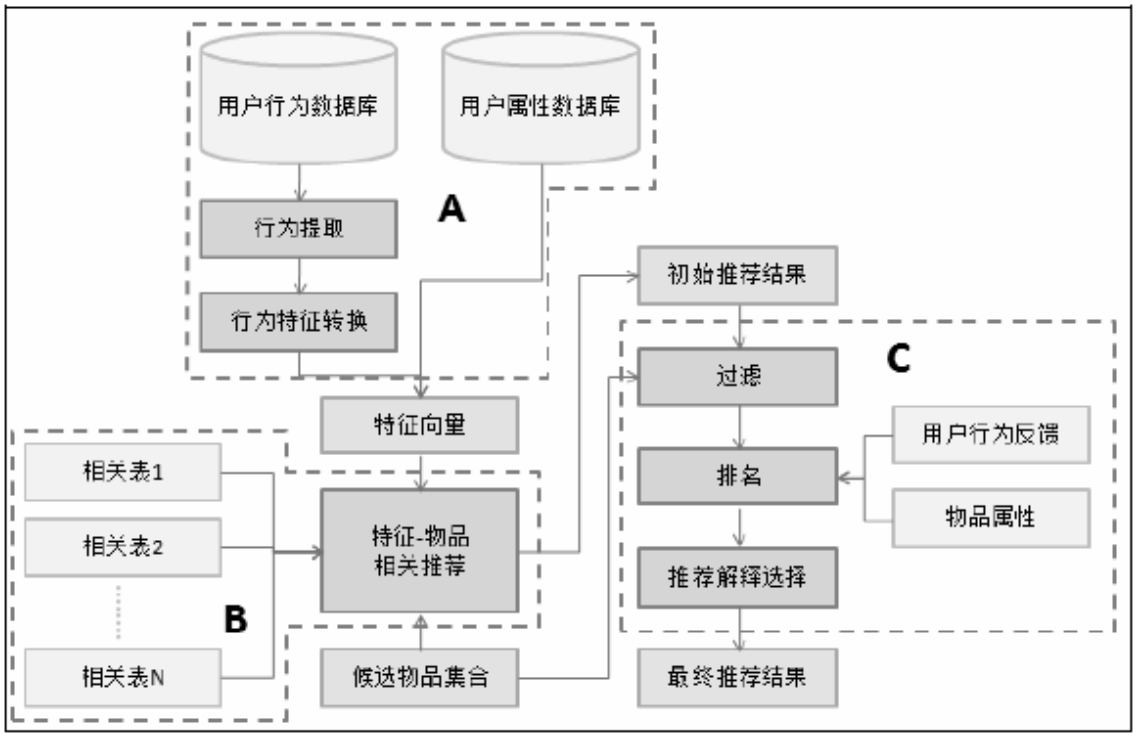
推荐系统实战

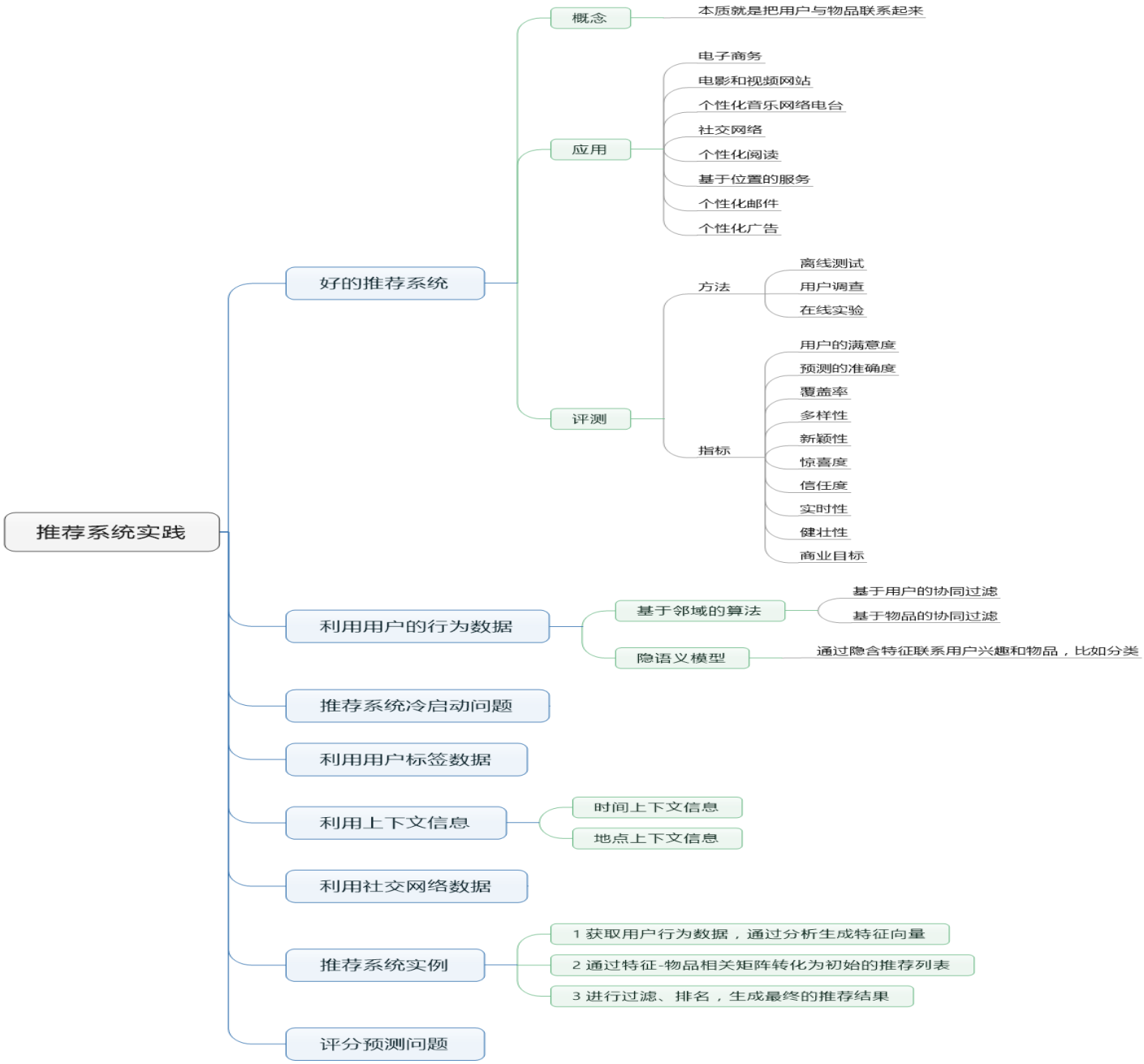
《推荐系统实践》—— 读后总结

在刚刚毕业的时候，当时的领导就问了一个问题——个性化推荐与精准营销的区别，当时朦朦胧胧回答不出。现在想想，他们可以说是角度不同。精准营销可以理解为帮助物品寻找用户，而个性化推荐则是帮助用户寻找物品。

什么是推荐系统？

那么什么是推荐系统呢？简单的来说，就是帮助用户和物品联系起来，让信息展现在对他感谢兴趣的用户面前。
在互联网最开始兴起的时候，最便捷的帮助用户的方法就是进行分类，比如当时的雅虎,hao123 等等。后来互联网兴起，这种分类已经装不下太多的信息，于是出现了搜索引擎，当用户需要什么东西的时候，可以直接主动的去获取。而推荐系统的出现，则帮助用户在没有明确的目的时，根据行为历史或者用户信息为用户提供有价值的东西。
所以一个完整的推荐系统需要包括前段的展示页面，后台的日志系统以及良好的推荐算法。




推荐系统实践
好的推荐系统
概念
本质就是把用户与物品联系起来
应用
电子商务
电影和视频网站
个性化音乐网络电台
社交网络
个性化阅读
基于位置的服务
个性化邮件
个性化广告
评测
方法
离线测试
用户调查
在线实验
指标
用户的满意度
预测的准确度
覆盖率
多样性
新颖性
惊喜度
信任度
实时性
健壮性
商业目标
利用用户的行为数据
基于邻域的算法
基于用户的协同过滤
基于物品的协同过滤
隐语义模型
通过隐含特征联系用户兴趣和物品，比如分类
推荐系统冷启动问题
利用用户标签数据
利用上下文信息
时间上下文信息
地点上下文信息
利用社交网络数据
推荐系统实例
1 获取用户行为数据，通过分析生成特征向量
2 通过特征-物品相关矩阵转化为初始的推荐列表
3 进行过滤、排名，生成最终的推荐结果
评分预测问题

