**《基于师生图书借阅记录的推荐系统》**

宋 汝 阳

华 中 师 范 大 学

计算机学院

**目录：**

1. **推荐算法**

**1.1根据人口统计学的推荐** -------------- **1**

**1.2基于内容的推荐**

**1.3基于关联规则的推荐**

**1.4基于协同过滤的推荐**

**1.4.1基于用户的推荐**

**1.4.2基于物品的推荐**

1. **结合算法进行分析** -------------- **5**

**2.1优缺点**

**2.1.1冷启动问题**

**2.1.2领域无关**

**2.1.3系统性能**

**2.1.4潜在兴趣**

**2.1.5热度过度推荐**

**2.2实现所需数据要求**

1. **具体实现** -------------- **7**

**3.1对于新用户**

**3.2对于老用户**

**3.2.1关联规则**

**3.2.2使用Apriori算法获取频繁项集\***

**3.2.3使用FP-Growth算法获取频繁项集\***

**3.2.3.1 FP-Tree**

**3.2.3.2 Header-Table**

**3.2.3.3 Conditional Pattern Base**

**3.2.3.4 Conditional Pattern Tree**

**3.2.3.5 判定**

**1.推荐算法**

**推荐**：根据用户兴趣和行为特点，向用户推荐所需要的信息或者商品，帮助用户在海量信息中快速发现真正需要的商品，提高用户黏性，促进信息点击和商品销售。推荐系统是基于海量数据挖掘分析的商业智能平台，推荐主要基于以下信息：（推荐算法显然在用户方的利益是可以减少时间和精力的花费，从而达到服务大众的作用）

1.热点信息

2.用户信息

3.用户历史浏览或者行为记录

4.社会化关系

【如果是基于借阅信息的推荐】：

1.借阅者的所属院系

2.借阅者的借阅记录

3.借阅者的身份

**1.1--根据人口统计学--**

简单的根据用户的基本信息发现用户的相关程度，然后将相似用户喜爱的其他物品推荐给当前用户。

class profile{

age

male\_or\_female

and\_so\_on

..

}

if A.profile ≈ B.profile

bind(A,B)

B.history-->A

**好处：**

1.因为不使用当前用户对于物品的喜好历史数据，所以对于新用户来讲没有“冷启动”的问题。

2.这个方法不依赖于物品本身的数据，所以这个方法在不同物品的领域都可以使用，他是领域独立的（domain-independent）。

**缺点：**

粗糙（图书等领域，得不到很好的推荐效果）可能涉及到一些与信息发现问题本身无关却比较敏感的信息。

**1.2--基于内容的推荐--**

**核心思想**：

根据推荐物品或内容的元数据，发现物品或者内容的相关性，然后基于用户以往的喜好记录，推荐给用户相似的物品

这种推荐系统多用于一些资讯类的应用上，针对文章本身抽取一些tag作为该文章的关键词，继而可以通过这些tag来评价两篇文章的相似度。

**优点**：

1.易于实现，不需要用户数据，因此不存在稀疏性和冷启动问题。

2.基于物品本身特征推荐，因此不存在过度推荐热门这个问题。

**缺点**:

对于tag的抽取：既要保证准确性又要具有实际意义，否则很难保证推荐结果的相关性。

**1.3--基于关联规则的推荐--**

**首要目标**：

挖掘出**关联规则**，也就是那些同时被很多用户购买的物品集合，这些集合内的物品可以相互进行推荐。

**演变来源：Apriori\*和FP-Growth\***

基于关联规则的推荐系统一般转化率较高因为当用户已经购买了频繁集合中的若干项目后，购买该频繁集合中其他项目的可能性更高。

**缺点**：

1.计算量较大，但是可以离线计算，因此影响不大。

2.由于采用用户数据，不可避免的存在冷启动和稀疏性问题。

3.存在热门项目容易被过度推荐的问题。

**1.4--基于协同过滤的推荐--**

这种算法基于一个“物以类聚，人以群分”的假设，喜欢相同物品的用户更有可能具有相同的兴趣。

基于协同过滤的推荐系统一般应用于有用户评分的系统之中，通过分数去刻画用户对于物品的喜好。

通过用户建立物品与物品之间的联系。

**1.4.1-->①User-based**

**--基于用户的推荐--**

**基本原理：**

根据所有用户对物品或者信息的偏好，发现与当前用户口味和偏好相似的“邻居”用户群，在一般的应用中是采用计算“K-Nearest Neighboor”的算法；然后，基于这个K个邻居的历史偏好信息，为当前用户进行推荐。

