**冷启动问题简介**

用户冷启动：新用户没有历史数据，如何个性化推荐

物品冷启动：新物品上架，怎么推荐

系统冷启动：如何在一个新开发的网站(既没有用户，也没有用户行为)上设计个性化推荐

解决方案：

对于新用户：

提供非个性化的推荐【热门排行榜】

粗粒度的个性化【用户注册时的年龄、性别等】

利用社交信息【获取用户在社交网站上的好友】

获取用户反馈【例如新浪微博的“你喜欢”】

对于新物品：

利用内容信息【内容相似的老物品】

对于新系统：

引入专家知识，通过一定的高效方式迅速建立起物品的相关度表

**利用用户注册信息**

人口统计学信息/用户兴趣的描述/从其他网站导入的用户站外行为数据

(1) 获取用户的注册信息；

(2) 根据用户的注册信息对用户分类；

(3) 给用户推荐他所属分类中用户喜欢的物品。

基于用户注册信息的推荐算法其核心问题是计算每种特征的用户喜欢的物品，这一变量用表示。



其中 是喜欢物品i的用户集合， 是具有特征f的用户集合

为了惩罚热门物品，改写为：

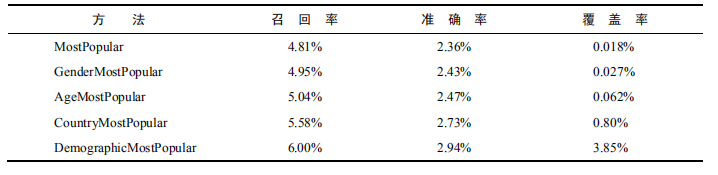


是一个比较大的数。

书中针对两个常用数据集给出了两个例子。

关于BookCrossing数据集，用实验证明了上面的第二个公式比第一个公式有着更高的推荐准确率。

关于Lastfm数据集，对比了不同的粒度和算法。



**选择合适的物品启动用户的兴趣**

解决用户冷启动问题的另一个方法是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。

在提供物品时，需要注意这些原则：

1. 比较热门。如果用户都没听说过，那就没得玩了
2. 具有代表性或区分度。老少咸宜的物品，喜欢也没什么象征意义。
3. 具有多样性。提供的物品要具有很高的覆盖率。

设计选择启动物品集合的算法——决策树。



上面这个式子描述了物品i的区分度。

是喜欢物品i的用户集合，是不喜欢物品i的用户集合，是没有对物品i评分的用户集合。是喜欢i的用户对其他物品评分的方差，其他以此类推。即，如果喜欢、不喜欢、不知道物品i的用户，这三类集合内的用户对其他物品的兴趣很不一致，那物品i具有很高的区分度。

在决策树中，Nadav Golbandi先从所有用户中找到最高区分度的物品，然后将用户分成3类，在每类用户中找到最具区分度的物品，以此类推。

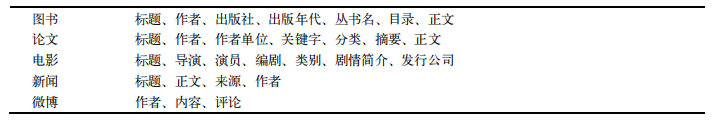
**利用物品的内容信息**

物品冷启动在UserCF和ItemCF算法中有着不同的影响。

UserCF不会受到特别严重的影响，这是因为它的推荐依据是“相似用户”。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。

而ItemCF则会受到冷启动的致命影响。因为ItemCF的原理是给用户推荐和他之前喜欢的物品相似的物品，每隔一段时间(一般是一天)，ItemCF算法重新计算并更新推荐列表，因此，当新物品加入时，内存中的物品相关表不会存在这个物品。由于用户行为日志非常庞大，基于用户行为计算物品相似度是非常耗时的，无法做到频繁更新。

