



01 推荐系统简介

原创

da_journeyer

已于 2023-03-08 17:45:33 修改

阅读量 1.3k

收藏 15

点赞数 5

版权

分类专栏:

推荐系统

文章标签:

搜索引擎

推荐算法



推荐系统 专栏收录该内容

0 订阅 9 篇文章

订阅专栏

01 推荐系统简介

个性化推荐(推荐系统)经历了多年的发展，已经成为互联网产品的标配，也是AI成功落地的分支之一，在电商(淘宝/京东)、资讯([今日头条](#) /微博)、音乐(网易云音乐/QQ音乐)、短视频(抖音/快手)等热门应用中，推荐系统都是核心组件之一。

推荐系统产生背景

- 信息过载 & 用户需求不明确
 - 分类目录 (1990s) : 覆盖少量热门网站。Hao123 Yahoo
 - 搜索引擎 (2000s) : 通过搜索词明确需求。Google Baidu
 - 推荐系统 (2010s) : 不需要用户提供明确的需求，通过分析用户的历史行为给用户的兴趣进行建模，从而主动给用户推荐能够满足他们兴趣和需求的信息。

什么是推荐系统

- 没有明确需求的用户访问了我们的服务，且服务的物品对用户构成了信息过载，**系统通过一定的规则对物品进行排序，并将排在前面的物品展示给用户，这样的系统就是推荐系统**

推荐系统 V.S. 搜索引擎

	搜索	推荐
行为方式	主动	被动
意图	明确	模糊
个性化	弱	强
流量分布	马太效应	长尾效应
目标	快速满足	持续服务
评估指标	简明	复杂

- 推荐个性化较强，用户被动的接受，希望能够提供持续的服务
- 搜索个性化弱，用户主动搜索，快速满足用户的需求

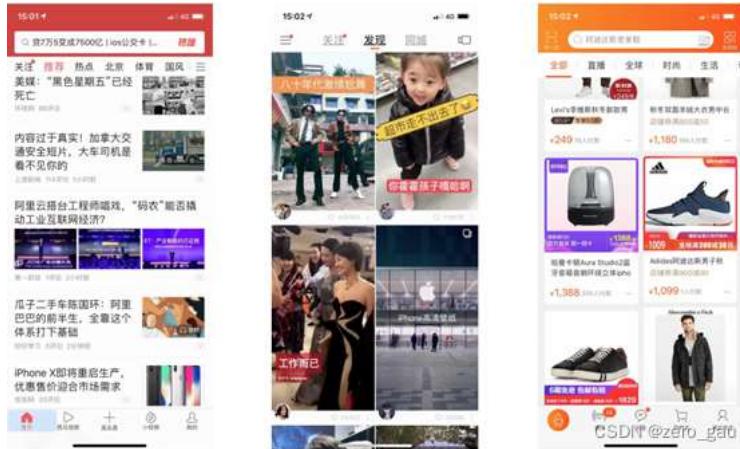
推荐系统的作用

- 高效连接用户和物品，发现长尾商品**
- 留住用户和内容生产者，实现商业目标**

推荐系统的工作原理

- 社会化推荐** 向朋友咨询，社会化推荐，让好友给自己推荐物品
- 基于内容的推荐** 打开搜索引擎，输入自己喜欢的演员的名字，然后看看返回结果中还有什么电影是自己没看过的
- 基于流行度的推荐** 查看票房排行榜，
- 基于协同过滤的推荐** 找到和自己历史兴趣相似的用户，看看他们最近在看什么电影





推荐系统和Web项目区别

- 稳定的信息流通系统 V.S. 通过信息过滤实现目标提升
 - web项目: 处理复杂逻辑, 处理高并发, 实现高可用, 为用户提供稳定服务, **构建一个稳定的信息流通的服务**
 - 推荐系统: 追求指标增长, 留存率/阅读时间/GMV (Gross Merchandise Volume电商网站成交金额)/视频网站VV (Video View)
- 确定 V.S. 不确定思维
 - web项目: 对结果有确定预期
 - 推荐系统: 结果是概率问题

02 推荐系统设计

2.1 推荐系统要素

- UI 和 UE(前端界面)
- 数据 (Lambda架构)
- 业务知识
- 算法

2.2 推荐系统架构

推荐系统整体架构





大数据Lambda架构

由Twitter工程师Nathan Marz(storm项目发起人)提出

Lambda系统架构提供了一个结合实时数据和Hadoop预先计算的数据环境和混合平台, 提供一个实时的数据视图

分层架构

- 批处理层(离线)

- 数据不可变, 可进行任何计算, 可水平扩展
- 高延迟 几分钟~几小时(计算量和数据量不同)
- 日志收集 Flume
- 分布式存储 Hadoop hdfs
- 分布式计算 Hadoop MapReduce & spark
- 视图存储数据库
 - nosql(HBase/Cassandra)
 - Redis/memcache
 - MySQL

- 实时处理层

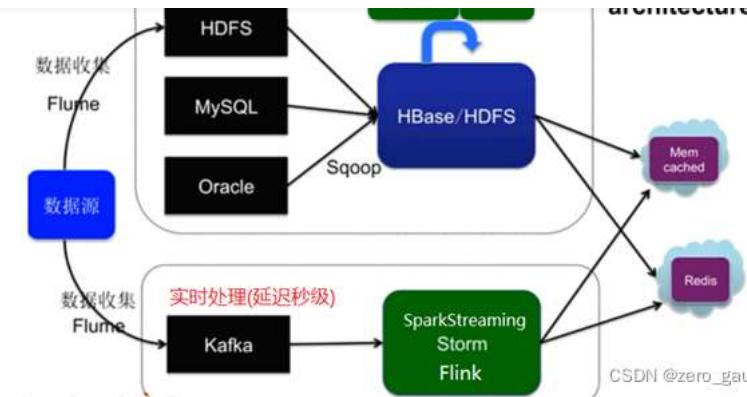
- 流式处理, 持续计算
- 存储和分析某个窗口期内的数据
- 最终正确性(Eventual accuracy)
- 实时数据收集 flume & kafka
- 实时数据分析 spark streaming/storm/flink

- 服务层

- 支持随机读
- 需要在非常短的时间内返回结果
- 读取批处理层和实时处理层结果并对其归并

Lambda架构图





- 离线计算和实时计算共同提供服务的问题
- 离线计算优缺点
 - 优点能够处理的数据量可以很大 比如pb级别
 - 缺点速度比较慢 分钟级别的延迟
- 实时计算
 - 优点响应快 来一条数据处理一条 ms级别响应
 - 缺点处理的数据量小一些
- 离线计算的框架
 - hadoop hdfs mapreduce
 - spark core , spark sql
 - hive
- 实时计算框架
 - spark streaming
 - storm
 - flink
- 消息中间件
 - flume 日志采集系统
 - kafka 消息队列
- 存储相关
 - hbase nosql数据库
 - hive sq操作hdfs数据

推荐算法架构

召回阶段(海选)

- 召回决定了最终推荐结果的天花板
- 常用算法:
 - 协同过滤(基于用户 基于物品的)
 - 基于内容 (根据用户行为总结出自己的偏好 根据偏好 通过文本挖掘技术找到内容上相似的商品)
 - 基于隐语义

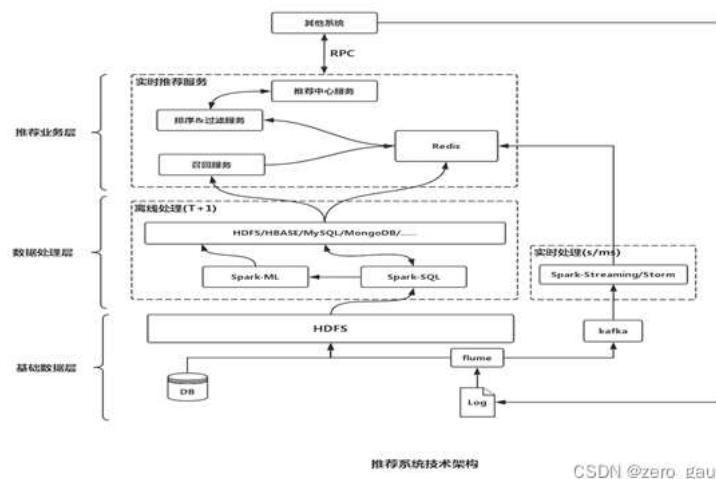
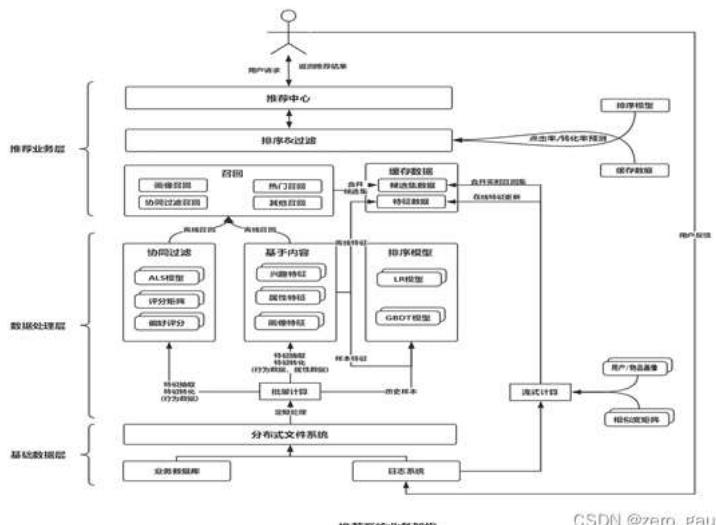
排序阶段

- 召回决定了最终推荐结果的天花板, 排序逼近这个极限, 决定了最终的推荐效果
- CTR预估 (点击率预估 使用LR算法) 估计用户是否会点击某个商品 需要用户的点击数据

策略调整



推荐系统的整体架构



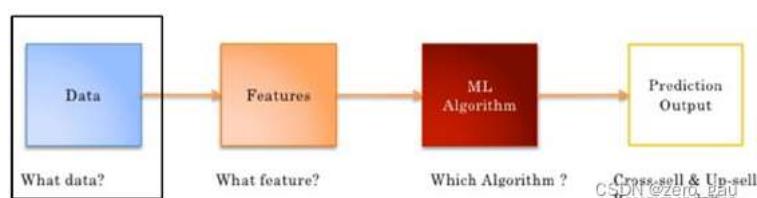
03 推荐算法

- 推荐模型构建流程
- 推荐算法概述
- 基于协同过滤的推荐算法
- 协同过滤实现

一 推荐模型构建流程

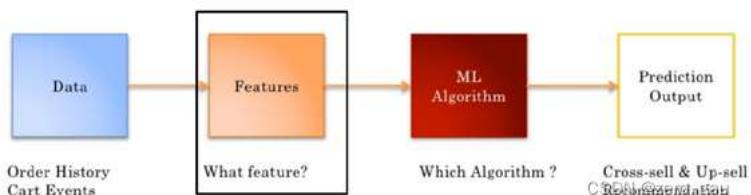
Data(数据)->Features(特征)->ML Algorithm(机器学习算法)->Prediction Output(预测输出)