**优点:**

推荐物品之间在内容上可能完全不相关，因此可以发现用户的潜在兴趣，并且针对每个用户生成其个性化的推荐结果。

**缺点：**

在于一般的Web系统中，用户的增长速度远远大于物品的增长速度，因此计算量的增长巨大，系统性能容易成为瓶颈。

**1.4.2-->②Item-based**

**--基于物品的推荐--**

**基本原理：**

使用所有用户对物品或者信息的偏好，发现物品和物品之间的相似度，然后根据用户的历史偏好信息，将类似的物品推荐给用户。

基于物品的协同过滤可以看做是关联规则推荐的一种退化，但由于协同过滤更多考虑了用户的实际评分，并且知识计算相似度而非寻找频繁集，因此可以认为基于物品的协同过滤准确率较高并且覆盖率较高。

**优点：**

与基于用户的推荐相比，这种推荐方法应用更为广泛，扩展性和算法性能更好，由于项目的增长速度一般较为平缓，因此性能变化不大。

**缺点:**

无法提供个性化的推荐结果。

**1.4.3基于协同过滤的推荐机制**是现今应用最为广泛的推荐机制，**优点**如下：

1.不需要对物品或者用户进行严格的建模，而且不要求物品的描述是机器可以理解的，所以这种方法是领域无关的。

2.这种方法计算出来的推荐是开放的，可以共用他人的经验，很好的支持用户发现潜在的兴趣偏好。

**缺点:**

1.方法的核心是基于历史数据，所以对新物品和新用户都有冷启动的问题。

2.推荐的效果依赖于用户历史偏好数据的多少和准确性。

3.大部分的实现中，用户历史偏好是用稀疏矩阵进行存储的，而稀疏矩阵上的计算有明显的问题，包括可能少部分人的错误偏好会对推荐的准确度有很大的影响。

4.对于一些特殊品味的用户不能给予很好的推荐。

5.由于以历史数据为基础，抓取和建模用户的偏好后，很难修改或者根据用户的使用演变，从而导致这个方法不够灵活。

1. **结合算法进行分析**

**2.1优缺点：**

不同的推荐算法存在不同的问题：冷启动，领域无关，系统性能，潜在兴趣，热度物品被过度推荐等。

**2.1.1冷启动:**

**描述**：

如何在没有大量用户行为数据的情况下设计个性化推荐系统并且让用户对推 荐系统满意，从而愿意使用推荐系统，这就是冷启动问题。

**应对**：

这个问题在于，我们的用户群分为两类，一类是有一定借阅记录基础的用户（暂 且称之为老用户）一类是新生或者是没什么借阅记录的用户（暂且称之为新用户）。 对于老用户，冷启动问题几乎不会干扰算法的性能，但是对于新用户，冷启动就会 卡住一部分算法的实现。

因此我们需要区别对待，对于老用户使用存在冷启动问题的算法，对于新用户 使用不存在冷启动问题的算法。

**如何确定新老用户？**

--个人看法--：

问题可以转化为：确定一个常数C，使得：

·(0<=p<C) 新用户

·(C<=p) 老用户 p为借阅数。

对于C的确定，我们可以利用二分法，对区间(1,MAX)进行log2MAX次迭 代，每次都对一个示例用户进行推荐，并且进行人工的评分，最终确定一个评 分最高的C。.

**2.1.2领域无关：**

**描述：**

我是这么理解的，就是书籍能否跨领域推荐，比如能否根据一本心理学的书籍 而推荐出一本计算机科学的书籍。

**应对：**

对于这个问题，自然是能推荐更好。因此不作为主要因素去进行考虑。

**2.1.3系统性能：**

由于现在是理论阶段的讨论和研究，所以以有足够的系统性能为支撑。

并且，个人估计数据不会达到很高的量，像淘宝或者Amazon一样。因此暂时忽略。

**2.1.4潜在兴趣：**

在一个简单的系统中不需要涉及这方面的内容，因为这个对于系统增加的负担和实 际效果并不成正比。但是这个功能可以放在后续研发上。在初期阶段不予考虑。

**2.1.5热度书籍过度推荐：**

这个问题我们可以写一个热度榜单，另行放置，因为统计出热度最高的几本书并不 是一个难事，我们只需要把这些书列一个Top10的榜单，然后剔除这些书籍被推荐的机 会。就可以解决这个问题了。