在前面我们说过，ItemCF在考察物品相似程度的时候，并不是依靠内容来考察，而是依靠用户行为的相关性来考察。因此，我们换一个思路，只好通过利用物品内容信息来计算相关表并频繁更新。物品内容信息大致有以下几类。



一般来说，可以用向量空间模型来表示物品的内容。

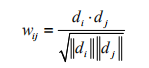


e和w分别是关键词和权重。如果物品是文本，就可以引用著名的TD-IDF公式来计算词的权重。



TF-IDF公式的介绍：<https://blog.csdn.net/asialee_bird/article/details/81486700>

在确定了关键词向量之后，可以通过向量之间的余弦相似度计算物品的内容相似度。



另外需要说明的是，虽然内容相似度计算简单，能频繁更新，而且能够解决物品冷启动的问题，但协同过滤还是有必要存在的。

这是因为内容过滤算法只考虑了物品的内容特征，忽视了用户行为，从而也忽视了物品的流行度以及用户行为中所包含的规律，所以它的精度比较低，但结果的新颖度却比较高。因此，如果用户的行为强烈受某一内容属性的影响，那么内容过滤算法是优秀的。但是并不是所有的数据集都具有这样的特点。

向量空间模型在内容数据丰富时可以获得比较好的效果，但在一些应用(特别是自然语言处理)中，我们要关注的不仅仅是文本，还有话题。比如“推荐系统的动态特性”和“基于时间的协同过滤算法研究”这两个题目，虽然文本完全不同，但是话题却是相似的。

代表性的话题模型有LDA。下面介绍一下生成模型LDA。

**LDA主题模型**

以下内容参考<https://blog.csdn.net/chenshulong/article/details/79049661>和<https://blog.csdn.net/selinda001/article/details/80446766>

在考虑LDA的原理之前，首先我们来看一个统计学习方法里的概念：生成模型&判别模型。

生成模型&判别模型这部分内容参考

<https://blog.csdn.net/dulingtingzi/article/details/80267726>，感谢原作者的辛苦总结。

在监督学习中，我们要完成这样一个任务，在给定输入X的前提条件下，给出一个输出Y。注意这里的X和Y表示的是系统模型的输出。

那么，比方说我的分类任务是给定一张图片，判断里边是大象还是狗，我有两种思路来完成这个任务。

1. 学习两个模型出来，分别对应大象和狗各有什么特点和分布，然后判断该图片的特征更像大象还是更像狗
2. 我们知道，大象的鼻子长，狗的鼻子短，我们只要划分一个阈值，如果鼻子的长度超过这个阈值就划分为大象，否则划分为狗。

第一种对应于生成模型，第二种对应于判别模型。

判别方法：由数据直接学习决策函数Y=f(X)或条件概率分布P(Y|X)作为预测模型，即判别模型。判别方法关心的是对于给定的输入X，应该预测什么样的输出Y。

生成方法：由数据学习联合概率分布P(X,Y), 然后由P(Y|X)=P(X,Y)/P(X)求出概率分布P(Y|X)作为预测的模型。该方法表示了给定输入X与产生输出Y的生成关系。

换言之，判别模型直接面对分类，不能反映训练数据本身的特性，但它寻找不同类别之间的最优分类面，反映的是异类数据之间的差异，直接面对预测，往往学习的准确率更高，由于直接学习P(Y|X)或Y=f（X），从而可以简化学习。生成模型则可以还原出联合概率分布，这样的方法之所以称为生成方法，是因为模型表示了给定输入X产生输出Y的生成关系。

生成算法尝试去找到底这个数据是怎么生成的（产生的），然后再对一个信号进行分类。基于你的生成假设，那么那个类别最有可能产生这个信号，这个信号就属于那个类别。判别模型不关心数据是怎么生成的，它只关心信号之间的差别，然后用差别来简单对给定的一个信号进行分类。