数据清洗/数据处理



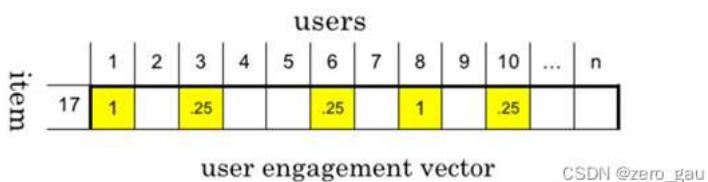
- 显性数据
 - Rating 打分
 - Comments 评论/评价
- 隐形数据
 - Order history 历史订单
 - Cart events 加购物车
 - Page views 页面浏览
 - Click-thru 点击
 - Search log 搜索记录
- 数据量/数据能否满足要求

特征工程



从数据中筛选特征

- 一个给定的商品，可能被拥有类似品味或需求的用户购买
- 使用用户行为数据描述商品

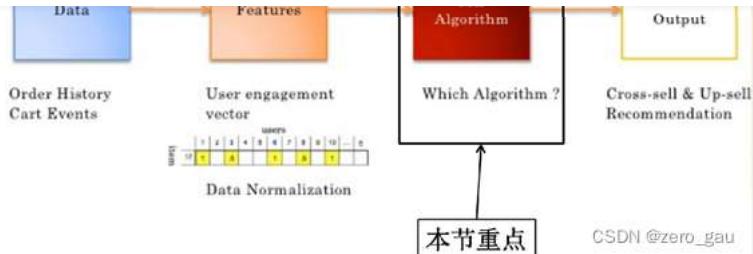


用数据表示特征

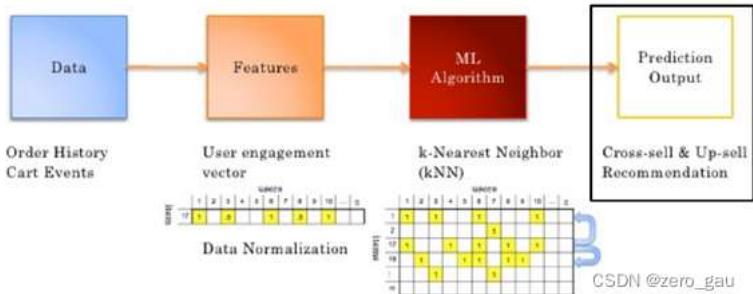
- 将所有用户行为合并在一起，形成一个user-item 矩阵

This diagram shows a user-item matrix. The columns are labeled 'users' (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ..., n) and the rows are labeled 'items' (1, 2, 3, 4, ..., m). Non-zero values are highlighted in yellow. The matrix is filled with values 1, .25, and some missing entries. The caption below reads 'CSDN @zero_gau'.

选择合适的算法



- 产生推荐结果



评估、模型上线

二 最经典的推荐算法：协同过滤推荐算法 (Collaborative Filtering)

算法思想：物以类聚，人以群分

基本的协同过滤推荐算法基于以下假设：

- “跟你喜好相似的人喜欢的东西你也很有可能喜欢”：基于用户的协同过滤推荐 (User-based CF)
- “跟你喜欢的东西相似的东西你也很有可能喜欢”：基于物品的协同过滤推荐 (Item-based CF)

实现协同过滤推荐有以下几个步骤：

1. 找出最相似的人或物品：TOP-N相似的人或物品

通过计算两两的相似度来进行排序，即可找出TOP-N相似的人或物品

2. 根据相似的人或物品产生推荐结果

利用TOP-N结果生成初始推荐结果，然后过滤掉用户已经有过记录的物品或明确表示不感兴趣的物品

以下是一个简单的示例，数据集相当于一个用户对物品的购买记录表：打勾表示用户对物品的有购买记录

- 关于相似度计算这里先用一个简单的思想：如有两个同学X和Y，X同学爱好[足球、篮球、乒乓球]，Y同学爱好[网球、足球、篮球、羽毛球]，可见他们的共同爱好有2个，那么他们的相似度可以用： $2/3 * 2/4 = 1/3 \approx 0.33$ 来表示。

User-Based CF



数据集	用户3	✓		✓		
相似度计算过程	用户4		✓	<th>✓</th> <th>✓</th>	✓	✓
	用户5	✓	✓	✓	✓	✓
	用户相似度计算1	用户1	用户2	用户3	用户4	用户5
	用户1	1	0.67*0.67	1*0.67	0.33*0.33	0.67*0.5
	用户2	0.67*0.67	1	0.33*0.5	0.67*0.67	0.67*0.5
相似度计算结果	用户3	0.67*1	0.33*0.5	1	0*0	1*0.5
	用户4	0.33*0.33	0.67*0.67	0*0	1	0.67*0.5
	用户5	0.67*0.5	0.67*0.5	1*0.5	0.67*0.5	1
	用户相似度计算2	用户1	用户2	用户3	用户4	用户5
	用户1	0.44		0.17	0.44	0.33
根据TOP-N相似用户的喜好进行推荐，并过滤	用户2	0.44		0.17	0	0.5
	用户3	0.67	0.17	0		0.33
	用户4	0.11	0.44	0		0.33
	用户5	0.33	0.33	0.5	0.33	
	推荐结果	TOP-2相似的用户	初始推荐结果	需要过滤的物品	最终推荐结果	
根据TOP-N相似用户的喜好进行推荐，并过滤	用户1	用户3、2	AC+ADE	ACD	E	
	用户2	用户1、4	ACD+BDE	ADE	BC	
	用户3	用户1、5	ACD+ABCE	AC	BDE	
	用户4	用户2、5	ADE+ABCE	BDE	AC	CSDN @zero_gau
	用户5	用户3、2	AC+ADE	ABCE	D	

Item-Based CF

数据集	用户/物品	物品A	物品B	物品C	物品D	物品E
相似度计算过程	用户1	✓		✓	✓	
	用户2	✓			✓	✓
	用户3	✓		✓		
	用户4		✓		✓	✓
	用户5	✓	✓	✓		✓
相似度计算结果	物品相似度计算1	物品A	物品B	物品C	物品D	物品E
	物品A	1	0.25*0.5	0.75*1	0.5*0.67	0.5*0.67
	物品B	0.25*0.5	1	0.5*0.33	0.5*0.33	1*0.67
	物品C	0.75*1	0.5*0.33	1	0.33*0.33	0.33*0.33
	物品D	0.5*0.67	0.5*0.33	0.33*0.33	1	0.67*0.67
得出TOP-N相似物品	物品相似度计算2	物品A	物品B	物品C	物品D	物品E
	物品A		0.125	0.75	0.33	0.33
	物品B	0.125		0.17	0.17	0.67
	物品C	0.75	0.17		0.11	0.11
	物品D	0.33	0.17	0.11		0.44
根据每个用户自身喜好找到相似物品进行推荐并过滤	TOP-N相似物品	TOP-2相似的物品				
	物品A	物品C、E				
	物品B	物品E、D				
	物品C	物品A、B				
	物品D	物品E、A				
	物品E	物品B、D				
根据每个用户自身喜好找到相似物品进行推荐并过滤	推荐结果	已经喜欢的物品	初始推荐结果	需要过滤的物品	最终推荐结果	
	用户1	ACD	CE+AB+EA	ACD	BE	
	用户2	ADE	CE+EA+BD	ADE	BC	
	用户3	AC	CE+AB	AC	BE	
	用户4	BDE	ED+EA+BD	BDE	A	CSDN @zero_gau
	用户5	ABCE	CE+ED+AB+BD	ABCE	D	

通过前面两个demo，相信大家应该已经对协同过滤推荐算法的设计与实现有了比较清晰的认识。

协同过滤思路介绍

- CF 物以类聚人以群分
- 做协同过滤，首先特征工程创建用户-物品的评分矩阵。
- 基于用户的协同过滤
 - 给用户A 找到最相似的N个用户
 - N个用户消费过哪些物品
 - N个用户消费过的物品中-A用户消费过的就是推荐结果
- 基于物品的协同过滤
 - 给物品A 找到最相似的N个物品
 - A用户消费记录 找到这些物品的相似物品
 - 从这些相似物品先去重-A用户消费过的就是推荐结果

三 相似度计算(Similarity Calculation)

- 相似度的计算方法

- 数据分类

- 布尔值(用户的行为 是否点击 是否收藏)
- 欧氏距离，是一个欧式空间下度量距离的方法。两个物体，都在同一个空间下表示为两个点，假如叫做 p, q ，分别都是 n 个坐标，那么欧式距离就是衡量这两个点之间的距离。欧式距离不适用于布尔向量之间

$$E(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

欧式距离的值是一个非负数，最大值正无穷，通常计算相似度的结果希望是[-1, 1]或[0, 1]之间，一般可以使用如下转化公式：

$$\frac{1}{1 + E(p, q)}$$

杰卡德相似度&余弦相似度&皮尔逊相关系数

余弦相似度

- 度量的是两个向量之间的夹角，用夹角的余弦值来度量相似的情况
- 两个向量的夹角为0度，余弦值为1，当夹角为90度是余弦值为0，为180度是余弦值为-1
- 余弦相似度在度量文本相似度，用户相似度 物品相似度的时候较为常用
- 余弦相似度的特点，与向量长度无关，余弦相似度计算要对向量长度归一化，两个向量只要方向一致，无论程度强弱，都可以视为“相似”

皮尔逊相关系数Pearson

- 实际上也是一种余弦相似度，不过先对向量做了中心化，向量 a, b 各自减去向量的均值后，再计算余弦相似度
- 皮尔逊相似度计算结果在[-1, 1]之间。-1表示负相关，1表示正相关
- 度量两个变量是不是同增同减
- 皮尔逊相关系数度量的是两个变量的变化趋势是否一致，不适合计算布尔值向量之间的相关度

杰卡德相似度 Jaccard

- 两个集合的交集元素个数在并集中所占的比例，非常适用于布尔向量表示
- 分子是两个布尔向量做点积计算，得到的就是交集元素的个数
- 分母是两个布尔向量做或运算，再求元素和