**2.2实现所需的数据要求:**

**2.2.1基于人口统计学的推荐：**

·用户的基本信息。

**2.2.2基于内容的推荐：**

·书籍的标签(tag)。

**2.2.3基于关联规则的推荐：**

·书籍之间的关联规则，即经常被一群人借阅的书籍集合。

**2.2.4基于协同过滤的推荐：**

·用户对书籍的评分。

**总结：**

很显然，对于书籍的标签，如果数据中没有，获取tag是一件很繁重的工作，且不 容易实现，而用户对于书籍的评分也只能在新数据中获取。

所以因为数据的限制，只能选择**基于人口统计学的推荐**和**基于关联规则的推荐**。

**2.3结论:**

**综上所述：**

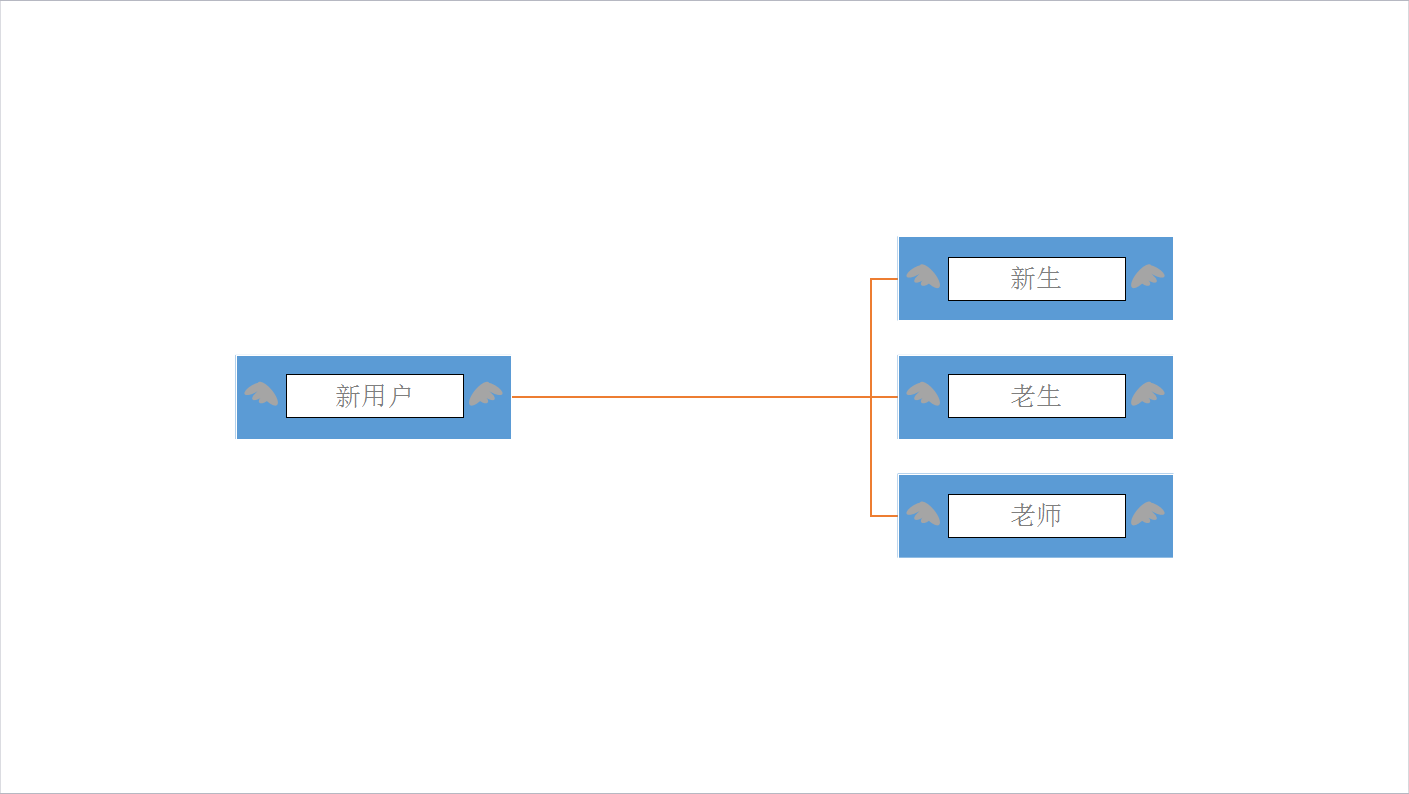
>·**对于新用户：**使用**基于人口统计学**的推荐。

>·**对于老用户：**使用**基于关联规则**的推荐。

1. **具体实现**

**3.1对于新用户：**

**>>基于人口统计学>>**



>·对于老生，我们可以从他们**同院系的老用户集合**中挑选相似的人选。

>·对于新生，我们可以从他们**院系的学长学姐集合**中挑选相似的人选。

>·对于教师，我们可以从他们**同院系的同事的集合**中挑选相似的人选。

很明显我们需要知晓这些新用户的院系，我们可以**从用户的ID上挖掘到现有的院 系**。

**case：对于转院的同学**

很显然我们对于学号（因为学校教师数量远小于学生）的简单解析就可 以获得一个学生入学时的时间以及院系，但是对于转院的学生来说，我们 只能从学校的数据中获取一个学生的院系。

这就让我们面临一个问题：对于小部分人群产生的这个问题，我们是使 用离线的直接解析学号的方式获取院系信息，还是使用在线的查询院系方 式。

对于学生这种特殊群体来说，学校所能获取的信息也不过就是一些基础信息， 并且**大多数和借阅书籍无关**（比如借阅一本书和你来自山东还是来自湖北，高 考考多少分根本没有关系）因此我们很难通过这些信息来判定两个用户是否相 似。因此我做出以下**改进**：

**基于现实情况对基于人口统计学的推荐的改进：**

1. **主观地将同院系作为用户相似的依据**
2. **由于这样使得相似度不可避免的降低，我们建立一个相似用户集合，并且从相似用户集合中依靠基于关联规则的推荐抽取出一个书籍集合，把这个集合作为推荐的全集。**

