推荐系统冷启动问题

推荐系统需要根据用户的历史行为和兴趣预测用户未来的行为和兴趣，因此大量的用户行为数据就成为推荐系统的重要组成部分和先决条件。

冷启动问题简介

- **用户冷启动** 用户冷启动主要解决如何给新用户做个性化推荐的问题。当新用户到来时，我们没有他的行为数据，所以也无法根据他的历史行为预测其兴趣，从而无法借此给他做个性化推荐。
- **物品冷启动** 物品冷启动主要解决如何将新的物品推荐给可能对它感兴趣的用户这一问题。
- **系统冷启动** 系统冷启动主要解决如何在一个新开发的网站上设计个性化推荐系统，从而在网站刚发布时就让用户体验到个性化推荐服务这一问题。

利用用户注册信息

- 获取用户的注册信息；
- 根据用户的注册信息对用户分类；
- 给用户推荐他所属分类中用户喜欢的物品。

对于每种特征f，计算具有这种特征的用户对各个物品的喜好程度$p(f,i)$，可以简单地定义为物品 i 在具有f的特征的用户中的热门程度：

$$p(f,i) = |N(i) \cap U(f)|$$

其中$N(i)$是喜欢物品 i 的用户集合，$U(f)$是具有特征 f 的用户集合。

可以比较准确地预测具有某种特征的用户是否喜欢某个物品。但是，在这种定义下，往往热门的物品会在各种特征的用户中都具有比较高的权重。也就是说具有比较高的$N(i)$的物品会在每一类用户中都有比较高的$p(f,i)$。给用户推荐热门物品并不是推荐系统的主要任务，推荐系统应该帮助用户发现他们不容易发现的物品。因此，我们可以将$p(f,i)$定义为喜欢物品 i 的用户中具有特征f的比例：

$$p(f,i) = \frac{|N(i) \cap U(f)|}{|N(i)| + \alpha}$$

选择合适的物品启动用户的兴趣

在新用户第一次访问推荐系统时，不立即给用户展示推荐结果，而是给用户提供一些物品，让用户反馈他们对这些物品的兴趣，然后根据用户反馈给提供个性化推荐。很多推荐系统采取了这种方式来解决用户冷启动问题，需要解决的首要问题就是如何选择物品让用户进行反馈。

- **比较热门** 如果要让用户对一个物品进行反馈，前提是用户知道这个物品是什么东西。
- **具有代表性和区分性** 启动用户兴趣的物品不能是大众化或老少咸宜的，因为这样的物品对用户的兴趣没有区分性。
- **启动物品集合需要有多样性** 在冷启动时，我们不知道用户的兴趣，而用户兴趣的可能性非常多，为了匹配多样的兴趣，我们需要提供具有很高覆盖率的启动物品集合，这些物品能覆盖几乎所有主流的用户兴趣。

基于用户对物品评分的方差度量这群用户兴趣的一致程度。如果方差很大，说明这一群用户的兴趣不太一致，反之则说明这群用户的兴趣比较一致。令$\sigma_u \in U'$为用户集合U'中所有评

分的方差，度量一个物品的区分度$D(i)$：

$$D(i) = \sigma_{u \in N^+(i)} + \sigma_{u \in N^-(i)} + \sigma_{u \in \overline{N}(i)}$$

分别为喜欢物品i的用户、不喜欢物品i的用户和不知道物品i的用户（即没有给i评分的用户）。如果这 3 类用户集合内的用户对其他的物品兴趣很不一致，说明物品i具有较高的区分度。

首先会从所有用户中找到具有最高区分度的物品 i，然后将用户分成 3 类。然后在每类用户中再找到最具区分度的物品，然后将每一类用户又各自分为 3 类，也就是将总用户分成 9 类，然后这样继续下去，最终可以通过对一系列物品的看法将用户进行分类。而在冷启动时，我们从根节点开始询问用户对该节点物品的看法，然后根据用户的选择将用户放到不同的分枝，直到进入最后的叶子节点。

利用物品的内容信息

物品冷启动需要解决的问题是如何将新加入的物品推荐给对它感兴趣的用户。

UserCF 算法对物品冷启动问题并不非常敏感。因为，UserCF 在给用户进行推荐时，会首先找到和用户兴趣相似的一群用户，然后给用户推荐这一群用户喜欢的物品。在很多网站中，推荐列表并不是给用户展示内容的唯一列表，那么当一个新物品加入时，总会有用户从某些途径看到这些物品，对这些物品产生反馈。那么，当一个用户对某个物品产生反馈后，和他历史兴趣相似的其他用户的推荐列表中就有可能出现这一物品，从而更多的人就会对这个物品产生反馈，导致更多的人的推荐列表中会出现这一物品，因此该物品就能不断地扩散开来，从而逐步展示到对它感兴趣用户的推荐列表中。可以考虑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。

