常用推荐算法及原理

一． 基本概念

推荐算法是计算机专业中的一种算法，通过一些数学算法，推测出用户可能喜欢的东西，目前应用推荐算法比较好的地方主要是网络，其中淘宝做的比较好。所谓推荐算法就是利用用户的一些行为，通过一些数学算法，推测出用户可能喜欢的东西。

二． 常用推荐算法

内容关联算法(Content-Based)

CB 算法的原理是将一个 item 的基本属性，内容等信息提取出来，抽成一个 taglist，为每个 tag 赋一个权重。

剩下的事情就跟一个搜索引擎非常类似了，将所有 item 对应的 taglist 做一下倒排转换，放到倒排索引服务器中存储起来。

当要对某一个 item 做相关推荐的时候，将这个 item 对应的 taglist 拿出来拼成一个类似搜索系统中的 query 表达式，再将召回的结果做一下排序作为推荐结果输出。

当要对某个用户做个性化推荐的时候，将这个用户最近喜欢/操作过的 item 列表拿出来，将这些 item 的 taglist 拿出来并 merge 一下作为用户模型，并将这个模型的 taglist 请求倒排索引服务，将召回的结果作为候选推荐给该用户。

该算法的好处是:

不依赖于用户行为，即不需要冷启动的过程，随时到随时都能推荐

可以给出看起来比较合理的推荐解释

item 被推荐的时效性可以做得很高，比如新闻类产品就需要用到该算法

该算法的坏处是:

需要理解 item 的内容，对音频/视频等不好解析内容的就不好处理了

对于一次多义以及一义多词等情况处理起来比较复杂

容易出现同质化严重的问题，缺乏惊喜

协同过滤算法(collaborative filtering)

CF 算法的原理是汇总所有<user,item>的行为对，利用集体智慧做推荐。其原理很像朋友推荐，比如通过对用户喜欢的 item 进行分析，发现用户 A 和用户 B 很像(他们都喜欢差不多的东西)，用户 B 喜欢了某个 item，而用户 A 没有喜欢，那么就把这个 item 推荐给用户 A。(User-Based CF)

当然，还有另外一个维度的协同推荐。即对比所有数据，发现 itemA 和 itemB 很像(他们被差不多的人喜欢)，那么就把用户 A 喜欢的所有 item，将这些 item 类似的 item 列表拉出来，作为被推荐候选推荐给用户 A。(Item-Based CF)

如上说的都是个性化推荐，如果是相关推荐，就直接拿 Item-Based CF 的中间结果就好啦。

该算法的好处是:

能起到意想不到的推荐效果，经常能推荐出来一些惊喜结果

进行有效的长尾 item

只依赖用户行为，不需要对内容进行深入了解，使用范围广

该算法的坏处是:

一开始需要大量的<user,item>行为数据，即需要大量冷启动数据

很难给出合理的推荐解释

原理

协同过滤算法具体实现的时候，又分为典型的两类:

基于领域的协同过滤算法

这类算法的主要思想是利用<user,item>的打分矩阵，利用统计信息计算用户和用户，item 和 item 之间的相似度。然后再利用相似度排序，最终得出推荐结果。

常见的算法原理如下:

User-Based CF

先看公式:

$$sim(i,j)=\frac{\sum_{x\in I_{ij}}\left(R_{i,x}-\overline{R_i}\right)\left(R_{j,x}-\overline{R_j}\right)}{\sqrt{\sum_{x\in I_{ij}}\left(R_{i,x}-\overline{R_i}\right)^2}\sqrt{\sum_{x\in I_{ij}}\left(R_{j,x}-\overline{R_j}\right)^2}}$$

该公式要计算用户 i 和用户 j 之间的相似度, I(ij)是代表用户 i 和用户 j 共同评价过的物品, R(i,x)代表用户 i 对物品 x 的评分, R(i)头上有一杠的代表用户 i 所有评分的平均分，之所以要减去平均分是因为有的用户打分严有的松，归一化用户打分避免相互影响。

该公式没有考虑到热门商品可能会被很多用户所喜欢，所以还可以优化加一下权重，这儿就不演示公式了。

在实际生产环境中，经常用到另外一个类似的算法 Slope One，该公式是计算评分偏差，即将共同评价过的物品，将各自的打分相减再求平均。

Item-Based CF

先看公式:

$$sim(i,j)=\frac{\sum_{x\in U_{ij}}\left(r_{i,x}-\overline{r_i}\right)\left(r_{j,x}-\overline{r_j}\right)}{\sqrt{\sum_{x\in U_{ij}}\left(r_{i,x}-\overline{r_i}\right)^2}\sqrt{\sum_{x\in U_{ij}}\left(r_{j,x}-\overline{r_j}\right)^2}}$$