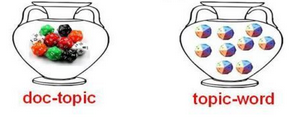
LDA就是一种生成模型。

Ok，现在我们知道，我们手里有一个文档的集合，我们来模拟一下这些文档都是怎么来的。

爱因斯坦曾经说过“上帝不会掷骰子”，现在我们就假设有一个全知全能的上帝，他面前有两个骰子罐，通过投掷这些骰子，生成一个个的词，最终拼成一个文档。

为了建模文档生成的过程，我们有两个问题：掷哪些？怎么掷？

回答第一个问题，第二个罐子里装的是topic-word骰子，每一个topic-word骰子都表示一个给定的topic对应的一些词汇；第一个罐子里装的是doc-topic骰子，它表示一篇文档内包含的一些topic。



上帝先随机地从第二个罐子里抽取K个topic-word骰子，编号为1到K。

然后，在每生成一篇新的文档之前，上帝从第一个罐子里抽取一个doc-topic骰子，然后反复随机抛掷这个骰子，得到一个topic编号z，选择K个topic-word骰子中编号为z的那个，投掷这个骰子，于是得到一个词。

为了讲清楚这个问题，我们需要理解，上面的比方中，每一个“骰子”，对应的是一种概率分布。

也就是说，在生成一系列文档之前，我们先选择了有哪些topic可能在我们的文档集合中出现(从第二个罐子里抽取K个topic-word骰子)，然后，对于生成的每一篇文档，我们逐个地产生单词。如何产生呢？对于整篇文档，先选择在这K个topic中我们要用到哪些(从第一个罐子里抽取一个doc-topic骰子)，再对于每一个词，选择这些topic中我们用哪个(得到一个topic编号z)。最终我们得到了一个确定的topic，然后，再对于这个topic中的若干单词，选择一个出来作为文档的下一个单词(投掷这个骰子，于是得到一个词)。

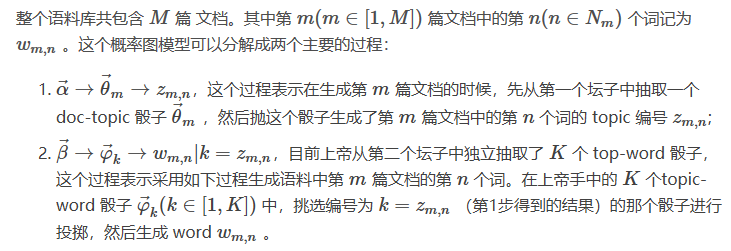
这就是文档的生成过程了。

这一过程是完全随机的吗？

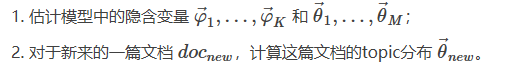
从读者的角度来讲，一篇文章的单词当然不可能是独立同分布出现的。从建模的角度来讲，如果是完全随机无规律的，那还统计学习个什么呢？

于是我们的任务就是，在文档生成过程满足以上假设的前提下，找到那个生成的规律。

下面的截图来自于<https://blog.csdn.net/chenshulong/article/details/79049661>，它描述了文档生成的过程。



于是，LDA模型的求解目标主要包含两个方面：



这也就是生成文档的规律。

根据我们在高中已经学过的知识，上面文档生成过程的第二步，也就是“掷骰子”这件事，满足多项式(multinomial)分布。



由于和都是隐含变量不容易观察到，需要借助可以观察的变量来求它。我们说好啦，我们观察一下和，反正它们是和、有关的多项式分布，倒着估计一下分布参数，不就得到、了吗？思路是这样的没错，可是是topic呀。我们能观察到的仅仅是文档中有哪些词而已。和、一样，都是难以观察的隐变量。

