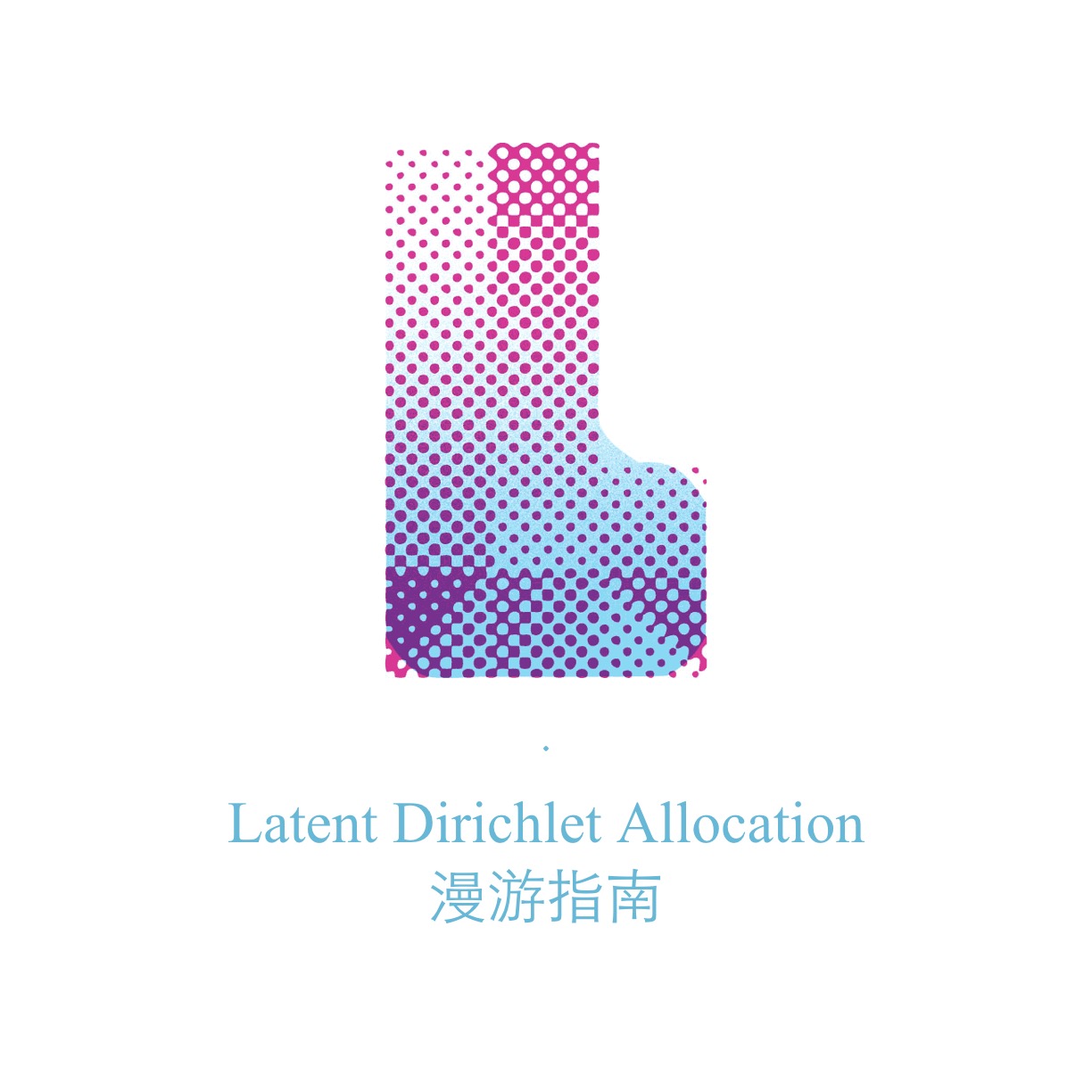
****

**马晨(**[**sharpstill@163.com**](mailto:sharpstill@163.com)**)**

**Version 1.0**

**2015年7月25日**

**前言**

LDA算法是主题模型领域非常著名的算法，值得深入研究应用，该算法也有很深刻的数学背景和技术启发。曾经有哲人说：万物皆数。我个人是个十分喜欢数学，喜欢算法，热爱技术的人，非常想从算法中寻找人工智能的永恒之道。我尤其记得19世纪的数学家赫尔曼.汉克尔说的好：

就大多数学科而言，一代人摧毁的正是另一代人所建造的，而他们所建立的也必将被另一代人所破坏。只有数学不同，每一代人都是在旧的建筑物上加进新的一层。

所以说，数学的价值还具有一种永世不灭的恒久性，其他学科的时尚潮流往往随着时代的变迁被人遗忘，那些旨在改变世界的理想，最终往往变成思想垃圾。而只有数学和算法与此不同。

我们探究前人伟大的成果时，就能体会到奥利弗.亥维赛的精辟论说：“逻辑能够很有耐性，因为它是永恒的”。

我在选择分析Latent Dirichlet Allocation(LDA)这个算法课题时，我考虑了很多因素，首先，该算法是已经被学术界和工业界广泛接受的；其次，该算法能带来更多的新技术启示；最后,该算法能为您的工作，您的研究带来最具实用性的技术启发。

LDA算法恰好满足了这个条件。

虽然网上已经有许多分析LDA算法的博客文章，但是这部作品对比网上的博文，仍旧有其**独特的价值**：

**1.这部作品理论与实践并重:**网上的同类文章非常零散，理论推导部分也缺乏**关键细节**，这部作品的每一条公式都由作者手把手为您推理（每一条公式都有详细的解释和备注），并且按照初学者的思路娓娓道来，从逻辑链条上打通算法的整个环节，让用户有醍醐灌顶的认识。并且在**实践部分**，作者以多年的工作实践经验为基础，精选了6个实现简单但又有巨大应用价值的LDA的应用方法，将成为读者未来工作实践不可多得的资料。

**2. 这部作品饱含了作者的独到见解：**这部作品最大的特色是从理论分析开始就有包含着许多作者自己独到的理解和分析，从不同角度完美解释算法的整个流程。

**3. 读者可以在这部作品各取所需：**有的工程师对于算法推导不是很感兴趣，这种情况下可以跳过前几章，直接从第4章读LDA算法怎么具体实现。如若未来有兴趣研究LDA算法的来龙去脉时，可以再来看前几章的理论推导部分。如果读者对大数据环境下的LDA感兴趣，包括怎么在Hadoop、Spark上实现LDA算法可以直接看第5章。

**4. 这部作品首次将LDA算法引入大数据时代：**大数据时代最大的特色就是信息爆炸，各种文本数据，用户生成（UGC）数据也变得非常庞大，网上查阅到的LDA算法资料大部分都是不能应对大数据环境的，这部作品的第5章深入浅出地讲解了大数据环境下怎么实现并行化的LDA算法。

**章节安排：**

第1章为相关背景介绍，介绍了算法知识的来源：从18世纪的欧拉讲到剑桥大学的David Blei。

第2章和第3章为算法的理论分析阶段：第2章为LDA算法的前置知识，为LDA做了理论工具上的准备。

在第2章力求做到关键证明不遗漏，这样就可以与后面第3章的LDA算法推导构成一个完整的推理链条！这一章的有些证明需要一些简单的微积分知识，但如果读者忘记了所有基础的微积分知识的话，那么看不懂某条证明，就请姑且相信我的推理是对的吧，跳过去往后看，日后再复习。

第3章为LDA算法推导部分，用严谨的数学推导和清晰的讲解（每个公式都做了清晰的标注），让读者认识该推理方法。

第4章为实现和应用部分，用伪代码方式庖丁解牛，讲解代码实现的精髓，然后结合作者多年的工作实践，写了该算法的几个最具实用价值的应用。

第5章为并行化，在大数据如火如荼的今天，要想大规模运行LDA算法，就要靠并行化技术了，这一章从2个算法的改进形式讲解了该技术的并行化，并且可以放在目前最流行的spark大数据引擎上运行。

本文是一个指南指引性质的作品，仍未完善完美，有许多地方未涉及到，比如变分推断推导LDA算法的过程等，未来这部作品还会完善补充，第二版会增加新的章节。本书还有些地方可能仍有疏漏，本人水平有限，如果大家发现纰漏，请及时联系我修正。**希望这部作品日臻完美。**

**作者：马晨 2015.7.7**

目录

[第1章 背景 1](#_Toc432238146)

[第2章 前置知识 5](#_Toc432238147)

[2.1 gamma函数 5](#_Toc432238148)

[2.2 二项分布(Binomial distribution) 6](#_Toc432238149)

[2.3 beta分布(beta distribution) 7](#_Toc432238150)

[2.4 多项分布(multinomial distribution) 11](#_Toc432238151)

[2.5 狄利克雷分布(dirichlet distribution) 12](#_Toc432238152)

[2.6 共轭先验分布(conjugacy prior) 13](#_Toc432238153)

[2.6.1 从二项分布到beta分布 14](#_Toc432238154)

[2.6.2 从多项分布到Dirichlet分布 16](#_Toc432238155)

[2.7 总结 18](#_Toc432238156)

[参考文献 18](#_Toc432238157)

[第3章 LDA的Gibbs Sampling推导 20](#_Toc432238158)

[3.1 unigram假设 20](#_Toc432238159)

[3.2 Latent Dirichlet Allocation介绍 23](#_Toc432238160)

[3.3 马尔可夫链 🡪 Metropolis-Hasting 🡪 Gibbs Sampling 26](#_Toc432238161)

[3.3.1 马尔可夫链(markov chain) 26](#_Toc432238162)

[3.3.2 Metropolis-Hasting算法 28](#_Toc432238163)

[3.3.3 Gibbs Sampling 30](#_Toc432238164)

[3.4 伟大的采样公式： Collapsed Gibbs Sampling采样公式推导 31](#_Toc432238165)

[3.5 总结 37](#_Toc432238166)

[参考文献 37](#_Toc432238167)

[第4章 实现与应用 39](#_Toc432238168)

[4.1 实现 39](#_Toc432238169)

[4.2 应用 46](#_Toc432238170)

[4.2.1 相似文档发现 46](#_Toc432238171)

[4.2.2 自动打标签 48](#_Toc432238172)

[4.2.3 LDA与LR（逻辑斯蒂回归）结合做新闻个性化推荐系统 49](#_Toc432238173)

[4.2.4 topic rank[10] 51](#_Toc432238174)

[4.2.5 word rank 54](#_Toc432238175)

[4.2.6 文章质量评分算法 56](#_Toc432238176)

[4.2.7 总结 60](#_Toc432238177)

[参考文献 60](#_Toc432238178)

[第5章 并行化 62](#_Toc432238179)

[5.1 AD-LDA 62](#_Toc432238180)

[5.2 spark-LDA 63](#_Toc432238181)

[5.2.1 切分块 64](#_Toc432238182)

[5.2.2 选择 66](#_Toc432238183)

[5.2.3 计算和合并 66](#_Toc432238184)

[5.2.4 总结 67](#_Toc432238185)

[参考文献 67](#_Toc432238186)

[第6章 变分贝叶斯的启蒙 69](#_Toc432238187)

[6.1 前置知识 69](#_Toc432238188)

[6.1.1 指数分布族(exponential family) 69](#_Toc432238189)

[6.1.2 再谈数学期望 72](#_Toc432238190)

[6.1.3 进一步观察指数分布族 75](#_Toc432238191)

[6.1.4 拉格朗日的杰作——拉格朗日乘数法 77](#_Toc432238192)

[6.1.5 指数分布族的一点深入的思考 80](#_Toc432238193)

[6.2 变分法的启蒙 81](#_Toc432238194)

[6.2.1 改变世界的方程：欧拉-拉格朗日方程 82](#_Toc432238195)

[6.2.2 E-L方程的两种降阶形式 85](#_Toc432238196)

[6.2.3 E-L方程与拉格朗日乘数法的联姻 86](#_Toc432238197)

[6.2.4 变分贝叶斯法的推导 93](#_Toc432238198)

[参考文献 93](#_Toc432238199)

[第7章 LDA的变分贝叶斯法 95](#_Toc432238200)

[7.1 Latent Dirichlet Allocation的另一个视角 95](#_Toc432238201)

[7.2 分析模型 97](#_Toc432238202)

[7.3 推导 100](#_Toc432238203)

[7.3.1 启蒙 100](#_Toc432238204)

[7.3.2 变分目标函数 102](#_Toc432238205)

[7.3.3 下界（lower bound） 103](#_Toc432238206)

[7.3.4 下界展开 105](#_Toc432238207)

[7.3.5 变分推断 108](#_Toc432238208)

[7.4 de Finetti’s theorem 109](#_Toc432238209)

[参考文献 109](#_Toc432238210)

[参考文献 111](#_Toc432238211)

*飞机和飞鸟形态、结构和原理都不相同，但都能飞翔，人工智能未来也许如此。*

# 背景

LDA算法使用的全部知识的渊源可以追溯到18世纪的欧拉，欧拉（Leonhard Euler ，1707年4月15日～1783年9月18日），瑞士数学家。欧拉一生贡献颇丰，1734年，欧拉解决巴塞尔问题就立即出名了，巴塞尔问题就是问式(1‑1)的值是多少。



(‑)

这个问题困扰了几个世纪的数学家，当时的数学家只知道该级数的值小于2，但不知道具体精确值，欧拉准确的推导出该式的值=，欧拉的方法聪明而新颖，他创造性将有限多项式的观察推广到无穷级数，并假设相同的性质对于无穷级数也是成立的：



(‑)

欧拉最后的发现是令人惊奇的，**这个数字在于圆周率无关的场合中出现了，这足以说明数学之中、自然之中、冥冥之中存在着某些神秘的联系。虽然以现代数学的眼光来看，欧拉的证明还不严密。但作为第一个（富有创造性的）证明，欧拉的这个证明永远有着其宝贵的价值。欧拉的另一个发现就是发现了gamma函数 ，该函数后被广泛应用于概率论，这个函数也是本文的主角之一。



图 1‑1 [Euler](http://baike.baidu.com/link?url=iIq4UAvChdqgvGhUy8cEilqX-pDPyzbvbcJ9fSLT-rym-g2EptGcyVGFtrKMsPNOz7XXZlhjvzFSAspowYbdMot3lbtoMD9PqEmNiDymj_mWDMfyYxvD314HHXLSpYx4RsywhZhqoiJM0L0ajuDzZq)

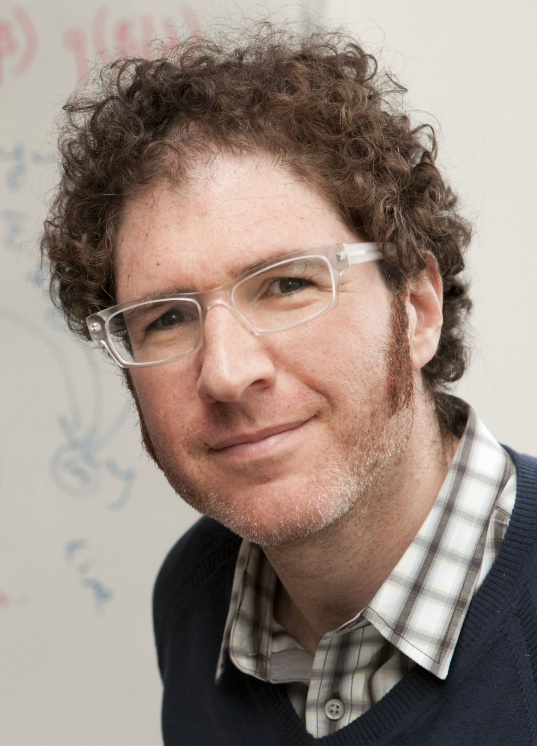
作为算法标题之一的Dirichlet， wiki一下，一个19世纪的人映入了我们的眼帘，Dirichlet（1805～1859）德国数学家，生与现德国 Duren（当时属法国），卒于哥廷根。他是解析数论的奠基者，也是现代函数观念的定义者。在本文中该数学家的主要贡献是Dirichlet分布。



**图 1‑2** [**Peter Gustav Lejeune Dirichlet**](http://en.wikipedia.org/wiki/Peter_Gustav_Lejeune_Dirichlet)

但是这还不是故事的全部，说到底19世纪的时候还没有发明计算机，LDA应该不是这哥们发明的，于是继续search，最后查明英国剑桥大学的David M.Blei是最初LDA论文的作者。Blei同学借用了Dirichlet Distribution，而创造了Latent Dirichlet Allocation。

下面这张照片的blei以PLSA（LDA之前的另一个概率模型）为基础，加上了贝叶斯先验，从而诞生了LDA算法，LDA算法最初的论文使用的是变分EM方法训练（Variational Inference）。该方法较为复杂，而且最后训练出的topic主题非全局最优分布，而是局部最优分布。后期发明了Collapsed Gibbs Sampling方法，推导和使用都较为简洁。Blei及其LDA算法正式的介绍如下：



**图 1‑3** [**David Blei**](http://en.wikipedia.org/wiki/David_Blei)

Latent Dirichlet Allocation是Blei等人于2003年提出的基于概率模型的主题模型算法，LDA是一种无监督机器学习技术，可以用来识别大规模文档集或语料库中的潜在隐藏的主题信息。该方法假设每个词是由背后的一个潜在隐藏的主题中抽取出来。

对于语料库中的每篇文档，LDA定义了如下生成过程（generative process）：

1. 对每一篇文档，从主题分布中抽取一个主题；

2. 从上述被抽到的主题所对应的单词分布中抽取一个单词；

3. 重复上述过程直至遍历文档中的每一个单词。

LDA认为每篇文章是由多个主题mix混合而成的，而每个主题可以由多个词的概率表征。所以整个程序的输入和输出如表 1‑1所示：

表 ‑1 LDA算法的输入与输出

|  |
| --- |
| **算法输入：**分词后的文章集（通常为一篇文章一行）  主题数K，超参数和 |
| **算法输出：**1.每篇文章的各个词被指定(assign)的主题编号：tassign-model.txt  2.每篇文章的主题概率分布：theta-model.txt  3.每个主题下的词概率分布：phi-model.txt  4.程序中词语word的id映射表：wordmap.txt  5.每个主题下概率排序从高到低top n特征词：twords.txt |

如果你想使用一下LDA算法，我建议你从Gibbs LDA++代码起步使用。（<http://gibbslda.sourceforge.net/>），你用一下，就会发现该算法使用方式还算傻瓜，并且生成的结果文件也挺规则，根据manual一看便懂。输入分词后的文件，一个文章一行，输出其中看到每个主题规则文件.twords如下格式所示：

|  |  |
| --- | --- |
| Topic 0th: | Topic 1th: |
| 食品 0.022937  不合格 0.006634  加多宝 0.006634  检测 0.006433  包装 0.004403  儿童 0.004038  抽检 0.004016  王老吉 0.003759  批次 0.003483  检验 0.003379  红罐 0.003342  样品 0.003312 | 农业 0.022957  农村 0.022404  农民 0.014940  土地 0.013734  粮食 0.007909  农产品 0.006644  耕地 0.005823  集体 0.005067  农户 0.004852  种植 0.004674  流转 0.004177  农业部 0.003693 |

# 前置知识

本章所描述的工具和线索在后期LDA算法的采样公式推导中会全部明了，关于为什么需要使用这些知识要素，这里面有很长的渊源和历史，比如在概率论和数理统计中，gamma函数被广泛使用，而在最终的LDA采样公式中，你会发现，gamma函数被神奇地约去消失了。我们在后面的章节中可以看到，LDA算法精妙之处在于用令人屏息的洞察力作为纽带，将全部零散的部件组合在了一起。

## gamma函数

所谓的gamma函数其实就是**阶乘的函数形式**，上过小学我们都知道n!=1⋅2⋅3…n，如果我问你3的阶乘是多少，你立即回答出1⋅2⋅3=6，但是如果我问你0.5阶乘是什么，如果没有gamma函数就无法回答了，欧拉经过不懈努力，终于发现阶乘的更一般的函数形式gamma函数，直接给出：



(‑)

可以直接算出

* 也可以算出



由于所以

设，则。当时，；当时，

则换元后

令

做二重积分换元法，做极坐标变换，令，，

则当t和u的区域D都为到，即积分区域为整个坐标轴的时侯

r半径的范围为0到，而**的范围为绕坐标轴一圈从0到 2**

使用雅可比行列式：



故

则又因为I的被积函数大于0，则I>0，最后得

**(2‑2)**

* 接下来验证

，做分部积分法（）



也正因为如此，

(‑)

## 二项分布(Binomial distribution)

在概率论中，二项分布即重复n次独立的伯努利试验。在每次试验中只有两种可能的结果（成功/失败），每次成功的概率为p，而且两种结果发生与否互相对立，并且相互独立，与其它各次试验结果无关，事件发生与否的概率在每一次独立试验中都保持不变，则这一系列试验总称为n重伯努利实验，当试验次数为1时，二项分布就是伯努利分布。

在给出二项分布之前，我们来做一个例子，假设你在玩CS这个游戏，你拿着狙击枪，敌人出现你打中敌人的概率是p，打不中敌人的概率是1-p，那么敌人**第一次出现你没打中而第二次出现你打中的概率是 (1-p)⋅p。如果敌人出现了n次，而你打中了其中的k次，而不确定具体在哪k次（第1次，还是第4次？），这样从n次中任取k次的次数是****，而这不确定的k次打中敌人的概率是：****，通过这个例子我们便得知了二项分布的概率。**

二项分布的概率密度函数是：



(‑)

## beta分布(beta distribution)

在概率论中，beta分布是指一组定义在区间(0,1)的连续概率分布，有两个参数和，且>0。

Beta分布的概率密度函数是：



(‑)

随机变量X服从参数为的beta分布通常写作：。

这个式子中分母的函数称为函数。

* 这里我们来证明一个重要的公式，该公式中的关系在LDA算法Gibbs Sampling采样公式中也有使用，这个关系也就是**函数和Gamma函数的关系**（该公式也被称为第一型欧拉积分）：



(‑)

**证明方法1：**

****

****

设,则

当的时候，；当的时候，



故原式=



x

t

t=x

图 2‑1 积分区域

在图 2‑1的上半部分（阴影部分）为积分区域，则交换积分次序：



再令，则的时候，；的时候，

按换元规则，则原式







，

所以

**证明方法2:**

****



t

s

t=s

t=-s



这便是坐标变换后新坐标的积分区间



* Beta分布的期望

如果，则





这个式子右边的概率对应到，

有，带回到E(p)式：

，化简后



这说明，beta分布的均值可以用来估计

对于后面2.6节提到的Dirichlet分布也有相似结论：

如果，可以证明：

(‑)

这个结论在LDA算法做完Gibbs sampling后，估计和时用到。

## 多项分布(multinomial distribution)

多项分布[[1]](#footnote-1)是二项分布的推广扩展，在n次独立试验中每次只输出k种结果中的一个，且每种结果都有一个确定的概率p。多项分布给出了在多种输出状态的情况下，关于成功次数的各种组合的概率。

举个例子，投掷n次骰子，这个骰子共有6种结果输出，且1点出现概率为p1，2点出现概率p2，…多项分布给出了在n次试验中，骰子1点出现x1次，2点出现x2次,3点出现x3次，…，6点出现x6次。这个结果组合的概率为：



(‑)

也可以用gamma函数表示（这个写法的形式和Dirichlet分布相似）：



(‑)

下面我们通过一个例题加深对多项分布的印象:

* 问题：同时投掷5枚骰子，出现两对点数一样的概率是多少？

解：现在先把问题简化成特定投掷到2个一点，2个二点，1个三点的概率是多大？

X1到X6表示六个点的出现次数之和为5，则：

 这里0!=1



再考虑，现在X1到X6其中2个取2,1个取1的种类有多少种？



先不考虑2,2,1三者顺序时共有种取法；再考虑下2,2,1三者交换顺序有3种，因为两个2先后交换仍为2,2。

所以X1到X6其中2个取2,1个取1的种类有

最后答案概率=

## 狄利克雷分布(dirichlet distribution)

dirichlet分布是beta分布在多项情况下的推广，也是多项分布的共轭先验分布（共轭先验分布在2.6节讲）。dirichlet分布的概率密度函数如下：

，这里

(‑)

二项分布和多项分布很相似，Beta分布和Dirichlet 分布很相似，而至于“Beta分布是二项式分布的共轭先验概率分布，而狄利克雷分布（Dirichlet分布）是多项式分布的共轭先验概率分布”这点在下文中说明。

另一个重要的公式是:

(‑)

为了简便表达，公式中引入了一个新的三角形符号的希腊字母（念“德尔塔”Delte）代表函数的多项版本，这个公式的结构和证明**相似于**上文中“函数和Gamma函数的关系—公式(2‑6)”，这个证明留给读者来完成。从此，公式中凡是出现积分中连乘时，就要像巴甫洛夫试验中流着口水的狗一样警觉，建立起“可以换成gamma函数”的条件反射。

我们下面给出公式(2‑11)的证明。

证明：

## 共轭先验分布(conjugacy prior)

In Bayesian probability theory, if the posterior distributions p(** |*x*) are in the same family as the prior probability distribution p(**), the prior and posterior are then called conjugate distributions, and the prior is called a conjugate prior for the likelihood function.[[2]](#footnote-2)

所谓的共轭，只是我们选取(choose)一个函数作为似然函数(likelihood function)的prior probability distribution，使得后验分布函数[[3]](#footnote-3)(posterior distributions)和先验分布函数形式一致。比如Beta分布是二项式分布的共轭先验概率分布，而狄利克雷分布(Dirichlet分布）是多项式分布的共轭先验概率分布。为什么要这样做呢？这得从贝叶斯估计[4]谈起：

根据贝叶斯规则，后验分布=似然函数\*先验分布



(‑)

*参数估计是一个重要的话题。对于典型的离散型随机变量分布：二项式分布，多项式分布；典型的连续型随机变量分布：正态分布。他们都可以看着是参数分布，因为他们的函数形式都被一小部分的参数控制，比如正态分布的均值和方差，二项式分布事件发生的概率等。因此，给定一堆观测数据集（假定数据满足独立同分布），我们需要有一个解决方案来确定这些参数值的大小，以便能够利用分布模型来做密度估计。这就是参数估计！*

*对于参数估计，一直存在两个学派的不同解决方案。一是频率学派解决方案：通过某些优化准则（比如似然函数）来选择特定参数值；二是贝叶斯学派解决方案：假定参数服从一个先验分布，通过观测到的数据，使用贝叶斯理论计算对应的后验分布。先验和后验的选择满足共轭，这些分布都是指数簇分布的例子。[[4]](#footnote-4)*

简而言之，假设参数也是变量而非常量，而且在做试验前已经服从某个分布p()（来源于以前做试验数据计算得到，或来自于人们的主观经验），然后现在做新试验去更新这个分布假设。如果不知道最大似然估计(Maximum Likelihood)的概念，参见附录。

### 从二项分布到beta分布

注意二项分布概率密度函数为，将参数去掉，变成形式：，再加上归一化因子B(**,**)（注意这个归一化因子含有k和n，但绝不含p），变为beta分布：，beta分布前一项函数是确保beta分布是归一化(normalized)。

* 求证：beta分布确实是二项分布的共轭先验分布

证明：

1. 似然函数二项分布：



1. 前驱分布beta分布如下：

，其中

1. 由于prior distribution \* likehood = post distribution







(‑)

这就可以看到后验分布(post distribution)又变为beta分布，也就是和先验(prior distribution)一致了，因此我们称之为共轭(conjugacy)。

观察到后验和先验都是beta分布，但变为了，超参数变了。如果以后有新增的观测值，后验分布又可作为先验分布来进行计算。具体来讲，在某一个时间点，有一个观测值，此时可以得到后验，之后，每一个观测值的到来，都以之前的后验作为先验，乘以似然函数后，得到修正后的新后验。在这每一步中，其实我们不需要管什么似然函数，我们可以将后验分布看作是以代表x=1出现“次数”的参数s和代表x=0出现“次数”的参数f为参数的beta分布：当有一个新的x=1的观测量到来的时候，s=**+1,f=**，即**的值相应的加1;否则s=**, f=** + 1即**的值加1。所以这也就是超参数(**,**)又被称之为伪计数(pseudo count)的原因。

我们可以把上面性质所表示的方法看作为序列方法（sequential approach），该方法是贝叶斯观点中很自然得到的学习方法。它非常适合实时学习场景。在某一时刻，有一个观测数据，或是一小批量数据，在下一批观测数据到来之前我们就可以丢弃它们，因为我们可以在一开始的小批量数据中得到我们的后验分布模型，当有新的一批数据到来时，只需要更新这个模型就够了。一个实时学习应用场景是：在所有数据到来之前，预测就必须通过之前稳定到达的一部分数据流来做出预测。注意到，因为这种序列方法不需要将所有数据都载入内存，因此他在大数据的应用将非常有效。当然，之前频率学派所使用的最大化似然函数方法也可以转换成这种序列式方法的。

另外，我们做先验分布的目的是估计参数，比如投掷硬币试验，我们需要根据已有的观测数据，估计下一次试验硬币的正面结果概率是多少。



(‑)

这正是beta分布的均值，结合beta分布的均值公式可以得到：

(‑)

### 从多项分布到Dirichlet分布

通过观察多项式分布的形式（），我们选取先验分布的形式为（保留带概率pk的项）：



(‑)