余弦相似度适合用户评分数据(实数值)，杰卡德相似度适用于隐式反馈数据(0, 1布尔值)(是否收藏，是否点击，是否加购物车)

items	2	1			.5		1	
	4	1	.5	1	1			

- Jaccard coefficient:

$$\text{sim}(a,b) = \frac{(1+1)}{(1+1+1)+(1+1+1+1)-(1+1)}$$

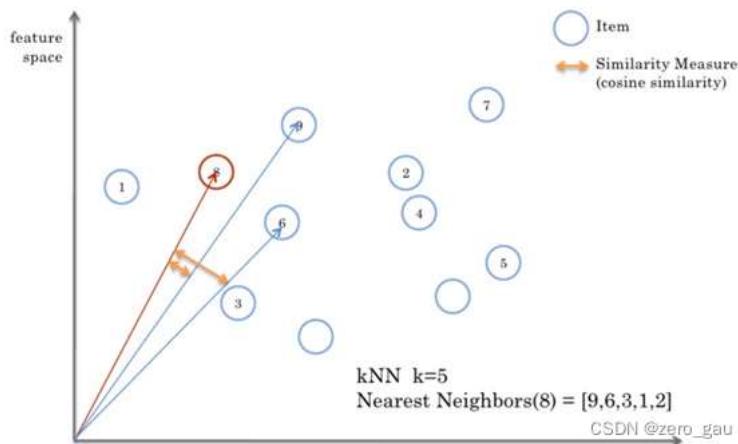
- Cosine similarity:

$$\text{sim}(a,b) = \cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|_2 * \|\vec{b}\|_2} = \frac{(1*1 + 0.5*1)}{\sqrt{(1^2 + 0.5^2 + 1^2) * (1^2 + 0.5^2 + 1^2)}}$$

- Pearson Correlation:

$$\begin{aligned} \text{corr}(a,b) &= \frac{\sum_i (r_{ai} - \bar{r}_a)(r_{bi} - \bar{r}_b)}{\sqrt{\sum_i (r_{ai} - \bar{r}_a)^2} \sqrt{\sum_i (r_{bi} - \bar{r}_b)^2}} = \frac{m \sum a_i b_i - \sum a_i \sum b_i}{\sqrt{m \sum a_i^2 - (\sum a_i)^2} \sqrt{m \sum b_i^2 - (\sum b_i)^2}} \\ &= \frac{\text{match_cols} * \text{Dotprod}(a,b) - \text{sum}(a) * \text{sum}(b)}{\sqrt{\text{match_cols} * \text{sum}(a^2) - (\text{sum}(a))^2} \sqrt{\text{match_cols} * \text{sum}(b^2) - (\text{sum}(b))^2}} \end{aligned}$$

- 余弦相似度



皮尔逊相关系数

Pearson's Correlation

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

用户相似度计算案例

1. 计算出用户1和其它用户之间的相似度

	1	2	3	4	5	6
1	0.25	0.577	0.866	0.289	0.577	
2	0.25	0.577	0	0.577	0.577	
3	0.577	0.577		0.333	0	0.333
4	0.866	0	0.333		0.333	0.333
5	0.289	0.577	0	0.333		0.667
6	0.577	0.577	0.333	0.333	0.667	

CSDN @zero_gau

2.按照相似度大小排序，K近邻 如K取4:

	1	2	3	4	5	6
1	0.25	0.577	0.866	0.289	0.577	
2	0.25		0.577	0	0.577	0.577
3	0.577	0.577		0.333	0	0.333
4	0.866	0	0.333		0.333	0.333
5	0.289	0.577	0	0.333		0.667
6	0.577	0.577	0.333	0.333	0.667	

CSDN @zero_gau

K = 4

~ Only get 4 most similar users (nearest neighbor)



0.866

0.577

0.577

0.289

0.25



CSDN @zero_gau

3.取出近邻用户的购物清单

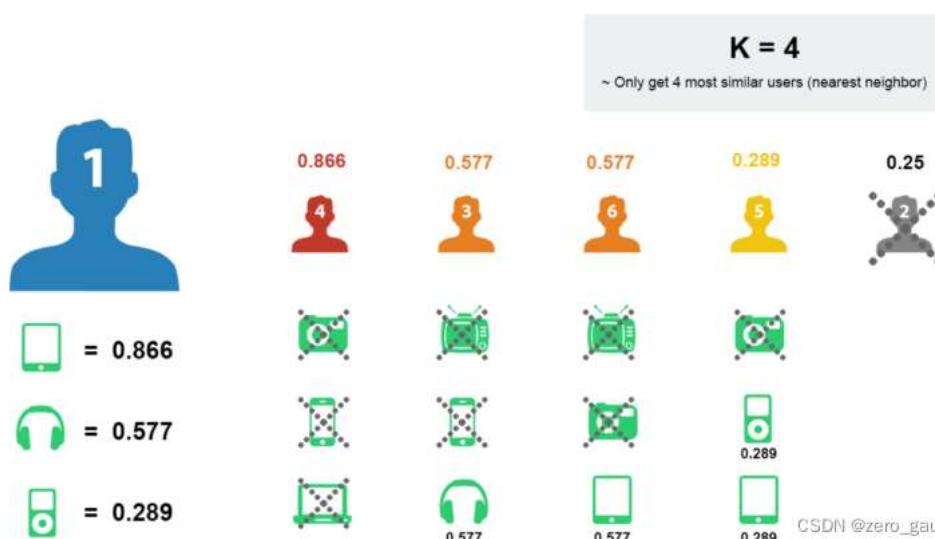




4.去除用户1已经购买过的商品



5.在剩余的物品中根据评分排序



6.物品相似度计算

- 余弦相似度对绝对值大小不敏感带来的问题

- 可以采用改进的余弦相似度，先计算向量每个维度上的均值，然后每个向量在各个维度上都减去均值后，在计算余弦相似度，用调整的余弦相似度计算得到的相似度是-0.1

Adjusted Cosine Similarity

$$s(i, j) = \frac{\sum_{u \in U} (R_{u,j} - \bar{R}_u)(R_{u,i} - \bar{R}_u)}{\sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_u)^2} \sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_u)^2}}$$

CSDN @zero_gau

物品相似度计算案例

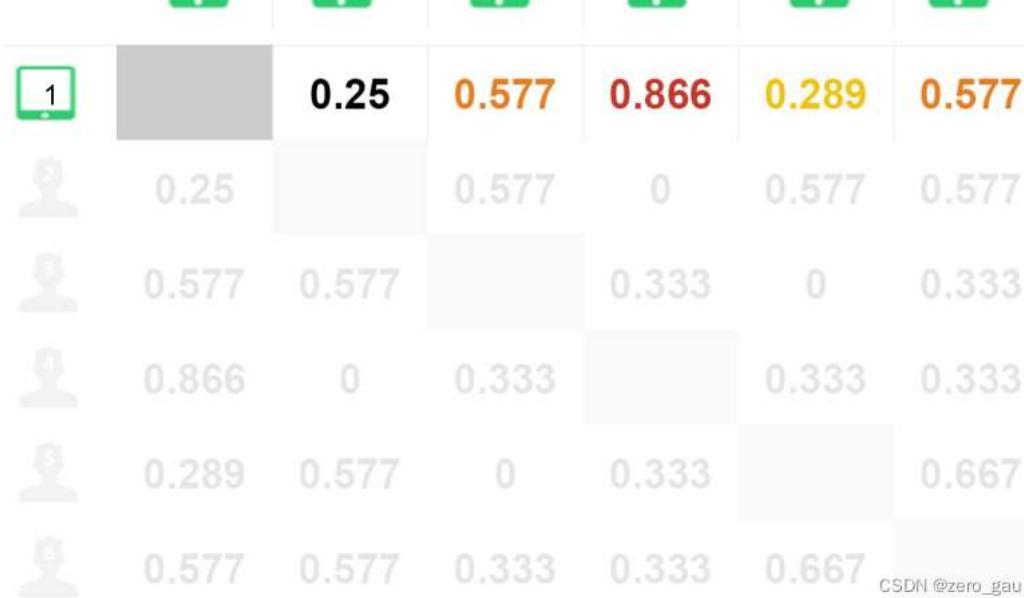
1.计算物品与物品间的相似度

	1	2	3	4	5	6
1	0.25	0.577	0.866	0.289	0.577	
2	0.25		0.577	0	0.577	0.577
3	0.577	0.577		0.333	0	0.333
4	0.866	0	0.333		0.333	0.333
5	0.289	0.577	0	0.333		0.667
6	0.577	0.577	0.333	0.333	0.667	

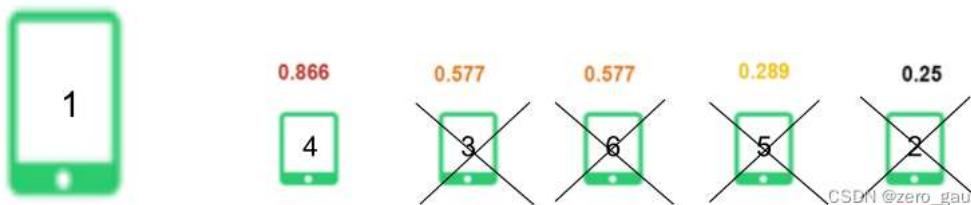
CSDN @zero_gau

2.找出物品1的相似商品





3.选择最近似的物品



4.基于用户与物品的协同过滤比较

用户/物品	物品A	物品B	物品C	物品D
用户A	✓		✓	推荐
用户B		✓		
用户C	✓		✓	✓



用户/物品	物品A	物品B	物品C
用户A	✓		✓
用户B	✓	✓	✓
用户C	✓		推荐



协同过滤推荐算法代码实现

- 构建数据集

```

1 users = ["User1", "User2", "User3", "User4", "User5"]
2 items = ["Item A", "Item B", "Item C", "Item D", "Item E"]
3 # 构建数据集
4 datasets = [
5     ["buy", None, "buy", "buy", None],
6     ["buy", None, None, "buy", "buy"],
7     ["buy", None, "buy", None, None],
8     [None, "buy", None, "buy", "buy"],
9     ["buy", "buy", "buy", None, "buy"],
10 ]