个性化的推荐系统应用

现在个性化推荐已经应用的很广泛了，比如：

- 1 电子商务网站、亚马逊：个性化推荐、相关推荐（打包和相似产品）
- 2 电影和视频网站，Netflix，YouTube,Hulu：基于物品用户评分进行推荐
- 3 个性化音乐网络电台：音乐推荐难度比较大，因为考虑到用户的心情、音乐很短、免费等等
- 4 社交网络：facebook,用户之间的网络关系、用户的偏好关系
- 5 个性化阅读：Google Reader,Digg
- 6 基于位置的服务：基于地理位置推送饭店
- 7 个性化邮件：帮助筛选出优先级高的邮件
- 8 个性化广告：CPM 按照看到广告的次数收费、CPC 按照点击广告的次数收费 、CPA 按照最后的订单收费，个性化推荐帮助用户找到他们感兴趣的东西；广告推荐帮助广告找到对他们感兴趣的用户。

  主要包括：上下文广告（通过用户浏览的内容）、搜索广告、个性化展示

基于行为数据

大多数的推荐系统都是基于用户行为的，当你浏览了一款商品，推荐列表将会更新，推荐一些与你浏览产品相关或者类似的产品。

常见的推荐算法就时基于用户或者物品的协同过滤。

- 基于用户的协同过滤，userCF,即会搜索你的好友喜欢的东西推荐给你
- 基于物品的协同过滤，itemCF,即搜索您喜欢的物品相类似的东西推荐给你

这两种算法都有各自的使用场景的优劣势。

表2-11　UserCF和ItemCF优缺点的对比

	UserCF	ItemCF
性能	适用于用户较少的场合，如果用户很多，计算用户相似度矩阵代价很大	适用于物品数明显小于用户数的场合，如果物品很多（网页），计算物品相似度矩阵代价很大
领域	时效性较强，用户个性化兴趣不太明显的领域	长尾物品丰富，用户个性化需求强烈的领域
实时性	用户有新行为，不一定造成推荐结果的立即变化	用户有新行为，一定会导致推荐结果的实时变化
冷启动	在新用户对很少的物品产生行为后，不能立即对他进行个性化推荐，因为用户相似度表是每隔一段时间离线计算的	新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他物品
	新物品上线后一段时间，一旦有用户对物品产生行为，就可以将新物品推荐给和对它产生行为的用户兴趣相似的其他用户	但没有办法在不离线更新物品相似度表的情况下将新物品推荐给用户
推荐理由	很难提供令用户信服的推荐解释	利用用户的历史行为给用户做推荐解释，可以令用户比较信服

推荐系统冷启动

对于很多公司都是在一定规模才引入推荐系统的，这时候已经拥有了大量的用户行为数据，做推荐算法就很容易了。但是有一些系统想在初期就引入，这就比较困难了。

因为既没有大量的物品，也没有太多的用户关系，做协同过滤就很费劲了。

因此可以考虑费个性化的推荐，比如热门排行、利用用户的注册信息、社交账号、反馈信息等进行推荐。之后再慢慢调整..

在系统的初期也可以考虑选择合适的物品启动用户的兴趣，需要有比较热门、代表性和区分行。

利用用户标签数据

基于标签是一种很简单很暴力的推荐方法，给用户打上相关的标签，然后就可以基于标签进行精准营销或者个性化推荐了。
一般打上的标签都是 物品定义、种类、所有者、观点、用户胡哦哦相关的。也可以分成：类型、时间、人物、地点、语言、等等
一般的标签都是由三元组组成（用户、物品、标签）
在打标签的时候还需要注意标签的清理。

利用上下文信息

因为用户的兴趣是变化的，可能随着季节的效应而变化（比如衣服、考试资料），也可能根据购买的历史（比如你买了一样东西，以后就再也不需要买了）。

因此时间是一个很重要的上下文环境，另外就是地理位置，比如吃饭、逛街等等。

数据挖掘、机器学习、深度学习的含义

数据挖掘：

data mining，是一个很宽泛的概念。字面意思就是从成吨的数据里面挖掘有用的信息。这个工作 BI（商业智能）可以做，数据分析可以做，甚至市场运营也可以做。你用 excel 分析分析数据，发现了一些有用的信息，然后这些信息可以指导你的 business，恭喜你，你已经会数据挖掘了。

机器学习：

machine learning，是计算机科学和统计学的交叉学科，基本目标是学习一个 x->y 的函数（映射），来做分类或者回归的工作。之所以经常和数据挖掘合在一起讲是因为现在好多数据挖掘的工作是通过机器学习提供的算法工具实现的，例如广告的 ctr 预估，PB 级别的点击日志在通过典型的机器学习流程可以得到一个预估模型，从而提高互联网广告的点击率和回报率；个性化推荐，还是通过机器学习的一些算法分析平台上的各种购买，浏览和收藏日志，得到一个推荐模型，来预测你喜欢的商品。

深度学习：

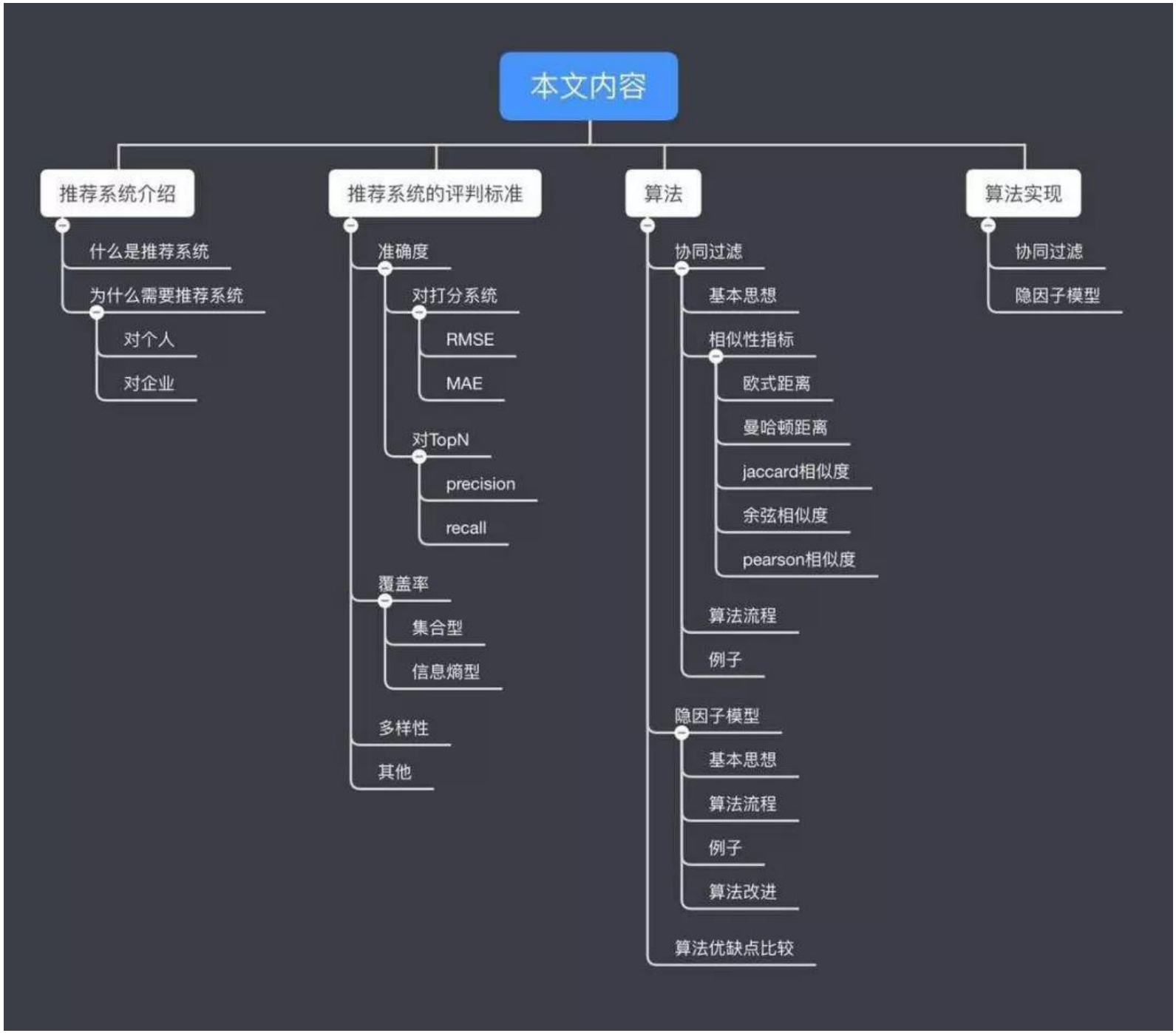
deep learning，机器学习里面现在比较火的一个 topic（大坑），本身是神经网络算法的衍生，在图像，语音等富媒体的分类和识别上取得了非常好的效果，所以各大研究机构和公司都投入了大量的人力做相关的研究和开发。