这里代表这个分布的超参数（或伪计数），。由于有这两个条件的限制(用等价于pk​)，因此{}之上的分布是K-1维度：

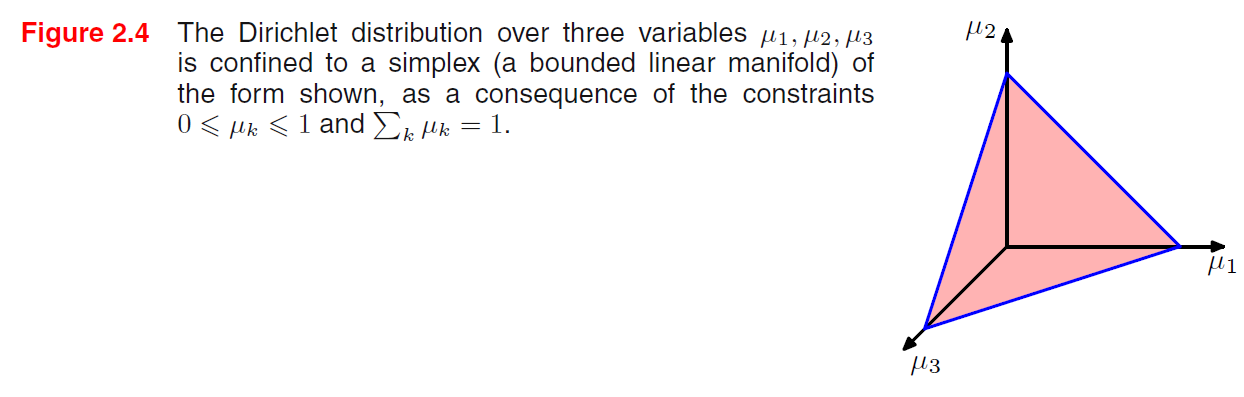


图 ‑2 节选自《Pattern Recognition and Machine Learning》[7] p77

图 2‑2的含义是假设一共有3个概率，这3个概率**的范围被限制在红色框所围的坐标轴内。

将(2‑16)式加入系数归一化(normalized),便可得到Dirchlet分布的概率密度表达式：

,这里

由于，这里mk是:做n次实验输出k编号结果的次数，这个mk也叫充分统计量。跟上一节一样，这里的mk就是根据新试验去更新超参数。注意，f函数内自变量的p1到pk-1而非pk，这是由于p1到pk-1一旦确定下来，而因为概率之和=1（ 以及），因此前面这些“自由变量”一旦被确定，最后一个变量=1-(p1+p2+...+pk)固定下来了而不“自由”了。

* 求证：dirichlet分布确实是多项分布的共轭先验分布

证明:

1. 似然函数多项分布



1. 先验分布dirichlet分布



1. 由于



又因为，用这个式子替换掉分母：

则原式

(‑)

## 总结

1. 贝叶斯学派采用给参数赋予先验分布，并使得先验与后验共轭，通过求后验均值来得到参数的估计，频率学派通过某个优化准则比如最大化似然函数来求得参数的估计；不管是哪个学派思想，都要用到似然函数，注意到频率学派所使用的似然函数是N次伯努利实验下的似然函数，但贝叶斯学派所使用的似然函数是二项式分布形式的似然函数（二项式分布是N次伯努利实验中出现事件A的次数的分布）。

2. 当拥有无限数据量时（beta分布式中s和f都趋向于无穷，dirichlet分布式中m趋向于无穷），贝叶斯方法和频率学派方法所得到的参数估计是一致的。当在有限的数据量下，贝叶斯学派的参数后验均值的大小介于先验均值和频率学派方法得到参数估计。比如在抛硬币实验中，当数据量有限时，先验均值为0.5，后验均值将会比先验大，比频率学派得到参数估计小。

3. 随着观测数据的增多，后验分布曲线越来越陡峭（越来越集中），即方差越来越小（后验方差总比前验方差小），当数据量无穷大时，方差趋近于0，即随着数据越来越多，后验的不确定性在减小，如图 2‑3所示。

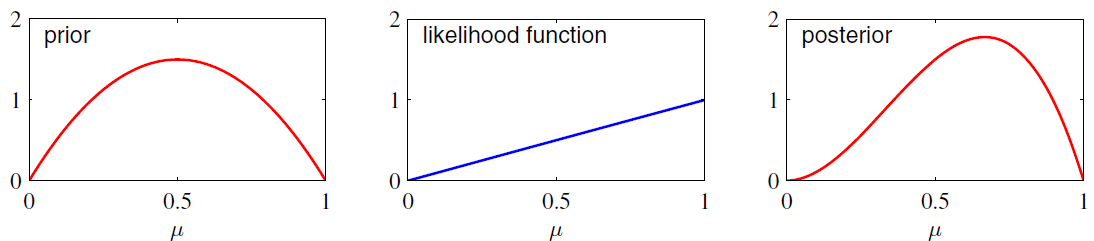


图 2‑3 后验分布曲线

## 参考文献

1. Christopher M. Bishop. Pattern Recognition And Machine Learning. Springer. 2007.10.1. p.77
2. 最大似然估计<http://www.cnblogs.com/liliu/archive/2010/11/22/1883702.html>
3. 陈希孺. 概率论与数理统计. 中国科学技术大学出版社. 2009.2.1
4. 从二项式分布到多项式分布-从Beta 分布到Dirichlet 分布：hi.baidu.com/leifenglian/item/636198016851cee7f55ba652#713670-tsina-1-11778-128ff9f28d958dae738be418601ffbcd

# LDA的Gibbs Sampling推导

## unigram假设

假设有N个服从i.i.d(独立同分布)的单词从一个多项式分布(multinomial)中抽取，在N个词中，我们关注vi的发生次数为n(t)，那么则记为w~Mult(w|p)，该文档生成的概率是



(‑)

其中nt为该单词在文章中出现的次数,V为字典中的单词个数。在这里其中每个词的概率。由于这种方法不考虑文章内单词间的顺序，因此被称之为词袋模型（bag of words）。可以注意到上面的(3‑1)式子忽略了第2章的多项式公式(2‑8)式中的多项式系数参数，因为其词袋模型的假设忽略了文档中词的连续信息（sequential information）。得到了生成一篇文章的概率，可以将生成一篇文章的概率扩展到整个语料集。这里可以引入一个新的概念：概率图模型，来画出这种模型，正如图 3‑1所示，图中被涂色的w表示可观测变量，方框表示重复抽取的次数，N表示一篇文档中总共N个单词，M表示M篇文档。也就是说，重复抽取M篇文档，每个文档抽取N个单词，这样的生成模型生成了整个语料（corpus）。

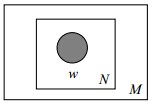


图 ‑1基于unigram的概率图模型

这个生成文档的概率可以用投掷骰子的游戏来模拟：上帝拿出一枚骰子，骰子有V个面，每个面代表一个词，每个面的概率是，投掷N次骰子每个面产生的次数分别是次，然后计算生成语料库的概率，见图 3‑2。

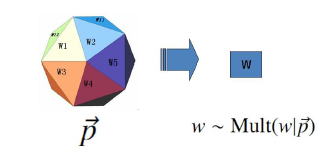
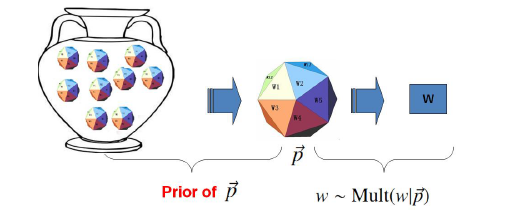


图 ‑2上帝投掷V个面的骰子生成文档

现在来引入Dirichlet分布作为多项分布的先验分布，则文本单词概率。贝叶斯学派下，我们不知道上帝到底用哪个骰子来投掷，所以从一个服从Dirichlet分布的坛子中抽取一个骰子，然后投掷生成文档，图 3‑3形象地解释了这个过程。这也正是贝叶斯学派和频率学派对于待估计参数解释的差别所在，频率学派认为这个待估计的参数（概率p）是定值，而只是我们不知道这个值是多少，需要用各种手段去估计（比如极大似然估计、矩估计等），而贝叶斯学派则认为该参数也是个随机变量，服从某种分布。



**图 3‑3贝叶斯观点下Unigram Model**

根据dirichlet分布函数密度公式，我们得到了超参数(hyper parameter)（或伪计数(pseudo count)）**的似然函数：



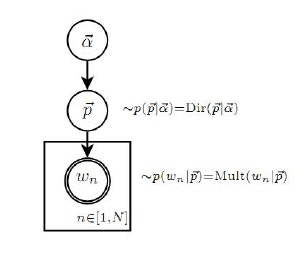
(‑)

因为，也即根据贝叶斯公式(2‑12)，所以推出的后验分布：



(‑)

将这个贝叶斯观点下的unigram model画成概率图模型如图 3‑4所示。



**图 3‑4 Unigram Model的概率图模型[[5]](#footnote-5)**

**在贝叶斯框架下，参数p可以得到估计，因为我们已经有了后验分布，**根据第2章的公式(2‑7)，**于是：**



(‑)

也就是说对于每一个pi，我们做如下参数估计即可：

上述参数估计很直观：每个单词产生的概率估计值是对应事件的先验的伪计数和数据中的计数的和在整体计数中的比例。进一步可以产生整体文本语料概率为：[3]







 //dirichlet 换成(利用公式(2‑11))



**(3‑5)**

## Latent Dirichlet Allocation介绍

Latent Dirchlet Allocation是Blei等人于2003年提出的一种概率主题模型，LDA是一种无监督机器学习模型，可以用来识别语料库中的潜在的主题信息。该方法假设每个词是由背后的一个潜在的主题中抽取出来。

LDA的图模型如下：

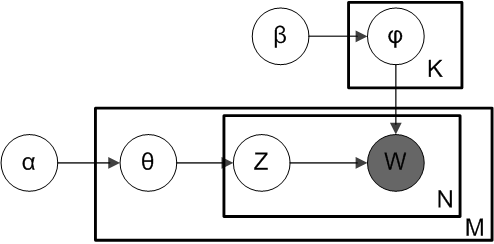


图 3‑5 LDA图模型表示

这个图模型表示法有时也称作“盘子表示法”（plate notation）。图中的阴影圆圈表示可观测变量（observed variable），非阴影圆圈表示潜在变量（latent variable），箭头表示两变量间的条件依赖性（conditional dependency），方框表示重复抽样，重复次数在方框的右下角。

M代表训练语料中的文章数；

K代表设置的主题数；

V代表训练语料的词表单词数；

是一个M\*K的矩阵，代表第m篇文章的主题分布；

是一个K\*V的矩阵，代表编号为k的主题之上的词分布；

是每篇文档的主题分布的先验分布Dirichlet分布的参数（也被称为超参数），其中；

是每个主题的词分布的先验分布Dirichlet分布的参数（也被称为超参数），其中；

w是可被观测的词；

z是每个对于被观测的词的潜在的主题分配。

对于语料库中的每篇文档，LDA定义了一个生成过程（generative process）。

Smoothed版本LDA模型的标准生成过程的描述如下：

1.选取，这里；

2.选取，这里；

3.对于**每个单词**位置Wi,j，这里，；

选取一个topic主题从

选取一个word词从

如果没看懂就换一种比喻表述方法，爱因斯坦曾说：上帝不扔骰子。我们这里假设我们的机器上帝是扔骰子的，所以我换一种投掷骰子的说法解释这个过程。

LDA的逻辑与3.1节生成文档的逻辑类似，这里我引用《LDA数学八卦》的一幅图如下说明LDA如何生成文档：

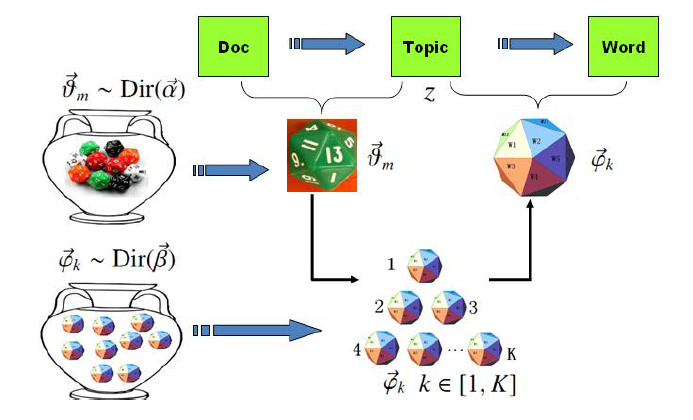


图 3‑6 LDA模型[[6]](#footnote-6)

上帝先在服从分布的坛子抽取的共K个topic🡪word骰子。每次生成一篇新的文档前，从服从的坛子中抽取出一个doc🡪topic骰子，然后重复以下步骤：

1. 投掷这个doc🡪topic骰子，得到一个topic编号z。
2. 从的共K个topic🡪word骰子中选择编号为z的那个，投掷这枚骰子，于是得到一个词w。

更佳的表述如下：

1. ，这个过程表示在生成第**m**篇文档的时候，先从服从的坛子中抽中一个doc-topic骰子，然后投掷这枚骰子生成了文档中第**n**个词的**主题(topic)编号**。[[7]](#footnote-7)此为Dirichlet- Multinomial共轭。

1. ，这个过程表述了如下动作生成语料中第m篇文档的第n个词：从先前服从分布的坛子已抽取好的的共K个topic🡪word骰子之中，挑选编号为的那个骰子进行投掷，然后生成word:。[[8]](#footnote-8)此为Dirichlet- Multinomial共轭。

简单来说，第i个过程就是，第ii个过程就是，根据条件概率的基本公式可得：



(‑)

而LDA的目标是找出每个词后潜在的主题，所以为了达到这个目标，需要计算后验概率(3‑7)。



(‑)

难点在于分母，简单分析一下分母的情况便可得出：公式(3‑7)**难以直接计算**。按照离散分布上边缘概率的处理方法：假设我们有概率，可以计算在所有c的可能值上求和，就消去c，而仅计算。所以文档中一个单词的概率是：



(‑)

由此可得公式(3‑7)的分母，也即整个语料集的所有单词的概率：



(‑)

式(3‑9)中的n是语料中所有单词实例的总数，因为计算分母陷入了项的难题，这个离散状态空间 （discrete state space）太大了以至于无法列举出来（enumerate）[3]。Thomas L.Griffiths等人从统计物理学的Potts Model获得灵感，开发出了LDA的蒙特卡洛马尔可夫（Monte Carlo Markov Chain）求解方法。这就是下文3.3节和3.4节所介绍的内容。

## 马尔可夫链 🡪 Metropolis-Hasting 🡪 Gibbs Sampling

在正式推导LDA的Gibbs Sampling采样公式之前，读者有必要了解为什么需要这样推导，做到知其然知其所以然。[[9]](#footnote-9)

### 马尔可夫链(markov chain)

马尔可夫链条通俗说就是根据一个转移概率矩阵去转移的随机过程（马尔可夫过程）[[10]](#footnote-10)，该随机过程在Page Rank算法中也有使用。如图 3‑7所示：

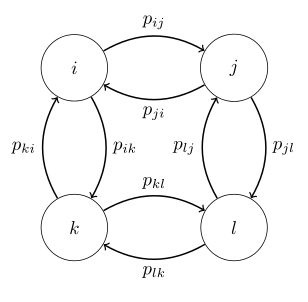
[](http://en.wikipedia.org/wiki/File:Kolmogorov_criterion_dtmc.svg)

图 3‑7马尔可夫转移图

通俗理解：这里的每个圆环类似一个岛屿(状态)，比如i到j的概率是pij，每个节点的出度概率之和=1。现在假设我要根据这个图去转移，那么首先需要把这张图“翻译”成如下的矩阵：



即该矩阵的第i行第j列pij表示从i岛屿走到j岛屿的概率。

这就是转移矩阵，我现在身处的位置用一个向量表示0=(i,j,k,l)。

现在假设我站在i岛屿，我的位置向量0=(1,0,0,0)，第一次转移，也就是相当于 ，相乘后向量1每一项为小于1的小数。就是说我有Pii的机会比率留在原始i岛屿，有Pij的机会到达j岛屿，……；第二次转移时，以我刚才的位置向量为基础得到；以此类推……

**有这么一种情况**，我的位置向量在若干次转移后会达到一个稳定状态，再转移向量也不变化了，这个状态称之为平稳分布状态\*，这个情况需要满足一个条件：这就是**Detailed Balance**。

下面通过一个例子深刻认识**Detailed Balance**：

首先要明确：一个马尔可夫链条要成为reversible markov chain 才有可能达到Detailed Balance条件，何为reversible markov chain?也就是满足Detailed Balance的 马尔可夫链条。是不是有点鸡生蛋蛋生鸡的味道。换一种说法，reversible markov chain也就是其转移概率满足Kolmogorov’s criterion的链条，这个Kolmogorov’s criterion即表示以下含义，在图 3‑7中：。

也就说一个封闭的环中，一个方向的概率连乘积=反过来方向的概率连乘积。

例子：我们用假设构造这样一个转移矩阵：

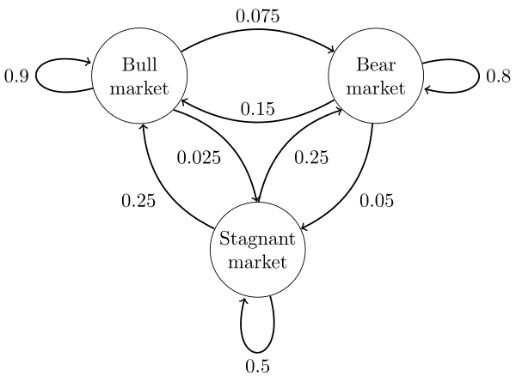


图 3‑8 detail balance的转移矩阵图

先构造出转移矩阵，

假设我们的初始向量为=(1, 0, 0)，现在用计算机软件计算矩阵乘法得到 × P 1000次后的stationary distribution = (0.625, 0.3125, 0.0625)。注意这个平稳分布有且只有一个（**是唯一的**）。

所谓的**detailed balance**就是：



(‑)

这里的是平稳分布(stationary distribution)的那个\*。代到这个例子里：

假设



则

所以



所以得出结论，detailed balance成立。

有了detailed balance， 马尔可夫链就可以收敛[[11]](#footnote-11)了，我们可以根据detailed balance去sampling生成一些点，使这些点收敛到stationary distribution。因此这些点就是满足这个stationary distribution概率分布的点。收敛在这里称之为burning-in，在burning-in之前的若干迭代步骤生成的点被抛弃掉。下面我们再来验证下：

由于，因此两边同时对i的所有可能值求和，也就是：

最后一步由于矩阵的每一行和=1（每个节点出度概率之和=1）。因此这个推导暗示了

在这个例子里，我们取j=2，也就是，则



另外这样的马尔可夫链还需要满足两个性质：1.irreducible， 马尔可夫链的所有状态节点需要可以彼此通信，不能有割裂的孤岛。2.aperiodic，非周期性，链条不会在特定的周期内在两个节点来回循环。

### Metropolis-Hasting算法

结合LDA算法，有了上述detail balance条件，受到这个平稳分布不再变化的启发，我们的**终极目标自然是：要使用一个马尔可夫链条，sample[[12]](#footnote-12)出一系列的状态点，使其最终的平稳分布状态就是我们给定的那个联合概率分布（该联合概率就是LDA里的文档集被生成(generate)的概率）。**

**MH算法的目的：**是根据一个需求的(desired distribution)概率分布**P(x)**生成一系列样本状态点（因此，这个算法可以生成任意的概率分布）。为了达到这个目的，该算法使用马尔可夫过程去到达一个平稳分布(x)(stationary distribution)，以使(x)=P(x)。

**MH算法推导：为了达到这个平稳分布，有两个条件需要满足：(1)满足detail balance：**（2）该平稳分布必须唯一（Ergodicity可遍历[[13]](#footnote-13)）。

MH算法的方式是设计一个马尔可夫过程（通过构造转移概率）来满足上述两个条件，用P(x)代替(x)后(注意斜体字*P*为转移矩阵)，现在重点来分析detail balance：



我们的方法是进一步将转移概率分解为两个子步骤：proposal distribution(建议概率)和acceptance distribution（接受概率）。**建议概率**是说我们给出状态x后转移到x’的条件概率，而**接受概率**是接受状态x’的条件概率。

所以转移概率，代入公式(3‑10)（detail balance），并整理可得：，也就是说这样我们得到了一个接受比率：“从状态到的接受概率”与“到的接受概率”的比率（接受率）。如果这个比率大于1，则我们按照建议概率转移到，否则停留在原地不动（拒绝接受建议）。Metropolis的选择是进一步将上述接受率改为如下形式（比率大于1就=1，上限为1）：



(‑)

接着就可以写出MH算法：

|  |  |
| --- | --- |
| **算法3.1 Metropolis-Hasting算法步骤** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 初始化，随机选择一个初始状态点x  for i=0 to N-1 do //第i个sample点为x(i)  根据随机选择一个新状态x’  计算接受率  生成0~1之间随机数*u*  if(*u* < **) then{ //若**<1，按照**为概率接受x’    }else{    } |

MH算法非常简单，但要小心设计建议概率g，其实有时候考虑MH算法的特殊情况（Gibbs Sampling is a special case of MH）让问题变得更简单。

### Gibbs Sampling

考虑我们目标要得到的是一个维度是n的多维的概率分布，**如果采用设置多维概率分布P里的完全条件概率(full conditionals)作为建议概率(proposal)**，那么接受率就会始终=1，一直接受。让我们来证明这一点[15]：

full conditionals的公式如下：



(‑)

第i次sample的点转移的proposal distribution：

由于*x*是n维变量，当j=1,2,…,n每一维度轮流循环时

{

我们如果设置

那么就可以造成**





}

**所以一旦在联合概率的full conditionals可用时，可以采用n维向量里轮流每一个维度循环的方式来迭代达到平稳状态。**

而为了得到full conditionals，就**要先写出联合概率（这也就引出了下一节的文本生成的联合概率）**，再用公式(3‑12)的方式求得。

因此通用的Gibbs Sampling迭代算法如算法3.2所示：

|  |  |
| --- | --- |
| **算法3.2 Gibbs Sampling算法步骤** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 初始化，随机选择一个初始状态点x1:n(0)  for i=0 to N-1 do{ //第i个sample点为x(i),x的下标j为x的第j维  sample  sample  …  sample  …  sample  } |

## 伟大的采样公式： Collapsed Gibbs Sampling采样公式推导

鉴于3.2节的式(3‑7)的难以求得，假如利用Collpased Gibbs Sampling，就可以使用3.3.3节的full conditional公式（式(3‑12)）写出：，用此公式来模拟。

现在我们想推导出Gibbs Sampling的采样公式，在做Gibbs Sampling公式推导之前最重要的一件事是写出**整个文本训练集生成的联合概率**（也就是按照3.2节的公式(3‑6)）：



(‑)

注意到第一项不包含**，第二项不包含**，所以两项可以分开处理。先不考虑，先来看第一项：也就是说给定topic的情况下的多项分布的似然函数（参照式(3‑1)）：



(‑)

这里的代表语料中的所有词数，这些词在topic的zi=k的条件下的多次独立“多项分布”试验中被产生，我们现在将这一项分解为两个连乘，一个over topic，一个over vocabulary[[14]](#footnote-14)：



(‑)

这里我们用代表topic k下单词t被观测到的次数，而最终目标函数由对**求积分得到。

(3‑15)式子的解释：首先不考虑超参数**，而是假设已知参数**，这个**就是那个K\*V维的矩阵，表示从每一个topic产生词的概率，然后把**积分掉，仿照公式**(3‑5)**的推理，再利用公式**(2‑11)**，可以求出第一部分表达式了[[15]](#footnote-15)：



(‑)

(3‑16)中的向量即代表：第k个topic下单词的分布情况，=(该topic下第1个单词的个数,该topic下第2个单词的个数,...)。我们发现这里对**的积分的处理：**被积分掉（integrating out）而消失了[[16]](#footnote-16)。这里积分的变量是**的所有可能值的集合，有点类似于边缘概率在离散分布上的处理，假设我们有概率p(a,b,c)，可以计算在所有c的可能值上求和，就消去c，而仅计算。

这个过程属于Dirichlet- Multinomial共轭结构。可以仿照这个推理过程，也可以被推理得到，将他重写为两个的乘积：



(‑)

这里表示第m篇文档，表示第m篇文档下第k号主题词数（该文档下topic k被指派给词的计数）。为了积分掉（integrating out）**，[[17]](#footnote-17)仿照公式(3‑16)的推理，同样利用公式(2‑11)，我们做如下推导[[18]](#footnote-18)：







(‑)

这里即代表：第m篇文档中的主题分布情况，=(1号主题词数,2号主题词数,...)。这也属于Dirichlet- Multinomial共轭结构。

所以联合分布可以得到：



(‑)

有了联合概率分布，紧接着在此基础上就可以根据公式(3‑12)推导full conditionals ，而现在我们已经讨论了推导最后的Collapsed Gibbs Sampling采样公式推导的全部要素，**唯有两点尚缺，这两点就是以下的事实**：

1. Gamma约去

利用



也就是说分子和分母里的gamma函数只差1时，如果分母较大，分子gamma函数内数字为分母gamma内数字小1，整个式子就等于。

1. 连乘号约去

由于full conditionals中包含分子分母，分子和分母的**唯一差别**只在与当前采样的第m篇文档第i个单词，所以其他无关乘积因子分子和分母皆因相等而约去。

当前第m篇文档的主题采样时：约去后为

当前第k号主题的主题采样时：约去后为

Collapsed Gibbs Sampling采样公式推导如下[[19]](#footnote-19)：









(‑)

这就是LDA的Collapsed Gibbs Sampling采样公式了，其中的和由于考虑了对称超参数，所以均使用同一个和代替不同的i和i代替。这一推导将永远是MCMC历史上的经典之作，它简洁、优雅，其关键之处还在于他利用了以下事实：Gamma函数在最终的公式中神奇地消失了。

gamma函数的出现有**两个作用**：

1. 换掉积分

2. 在分子分母同时出现gamma函数时利用gamma是阶乘的特性，而约去。

做完主题采样后，根据期望公式(3‑4)就可以得到mat(doc->topic)和mat(topic->word)两个**重要的矩阵**：

,其中每个向量里的每一项为



(‑)

，其中每个向量里的每一项为



(‑)

## 总结

我们可以清楚的看到，在整个LDA迭代过程中，只是不断地对每个单词的topic编号重新指定，而mat(doc->topic)和mat(topic->word)两个概率矩阵是迭代训练完成后通过期望公式计算出来的。

**训练后如何验证模型质量的好坏，验证结果的正确与否(或者验证训练是否已经收敛)呢？这里使用了一个perplexity的公式：**



(‑)

Perplexity的意义：b可以设置为2或e，其中H(q)就是该概率分布的熵。当概率q的K平均分布的时候，带入上式可以得到q的perplexity值=K。公式里的xi为测试文本，可以是句子或者文本，N是测试集的大小（用来归一化），对于未知分布q，**perplexity的值越小，说明模型越好。**



这里的指的是第d篇文档中的所有词。

## 参考文献

1. Philip Resnik and Eric Hardisty. Gibbs Sampling for the Uninitiated. CS-TR-4956 April 2010
2. Gregor Heinrich. Parameter estimation for text analysis.
3. Thomas L.Griffiths, Mark Steyvers. Finding scientific topics
4. 靳志辉(Rickjin). LDA数学八卦 2013.2.8
5. Mark Steyvers. Probabilistic Topic Models.
6. wiki: <https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm>
7. wiki: https://en.wikipedia.org/wiki/Detailed\_balance
8. Christophe Andrieu, Nando de Freitas, Arnaud Doucet, Michael I.Jordan. An Introduction to MCMC for Machine Learning. Kluwer Academic Publishers. 2001.9.10
9. ~~Gregory F.Lawler. 随机过程导论(原书第2版).机械工业出版社. 2010.9. 第1章~~

# 实现与应用

根据前面的推导，LDA的代码实现近在眼前，而Gibbs Sampling的LDA代码实现非常简洁，成为了机器学习界的典型案例。这一章，我分析java版本的JGibbsLDA代码(或Gibbs LDA++)后，庖丁解牛，分析该代码实现。

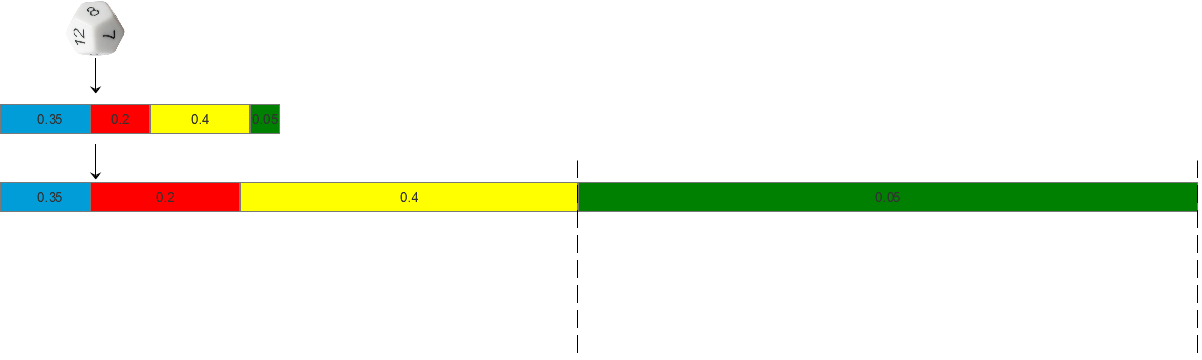
## 实现

在分析LDA的代码实现之前，我们先来实现一个小算法（该小算法可以作为面试题），题目如下：

**投掷骰子程序**：写出一个函数，骰子有K个面，各面概率不一，给定各面的概率，**请写出投掷骰子后，骰子被投掷到的面编号的输出**。这里可将骰子抽象为一个array数组(double类型数组)，这个数组内的元素为代表骰子各面的概率，函数逻辑为根据数组中各个index上的double浮点数概率，输出随机投掷到的那个array index编号：

|  |  |
| --- | --- |
| **算法4.1 Cumulative algorithm** | |
| 输入：double array[],array每项代表概率  输出：随机投掷到的array index | |
| 1  2  3  4  5  6  7  8  9  10  11 | double p[] = new double[length];  copy array’s element to p  for index from 1 to array.length do{  p[index] += p[index-1]; //逐项累积  }  double u = random double in [0~ 1] \* p[last element]  for index from 0 to p.length-1 do{  if (p[index] > u){  break  }  return index |