```



ハノ、カジルリコト、ソノカタノヘバノハコト、ハシヒロタツテヨクタツムロ、アリカヒリヒンヌスルホタツテハツカヘスルナリ。

```

1 users = ["User1", "User2", "User3", "User4", "User5"]
2 items = ["Item A", "Item B", "Item C", "Item D", "Item E"]
3 # 用户购买记录数据集
4 datasets = [
5     [1, 0, 1, 1, 0],
6     [1, 0, 0, 1, 1],
7     [1, 0, 1, 0, 0],
8     [0, 1, 0, 1, 1],
9     [1, 1, 0, 1, 1],
10 ]
11
12 import pandas as pd
13
14 df = pd.DataFrame(datasets, columns=items, index=users)
15 print(df)
16 #      Item A  Item B  Item C  Item D  Item E
17 # User1      1      0      1      1      0
18 # User2      1      0      0      1      1
19 # User3      1      0      1      0      0
20 # User4      0      1      0      1      1
21 # User5      1      1      1      0      1

```

- 有了数据集，接下来我们就可以进行相似度的计算，不过对于相似度的计算其实有很多专门的相似度计算方法的，比如余弦相似度、皮尔逊相关系数、杰卡德相似度等等。这里我们选择使用杰卡德相似系数 [0, 1]

```

1 # 直接计算某两项的杰卡德相似系数
2 import numpy as np
3 # from sklearn.metrics import jaccard_similarity_score
4 from sklearn.metrics import jaccard_score
5 # 计算Item A 和Item B的相似度
6 print(jaccard_score(df["Item A"], df["Item B"]))
7
8 # 计算所有的数据两两的杰卡德相似系数
9 from sklearn.metrics.pairwise import pairwise_distances
10 # pairwise_distances, 注意该方法返回的是余弦距离, 余弦距离= 1 - 余弦相似度
11 # 计算用户间相似度
12 user_similar = 1 - pairwise_distances(np.array(df), metric="jaccard")
13 user_similar = pd.DataFrame(user_similar, columns=users, index=users)
14 print("用户之间的两两相似度: ")
15 print(user_similar)
16
17 # 计算物品间相似度
18 item_similar = 1 - pairwise_distances(np.array(df.T), metric="jaccard")
19 item_similar = pd.DataFrame(item_similar, columns=items, index=items)
20 print("物品之间的两两相似度: ")
21 print(item_similar)
22
23 # 0.2
24 # 用户之间的两两相似度:
25 #      User1  User2  User3  User4  User5
26 # User1  1.000000  0.50  0.666667  0.2  0.4
27 # User2  0.500000  1.00  0.250000  0.5  0.4
28 # User3  0.666667  0.25  1.000000  0.0  0.5
29 # User4  0.200000  0.50  0.000000  1.0  0.4
30 # User5  0.400000  0.40  0.500000  0.4  1.0
31 # 物品之间的两两相似度:
32 #      Item A  Item B  Item C  Item D  Item E
33 # Item A  1.00  0.200000  0.75  0.40  0.400000
34 # Item B  0.20  1.000000  0.25  0.25  0.666667
35 # Item C  0.75  0.250000  1.00  0.20  0.200000
36 # Item D  0.40  0.250000  0.20  1.00  0.500000
37 # Item E  0.40  0.666667  0.20  0.50  1.000000

```



有了两两的相似度，接下来就可以筛选TOP-N相似结果，并进行推荐了

User-Based CF

```

1 import pandas as pd
2 import numpy as np
3 from pprint import pprint
4
5 users = ["User1", "User2", "User3", "User4", "User5"]
6 items = ["Item A", "Item B", "Item C", "Item D", "Item E"]
7 # 用户购买记录数据集
8 datasets = [
9     [1, 0, 1, 1, 0],
10    [1, 0, 0, 1, 1],
11    [1, 0, 1, 0, 0],
12    [0, 1, 0, 1, 1],
13    [1, 1, 1, 0, 1],
14 ]
15
16 df = pd.DataFrame(datasets, columns=items, index=users)
17 print('df: ')
18 print(df)
19 # 计算所有的数据两两的杰卡德相似系数
20 from sklearn.metrics.pairwise import pairwise_distances
21 # 计算用户间相似度
22 user_similar = 1 - pairwise_distances(np.array(df), metric="jaccard")
23 user_similar = pd.DataFrame(user_similar, columns=users, index=users)
24 print("用户之间的两两相似度: ")
25 print(user_similar)
26
27 topN_users = {}
28 # 遍历每一行数据
29 for i in user_similar.index:
30     # 取出每一列数据，并删除自身，然后排序数据
31     _df = user_similar.loc[i].drop([i])
32     _df_sorted = _df.sort_values(ascending=False)
33
34     top2 = list(_df_sorted.index[:2])
35     topN_users[i] = top2
36
37 print("Top2相似用户: ")
38 pprint(topN_users)
39
40 rs_results = {}
41 # 构建推荐结果
42 for user, sim_users in topN_users.items():
43     rs_result = set() # 存储推荐结果
44     for sim_user in sim_users:
45         # 构建初始的推荐结果
46         rs_result = rs_result.union(
47             set(df.loc[sim_user].replace(0, np.nan).dropna().index))
48     # 过滤掉已经购买过的物品
49     rs_result -= set(df.loc[user].replace(0, np.nan).dropna().index)
50     rs_results[user] = rs_result
51 print("最终推荐结果: ")
52 pprint(rs_results)
53
54 # df:
55 #      Item A  Item B  Item C  Item D  Item E
56 # User1      1      0      1      1      0
57 # User2      1      0      0      1      1
58 # User3      1      0      1      0      0
59 # User4      0      1      0      1      1
60 # User5      1      1      1      0      1
61 # 用户之间的两两相似度:
62 #          User1  User2  User3  User4  User5
63 # User1  1.000000  0.50  0.666667  0.2  0.4
64 # User2  0.500000  1.00  0.250000  0.5  0.4
65 # User3  0.666667  0.25  1.000000  0.0  0.5
66 # User4  0.200000  0.50  0.000000  1.0  0.4
67 # User5  0.400000  0.40  0.500000  0.4  1.0

```



```

70 # 'User2': ['User1', 'User4'],
71 # 'User3': ['User1', 'User5'],
72 # 'User4': ['User2', 'User5'],
73 # 'User5': ['User3', 'User1']}
74 # 最终推荐结果:
75 # {'User1': {'Item E'},
76 # 'User2': {'Item B', 'Item C'},
77 # 'User3': {'Item B', 'Item D', 'Item E'},
78 # 'User4': {'Item A', 'Item C'},
79 # 'User5': {'Item D'}}}

```

Item-Based CF

```

1 import pandas as pd
2 import numpy as np
3 from pprint import pprint
4
5 users = ["User1", "User2", "User3", "User4", "User5"]
6 items = ["Item A", "Item B", "Item C", "Item D", "Item E"]
7 # 用户购买记录数据集
8 datasets = [
9     [1, 0, 1, 1, 0],
10    [1, 0, 0, 1, 1],
11    [1, 0, 1, 0, 0],
12    [0, 1, 0, 1, 1],
13    [1, 1, 1, 0, 1],
14 ]
15
16 df = pd.DataFrame(datasets, columns=items, index=users)
17 print('df: ')
18 print(df)
19 # 计算所有的数据两两的杰卡德相似系数
20 from sklearn.metrics.pairwise import pairwise_distances
21 # 计算物品间相似度
22 item_similar = 1 - pairwise_distances(np.array(df.T), metric="jaccard")
23 item_similar = pd.DataFrame(item_similar, columns=items, index=items)
24 print("物品之间的两两相似度: ")
25 print(item_similar)
26
27 topN_items = {}
28 # 遍历每一行数据
29 for i in item_similar.index:
30     # 取出每一列数据，并删除自身，然后排序数据
31     _df = item_similar.loc[i].drop([i])
32     _df_sorted = _df.sort_values(ascending=False)
33
34     top2 = list(_df_sorted.index[:2])
35     topN_items[i] = top2
36
37 print("Top2相似物品: ")
38 pprint(topN_items)
39
40 rs_results = {}
41 # 构建推荐结果
42 for user in df.index: # 遍历所有用户
43     rs_result = set()
44     for item in df.loc[user].replace(0,
45                                         np.nan).dropna().index: # 取出每个用户当前已购物品列表
46         # 根据每个物品找出最相似的TOP-N物品，构建初始推荐结果
47         rs_result = rs_result.union(topN_items[item])
48     # 过滤掉用户已购的物品
49     rs_result -= set(df.loc[user].replace(0, np.nan).dropna().index)
50     # 添加到结果中
51     rs_results[user] = rs_result
52
53 print("最终推荐结果: ")
54 pprint(rs_results)
55 # df:
56 #      Item A  Item B  Item C  Item D  Item E

```



```

59 # User3      1      0      1      0      0
60 # User4      0      1      0      1      1
61 # User5      1      1      1      0      1
62 # 物品之间的两两相似度:
63 #       Item A   Item B   Item C   Item D   Item E
64 # Item A    1.00  0.200000  0.75  0.40  0.400000
65 # Item B    0.20  1.000000  0.25  0.25  0.666667
66 # Item C    0.75  0.250000  1.00  0.20  0.200000
67 # Item D    0.40  0.250000  0.20  1.00  0.500000
68 # Item E    0.40  0.666667  0.20  0.50  1.000000
69 # Top2相似物品:
70 # {'Item A': ['Item C', 'Item D'],
71 # 'Item B': ['Item E', 'Item C'],
72 # 'Item C': ['Item A', 'Item B'],
73 # 'Item D': ['Item E', 'Item A'],
74 # 'Item E': ['Item B', 'Item D']}
75 # 最终推荐结果:
76 # {'User1': {'Item B': 'Item E'},
77 # 'User2': {'Item C': 'Item B'},
78 # 'User3': {'Item B': 'Item D'},
79 # 'User4': {'Item C': 'Item A'},
80 # 'User5': {'Item D'}}}

```

关于协同过滤推荐算法使用的数据集

在前面的demo中，我们只是使用用户对物品的一个购买记录，类似也可以是比如浏览点击记录、收听记录等等。这样数据我们预测的结果其实相当于是在预测用户是否对某物品感兴趣，对于喜好程度不能很好的预测。

因此在协同过滤推荐算法中其实会更多的利用用户对物品的“评分”数据来进行预测，通过评分数据集，我们可以预测用户对于他没有评分过的物品的评分。其实现原理和思想都是一样的，只是使用的数据集是用户-物品的评分数据。

关于用户-物品评分矩阵

用户-物品的评分矩阵，根据评分矩阵的稀疏程度会有不同的解决方案

- 稠密评分矩阵

稠密评分数 据集	用户/物品	物品A	物品B	物品C	物品D	物品E
	用户1	5	3	4	4	?
	用户2	3	1	2	3	3
	用户3	4	3	4	3	5
	用户4	3	3	1	5	4
	用户5	1	5	5	2	1 _{CSDN @zero_gan}