总结下，数据挖掘是个很宽泛的概念，数据挖掘常用方法大多来自于机器学习这门学科，深度学习是机器学习一类比较火的算法，本质上还是原来的神经网络。

推荐系统实战第 01 课 推荐系统简介：“猜你喜欢”的背后揭秘--10 分钟教你用 Python 打造推荐系统

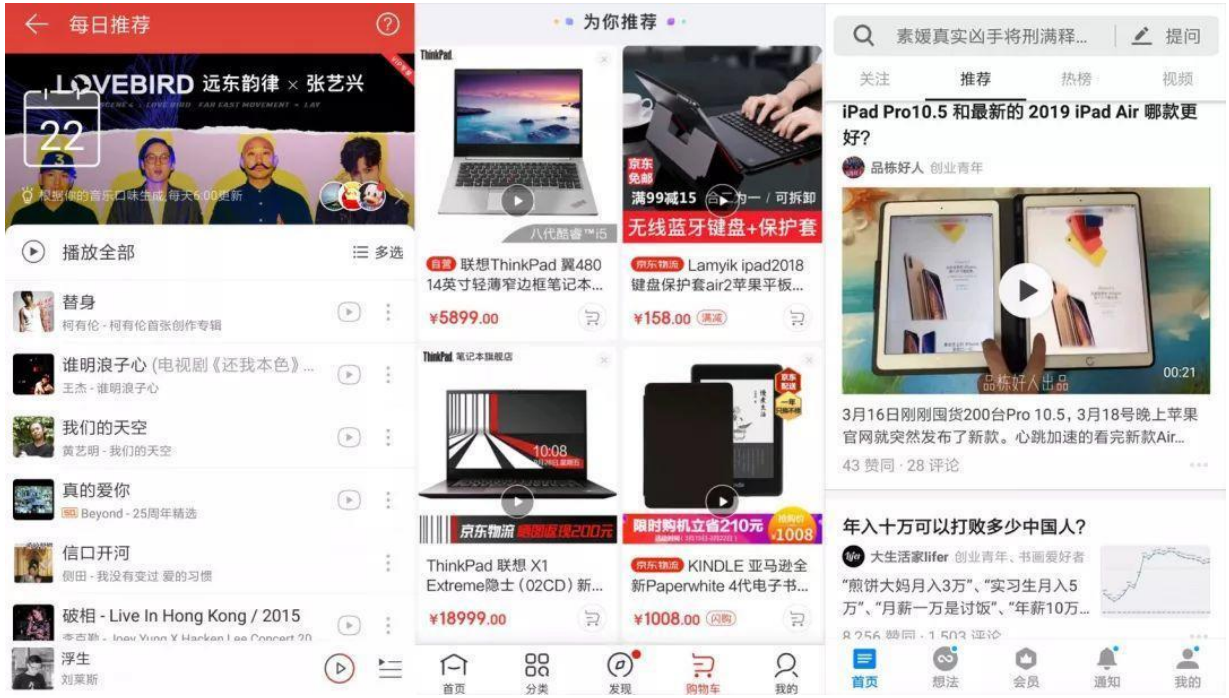
话说，最近的瓜实在有点多，从我科校友李雨桐怒锤某男、陈羽凡吸毒被捕、蒋劲夫家暴的三连瓜，到不知知网翟博士，再到邓紫棋解约蜂鸟、王思聪花千芳隔空互怼。
而最近的胜利夜店、张紫妍巨瓜案、最强大脑选手作弊丑闻，更是让吃瓜群众直呼忙不过来：**瓜来的太快就像龙卷风，扶我起来，我还能吃！**
说到底，这其实是一个**信息过载**的时代：公众号每天数十条的推送、朋友圈的晒娃晒旅游、各种新闻报道扑面而来令人眼花缭乱、目不暇接……
那么问题来了，怎么找到自己的关注点呢？俗话说得好，有问题，找度娘，输入关键词一回车就完事儿了。
然而，懒是人的天性，而有的人（比如小编）则连关键词都懒得搜，**希望计算机能自动挖掘我们的兴趣点，并为我们推荐感兴趣的内容**，所以我们就迫切地需要推荐系统来帮助我们了。那么现在我们就来讲讲**推荐系统**吧~

目录


本文内容
推荐系统介绍
什么是推荐系统
为什么需要推荐系统
对个人
对企业
推荐系统的评判标准
准确度
对打分系统
RMSE
MAE
对TopN
precision
recall
覆盖率
集合型
信息熵型
多样性
其他
算法
协同过滤
基本思想
相似性指标
欧式距离
曼哈顿距离
jaccard相似度
余弦相似度
pearson相似度
算法流程
例子
隐因子模型
基本思想
算法流程
例子
算法改进
算法优缺点比较
算法实现
协同过滤
隐因子模型

01 什么是推荐系统

推荐系统相信大家并不陌生，从“我有歌也有热评”的云村里的每日歌曲推荐，淘宝的猜你喜欢，再到外卖 APP 和视频网站的推送，推荐系统似乎成了各种 APP 的宠儿（请忽略小编的老年人口味）。



热门 app 的推荐系统

简单来说，**推荐系统就是根据用户的各种数据（历史行为数据、社交关系数据、关注点、上下文环境等）在海量数据中判断用户感兴趣的 item 并推荐给用户的系统。**

“哇这个东西就是我想要的。”

“诶，这首歌还真好听。”

“emm 这部电影还挺对我胃口的”

推荐系统对用户而言，能够简化搜寻过程、发现新鲜好玩令人惊喜的东西；而对商家而言，则是能够提供个性化服务，提高用户的信任度、粘度以及活跃度，从而提高营收。

02 推荐系统的评判标准

一个完整的推荐系统往往是十分复杂的。既然如此，仅仅通过准确率一个标准推荐系统作评测是远远不够的，为此我们需要定义多个标准，**从多维度评价一个推荐系统的好坏**。

符号	含义
U	全体USER的集合
I	全体ITEM的集合
T	全体USER行为集合
i	ITEM I
u	USER u
r_{ui}	USER u 对ITEM i 的实际打分
$\hat{r}_{ui}$	推荐系统预测的USER u 对ITEM i的打分
R(u)	推荐系统对USER u的推荐列表
T(u)	User u 的行为集合
p(i)	ITEM i被推荐的概率
X,Y	一个user的打分列向量
μ_x ,μ_y	X,Y分量的均值
e_{ij}	USER u对ITEM i打分的损失函数
p_{ik}	分解矩阵P的第i行第k列的分量
q_{kj}	分解矩阵Q的第k行第j列的分量
$S_{(i,j)}$	一个推荐列表中ITEM i和ITEM j的相似度

notation

1.准确度

对**打分系统**（比如说淘宝的评论一到五星打分）而言，一说到评判标准，最先想到的肯定就是**均方根误差 RMSE** 和**平均绝对误差 MAE** 啦：

$$RMSE = \sqrt{\frac{\sum_{u,i \in T}(r_{ui} - \hat{r}_{ui})^2}{|T|}}$$

$$MAE = \frac{\sum_{u,i \in T}|r_{ui} - \hat{r}_{ui}|}{|T|}$$

RMSE 和 MAE 计算公式
对 **TOP N 推荐**（生成一个 TOP N 推荐列表）而言，**Precision** 和 **Recall** 则是我们的关注点：

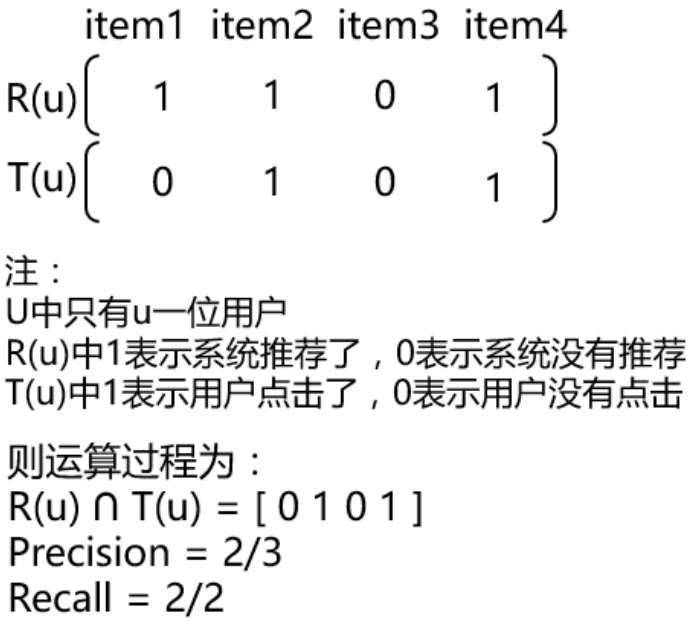
$$Precision = \frac{\sum_{u \in U}|R(u) \bigcap T(u)|}{\sum_{u \in U}|R(u)|}$$