**3.给出推荐之后，根据新用户的新的借阅记录，来改变从推荐全集中 挑选的书籍的分类的比重。**

伪代码：

**for user in x-academic.x-grade.student:**

**for book in user.books:**

**if book not in book-set:**

**add book in book-set**

**book-set.book.count++**

**sort book-set.book by count**

**delete (L,0) in book-set**

**make a set in random from book-set**

**recommend set to x-academic.(x-1)-grade.student**

**3.2对于老用户**

**>>基于关联规则>>**

对于老用户，所有的老用户的借阅记录次数都大于某一阈值C。

**3.2.1关联规则**：

关联规则是形如X->Y的蕴含表达式,其中X和Y是不相交的项集,即X∩Y=∅。

关联规则的**强度**可以用它的**支持度(support)**和**置信度(confidence)**来度量

**支持度确定规则可以用于给定数据集的频繁程度（能够关联的次数）**

**置信度确定Y在包含X的交易中出现的频繁程度（能够成功关联的次数）**

支持度s和置信度c的形式定义如下：

**||--> ·s(X->Y)=σ(X∪Y)/N**

**||--> ·c(X->Y)=σ(X∪Y)/σ(X)**

因此，大多数关联规则挖掘算法通常采用的一种策略是：

将关联规则挖掘任务分解为如下两个主要的子任务：

1. **频繁项集的产生：**

其目标是发现满足最小值尺度阈值的所有项集，这些项集称作 **频繁项集(frequent itemset)**

1. **规则的产生：**

其目标是从上一步发现的频繁项集中提取所有高置信度的规 则，这些规则称作**强规则(strong rule)**

显而易见。频繁项集的产生所需要的计算开销**远大于**产生规则所需的计算 开销。

3.2.2使用Apriori算法获取频繁项集\*

**暴力挖取**：

最容易想到的、也是最直接的关联关系挖掘的方法或许就是暴力搜索 (Brute-force)的方法。

但是暴力的计算量过大，一个包含k个项的数据集可能产生2^k-1个 频繁项集。

发现频繁项集的一种原始方法是确定每一个候选项集(candidate -itemset)的支持度计数。为了完成这一任务，必须将每个候选项集与每个交易进行比较。

如果候选项集包含在交易中则候选集的支持度计数增加。

**复杂度O(NMω),N是交易数,M=2^k-1是候选项集数，而ω是交易的最大宽度(也就是交易中最大的项数)**

**先验原理**：

对于本身复杂度极高的Brute-force，我们必须设法降低产生频繁项 集的计算复杂度。此时我们可以利用支持度对候选项集进行剪枝。

**Apriori定律1：**

**如果一个集合是频繁项集，则它所有的子集都是频繁项集。**

**Apriori定律2：**

**如果一个集合不是频繁项集，则它的所有的超集都不是频繁项集。**

剪枝原则：

依据apriori定律2，当一个集合不是频繁项集时，剪除所有向 下的集合。

**频繁项集的产生**：

**R.Agrawal** 和 **R. Srikant**于1994年在文献中提出了Apriori算法，该算法的描述如下：

===========================================================

·Let k=1

·Generate frequent itemsets of length k

·Repeat until no new frequent itemsets are identified

·Generate length (k+1) candidate itemsets from length k frequent itemsets

·Prune candidate itemsets containing subsets of length k+1 that are infrequent