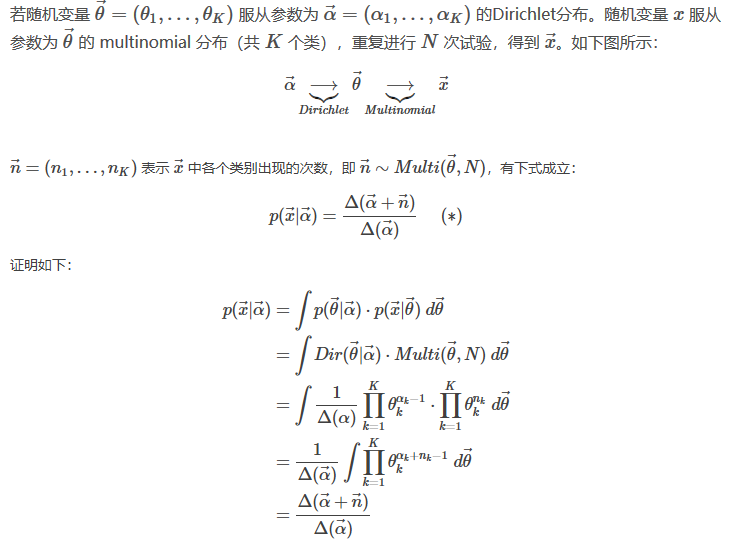
一种可行的方法是，我们首先计算的后验概率，然后对该后验概率分布进行采样。将得到的样本作为隐含变量的观察值。

解释一下上面这句话。要估计的，是和，与、有关的，是和。是不能被观察到的，但可以被观察到。如果我们能算出，那么根据我们观察到的，就建立起计算和的桥梁。

所以可以被算出来吗？幸运的是，可以。

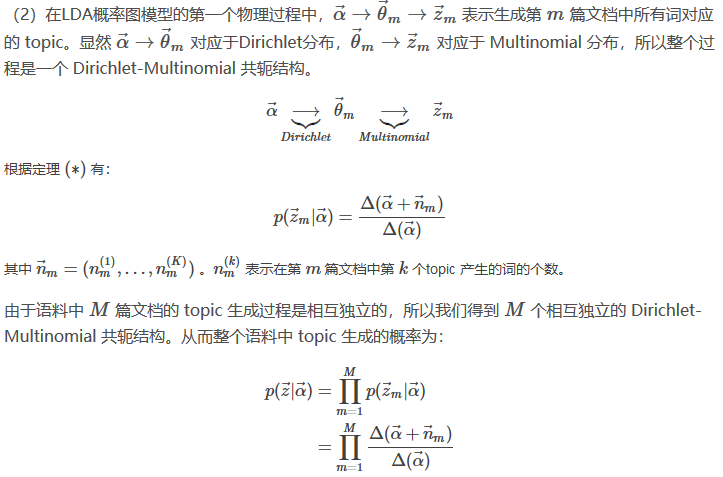
这里先介绍一个定理。

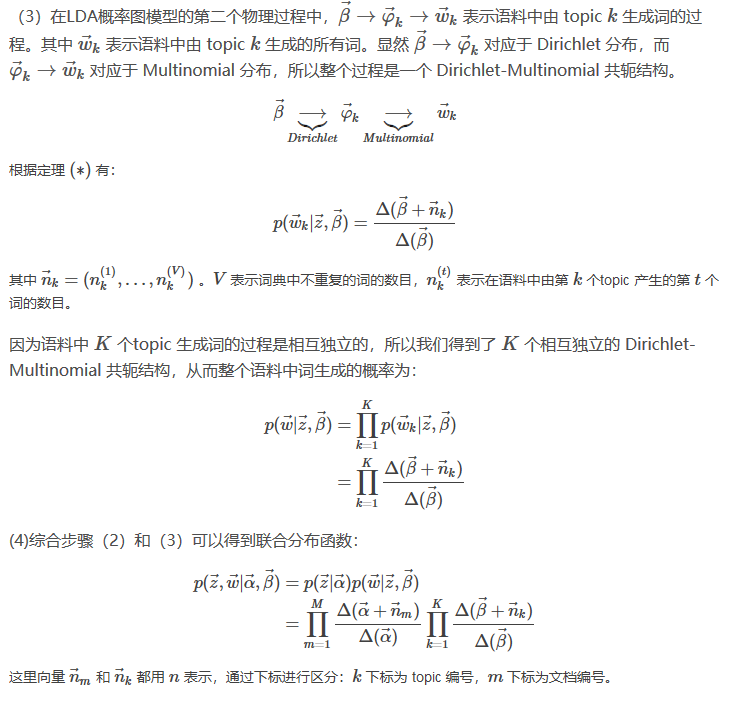
(数学推导不想打了，截图来自<https://blog.csdn.net/chenshulong/article/details/79049661>，公式拼写不易，再次感谢原作者)