$$Recall = \frac{\sum_{u \in U}|R(u) \bigcap T(u)|}{\sum_{u \in U}|T(u)|}$$

Precision 和 Recall 计算公式

说人话版本：Precision 就是指系统推荐的东西中用户感兴趣的有多少，Recall 就是用户感兴趣的东西中你推荐了多少

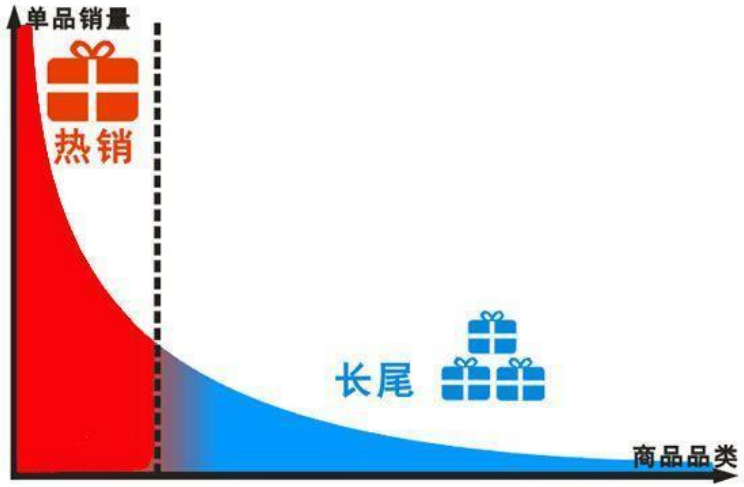
举个栗子：



2.覆盖率

表示推荐系统对 item 长尾的发掘能力

科普知识
马太效应：强者愈强，弱者愈弱
长尾效应：大多数的需求会集中在头部（爆款商品），而分布在尾部的需求是个性化的、零散的、小量的需求（冷门商品）。但这部分差异化的、少量的需求会在需求曲线上面形成一条长长的“尾巴”。**如果将所有非流行的市场累加起来就会形成一个比流行市场还大的市场。**



由长尾效应可知，满足用户个性化的需求十分重要，所以**推荐系统要尽可能地挖掘出合用户口味的冷门商品**；如果推荐系统严重偏向推荐热门商品，那么只会造成热门商品越热门，冷门商品越冷门，对商家的营收十分不利。

所以这就要求推荐系统的**覆盖率**要高(推荐系统对所有用户推荐的 item 占 item 总数的比例)：

$$Coverage = \frac{|U_{u \in U} R(u)|}{|I|}$$

覆盖率还有另一种计算方式：**信息熵**

$$p = \sum_{n=1}^{N} p(i) log p(i)$$

其中，p(i)=第 i 件 item 被推荐次数/所有 item 总被推荐次数

科普时间：

“信息是用来消除随机不确定性的东西。”由高中化学知道，熵是用来衡量一个系统的混乱程度的，熵越大，混乱程度越高。而同样的，**信息熵越大，对一件事情的不确定性就越大**。

32 支球队打世界杯，只有一队胜利，但是我们不知道关于比赛、关于球队的任何信息，也就是说每支球队获胜的概率为 1/32。如果我们想要知道哪支队伍胜利，我们只能无任何根据地瞎猜，那么最要猜几次呢？通过折半查找法我们可以发现，我们顶多五次就能找到胜利的球队。所以说，**这个问题的最大信息熵就是 5bit**（信息熵在 p(i)全部相等时最大）。

如果这时候，我们知道了更多的信息，比如说是哪国球队（德国队和中国队你选哪个？）、球队的既往胜率，那么这时候，各个球队获胜的概率就发生了变动，而此时信息熵也得到了降低。

回到我们的推荐系统中来，**信息熵越大，表明 item 之间的 p(i)越接近，也就是说每个 item 被推荐的次数越接近， 即覆盖率越大**。

3.多样性

表示推荐列表中物品之间的不相似性

Q：为什么需要多样性？

A：试想，一个喜欢裙子的用户，如果你给她推荐的十件商品都是裙子，那她可能也只会买一条最合心意的裙子，倒不如把一部分的推荐名额给其他种类的商品；另外，一位用户买了一台计算机，你还给他推荐另外的计算机吗？从商家的角度看，推荐鼠标、键盘等是最好的选择。

所以我们的推荐列表需要**尽可能地拓宽种类，增加用户的购买欲望。**

$$Diversity(R(u)) = 1 - \frac{\sum_{i,j \in R(u), i \neq j} S(i,j)}{\frac{1}{2}|R(u)|(|R(u)|-1)}$$

$$Diversity = \frac{1}{|U|} \sum_{u \in U} Diversity(R(u))$$

4.还有其他的评判标准

新颖度：新颖的推荐是指给用户推荐那些他们以前没有听说过的物品。
惊喜度：推荐结果和用户的历史兴趣不相似，但却让用户觉得满意。（而新颖性仅仅取决于用户是否听说过这个推荐结果。）
信任度：推荐系统给你推荐的依据是什么（“你的朋友也喜欢这首歌”比起“喜欢那首歌的人也喜欢这首歌”更能让用户信任）

03 算法（具体实现请看第四部分）

1．协同过滤****（collaborative filtering）

回想一下，当你遇到剧荒、歌荒时，会怎么做？
“诶，小陈，最近有啥好看的电视剧，给我安利安利呗。”
相信大多数人首先想到的都是找一个趣味相投的朋友，问一下他们有啥好推荐的。
协同过滤也就是基于这样的思想。而协同过滤分为两种：**user-based**（基于用户，找到最相似的 user）和 **item-based**（基于商品，找到最相似的 item）。
那么，协同过滤需要解决的核心问题是：
如何找到最相似的 user/item？
因此我们需要衡量**相似性**的指标：

1.欧氏距离

$$dist = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2.Jaccard相似度

$$J(X,Y) = \frac{|X \bigcap Y|}{|X \bigcup Y|}$$

3.Cos相似度

$$cos(X,Y) = \frac{X^T Y}{|X| \cdot |Y|}$$

4.Pearson相似度

$$Pearson = \frac{\sum_{i=1}^{n}(X_i - \mu_X)(Y_i - \mu_Y)}{\sqrt{\sum_{i=1}^{n}(X_i - \mu_X)^2}\sqrt{\sum_{i=1}^{n}(Y_i - \mu_Y)^2}}$$

其实 Pearson 相似度是考虑了 user/item 之间的差异而来。

Q:为什么要减去 mean？
A:比如，user1 打分十分苛刻，3 分已经是相当好的商品；而 user2 是一位佛系用户，无论商品有多差，打分时 3 分起步。那么我们可以说 user1 的 3 分和 user2 的 5 分是等价的。而 Pearson 相似度就是通过减去 mean 来进行相对地归一化。

user-based/item-based 算法流程：

1. 计算目标 user/item 和其余 user/item 的相似度
2. 选择与目标 user/item 有正相似度的 user/item
3. 加权打分

举个栗子（使用 Pearson 相似度和 user-based）：
我们有这样一个打分表，想要预测 USER2 对 ITEM3 的打分

sim=	-0.5		-0.2858	0.4999	0.2236
	USER1	**USER2**	**USER3**	**USER4**	**USER5**
ITEM1	2	5		3	5
ITEM2		3	4	1	4
ITEM3	4	?	3		2
ITEM4	3		4	5	1

Step1:依次计算USER i (i≠2)和USER2的相似度
mean1 = (2+4+3)/3 = 3
mean2 = (5+3)/2 = 4
row1 = [2-3, 0, 4-3, 3-3] = [-1, 0, 1, 0]
row2 = [5-4, 3-4, 0, 0] = [1, -1, 0. 0]
sim(USER1,USER2) = row1*row2T /(|row1|·|row2|) = -1/2
同理，得到
sim(USER3,USER2) = - 0.2858, sim(USER4,USER2) =0.4999,
sim(USER5,USER2) =0.2236
Step2:加权打分
选择正相似度的USER4和USER5
而USER4没有使用过ITEM3
所以预测值=2*0.2236/0.2236 = 2
而mean2 = 4
看起来是个相当低的分数，所以就不给USER2推荐ITEM3啦