- 稀疏评分矩阵

在这里插入图片描述

这里先介绍稠密评分矩阵的处理，稀疏矩阵的处理相对会复杂一些，我们到后面再来介绍。

使用协同过滤推荐算法对用户进行评分预测

数据集

稠密评分数 据集	用户/物品	物品A	物品B	物品C	物品D	物品E
	用户1	5	3	4	4	?
	用户2	3	1	2	3	3
	用户3	4	3	4	3	5
	用户4	3	3	1	5	4
	用户5	1	5	5	2	1 _{CSDN @zero_gan}

目的：预测用户1对物品E的评分

构建数据集

注意这里构建评分数据时，对于缺失的部分我们需要保留为None，如果设置为0那么会被当作评分值为0去对待

```

3 # 用户购买记录数据集
4 datasets = [
5     [5, 3, 4, 4, None],
6     [3, 1, 2, 3, 3],
7     [4, 3, 4, 3, 5],
8     [3, 3, 1, 5, 4],
9     [1, 5, 5, 2, 1],
10 ]

```

计算相似度

对于评分数据这里我们采用皮尔逊相关系数[-1, 1]来计算, -1表示强负相关, +1表示强正相关

pandas中corr方法可直接用于计算皮尔逊相关系数

```

1 df = pd.DataFrame(datasets, columns=items, index=users)
2 print('df ')
3 print(df)
4 print("用户之间的两两相似度: ")
5 # 直接计算皮尔逊相关系数
6 # 默认是按列进行计算, 因此如果计算用户间的相似度, 当前需要进行转置
7 user_similar = df.T.corr()
8 print(user_similar.round(4))
9
10 print("物品之间的两两相似度: ")
11 item_similar = df.corr()
12 print(item_similar.round(4))

```

```

1 # df
2 #      Item A  Item B  Item C  Item D  Item E
3 # User1      5      3      4      4    NaN
4 # User2      3      1      2      3   3.0
5 # User3      4      3      4      3   5.0
6 # User4      3      3      1      5   4.0
7 # User5      1      5      5      2   1.0
8 # 用户之间的两两相似度:
9 #      User1  User2  User3  User4  User5
10 # User1  1.0000  0.8528  0.7071  0.0000 -0.7921
11 # User2  0.8528  1.0000  0.4677  0.4900 -0.9001
12 # User3  0.7071  0.4677  1.0000 -0.1612 -0.4666
13 # User4  0.0000  0.4900 -0.1612  1.0000 -0.6415
14 # User5 -0.7921 -0.9001 -0.4666 -0.6415  1.0000
15 # 物品之间的两两相似度:
16 #      Item A  Item B  Item C  Item D  Item E
17 # Item A  1.0000 -0.4767 -0.1231  0.5322  0.9695
18 # Item B -0.4767  1.0000  0.6455 -0.3101 -0.4781
19 # Item C -0.1231  0.6455  1.0000 -0.7206 -0.4276
20 # Item D  0.5322 -0.3101 -0.7206  1.0000  0.5817
21 # Item E  0.9695 -0.4781 -0.4276  0.5817  1.0000

```

可以看到与用户1最相似的是用户2和用户3; 与物品A最相似的物品分别是物品E和物品D。

注意: **我们在预测评分时, 往往是通过与其有正相关的用户或物品进行预测, 如果不存在正相关的情况, 那么将无法做出预测。**这一点尤其是在稀疏评分矩阵中尤为常见, 因为稀疏评分矩阵中很难得出正相关系数。

评分预测

User-Based CF 评分预测: 使用用户间的相似度进行预测

关于评分预测的方法也有比较多的方案, 下面介绍一种效果比较好的方案, 该方案考虑了用户本身的评分以及近邻用户的加权平均相似度打分来进行预测:



$$pred(u, i) = r_{ui} - \frac{\sum_{v \in U} sim(u, v)}{\sum_{v \in U} |sim(u, v)|}$$

我们要预测用户1对物品E的评分，那么可以根据与**用户1最近邻的用户2和用户3**进行预测，计算如下：
用户1和用户2的相关性分别为 0.85, 0.71；用户2和用户3，对E喜好的值分别为 3, 5；

$$pred(u_1, i_5) = \frac{0.85 * 3 + 0.71 * 5}{0.85 + 0.71} = 3.91$$

最终预测出用户1对物品5的评分为3.91

Item-Based CF 评分预测：使用物品间的相似度进行预测

这里利用物品相似度预测的计算同上，同样考虑了用户自身的平均打分因素，结合预测物品与相似物品的加权平均相似度打分来进行预测

$$pred(u, i) = \hat{r}_{ui} = \frac{\sum_{j \in I_{rated}} sim(i, j) * r_{uj}}{\sum_{j \in I_{rated}} sim(i, j)}$$

我们要预测用户1对物品E的评分，那么可以根据与物品E最近邻的物品A和物品D进行预测，计算如下：
用户1对物品A和D的喜爱为 5, 4；物品E和物品A和物品D的相关性为0.97, 0.58

$$pred(u_1, i_5) = \frac{0.97 * 5 + 0.58 * 4}{0.97 + 0.58} = 4.63$$

对比可见，User-Based CF预测评分和Item-Based CF的评分结果也是存在差异的

因为严格意义上他们其实应当属于两种不同的推荐算法，各自在不同的领域不同场景下，都会比另一种的效果更佳，但具体哪一种更佳，必须经过合理的效果评估，因此在实现推荐系统时这两种算法往往都是需要去实现的，然后对产生的推荐效果进行评估分析选出更优方案。

基于模型的方法

思想

- 通过机器学习算法，在数据中找出模式，并将用户与物品间的互动方式模式化
- 基于模型的协同过滤方式是构建协同过滤更高级的算法

近邻模型的问题

- 物品之间存在相关性，信息量并不随着向量维度增加而线性增加
- 矩阵元素稀疏，计算结果不稳定，增减一个向量维度，导致近邻结果差异很大的情况存在

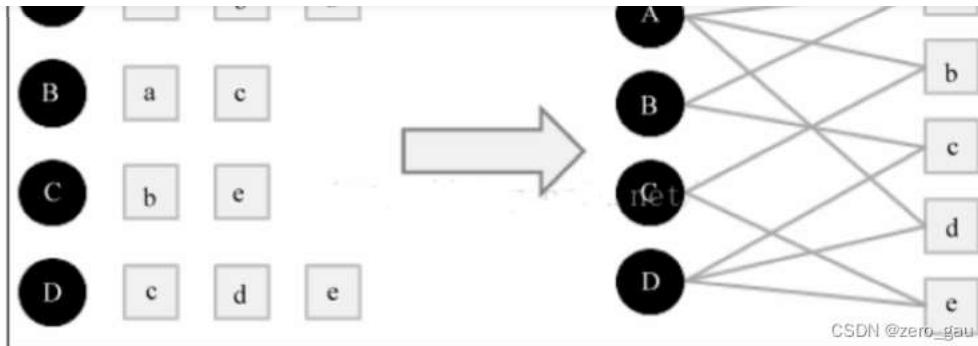
算法分类

- 基于图的模型
- 基于矩阵分解的方法**

基于图的模型

- 基于邻域的模型看做基于图的模型的简单形式





- 原理
 - 将用户的行为数据表示为二分图
 - 基于二分图为用户进行推荐
 - 根据两个顶点之间的路径数、路径长度和经过的顶点数来评价两个顶点的相关性

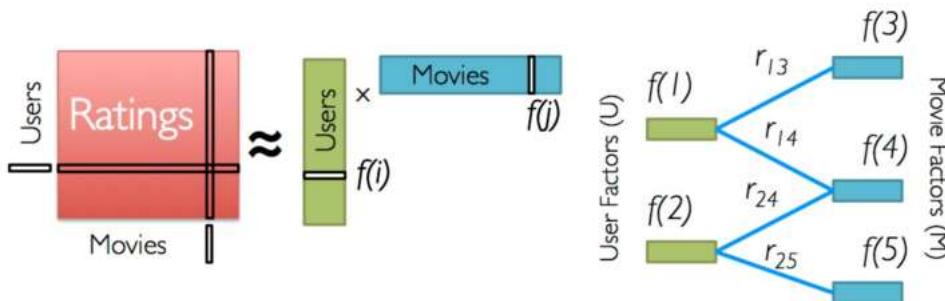
基于矩阵分解的模型

- 原理
 - 根据用户与物品的潜在表现，我们就可以预测用户对未评分的物品的喜爱程度
 - 把原来的大矩阵，近似分解成两个小矩阵的乘积，在实际推荐计算时不再使用大矩阵，而是使用分解得到的两个小矩阵
 - 用户-物品评分矩阵A是M × N维，即一共有M个用户，n个物品 我们选一个很小的数 K (K<< M, K<< N)
 - 通过计算得到两个矩阵U 和 V，U是M * K矩阵，矩阵V是N * K， $U_{m*k} V_{n*k}^T$ 约等于A_{m*n}
 - 类似这样的计算过程就是矩阵分解

• 基于矩阵分解的方法

- ALS交替最小二乘
 - ALS-WR(加权正则化交替最小二乘法): alternating-least-squares with weighted-λ -regularization
 - 将用户(user)对商品(item)的评分矩阵分解为两个矩阵：一个是用户对商品隐含特征的偏好矩阵，另一个是商品所包含的隐含特征的矩阵。在这个矩阵分解的过程中，评分缺失项得到了填充，也就是说我们可以基于这个填充的评分来给用户做商品推荐了。
- SVD奇异值分解矩阵
- ALS方法

Low-Rank Matrix Factorization:



Iterate:

$$f[i] = \arg \min_{w \in \mathbb{R}^d} \sum_{j \in \text{Nbrs}(i)} (r_{ij} - w^T f[j])^2 + \lambda \|w\|_2^2$$

CSDN @zero_gau

- 与传统的矩阵分解SVD方法来分解矩阵 $R(R \in \mathbb{R}^{m \times n})$ 不同的是，ALS(alternating least squares)希望找到两个低维矩阵，以 $\hat{R} = XY$ 来逼近矩阵 R ，其中， $X \in \mathbb{R}^{m \times d}$, $Y \in \mathbb{R}^{d \times n}$ ，这样，将问题的复杂度由 $O(m^*n)$ 转换为 $O((m+n)^*d)$ 。
- 计算 X 和 Y 过程：首先用一个小于1的随机数初始化 Y ，并根据公式求 X ，此时就可以得到初始的 XY 矩阵了，根据平方差和得到的 X ，重新计算并覆盖 Y ，计算差平方和，反复进行以上两步的计算，直到差平方和小于一个预设的数，或者迭代次数满足要求则停止。