算法4.1在gibbs sampling中起到重要作用。该方法被称为“累积法”，将数组内概率先逐项累积，最后用该p数组范围内的均匀分布随机浮点数来随机投掷后，即可得到index值。



For循环逐项比对,直到第一项大于u

K面骰子各面概率不均

0~1的均匀随机数\*last elment size (scale up)

**double u**

Cumulative method

图 ‑1 cumulative method

这个方法初看起来不是很明显，为何要累积？我们以一个**直观的例子**来说明这个问题，由于累积变换后数组的最后一项的长度就等于变换前的整体数组长度，在下面这个例子中假设我们要随机投掷到红色部分的面，这个面的机率只有5%，而我们计算机程序的随机数生成器则是生成0~1之中每个小数机会都相等的均匀随机数。这个例子的图示如下：

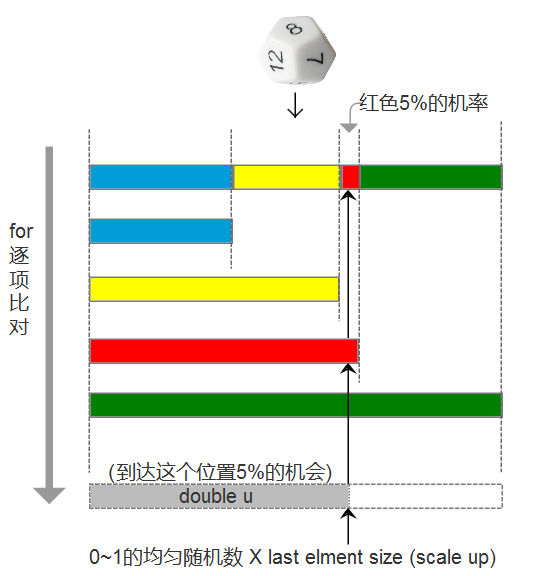


图 ‑2 cumulative method的一个直观例子

在上面这个例子中，蓝色和黄色的概率都在前面的for循环中被过滤了，因此在比对中我们如期到达了红色的面，此时判断double u<累积后的红色概率而退出循环。我们紧接着来实现Collapsed Gibbs sampling的代码。[[20]](#footnote-20)

根据前面的推导采样公式(3‑20)可以写出[[21]](#footnote-21)：

|  |  |
| --- | --- |
| **算法4.2 LDA Collapsed Gibbs Sampling** | |
| 输入：文档集(分词后)，K(主题数)，**，**，iter\_number(迭代次数)  输出：mat(doc->topic)和mat(topic->word)、tassign文件(topic assignment） | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | **申请几个统计量（int数组（或二维数组））：**   |  | | --- | | nw[][]:number of instances of word/term i assigned to topic j，size V x K  nwsum[]:total number of words assigned to topic j，size K  nd[][]:第i篇文档里被指定第j个主题词的次数， size M x K  ndsum[]:total number of words in document i，size M  z[][]: 是int二维数组[[22]](#footnote-22)，topic assignment for each word，  size M x per\_doc\_word\_len表示第m篇文档第n个word被指定的topic index |   **Initial阶段：**  for (m, doc) in doc\_set{ //m是doc编号  for word in doc{ //word均有wordid  topic\_index = random int from [0~ K-1]; 初始化阶段随机指定  z[m][n] = topic\_index; //将随机产生的主题存入z二维数组变量中  nw[word\_id][topic\_index] ++; //为相应的统计量+1  nwsum[topic\_index]++;  nd[m][ topic\_index]++;  ndsum[m]++;  }  }  //Initial初始化结束，后面的迭代使用了-1🡪采样公式重新分配🡪+1三重奏。  **Collasped Gibbs Sampling迭代阶段：（每一轮迭代有三重for循环）**  for iter in iter\_number{ //迭代iter\_number次  for (m, doc) in doc\_set{ //m是doc编号  for word in doc{  t=从z[m][n]中取得当前word的主题编号(初始化来自随机)  令nw[word\_id][t]、nwsum[t]、nd[m][ t]三个统计量均-1  double p[] = new double p[K]  for k in [0,1,2,…K-1]{ //从0到K-1号每个主题计算概率    //按照上述公式生成每个主题的概率存到临时概率数组p中    }  new\_t = 输入数组p到算法4.1 cumulative method随机投掷  令nw[word\_id][new\_t]、nwsum[new\_t]、nd[m][new\_t]三个统计量均+1  }  }  }  **输出阶段（也可以在迭代一半时候即可输出）：**  根据z数组输出文件tassign.txt  根据nw、nwsum、nd、ndsum套用公式(3‑4)生成文件theta.txt和文件phi.txt  文件theta格式:大矩阵文件，M行K列  文件phi格式:大矩阵文件，K行V列 |

下面讨论几个实现时的注意点：

**1. read old train file:**所谓的model文件就是算法4.2输出的这几个矩阵文件，如果输出了这几个文件，中断训练后，下次想再继续的时候：nw、nwsum、nd、ndsum这四个统计量均从tassign文件中可以读取得到。

**2. predict:**另外，如果已经训练过一个model了，如果有一篇新文档，需要对其predict（预测）新文档上的主题分布，可以利用已经训练的trn\_nw和trn\_nwsum用以下公式推断：



注意看这个公式，trn\_nw和trn\_nwsum相当于起到了伪计数(pseudo count)的作用，公式后一项因子只与**当前文档**的主题词数计数有关，故不必加入已有train model的计数。此外，predict在不管是初始化还是迭代时只修改new\_nw和new\_nwsum统计量，其余采样过程都一样。[[23]](#footnote-23)

由于trn\_nw二维数组变量已经很大，因此new\_nw变量对其影响不是特别的大，所以可以减少一些对文档预测时候的采样次数，通常情况下，20次采样足矣。假设对一篇1000字的文档（主题数100）进行20次Gibbs Sampling采样进行predict，大概需要花费多少时间复杂度呢？可以简单地做一个估算：



这个数字大概是需要200万次循环，在笔者的机器上仍需花费1秒钟。

另外，-1 🡪 重新分配topic 🡪 +1三重奏我将其形象比喻为在采样每个单词前，从各个统计量中**抠掉**该单词（及其主题），然后重新分配。

再来用两张图，来说明：-1 🡪 重新分配topic🡪 +1三重奏：

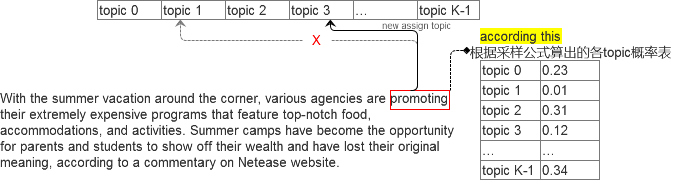


图 4‑3 -1 🡪 reassign topic 🡪 +1

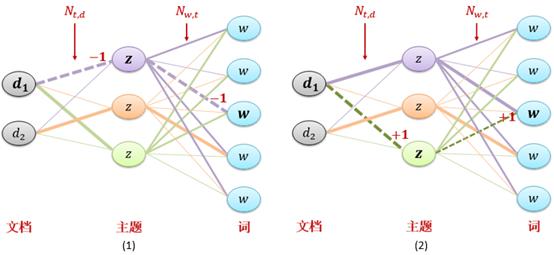
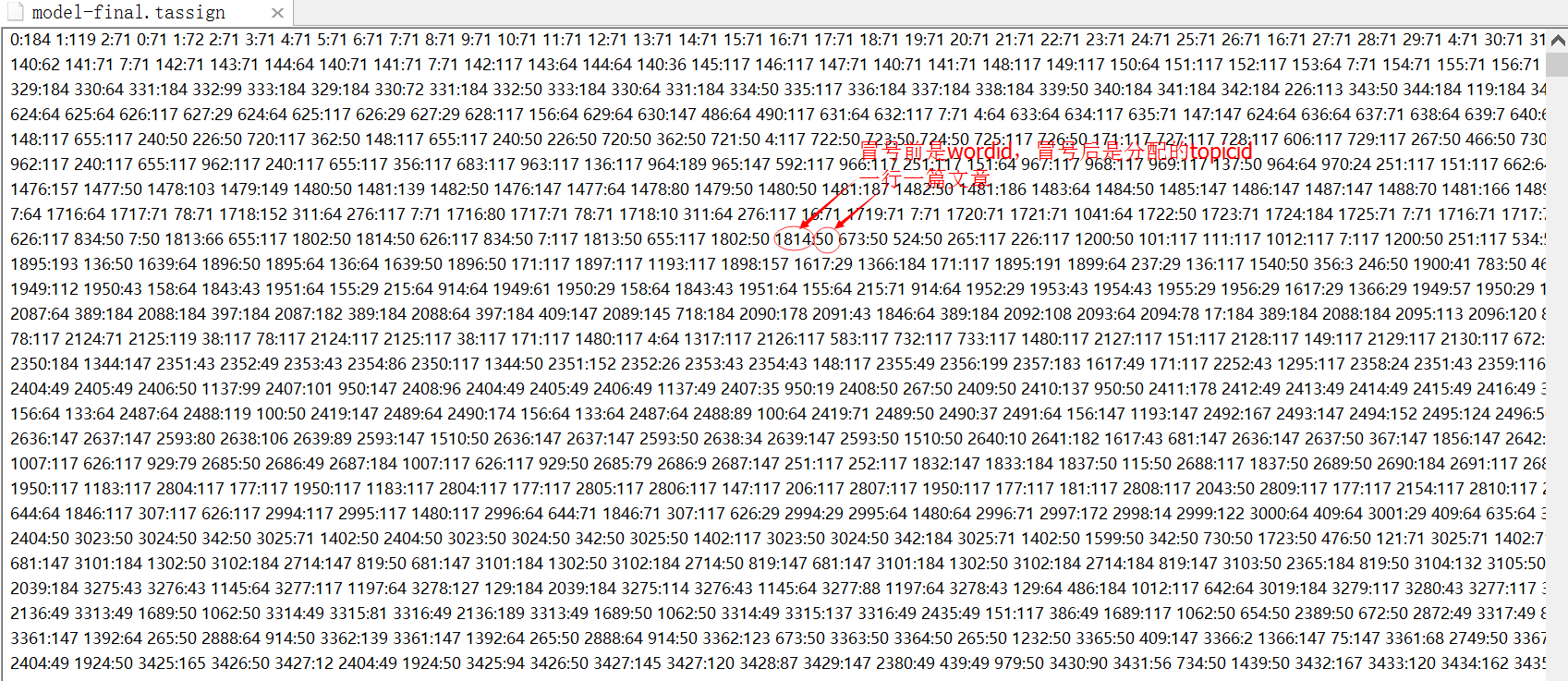


图 4‑4 -1 🡪 reassign topic 🡪 +1

JGibbs LDA或Gibbs LDA++输出的**文件样例**如下：

**1. tassign文件**（一行一个doc，冒号前是wordid,冒号后是topicid）



**2. theta文件**（矩阵文件：行号=doc idx，列号=topic idx）

|  |  |
| --- | --- |
|  | topic0 topic1 topic2 topic3 topic4 topic5 topic6 topic7 topic8 topic9 topic10 topic11 topic12 topic13 topic14 … |
| Doc1  Doc2  Doc3  … |  |

1. **phi文件**（矩阵文件：行号=topic idx，列号=word idx）

|  |  |
| --- | --- |
|  | word0 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word11 word12 … |
| topic0  topic1  topic2  … |  |

**4. wordmap文件**（word🡪wordid）



**算法复杂度分析：**

**1. 时间复杂度**：Collapsed Gibbs Sampling LDA最大的特点是每次迭代内部要经历三重for循环，因此每轮迭代经历的循环数是doc\_num \* per\_word\_num \* K = corpus\_word\_count \*K，所以最终的时间复杂度=O(iter\_num \* corpus\_word\_count \* K)，这就说明了你启动时设置的主题数K越多，算法运行就越耗时间。

**2. 空间复杂度**：算法运行中需要几个统计量，耗费空间最多的有两个nw和nd，其中nw占用内存空间最大=主题数K \* vocabulary\_size(词典中unique单词数)，所以算法的空间复杂度=O(K \* V)。例如，考虑K=300，V=10万词，且都用int（4字节）来存储，二维数组nw需要多大的内存呢？



**参数设置：**

**topic number K：**许多读者问，如何设置主题个数，其实现在没有特别好的办法[[24]](#footnote-24)，目前只有交叉验证（cross validation）：通过设置不同的K值训练后验证比较求得最佳值。我的建议是一开始不要设置太大而逐步增大实验，Blei在论文《Latent Dirichlet Allocation》提出过一个方法，采用设置不同的topic数量，画出topic\_number-perplexity曲线；Thomas L. Grifﬁths等人在《Finding scientific topics》也提出过一个验证方法，画出topic\_number-logP(w|T)曲线，然后找到曲线中的纵轴最高点便是topic数量的最佳值。有兴趣的读者可以去读读这两篇论文原文的相应部分。这个参数同时也跟文章数量有关，可以通过一个思想实验来验证：设想两个极端情况：如果仅有一篇文章做训练，则设置几百个topic不合适，如果将好几亿篇文章拿来做topic model，则仅仅设置很少topic也是不合适的。

****、******超参数：**这个超参数其实作为伪计数，不是很影响算法的结果。

**迭代次数iter\_number：**gibbs sampling前面1000次甚至更多可以抛弃，这样会达到更好的效果。

## 应用

算法实用才更有价值，我将LDA称之为万能工具，在许多文本分析的角度都可以应用（甚至推荐领域），下面我们通过几个实用例子来谈谈如何应用LDA算法。选取应用的原则是：利用基本的LDA算法的输出即可做出、无需改造原有LDA代码的，这也是对读者要求较小的应用方式。

### 相似文档发现

这个方法可以被用作新闻推荐中，正文详情页的“相关推荐”，该方法所述的相似文档是指的“主题层面”上的相似，这就比其他的基于word来挖掘的相似度更有意义，整个方法非常简单：

1. **Hellinger distance**

该距离是度量两个概率分布差异的距离，如下所述（当然你也可以用KL距离等）：



(‑)

由于前面的是常数，不影响排序，因此两个文章的相似度可以用下面的距离做出：



(‑)

**(2)算法步骤**

由于LDA可以将文章表征成多主题分布的混合，所以利用上述公式，这样仅仅用了一个文件：theta文件，就可以分析出主题层面上的距离。

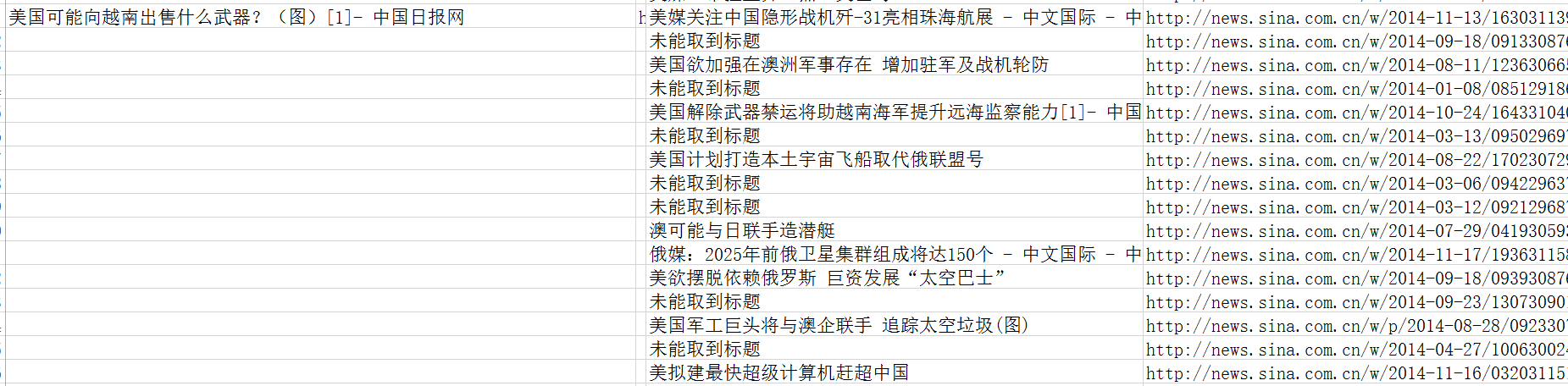
输入：LDA训练后的theta文件

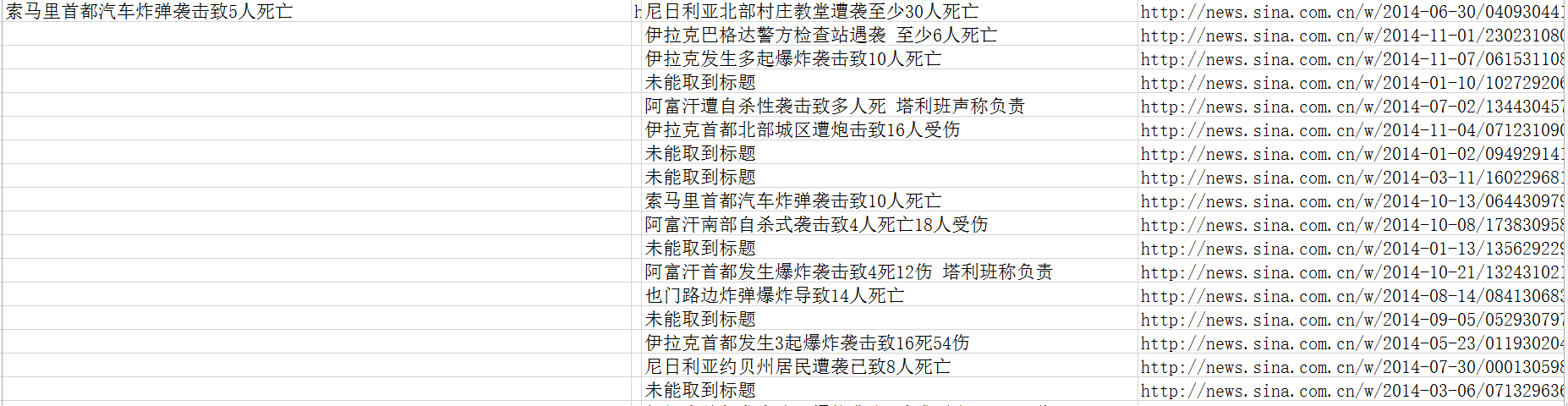
**步骤：**1. 读取LDA训练后的theta文件

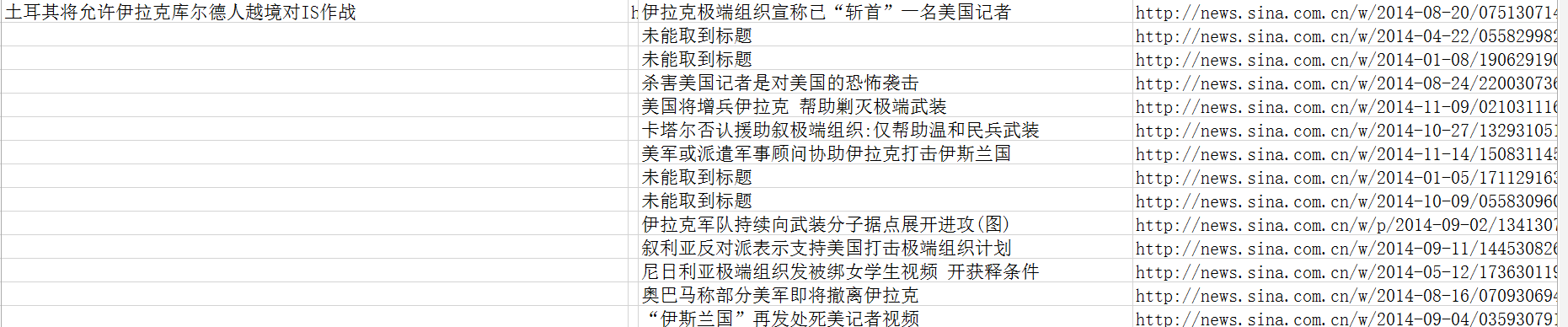
2. 为每个doc分别计算其他doc与该doc的主题距离（用上面的公式），再按照距离从小到大排序，输出每个doc最相关的前n个主题文章。

**(3) 结果演示**

比如ISIS和极端组织字面上并不相等，但表达意思一致，在这种情况下，依靠基于LDA的办法就可以成功为其推荐出极端组织的文章，部分结果如下所示（截图左边为“正文页主文章”，右边一系列为与之主题相似的文章）：







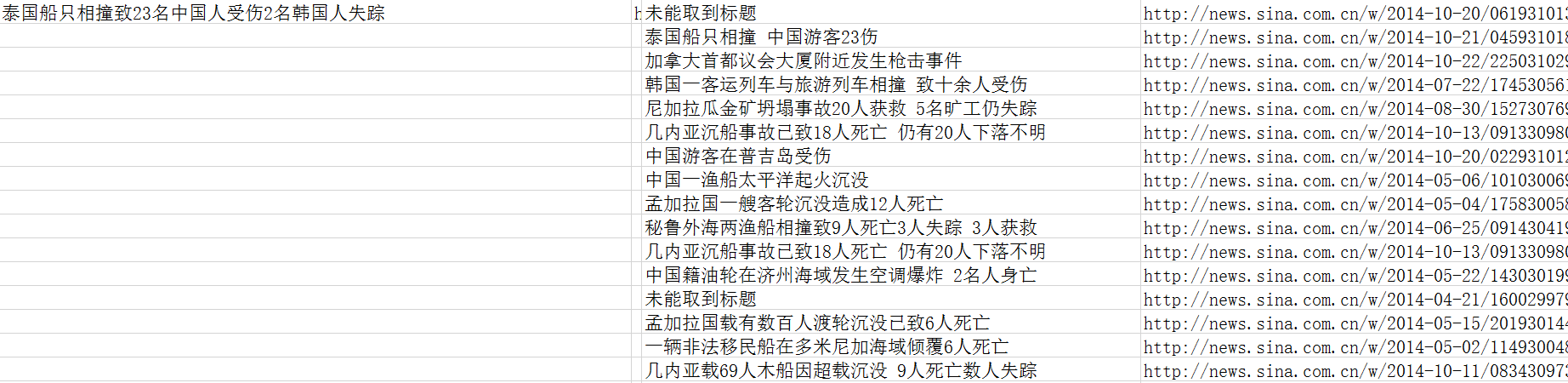


图 4‑5主题相似的文章推荐

### 自动打标签

利用LDA的训练结果做文章的简易自动打标签同样可以得到不错的结果，该方法的实现也异常简单：

算法输入：theta文件（doc🡪topic），phi文件（topic🡪word）

算法步骤：

1.分词（可以考虑使用stanford segment）

2.词性标注（可以考虑使用stanford chinese POS tagging）

3.去掉副词、介词等非实意词。

4.LDA训练

5.通过读取theta文件找出每篇文章概率最大那个主题，获得主题编号。

6.读取phi文件里每个主题下word概率，读入内存。

7.根据该文章最大主题编号找出**该文章下**该概率最大主题编号下的概率最大n个word词（max top n），（换句话说：该文章最大主题下的最大概率的n个词）作为该文章标签输出。

输出结果样例如下所示，可以看到，由于是同一个主题下最大概率的n个词，因此仅从标签表达意义就可以看出文章大体是讲什么的。



图 4‑6基于LDA打标签

### LDA与LR（逻辑斯蒂回归）结合做新闻个性化推荐系统

常见的个性化新闻推荐手段有基于关键词的，基于新闻频道栏目的等，但是基于关键词的由于粒度较细，不太容易满足推荐系统的“多样性”，而基于喜爱栏目（比如NBA）的个性化推荐粒度又太粗，所以就比较有必要研究基于主题的推荐系统了。

下面我分别介绍两种方法为用户推荐用户喜爱主题的新闻文章。

**方法1：LDA + LR（逻辑斯蒂回归）**

这个方法基于这样的假设：用户从新闻列表页点击的新闻时，肯定是对该新闻较为喜欢，而上下相邻的几条新闻未点击，肯定用户的眼睛看到了而未点：表明不喜欢。

**算法步骤：**

1.收集点击日志：

用户在一个新闻网站上（比如一个门户）通常的点击情况图 4‑7所示，我们需要收集下用户在列表页点击的文章和用户未点击的前几个和后几个文章（收集时需要做标识区别）。

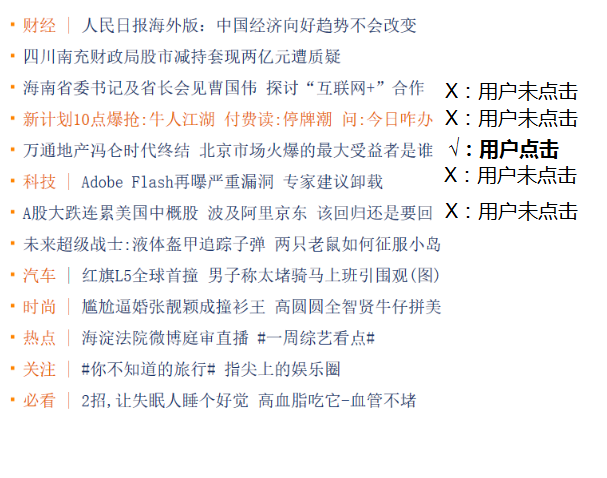


图 4‑7需要收集的用户点击的item条目和上下几条未点击

2. 使用LDA为新闻文章做主题维度特征提取



图 4‑8将文章用LDA特征提取(theta文件)

3. LR算法使用log点击数据做训练[[25]](#footnote-25)

将文章通过LDA训练后降维为K个主题（比如200个主题后），每个主题的概率为一个小于1的浮点小数。在利用前面的点击log数据，就可以将**每个用户点击标为正样本，用户未点击为负样本**。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **docid** | **用户点击** | **Topic1** | **Topic2** | **Topic3** | **…** | **TopicK** |
| 842 | 0 | 0.12 | 0.02 | 0.42 | … | 0.2 |
| 886 | 0 | 0.42 | 0.15 | 0.08 | … | 0.03 |
| 822 | 0 | 0.14 | 0.52 | 0.05 | … | 0.26 |
| 981 | 1 | 0.05 | 0.2 | 0.17 | … | 0.47 |
| 417 | 0 | 0.49 | 0.3 | 0.17 | … | 0.23 |

表 4‑1某一个用户的LR训练数据(1为点击，0为未点击)

紧接着就可以使用LR训练出该用户对K个topic里每个topic上的喜好权重weight了，再用这个weight向量对**每篇新的新闻文章**使用线性加权公式: doc\_score = w1 \* topic1 + w2 \* topic2 + …，从大到小按照score排序后，可以立即为用户提供个性化推荐。

**注意**：考虑到通常很多网站用户粘性没有那么大，一个用户的点击记录不可能很多，可以先使用用户聚类算法将用户聚类成一组一组，然后**一组用户作为训练集**训练topic喜好程度的weight权重。

**方法2：user profile记录喜好topic法**

算法步骤：

**1.提取topic：**文章LDA训练后的theta文件，提取每篇文章概率最大的前3个topic主题

**2.save topic🡪user profile：**当用户访问文章A后，就把文章A的top 3的topic贴到用户兴趣档案里，并将概率分值相加，如果过了1天用户没有访问网站或访问这个topic的文章，就按照该topic乘以0.8衰减，直至衰减到0。

**3.产生推荐：**下次用户再次访问网站时，从用户兴趣档案找出用户最感兴趣（分值最大）的k个topic，然后选取这几个topic下热门的文章为用户推荐。

### topic rank[10]

由于LDA输出的主题编号是随机的，主题也是未经排序的，因此可以开发一个为每个主题打分的算法，将显著的、有特色的主题打分高往前排，而将垃圾的、意义不明的主题往后排，这种排序算法也是很有意义的。

算法步骤：

该算法总体而言核心思路就是计算当前topic的概率分布与**垃圾分布**的距离。

**1.垃圾topic分布分为两种，准备2个距离：**

(1) topic🡪doc：由于LDA算法运行出的theta文件是doc🡪topic的概率分布，我们需要转换成topic🡪doc的概率分布，所以k号topic的第m篇文章的概率为，(这里的D是训练文章数)Topic在doc上的概率向量（其中D是语料集的文章总数），衡量其与下面这个背景噪音向量的距离（使用KL divergence、1-余弦、1-皮尔逊相关度等），距离越远分值越高，主题越有特色。

(2) topic🡪word：使用phi文件的topic🡪word的概率分布，计算当前topic的phi向量与向量的距离。

**2. 加权排序**

将上述几个当前topic向量与垃圾向量的加权求和后排序。final\_score=a \* scoretopic->doc + b\*scoretopic->word即可。