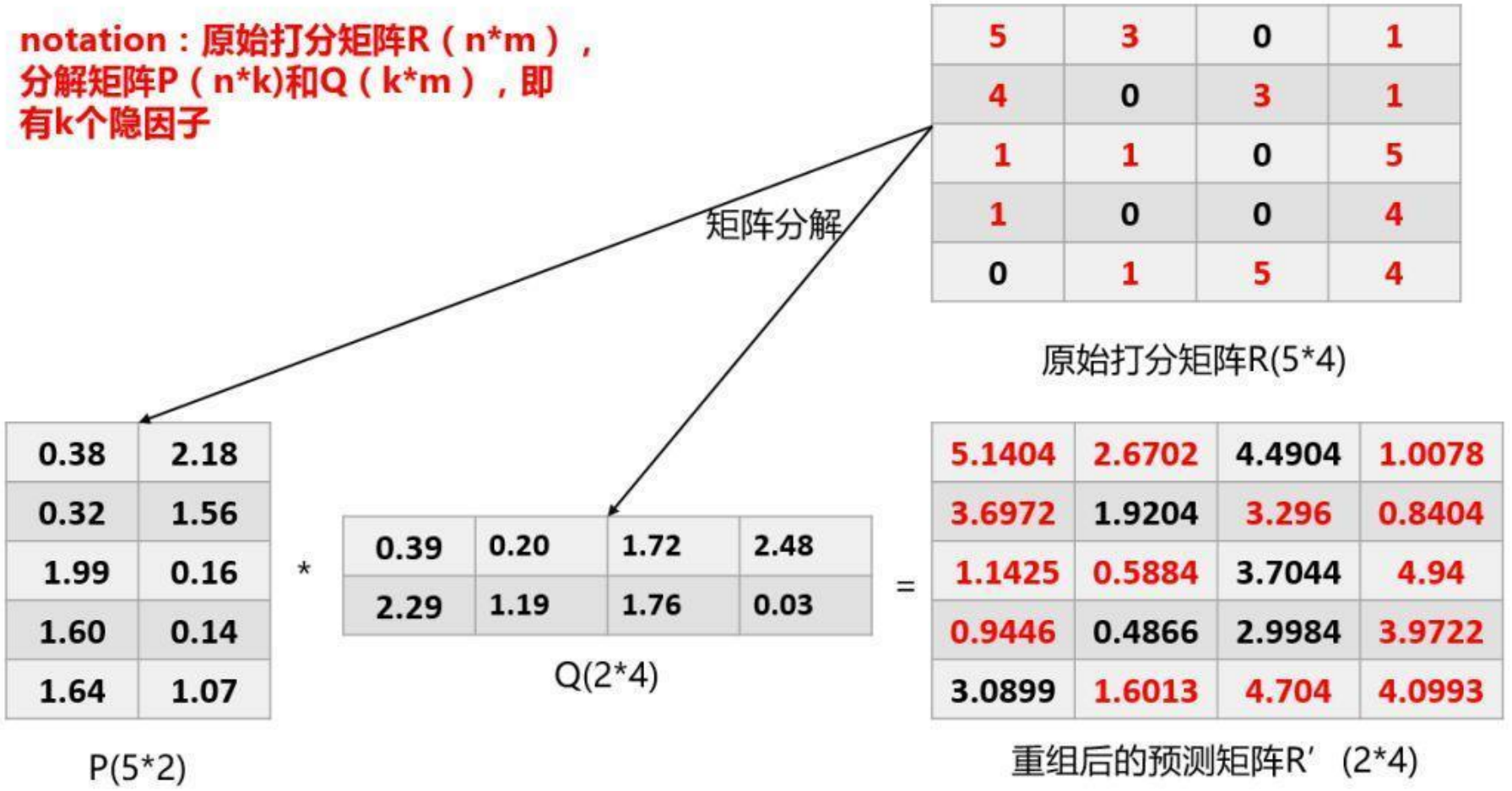
2. 隐因子模型（Latent Factors Model）

我们有一个 user 对 item 的打分矩阵，但是有一些位置是空着的（我们候选推荐的 item），所以我们要做的，就是把这些空位一网打尽，一次性填满。

隐因子模型的基本思想就是：

打分矩阵 R 只有两个维度：user 和 item，那么能否引入影响用户打分的隐藏因素（即隐因子,不一定是人可以理解的，如同神经网络一样的黑箱）在 user 和 item 之间搭一座桥（**分解为多个矩阵，使得这些矩阵的乘积近似于 R**）

举个例子:



隐因子模型举例

仔细观察，在非 0 的部位上 R 和 R’十分接近，而在 0 的部位上，R’得到了填充，而这可以作为我们推荐的依据。

Q:如何分解矩阵？

A:说到矩阵分解，首先想到的就是 SVD 了（Python AI 教学|SVD（Singular Value Decomposition）算法及应用）。

*然而，SVD 的时间复杂度为 O（n3），在这里小编推荐另一种实现：**梯度下降***

算法流程：

1.定义损失函数

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

2.求解梯度

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}q_{kj}$$

$$\frac{\partial}{\partial q_{kj}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}p_{ik}$$

3.迭代更新

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

隐因子模型的梯度下降实现算法流程

算法改进：

前面提到了用户之间评分标准的差异性（某些用户比较严格），而我们通过引入**正则化项**和**偏差项**（user bias 和 item bias）来进行优化。由于篇幅原因，本文不作具体解释，感兴趣的朋友麻烦自行百度啦～

$$\min \sum_{(x,i)\in R} (r_{xi} - (\mu + b_x + b_i + q_i p_x))^2 + (\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2)$$

3．算法优缺点比较：

冷启动问题：

对于冷启动问题，一般分为三类：

用户冷启动：如何对新用户做个性化推荐。
物品冷启动：如何将新加进来的物品推荐给对它感兴趣的用户。
系统冷启动：新开发的网站如何设计个性化推荐系统。

	协同过滤	隐因子模型
优点	1.流程简单 2.可解释性强 3.不需要知道item和user的具体信息，只要有用户评分表就可	1.预测精度较高 2.能更好地挖掘user和item之间的隐藏联系
缺点	1.不能有效地应对冷启动问题 2.可能存在gray sheep问题（没有相似的用户）	1.模型训练比较费时 2.不具有良好的可解释性。分解出来的用户和物品矩阵的每个维度，无法和现实生活中的概念来解释，无法用现实概念给每个维度命名，只能理解为潜在语义空间

	UserCF	ItemCF
性能	适用于用户较少的场合，如果用户很多，计算用户相似度矩阵代价很大	适用于物品数明显小于用户数的场合，如果物品很多（网页），计算物品相似度矩阵代价很大
领域	时效性较强，用户个性化兴趣不太明显的领域	长尾物品丰富，用户个性化需求强烈的领域
实时性	用户有新行为，不一定造成推荐结果的立即变化	用户有新行为，一定会导致推荐结果的实时变化
冷启动	在新用户对很少的物品产生行为后，不能立即对他进行个性化推荐，因为用户相似度表是每隔一段时间离线计算的	新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他物品
	新物品上线后一段时间，一旦有用户对物品产生行为，就可以将新物品推荐给和对它产生行为的用户兴趣相似的其他用户	但没有办法在不离线更新物品相似度表的情况下将新物品推荐给用户
推荐理由	很难提供令用户信服的推荐解释	利用用户的历史行为给用户做推荐解释，可以令用户比较信服

在 user-based 和 item-based 之间，一般使用 item-based，因为 item-based 稳定性高（user 的打分标准、喜好等飘忽不定，有很大的不确定性），而且 user 太多时计算量大。