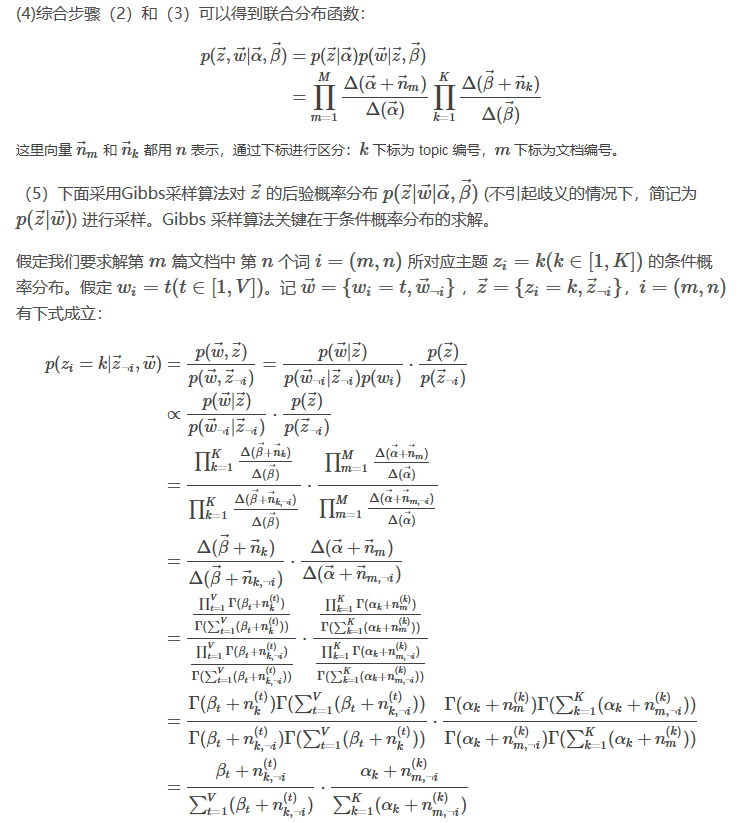


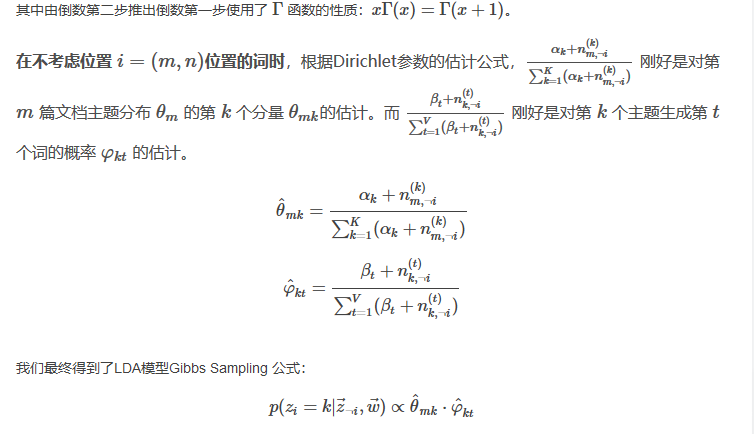
数学推导没有什么可说的。关于Dirichlet分布的意义，可以参见<什么是狄利克雷分布？狄利克雷过程又是什么(https://www.zhihu.com/question/26751755)？>