ItemCF 算法的原理是给用户推荐和他之前喜欢的物品相似的物品。ItemCF 算法会每隔一段时间利用用户行为计算物品相似度表，在线服务时 ItemCF 算法会将之前计算好的物品相关度矩阵放在内存中。因此，当新物品加入时，内存中的物品相关表中不会存在这个物品，从而 ItemCF 算法无法推荐新的物品。解决这一问题的办法是频繁更新物品相似度表，但基于用户行为计算物品相似度是非常耗时的事情，主要原因是用户行为日志非常庞大。而且，新物品如果不展示给用户，用户就无法对它产生行为，通过行为日志计算是计算不出包含新物品的相关矩阵的。为此，我们只能利用物品的内容信息计算物品相关表，并且频繁地更新相关表。

物品的内容可以通过向量空间模型表示，该模型会将物品表示成一个关键词向量。对物品 d，它的内容表示成一个关键词向量如下：

$$d_i = \{(e_1, w_1), (e_2, w_2), \cdots\}$$

如果物品是文本，可以用信息检索领域著名的 TF-IDF 公式计算词的权重：

$$w_i = \frac{TF(e_i)}{\log(DF(e_i))}$$

向量空间模型的优点是简单，缺点是丢失了一些信息，比如关键词之间的关系信息。不过在绝大多数应用中，向量空间模型对于文本的分类、聚类、相似度计算已经可以给出令人满意的结果。

在给定物品内容的关键词向量后，物品的内容相似度可以通过向量之间的余弦相似度计算：

$$w_{ij} = \frac{d_i * d_j}{\sqrt{\|d_i\| \|d_j\|}}$$

```
function CalculateSimilarity(D):
    for di in D:
```

```
        for dj in D:
            w[i][j] = CosineSimilarity(di, dj)
    return w
```

可以首先通过建立关键词—物品的倒排表加速这一计算过程：

```
function CalculateSimilarity(entity-items):
    w = dict()
    ni = dict()
    for e,items in entity_items.items():
        for i,wie in items.items():
            addToVec(ni, i, wie * wie)
            for j,wje in items.items():
                addToMat(w, i, j, wie, wje)
    for i, relate_items in w.items():
    relate_items = {x:y/math.sqrt(ni[i] * ni[x]) for x,y in relate_items.items()}
```

向量空间模型在内容数据丰富时可以获得比较好的效果，LDA 作为一种生成模型，有 3 种元素，即文档、话题和词语。初始化和迭代两部分：首先要对 z 进行初始化，而初始化的方法很简单，假设一共有 K 个话题，那么对第 i 篇文章中的第 j 个词，可以随机给它赋予一个话题。同时，用 NWZ(w,z)记录词 w 被赋予话题 z 的次数，NZD(z,d)记录文档 d 中被赋予话题 z 的词的个数。

```
foreach document i in range(0,|D|):
    foreach word j in range(0, |D(i)|):
        z[i][j] = rand() % K
        NZD[z[i][j], D[i]]++
        NWZ[w[i][j], z[i][j]]++
        NZ[z[i][j]]++
```

在初始化之后，要通过迭代使话题的分布收敛到一个合理的分布上去。

```
while not converged:
    foreach document i in range(0, |D|):
        foreach word j in range(0, |D(i)|):
            NWZ[w[i][j], z[i][j]]—
            NZ[z[i][j]]—
            NZD[z[i][j], D[i]]—
            z[i][j] = SampleTopic()
            NWZ[w[i][j], z[i][j]]++
            NZ[z[i][j]]++
            NZD[z[i][j], D[i]]++
```

在使用 LDA 计算物品的内容相似度时，我们可以先计算出物品在话题上的分布，然后利用两个物品的话题分布计算物品的相似度。比如，如果两个物品的话题分布相似，则认为两个物品具有较高的相似度，反之则认为两个物品的相似度较低。计算分布的相似度可以利用 KL 散度：

$$D_{KL}(p||q) = \sum_i p(i) ln \frac{p(i)}{q(i)}$$

其中 p 和 q 是两个分布，KL 散度越大说明分布的相似度越低。

发挥专家的作用

既没有用户的行为数据，也没有充足的物品内容信息来计算准确的物品相似度。那么，为了在推荐系统建立时就让用户得到比较好的体验，很多系统都利用专家和机器学习相结合的方法来进行解决系统冷启动问题。