为了验证这个算法，我专门设计了一个页面，并且为每个主题人工标注了一个名字，如下截图：

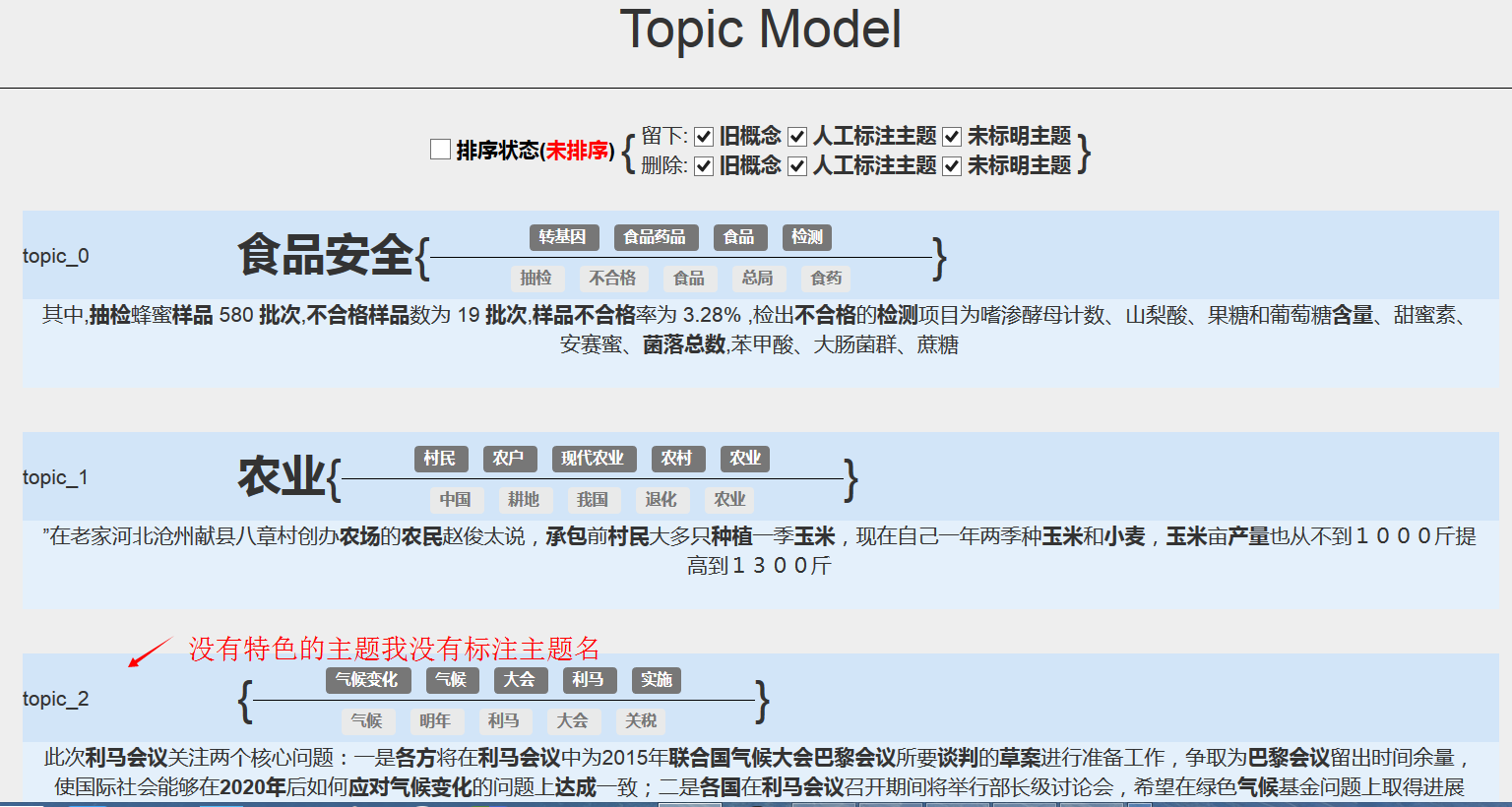
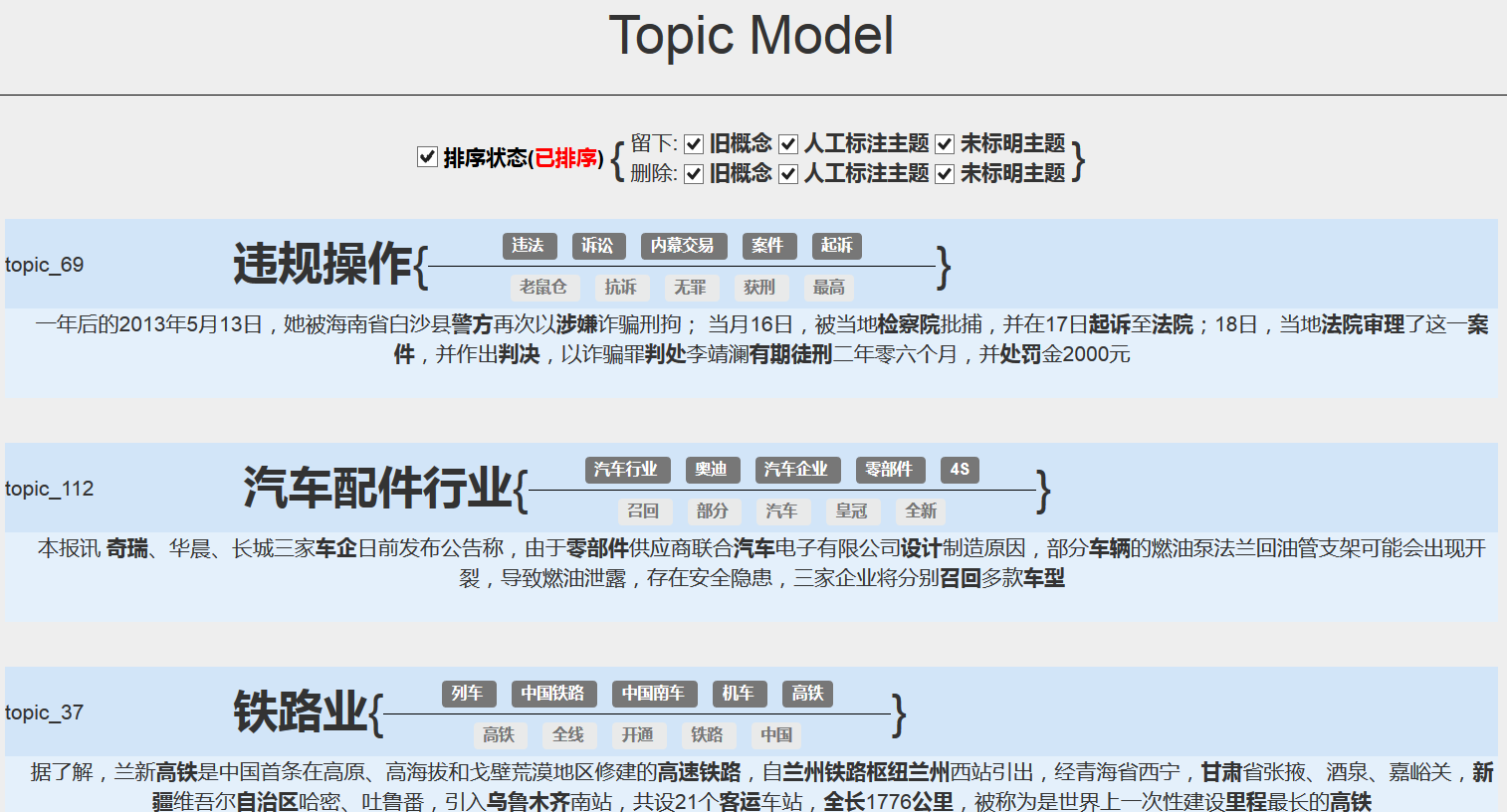


图 4‑9 rank排序前的主题



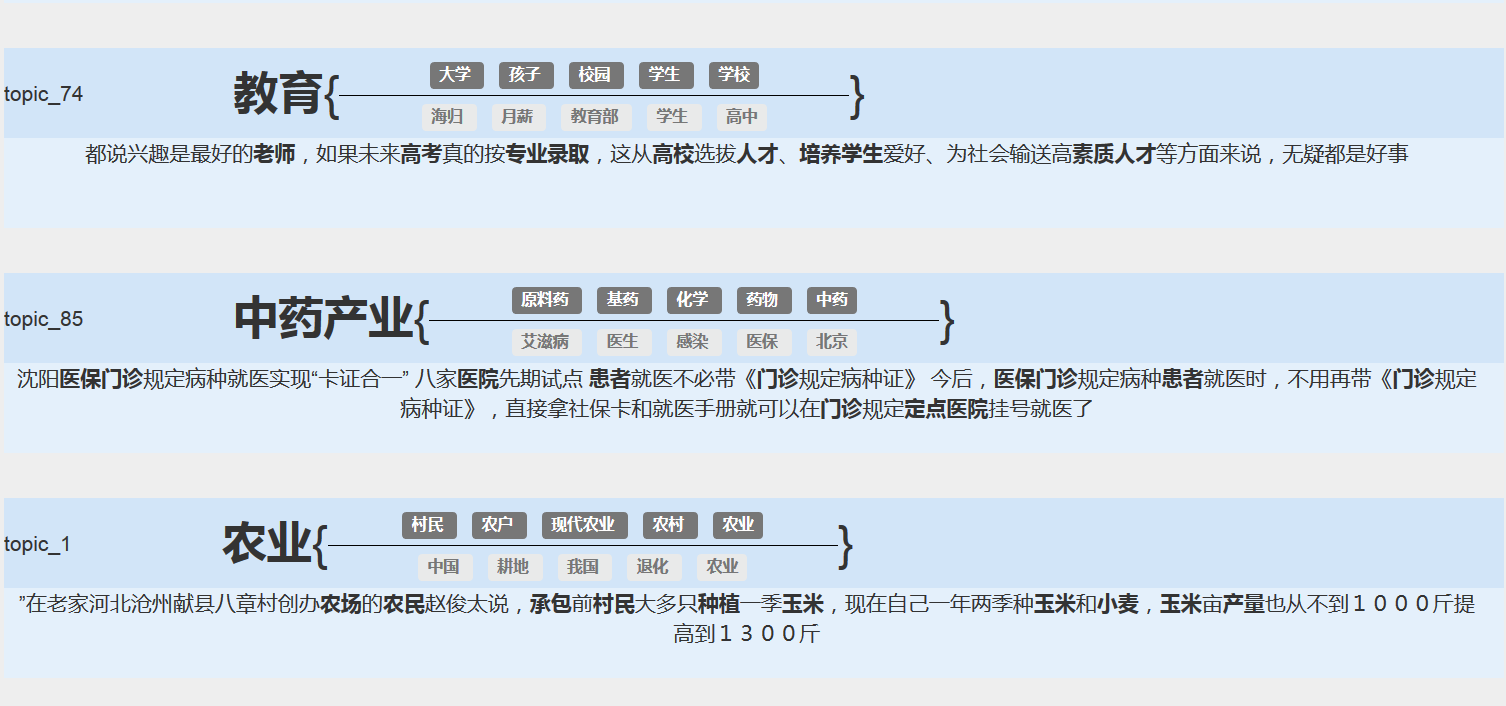


图 4‑10排序后的前几位高分topic



图 4‑11 排序后低分的垃圾topic

可以看到在排序后，出人意料地将人工标注的名称的特色topic打分较高（排序后的前几位均有名称标注），而排序后分值最低的垃圾topic则普遍没有标注名称，这说明人工挑选的topic与算法打分结果相一致。

### word rank

仿照topic rank的思路，就可以做出word rank，首先考虑下word rank有什么用，在哪些场景有使用，再给出基于LDA的word rank算法

word rank的应用场景之一：**协助文本分类器**

比如在淘宝网搜索时，经常需要“分类搜索”的功能，如下截图：

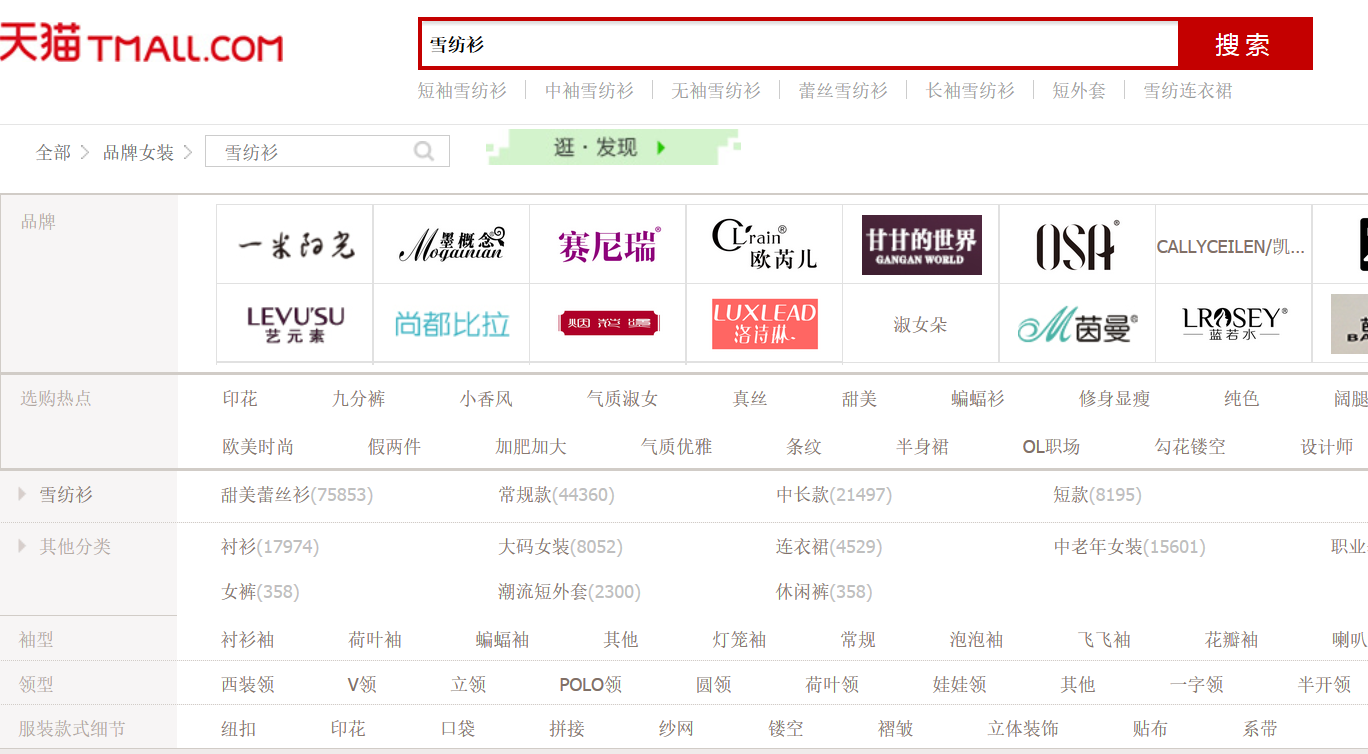


图 4‑12分类搜索

现在我们需要按照店家给出的商品标题描述分类，但是，如果你仔细观察店家给出的商品标题，会发现如下情况：



图 4‑13商品的标题描述会误导搜索分类

如果我按照传统的分词后TF-IDF提取word特征的方式分类，就会出问题，比如上面这个例子中，店家为了增加他们被搜索命中的机会，通常在标题上填写很多重复冗余无用的信息，比如图上的标题中“套头”这个词的意思是：没有扣子或者拉链的，必须从头上套着穿的。但是这个词是不能用作分类的依据的，搜索时我只想按照商品的主要特征词来分类，而非“套头”。

如果有一个方法可以获取到每个分类下的主要特征词，而忽略每个分类下冗余词，那么就可以正确的分类了。所以**步骤**是：1.根据印花、真丝等类别词匹配(match)找出相应的商品的文本；2.**每个分类**执行word rank算法，为每个词打分，过滤掉分值最低的垃圾词；3.word rank的分值作为SVM/LR等分类器的特征依据。

分类器在另一个情况下也可以使用word rank作为辅助：仅有正样本。比如考虑这样一种情形，希望对新闻分类，找出“奇闻异事”类型的新闻作为推荐，预计可提高一些点击量。所谓的“奇闻异事”类型的新闻是指类似“连体婴儿”、“怪人怪事”、“外星人UFO”之类的报道，困难在于这种分类训练集仅可以找到一些“奇闻异事”的报道作为正样本（例如用爬虫去猎奇网站抓取），但不太容易获得“非奇闻异事”即负样本作为分类器的训练集。此时可以对这些“奇闻”的语料集进行word rank找出rank排名靠前具有代表特征的词，譬如“金字塔”、“连体人”等，不含这些词的文章可被视为分类器的负样本进行训练。

下面的算法步骤详述了word rank算法：

1.计算word在每个主题上的概率分布，得到向量

读取phi文件，由于phi中的概率值是topic🡪word 的，而我们需要的是word🡪topic的反向关系，因此使用，其中每一项的，(这里的大写K是主题总数)。

2.计算垃圾/噪音word向量

，每一项都是1/K的平均值。

3.使用KL距离度量两者距离

可以采用1-余弦相似度，或KL距离，或几种距离度量的平均值度量两者距离，最后均一化，距离越大的分值越高。

我以汽车新闻类型的文章为案例，用该算法输出样例：

|  |  |
| --- | --- |
| word rank得分较高的word | word rank得分较低的word |
| 瑞虎 0.919845109474  专车 0.919889049627  奔腾 0.920182327547  吉利 0.920315761625  菲翔 0.920482668896  凯迪拉克 0.920863099045  长城 0.920909053849  本田 0.920911274423  福克斯 0.921148048364  宝骏 0.921434482516  荣威 0.921925139295  路虎 0.922897265021  瑞风 0.922900276469  捷豹 0.923057646905  景逸 0.923665401422  海马 0.924589687164  奔驰 0.924722328098  沃尔沃 0.92505266906  南通 0.92513072146  东风标致 0.925149629944  哈弗 0.925476645704  雷克萨斯 0.926134979376  马自达 0.926283860082  起亚 0.926606816176  奥迪 0.926743050409  昆山 0.927461877372 | 悍马 0.0163962476521  加倍 0.0181429299963  韦伯 0.0183433086821  充充电 0.0185650744705  假定 0.0187398834494  程惊雷 0.0188122305783  六百 0.019123897586  投运 0.0191464581692  一臂之力 0.0191952400808  欧阳明 0.0194425138855  发卡 0.0194874355798  教育机构 0.0195078269565  交卷 0.0195195159147  污染天气 0.0195711744352  于海霞 0.0195859600484  源点 0.0195859600484  暂住证 0.0195920128674  一概而论 0.0196311823389  刘理萌 0.0196311823389  长信 0.0196311823389  语气 0.0196637204583  娄格嘉 0.0197160360731  购物网站 0.0197221613537  私企 0.0197550919891  文德 0.0197550919891  奸商 0.0197611161447 |

表 4‑2 word rank汽车文章例子

可以看到得分较高的word大部分均为汽车品牌，大部分的得分就较为准确，经过测试，平均可以达到80%的准确度，但得分低的里面“悍马”的分就太低了，这就表明了此算法仍有优化空间（留给读者，可以结合topic rank的score进行优化）。

### 文章质量评分算法

这是迄今为止，由我发现的基于原始的LDA算法训练结果就能产出的最强大的技术之一。说的通俗点，这个文章质量评分算法就好比“计算机程序自动为高考作文判分”。其实将topic rank思路扩展开，便可以延伸出此技术。

比如博文推荐方面，由于博文质量由用户自主撰写而成，良莠不齐，因此需要一个博文质量自动评分系统。在经过LDA算法对文本的训练后，可以根据topic主题在当前文章的分布情况以及文章中各个词word在主题topic上的概率分布信息开发出一个新的基于LDA的文本质量评估算法。这个算法对比以往的基于SVM分类器等方法有**几个关键优势**：1.该算法是无监督的，不需为每篇文章标注垃圾分类数据，2.该算法可以为每篇文章产生一个分值，而这是二分类的分类器做不到的。

该算法的步骤是：

1.先找出该文档的单词在特定主题下生成的概率分布即如下公式：



(‑)

在这个公式中等式左边的向量是指该文档的生成概率向量，其中每一项指的是这个文档的一个词在该主题上产生的概率 \* 文档中该主题的概率。这个式子里的即，而即，其中来自于Gibbs Sampling采样时为每个单词分配的主题编号，注意到w1~wN是指从词典中编号1直到N的所有词，如果该文档d中不含单词编号i的词，则向量中编号i位置的概率为0。

2.生成这个向量后，与“众文档平均概率向量[[26]](#footnote-26)”（Mass Article probability vector）的概率进行距离度量，其中，“众文档平均概率向量”这个向量由如下公式得出：



(‑)

其中p(k)表示k编号主题在整个语料库（corpus）中的概率，可以由如下公式得到：



(‑)

其中,**d,k表示每个文档中主题k的概率值。分子是指语料库(corpus)的所有文字的k编号主题，而分母的部分相当于是语料库所有文章的所有**之和。

3.使用KL距离(4‑6)、1-cosine相似度(4‑7)、1-pearson correlation coefficient(4‑8)三种相似度/距离度量标准公式（criterion）衡量上面这两个向量之间的距离。



(‑)



(‑)



(‑)

距离算出来后，再使用公式



(‑)

来将三种不同度量标准的距离重新归一化到0~1之间。越接近“众文档平均概率向量”的文章质量就越高，反之质量就越垃圾。最后使用



(‑)

返回这三种距离的平均值作为对文档的评分分值（区间0~1之间）。

先以旅游博客作为打分测试集，整个质量打分算法运行结果如下文件所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **旅游博文评分较高的10篇文章** | | | |
| **网址url** | **评分** | **阅读量** | **文章标题** |
| http://blog.sina.com.cn/s/blog\_53bd0d1701015f4l.html | 0.998066008917 | 61 | 山西徒步路线 |
| http://blog.sina.com.cn/s/blog\_69cb346901017r65.html | 0.991291044954 | 697 | 关于海洋航行者号“神户”“长崎”“济州岛”上岸观光的攻略之毛毛雨 |
| http://blog.sina.com.cn/s/blog\_65376d0501013ikj.html | 0.981205704494 | 17 | 最全全国旅游景点逃票大攻略 |
| http://blog.sina.com.cn/s/blog\_459154f301017xz3.html | 0.979939083754 | 189 | 【心无羁，行不止】独家奉送新鲜出炉西藏10DAYS跟团+拉萨自由行攻略-近3万字吐血整理你读完绝不会失望 |
| http://blog.sina.com.cn/s/blog\_6d4f6b220101b8cx.html | 0.979059053598 | 216 | 东欧德奥匈捷斯五国十日游记[无图] |
| http://blog.sina.com.cn/s/blog\_45e5ae5a01014mry.html | 0.978861793044 | 110 | 《康藏朝圣之旅》&nbsp;第三篇&nbsp;魂牵拉姆纳错&nbsp;&nbsp;（转载） |
| http://blog.sina.com.cn/s/blog\_3d4ceed401017a48.html | 0.976334174062 | 278 | 2012莫斯科、圣彼得堡七日游之流水账 |
| http://blog.sina.com.cn/s/blog\_4a0ab71d01014per.html | 0.975317724569 | 526 | 美国两周游游记（纽约、新泽西、堪萨斯、拉斯维加斯、洛杉矶） |
| http://blog.sina.com.cn/s/blog\_51688d58010168sk.html | 0.974945350427 | 69 | 2012年十月金秋，走入绚彩额济纳,绝美胡杨林 |
| http://blog.sina.com.cn/s/blog\_3e1b470f010179ve.html | 0.972611234889 | 191 | 泰国-大马一月游 |

表 4‑3评分较高的10篇文章

|  |  |  |  |
| --- | --- | --- | --- |
| **旅游博文评分较低的10篇文章** | | | |
| **网址url** | **评分** | **阅读量** | **文章标题** |
| http://blog.sina.com.cn/s/blog\_a2621cf20100yri6.html | 0.0825872 | 0 | 米径3公分30元/株 |
| http://blog.sina.com.cn/s/blog\_a259d83e01014lkt.html | 0.0818101 | 2 | 米径2-3-4-5-6-8-10-12公分 |
| http://blog.sina.com.cn/s/blog\_a2d9fcda01012twf.html | 0.0817654 | 0 | 16-60-190-260-360-760-1200元/株 |
| http://blog.sina.com.cn/s/blog\_7308354b01018uzp.html | 0.08163 | 437 | 神道八百万神 |
| http://blog.sina.com.cn/s/blog\_9e05cefd01013gch.html | 0.0816215 | 5 | 2-3分枝地径2公分高180公分4元/株 |
| http://blog.sina.com.cn/s/blog\_a50c721201015smj.html | 0.0683244 | 3 | [原创]www.pc-ge.com湖南阳光板批发&nbsp;阳光板价格阳光板价格PC阳光板 |
| http://blog.sina.com.cn/s/blog\_71663f9201016cr7.html | 0.06791 | 474 | (23)四川律师通信录&nbsp;Szechwan&nbsp;Lawyers&nbsp;Contacts |
| http://blog.sina.com.cn/s/blog\_4d56fe5b01012ypz.html | 0.0612046 | 59 | 解放前测绘资料目录-甘肃 |
| http://blog.sina.com.cn/s/blog\_a54beb4601015n17.html | 0.0595885 | 4 | [转载]www.pc-ge.com湖南阳光板批发&nbsp;2 |
| http://blog.sina.com.cn/s/blog\_a574e2c801013dgd.html | 0.0220331 | 20 | 一色的青瓦，飞檐碉兽，古香古色，看起来如同沷墨山水 |

表 4‑4评分较低的10篇文章

可以看到打分最低的“垃圾文章”有以下若干种类型：

1．批发货物的牛皮癣广告，如《[原创]www.pc-ge.com湖南阳光板批发 阳光板价格阳光板价格PC阳光板》；

2. 各种名词列举的博文，如<http://blog.sina.com.cn/s/blog_4a708d1301014gvx.html>，《互为近义词的成语大全》；还有《河南花木报价表，各种花木价格表》；

3. 简单列举的景点，如<http://blog.sina.com.cn/s/blog_577b695a01017bbd.html>，《中国国家级自然景区名录》；

4. 航班和汽车等时刻表，如<http://blog.sina.com.cn/s/blog_5776fab60100zeue.html>，《成都双流国际机场完整航班时刻表》等。

**实时接口（online learning）：**

由于LDA训练需要时间，如果我们需要一个实时接口，实时为新文章打分，可以采用以下策略：先将LDA的model训练完成后，将**nw（稀疏矩阵）、平均概率向量、wordmap、距离最大值和最小值（用于均一化）存于类似redis等的数据库cache下来，此为train模型的记忆。**在线上文章分词后实时传送给接口后，连接读取redis中的模型记忆，使用前面介绍的predict方法为新文章预测主题分布后，再按照上面批量打分的方法衡量距离打分即可。

另外可以进一步挖掘一下文章的“阅读点击量”和“质量评分”之间是否存在关系，但是博客文章的点击阅读量(PV)经常与作者知名度、作者粉丝多少有关，因此可以按照区间平均点击阅读量(PV)来看两者的关系：

|  |  |  |
| --- | --- | --- |
| 分值区间 | 平均pv | 文章数 |
| 0.9~1.0 | 449.841 | 390 |
| 0.8~0.9 | 588.2538 | 2975 |
| 0.7~0.8 | 399.3124 | 3073 |
| 0.6~0.7 | 341.1599 | 2277 |
| 0.5~0.6 | 167.3782 | 3038 |
| 0.4~0.5 | 149.2586 | 1655 |
| 0.3~0.4 | 137.2095 | 797 |
| 0.2~0.3 | 71.75649 | 501 |
| 0.1~0.2 | 76.86979 | 192 |
| 0~0.1 | 58 | 20 |

表 4‑5区间平均PV与文章质量打分的关系

图 4‑14区间平均PV与文章质量打分的关系

可以从上图看到，按照每0.1分为一个区间进行统计文章的平均pv，会发现存在一定关系的，即：文章打分高的点击阅读量相对来说更高，文章打分最低的点击阅读量相对来说更低。基于这一点，可以有理由将文章质量打分算法融入推荐系统，过滤掉垃圾文章，提高点击率。

当然该方法也可用于“没有标注训练集的文本垃圾过滤”等场景中。

### 总结

除了上面提到的应用外，LDA还有许许多多其他的应用，限于篇幅有限，就不一一展示了，算法优化无止境，上面提到的应用都有很多优化余地，这些就留给读者完成。

## 参考文献

1. 持之以恒. Reading Note : Parameter estimation for text analysis 暨LDA学习小结. http://www.xperseverance.net/blogs/tag/lda/ . 2013.3.5
2. Xuan-Hieu Phan and Cam-Tu Nguyen. GibbsLDA++ Reference. http://gibbslda.sourceforge.net/
3. David M.Blei & John D. Lafferty. Topic Models.
4. 赵学敏 王莉峰 王流斌 孙振龙 严浩 靳志辉 王益. Peacock：大规模主题模型及其在腾讯业务中的应用.
5. Loulwah AlSumait and Daniel Barbara,James Gentle,Carlotta Domeniconi. Topic Signiﬁcance Ranking of LDA Generative Models.
6. David M.Blei & John D. Lafferty. Topic Models.
7. Allison J.B.Chaney and David M.Blei. Visualizing Topic Models. Association for the Advancement of Artiﬁcial Intelligence (www.aaai.org). 2012

# 并行化

通过上一章的实现可以看到，LDA算法运行的时间复杂度还是比较高的，跟主题数K和训练语料数据量都有关，如果主题数过多，更是会造成训练时间的大幅上涨。因此如果训练文本语料数据或主题数过多时，有必要开发并行计算版本的LDA。

使用Gibbs Sampling版本的LDA开发并行化，存在着一个**主要难题**，就是nw、nwsum、nd、ndsum这几个统计量的并行更新问题，更具体地说，如果简单地将文章集(doc set)拆分成若干份并行计算，A进程和B进程同时启动时，由于每次重新指定主题(reassign topic)后，都会修改掉统计量（-1 🡪 重新分配topic 🡪 +1），因此就会造成修改读写冲突，破坏统计量的一致性。**3个依赖**：(1)从列上看：不同文章并行化时，某文章的单词a“依赖于”另一篇文章相同单词a采样后修改的nw和nwsum；(2)从行上看：同一篇文章的后一个单词“依赖于”前一个单词采样后修改的nd和ndsum。(3)从主题层面看，同一个主题的后一次采样“依赖于”同一个主题前一次采样的nwsum。

