的自然。我们的评分可以是一个连续的值而不是离散的值，通过回归模型我们可以得到目标用户对某商品的预测打分。

### (5) 矩阵分解

用矩阵分解做协同过滤是目前使用也很广泛的一种方法。由于传统的奇异值分解SVD要求矩阵不能有缺失数据，必须是稠密的，而我们的用户物品评分矩阵是一个很典型的稀疏矩阵，直接使用传统的SVD到协同过滤是比较复杂的。

目前主流的矩阵分解推荐算法主要是SVD的一些变种，比如FunkSVD，BiasSVD和SVD++。这些算法和传统SVD的最大区别是不再要求将矩阵分解为 $U\Sigma V^T U\Sigma V^T$ 的形式，而变是两个低秩矩阵 $PTQPTQ$ 的乘积形式。

### (6) 用神经网络乃至深度学习做协同过滤

以后的一个趋势。目前比较主流的用两层神经网络来做推荐算法的是**限制玻尔兹曼机(RBM)**。在目前的Netflix算法比赛中，RBM算法的表现很牛。当然如果用深层的神经网络来做协同过滤应该会更好，大厂商用深度学习的方法来做协同过滤应该是将来的一个趋势。

### (7) 推荐系统新方向

a) **基于集成学习的方法和混合推荐**:这个和混合推荐也靠在一起了。由于集成学习的成熟，在推荐算法上也有较好的表现。一个可能取代逻辑回归的算法是**GBDT (必须掌握)**。目前GBDT在很多算法比赛都有好的表现，而有工业级的并行化实现类库。

b) **基于矩阵分解的方法 (LFM)**：矩阵分解，由于方法简单，一直受到青睐。目前开始渐渐流行的矩阵分解方法有分解机(Factorization Machine)和张量分解(Tensor Factorization)。

CNN和RNN的推荐算法。

编辑于 2019-12-05

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

协同过滤

个性化推荐

推荐系统

## 文章被以下专栏收录



**Data Analyst**

一起揭开数据背后不为人知的秘密吧！

关注专栏

## 推荐阅读

### 推荐系统--关于推荐系统的一些思考和协同过滤

过年前打算学习一下推荐，买了一本国内推荐系统的经典，项亮的《推荐系统实践》，因为封城，导致3月末才拿到这本书，今天好好看了一下。正式开始内容之前，想说点题外话，第一个是关于为什...

吴祺育

▲ 赞同 10 ▼

● 2 条评论

➦ 分享

♥ 喜欢

★ 收藏

📄 申请转载

...