04 手把手打造一个推荐系统

都说女人心海底针，还在为选什么电影才能打动妹子烦恼吗？还在担心无法彰显自己的品味吗？相信下面的电影推荐系统代码一定能够回答你关于如何选择电影的疑惑。（捂脸，逃）

1. 数据源

小编通过问卷调查获取了朋友圈对 15 部电影的评分（1 代表第一个选项即没看过，$2_{6\text{ 表示 }1}$5 星）

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	序号	提交答卷时间	所用时间	来源	来源详情	这个杀手不太冷	流浪地球	喜剧之王	夏洛特烦恼	冰雪奇缘	复仇者联盟3:无限战争	怦然心动	战狼2	三傻大闹宝莱坞	蝙蝠侠：黑暗骑士	血战钢锯岭	秒速五厘米
2	1	2019/3/20 21:57:36	44秒	链接	直接访问	1	1	1	1	4	4	1	5	6	4	4	
3	2	2019/3/20 22:01:11	34秒	手机提交	直接访问	5	5	1	1	4	5	1	5	1	5	6	
4	3	2019/3/20 22:01:42	156秒	链接	直接访问	5	6	6	4	5	5	5	5	6	6	4	
5	4	2019/3/20 22:02:39	39秒	微信	N/A	5	5	5	2	6	4	5	6	1	1	1	
6	5	2019/3/20 22:05:28	36秒	微信	N/A	1	5	6	4	1	1	1	4	5	5	4	
7	6	2019/3/20 22:10:01	34秒	微信	N/A	1	6	6	6	5	1	5	6	1	1	1	
8	7	2019/3/20 22:10:54	44秒	微信	N/A	5	5	6	5	6	5	5	5	5	6	6	
9	8	2019/3/20 22:11:04	65秒	微信	N/A	4	4	6	6	4	6	4	4	4	5	5	
10	9	2019/3/20 22:14:23	52秒	微信	N/A	1	6	6	1	4	5	1	6	5	5	1	
11	10	2019/3/20 22:18:12	30秒	微信	N/A	6	4	1	1	3	6	6	1	6	6	6	
12	11	2019/3/20 22:28:49	37秒	微信	N/A	6	6	6	6	4	5	5	6	4	6	6	
13	12	2019/3/20 22:40:13	45秒	微信	N/A	1	6	1	1	1	1	6	1	4	1	1	
14	13	2019/3/20 23:13:21	49秒	微信	N/A	1	5	1	1	1	1	4	6	6	1	1	
15	14	2019/3/20 23:15:16	87秒	微信	N/A	5	5	5	5	6	5	5	2	5	5	5	
16	15	2019/3/20 23:21:34	24秒	微信	N/A	4	4	1	4	1	5	4	4	4	4	1	
17	16	2019/3/20 23:55:21	44秒	微信	N/A	5	1	5	5	5	1	1	5	1	1	1	
18	17	2019/3/21 12:03:49	42秒	微信	N/A	5	1	4	3	5	6	6	4	1	6	5	
19	18	2019/3/21 12:05:48	47秒	微信	N/A	6	5	1	1	4	5	4	4	4	4	1	
20	19	2019/3/21 12:05:52	27秒	微信	N/A	6	5	1	5	5	5	6	4	6	6	6	
21	20	2019/3/21 12:06:25	42秒	微信	N/A	6	6	6	6	4	5	6	6	6	5	4	

2.代码

注：python 有许多方便计算的函数，如 norm()计算向量的模和 corrcoef()计算 pearson 相似度，不过为了广大朋友们记忆深刻，小编这里自己来实现这些计算~

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

"""
Created on Thu Mar 21 19:29:54 2019

@author: o
"""
import pandas as pd
import numpy as np
from math import sqrt
import copy

def load_data(path):
    df = pd.read_excel(path)
    #去掉不需要的列
    df = df[df.columns[5:]]
    #1 表示没看过，2~6 表示 1~5 星
```

```
    df.replace([1,2,3,4,5,6],[0,1,2,3,4,5],inplace = True)
    columns = df.columns
    df = np.array(df)
    #测试过程中发现有 nan 存在，原来是因为有人恶作剧，全填了没看过，导致分母为 0
    #此处要删除全为 0 的行
    delete = []
    for i in range(df.shape[0]):
        all_0 = (df[i] == [0]*15)
        flag = False
        for k in range(15):
            if all_0[k] == False:
                flag = True
                break
        if flag == False:
            delete.append(i)
            print(i)
    df = np.delete(df,delete,0)
    return df,columns


#定义几种衡量相似度的标准
#余弦相似度
def cos(score,your_score):
    cos = []
    len1 = 0
    for i in range(15):
        len1 += pow(your_score[i],2)
    len1 = sqrt(len1)
    for i in range(score.shape[0]):
        len2 = 0
        for k in range(15):
            len2 += pow(score[i][k],2)
        len2 = sqrt(len2)
        cos.append(np.dot(your_score,score[i])/(len1*len2))
```

```
    return cos

#欧氏距离
def euclidean(score,your_score):
    euclidean = []
    for i in range(score.shape[0]):
        dist = 0
        for k in range(score.shape[1]):
            dist += pow((score[i][k]-your_score[k]),2)
        dist = sqrt(dist)
        euclidean.append(dist)
    return euclidean


#pearson 相似度
#Python 有内置函数 corrcoef()可以直接计算，不过这里还是手写巩固一下吧~
def pearson(score,your_score):
    pearson = []
    n = score.shape[1]
    sum_y = 0
    count = 0
    #计算目标用户打分的均值
    for i in range(n):
        if your_score[i]!=0:
            count += 1
            sum_y += your_score[i]
    mean_y = sum_y/count
    print('\n')
    print('你的平均打分为：',mean_y)
    print('\n')
    #计算目标用户打分向量的长度
    len1 = 0
    for i in range(n):
        if your_score[i]!=0:
```

```
            your_score[i] -= mean_y
        len1 += pow(your_score[i],2)
    len1 = sqrt(len1)
    #print(len1,mean_y,your_score)

    for i in range(score.shape[0]):
        #计算其他用户打分的均值
       # print(i,score[i])
        count = 0
        sum_x = 0
        for k in range(n):
            if score[i][k]!=0:
                count += 1
                sum_x += score[i][k]
        mean_x = sum_x/count
        #计算其他用户打分向量的长度
        len2 = 0
        for k in range(n):
            if score[i][k]!=0:
                score[i][k] -= mean_x
            len2 += pow(score[i][k],2)
        len2 = sqrt(len2)
        #print(len2,mean_x,score[i],'\n','\n')
        #分母不可为零，不然会产生 nan
        if len2 == 0:
            pearson.append(0)
        else:
            pearson.append(np.dot(your_score,score[i])/len1/len2)
    return pearson,mean_y


#找到相似度最高的用户
def find_nearest(sim):
    index = [i for i in range(len(sim))]
```

```
    #index 和 sim 的元组列表
    sorted_value = list(zip(index,sim))
    #降序排序
    sorted_value = sorted(sorted_value,key = lambda x : x[1],reverse = True)
    return sorted_value


#user-based collaborative_filtering
def collaborative_filtering(score,your_score,movies):
    #目标用户对 15 部电影有无看过的 bool 列表
    seen1 = np.array([bool(i) for i in your_score])
    #使用 pearson 过程中会改变 score 矩阵的值，需要用 deepcopy 复制一份
    score1 = copy.deepcopy(score)
    #几种相似度的衡量
    #sim = cos(score,your_score)
    #sim = euclidean(score,your_score)
    sim, mean_target= pearson(score1,your_score)
    #找到最相似的用户
    sorted_value = find_nearest(sim)

    #找到相似度>0 的用户数量
    count = 0
    for i in range(score.shape[0]):
        if sorted_value[i][1]<=0:
            break
        else:
            count += 1
    #取根值，去掉正相似度中偏低的 user
    count = int(sqrt(count))

    #加权打分 进行推荐
    print('使用 user-based 协同过滤进行加权预测打分：')
    for i in range(score.shape[1]):
        #如果目标用户没看过
```

```
        if not seen1[i]:
            #初始化分子分母
            numerator = denominator = 0
            for k in range(count):
                index = sorted_value[k][0]
                if score[index][i] != 0:
                    numerator += score[index][i]*sorted_value[k][1]
                    denominator += sorted_value[k][1]
            if not denominator:
                print(movies[i],'无相关度高的人看过，无法预测得分')
            elif numerator/denominator > mean_target:
                print(movies[i],':',numerator/denominator,'推荐观看')
            else:
                print(movies[i],":",numerator/denominator,'不推荐观看')
    return None


#梯度下降+隐因子模型
def latent_factors(score,your_score,movies):
    #目标用户的打分向量整合进打分矩阵
    score1 = np.vstack([your_score,score])
    #打分矩阵的维度
    n,m = score1.shape[0],score1.shape[1]
    #隐因子数量设为 K
    K = 15
    #最大迭代次数
    max_iteration = 1000
    #学习速率和正则化因子
    alpha = 0.01
    beta = 0.01
    #LossFunction 改变值小于 threshold 就结束
    threshold = 0.7
    #迭代次数
    count = 0
```

```
188    #初始化分解后的矩阵 P、Q
189    p = np.random.random([n,K])
190    q = np.random.random([m,K])
191    #全体用户对 15 部电影有无看过的 bool 矩阵
192    bool_matrix = [[bool(k) for k in i] for i in score1]
193
194    while True:
195        count += 1
196        #更新 P Q 矩阵
197        for i in range(n):
198            for j in range(m):
199                if bool_matrix[i][j]:
200                    eij = score1[i][j] - np.dot(p[i],q[j])
201                    for k in range(K):
202                        #同时更新 pik 和 qjk
203                        diff=[0,0]
204                        diff[0] = p[i][k] + alpha*(2*eij*q[j][k]-beta*p[i][k])
205                        diff[1] = q[j][k] + alpha*(2*eij*p[i][k]-beta*q[j][k])
206                        p[i][k] = diff[0]
207                        q[j][k] = diff[1]
208        #计算误差
209        error = 0
210        for i in range(n):
211            for j in range(m):
212                if bool_matrix[i][j]:
213                    error += pow((score1[i][j]-np.dot(p[i],q[j])),2)
214                    for k in range(K):
215                        error += beta/2*(pow(p[i][k],2)+pow(q[j][k],2))
216        RMSE = sqrt(error/n)
217        print(count,'root_mean_square_error:',RMSE)
218        if RMSE<threshold or count>max_iteration:
219            break
220
221    #打印目标用户没看过的电影
```