## AD-LDA

为了解决这个问题，David Newman等人提出了AD-LDA算法[12]，这种方法将文章分散到不同的机器上（**按行拆分**），该算法由于可以被转换成map-reduce操作，因此早期的LDA并行化实现都采用AD-LDA，该算法的核心是提出了global update操作（解决依赖1），整个步骤如下所示：

|  |
| --- |
| 算法5.1 AD-LDA算法 |
| 算法步骤：   1. 将训练集中的D篇文章分散到P个机器上。并按照以下规则将统计量分散到各个机器上：  |  | | --- | | nd、ndsum：这两个统计量仅与文章有关，拆分到各个机器上；  nw、nwsum：这两个统计量被完整copy到各个机器上，而不是拆分。  我们用nw\_p、nwsum\_p代表各个机器上的copy版本，注意到这个数组与全局占用内存一致。 |  1. 在各个机器p上，使用nw\_p,nwsum\_p,nd\_p,ndsum\_p执行原始单机LDA算法（各个机器互不知晓对方存在），注意几个统计量都被替换为了copy版本，即(nw\_p=nw，nwsum\_p=nwsum)，各机器中的统计量（nw\_p等）独立修改，其余过程与单机LDA算法（算法4.2）完全一致。 2. merge back:在各个机器分布式Gibbs Sampling运行结束后，z\_p数组已经被指定topic完成。执行global update操作：   [[27]](#footnote-27)  nw是所有进程在Gibbs Sampling之前的统计量（全局nw），在各个机器上迭代结束后，nw\_p被合并回了全局统计量nw，紧接着，全局nwsum(topic’s word count)也可以被计算出来：     1. 一轮迭代结束，转第2步 |

注意到这个算法被看作单机版本的Gibbs Sampling的近似(approximate)，因为在各个机器上不同进程启动Gibbs Sampling时，nw\_p和nwsum\_p互不知晓其他机器的存在而进行采样，就会造成前面提到的修改冲突，破坏统计量一致性，所以说这一步其实已经产生了误差，不能完全等同于单机版的Gibbs sampling，但其后的global update在每一轮迭代后都将这个问题尽可能地修复了。

**问题：**内存问题——AD-LDA算法仍然未能解决耗费内存最大空间的统计量nw的问题，因此如果词典vocabulary和topic数量过多时，仍然在每个机器上都会耗费较大内存，而由于单台机器的内存有限，尤其在Hadoop等对每个job内存限制时更不可接受。

## spark-LDA

由于存在上述问题，邱卓林研究了一个基于spark的LDA分布式方法[[28]](#footnote-28)，该方法虽然题目是基于spark的，但其实是个LDA的分布式Gibbs Sampling的一般性的原理，可以被实现在任意并行框架中（例如MPI），该方法克服了AD-LDA占用内存大的问题，并且误差比AD-LDA更小。算法用到的变量如表 5‑1。

|  |
| --- |
| X：数据集  XP ：切分后的第p份子数据集  nd：全局document🡪topic count统计量  nd\_i：nd的第i部分  nw\_j：nw的第j部分  nwsum：topic k被指定的单词总数  nwsum\_p：在第p份子块中sampling后重新统计的topic k的单词总数 |

表 5‑1 spark-LDA变量含义

算法步骤如下：

### 切分块

Spark-LDA类似于AD-LDA，也是将原始训练数据进行了切分后分散到各个机器上去执行，但是进一步考虑了列切分，由于LDA的采样中可以**不考虑**同一篇文章中单词出现的顺序，所以只要没有统计量更新依赖，就可以把文章像切蛋糕一样切分开并行执行。故这个算法的**关键思想**在于训练语料文本矩阵中**同一行(同一篇文章)**或**同一列(同一个单词)**的word都会产生**统计量更新依赖**，所以**同一行或同一列**的数据不能被同时sampling采样。（这里的行列是指每篇文章排序后的wordid）如表 5‑2所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **doc编号** | **文章原始内容** | **转换成wordid后** | **对wordid排序** |
| doc m | He published several books, including Chromic Salts Technology, Rejuvenating Chemical Industry through Science and Technology, Soft Science and Reform, Large Linear Target Programming and Application, Research in China's Economic Development and Reform and Economic Reform and Development in China. | 306 310 482 326 497 403 400 323 375 394 392 371 415 343 417 316 329 440 417 484 436 305 444 392 463 312 315 425 | 305 306 310 312 315 316 323 326 329 343 371 375 392 392 394 400 403 415 417 417 425 436 440 444 463 482 484 497 |
| doc n | In recent years, he devoted himself to the use of complexity science to study issues relating to the development and reform of China, made enormous efforts to explore and explain the characteristics and law of development of the fictitious economy and actively studied and promoted the development of venture capital in China. | 444 312 384 306 458 372 471 424 466 359 406 415 490 476 496 312 343 315 379 393 454 454 447 361 397 325 336 482 374 495 338 492 393 425 | 306 312 312 315 325 336 338 343 359 361 372 374 379 384 393 393 397 406 415 424 425 444 447 454 454 458 466 471 476 482 490 492 495 496 |

表 5‑2先将原始训练文章转换为排序后的wordid

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Doc\_m | 305 | 306 | 310 | 312 | 315 | 316 | 323 | 326 | 329 | 343 | 371 | 375 | 391 | 392 | 394 | 400 | 403 | 415 | 417 |
| Doc\_p |  | 306 | 310 | 312 |  |  | 323 |  | 329 | 343 |  |  |  | 392 |  |  | 403 |  | … |
| Doc\_q |  | 306 | 310 |  |  |  | 323 |  |  |  |  | 375 | 391 |  |  | 400 | 403 | 415 | … |

表 5‑3每篇文章排序后wordid，原训练语料🡪矩阵

上表是将原始语料转换为排序后(sorted)的矩阵后（并且对齐后），同一行和同一列的wordid在sampling时会产生统计量依赖。空的格子表示该文章没有这个词而被对齐留空。以上表中323为例，**画十字线**的上的单词均不可被同时并行执行sampling。

经过分析，整个算法的步骤：可以先将原始数据X划分为P\*P的份，这里的P是我们设置并发数，如图 5‑1所示的P=3，每一份数据都相应地标注了坐标，注意到由于十字线上(同一行列)不可以被同时执行，因此图 5‑1的执行分为几个阶段执行，对角线上的(0,0)、(1,1)、(2,2)这3块数据先被同时执行，而每一块内部执行单机版的Gibbs Sampling，这三块并行执行结束后，再执行另外不在十字线上的3块(0,1)、(1,2)、(2,0)，这三块执行结束后最后执行(0,2)、(1,0)、(2,1)三块（如图 5‑1的shuffle步骤之后的三行小组，三行行内并行计算，行间串行计算）：

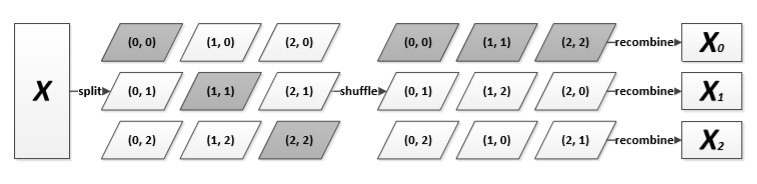


图 5‑1 spark-LDA切分流程，并发度3

这里的“一块”的概念是包括多行和多列的，我们希望每块的数量都均等，这样在每台机器上才能负载均衡。但在切分时候由于“对齐”会造成空的word格子的原因，所以，简单地按照列数均分和行数均分来切分是不行的。为了尽可能每块均等，可以分割多次，然后挑选出各个块之间**差距的最大值最小**的那一次。

**切分策略：**(1)首先将数据集X按照列切分成均等的P份，如果包含的单词数不均等，可以随机交换列，再切分来保证均等。（2）在将切好的P份列，切成P份行，如果不相等，多次**随机交换行**，最后挑选出各个块之间**差距的最大值最小**的那一次。

### 选择

切分成P \* P块这个步骤后，就是挑选出哪些无冲突的P个块可以同时被并发执行（组内并行，组间串行）。通用的策略有2种：

1. **八皇后法**

八皇后问题是一个经典的问题，在8X8格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。所以这里的挑选无冲突块类似于八皇后摆棋子，我们这里的情况是弱化的八皇后问题，即挑选的块可以出现在同一个斜线上。八皇后的经典解法是回溯法**[[29]](#footnote-29)**。

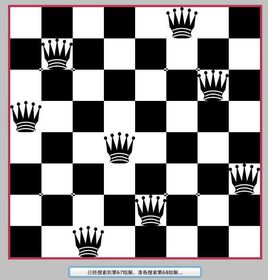


图 5‑2八皇后问题

1. **对角线法**

腾讯的peacock系统[17]采用的是更简单的对角线法，由于同一行或同一列的块不能被同时选择，因此选择对角线肯定是没有问题的。沿着一条条对角线进行采样计算，一旦一条对角线计算完成后，就并行计算下一条对角线，同一对角线内并行计算，不同对角线间串行计算。

### 计算和合并

由于上述算法过程中的方法是：组内并行，组间串行，所以小组并行执行的一次迭代完毕后，小组的nd、nw等统计量需要同步(sync)到下一个小组内。而组内各个块内sampling计算过程与**单机版gibbs sampling完全一致**，只是将统计量切换：nd、ndsum和nw切分成P份，比如第j份是nd\_j和nw\_j，由于在公式(3‑20)中，nwsum(topic k被指定的单词的总数)无论如何都会产生冲突(依赖3)，所以整个算法的误差也就在这个统计量了。因此在第p台机器上的nwsum被修改为nwsum\_p后（并行计算完结后）都需要用global update方法merge back。



(‑)

除了nwsum需要merge back之外，其余几个统计量，nd\_j和nw\_j都需要合并到下个小组的文档d和单词w。（这种合并过程可以用spark的broadcast）整个过程见图 5‑3。

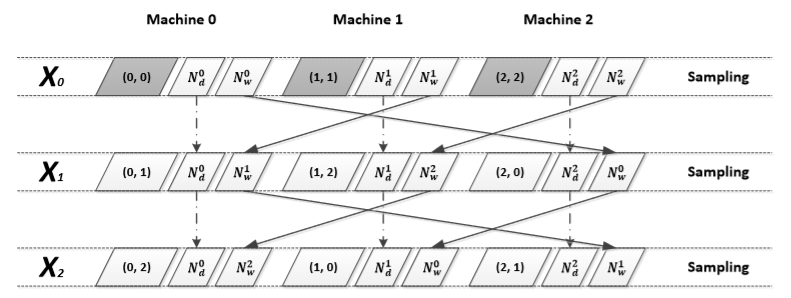


图 5‑3计算后合并，同一个单词的nw和同一个文档的nd统计量被合并

### 总结

**实验证明：**这种算法产生的混淆度(perlexity)与单机版Gibbs Sampling一致，因此误差非常小，而且分布式计算中的耗费内存也可以接受。但是实现复杂度稍高，需要比较好的spark编程功底。

(1)整个训练文章集(doc set)不仅以行切分(行指文章)，也以列切分。

(2)各个小块被切分尽可能数据量均等。

(3)统计量nw和nd因此也被切分到各个机器进程上。

(4)执行流程为：每轮迭代组内并行，组间串行。

(5)同一行或同一列的数据不能在一个小组内的不同块上同时并行执行。

(6)各个独立进程运行完毕后，误差仅在nwsum，通过global update合并nwsum。

## 参考文献

1. David Newman, Arthur Asuncion, Padhraic Smyth, Max Welling. Distributed Inference for Latent Dirichlet Allocation. {newman,asuncion,smyth,welling}@ics.uci.edu
2. Zhuolin Qiu. Gibbs Collapsed Sampling for Latent Dirichlet Allocation on Spark. JMLR W&CP 36 : 17–28, 2014
3. ~~A.E. Brockwell. Parallel Markov Chain Monte Carlo Simulation By Pre-Fetching. 2005~~
4. ~~Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng. LightLDA: Big Topic Models on Modest Compute Clusters.~~ [~~arXiv:1412.1576~~](http://arxiv.org/abs/1412.1576) ~~. 2014~~

# 变分贝叶斯的启蒙

在我们的故事叙述到这里的时候，读者已经发现，Gibbs Sampling技术虽然实现简单且输出效果不错，但是其采样的效率犹如“大力神海格力斯式的笨重劳动” （欧拉语）。因此有必要看看从另一种思路出发而产生的技术，变分贝叶斯法（Variational Bayes）推导是Blei的LDA论文原作中的经典方法，如果说学习LDA模型，但没有学过原作中的VB法推导，仍然称不上学会了LDA。这种方法运行速度比Gibbs Sampling方法快，而且这个方法可以不经过采样z直接推断出等参数。虽然有读者会争论说这种方法得到的参数陷入了局部最优，不是很精确，但是在后期我们可以看到这个变分思想作为新技术的启示，结合其他技术（包括Gibbs Sampling等）可以产生巨大威力。这一章主要讲述变分贝叶斯(Variational Bayes)技术的启蒙知识。

## 前置知识

在正式讨论这种技术之前，在第2章的一些前置知识基础上，有必要需要补充一些简单的基础知识。记住：真理的碎片有可能散落在不起眼的地方或特别的地方，所以我们需要将它们捡拾收集起来，以达到我们的技术目标。

### 指数分布族(exponential family)

指数，也就是数学上的次方之意，比如平方的指数=2，立方的指数=3，随着越来越多的分布被发现，人们逐渐发现有一些特定的分布可以被总结为“指数分布族”，也就是利用特定的方法，可以将这些分布转换为某一种相同的**形式，这些形式的期望等性质都比较好求解，具有统一的形式**。下面直接给出指数分布族的密度函数：



(‑)

你可以特别注意到这里的指数指的是**自然常数e的指数**，公式中exp{x}也就是e的x次方，这个公式的**、g(**)、u(x)都不是随便定义，均有意义，解释如下：

x : 有可能是标量（譬如一个实数）或是一个向量

** : 自然参数(natural parameter)，更详细的解释在下文例子之后

u(x): 充分统计量(sufficient statistics)(后文会进一步解释这个概念)

g(**) : 归一化参数，由于p(x|**)的概率之和=1，这个g(**)是用于将其归一化的（还有个名称叫partition function）。

即： ，所以

通常公式(6‑1)也可以写为：



(‑)

注意到这个公式与前面式(6‑1)的区别仅仅在于形式上从g(**)被转化为了a(**)，这个a(**)被称为-log(归一化参数)（log-normalizer or log-partition function），见公式(6‑3)。



(‑)

将一些普通分布转化为指数分布族的形式方法也非常简单：

Step 1：将p(x|**)凑成指数形式exp{ln(p (x|**))}

Step 2：整理为公式(6‑1)或公式(6‑2)的形式。

通过下面几个例子可以加深理解这一点：

例 6.1 Bernoulli分布的指数分布族形式

伯努利分布与前文所述的二项分布相似，只是没有了二项式系数。



先对这个式子进行指数变换，则原式变为



第二步，对这个式子整理:



现在已经凑出了**，剩下的目标是凑出g(**)的形式，所以**目标转变**为将(1-**)转换为**的函数形式g(**)



所以

最终

上述式子中归一化参数g(**)= σ(-**); h(x)=1

经历了这个例子，相信读者也掌握了这个将其他分布（[normal](https://en.wikipedia.org/wiki/Normal_distribution), [exponential](https://en.wikipedia.org/wiki/Exponential_distribution), [log-normal](https://en.wikipedia.org/wiki/Log-normal_distribution), [gamma](https://en.wikipedia.org/wiki/Gamma_distribution), [chi-squared](https://en.wikipedia.org/wiki/Chi-squared_distribution), [beta](https://en.wikipedia.org/wiki/Beta_distribution), [Dirichlet](https://en.wikipedia.org/wiki/Dirichlet_distribution), [Bernoulli](https://en.wikipedia.org/wiki/Bernoulli_distribution), [categorical](https://en.wikipedia.org/wiki/Categorical_distribution), [Poisson](https://en.wikipedia.org/wiki/Poisson_distribution), [geometric](https://en.wikipedia.org/wiki/Geometric_distribution), [inverse Gaussian](https://en.wikipedia.org/wiki/Inverse_Gaussian_distribution), [von Mises](https://en.wikipedia.org/wiki/Von_Mises_distribution) and [von Mises-Fisher](https://en.wikipedia.org/wiki/Von_Mises-Fisher_distribution) distributions）转换为指数分布族的方法。那我们再来个例子：

例 6.2 狄利克雷分布 (Dirichlet Distribution)的指数分布族形式

我们又来到了我们熟悉的**Dirichlet Distribution，**找到第2章我们学过的概率密度函数公式(2‑10)，将概率p字母用**代替后如公式(6‑4)所示。



(‑)

先对这个式子进行指数变换，则原式变为:



(‑)

在Dirichlet Distribution的指数分布族表达式形式中，h(x)=1

有人也许会想，只不过把分布函数的形式换了种写法罢了，这有什么用啊？其实读者大可不必如此着急，一时不明了的概念在后期会柳暗花明，指数分布族的用处在于与其他方法相结合（比如贝叶斯估计等），在后文中我们可看到这一点。

很多初学者会疑惑，为何LDA要选Dirichlet分布？在这里也可以解释，Dirichlet分布是指数分布族，且具有有限维度的充分统计量（sufficient statistics），也是多项分布（multinomial distribution）的共轭先验分布。因此在第7章的推断和参数估计中会利用到Dirichlet分布的这些特性。

**再论****：通过上面两个例子，可以清晰地发觉一个事实：***不过是概率密度函数中参数，***一系列不同的***产生不同的参数空间***

### 再谈数学期望

读者注意了，这里的期望是指概率论与数理统计中的期望，和我们日常口语的期望不是一回事，统计学中期望这个概念究竟是什么意思呢？其实应该有个更平易近人的名字：随机变量之平均值，为什么要起这么个不够形象易懂的名字？由于概率论最早从研究赌博开始发展，当初期望主要是用于计算赌局的赢钱金额数，所以后来也就把这个名词沿袭下来了。

我通过一些例子谈谈数学期望，我们从平均数开始出发，假如要计算一个班级的平均成绩：E(X) = (x1+ x2+ x3+…+ xN)/N；让我们再加上概率这个权重，也就是说，每个变量x的概率权重不等，所以E(X)期望等于“x的可能值与其概率之积的累加”。



这个式子对初学者而言可能看起来不是很明显，那换种说法再来解释一下，假设我做N次试验，每次把X的取值记下来，在这N次试验中，有N1次取到x1，有N2次取到x2，…，而这N次试验中X的总共取值为x1N1+ x2N2+…+ xNNN，而**平均**每次试验中X的取值，记为E(X)

E(X) = (x1N1+ x2N2+ x3N3+…+ xNNN)/N

= x1(N1/N) + x2(N2/N) +…+ xN(NN/N)

N1/N是事件x1发生的频率，按照概率统计的定义，当N很大时候，频率接近概率，也即N1/N接近p1，因此就得到了下面这个公式：



(‑)

一个关于期望的性质是：若干个随机变量之和的期望等于各变量的期望之和。



( ‑ )

读者可以想想为什么该性质能成立，这个性质的证明留给读者来完成。

拥有了期望的公式(6‑6)，我们还不满足，想进一步探究随机变量**函数**的期望，也就是下面的定理。

**求证：**（随机变量函数的期望），设随机变量为离散型，有分布P(X=xi)=pi(i=1,2…)；或者为连续型，有概率密度函数f(x)，则:

1. 随机变量离散型时：

( ‑ )

(2)随机变量连续型时：

( ‑ )

**证明：**

**(1)先来看第一种离散**的情况，这比较好证明，因为P(X=xi)=pi,所以P(g(X) =g(xi))=pi,由此立即可以得证(1)中等式。

**(2)再来看第二种连续**型随机变量的情况，我们下面来证明等式：

在连续情况下：

我们先假设g(X)函数为严格上升并可导，**这个证明一旦想清楚换元就很简单**，下面给出证明

1. 先来观察Y=g(x)，由于g(x)属于严格上升函数，因此分布函数换成反函数X=h(y)表达：



P关于y求导数，得到y的密度函数

1. 又因为h是g的反函数，则h(g(x))=x(此式左右两边同时对x求导)，

推理出即

因此，

把y换成g(x),则

最后

关于g(X)非单调函数的证明超出了本书范围，留给读者来完成。

**证明方法2：**

我们再来给出第二种证明方法，我们在这里仅就证明（非负）情况下式( 6‑9 )成立。

要证明式( 6‑9 )，还需要一个另一个结论：对于**非负随机变量Y**，有



( ‑ )

先来看看如何得到这个结论，我们假定Y是一个连续型随机变量，其密度函数为，此时得到



此处利用了事实，交换上述的积分次序，可以得到



得到结论( 6‑10 )之后，我们接着来证明( 6‑9 )式：

对于非负函数，根据式( 6‑10 )，有



交换积分次序，可得：



这样，式( 6‑9 )就得到了证明。为一般条件下的证明留给读者来完成。

当然你不关心上述的证明也可以，只需要记住这个公式，这个公式中E的下标f表示是关于概率密度函数f(x)来求导的，通常也可以写作<g(X)>f，整体表达见公式(6‑11)。



(‑)

公式(6‑11)是一个众所周知的基本的概率论公式，在后文的变分推断技术推导中，这个公式会被反复用到。

最后来看一下二维(多维)随机变量函数的数学期望：

**求证：**二维(多维)随机变量函数的数学期望有以下命题:

(1)设随机变量离散型随机变量(X,Y)有联合分布列为，为二维函数，则的期望为：



( ‑ )

(2)设连续型随机变量(X,Y)的联合分布密度为，为二维函数，则的期望为：



( ‑ )

**证明：**这里仅就（非负）证明此命题。

要证明式( 6‑13 )，就要利用前面已经得到的结论公式( 6‑10 )：



将概率



代入期望公式可得：



将上述三重积分交换积分次序，得到：



这样便证明了式( 6‑13 )。

了解了数学期望的相关知识，自然可以对指数分布族再做一番探究。

### 进一步观察指数分布族

**观察1.** 先来观察一下公式(6‑3)中的a(**)，如果对a(**)关于**求导，会得到什么呢？



这里的log其实是ln（以e为底的对数），考虑到导数的性质，，应用**复合函数的链式求导法则**，在这次的求导中f函数就是log，因此：



 //链式求导法则





由此可见，a(**)关于**求导得到了E(u(x))。

现在我们将这个结论用到我们最熟悉的dirichlet distribution中，由于按照**(6‑5)**式中的狄利克雷分布的指数分布族表达式，u(x)=ln(**)，又由于



这里由于看到要对log-gamma函数求导，因此引入一个新的数学符号，下面公式中这个长得像鱼叉一样的符号是希腊字母Psi（普赛），该函数就是Digamma函数：



则可以轻松得到以下公式：



(‑)

这个狄利克雷分布的关于ln(*i*)的期望表达式在Blei用变分推断法推导LDA时被反复用到，是个**十分重要**的式子。

**观察2.** u(x)被称为充分统计量（sufficient statistics），何为充分之意，为何叫充分统计量，可以这么理解：对于要估计的参数**来说，**的似然函数**仅仅依赖**于u(x)。为了清晰地看出“充分”这一点，下面我们使用极大似然估计（MLE）简单地估计一下参数**:







紧接着按照极大似然估计的标准做法对c求导数使其等于0：

 //h被看作常数忽略

由于，为了使上述求导等于0，则将此式子的期望替换代入上式，所以整理可得，也就是说最后所求** MLE**必须满足**使“充分统计量u(x)关于极大似然估计值** MLE的期望”=“充分估计量u(xi)关于所有样本点xi的平均值”。这样就得到了最终**的极大似然估计** MLE。所以可以清晰地看到充分统计量概念的来历，这里我们不需要存储每个数据点x，而仅仅需要存储就可以估计出参数**，这便是充分统计量之要义。

### 拉格朗日的杰作——拉格朗日乘数法

拉格朗日(全名为约瑟夫·路易斯·拉格朗日)，法国著名的3L人物之一，（法国18世纪后期到19世纪初数学界著名的三个人物：勒让德（adrien-marielegendre)、[拉格朗日](http://baike.baidu.com/subview/19783/19783.htm)（joseph louis lagrange)和[拉普拉斯](http://baike.baidu.com/subview/5864/5144318.htm)（pierre-simonlaplace)三个人的姓氏的第一个字母为“L”，又生活在同一时代，所以人们称他们为“三L”。）拉格朗日是数学界无畏的高手，在数学，力学，天文学都做出过杰出的贡献，另外拉格朗日与欧拉合作开启了变分法这一全新的学科。后来变分法又与贝叶斯方法所结合产生出了强大的威力，为机器学习打开了一扇新的窗户。本节所讲的是另一个拉格朗日广为人知的成果：拉格朗日乘数法。



**图 6‑1 约瑟夫·路易斯·拉格朗日**

任何学过高等数学或微积分的人都接触过拉格朗日乘数法，什么是拉格朗日乘数法？简而言之：条件极值。

要z=f(x,y)在条件g(x,y)=0下的求极值的自变量x，y的情况，譬如：要用**指定表面积**的铁皮围城一个立方体，体积最大为多少？

所谓的条件极值，有个更正式的名字的叫：约束最优化（constraint optimization），是指在一个或多个条件的约束下，求一个函数的最佳极值。举例来说，我们需要得到z=f(x,y)的最小值，但前提是x,y必须满足方程g(x,y)=0。最直截了当的做法是解出条件方程，用x=h(y)。将此式代入z=f(x,y)即可得到单一变量的函数z=f(h(y),y)，求出导数dz/dy=0即可得到y的极值。但这个方法有其不足，首先明确看到的一点就是不一定能得到x=h(y)。

天才拉格朗日发明的乘数法，优雅而简洁地解决了这个问题。拉格朗日乘数法的步骤只有2个部分，非常简洁，我们先来给出拉格朗日乘数法的解法步骤，再来通过一个直观的例子解释这个解法。

|  |
| --- |
| 设给定函数z=f(x,y)和附加条件g(x,y)=0，为寻找z=f(x,y)在附加条件下的极值点，先做拉格朗日函数L(x,y)=f(x,y)+**g(x,y),其中**为新引入的一个乘法因子参数，求L(x,y)对x和y的一阶偏导数，令它们=0，并与附加条件联立：    由上述方程组解出x，y及**，如此求得的(x,y)，就是函数z=ƒ(x,y)在附加条件g(x,y)=0下的可能极值点。 |

在多维(多元)函数情况下，也是一样的，轮流对各个变元求偏导即可，我们可以将符号变一下，使用和表示梯度，所谓的梯度，其实就是f和g对各个变元(x,y,z,…)的偏导数，如果将上面的方程组中的式子移项就会发现，上述方程即可写为和g(x,y,z,…)=0

**例 6.3** 求函数f(x,y)=3x+4y在圆x2+y2=1的极大值和极小值

解：此题的解答如果使用拉格朗日乘数法非常简单，首先，构造出一个拉格朗日函数L(x,y)= 3x+4y-**(x2+y2-1)











所以

最后将x和y代入f(x,y)得到极大值为5，极小值为-5。

下面我们通过图 6‑2，更深刻理解拉格朗日乘数法。

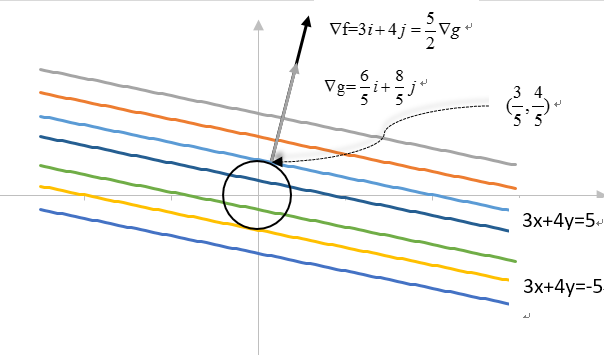


图 ‑2 拉格朗日乘数法小例子图

这张图中的一排平行直线就是我们要寻求极大值的f(x,y)，而圆形就是限制条件x2+y2=1，此题需要求的就是离原点最远的点是**哪一条**（并且与圆相交）？**同圆相切的直线是离原点最远的直线！**还记得我们前面提到过的(**可正可负)吗？这个意义也就明白了，在切点，任何同这个直线正交（垂直）的向量也是同圆正交的向量。因此，f的梯度是g的倍数**。

这个方法遵循了数学优秀方法的一贯特征：简洁优雅。拉格朗日也将自己的名字永远地刻在了这个方法上。当然，拉格朗日乘数法也可以扩展延伸限制条件为多个方程的情况。

多个约束条件下的拉格朗日乘数法：



也即构造拉格朗日函数：



(‑)

后续步骤和单个限制条件时相同。

当然拉格朗日乘数法的学问远不止于此，弱水三千只取一瓢，我的原则是，只取到需要足够使用的部分就可以了，未来如果需要用到其更深的部分，再取之即可。

### 指数分布族的一点深入的思考

在了解了拉格朗日乘数法之后，我们可以重返指数分布族了，对其进行更深的理解，这一小节主要是从理论上对指数分布族进行一些起源的发掘，与后面LDA的变分法推导关系并不大，这一小节是解决部分读者这样一个疑问：为什么指数分布族的形如 (6‑1)式？这里我引入一个概念，最大熵（Maximum Entropy）。

别被“熵”这个概念吓到，熵用来代表“随机变量不确定性的程度”，也就是信息无序的程度，变量的不确定性越大，熵也就越大，搞清楚他所需要的信息量也就越大。熵最早来自于热力学，后来被伟大的信息论之父香农用到了信息论中，香农当时想出这个熵的计算方法后，还没有想清楚该起什么名字，于是去问冯诺依曼，冯诺依曼建议叫“熵”，并说：“没有人知道熵是什么东西，因此你在争论中可以无往而不胜”。

熵的计算公式非常简单：



(‑)

为了简便起见，这里我们使用自然常数e作为对数log的底[[30]](#footnote-30)，特别注意到公式中的负号，因为概率P是一个小于1的数字，因此log对数是小于0的数字，所以求和后为负数，前面再加一个负号，就使得熵为正数了。

现在假设我们想找到一个特殊的概率分布p(X)，我们所知道的所有信息只是关于函数的期望等于常数：，其中是一系列的关于x的随机变量的函数，目标只有一个：在满足这些条件的情况下，概率分布p(X)的**最佳选取**是怎样的？

这里就引入了最大熵的选取原则，最大熵，就是在已知部分知识的前提下，关于未知分布最合理的推断就是符合已知知识不确定性最大或最随机的推断，这是我们可以作出的唯一不偏不倚的选择，任何其它的选择都意味着我们增加了其它的约束和假设，这些约束和假设根据我们掌握的信息无法作出。换而言之，引用吴军的《数学之美》中的一个通俗解释就是：当我们需要对一个随机事件的概率分布进行预测时，我们的预测应当满足全部已知的条件，而对未知的情况不要做任何主观假设。（不做主观假设这点很重要）在这种情况下，概率分布最均匀，预测的风险最小。因为这时概率分布的信息熵最大，所以人们称这种模型叫"最大熵模型"。

巴拉巴拉，说了这么多，其实也可以死记住一点：熵最大的选择就是最优选择。现在我们的任务很简单了，可以看到选取的p(X)概率分布是[p(x1),p(x2),…p(xn)]等等的一个向量的函数，只需要最大化(maximize)公式(6‑16)，现在再加上优化的限制条件(constraint condition)。



st. 和

其中为一组特征函数，而优化中约束的意义就在于这一组特征函数在模型下的均值等于其数据上的均值（为数据分布），也可以将约束条件写为：

st. 

