【RS】协同过滤-进阶篇

原创 机智的叉烧 CS的陋室 2019-03-09

Believer

Imagine Dragons/Kaskade - Believer (Kaskade Remix)

【RS】

本栏目是结合我最近上的七月在线的课、自己自学、以及一些个人的经验推出的专栏，从推荐系统的基础到一些比较好的case，我都会总结发布，当然，按照我往期的风格，更加倾向于去讨论一些网上其实讲得不够的东西，非常推荐大家能多看看并且讨论，欢迎大家给出宝贵意见，觉得不错请点击推文最后的好看，感谢各位的支持。

往期回顾：

- 技术向：推荐学习推荐系统（深度思考，不是广告）
- 【RS】推荐系统的评估
- 【RS】协同过滤-基础篇
- 【RS】协同过滤-user_based
- 【RS】协同过滤-item_based

在前面的章节中，我谈到了比较基础的协同过滤，然而我们也能很清楚的发现，基本的UserCF VS ItemCF存在很多漏洞，这些漏洞将会会导致给用户推荐的内容并不合适，这是协同过滤的最后一个章节，我会对这两种模式进行对比，同时讨论一下协同过滤的而一些改进和拓展，最后谈一谈协同过滤在整个推荐系统中的地位和作用，以便在实战中可以考虑实际情况选用。

懒人目录

UserCF VS ItemCF

这两者的对比是一个一直比较常见的问题，此处我引用的是项亮在推荐系统实践中的对比，大家可以大概了解一下。

上面对比都比较直接的把不同的点给大家说出来，至于具体的原因并不一定清楚，而其实在现实应用中，方法的选择更应该看的是对原理的理解和把握而不是方法本身，所以此处我给大家展开谈一下。

要谈两者的区别，肯定是要从两者的核心谈起。UserCF（后面简称U了）是从用户角度出发，去推荐有共同爱好的用户喜欢的产品，而ItemCF（后面简称I了）则是从物品出发的，推荐和当前用户喜欢的产品相似的产品，因此其实可以看到U会反映出一个小群体的喜好甚至是热点，而I则能够较好地描述一个用户的当前甚至是历史兴趣进一步，可以体现U的一种社会化，群体化，而I则更倾向于个性化，这就是两者最核心的区别。

两者没有任何的优劣之分，重点在于两者各有倾向，都有非常适合的应用场景。以新闻为例，对新闻而言，对个性化要求并不高，大家看新闻都倾向于看热点新闻，如微博热搜等，几乎所有用户都爱关注，而在个性化而言，用户的喜欢是比较粗粒度的，如娱乐新闻，体育新闻等，所以此处使用U会比使用I更合适；而另一方面，使用U优于I还有另一层原因，就是Item更新的速度更为快速，ItemCF需要维护的物品相关度表在瞬息万变的新闻市场需要有大量的更新，而用户相比之下则比较稳定，因此使用I会导致相关度表维护成本加大，需要大量的更新，因此使用U更为合适。

具体用哪个要因地制宜，下面是我总结的一些选择的规则，供大家参考，如有错误或者遗漏欢迎补充：

- 用户和物品想比，哪个数量多或者更新快，则建议使用另一个的CF；
- 用户个性化需求强烈，则ItemCF合适，且ItemCF的可解释性较强
- 要求对新用户友好，ItemCF比较合适
- 对时效性要求高，如上面提到的新闻，则用U

协同过滤拓展

哈利波特问题

哈利波特问题是指，在哈利波特最火的时间下，绝大部分的人都会买哈利波特的书，此时ItemCF就会认为他与其他的书都很相似，此时会出现一些问题，首先看看在0-1的情况下评判相似度的公式：

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

N(j)表示喜欢j产品的用户集合，|N(j)|表示喜欢j产品的用户个数，如果物品j非常流行，分子会无限接近\|N(j)\|，尽管分母做了控制，但是可以发现物品j和大量物品都很接近，此时很可能就一直推哈利波特，尽管用户已经买过等，其他商品也被挤下去，再者，有些用户其实并不喜欢。因此，最简单的方式就是使用惩罚，给热门商品进行降权，从而得到解决。

$$w_{ij} = \frac{\left|N(i) \cap N(j)\right|}{\left|N(i)\right|^{1-\alpha}\left|N(j)\right|^{\alpha}}$$