```
    seen1 = np.array([bool(i) for i in your_score])
    print('你没看过的影片有：')
    for i in range(m):
        if not seen1[i]:
            print(movies[i])
    print('\n')

    #输出梯度下降预测分数
    predict = np.dot(p[0],q.T)
    print('使用梯度下降+隐因子模型进行预测打分：')
    for i in range(m):
        if not seen1[i]:
            print(movies[i],':',predict[i])
    return None


def recommend():
    #请自行修改路径
    path = r'C:\Users\o\Desktop\请给 15 部电影打分吧.xls'
    #原始打分矩阵，电影名称列表
    score, movies = load_data(path)
    #必须转换成 Float 类型，不然会出现分母为 0 的情况
    score = score.astype(float)
    your_score = []
    #构造目标用户打分向量
    print('请依次输入你对 15 部电影的打分(0 表示没看过，1~5 表示 1~5 星，以空格分隔):')
    print(movies)
    str = input()
    your_score = np.array([int(i) for i in str.split(' ')]).astype(float)
    #进行预测
    latent_factors(score,your_score,movies)
    collaborative_filtering(score,your_score,movies)
    return None

```

```
256
257if __name__=='__main__':
258    recommend()
```

用 python 做推荐系统（一）

一、简介：

推荐系统是最常见的数据分析应用之一，包含淘宝、豆瓣、今日头条都是利用推荐系统来推荐用户内容。推荐算法的方式分为两种，一种是根据用户推荐，一种是根据商品推荐，根据用户推荐主要是找出和这个用户兴趣相近的其他用户，再推荐其他用户也喜欢的东西给这个用户，而根据商品推荐则是根据喜欢这个商品的人也喜欢哪些商品区进行推荐，现在很多是基于这两种算法去进行混合应用。本文会用 python 演示第一种算法，目标是对用户推荐电影。

二、获取数据：

在 movielens 上，有许多的用户对电影评价数据，可以至（https://grouplens.org/datasets/movielens/）进行下载，下载完后打开资料夹，有个叫 u.data 的资料夹，打开会看到以下的数据

```
196    242    3    881250949
186    302    3    891717742
22     377    1    878887116
244    51     2    880606923
166    346    1    886397596
298    474    4    884182806
115    265    2    881171488
253    465    5    891628467
305    451    3    886324817
6      86     3    883603013
62     257    2    879372434
```

第一列代表用户 ID，第二列代表电影的 ID，第三列代表评分（1-5 分），第四列是时间戳

三、数据预处理

拿到了原始数据后，我们会发现几个问题，就是 1、我根本不需要时间戳。2、同一个用户的评价散落在不连续好几行，不好进行分析。这个时候我们就需要进行数据的预处理，首先观察这份数据发现由于每个用户评价的电影都不相同，可能在 200 部里随机挑了 5-10 部来评分，所以如果用表格来显示的话会有很多的空格，这个时候 KV 型数据储存方式就很好用，利用一个键（key）对应一个值（value），这个时候就可以利用 python 的字典（dict），他可以记录键值对应。举例来说，我用户 ID 为‘941’的用户，对电影 ID 为‘763’的评价是 3 分，那我只需要储存【941】【763】=3 这样就可以了，并把 user 合并，让界面更美观，如下图

```
3.0}, '941': {'147': 4.0, '124': 5.0, '117': 5.0, '181': 5.0, '993': 4.0, '258': 4.0, '7': 4.0, '475': 4.0, '257': 4.0,
'15': 4.0, '455': 4.0, '222': 2.0, '358': 2.0, '763': 3.0, '298': 5.0, '408': 5.0, '300': 4.0, '919': 5.0, '273': 3.0, '
1': 5.0, '294': 4.0, '1007': 4.0}}
```

可以看到我把用户 ID 为'941'评价的电影都列了出来，后面还跟了评分值，这样我后面在做分析的时候读取数据就比较方便了，下面是数据读取到处理的代码

```
def load_data():
    f = open('u.data')
    user_list={}
    for line in f:
        (user,movie,rating,ts) = line.split('\t')
        user_list.setdefault(user,{})
        user_list[user][movie] = float(rating)
    return user_list
```

user_list 就是我们建立用来分析的名单

三、算法：

这边是使用最简单的欧几里得距离算法，简单来说就是将两人对同一部电影的评价相减平方再开根号，比如 A 看了蝙蝠侠给了 5 分，B 看了给了 5 分，但 C 看了给分，AB 距离是 0， AC 距离是 2，可以得知 A 和 B 的喜好比较相近，当然现在推荐系统算法很多，这边只是挑了一个比较简单的算法，下面是算法的代码

```
def calculate():
    list = load_data()
    user_diff = {}
    for movies in list['7']:
        for people in list.keys():
            user_diff.setdefault(people,{})
            for item in list[people].keys():
                if item == movies:
                    diff = sqrt(pow(list['7'][movies] - list[people][item],2))
                    user_diff[people][item] = diff
    return user_diff
```

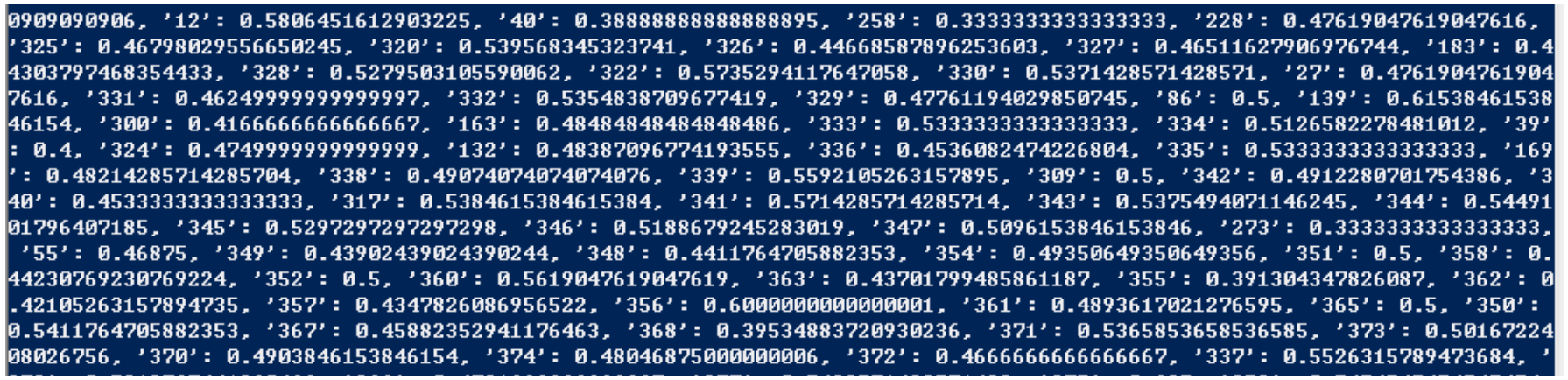
这边挑了其中一位 ID 为 7 的用户，我的任务是帮他找出他可能会感兴趣的电影，所以先计算所有用户跟 7 的距离，由于 7 跟其他用户都看了不同的电影，所以要先找出共同看过的电影再将所有电影的距离列出来。

接下来再把所有电影的距离取平均值，由于我们想知道的是相似度，相似度与平均值成反比，所以我们将距离倒过来就是相似度，另外为了让相似度这个数介于 0~1，所以用了 1/（1+距离）这个算法，以下为代码

```
def people_rating():
    user_diff = calculate()
    rating = {}
    for people in user_diff.keys():
```

```
        rating.setdefault(people,{})
        a = 0
        b = 0
        for score in user_diff[people].values():
            a+=score
            b+=1
        rating[people] = float(1/(1+(a/b)))
    return rating
```

可以看到虽然代码有点丑，不过还是可以跑出个结果，下面就是结果，可以看到跟所有用户的相似度，可以看到跟 ID 为 12 的用户相似度为 0.58， 跟 ID 为 258 的用户相似度为 0.333



现在就是要从这里面找出几个相似度比较高的用户，也很简单，利用 sort 排个序，再选出前五个，下面为代码

```
def top_list():
    list = people_rating()
    items = list.items()
    top = [[v[1],v[0]] for v in items]
    top.sort(reverse=True)
    print(top[0:5])
```

下面为结果

可以看到第一名就是他自己，然后 547 和 384 也和他非常契合，不过 45 名的相似度就下降的非常的快，现在我们来检视 547 跟 384 的菜单吧