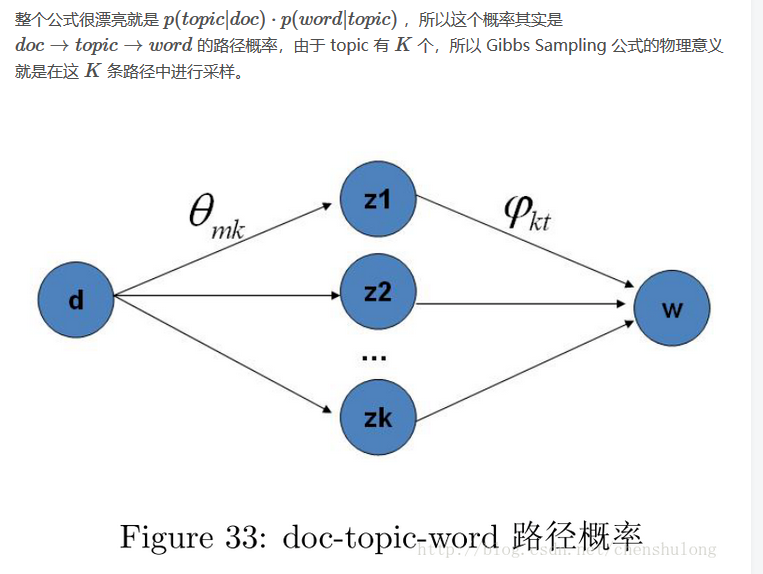
然后我们回头看LDA的生成。







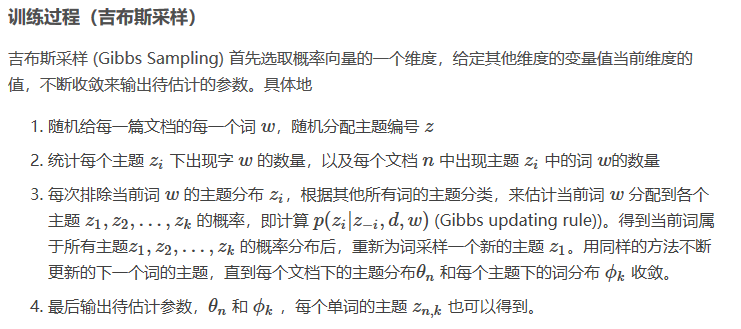




上面的数学推导一步步看都是没有问题的。最终得到的公式其实很简洁，就是*p*(*topic*|*doc*)⋅*p*(*word*|*topic*)。现在，分析其实已经形成了一个闭环，可以求出我们想要的参数了。

重新梳理一下思路：我们已经知道的表达式，现在对文档中每个词对应的topic随机初始化，对每一个词，就可以通过上面推导的公式给出的分布。根据这个分布，我们给这个词重新赋一个合理的、不是随机得来的topic。接着我们再去更新下一个词。在这个过程中，每一个词对应的topic在变化，在变化，*p*(*topic*|*doc*)和*p*(*word*|*topic*)在变化。当这些量都收敛时，我们就估计得到了想要的参数。

为什么这些量会收敛？因为在假设的前提下，文档就是根据这些规律生成的，我们现在所做的，是找到这些规律中的参数。只要我们根据数据不断地自适应更新，就可以收敛到一个比较优秀的解，也就是我们要找的生成模型。



Ok，回到推荐系统的问题上来。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。

**发挥专家的作用**

很多推荐系统在建立时，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进行标注。这方面的代表系统是个性化网络电台Pandora和电影推荐网站Jinni。

说起来这可能是个比较“笨”的方法，也就是引入人工的因素，用domain knowledge来干扰结果，从而减少冷启动的影响。