04 案例—基于协同过滤的电影推荐

前面我们已经基本掌握了协同过滤推荐算法，以及其中两种最基本的实现方案：User-Based CF 和 Item-Based CF，下面我们将利用真是的数据来进行实战演练。

案例需求 演示效果

分析案例

数据集下载

MovieLens Latest Datasets Small

建议下载 [ml-latest-small.zip](#)，数据量小，便于我们单机使用和运行

目标：根据 [ml-latest-small/ratings.csv](#)（用户-电影评分数据），分别实现User-Based CF和Item-Based CF，并进行电影评分的预测，然后为用户实现电影推荐

数据集加载

加载ratings.csv，并转换为用户-电影评分矩阵

```

1 import os
2
3 import pandas as pd
4 import numpy as np
5
6 DATA_PATH = "./datasets/ml-latest-small/ratings.csv"
7 CACHE_DIR = "./datasets/cache/"
8
9 def load_data(data_path):
10     """
11         加载数据
12         :param data_path: 数据集路径
13         :param cache_path: 数据集缓存路径
14         :return: 用户-物品评分矩阵
15     """
16     # 数据集缓存地址
17     cache_path = os.path.join(CACHE_DIR, "ratings_matrix.cache")
18
19     print("开始加载数据集...")
20     if os.path.exists(cache_path):      # 判断是否存在缓存文件
21         print("加载缓存中...")
22         ratings_matrix = pd.read_pickle(cache_path)
23         print("从缓存加载数据集完毕")
24     else:
25         print("加载新数据中...")
26         # 设置要加载的数据字段的类型
27         dtype = {"userId": np.int32, "movieId": np.int32, "rating": np.float32}
28         # 加载数据，我们只用前三列数据，分别是用户ID，电影ID，已经用户对电影的对应评分
29         ratings = pd.read_csv(data_path, dtype=dtype, usecols=range(3))
30         # 透视表，将电影ID转换为列名称，转换成为一个User-Movie的评分矩阵
31         ratings_matrix = ratings.pivot_table(index=["userId"], columns=["movieId"], values="rating")
32         # 存入缓存文件
33         ratings_matrix.to_pickle(cache_path)
34         print("数据集加载完毕")
35     return ratings_matrix
36
37 if __name__ == '__main__':
38
39

```



相似度计算

计算用户或物品两两相似度：

```

1 def compute_pearson_similarity(ratings_matrix, based="user"):
2     ...
3     # 计算皮尔逊相关系数
4     :param ratings_matrix: 用户-物品评分矩阵
5     :param based: "user" or "item"
6     :return: 相似度矩阵
7     ...
8     user_similarity_cache_path = os.path.join(CACHE_DIR, "user_similarity.cache")
9     item_similarity_cache_path = os.path.join(CACHE_DIR, "item_similarity.cache")
10    # 基于皮尔逊相关系数计算相似度
11    # 用户相似度
12    if based == "user":
13        if os.path.exists(user_similarity_cache_path):
14            print("正从缓存加载用户相似度矩阵")
15            similarity = pd.read_pickle(user_similarity_cache_path)
16        else:
17            print("开始计算用户相似度矩阵")
18            similarity = ratings_matrix.T.corr()
19            similarity.to_pickle(user_similarity_cache_path)
20
21    elif based == "item":
22        if os.path.exists(item_similarity_cache_path):
23            print("正从缓存加载物品相似度矩阵")
24            similarity = pd.read_pickle(item_similarity_cache_path)
25        else:
26            print("开始计算物品相似度矩阵")
27            similarity = ratings_matrix.corr()
28            similarity.to_pickle(item_similarity_cache_path)
29    else:
30        raise Exception("Unhandled 'based' Value: %s" % based)
31    print("相似度矩阵计算/加载完毕")
32    return similarity
33
34 if __name__ == '__main__':
35
36     ratings_matrix = load_data(DATA_PATH)
37
38     user_similar = compute_pearson_similarity(ratings_matrix, based="user")
39     print(user_similar)
40     item_similar = compute_pearson_similarity(ratings_matrix, based="item")
41     print(item_similar)

```

注意

以上实现，仅用于实验阶段，因为工业上、或生产环境中，数据量是远超过我们本例中使用的数据量的，而pandas是无法支撑起大批量数据的运算的，因此工业上通常会使用spark、mapReduce等分布式计算框架来实现，我们后面的课程中也是建立在此基础上进行实践的。

但是正如前面所说，推荐算法的思想和理念都是统一的，不论使用什么平台工具、有多大的数据体量，其背后的实现原理都是不变的。

所以在本节，大家要深刻去学习的是推荐算法的业务流程，以及在具体的业务场景中，如本例的电影推荐，如何实现出推荐算法，并产生推荐结果。

05 案例-算法实现：User-Based CF 预测评分

评分预测公式：

$$pred(u, i) = \hat{r}_{ui} = \frac{\sum_{v \in U} sim(u, v) * r_{vi}}{\sum_{v \in U} |sim(u, v)|}$$



- 实现评分预测方法: `predict`

```

1 def predict(uid, iid, ratings_matrix, user_similar):
2     ...
3     预测给定用户对给定物品的评分值
4     :param uid: 用户ID
5     :param iid: 物品ID
6     :param ratings_matrix: 用户-物品评分矩阵
7     :param user_similar: 用户两两相似度矩阵
8     :return: 预测的评分值
9     ...
10    print("开始预测用户<%d>对电影<%d>的评分..." % (uid, iid))
11    # 1. 找出uid用户的相似用户
12    similar_users = user_similar[uid].drop([uid]).dropna()
13    # 相似用户筛选规则: 正相关的用户
14    similar_users = similar_users.where(similar_users > 0).dropna()
15    if similar_users.empty is True:
16        raise Exception("用户<%d>没有相似的用户" % uid)
17
18    # 2. 从uid用户的近邻相似用户中筛选出对iid物品有评分记录的近邻用户
19    ids = set(ratings_matrix[iid].dropna().index) & set(similar_users.index)
20    finally_similar_users = similar_users.ix[list(ids)]
21
22    # 3. 结合uid用户与其近邻用户的相似度预测uid用户对iid物品的评分
23    sum_up = 0      # 评分预测公式的分子部分的值
24    sum_down = 0    # 评分预测公式的分母部分的值
25    for sim_uid, similarity in finally_similar_users.iteritems():
26        # 近邻用户的评分数据
27        sim_user_rated_movies = ratings_matrix.ix[sim_uid].dropna()
28        # 近邻用户对iid物品的评分
29        sim_user_rating_for_item = sim_user_rated_movies[iid]
30        # 计算分子的值
31        sum_up += similarity * sim_user_rating_for_item
32        # 计算分母的值
33        sum_down += similarity
34
35    # 计算预测的评分值并返回
36    predict_rating = sum_up / sum_down
37    print("预测出用户<%d>对电影<%d>的评分: %.2f" % (uid, iid, predict_rating))
38    return round(predict_rating, 2)
39
40 if __name__ == '__main__':
41     ratings_matrix = load_data(DATA_PATH)
42
43     user_similar = compute_pearson_similarity(ratings_matrix, based="user")
44     # 预测用户1对物品1的评分
45     predict(1, 1, ratings_matrix, user_similar)
46     # 预测用户1对物品2的评分
47     predict(1, 2, ratings_matrix, user_similar)

```

- 实现预测全部评分方法: `predict_all`

```

1 def predict_all(uid, ratings_matrix, user_similar):
2     ...
3     预测全部评分
4     :param uid: 用户ID
5     :param ratings_matrix: 用户-物品打分矩阵
6     :param user_similar: 用户两两间的相似度
7     :return: 生成器, 逐个返回预测评分
8     ...
9
10    # 准备要预测的物品的id列表
11    item_ids = ratings_matrix.columns
12    # 逐个预测
13    for iid in item_ids:
14        try:
15            rating = predict(uid, iid, ratings_matrix, user_similar)
16        except Exception as e:
17            print(e)

```



```

20 if __name__ == '__main__':
21     ratings_matrix = load_data(DATA_PATH)
22
23     user_similar = compute_pearson_similarity(ratings_matrix, based="user")
24
25     for i in predict_all(1, ratings_matrix, user_similar):
26         pass

```

- 添加过滤规则

```

1 def _predict_all(uid, item_ids, ratings_matrix, user_similar):
2     ...
3     预测全部评分
4     :param uid: 用户id
5     :param item_ids: 要预测的物品id列表
6     :param ratings_matrix: 用户-物品打分矩阵
7     :param user_similar: 用户两两间的相似度
8     :return: 生成器，逐个返回预测评分
9     ...
10    # 逐个预测
11    for iid in item_ids:
12        try:
13            rating = predict(uid, iid, ratings_matrix, user_similar)
14        except Exception as e:
15            print(e)
16        else:
17            yield uid, iid, rating
18
19 def predict_all(uid, ratings_matrix, user_similar, filter_rule=None):
20     ...
21     预测全部评分，并可根据条件进行前置过滤
22     :param uid: 用户ID
23     :param ratings_matrix: 用户-物品打分矩阵
24     :param user_similar: 用户两两间的相似度
25     :param filter_rule: 过滤规则，只能是四选一，否则将抛异常: "unhot", "rated", ["unhot", "rated"], None
26     :return: 生成器，逐个返回预测评分
27     ...
28
29     if not filter_rule:
30         item_ids = ratings_matrix.columns
31     elif isinstance(filter_rule, str) and filter_rule == "unhot":
32         '''过滤非热门电影'''
33         # 统计每部电影的评分数
34         count = ratings_matrix.count()
35         # 过滤出评分数高于10的电影，作为热门电影
36         item_ids = count.where(count > 10).dropna().index
37     elif isinstance(filter_rule, str) and filter_rule == "rated":
38         '''过滤用户评分过的电影'''
39         # 获取用户对所有电影的评分记录
40         user_ratings = ratings_matrix.ix[uid]
41         # 评分范围是1-5，小于6的都是评分过的，除此以外的都是没有评分的
42         _ = user_ratings < 6
43         item_ids = _.where(_ == False).dropna().index
44     elif isinstance(filter_rule, list) and set(filter_rule) == set(["unhot", "rated"]):
45         '''过滤非热门和用户已经评分过的电影'''
46         count = ratings_matrix.count()
47         ids1 = count.where(count > 10).dropna().index
48
49         user_ratings = ratings_matrix.ix[uid]
50         _ = user_ratings < 6
51         ids2 = _.where(_) == False).dropna().index
52         # 取二者交集
53         item_ids = set(ids1) & set(ids2)
54     else:
55         raise Exception("无效的过滤参数")
56
57     yield from _predict_all(uid, item_ids, ratings_matrix, user_similar)
58