可以看到，alpha=0.05时就是上面的公式，通过增加alpha就能够达到惩罚的效果，在分母，alpha的变大能够使|N(j)|更被看重，从而另其被除的更多。

在ItemCF会出现的问题在UserCF中同样会出现，很多人都会买《新华字典》但是不一定会买《推荐系统实践》，此时前者会影响两人的相似度，因此可以进行削弱，此处的可进行降权，降权的方式如下：

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

N(u)和N(v)表示用户u和v喜欢的产品集合。要避免值之间的差距太大，最简单的方式就是用log进行平滑，而避免数据异常，此处加一。

有关实时性的讨论

实时性的实现方式很多，且协同过滤也不是要求实时性的主力（这个在后面会谈到），但是这里还是通过一篇论文谈一下这块的思路。由于协同过滤的计算量大，尤其在用户众多的情况，训练时间很长，所以一般是定时训练，这种情况下很多时候，协同过滤召回的可能是用户上周喜欢的东西，而但凡是线上的项目，都需要考虑实时性，此时就需要考虑用更为实时的方式。腾讯在2015年提出了一种实时方法以解决这种实时性的问题，核心的思路在于相似度的计算的调整。

首先就是用户的行为各异，难以进行描述，在本文就采用下面的模式来进行计算，用显式的打分来代替行为，例如点击、浏览、购买、分享、评论等，为这些行为打分，然后取里面的最高分作为用户对该物品的打分，例如点击的喜欢程度当然不如购买的高，购买的喜欢程度不如好评的高，等等。这时候对用户而言，两

个商品的相似度就可以用这两个商品的打分的最小值作为参照。（符号比较明显，我这里就不赘述啦，我尽量把原理讲清楚）

$$\text{co-rating}(i_p, i_q) = \min(r_{u,p}, r_{u,q})$$

既然得到了一个用户对两个商品的相似度，那对这两对商品，他们在总体用户中的相似度也就有了，用余弦距离表示就是下面的形式。

$$sim(i_p, i_q) = \frac{\sum_{u \in U} \min(r_{u,p}, r_{u,q})}{\sqrt{\sum r_{u,p}} \sqrt{\sum r_{u,q}}}$$

上面是4.1.2中提到的内容，然后在4.1.3进行了拓展，仔细观察可以发现其实上面的公式完全一样，但是在4.1.3讲了对实时性的实现。实时性的体现在，经过对用户行为的监控，能够表达对用户喜爱程度的变化，此时整个相似度就会实时更新，以惠及所有有关的商品和相似度，实现更高级别的实时性。

协同过滤的召回作用

协同过滤能够很好地通过行为识别用户的喜好程度，借助相似度来按计算用户对未知商品喜好程度，以达到推荐的效果，然而可以看到，里面使用的信息非常有限，只有简单的用户喜好信息，而忽略了各种诸如地点、时间、用户性格等的多重信息，而产品的信息也被忽略，产品品牌、规则等，此时推荐内容非常受限，因此只是一种“粗排”，在比较大且成熟的推荐系统项目中，后续还需要经过精排和各种策略来实现精确排序，因此此处可以说协同过滤可以起到的是一个召回作用。

协同过滤的召回作用可以体现在下面几点：

- 协同过滤得到的内容更加有可解释性
- 协同过滤的得不到的内容，大概率是用户不喜欢的内容，可缩小精排范围
- 由于是粗排，则并不一定就要很高的准确性和实时性

因此协同过滤大部分情况下是一个召回作用。当然的，在一些初步形成的系统中，协同过滤也能有较好的效果，不容忽略。

协同过滤篇章结束，后续内容敬请期待。

参考文献

[1] 项亮，《推荐系统实践》