这种条件极值又一次激起了我们的条件反射：拉格朗日乘数法。

定义拉格朗日函数：



(‑)

紧接着就是烂熟于心的求导了：







方程组中仅仅使用第一个方程，则可以见到：



(‑)

公式中的k)T，**x)fxfxfxfx)T，这就昭示了这样一个事实：P(X)属于指数分布族。

## 变分法的启蒙

变分法的源头可以追溯到牛顿时代，从最经典的最速降线问题开始，几位当时顶级的数学高手为变分法做出了开创性的思考，直到欧拉-拉格朗日方程（E-L方程）的确立，才将变分法置于一个不可置疑的基础上。

### Jensen不等式

Jensen不等式在求解变分贝叶斯时得到变分下界时有重要作用，这个不等式也非常简单。首先来看看凸函数(convex function)的定义：

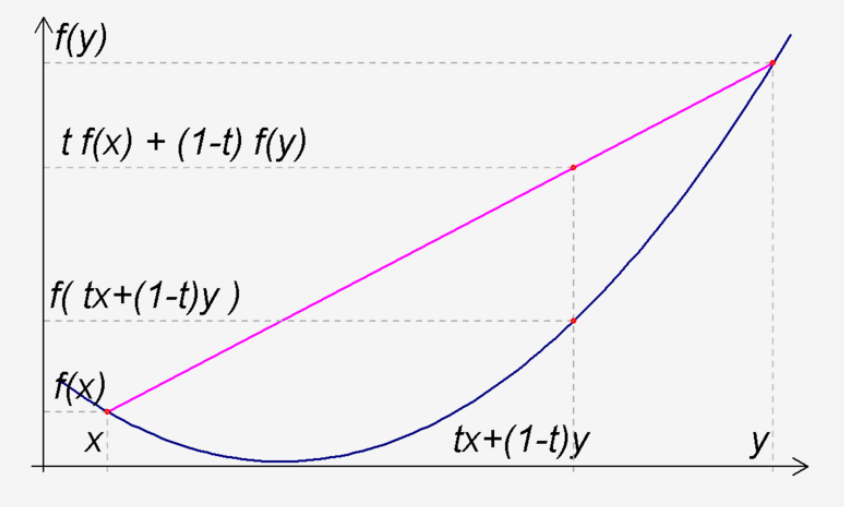


图 6‑3 凸函数（convex function）

凸函数是定义在某个向量空间的凸子集C（区间）上的实值函数f，如果定义在其定义域C上任意两点x，y，以及，有：



( ‑ )

凸函数的函数图像如图 6‑3所示，这个图像有个很形象的解释，(x,f(x))和(y,f(y))两点的连线位于凸函数曲线之上。

了解了凸函数的概念之后，这里首先给出Jensen不等式的定义，再给出其证明。

定义 . Jensen不等式

令f(x)是在一个定义在区间I上的凸函数。如果且且，



( ‑ )

或者简化一点，可以使用期望公式表示（换用表示）：



( ‑ )



( ‑ )

证明：

这个证明的主要思路是数学归纳法，先证明N=1的结论成立，再假设N=k-1时已经成立，证明N=k成立，那么得出结论：整个不等式就证明是成立的。

先来看N=1的情况，此时，由于只有一项，那么，所以

再来看N=2的情况，如果和是两个任意的非负实数，且，f函数的凸性质（也即凸函数公式( 6‑19 )）就保证了：

现在假设N=k-1时已经成立，则设，其中，则：



 //数学归纳假设

 //利用N=2的情况





所以得证。

现在来看一个利用该性质的例子，假设，由于该函数属于凹函数（concave）[[31]](#footnote-31)，因此不等式符号与( 6‑22 )式相反，所以：



这个结论在标准EM算法的推导中被用到。

### 改变世界的方程：欧拉-拉格朗日方程

6.2.2节以及本章之后的小节与LDA变分贝叶斯法的关系不大，这是由于blei使用了简化版的变分EM方法。所以6.2.2节以及本章之后的小节是为了启发其他新的技术思路而设置，读者若较为着急掌握LDA的变分方法，可跳过这些内容直接去看第7章。

欧拉-拉格朗日方程(euler-lagrange equation)确实可称得上是改变世界的方程，变分法从一个17世纪末简单的竞赛挑战谜题线索开始，一步一步深刻影响了物理学和整个世界，而奠基变分法不可动摇地位的就是欧拉-拉格朗日方程。这两位才华横溢的数学家将自己的名字刻入了这个不可置疑的方程，这个方程在物理学的经典定律“最小作用力原理[[32]](#footnote-32)”中也得到应用，这个原理后面不可缺少强大武器是泛函分析的变分法。变分法自然也是微积分发展到泛函时的必然产物（**所谓泛函，就是函数的函数**），如果说变分法和泛函是一座建筑大厦，那么欧拉-拉格朗日方程就像这个建筑大门的钥匙，因此我们首先从这个方程开始起步。

首先直接给出欧拉-拉格朗日方程（E-L方程），接着证明他：



(‑)

这个方程其实是个微分方程（因为其中出现了导数），凡是**满足这个微分方程的解**的函数f必为一系列函数（泛函）中令F的积分到达极值的那个函数。简而言之：泛函极值。

泛函的英文单词是functional，恰好与函数式编程（functional program）是同一个单词，泛函极值是以积分的面目示人，我们接下来证明这个方程：

**证明：**

我们首先说明我们的目标，我们希望能在区间f(a)=A和f(b)=B之间找到一个函数f，这个函数f能令泛函（functional）的积分*J*达到极大值：



(‑)

这里的F就是函数的函数，我们假设F是一阶可导的函数，利用这个假设，我们勇敢地将我们的推理进行下去。

**第一个观察：**F的两个端点分别是a和b，这两个端点是固定死的，而泛函变化的部分在中间，一系列函数中哪个函数让J逼近极值？（如图 6‑3）

**f(a)**

**f(b)**

**a**

**b**

which one?

图 6‑4 泛函积分的函数图

**第二个观察：**现在我们假设其中有一个函数恰好达到极大值！这个函数就是f(x)，那么意味着这样一个事实：只要有一点点**轻微的扰动，便可让该泛函积分J缩小！（或者在f(x)为极小值时，有轻微扰动，就会让J增大）**所以将这个说法用数学的方式表达出来，便是假设一个轻微扰动后的函数：



(‑)

这里的是一个很小的正数，而是一个新定义的关于x的函数，这里的即代表之上轻微的扰动（这便是E-L方程的神来之笔，这里的还可用一个希腊符号简写：表示的变分）。这里的就可看成是一系列实验的可能曲线，而代表取到极值的那个最佳曲线！结合第一个观察，可以看到两个端点是固死的，也即。

现在我们要把这个“轻微扰动”的思想代回到我们的泛函公式(6‑20)：



(‑)

读者应该敏锐地观察到上面公式中的J的下标是，这是因为定积分在区间a和b上积分之后的最终结果中x会消失掉，而变成了的函数。

工作进行到这里，出现了最后一个也是最重要的一个观察。

**第三个观察：**，这个观察的思想是：**如果****恰好是令J达到极值的那一个函数，那么泛函积分J对求导就等于**。（这是众所周知的道理：极值求导），而在的时候恰好，所以就得出了这个观察的结论！

第三个观察也可换一种更为通俗易懂的说法：代表一系列实验的可能曲线，当的时候所有的曲线都会收敛于最佳曲线，而此时J对求导就等于。为什么偏偏要对求导呢？如前文所言，**这是因为J是一个定积分，该定积分的积分变量是x，积分后所有的x都会消失，而变为关于的函数。这便是变分法的精髓所在，用一个来应对形形色色不同的函数。**

有了这三个重要的观察，后面的工作就回到我们熟悉的求导了：



(‑)

这就将我们的求导从J对求导转移到了F对求导：







所以，



当时，，，恰恰在此时，J有极值：



此时对上面积分内的**第二部分使用利用分部积分公式**



等式的第一项使用边界固定点上，所以第一项=0。

等式的第二项则为：



该部分代回原式：



(‑)

最后的一步很简单，因为要满足这个积分式子=0，而是一个函数，取值不一定为0，所以为了保证整个积分=0，必然就使得方括号内的式子=0，这就导致了E-L方程：



得证（Q.E.D.）。

正如众所周知的对函数求导=0就可以求得函数极值一样，泛函极值就通过这个E-L的微分方程求得。

### E-L方程的两种降阶形式

上面的这个公式是E-L方程的最基本公式形式，在向前一步，便可得到平时使用时更为方便的公式降阶形式：

为了方便，我们换一下符号，设，因此泛函变为：，原方程(6‑19)变为：



(‑)

1. 当函数**不显含**时，函数*F*变为。方程(6‑25)变为：



(‑)

1. 当函数**不显含**时，函数*F*变为。泛函积分(6‑20)变为：



设，则

*L*是我们定义的新函数，注意到不显含，因此：



特别注意到上式子中成了新的积分变量。

类比第一种情况，就会发现：这里的L与第一种情况的F类似（只是和符号交换了），因此：





这样就得到了不显含时，E-L方程非常方便的形式：



(‑)

这个形式非常简便易行。已经了解到了欧拉-拉格朗日方程(euler-lagrange equation)公式之后，我们需要将步子迈得大一点，更近一步，在实际情况中，往往碰到的是有限制条件的泛函优化问题，这就需要我们将拉格朗日乘数法和E-L方程进行配合使用。

### E-L方程与拉格朗日乘数法的联姻

E-L方程和拉格朗日乘数法联合使用可以发挥更强大的能力，该方法备受我国的钱三强推崇，这种方法后来在有限元方法中也有广泛应用。我们从6.2.1节所述的第三种观察来推导与拉格朗日乘数法的联姻，我们需要分成两种情况讨论，一种是限制条件G函数中自变量不包含导数的推导；另一种是通用的简洁解决方法。

**(1)限制条件自变量不包含导数的情况**

假设我有两个函数需要优化，y和z，自然而然写出下列公式：



(‑)

在条件的条件写的泛函极值。

由于函数G等于常数0，常数求导必然等于0，另外x不含，因此：



(‑)

由于，，再代回上一步的公式，就很自然得到两者的比例关系（限制条件的作用如此展现）：



将这个比例关系式代回到(6‑28)式子中，取消**，那么就得到：



与前面L方程的推导一样，这一部由于必须要让整个积分结果=0，所以自然而然：



再整理一下（移项可得），可以想到F和y，z均是只含x的函数，因此设计一个就得到：



这里的为何前面要加负号，下一步马上明了：



第一条方程式已经出现。

z也是同理，再加上限制条件，因此得到三个方程式：



(‑)

三个方程，三个未知数：，因此方程组理论上可解出。

**(2)通用简洁方法的解释**

有人或许会问，为何这里的是一个关于x的函数，而非像前面介绍的拉格朗日乘数法那样是一个实数？[[33]](#footnote-33)那我们现在再从另一个角度（更简洁的角度）看待这个问题，还记得拉格朗日乘数法的写作方式吗？又会回忆起这些如同咒语般的死步骤：

|  |
| --- |
| 设给定函数z=f(x,y)和附加条件g(x,y)=0，为寻找z=f(x,y)在附加条件下的极值点，先做拉格朗日函数L(x,y)=f(x,y)+**g(x,y)，然后依次对各个变量求导令其等于0联立方程组求解。 |

现在情况来到了泛函的条件极值，其实方法也是一样的，我们这次将限制条件改为带导数的限制条件G，那么这次的题目就可以如下：

题目： 在限制条件中的最优化的泛函极值。

求解：

仿照一般的拉格朗日乘数法，写出拉格朗日函数，也就是：

就是拉格朗日乘子，请注意到这个式子中F和G函数都仅是关于x的函数，而J是关于y的函数。富有怀疑精神的人会争论，这里的G函数=0啊，因此这个函数与原始的泛函J没有区别！没错，但是后一项仍旧有其意义，主要在于如果做此偏导，就会发现 ，对L的偏导恰好等于，也就是限制条件从拉格朗日函数求导中出现了。

为了解决这个拉格朗日函数问题，我们需要将其写成一般的变分函数问题，也就是将其转变为能用到E-L方程的形式，现在的问题就在于要将弄进积分符号中，因为，所以你可以直接塞到积分符号里面即可！（而需要注意的是积分中需要对各个x都加上,因此变成了）

那么囊括了因素的函数就变成了新的，原来的泛函变为了新的泛函：



(‑)

现在只需要简单地写出根据这个新泛函的E-L方程即可：



(‑)

拿到了这个公式，你现在再来观察上文中的方程组(6‑30)，就会很明白了，看看方程组中的第一个是如何得到的：



(‑)

如果按照这个通用方法来解释，就会如此：





由于G函数=0，可以被凑成积分形式（0的积分还是0），而也被凑入积分，因为针对每个x点都要做限制而变成了函数形式，这一点与(6‑15)式相似。

这时，就会本能察觉到：，因此应用E-L方程(6‑32)，由于G中不含：，因此G对求导=0，那么可以得到方程(6‑33)了。

我们之后通过一个例题加深理解这个技术。

**例 6.4** 一块钢板围成什么曲面做成的半壁容器其容积最大? 如果只考虑底面，就可化成平面问题，定长直线，围成什么曲线使其所围面积最大。(见图 6‑4)

l

图 6‑5 例 6.4的示意图

解：我们很快发现这个是有条件限制（长方形中的长=l）下的泛函极值问题。这就要求我们同时**配合使用E-L方程和拉格朗日乘数法**。

回忆起“弧微分”的公式，就可以将题目中的条件写下：

上图中长方形中长的条件 ，要优化泛函，注意到限制条件可写为，（我们先忽略，因为其位置处在积分符号外），如果连积分符号带被积函数整个直接塞入(6‑31)，由于限制条件本身就带有积分符号（而且积分上下限相同），那这么做未免生硬，其实可以直接如下处理更简单：



 ，所以

这就得到了在均包含积分符号时候的简便的拉格朗日函数方法的构造方法，即直接提取出上面2个公式中的被积函数，其中的**便是拉格朗日乘子，特别注意到这个**是一个实数而非函数，因为限制是加于整个限制条件之上而非限制在每一个x点。后续的处理与标准的拉格朗日乘数法有所不同，因为泛函不需要分别对拉格朗日函数中每个变量求导（求导得0解方程那是普通函数求极值的做法！），但替换为直接将这个函数代入E-L方程即可。

注意到这个方程不显含，根据降阶的欧拉-拉格朗日公式(6‑27)，则可以顺利写出下面的公式：







我们画出图 6‑5。

(C1,C)

x=a拉格朗日函数

拉格朗日函数

**

**

图 6‑6 解出的圆的图像

回忆起三角函数的世界，抛却那些复杂而灵巧的三角函数公式变换，弱水三千只取一瓢，只需要回忆起1个公式即可：

由于我们看到这个公式中包含，这就触发了我们引入三角函数换元的灵感：

设,则原公式变为：(你应该知道)



所以

由于

两边同时积分

这样就解出了x和y的参数方程组：



如果对中学的圆的参数方程还有印象，回想起： ，这就把上面的方程组化为下面的形式：



这就恰好是圆的标准方程，圆心点在(C1,C)，半径是**，通过x=0和x=a两点。最后因为限制条件内的半圆周长=l，所以。

**例 6.5** 上一个例子的限制条件和泛函交换一下呢？

优化目标： ，限制条件：

也就是说：什么样的曲线下包围给定面积的情况下最短？

跟上一例一样，提取出被积函数,构造拉格朗日函数：



然后代入降阶的欧拉-拉格朗日公式(6‑27)：





设,则原公式变为：(你应该知道)



跟上一例子一样，得到x如下：



最后同样得到圆的标准方程：



还是圆弧！也就是说：如果曲线下方包围的面积给定了，这个曲线在圆弧时候达到曲线长度的最小值！因为可以想象，这条圆弧划出了足够的高度，而又圆满地包围了这个面积。

如果读者足够细心，会发现上例和本例有个区别，一个求极大值，另一个求极小值，没错，E-L方程与普通函数判定极值时所用的求导得0方法一样，无法判定求得是极大还是极小，而仅能判断求得的是极值。

从这道例题中学到的一点便是：如何将欧拉-拉格朗日方程（E-L方程）与拉格朗日乘数法配合起来使用，这招在后期的变分贝叶斯推导中也有相应使用。

更多的例子请看：<http://www.exampleproblems.com/wiki/index.php/Calculus_of_Variations>

### 变分贝叶斯法的推导

现在我们暂且抛却变分法辉煌的过去，将着眼点限于变分贝叶斯法，如果说变分法就像牛顿力学的三大定律（事实上拉格朗日用变分法确实发展出分析力学），那么变分贝叶斯法相当于成功应用力学定律制造出的飞机、火箭。在这个经典例子中，也可以清晰地看出历史上前人伟大的智慧是如何影响现代科技的。

是时候引入我们的变量符号了，我们再次引入了**隐变量Z，观测数据D**，如同前面所述的那样，隐变量Z是我们问题的求索目标。变分法的启发了我们这样一个想象**：能否找一个近似于“隐变量的概率条件分布P(Z|D)”的“分布Q(Z)”来作为我们的近似分布**？如果能找到这个近似分布Q。后面就可以对这个分布进行求导，进行分析，进行你所用到的一切关于概率分布和函数的知识进行解剖了，起点的关键是需要找到这么一个Q。

当然很多时候很难找出这么一个非常近似的分布，所以我们只需要“尽可能近似“即可，接下来的问题是如何衡量近似概率分布Q(Z)和P(Z|D)呢？如何表示“尽可能”一词呢？

首先，找一个不错的距离公式作为我们的出发点，我们选择一个经常使用衡量两个概率分布之间距离的公式：KL距离。



**( 6‑38 )**

初看起来如此平淡无奇的一个KL距离公式怎么会成为推导变分贝叶斯的关键要素呢？到后面你就会看到，选择这个公式衡量距离是一个聪明的选择。

我们接着KL公式将我们的方法向前推导，先变换一下KL距离公式的写法。



（此部分与后面的第7章LDA变分贝叶斯关系不是很大，作为启发新技术设置，未完待续…）

## 参考文献

1. 季泉生. 通俗简易讲解变分问题 PPT. 2013.12.13
2. 华俊豪. 基于变分贝叶斯方法的医学图像分割. 浙江工业大学本科毕业设计2013
3. Gilbert Strang. 7.2 Calculus of Variations. 2006
4. Aniket C. Aranake. Notes on the Calculus of Variations and Lagrange Multipliers. 2011.02.13
5. Martin J. Wainwright, Michael I. Jordan. Graphical Models, Exponential Families, and Variational Inference. Foundations and Trends in Machine Learning. 2008:1-305
6. David M.Blei. Variational Inference(讲义)
7. Dimitris Tzikas, Aristidis Likas, Nikolaos Galatsanos. Life After the EM Algorithm: The Variational Approximation for Bayesian Inference.
8. Christopher M.Bishop. Pattern Recognition and Machine Learning. Springer. 2006:Chapter 10(Approximate Inference)& p.490
9. Joel Hass, Maurice D.Weir, George B.Thomas, Jr.李伯民译 托马斯大学微积分. 机械工业出版社2009.03:p.673 – 678
10. 陈希孺. 概率论与数理统计. 中国科学技术大学出版社. 2009.2.1. p.103,p.110

# LDA的变分贝叶斯法

如果说Collapsed Gibbs Sampling版本的LDA犹如《新约圣经》，那么Variational Bayes法推导的LDA就像《旧约圣经》一样，如果说一个人学习LDA却不知道LDA的变分推断方法，就像一个基督徒不读《圣经》一样，这仍然称不上了解LDA，溯本求源是我们应有的态度。LDA的变分贝叶斯法既是LDA模型的起点，也是一个启发新技术的契机。

## Latent Dirichlet Allocation的另一个视角

“A wise man never knows all, only fools know everything.”

--非洲谚语

如果你已经学过Gibbs Sampling版的LDA，从这一章开始，我需要你洗净脑袋，不要以为知道关于LDA所有的事情了。在这一章里，以David M.Blei的LDA原始论文《Latent Dirichlet Allocation》为依据重新组织撰写[[34]](#footnote-34)，但我会补充进许多原始论文里没有的技术细节，以帮助读者理解。

尽管在前面第3章的3.2节已经详尽介绍过了LDA算法的生成规则，这里需要说明的是，历史上第一次出现的LDA算法，在Blei原始经典论文中一开始的LDA的概率模型图不是这个样子(basic版本)。图 7‑1给出smoothed和basic两种版本的概率模型图，应该特别注意到节点的不同，在basic LDA的模型图中，节点直接被用来重复采样生成每个词。

|  |  |
| --- | --- |
| http://img.blog.csdn.net/20141120172828681 |  |
| 图 7‑1 左图：smoothed版本LDA模型图 | **右图：basic LDA模型图** |

在这个故事里，LDA的**生成咒语**如下：

1. 选取文档长度（文档中的单词个数）
2. 选取文档中的主题分布

如前面章节所述，这里的是Dirichlet分布的超参数，一共维（而是设置的主题总数），而是产生该文章主题的多项分布的概率参数，其中是第i个主题被选择的概率，从Dirichlet产生参数后，再用去产生具体主题z。

3．上面两步完成后，进而产生一篇文章N个词里的每个词：

For each word  in  words:

(a)选取一个topic 

(b)选取一个word  from 

请读者格外注意：这里的的含义与前文的Gibbs版本**有所不同**，它是一个多项分布的参数，是一个的矩阵，表示从Zi到Wj的产生概率即：



( ‑ )

这与Gibbs版本的矩阵含义相同，**但是，这里的不是从某个分布（在Gibbs版本中，）中抽取的随机量，而是一个确定的量（fixed quantity），这个确定的量就是我们模型中将要估计的**。另外，在推导之前还有些简化模型的工作要做，第步只是生成文档的长度，Possion分布假设不是很重要[[35]](#footnote-35)，所以可以使用真实的文档长度分布（document length distribution）来代替。还有，注意到N（文档的长度即单词个数N）与其他变量（和z）无关，所以只是一个辅助变量，我们会在后续的推导中忽略这个变量的随机性。

这个推导故事中的z指定的使用与Gibbs版本略有不同，所以如果熟悉Gibbs版本的读者需要稍微重新认识一下LDA，所有的变量的具体含义见表 7‑1。

表 ‑1 basic LDA各个变量的解释

|  |
| --- |
| K：主题个数(用户设置)  M：训练预料中的文章数  N：一篇文章内的单词数  V：语料中所有使用到的单词构成的词典单词数  w：word单词，这个版本的w代表文章中的具体单词，具体使用时是这样的： 表示第d篇文档里的第n个单词[[36]](#footnote-36)，因此是个三维矩阵（三维数组）形式存储<d,n,v>（第d篇文档的第n个单词是单词v（单词表的word\_id=v））。你应该注意到公式( 7‑1 )里的里的j也是上标，这是因为在这个版本里我们将word使用one-hot的**向量**表示（比如在{1,2,3,…,V}中只有用到的单词v为1，其他皆为0，因此，而，）  z：对每个的单词的主题的分配指定。如同w，这个版本里z也与前面介绍的Gibbs版本略有不同，主要区别在于这个z是一个三维的矩阵，存储方式为<d,n,i>，表示第d篇文档里的第n个单词是否是编号为i的主题。  ：与Gibbs版本一样，文章的主题分布  ：与Gibbs版本一样，的超参数，  ：表示从主题到单词的产生概率。它是一个多项分布的参数，是一个的矩阵，表示从到的产生概率即，这与Gibbs版本的矩阵含义类似，但有所区别的是**是确定的量（fixed quantity）**。  Doc：文档使用，这里的是第n个单词（正如前文所述，每个word是个向量）  Corpus：训练语料库由M篇文档组成， |

## 分析模型

解决这个模型的关键之处在于求解给定一篇文档时的隐变量(latent variable)的后验分布（posterior distribution）：



(‑)

这个公式是LDA变分推导的症结所在，我们这里的做法是先分析每一篇独立的文档，整个语料集的概率就是将一个个独立的概率乘起来，这也就意味着文档之间的概率相互独立。

首先我们观察公式(7‑2)的分子，根据模型图 7‑1右图将分子分解开：



( ‑ )

我们一项一项来看，这里的代表了给定一篇文章的长度为N（单词个数为N），其中每个词从的概率矩阵中按照每个词的主题分配z选取，在该情况下的该文章的概率。所以我们可以将其分解成每一个单词(word n)的概率相乘起来：



( ‑ )

接下来，就是向量中第i个topic主题编号分量的概率，该概率使得当前文章第n个单词的第i个topic的主题分配。最后来源于Dirichlet分布的公式(2‑10)。

综上所述，现在已经可以推导出给定参数和时，文档的主题分布、topic分配的集合、和**一篇文档内的单词集**的联合概率公式：



(‑)



( ‑ )

这里的代表向量中的第个分量，我们可以重新将其表达为指数上标的表示法：利用整个字典长度V，由于，（对语料词典中所有单词连乘中只有当前文章第n个单词）,同理可得，因此公式( 7‑6 )也可以写为：



( ‑ )

为了加深理解，图 7‑2进一步做了解释：

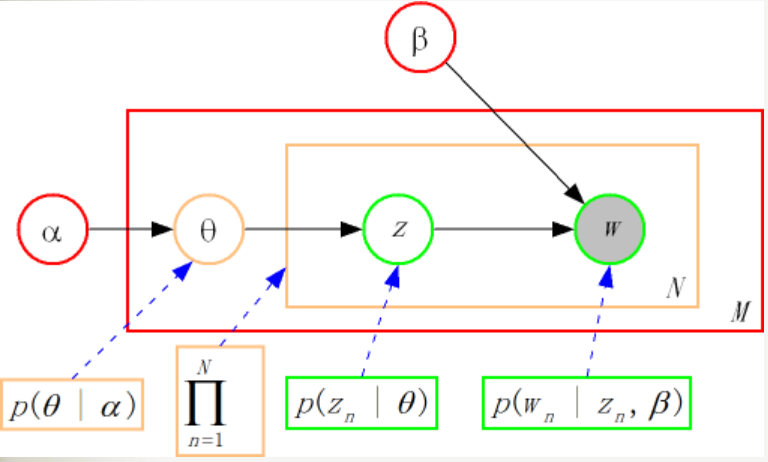


图 7‑2 生成一篇文档概率的解释图

在图 7‑2中，LDA的三个表示层被用三种颜色表示了出来：

1.corpus-level (红色)：和是语料级别的参数，也就是说对于每个文档都是一样的，因此在生成整个文档集(corpus)过程中只需要采样(sample)一次。

2.document-level (橙色)：是文档级别的参数，意即每个文档的参数是不一样的，所以对于每个文档都要采样(sample)一次。

3.word-level (绿色)：最后z和w都是单词级别的变量，单词级别的变量w和z为每篇文档的每个单词采样(sample)一次。

现在我们来看(7‑2)的分母：对公式(7‑5)中的求积分，对z的可能值求和，消去和z之后，就可以得到**一篇文档**的边缘分布。



(‑)

公式(7‑8)中的是指**一篇文档**上的第n个单词的主题分配，N表示该文档的单词总数。

此处可以进一步理解下公式(7‑8)，公式(7‑8)中将z的所有可能值求和就可以消去z，这实际上就构成了给定一篇文档中单词（word）的概率，因此求和符号sigma里那一部分便可以写成：



(‑)

注意到这个概率是一个随机变量因其依赖于，我们现在可以定义下面所述的对于一篇文档的生成过程（generative process）：

1. 选取文档的主题分布
2. For each word  of N words:
3. 从概率中选取一个单词

这个过程也就将文档边缘分布以连续混合分布（continuous mixture distribution）定义出来[[37]](#footnote-37)：



(‑)

可以发现公式(7‑8)实际就是公式(7‑9)代入公式(7‑10)之后的结果。公式(7‑10)中可看成是连续混合分布中的混合成分（mixture components），可看成是混合权重（mixture weights）。

最后，将各个单文档的边缘分布乘起来（共M篇文章），就可得到整个语料集的概率：



(‑)

公式(7‑11)中的指的是第d篇文章中的第n个词的主题分配。

如前所述，解决这个模型的关键之处在于求解(7‑2)，但不幸的是，这个公式非常难以求解（intractable）：分子是公式(7‑5)，而分母是公式(7‑8)，问题就在于这个分母，仔细观察分母，可以将公式(7‑8)替换为具体的模型参数，便将公式(7‑8)写成：



(‑)

公式(7‑12)的一点解释：在这里使用公式(2‑10)替换掉公式(7‑8)中的，并使用了式(7‑13)和(7‑14)的替换（这个过程类似于公式( 7‑7 )）。



(‑)



(‑)

由于公式(7‑12)在对潜在主题z的求和sigma中出现了一对和，所以造成难以求解（intractable）（Dickey, 1983）[1]，由于精确推断（exact inference）不可得，所以Blei决定另辟蹊径，采用近似推断（approximate inference）。