```
!!!number547!!! {'328': 4.0, '316': 5.0, '301': 3.0, '354': 4.0, '311': 2.0, '347': 4.0, '258': 4.0, '312': 4.0, '333':
4.0, '332': 3.0, '340': 4.0, '345': 5.0, '315': 4.0, '313': 5.0, '338': 2.0, '303': 3.0, '289': 3.0, '319': 4.0, '302':
5.0, '294': 1.0, '321': 4.0, '269': 3.0, '751': 4.0}
!!!number384!!! {'272': 5.0, '355': 4.0, '689': 4.0, '343': 3.0, '347': 4.0, '748': 4.0, '989': 4.0, '878': 4.0, '328':
4.0, '271': 4.0, '316': 5.0, '313': 5.0, '289': 5.0, '333': 4.0, '751': 4.0, '300': 4.0, '329': 3.0, '302': 5.0, '327':
4.0, '879': 4.0, '258': 4.0, '286': 4.0}
```

我们先来验证一下 547 跟 384 跟我们 7 号用户的相似度为什么这么高，下面代码可以找出 547 跟 7 号用户共同看过的电影跟评分

```
list = load_data()
for k,v in list['7'].items():
    for kk,vv in list['547'].items():
        if k == kk:
            print(k,v,kk,vv)
```

跑出来的结果如下

```
269 3.0 269 3.0
294 1.0 294 1.0
258 4.0 258 4.0
```

原来他们只共同看了三部电影，给的评分还一样，所以距离才为 0

再用同样方法来看 384 跟 7 号

```
300 4.0 300 4.0
258 4.0 258 4.0
286 4.0 286 4.0
```

可以看到他们也是刚好看了 3 部一样的电影，给的分数也是一样，有趣的事他们三个都看了 ID 为 258 的电影，也都给了 4 分

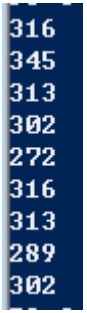
接下来就是最后一步了，我们要找出 547 和 384 看过但 7 号没看过的电影，再从里面找出评分高的推荐给 7 号，下面为代码

```
def find_rec():
    rec_list = top_list()
    first = rec_list[1][1]
    second = rec_list[2][1]
    all_list = load_data()
    for k,v in all_list[first].items():
```

```
        if k not in all_list['7'].keys() and v == 5:
            print (k)

    for k,v in all_list[second].items():
        if k not in all_list['7'].keys() and v == 5:
            print (k)
```

最后跑出来的结果为



以上为 547 跟 384 的推荐名单，可以看到 316/302/313 都是两人共同推荐，代表这三部片应该是很好看，所以如果要推荐给 7 号用户的话可以选择这三片来推荐。

四、总结

在演练的过程里面我们可以看出这个算法的许多缺点，例如最高分的其实是因为他们共同看过的电影少，分数又刚好相同，很难说明这就是有共同的兴趣，然后相似度的落差太大，前两名都是 1,三四名就掉到了 0.75，可见三四名应该是与 7 号共同看过 4 部电影，但有其中一部的评分差了一分，导致分数骤降，而大部分的用户都集中在 0.3-0,5 之间，分数曲线极度不平滑。虽然有这些缺点，但这些算出来的推荐结果还是有代表了一定的意义，至少可以代表是与 7 号用户品味相似的人给出的高分电影，而 7 号尚未给过评分。

用 python 做推荐系统（二）

一、简介

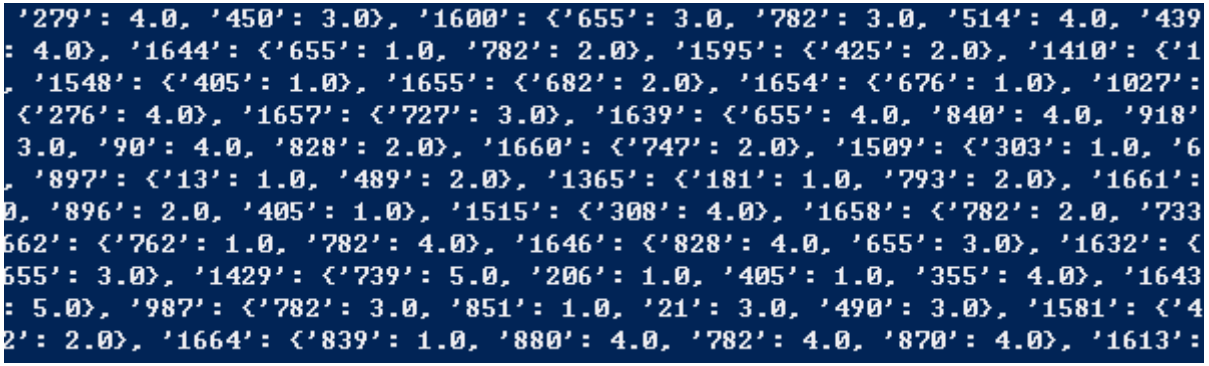
继上一篇基于用户的推荐算法，这一篇是要基于商品的，基于用户的好处是可以根据用户的评价记录找出跟他兴趣相似的用户，再推荐这些用户也喜欢的电影，但是万一这个用户是新用户呢？或是他还没有对任何电影做评价，那我们要怎么去推荐他可能会有兴趣的东西呢？这边就是要介绍基于商品的相似度，我们打开豆瓣随便查看一部电影，会看到下面有一个栏位是喜欢这部电影的人也喜欢哪些电影，就是利用了商品相似度的概念。商品相似度还有一个好处，就是可以"事先"计算好，由于商品相似度每个用户看到的结果都会是一样的，他可以事先就先算好放在那，等有一批新商品进入时再计算，比较不需要为每个用户都计算一遍，这是他的一个很大的优势。原理也很简单，就是找出喜欢这个电影的用户，他们也喜欢哪些电影，下面就是利用 pyhton 来做示范。

二、数据预处理

这次我们还是沿用之前在 movielens 下载的数据，但由于我们的“目标”变了，所以数据预处理的方式也要做些调整，之前我们是以人为键值（key），后面跟了他评价的电影和评分，现在我们要改成以电影为键值，后面跟了评价他的人和给出的评分，这样做是方便到时候算法代码比较好写，下面是数据读取和预处理的代码。

```
def load_data():
    f = open('u.data')
    movie_list={}
    for line in f:
        (user,movie,rating,ts) = line.split('\t')
        movie_list.setdefault(movie,{})
        movie_list[movie][user] = float(rating)
    return movie_list
```

用 print 查看 movie_list 的样子



可以看到，与前面以人为主的相比，很多电影的评分人数都只有一个，所以数据量的不足也会影响到最后算出来的结果。

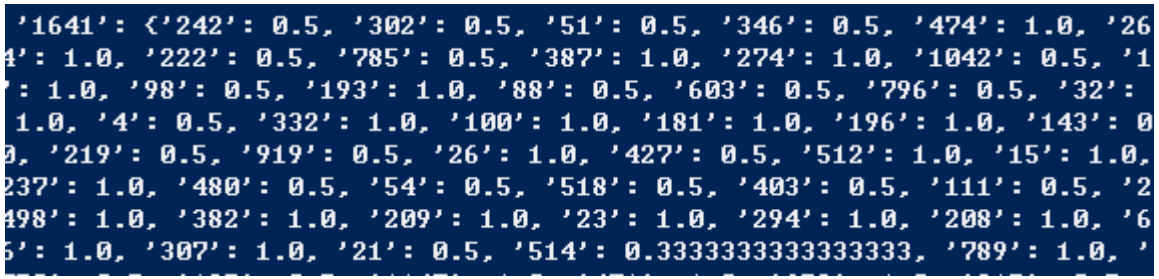
三、数据分析

这边跟前面以人为主的推荐有点不一样，上一篇我挑了 7 号用户作为我们的推荐对象，但商品我要对‘’所有商品‘’都找出他们的相似商品，计算量就会大很多，下面为代码

```
def calculate():
    list = load_data()
    movie_diff = {}
    for movie1 in list.keys():
        movie_diff.setdefault(movie1,{})
        for movie2 in list.keys():
            if movie1 != movie2:
                a = 0
```

```
                b=0
                for name1 in list[movie1].keys():
                    for name2 in list[movie2].keys():
                        if name1 == name2:
                            diff = sqrt(pow(list[movie1][name1] -list[movie2][name2],2))
                            b += 1
                            a += diff
                if b != 0:
                    movie_diff[movie1][movie2] = 1/(1+(a/b))
    print(movie_diff)
```

这次跑的时间长了很多，因为要拿所有电影跟其他所有电影进行比较，而且是要比评价电影的所有人，所以计算量大很多，以下是跑完出来的结果，可以看到其实很多的相关性都是 1，代表其实由于数据量太少，所以出来的结果参考价值并不大。



四、后续改进

以下是经过这两次的演练的心得

1、欧几里得虽然简单快速，但他出来的结果并不好，下次可以试试其他的算法

2、for 循环可以用矩阵来代替，效率会更好

3、可以尝试跑数据量大数据，出来的效果会比较好