该公式跟 User-Based CF 是类似的，就不再重复解释了。

这类算法会面临两个典型的问题:

矩阵稀疏问题

计算资源有限导致的扩展性问题

基于此，专家学者们又提出了系列基于模型的协同过滤算法。

基于模型的协同过滤算法

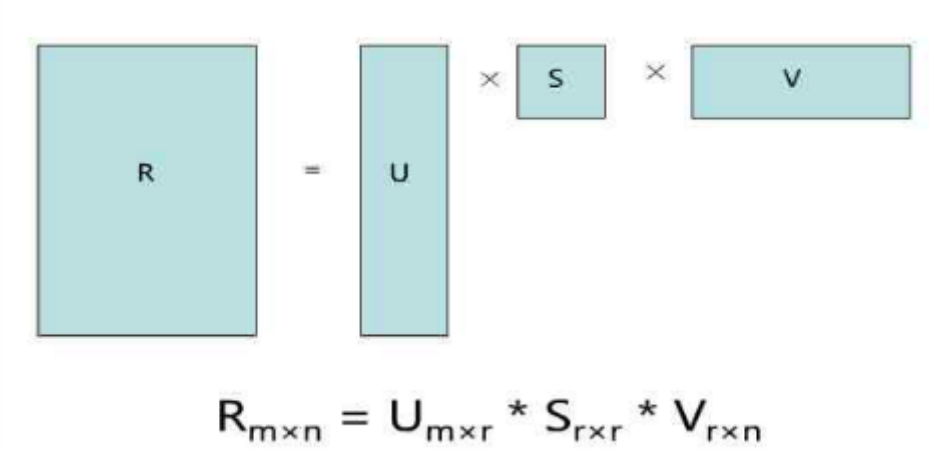
基于模型的研究就多了，常见的有:

基于矩阵分解和潜在语义的

基于贝叶斯网络的

基于 SVM 的

这儿只简单介绍一下基于矩阵分解的潜在语义模型的推荐算法。该算法首先将稀疏矩阵用均值填满，然后利用矩阵分解将其分解为两个矩阵相乘，如下图:



看一个实际的例子:

Index Words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
book			1	1					
dads						1			1
dummies		1						1	
estate							1		1
guide	1					1			
investing	1	1	1	1	1	1	1	1	1
market	1		1						
real							1		1
rich						2			1
stock	1		1					1	
value				1	1				

book	0.15	-0.27	0.04
dads	0.24	0.38	-0.09
dummies	0.13	-0.17	0.07
estate	0.18	0.19	0.45
guide	0.22	0.09	-0.46
investing	0.74	-0.21	0.21
market	0.18	-0.30	-0.28
real	0.18	0.19	0.45
rich	0.36	0.59	-0.34
stock	0.25	-0.42	-0.28
value	0.12	-0.14	0.23

×

3.91	0	0
0	2.61	0
0	0	2.00

×

T1	T2	T3	T4	T5	T6	T7	T8	T9
0.35	0.22	0.34	0.26	0.22	0.49	0.28	0.29	0.44
-0.32	-0.15	-0.46	-0.24	-0.14	0.55	0.07	-0.31	0.44
-0.41	0.14	-0.16	0.25	0.22	-0.51	0.55	0.00	0.34

这个例子中，原始矩阵中包含了网页的 Title 和切词后的 term 之间关系，可以类比为推

荐系统中的评分。然后用 SVD 做矩阵分解之后，针对每个 term 会对应一个 3 维向量，针对每个 Title 也会对应一个 3 维向量。

那么接下来可以做的事情就很多了，如果要计算 term 和 title 的相似度，只需要将这两个 3 为向量做内积得到的分值即可;

还可以将 term 和 title 都投影到这 3 维空间中，然后利用各种聚类算法，将用户和 item，item 和 item，用户和用户的分类都给求出来。

该算法的核心是在于做矩阵分解，在矩阵大了的情况下计算量是非常夸张的，在实际生产环境中会常用梯度递归下降方法来求得一个近似解。

组合推荐技术

其实从实践中来看，没有哪种推荐技术敢说自己没有弊端，往往一个好的推荐系统也不是只用一种推荐技术就解决问题，往往都是相互结合来弥补彼此的不足，常见的组合方式如下:

混合推荐技术: 同时使用多种推荐技术再加权取最优;

切换推荐技术: 根据用户场景使用不同的推荐技术;

特征组合推荐技术: 将一种推荐技术的输出作为特征放到另一个推荐技术当中;

层叠推荐技术: 一个推荐模块过程中从另一个推荐模块中获取结果用于自己产出结果