## 推导

### 启蒙

我们先来看看标准的变分贝叶斯（Variational Bayes）求解方法：

变分推断（Variational Inference）的目的是要找出最接近于真实模型的隐变量后验概率分布p的那个近似分布q，然后用q来替代p，那么问题来了，“接近”这个概念如何在具体计算中体现，也就是说如何衡量两个概率分布的相似度，幸运的是，这已经有了现成的工具：通常使用KL距离。稍做一点简单的推导，就会发现“优化变分下界”（Evidence lower bound/ELBO）的诀窍。

近似分布q到真实分布p的KL距离的距离公式，这里使用q代表近似变分分布，p代表真实分布（推导中的D是指可观测到的数据Data，Z是指不可观测到的隐变量）：





由于，因此应用乘法分配律：



式子右边的被称为已知/观测到的数据的对数证据(evidence)或对数似然函数(log-likehood)，此时将移项到等式左边，即可得到：



( ‑ )

( 7‑15 )式中的便是所谓的变分证据下界（Evidence lower bound/ELBO），应用函数期望公式( 6‑8 )，所以可以将ELBO写作:



( ‑ )

我们从另一个角度看下这个ELBO的下界公式( 7‑16 )，回忆起EM算法推导中出现过的Jesen不等式，如果f是一个凹函数[[38]](#footnote-38)（concave），那么根据Jesen不等式：



( ‑ )

如果读者没有学过Jesen不等式，请回顾第6章6.2.1节。

因为对数函数log属于凹函数（concave）:









因为是不等式的大于等于关系，所以ELBO确实是的下界(bound)。从式( 7‑15 )就会发现最小化(minimize)KL距离等价于最大化(maximize) ELBO式( 7‑16 )，即：



( ‑ )

因此推导思路转向研究如何选取q函数使得ELBO最大，也就是所谓的泛函极值问题。经过漫长复杂的推导，将会得到这个q函数。如果做出这个推导并观察规律会发现，在p分布函数为指数分布族的情况下，求得的q函数也会为属于指数分布族的同样的分布函数。（譬如p和q都服从Dirichlet分布）。

### 变分目标函数

Blei觉得上文中标准的变分推断（variational inference）比较复杂，他使用了较为简单的方法：convexity-based variational inference。Blei首先将原有的LDA模型图进行了简化，便得到了变分分布模型作为原模型的近似替代。为了方便，这里将原始LDA的模型图重新画下来，并画出用于替代的变分分布模型图（见图 7‑3）。

|  |  |
| --- | --- |
|  |  |
| 图 ‑3 左图：LDA模型图 | 右图：用于近似LDA后验分布的变分分布模型图 |

正是因为和通过节点、z和w的连接边耦合（couple）在一起，造成难以求解，又由于原分布采用了指数分布族的Dirichlet分布作为的先验分布：、多项分布（Multinomial Distribution）作为z服从的分布：，那么就意味着变分分布函数也应该如此。仍然采用Dirichlet分布和Multinomial分布作为先验分布，绕开标准的变分贝叶斯，直接假设出一个变分分布函数q函数，接着我们就回到了我们熟悉的函数求极值方法：求导。

经过上述分析，只要简化原有模型，去掉多余的边和节点（比如节点w）这个被简化的模型如图 7‑3右图所示。用简化的变分模型的目的是要找出这个最接近于真实分布的变分模型的参数，类似于变分法找出最接近真实极限函数的那个函数。

值得注意的是，图 7‑3右图简化模型中引入了两个新变量：和，用于替代原有的超参数。在这个变分分布中，，所以这里的维度与相同，是文档级别（document level）的参数，每篇文档对应的不同，但都为长度的向量。而，所以的维度与z相同，是三维矩阵<d,n,i>。为了简便，后面的公式书写中省略了向量的箭头，用加粗代替。

按照简化后的变分模型图 7‑3右图，变分分布的分布函数如下：



( ‑ )

式( 7‑19 )中的N指的是一篇文章的词数，这里的Dirichlet分布参数和多项分布（multinomial distribution）参数均被称为自由变分参数（free variational parameter）。现在工作转向寻找最优化的变分参数和，也就是找到使得变分分布和真实分布KL距离最小的最优化的参数：



( ‑ )

如同( 7‑15 )的推导，我们可以得到已知数据的对数证据（evidence），如下：



( ‑ )

其中代表当前的一篇文章，回忆起等价关系( 7‑18 )，此时我们可以转向优化变分下界。

### 下界（lower bound）

最基本的思路与标准变分贝叶斯方法一样，Blei首先利用Jesen不等式找到对数似然函数（该对数似然函数也被称为evidence）的可调节的ELBO下界，与公式(7‑5)到公式(7‑9)推导一样，为求简洁，在q函数的书写中中省略了和，将该对数似然公式写出：





可以观察到上式实际就是第6章的二维随机变量函数的数学期望公式( 6‑13 )：



利用Jesen不等式，可得下界：



( ‑ )

( 7‑22 )式便是下界（lower bound）了，现在先来看( 7‑22 )式最右边一项，这里我们做平均场假设，也就是假设可以因式分解，可以得到：



( ‑ )

值得注意的是，实际情况中，主题的分配z的与 文章的主题分布并不是独立的，这也就造成了一定问题，Blei管不了这么多了，现在不是讨论这个的时候，他只需要大胆地将他的逻辑向前推进，而将其中的小问题留给了后来人来解决。根据对数函数的性质，所以可得：



( ‑ )

来看等式( 7‑22 )的左边那一项，这一项的分布为真实分布p，观察图 7‑3的LDA模型图可得到因式分解：



( ‑ )

又利用到关于期望的性质公式( 6‑7 )，便可以将下界写为：



( ‑ )

注意到该式一个鲜明的特点是同时融合了模型参数和变分参数，也即图 7‑3左图的参数和，以及右图的参数和同时出现在里面，这就为我们后期的迭代优化提供了契机。

### 下界展开

还记得那句台词吗？“集齐七颗龙珠，可以召唤神龙”。公式( 7‑26 )中一共5项，现在需要集齐这5项的展开表达式，我们逐一来看。

**第1项 ：**

需要注意到是服从Dirichlet分布的，因此：



所以：



又因为q函数不含，所以可以将包含的项从期望中提出：



再利用第6章的公式(6‑14)可以轻松得到：



因此，第一项最终得到：



**第2项：**

回忆起表 7‑1中对z的解释，但是这里的变分推断只针对一篇文档分析，用表示当前文档分配第n个词被指定到第i号topic，否则，。并且利用，就可以直接展开第2项。





 //



 //



**第3项：**







 //q函数不含

**第4项：**

 //



**第5项：**

由于，其中第n个词的topic指定是i号topic时，否则，可将提到指数，仿照( 7‑7 )式的处理，可得：











**最后，五项齐全，可以写出下界：**



(‑)

你一定认为Blei已经迷失在变分下界函数的汪洋大海了，其实得到这一步之后，后面的推导已经非常简单，在下面的小节中，将会展示如何寻找最优变分参数和，令其最大化maximize变分下界。

### 变分推断

其实Blei这个版本的变分推断更像是变分贝叶斯和极大似然估计混合的版本。迭代中的第一个步骤被称为变分推断，第二个步骤被称为参数估计。这个方法其实就是变分（variational EM），不了解普通算法的读者请参考Bishop写的《Pattern Recognition and Machine Learning》的9.2~9.4节（2010），在步骤，我们使用近似于原始LDA后验分布的**变分分布**去寻找变分参数（variational parameter）的最优解。在步骤，使用步骤中已获得的变分参数估计去估计（比如极大似然估计（parameter estimate））原模型参数（model parameter），从而进一步最大化模型的下界（bound）。具体到LDA中：

步骤：目标找到最优的变分参数和以使得下界函数最大化，见图 7‑3右图所示，这一步的和是document level，因此只针对一篇文章优化。获得这些参数之后可以去计算完整数据的log-likehood的期望（expectation of the log likehood of the complete data）。

步骤：是假设已知变分参数和去估计模型参数和，这一步的和针对整个语料集优化。使用极大似然估计的估计似然函数为：



我们先来看E步骤，需要注意到在下界公式(7‑27)中，注意该公式指的是一篇文章的变分下界。

1. **maximize wrt. **

先来研究，也就是第n个单词是被第i个潜在topic产生的概率。应该特别注意到一个词在不同主题的概率之和等于1，即限制条件：



( ‑ )

注意该限制条件与Gibbs版本的归一化，即第3章的公式(3‑22)不同。

将变分下界公式(7‑27)只留下有关的项，令为令的合适的那个单词v，也就是令的v。由于存在限制条件( 7‑28 )，利用拉格朗日乘数法求解条件极值，加入拉格朗日乘子：



(‑)

关于求导(其中一项用到导数的乘积法则)：



令其等于0，可得：



将移到等式右边，并整理可得：







( ‑ )

注意此式中无需考虑，将其忽略，因其对所有的topic i而言此数字均相同。

1. **maximize wrt. ：**

现在轮到参数，其中，下界函数只留下有关的项：



( ‑ )

公式( 7‑31 )已经得到，如果读者去看Blei的《Latent Dirichlet Allocation》论文原文，会发现Blei在这里的处理有点问题。

关于求导，忽略其他无关参数，这样就消除了一些sigma符号。





令其等于0：



所以根据这个方程只要令取值为( 7‑32 )式，便可以令下界得到最大值：



( ‑ )

可以注意到计算的公式( 7‑30 )的表达式中出现了，而计算的公式( 7‑32 )的表达式中出现了，因此两者交替轮流进行计算，直到下界收敛**。**

### 参数估计

参数估计(parameter estimation)就是变分EM的M步骤，在这个步骤中，主要任务是利用E步骤已获得的和来获取模型参数和。我们仍然使用变分下界作为真实语料的边缘似然函数（marginal log likelihood）的替代。另外，这一节估计的和是corpus level的参数，因此似然函数（log-likehood）就是将单个文档的似然函数加起来，而且，总体的变分下界也是单个文档变分下界加起来。在这一节里，总体的变分下界的符号我们仍然“滥用”了上一节的变分下界符号L。

**1．Maximize Lower Bound wrt. **

与上一节一样，L下界函数只留下有关的项，另外，ij表示从主题到单词的产生概率，而topic i下所有词产生概率之和=1，因此有限制条件：



( ‑ )

应该注意到这个限制条件与式( 7‑28 )在求和公式上的区别。限制条件意味着需要使用拉格朗日乘数法的公式(6‑15)，并使用d作为文档编号的下标，写出L下界函数：



( ‑ )

令其关于求导，令其等于：



( ‑ )

在式( 7‑35 )中，的下标，代表文档d的第n个词，是示性函数，表示当满足括号内的条件时返回1，否则返回0。



因为，因此我们可以忽略拉格朗日乘子，直接使用归一化公式：



( ‑ )

**2．Maximize Lower Bound wrt. **

L下界函数只留下有关的项：



关于求导，为避免冲突，将上式中的改为：



( ‑ )

可以看到这个求导表达式依赖于，而，因此的方程无法直接解出来，必须使用迭代算法来近似估计。Blei从故纸堆中翻出了牛顿的珍宝：牛顿-拉菲森迭代法。我这里简要讲述下blei对的处理办法，而牛顿方法的详细故事留在另一章讲述。

这里直接给出迭代的公式（该公式的推导留给牛顿迭代法一章）：



( ‑ )

现在迭代需要的所有的公式都已经齐备，在下一章中，将着重讲解代码实现。

## 误差讨论

一个优秀的研究者在阅读前人论文时应该带有一些批判性的精神，分析到前人成果的不足，这样才能提出自己新的思想。事实上，如果在推导中提高警觉，便可很轻易地发现推导中存在的一些造成误差的地方，这里简单列举一下。首先，第一个误差出现在公式( 7‑23 )，前文已述。另一个不易发现的误差出现在下界展开的第2项和第3项、第5项的展开式推导中，如果读者的概率论基础足够扎实，会注意到关于期望的一个事实：若干个**独立**随机变量之积的期望等于各变量的期望之积，即：



( ‑ )

因此第2项的展开推导中的拆开为式存在一定误差的，因为其实际情况中主题的分配z的与 文章的主题分布往往并不是独立的；第项的展开也是同理，拆开为的过程中，表示从主题到单词的产生概率，也与主题分配z存在一定关系，两者不能完全视为独立；第5项的展开推导中的拆开中，由于更是让这项误差较大。

可以说，Blei推导LDA时做了一些妥协，对某些量进行了近似处理。正如一代物理学家保罗·狄拉克[[39]](#footnote-39)所言：“原先，我只对完全正确的方程感兴趣。然而我所接受的工程训练教导我要容许近似，有时候我能够从这些理论中发现惊人 的美，即使它是以近似为基础……。如果没有这些来自工程学的训练，我或许无法在后来的研究作出任何成果……我持续在之后的工作运用这些不完全严谨的工程数学，我相信你们可以从我后来的文章中看出来……那些要求所有计算推导上完全精确的数学家很难在物理上走得很远。”

狄拉克之言机器学习上也是同样适用的，理解了“允许近似”这一点，将会在机器学习之路上走得更远。

## 参考文献

1. David M.Blei & Andrew Y.Ng. Latent Dirichlet Allocation
2. Wayne Xin Zhao(batmanfly@ gmail.com) Note1: Variational Methods for Latent Dirichlet Allocation. Spring 2013
3. Zhao Zhou. Variational Inference for LDA. The Hong Kong University of Science and Technology. (pdf演示文稿)
4. Colorado Reed. Latent Dirichlet Allocation: Towards a Deeper Understanding
5. Mohammad Emtiyaz Khan. Exchangeable Sequences, Polya’s Urn and De-Finetti’s Theorem. CS,UBC. October 6, 2007.
6. JOS´E M. BERNARDO. The Concept of Exchangeability and its Applications. Bhattacharya Memorial Volume. December 9, 1996.
7. David M.Blei. Variational Inference(讲义)

# LDA变分EM实现

对于实干家来说，来到了令人兴奋的一章，这一章将系统性地介绍Blei的LDA的C语言源代码的实现。回忆起第一次写程序的情形，运行一段代码，兴奋地看到“hello world”出现在闪闪发光的绿色屏幕，本章也将令人兴奋的LDA变分EM搬上电脑屏幕，这也算是对前文漫长的推导的一个补偿。

如果初看Blei版本的源代码，初学者往往会迷惑于代码中一些精妙的地方。所以这一章的安排是这样的：首先根据前文推导得到的迭代公式写出一个伪代码的框架，后面深入剖析Blei的C源代码。

## 伪代码

回忆一下上一章，变分LDA重要的步骤就是变分EM，为了加深记忆，在此处重新写出这些重要的公式。

(1)E-step:

对一篇文档，利用初始化或先前迭代已估计的和去估计参数和。





(2)M-step:

最大化variational inference中的下界，求出此时的和。





其中在M步骤中被反复迭代得到估计值。

不浪费时间，首先直接写出伪代码：

表 ‑1 LDA的变分EM算法步骤

|  |
| --- |
| **输入输出数据：**  输入：主题个数K  包含M篇文章的语料集，每篇文章d拥有个单词  输出：模型参数：，，z  **算法步骤：**  初始化，其中i代表共K个主题中的一个主题编号，n代表个单词中其中一个index。  初始化，其中i代表共K个主题中的一个主题编号。  初始化。  初始化，其中i代表共K个主题中的一个主题编号，j代表语料词典共V个单词中的一个。  //E-step（确定和以便于计算似然函数）  loglikehood := 0 //似然函数初始化0  **for** d = 1 to M{ //遍历语料库的所有文章  repeat{  for n=1 to Nd{ //遍历一个文章的词  for i =1 to K{ //遍历所有主题编号    }endfor  将归一化（normalize），使其之和=1  }endfor    }until 和收敛  loglikehood := loglikelihood +  }**endfor**  //M-step(极大化变分分布的log似然函数)  **for** d = 1 to M{ //遍历语料库的所有文章  for i=1 to K{ //遍历所有主题编号  for j=1 to V{ //遍历语料库中的词典所有单词编号    ｝endfor  将归一化（normalize），使其之和=1  ｝endfor  ｝**endfor**  通过迭代公式( 7‑38 )计算  if(loglikelihood收敛) then{  return parameters  }else{  Go back to E-step  }endif |

## 工程优化分析

可以看到上述迭代算法中反复出现了归一化（normalize）的操作。这在具体工程实践时采用了一个数学上的小技巧，这便是鲜为人知的log\_sum技巧，给出log(a)和log(b)，返回log(a + b):







有了这个小的技巧，在归一化计算的时候，换用对数来归一化就可以加快运算速度：计算出log(),log(),log(),…,log()之后，利用此技巧就可以不断两两迭代计算出log(，最后要归一化的公式如下：



( ‑ )

采用这种优化后，在迭代计算时也换成了log，因此迭代公式换成了：



( ‑ )

另一个工程上的优化是求解时的优化，考虑到求解公式( 7‑32 )时已经得到了上一轮迭代的信息，因此可以利用这个信息加快速度。来看看Blei的代码是如何对优化的，Blei使用了下面的公式：



( ‑ )

其中上标t表示第t轮迭代，而下标i表示第i个topic，其中。表示该文章中第n个词的计数统计(count)。

公式( 8‑3 )是如何得到的呢？细致想来其实不难，首先第一轮初始化时：



此后t+1轮迭代时必须使用如下迭代公式：









公式( 8‑3 )由此得到，公式( 8‑3 )是一个递推公式，由于第一轮初始化时有信息，后面的递推公式才能成功地去除了。另外由于只在第d篇文档中求解操作，因此忽略的d下标。

## Blei的变分LDA（C语言版）源代码剖析

### 源代码下载

在阅读这一节的源代码分析时，我强烈建议读者去下载一份Blei的C语言版源代码去阅读，github上的项目网址为：<https://github.com/Blei-Lab/lda-c>

SSH的clone的url为：git@github.com:Blei-Lab/lda-c.git

### 使用命令与配置

1.输入格式

得到了伪代码，下面以Blei原版C代码进行细致分析，首先来看下Blei版LDA的输入格式：

[M] [term\_1]:[count] [term\_2]:[count] ... [term\_N]:[count]

其中M是该文章的去重后的单词数，count是每个单词出现的次数，term\_x是每个单词的wordid，这不是string。

2.命令行参数

编译后之后，就可以在shell调用执行了，训练的调用命令如下：

lda est [alpha] [k] [settings] [data] [random/seeded/\*] [directory]

其中[random/seeded/\*]代表主题生成方式，“random”随机生成每个主题；“seeded”从一个随机选取的文章中平滑地生成主题分布；或者你可以指定一个以前已经训练好的模型去初始化模型（传入已经训练好的模型名）。

此外，在已经训练好的模型基础上，可以推断(inference)新文章的主题模型，命令如下：

lda inf [settings] [model] [data] [name]

在新数据上执行变分推断时使用已训练好的[model].\*文件。

之前训练好的[model].\*文件将会包含以下文件：

1．<iteration>.other包含；

2. <iteration>.beta包含主题分布的log对数，每一行代表一个主题；在第k行，每一个元素是log(p(w|z=k))

另外有两个文件将会被创建：[name].gamma是每篇文章的变分狄利克雷参数（variational Dirichlet parameters），也即doc🡪topic分布的矩阵文件，每一行是一个doc，而每一列是一个topic的概率；[name].likelihood则是每篇文章的似然函数的下界。

3.配置参数配置

参数配置放置在settings.txt文件中，共有以下设置项：

var max iter [integer e.g., 10 or -1]

var convergence [float e.g., 1e-8]

em max iter [integer e.g., 100]

em convergence [float e.g., 1e-5]

alpha [fixed/estimate]

详细解释如下：

[var max iter]

一篇文档的变分分布的坐标上升法（coordinate ascent variational inference）最大的迭代次数。如果设置-1则为完全执行变分推断，直到变分收敛条件被满足。

[var convergence]

变分推断的收敛条件。(score\_old - score) / abs(score\_old)如果小于这个数值，则停止迭代（在迭代超过最大迭代次数之后）。请注意这个参数是在一篇特定的文档上的“下界”。

[em max iter]

变分EM的是最大的迭代次数。

[em convergence]

变分EM的收敛条件。(score\_old - score) / abs(score\_old)如果小于这个数值，则停止迭代（在迭代超过最大迭代次数之后）。请注意这个参数是整个语料集的似然函数的“下界”。

[alpha]

有两个设置选项，如果设置成[fixed]，则alpha在迭代中不更新。如果设置成[estimate]，则alpha会与topic distributions一起在迭代中被估计。

### 源代码情况一览

表 8‑2 源代码中用到的变量一览

|  |  |
| --- | --- |
| var\_gamma | double二维数组，存储容量M X K，doc-topic分布，每篇文档都会计算其topic分布 |
| phi | double二维数组，存储容量N(最长的一篇文章的单词数) X K，word-topic分布，针对每篇文档，计算文档中每个word的topic分布 |
| alpha | double类型，只是一个浮点数（由于对称超参数处理） |
| class\_word | double二维数组，存储容量K X V，变量 |
| class\_total | double一维数组，存储容量K，为了将归一化而设置 |
| log\_prob\_w | double二维数组，存储容量K X V，变量 |
| lda\_model | lda的模型参数，里面包括beta以及alpha |
| lda\_suffstats | 记录统计信息，比如每个topic上每个word出现的次数，这是为了计算lda\_model而存在 |
| corpus | 语料集的全部文档信息 |
| document | 文档的具体信息，包括word信息 |

表 8‑3代码文件与解释：

|  |  |  |
| --- | --- | --- |
| 文件名 | 典型函数 | 功能解释 |
| lda.h | typedef struct lda\_suffstats; … | 代码中用到的数据结构 |
| lda-data.c | read\_data | 读取训练语料集的文档，返回corpus结构体指针 |
| lda-model.c | corpus\_initialize\_ss/random\_initialize\_ss | 初始化LDA的统计信息 |
| lda\_mle | LDA变分EM的M步骤 |
| lda-estimate.c | run\_em | 变分Em算法框架 |
| doc\_e\_step | LDA变分EM的E步骤 |
| read\_settings | 读取setting.txt配置文件 |
| infer | 根据已训练好模型去预测 |
| lda-inference.c | lda\_inference | 迭代训练计算γ以及φ（被doc\_e\_step调用），并且在预测新文档主题分布时被infer函数调用 |
| compute\_likelihood | 根据公式(7‑27)计算变分下界 |

先来看看代码中使用到一些数据结构体，位于文件lda.h，如下所示：

//文件：lda.h

/\* 一篇文档的结构体,word以int指针存储 \*/

typedef struct

{

int\* words;

int\* counts;

int length;

int total;

} document;

/\* 语料集的结构体 \*/

typedef struct

{

document\* docs;

//单词总数V

int num\_terms;

//文章总数M

int num\_docs;

} corpus;

typedef struct

{

double alpha;

//log(i,w)

double\*\* log\_prob\_w;

//主题总数K

int num\_topics;

int num\_terms;

} lda\_model;

typedef struct

{

//topic🡪word的统计量，为了未来计算log(i,w)所使用的统计量

double\*\* class\_word;

//为了将log(i,w)归一化而设置

double\* class\_total;

double alpha\_suffstats;

int num\_docs;

} lda\_suffstats;

值得一提的是上述代码最后出现的这个lda\_suffstats结构体，在程序中是一个全局变量，其统计的信息跨越了不同的文档。且来看看它是如何初始化的，根据传入的命令行参数random/seeded/\*不同，有不同的初始化函数相对应。

//文件：lda-model.c

/\*\*

功能：传入seeded。初始化语料集的统计信息

\*\*/

void corpus\_initialize\_ss(lda\_suffstats\* ss, lda\_model\* model, corpus\* c)

{

int num\_topics = model->num\_topics;

int i, k, d, n;

document\* doc;

**for** (k = 0; k < num\_topics; k++) //遍历所有topic编号

{

**for** (i = 0; i < NUM\_INIT; i++)

{

d = floor(myrand() \* c->num\_docs);//随机挑选一篇文章

printf("initialized with document %d\n", d);

doc = &(c->docs[d]);

**for** (n = 0; n < doc->length; n++) //遍历每个单词

{

//为每个编号的topic的topic-word的统计量增加count

ss->class\_word[k][doc->words[n]] += doc->counts[n];

}

}

//这个for循环是为了平滑smoothed，为每个词的topic-word的统计量都初始化+1

**for** (n = 0; n < model->num\_terms; n++)

{

ss->class\_word[k][n] += 1.0;

ss->class\_total[k] = ss->class\_total[k] + ss->class\_word[k][n];

}

}

}

所谓的class\_word是topic🡪word的统计量，该变量在步骤中统计信息，是为了未来估计所使用的统计量[[40]](#footnote-40)；class\_total是为了未来计算时归一化的目的（代码位于lda-model.c文件下的步骤，即lda\_mle函数中。

而当命令行参数传入random时，使用了下面的代码进行随机初始化：

//文件：lda-model.c

/\*\*

功能：传入random。初始化语料集的统计信息

\*\*/

void random\_initialize\_ss(lda\_suffstats\* ss, lda\_model\* model)

{

int num\_topics = model->num\_topics;

int num\_terms = model->num\_terms;

int k, n;

**for** (k = 0; k < num\_topics; k++)

{

**for** (n = 0; n < num\_terms; n++)

{

ss->class\_word[k][n] += 1.0/num\_terms + myrand(); //myrand()是0~1的随机小数

ss->class\_total[k] += ss->class\_word[k][n];

}

}

}

### Variational EM代码剖析

执行EM算法，针对每篇文档，利用其单词以及初始化的和信息，更新模型的变分下界，直至收敛。

**1．E-step**

利用8.2节提出的工程优化分析公式( 8‑1 )、( 8‑2 )、( 8‑3 )先写出以下代码迭代求解和（这段代码位于lda-inference.c：

/\*

\* variational inference

\*/

**double** **lda\_inference**(document\* doc, lda\_model\* model, **double**\* var\_gamma, **double**\*\* phi)

{

**double** converged = **1**;

**double** phisum = **0**, likelihood = **0**;

**double** likelihood\_old = **0**, oldphi[model->num\_topics];

**int** k, n, var\_iter;

**double** digamma\_gam[model->num\_topics];

// compute posterior dirichlet

**for** (k = **0**; k < model->num\_topics; k++)

{

//初始化γ以及φ

var\_gamma[k] = model->alpha + (doc->total/((**double**) model->num\_topics));

digamma\_gam[k] = digamma(var\_gamma[k]);

**for** (n = **0**; n < doc->length; n++)

phi[n][k] = **1.0**/model->num\_topics;

}

var\_iter = **0**;

**while** ((converged > VAR\_CONVERGED) &&

((var\_iter < VAR\_MAX\_ITER) || (VAR\_MAX\_ITER == -**1**)))

{

var\_iter++;

**for** (n = **0**; n < doc->length; n++)

{

phisum = **0**;

**for** (k = **0**; k < model->num\_topics; k++)

{

oldphi[k] = phi[n][k];

//使用公式( 8‑2 )计算φ

phi[n][k] = digamma\_gam[k] + model->log\_prob\_w[k][doc->words[n]];

**if** (k > **0**)

phisum = log\_sum(phisum, phi[n][k]);

**else**

phisum = phi[n][k]; // note, phi is in log space

}

**for** (k = **0**; k < model->num\_topics; k++)

{

//使用公式( 8‑1 )均一化，然后log(φ) -> φ

phi[n][k] = exp(phi[n][k] - phisum);

//使用公式( 8‑3 )计算γ

var\_gamma[k] =

var\_gamma[k] + doc->counts[n]\*(phi[n][k] - oldphi[k]);

digamma\_gam[k] = digamma(var\_gamma[k]);

}

}

likelihood = compute\_likelihood(doc, model, phi, var\_gamma);

**assert**(!isnan(likelihood));

converged = (likelihood\_old - likelihood) / likelihood\_old;

likelihood\_old = likelihood;

}

**return**(likelihood);

}

**2．M-step**

后面的代码中，Blei使用了牛顿迭代法来更新，我们重新写出这些公式，即：



可以观察到迭代公式中需要使用到的因素有、，另外特别提醒的是在具体编程实践中，Blei对的处理是使用对称超参数，每个的值相等，所以只使用一个，而忽略的下标。因而第7章的求导公式( 7‑37 )就会有所变化，在正式展示代码前需要重新写出这些求导公式：

其中L只留下关于的项变为：



求出L关于的一阶导数：



( ‑ )

再求出L关于的二阶导数：



( ‑ )

实践中，Blei首先计算了的充分统计量alpha\_suffstats（位于lda-estimate.c文件下的doc\_e\_step函数中）：



( ‑ )

紧接着立即动手实现了三个小函数alhood、d\_alhood、d2\_alhood：

/\*

\* objective function and its derivatives

\*/

**double** **alhood**(**double** a, **double** ss, **int** D, **int** K)

{ **return**(D \* (lgamma(K \* a) - K \* lgamma(a)) + (a - **1**) \* ss); }

**double** **d\_alhood**(**double** a, **double** ss, **int** D, **int** K)

{ **return**(D \* (K \* digamma(K \* a) - K \* digamma(a)) + ss); }

**double** **d2\_alhood**(**double** a, **int** D, **int** K)

{ **return**(D \* (K \* K \* trigamma(K \* a) - K \* trigamma(a))); }

函数中的lgamma是C语言数学库中的log gamma，而digamma在是log gamma关于自变量的导数，trigamma是log gamma关于自变量的二阶导数，ss就是前文计算出的alpha\_suffstats。这些函数代码中的变量D就是前文推导中所用的整个语料库的文章数M，a就是我们要迭代寻找最优化的。将上述三个函数代码翻译成更清晰的数学形式如下：







之后就是牛顿迭代法循环迭代的主要逻辑函数了：

/\*

\* newton method

\* @param ss alpha\_suffstats

\* @param D num\_docs

\* @param K num\_topics

\*/

**double** **opt\_alpha**(**double** ss, **int** D, **int** K)

{

**double** a, log\_a, init\_a = **100**;

**double** f, df, d2f;

**int** iter = **0**;

log\_a = log(init\_a);

**do**

{

iter++;

a = exp(log\_a);

**if** (isnan(a))

{

init\_a = init\_a \* **10**;

printf("warning : alpha is nan; new init = %5.5f\n", init\_a);

a = init\_a;

log\_a = log(a);

}

f = alhood(a, ss, D, K);

df = d\_alhood(a, ss, D, K);

d2f = d2\_alhood(a, D, K);

**log\_a = log\_a - df/(d2f \* a + df);** //公式( 7‑38 )

printf("alpha maximization : %5.5f %5.5f\n", f, df);

}

//NEWTON\_THRESH = 1e-5, MAX\_ALPHA\_ITER = 1000

**while** ((fabs(df) > NEWTON\_THRESH) && (iter < MAX\_ALPHA\_ITER));

**return**(exp(log\_a));

}

轮到，在实际的代码中在迭代公式里被冠以class\_word的名字。并且使用了如下公式：



为了将归一化，代码中又同时计算了class\_total变量：



下面的代码生动地体现了这一点(代码中的doc->counts[n]是由于程序的输入语料文件格式是[term\_N]:[count]，重复的单词只输入count次数)。

**for** (n = 0; n < doc->length; n++)

{

**for** (k = 0; k < model->num\_topics; k++)

{

ss->class\_word[k][doc->words[n]] += doc->counts[n]\*phi[n][k];

ss->class\_total[k] += doc->counts[n]\*phi[n][k];

}

}

如果读者亲自去下载阅读Blei的源代码，会发现这段代码位于lda-estimate.c文件下的doc\_e\_step函数中，这就说明Blei的实际代码中将的迭代提前到了步骤的函数中（在前面的理论分析中本属于步骤计算），这也阐释了变分算法的编程实践中步骤和步骤有时没有明显的分界线。

由于E步骤中的计算公式( 8‑2 )同样用到了信息，但是是以的形式出现的，因此有了下面的归一化公式：



在程序代码中又被冠名为log\_prob\_w，换成代码里的变量名：



经过前面E步骤对于class\_word和class\_total统计量的准备，下面的代码就是极大似然估计（M步骤）了。

*/\**

*\* compute MLE lda model from sufficient statistics*

*\*/*

void lda\_mle(lda\_model\* model, lda\_suffstats\* ss, int estimate\_alpha)

{

int k; int w;

//compute log(i,vthen can use it to iteratively compute 

**for** (k = 0; k < model->num\_topics; k++)

{

**for** (w = 0; w < model->num\_terms; w++)

{

**if** (ss->class\_word[k][w] > 0)

{

model->log\_prob\_w[k][w] = log(ss->class\_word[k][w]) -log(ss->class\_total[k]);

}

**else**

model->log\_prob\_w[k][w] = -100;

}

}

//setting配置选项中alpha=estimate则estimate\_alpha = 1;否则estimate\_alpha = 0

**if** (estimate\_alpha == 1) //newton method iteratively compute 

{

model->alpha = opt\_alpha(ss->alpha\_suffstats,

ss->num\_docs,

model->num\_topics);

printf("new alpha = %5.5f\n", model->alpha);

}

}

最后展现给读者的是整个变分EM的框架代码：

void run\_em(char\* start, char\* directory, corpus\* corpus)

{

int d, n;

lda\_model \*model = NULL;

//var\_gamma = M x K

//phi = N(语料里最长的文章内的单词数) X K

double \*\*var\_gamma, \*\*phi;

//留给读者去阅读源码:phi难道在不同文档d共用一个?