```



```

62     user_similar = compute_pearson_similarity(ratings_matrix, based="user")
63
64     for result in predict_all(1, ratings_matrix, user_similar, filter_rule=["unhot", "rated"]):
65         print(result)

```

- 根据预测评分为指定用户进行TOP-N推荐：

```

1 def top_k_rs_result(k):
2     ratings_matrix = load_data(DATA_PATH)
3     user_similar = compute_pearson_similarity(ratings_matrix, based="user")
4     results = predict_all(1, ratings_matrix, user_similar, filter_rule=["unhot", "rated"])
5     return sorted(results, key=lambda x: x[2], reverse=True)[:k]
6
7 if __name__ == '__main__':
8     from pprint import pprint
9     result = top_k_rs_result(20)
10    pprint(result)
11

```

06 案例—算法实现：Item-Based CF 预测评分

评分预测公式：

$$pred(u, i) = \hat{r}_{ui} = \frac{\sum_{j \in I_{rated}} sim(i, j) * r_{uj}}{\sum_{j \in I_{rated}} sim(i, j)}$$

算法实现

- 实现评分预测方法：`predict`

- 方法说明：

利用原始评分矩阵、以及物品间两两相似度，预测指定用户对指定物品的评分。

如果无法预测，则抛出异常

```

1 def predict(uid, iid, ratings_matrix, item_similar):
2     ...
3     预测给定用户对给定物品的评分值
4     :param uid: 用户ID
5     :param iid: 物品ID
6     :param ratings_matrix: 用户-物品评分矩阵
7     :param item_similar: 物品两两相似度矩阵
8     :return: 预测的评分值
9     ...
10    print("开始预测用户<%d>对电影<%d>的评分..." % (uid, iid))
11    # 1. 找出iid物品的相似物品
12    similar_items = item_similar[iid].drop([iid]).dropna()
13    # 相似物品筛选规则：正相关的物品
14    similar_items = similar_items.where(similar_items > 0).dropna()
15    if similar_items.empty is True:
16        raise Exception("物品<%d>没有相似的物品" % iid)
17
18    # 2. 从iid物品的近邻相似物品中筛选出uid用户评分过的物品
19    ids = set(ratings_matrix.ix[uid].dropna().index) & set(similar_items.index)
20    finally_similar_items = similar_items.ix[list(ids)]
21
22    # 3. 结合iid物品与其相似物品的相似度和uid用户对其相似物品的评分，预测uid对iid的评分
23    sum_up = 0      # 评分预测公式的分子部分的值
24    sum_down = 0    # 评分预测公式的分母部分的值
25    for sim_iid, similarity in finally_similar_items.iteritems():
26        # 近邻物品的评分数据
27        sim_item_rated_movies = ratings_matrix[sim_iid].dropna()
28        # uid用户对相似物品物品的评分

```



```

31     sum_up += similarity * sim_item_rating_from_user
32     # 计算分母的值
33     sum_down += similarity
34
35     # 计算预测的评分值并返回
36     predict_rating = sum_up/sum_down
37     print("预测出用户<%d>对电影<%d>的评分: %0.2f" % (uid, iid, predict_rating))
38     return round(predict_rating, 2)
39
40 if __name__ == '__main__':
41     ratings_matrix = load_data(DATA_PATH)
42
43     item_similar = compute_pearson_similarity(ratings_matrix, based="item")
44     # 预测用户1对物品1的评分
45     predict(1, 1, ratings_matrix, item_similar)
46     # 预测用户1对物品2的评分
47     predict(1, 2, ratings_matrix, item_similar)

```

- 实现预测全部评分方法: `predict_all`

```

1 def predict_all(uid, ratings_matrix, item_similar):
2     ...
3     预测全部评分
4     :param uid: 用户id
5     :param ratings_matrix: 用户-物品打分矩阵
6     :param item_similar: 物品两两间的相似度
7     :return: 生成器, 逐个返回预测评分
8     ...
9
10    # 准备要预测的物品的id列表
11    item_ids = ratings_matrix.columns
12    # 逐个预测
13    for iid in item_ids:
14        try:
15            rating = predict(uid, iid, ratings_matrix, item_similar)
16        except Exception as e:
17            print(e)
18        else:
19            yield uid, iid, rating
20
21 if __name__ == '__main__':
22     ratings_matrix = load_data(DATA_PATH)
23
24     item_similar = compute_pearson_similarity(ratings_matrix, based="item")
25     for i in predict_all(1, ratings_matrix, item_similar):
26         pass

```

- 添加过滤规则

```

1 def _predict_all(uid, item_ids, ratings_matrix, item_similar):
2     ...
3     预测全部评分
4     :param uid: 用户id
5     :param item_ids: 要预测物品id列表
6     :param ratings_matrix: 用户-物品打分矩阵
7     :param item_similar: 物品两两间的相似度
8     :return: 生成器, 逐个返回预测评分
9     ...
10
11    # 逐个预测
12    for iid in item_ids:
13        try:
14            rating = predict(uid, iid, ratings_matrix, item_similar)
15        except Exception as e:
16            print(e)
17        else:
18            yield uid, iid, rating
19
20 def predict_all(uid, ratings_matrix, item_similar, filter_rule=None):
21     ...

```

```

24     :param ratings_matrix: 用户-物品打分矩阵
25     :param item_similar: 物品两两间的相似度
26     :param filter_rule: 过滤规则, 只能是四选一, 否则将抛异常: "unhot", "rated", ["unhot", "rated"], None
27     :return: 生成器, 逐个返回预测评分
28     """
29
30     if not filter_rule:
31         item_ids = ratings_matrix.columns
32     elif isinstance(filter_rule, str) and filter_rule == "unhot":
33         """过滤非热门电影"""
34         # 统计每部电影的评分数
35         count = ratings_matrix.count()
36         # 过滤出评分数大于10的电影, 作为热门电影
37         item_ids = count.where(count>10).dropna().index
38     elif isinstance(filter_rule, str) and filter_rule == "rated":
39         """过滤用户评分过的电影"""
40         # 获取用户对所有电影的评分记录
41         user_ratings = ratings_matrix.ix[uid]
42         # 评分范围是1-5, 小于6的都是评分过的, 除此以外的都是没有评分的
43         _ = user_ratings<6
44         item_ids = _.where(_==False).dropna().index
45     elif isinstance(filter_rule, list) and set(filter_rule) == set(["unhot", "rated"]):
46         """过滤非热门和用户已经评分过的电影"""
47         count = ratings_matrix.count()
48         ids1 = count.where(count > 10).dropna().index
49
50         user_ratings = ratings_matrix.ix[uid]
51         _ = user_ratings < 6
52         ids2 = _.where(_ == False).dropna().index
53         # 取二者交集
54         item_ids = set(ids1)&set(ids2)
55     else:
56         raise Exception("无效的过滤参数")
57
58     yield from _predict_all(uid, item_ids, ratings_matrix, item_similar)
59
60 if __name__ == '__main__':
61     ratings_matrix = load_data(DATA_PATH)
62
63     item_similar = compute_pearson_similarity(ratings_matrix, based="item")
64
65     for result in predict_all(1, ratings_matrix, item_similar, filter_rule=["unhot", "rated"]):
66         print(result)

```

- 为指定用户推荐TOP-N结果

```

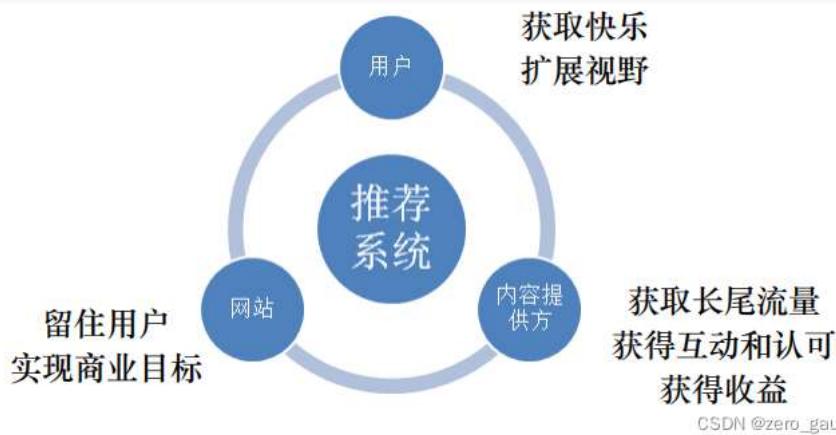
1 def top_k_rs_result(k):
2     ratings_matrix = load_data(DATA_PATH)
3
4     item_similar = compute_pearson_similarity(ratings_matrix, based="item")
5     results = predict_all(1, ratings_matrix, item_similar, filter_rule=["unhot", "rated"])
6     return sorted(results, key=lambda x: x[2], reverse=True)[:k]
7
8 if __name__ == '__main__':
9     from pprint import pprint
10    result = top_k_rs_result(20)
11    pprint(result)

```

07 推荐系统评估

好的推荐系统可以实现用户, 服务提供方, 内容提供方的共赢





显示反馈和隐式反馈

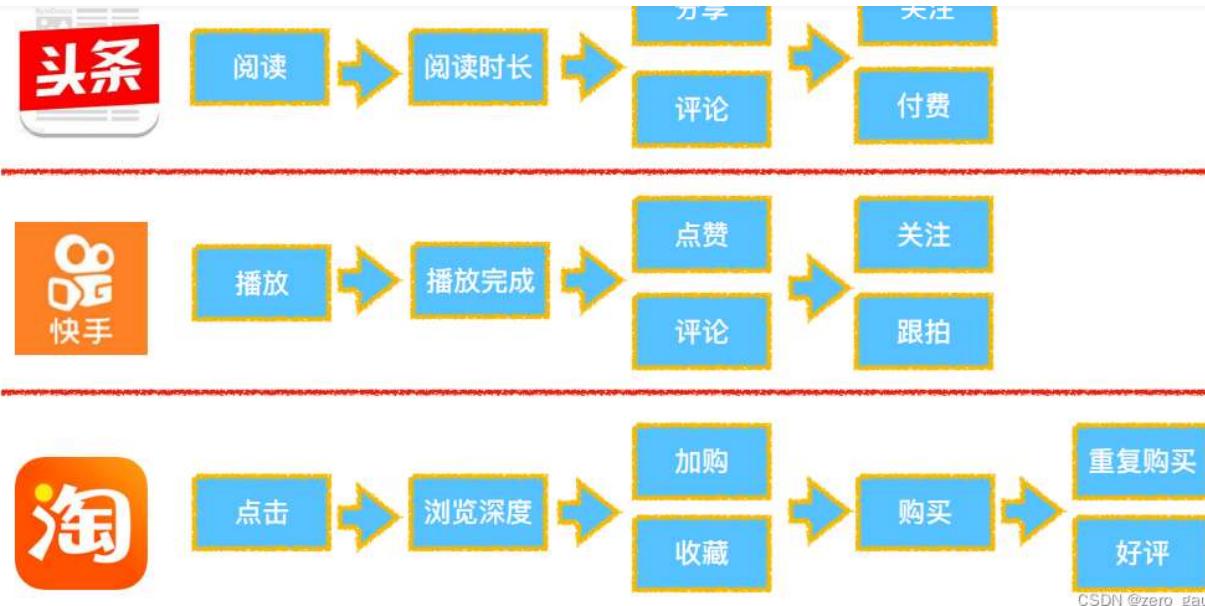
	显式反馈	隐式反馈
例子	电影/书籍评分 是否喜欢这个推荐	播放/点击 评论 下载 购买
准确性	高	低
数量	少	多
获取成本	高	低