·Count the support of each candidate by scanning the DB

·Eliminate candidates that are infrequent, leaving only those that are frequent

===========================================================

D为总集

1. 建立频繁-1-项集
2. for k=2 频繁-k-项集不为空

从频繁-k-1-项集中**产生候选-K-项集**

计算候选-k-项集的**支持度计数**

去除小于支持度计数阈值的集合

返回频繁-k-项集

1. 返回频繁项集

产生候选项集：

从频繁-k-1项集中挑选两项，使得：

前k-2项相同

合并这两项

计算支持度计数：

for 所有的事务 in D：

产生这个事务的子集集合

判断候选项集是否属于这个子集

===========================================================

**3.2.3使用FP-Growth算法获取频繁项集\***

Apriori算法的缺点在于，过多的重复扫描数据库。而另一种方法 可以避免这种情况。

FP-Growth算法是一种基于FP-Tree(Frequent Pattern Tree)的算 法。

算法使用递归解决多个子问题的方式来产生频繁项集，相比apriori也更容易实现。

**3.2.3.1.FP-Tree Frequent Pattern Tree | 频繁模式树：**

Frequent pattern tree是一种前缀树，它可以看做是所有事 务的投影。

建树方法：insert\_fp\_tree(p|P,T)

首先我们统计出所有的频繁一项集。

然后对所有的事务进行过滤和排序(降序)。

把每一条事务看作是(头|后缀)的形式，后缀可以为空。

然后递归的调用insert\_fp\_tree()

每次进行判断:

若当前结点的儿女中有p，那么这个儿女的频繁度+1

若没有，新建这个儿女，频繁度设置为1

然后递归地，把后缀P进行分割，然后对下一个结点进行 insert\_fp\_tree()

**3.2.3.2.Header\_Table：**

Header\_Table的存在是为了方便进行快速查找。

Header\_Table有三个域：item\_name,count,next

item\_name是这个item的唯一标识符

count是这个item的支持度

next指向FP-Tree中的一个item的点，并且连接成串

**3.2.3.3.Conditional Pattern Base | 条件模式基：**

条件模式基是由FP-Tree中所有的item的前缀构成的。

从事务数据的角度上讲，就是包含item(可以不止一个元素) 的所有事务集

generate conditional pattern base:

构造一个条件模式基，我们利用Header\_Table对所有的 item进行浏览，然后保存前缀路径即可。

**3.2.3.4.Conditional Pattern Tree | 条件模式树：**

从条件模式基构建出的FP-Tree称为条件模式树。

对条件模式基中所有item的频繁度进行加和，然后用阈值过 滤，新的由过滤后的前缀路径形成的FP-Tree就是条件模式树 了。

**3.2.3.5.判定：**

如果一个item(可以不止一个)对应的条件模式树为空或者仅 有一条路径。

那么：

**item∪tree即为一个频繁项集。由apriori定理1可知， 其中所有的子集都是频繁项集。**

Apriori算法和FP-Growth算法各有利弊，在FP-Tree不是很茂盛的情况下，取用后者是比较好的选择。因此我们可以尽量的减小FP-Tree的宽度然后使用FP-Growth算法。比如分院系产生一个频繁项集用于推荐。

因此在对于老用户的推荐上，可以得出方法：

1.首先对于一些显而易见的情况进行分类，尽量减小FP-Tree的宽度。

2.然后调用FP-Growth算法来产生频繁项集。

3.将老用户的借阅记录和频繁项集进行结合，产生推荐目录。

以上是对于新老用户的推荐算法选择以及具体步骤。