// allocate variational parameters

var\_gamma = malloc(sizeof(double\*)\*(corpus->num\_docs));

**for** (d = 0; d < corpus->num\_docs; d++)

var\_gamma[d] = malloc(sizeof(double) \* NTOPICS);

int max\_length = max\_corpus\_length(corpus);

phi = malloc(sizeof(double\*)\*max\_length);

**for** (n = 0; n < max\_length; n++)

phi[n] = malloc(sizeof(double) \* NTOPICS);

// initialize model

char filename[100];

lda\_suffstats\* ss = NULL;

//start是命令行參數

**if** (strcmp(start, "seeded")==0)

{

model = new\_lda\_model(corpus->num\_terms, NTOPICS);

ss = new\_lda\_suffstats(model);

corpus\_initialize\_ss(ss, model, corpus);

//initial log\_prob\_w = log(ss->class\_word[k][w]) - log(ss->class\_total[k])

lda\_mle(model, ss, 0);

model->alpha = INITIAL\_ALPHA;

}

**else** if (strcmp(start, "random")==0)

{

model = new\_lda\_model(corpus->num\_terms, NTOPICS);

ss = new\_lda\_suffstats(model);

random\_initialize\_ss(ss, model);

//initial log\_prob\_w = log(ss->class\_word[k][w]) - log(ss->class\_total[k])

lda\_mle(model, ss, 0);

model->alpha = INITIAL\_ALPHA;

}

**else**

{

model = load\_lda\_model(start);

ss = new\_lda\_suffstats(model);

}

sprintf(filename,"%s/000",directory);

save\_lda\_model(model, filename);

// run expectation maximization

int i = 0;

double likelihood, likelihood\_old = 0, converged = 1;

sprintf(filename, "%s/likelihood.dat", directory);

FILE\* likelihood\_file = fopen(filename, "w");

**while** (((converged < 0) || (converged > EM\_CONVERGED) || (i <= 2)) && (i <= EM\_MAX\_ITER))

{

i++; printf("\*\*\*\* em iteration %d \*\*\*\*\n", i);

likelihood = 0;

zero\_initialize\_ss(ss, model);

// e-step

**for** (d = 0; d < corpus->num\_docs; d++)

{

**if** ((d % 1000) == 0) printf("document %d\n",d);

likelihood += doc\_e\_step(&(corpus->docs[d]),

var\_gamma[d],

phi,

model,

ss);

}

// m-step

lda\_mle(model, ss, ESTIMATE\_ALPHA);

// check for convergence

converged = (likelihood\_old - likelihood) / (likelihood\_old);

**if** (converged < 0) VAR\_MAX\_ITER = VAR\_MAX\_ITER \* 2;

likelihood\_old = likelihood;

// output model and likelihood

fprintf(likelihood\_file, "%10.10f\t%5.5e\n", likelihood, converged);

fflush(likelihood\_file);

**if** ((i % LAG) == 0)

{

sprintf(filename,"%s/%03d",directory, i);

save\_lda\_model(model, filename);

sprintf(filename,"%s/%03d.gamma",directory, i);

save\_gamma(filename, var\_gamma, corpus->num\_docs, model->num\_topics);

}

}

// output the final model

sprintf(filename,"%s/final",directory);

save\_lda\_model(model, filename);

sprintf(filename,"%s/final.gamma",directory);

save\_gamma(filename, var\_gamma, corpus->num\_docs, model->num\_topics);

// output the word assignments (for visualization)

sprintf(filename, "%s/word-assignments.dat", directory);

FILE\* w\_asgn\_file = fopen(filename, "w");

**for** (d = 0; d < corpus->num\_docs; d++)

{

**if** ((d % 100) == 0) printf("final e step document %d\n",d);

likelihood += lda\_inference(&(corpus->docs[d]), model, var\_gamma[d], phi);

write\_word\_assignment(w\_asgn\_file, &(corpus->docs[d]), phi, model);

}

fclose(w\_asgn\_file);

fclose(likelihood\_file);

}

读者应该注意到判断最终收敛使用了likehood来判断，这其实就是在使用下界公式(7‑27)，下面的代码严格执行并计算了这个公式。

*/\**

*\* compute likelihood bound*

*\**

*\*/*

double

compute\_likelihood(document\* doc, lda\_model\* model, double\*\* phi, double\* var\_gamma)

{

double likelihood = 0, digsum = 0, var\_gamma\_sum = 0, dig[model->num\_topics];

int k, n;

**for** (k = 0; k < model->num\_topics; k++)

{

dig[k] = digamma(var\_gamma[k]);

var\_gamma\_sum += var\_gamma[k];

}

digsum = digamma(var\_gamma\_sum);

likelihood =

lgamma(model->alpha \* model -> num\_topics)

- model -> num\_topics \* lgamma(model->alpha)

- (lgamma(var\_gamma\_sum));

**for** (k = 0; k < model->num\_topics; k++)

{

likelihood +=

(model->alpha - 1)\*(dig[k] - digsum) + lgamma(var\_gamma[k])

- (var\_gamma[k] - 1)\*(dig[k] - digsum);

**for** (n = 0; n < doc->length; n++)

{

**if** (phi[n][k] > 0)

{

likelihood += doc->counts[n]\*

(phi[n][k]\*((dig[k] - digsum) - log(phi[n][k])

+ model->log\_prob\_w[k][doc->words[n]]));

}

}

}

**return**(likelihood);

}

### 预测推断新文档

变分EM的推断预测功能在lda\_estimate.c代码文件下的infer函数中。可以从下面的这段代码中看到，在为新文档推断主题分布时，var\_gamma二维数组变量()为新分配内存构造出来，可谓白纸一张，而这恰是由于var\_gamma是doc-topic数组，仅与新文档有关，而与历史训练文档无关。而这段程序中调用了前文介绍过的lda\_inference函数，并传入了load\_model函数读出的历史训练得到的log\_prob\_w用以迭代计算（公式( 8‑2 )），皆因log\_prob\_w变量代表，是topic-word数组，会受到历史已训练的文档的影响。

void infer(char\* model\_root, char\* save, corpus\* corpus)

{

FILE\* fileptr;

char filename[100];

int i, d, n;

lda\_model \*model;

double \*\*var\_gamma, likelihood, \*\*phi;

document\* doc;

**model = load\_lda\_model(model\_root);**

//新分配内存，初始化var\_gamma

var\_gamma = malloc(sizeof(double\*)\*(corpus->num\_docs));

**for** (i = 0; i < corpus->num\_docs; i++)

var\_gamma[i] = malloc(sizeof(double)\*model->num\_topics);

sprintf(filename, "%s-lda-lhood.dat", save);

fileptr = fopen(filename, "w");

**for** (d = 0; d < corpus->num\_docs; d++)

{

**if** (((d % 100) == 0) && (d>0)) printf("document %d\n",d);

doc = &(corpus->docs[d]);

phi = (double\*\*) malloc(sizeof(double\*) \* doc->length);

**for** (n = 0; n < doc->length; n++)

phi[n] = (double\*) malloc(sizeof(double) \* model->num\_topics);

likelihood = **lda\_inference**(doc, model, var\_gamma[d], phi);

fprintf(fileptr, "%5.5f\n", likelihood);

}

fclose(fileptr);

sprintf(filename, "%s-gamma.dat", save);

save\_gamma(filename, var\_gamma, corpus->num\_docs, model->num\_topics);

}

### load model

如同8.3.5节所言，load\_model不读取gamma的信息，load\_model读取历史已训练好的模型中的以下信息：

1. 从<iteration>.beta文件中读取log\_prob\_w信息；
2. 从<iteration>.other文件中读取出主题个数K，语料词典单词总数V，alpha浮点数值

关于beta文件和other文件的详细信息请看8.3.7节。

### 运行效果与终止条件

如果按照说明readme文件运行blei版LDA程序，屏幕就会出现如下提示：

[root@localhost lda]# ./lda est 1.0 100 ../settings.txt lda-input.txt random ./

reading data from lda-input.txt

number of docs : 10

number of terms : 3197

\*\*\*\* em iteration 1 \*\*\*\*

document 0

alpha maximization : -12999.24770 -186.45954

alpha maximization : -1567.95766 -178.24854

alpha maximization : 2346.00064 -157.78173

alpha maximization : 3529.08773 -113.75411

alpha maximization : 3784.24192 -48.77433

alpha maximization : 3807.67408 -6.63685

alpha maximization : 3807.98722 -0.10092

alpha maximization : 3807.98728 -0.00002

alpha maximization : 3807.98728 -0.00000

new alpha = 2.74269

\*\*\*\* em iteration 2 \*\*\*\*

…

可见，由于进行了alpha迭代估计，所以将alpha的估计值也会打印出来。运行结束后会输出以下几个文件：

**1．<iteration>.beta**

这个文件会在每n轮迭代时输出一个beta值，比如000.beta、005.beta等。而最后一轮迭代后会输出final.beta。

这个文件的格式与Gibbs Sampling版本LDA的phi文件格式相同，每一行为代表一个topic，第k行的第v列=log\_prob\_w[k][v]，即。

**2.<iteration>.gamma**

这个文件会在每n轮迭代时输出一个值，比如000.gamma、005.gamma等。而最后一轮迭代后会输出final.gamma。

这个文件的格式与Gibbs Sampling版本LDA的theta文件格式相同，每一行为代表一个doc，第m行代表第m篇文档的主题分布，第m行的第k列=var\_gamma[m][k]，即 。

有人会发出疑问：Gibbs Sampling输出文件的theta和phi文件去哪了？其实在这个版本中使用了超参数log\_prob\_w和var\_gamma的估计值来代替theta和phi作为输出。

**3.<iteration>.other**

这个文件最为简单，就是alpha，主题个数K，词典单词数V的输出，典型的样例如下:

num\_topics 100

num\_terms 3197

alpha 0.0136578276

**4．word-assignments.dat**

这个文件只会在最后一轮迭代运行结束后输出，这个文件类似gibbs sampling版LDA的tassign文件，该文件的输出结果见图 8‑1。

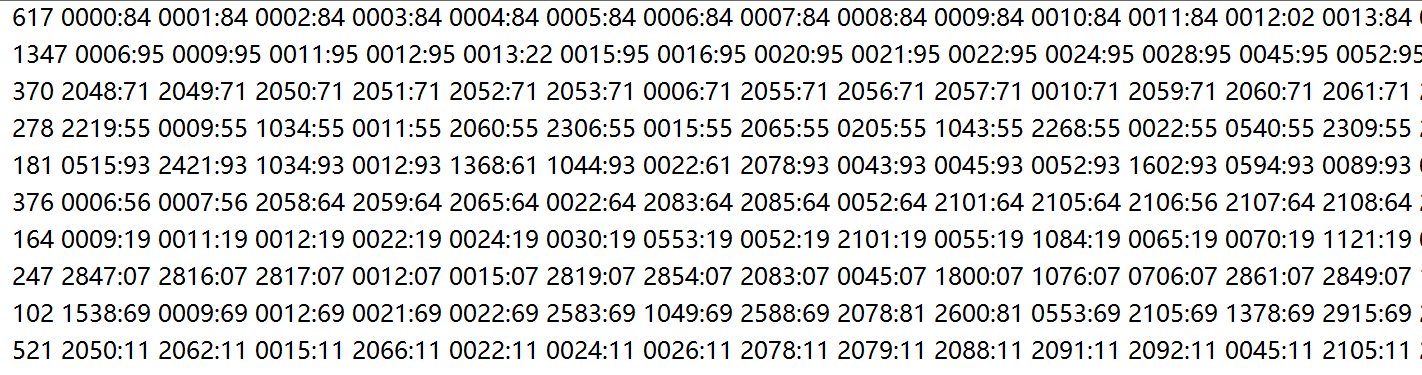


图 8‑1 word-assignments.dat输出样例

这个文件的每一行是一个doc文档的主题指定，每行第一个数字是该文档去重后的单词数，而后续的数字为：word\_id:phi[word\_id]里概率最大的topic编号，注意到这与Gibbs Sampling的主题z编号的生成方式有很大不同。（想想为什么？）从下面这段代码可以清楚地看到这一点。

/\*

\* writes the word assignments line for a document to a file

\*/

void write\_word\_assignment(FILE\* f, document\* doc, double\*\* phi, lda\_model\* model)

{

int n;

fprintf(f, "%03d", doc->length);

for (n = 0; n < doc->length; n++)

{

//argmax(int\* array, int K) return max element index from array[0,…,K-1]

fprintf(f, " %04d:%02d",doc->words[n], argmax(phi[n], model->num\_topics));

}

fprintf(f, "\n");

fflush(f);

}

**5.likelihood.dat**

每轮迭代时都会在EM步骤之后输出一个似然函数的下界值，输出两列，以tab分隔：第一列是当前迭代时，所有文档集总共的下界之和（将每篇文档调用lda-inference.c中的compute\_likelihood加和起来）；第二列是converged收敛值，converged = (likelihood\_old - likelihood) / (likelihood\_old)，这个公式计算得到的converged值最终用于终止迭代的条件（converged < 0或converged > settings.txt设置的EM\_CONVERGED）。

文件样例如下：

-53850.3207874086 inf

-48731.8030157054 9.50508e-02

-42982.5729925872 1.17977e-01

-41005.7009654659 4.59924e-02

-40497.9670195761 1.23820e-02

-40276.4152025643 5.47069e-03

-40159.6585856079 2.89888e-03

-40089.4897857436 1.74725e-03

**6．topics.py**

另外该程序包中还提供了一个topics.py的python脚本文件，用来输出人们可以直接理解的top n主题词（beta文件），类似Gibbs LDA++输出的twords文件，操作如下：

[root@localhost lda]# python topics.py 025.beta wordmap.txt 10

如此便输出了每个主题下概率最大的10个特征词：

topic 010

阎王关 254959

多布 147757

人間正道是滄桑 276624

霓虹灯 20385

查扣 110500

教导者 273769

大坪 86792

西西弗 102334

舞吧 89873

軟塌塌 241457

topic 011

金丝雀 93679

两个人 777

海鸭蛋 162481

后排 18353

下垂 20082

慢性 79589

許多 71571

骚客 16193

潮州府 82963

步兵团 33387

# 附录

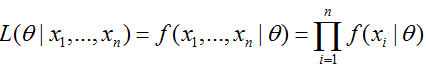
**极大似然估计(MLE)[13]**

最大似然估计提供了一种给定观察数据来评估模型参数的方法，即：“模型已定，参数未知”。简单而言，假设我们要统计全国人口的身高，首先假设这个身高服从正态分布，但是该分布的均值与方差未知。我们没有人力与物力去统计全国每个人的身高，但是可以通过采样，获取部分人的身高，然后通过最大似然估计来获取上述假设中的正态分布的均值与方差。

最大似然估计中采样需满足一个很重要的假设，就是所有的采样都是独立同分布(IID)的。下面我们具体描述一下最大似然估计：

首先，假设C:\Users\chen\AppData\Local\Temp\enhtmlclip\Image.png为独立同分布的采样，θ为模型参数,f为我们所使用的模型，遵循我们上述的独立同分布假设。参数为θ的模型f产生上述采样可表示为C:\Users\chen\AppData\Local\Temp\enhtmlclip\Image(1).png

回到上面的“模型已定，参数未知”的说法，此时，我们已知的为，未知为θ，故似然定义为:



在实际应用中常用的是两边取对数，得到公式如下：

C:\Users\chen\AppData\Local\Temp\enhtmlclip\Image(3).png

其中C:\Users\chen\AppData\Local\Temp\enhtmlclip\Image(4).png称为对数似然，而C:\Users\chen\AppData\Local\Temp\enhtmlclip\Image(5).png称为平均对数似然。而我们平时所称的最大似然为最大的对数平均似然，即：

C:\Users\chen\AppData\Local\Temp\enhtmlclip\Image(6).png

**例子1：**假如有一个罐子，里面有黑白两种颜色的球，数目多少不知，两种颜色的比例也不知。我 们想知道罐中白球和黑球的比例，但我们不能把罐中的球全部拿出来数。现在我们可以每次任意从已经摇匀的罐中拿一个球出来，记录球的颜色，然后把拿出来的球 再放回罐中。这个过程可以重复，我们可以用记录的球的颜色来估计罐中黑白球的比例。假如在前面的一百次重复记录中，有七十次是白球，请问罐中白球所占的比例最有可能是多少？很多人马上就有答案了：70%。而其后的理论支撑是什么呢？

我们假设罐中白球的比例是p，那么黑球的比例就是1-p。因为每抽一个球出来，在记录颜色之后，我们把抽出的球放回了罐中并摇匀，所以每次抽出来的球的颜 色服从同一独立分布。这里我们把一次抽出来球的颜色称为一次抽样。题目中在一百次抽样中，七十次是白球的概率是P(Data | M)，这里Data是所有的数据，M是所给出的模型，表示每次抽出来的球是白色的概率为p。如果第一抽样的结果记为x1，第二抽样的结果记为x2... 那么Data = (x1,x2,…,x100)。

P(Data | M)

= P(x1,x2,…,x100|M)

= P(x1|M)P(x2|M)…P(x100|M)

= p^70(1-p)^30.

那么p在取什么值的时候，P(Data |M)的值最大呢？将p^70(1-p)^30对p求导，并其等于零。

70p^69(1-p)^30-p^70\*30(1-p)^29=0。

解方程可以得到p=0.7。

在边界点p=0,1，P(Data|M)=0。所以当p=0.7时，P(Data|M)的值最大。这和我们常识中按抽样中的比例来计算的结果是一样的。

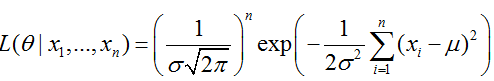
**例子2：**假如我们有一组连续变量的采样值（x1,x2,…,xn），我们知道这组数据服从正态分布，标准差已知。请问这个正态分布的期望值为多少时，产生这个已有数据的概率最大？

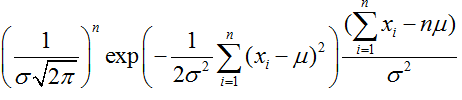
P(Data | M) = ?

根据公式

C:\Users\chen\AppData\Local\Temp\enhtmlclip\Image(7).png

可得:



对μ求导可得 ,则最大似然估计的结果为μ=(x1+x2+…+xn)/n

由上可知最大似然估计的一般求解过程：

（1）写出似然函数；

（2）对似然函数取对数，并整理；

（3）求导数 ；

（4）解似然方程

注意：最大似然估计只考虑某个模型能产生某个给定观察序列的概率。而未考虑该模型本身的概率。这点与贝叶斯估计区别。

# 参考文献

* 1. Philip Resnik and Eric Hardisty. Gibbs Sampling for the Uninitiated. CS-TR-4956 April 2010
  2. Gregor Heinrich. Parameter estimation for text analysis.
  3. 靳志辉(Rickjin). LDA数学八卦 2013.2.8
  4. 陈希孺. 概率论与数理统计. 中国科学技术大学出版社. 2009.2.1
  5. David M.Blei & Andrew Y.Ng. Latent Dirichlet Allocation
  6. 持之以恒. Reading Note : Parameter estimation for text analysis 暨LDA学习小结. <http://www.xperseverance.net/blogs/tag/lda/> . 2013.3.5
  7. Christopher M. Bishop. Pattern Recognition And Machine Learning. Springer. 2007.10.1. chapter 2 & chapter 11
  8. Mark Steyvers. Probabilistic Topic Models.
  9. David M.Blei & John D. Lafferty. Topic Models.
  10. Loulwah AlSumait and Daniel Barbara,James Gentle,Carlotta Domeniconi. Topic Signiﬁcance Ranking of LDA Generative Models.
  11. Zhuolin Qiu. Gibbs Collapsed Sampling for Latent Dirichlet Allocation on Spark. JMLR W&CP 36 : 17–28, 2014
  12. David Newman, Arthur Asuncion, Padhraic Smyth, Max Welling. Distributed Inference for Latent Dirichlet Allocation. {newman,asuncion,smyth,welling}@ics.uci.edu
  13. 最大似然估计 (Maximum likelihood estimation) <http://www.cnblogs.com/liliu/archive/2010/11/22/1883702.html>
  14. Xuan-Hieu Phan and Cam-Tu Nguyen. GibbsLDA++ Reference. <http://gibbslda.sourceforge.net/>
  15. Christophe Andrieu, Nando de Freitas, Arnaud Doucet, Michael I.Jordan. An Introduction to MCMC for Machine Learning. Kluwer Academic Publishers. 2001.9.10
  16. Allison J.B.Chaney and David M.Blei. Visualizing Topic Models. Association for the Advancement of Artiﬁcial Intelligence ([www.aaai.org](http://www.aaai.org)). 2012
  17. 赵学敏 王莉峰 王流斌 孙振龙 严浩 靳志辉 王益. Peacock：大规模主题模型及其在腾讯业务中的应用.

1. 陈希孺《概率论与数理统计》 p60, p67-例2.7 [↑](#footnote-ref-1)
2. http://en.wikipedia.org/wiki/Conjugate\_prior [↑](#footnote-ref-2)
3. 后验之意为执果索因，2.6节其实是贝叶斯估计，欲了解更多，参见陈希孺《概率论与数理统计》p155 [↑](#footnote-ref-3)
4. 采用贝叶斯法的出发点在于如果仅仅做了很少几次试验来估计参数（比如均值或概率p等），就会由于小样本数据造成估计不准。但利用之前已经拥有的经验（以前做过的试验数据）就可以令估计更为准确合理 [↑](#footnote-ref-4)
5. 图片来自于《LDA数学八卦》 [↑](#footnote-ref-5)
6. 图片来自于《LDA数学八卦》 [↑](#footnote-ref-6)
7. 这里足见是主题分布的超参数 [↑](#footnote-ref-7)
8. 这里足见是词分布的超参数 [↑](#footnote-ref-8)
9. LDA推断方法有MCMC和变分EM，这个版本讲的MH算法和Gibbs Sampling就是MCMC算法的一种 [↑](#footnote-ref-9)
10. 马尔可夫链是个很简单的概念，百度/google一搜一大堆，我在这里用通俗说法 [↑](#footnote-ref-10)
11. 关于markov chain收敛的更严格详细的证明，参见：<http://users.aims.ac.za/~ioana/notes-ch4-2x1.pdf> 4.4 [↑](#footnote-ref-11)
12. Sample的意思也就是一个随机数生成器：按照某种概率分布来随机生成一系列数字 [↑](#footnote-ref-12)
13. <https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm>:第(2)个条件的Ergodicity可以进一步用两个条件来表示：(1) be aperiodic—the system does not return to the same state at fixed intervals; and (2) be positive recurrent—the expected number of steps for returning to the same state is finite. [↑](#footnote-ref-13)
14. 一个单词可同时被多个主题生成，**就是从某个主题产生单词的概率，所以拆分成各主题的乘积 [↑](#footnote-ref-14)
15. 引自《parameter estimation for text analysis》 p21 [↑](#footnote-ref-15)
16. 这就是Collapsed Gibbs Sampling的Collapsed一词来历，这个integrating out出自《Gibbs Sampling for the Uninitiated》p15 这里可以积掉的深层原因是每个词上的topic采样都是独立的，而非《Uninitiated》一文里p18里的选定类标签L后一次性都将生成完一篇文章的所有word。（《Uninitiated中文版》的盘子图**一对多：一个源头箭头指向一篇文章的所有word） [↑](#footnote-ref-16)
17. 上面的和 “可以”被积分掉,原因在于为每个单词是被各topic独立生成，不互相影响和,因此可被积分掉. [↑](#footnote-ref-17)
18. 引自《parameter estimation for text analysis》p21 [↑](#footnote-ref-18)
19. further reading：

    普林斯顿大学材料：[Appendix D] Derivation of Gibbs sampling equations <https://lists.cs.princeton.edu/pipermail/topic-models/attachments/20110210/89b1646c/attachment-0001.pdf>

    HP实验室Note：Gibbs Sampling Derivation for LDA and TOT <http://home.in.tum.de/~xiaoh/pub/TOTGibbs.pdf> [↑](#footnote-ref-19)
20. 具体java版本代码请参考JGibbs LDA开源项目 [↑](#footnote-ref-20)
21. 等价于theta，等价于phi [↑](#footnote-ref-21)
22. 第二维每篇文档不一样（变长），第二维=per doc word count [↑](#footnote-ref-22)
23. See：https://lists.cs.princeton.edu/pipermail/topic-models/2012-June/001957.html [↑](#footnote-ref-23)
24. HDP等较为复杂的模型可以自动确定这个参数，但是模型复杂，计算复杂 [↑](#footnote-ref-24)
25. 关于逻辑回归的详细分析我另著一本小册子《LR无痛优化指引》 [↑](#footnote-ref-25)
26. 有时我也称之为上帝向量或完美向量 [↑](#footnote-ref-26)
27. 注意到公式5.1正确地反映了全局topic assignment z，因为更新后的nw如实地反映了各个机器上topic assignment z，由于各机器p启动Gibbs Sampling时都是从全局nw开始修改: [↑](#footnote-ref-27)
28. 详见《Collapsed Gibbs Sampling for Latent Dirichlet Allocation on Spark》，该方法思想类似腾讯peacock [↑](#footnote-ref-28)
29. http://fdcwqmst.blog.163.com/blog/static/16406145520109294346965/ [↑](#footnote-ref-29)
30. 因为，但信息论中由于考虑到要利用的数值都是二进制的，所以熵通常为以2为底。 [↑](#footnote-ref-30)
31. 从的函数图像也可看出这一点 [↑](#footnote-ref-31)
32. 最小作用量原理是一个极其重要的物理定律 [↑](#footnote-ref-32)
33. 相关讨论请看：<http://math.stackexchange.com/questions/279518/constrained-variational-problems-intuition> [↑](#footnote-ref-33)
34. 主要推导部分在第5节和附录(Appendix)部分 [↑](#footnote-ref-34)
35. 关于Possion分布为何物，请看本章后记 [↑](#footnote-ref-35)
36. 上标表示词表中的单词v，下标表示来源于文章中的哪个位置 [↑](#footnote-ref-36)
37. 与高斯混合模型类似 [↑](#footnote-ref-37)
38. 这里凹函数(concave)和凸函数(convex)与我国中学教材定义相反 [↑](#footnote-ref-38)
39. 狄拉克在布里斯托大学工程学院学习电机工程。尽管最喜欢的科目是数学，狄拉克后来声称这段工程教育对他影响深远。 [↑](#footnote-ref-39)
40. 正如8.2节工程优化所述，代码中均使用log对数形式保存值 [↑](#footnote-ref-40)