常用评估指标

- 准确性 • 信任度
- 满意度 • 实时性
- 覆盖率 • 鲁棒性
- 多样性 • 可扩展性
- 新颖性 • 商业目标
- 惊喜度 • 用户留存

- 准确性 (理论角度) Netflix 美国录像带租赁
 - 评分预测
 - RMSE MAE
 - topN推荐
 - 召回率 精准率
- 准确性 (业务角度)





- 覆盖度
 - 信息熵，对于推荐越大越好
 - 覆盖率
- 多样性&新颖性&惊喜性
 - 多样性：推荐列表中两两物品的不相似性。（相似性如何度量？）
 - 新颖性：未曾关注的类别、作者；推荐结果的平均流行度
 - 惊喜性：历史不相似（惊）但很满意（喜）
 - 往往需要牺牲准确性
 - 使用历史数据为预测用户对某个物品的喜爱程度
 - 系统过度强调实时性
- Exploitation & Exploration 探索与利用问题
 - Exploitation(开发利用)：选择现在可能最佳的方案
 - Exploration(探测搜索)：选择现在不确定的一些方案，但未来可能会有高收益的方案
 - 在做两类决策的过程中，不断更新对所有决策的不确定性的认知，优化长期的目标
- EE问题实践
 - 兴趣扩展：相似话题，搭配推荐
 - 人群算法：userCF 用户聚类
 - 平衡个性化推荐和热门推荐比例
 - 随机丢弃用户行为历史
 - 随机扰动模型参数
- EE可能带来的问题
 - 探索伤害用户体验，可能导致用户流失
 - 探索带来的长期收益(留存率)评估周期长，KPI压力大
 - 如何平衡实时兴趣和长期兴趣
 - 如何平衡短期产品体验和长期系统生态
 - 如何平衡大众口味和小众需求
- 评估方法
 - 问卷调查：成本高
 - 离线评估：
 - 只能在用户看到过的候选集上做评估，且跟线上真实效果存在偏差



- 在线评估: 灰度发布 & A/B测试 50% 全量上线
- 实践: 离线评估和在线评估结合, 定期做问卷调查

08 推荐系统的冷启动问题

推荐系统冷启动概念

- 用户冷启动: 如何为新用户做个性化推荐
- 物品冷启动: 如何将新物品推荐给用户 (协同过滤)
- 系统冷启动: 用户冷启动+物品冷启动
- 本质是推荐系统依赖历史数据, 没有历史数据无法预测用户偏好

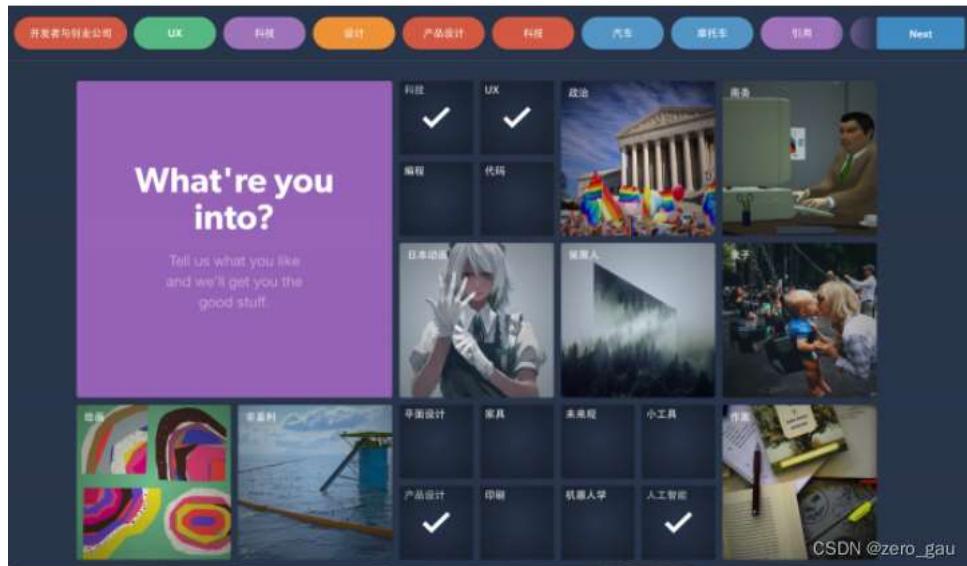
用户冷启动

1. 收集用户特征

- 用户注册信息: 性别、年龄、地域
- 设备信息: 定位、手机型号、app列表
- 社交信息、推荐素材、安装来源



2 引导用户填写兴趣

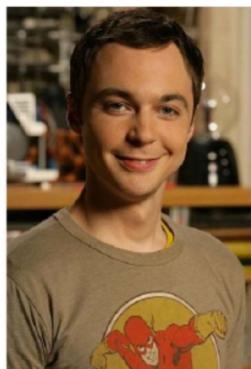
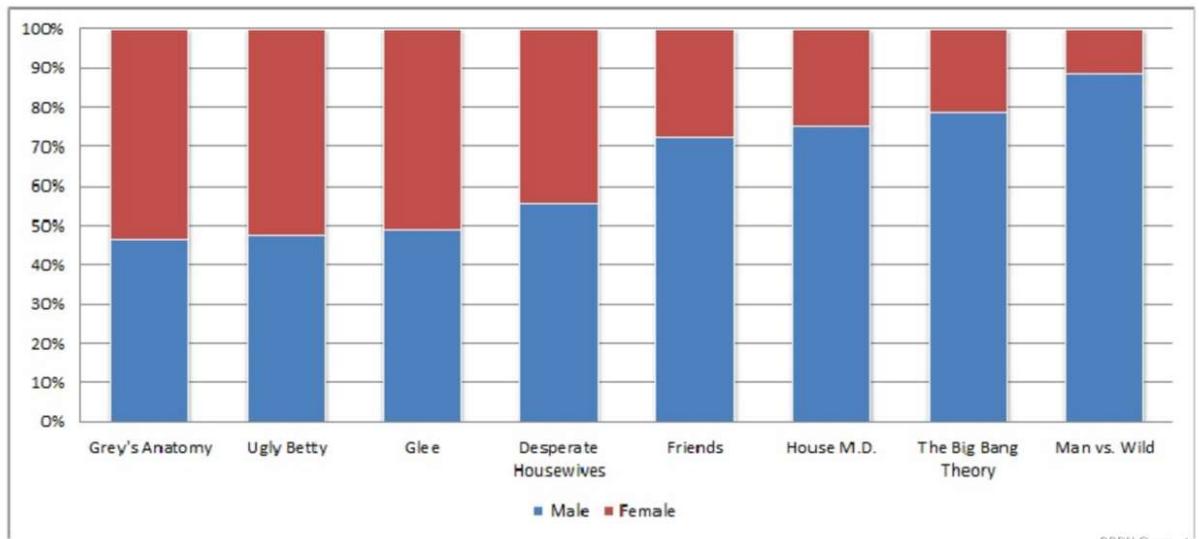


3 使用其它站点的行为数据

例如腾讯视频& QQ音乐 今日头条&抖音

新老用户推荐策略的差异

- Explore Exploit力度
- 使用单独的特征和模型预估
- 举例 性别与电视剧的关系



Male
Age : 20-30
Theoretical physicist
Doctor
American
Irreligious

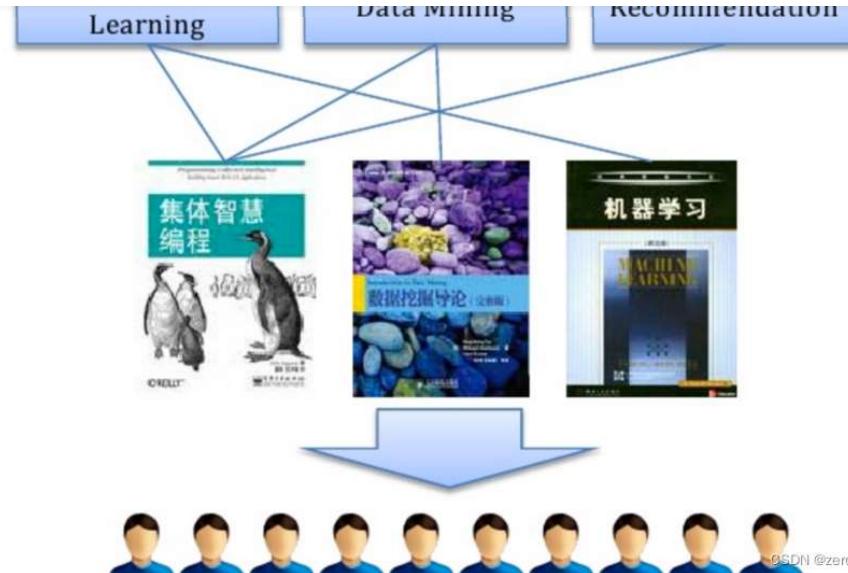


CSDN @zero_gau

物品冷启动

- 给物品打标签
- 利用物品的内容信息，将新物品先投放给曾经喜欢过和它内容相似的其他物品的用户。





CSDN @zero_gau

- 系统冷启动
 - 基于内容的推荐 系统早期
 - 基于内容的推荐逐渐过渡到协同过滤
 - 基于内容的推荐和协同过滤的推荐结果都计算出来 加权求和得到最终推荐结果



显示推荐内容



da_journeyer



5



15



¥



0



专栏目录

