西 南 交 通 大 学

本科毕业设计（论文）

基于Spark平台的LDA算法的设计与实现

年 级： 2012 级

学 号： 20122714

姓 名： 杨宗霖

专 业： 物联网工程

指导教师： 李天瑞教授

二零一六年六月

院系 信息科学与技术学院 专 业 物联网工程

年级 2012级 姓 名 杨宗霖

题目 基于Spark平台的LDA算法的设计与实现

指导教师

评 语

指导教师 (签章)

评 阅 人

评 语

评 阅 人 (签章)

成 绩

答辩委员会主任 (签章)

年 月 日

**毕业设计（论文）任务书**

班 级 物联网1班 学生姓名 杨宗霖 学 号 20122714

发题日期： 2015年11月30日 完成日期： 2016年6月4日

题 目 基于Spark平台的LDA算法的设计与实现

1、本论文的目的、意义 随着信息技术及互联网的发展，人类所面对的网络信息爆炸式增长，因此如何将杂乱无章的海量文本信息数据快速转换成有用的信息和知识，一直是自然语言处理领域的研究热点。LDA是一种非监督机器学习算法，可以用来识别大规模文档集或语料库中潜藏的主题信息。然而在实际应用中，算法在计算大规模数据时速度较慢，原因是需要对大量数据反复迭代处理。根据LDA算法学习过程的可并行的特点，可以考虑将LDA算法部署到云平台上以提高算法的执行效率。

Spark是发源于美国加州大学伯克利分校AMPLab的大数据分析平台，它立足于内存计算，从多迭代批量处理出发，一经提出在学术界和业界都备受关注。与传统的基于磁盘操作的云平台如Hadoop相比，Spark能将任务之间的中间结果保存到内存之中，大大提高了效率。本文将Spark平台与LDA算法结合，通过实验测试LDA在内存并行框架中执行带来的效率提升，从而为多目标优化问题提供更加理想的解决方案。

2、学生应完成的任务

（1）查阅相关资料

（2）学习LDA算法

（3）学习Spark平台和Scala语言

（4）在Spark平台上实现LDA算法

（5）在集群上进行实验并对结果进行分析总结

（6）完成论文撰写

3、论文各部分内容及时间分配：（共 18 周）

第一部分 学习LDA算法 (3周)

第二部分 学习Spark系统与Scala语言 (3周)

第三部分 在Spark平台上实现LDA算法 (4周)

第四部分 分析总结集群实验结果 (3周)

第五部分 毕业设计论文撰写 (4周)

评阅及答辩 (1周)

备 注

指导教师： 2015年11月30日

审 批 人： 2015年 月 日

摘 要

本文针对海量文本处理的迫切需求，在大数据背景下对传统串行主题模型进行研究，并针对其中LDA（Latent Dirichlet Allocation）算法在处理大规模文档集时需要反复迭代的特点，将其与当下流行的基于内存分布式计算的大数据处理平台Spark结合，给出了一种基于Spark平台的并行LDA算法。

首先，对LDA文本主题生成模型进行了详细的介绍，在对比了LDA的几种推断方法的基础之上，选择了MAP EM算法作为推断模型参数的方法。由于用EM算法学习LDA模型存在一个潜在的图结构，给出了基于图的并行LDA算法，并将其实现在Spark GraphX之上，该并行方案实现了一种同时为所有文档中所有的词分配主题的方法，使得LDA算法效率大大提高。

其次，针对并行LDA算法基于图的实现方式，给出了一种基于图的LDA模型困惑度计算方法，它能够有效地利用原有图的词顶点快速地计算出测试集的困惑度；还针对输入数据集的格式，给出了文本预处理的并行实现方案。

最后，在实验环节中，通过选取经典文本数据集，设计了用于寻找最优主题数目和迭代次数的困惑度实验；并进行了并行LDA算法的主题聚类结果展示实验，说明了所实现的并行算法聚类结果良好；还设计了Speed-up、Size-up和Scale-up三个并行指标评测实验，通过实验验证了所实现的并行LDA算法具有比较良好的加速性能和可拓展性，同时还具有一定的承载能力。

关键词：大数据；Spark；LDA；EM；并行计算

Abstract

Aiming to the urgent need of massive text processing, this thesis discusses traditional serial topic models under the background of big data and combines one of its algorithms, LDA (Latent Dirichlet Allocation), with Spark, a current popular big data processing platform that is based on distributed memory computing, as this algorithm comes with repeated iteration when dealing with large-scale document collections. As a result, a parallel LDA algorithm based on Spark is presented.

Firstly, LDA text topic generation model is introduced in detail. With a comparison of several inference technologies of LDA, EM algorithm whose E-step is based on MAP to solve the posterior distribution of the hidden variable is chosen to infer model parameters. Due to the fact that utilizing EM algorithm to learn LDA model owns an underlying graph structure, a parallel implementation of LDA algorithm based on graph that is implemented with Spark GraphX is provided. This parallel scheme realizes a method assigning topic to all the words in all documents at the same time, thus the efficiency of LDA algorithm is greatly increased.

Next, a computing method of calculating the perplexity of LDA model based on graph is given. It can effectively use word vertices of the original graph to calculate the perplexity of test set in a short time. Also, a parallel implementation scheme of text preprocessing depending on the format of input text data set is provided.

Finally, in the experimental section, by selecting classic text data sets, perplexity experiments are designed to search the optimal number of topics and iterations. In addition, topic clustering experiments of the parallel LDA algorithm are carried. It shows that the clustering results are good. Moreover, three experiments for parallel performance evaluation including Speed-up, Size-up and Scale-up are designed. It is demonstrated that the implemented parallel LDA algorithm has good Speed-up performance, good scalability and certain carrying capacity.

**Keywords:** Big Data; Spark; LDA; EM; Parallel Computing

目 录

[第1章 绪 论 1](#_Toc453100125)

[1.1 论文背景与意义 1](#_Toc453100126)

[1.2 相关研究现状 2](#_Toc453100127)

[1.2.1 LDA的多种推断技术 2](#_Toc453100128)

[1.2.2 LDA的并行研究 2](#_Toc453100129)

[1.3 本文的研究工作 3](#_Toc453100130)

[1.4 本文章节安排 4](#_Toc453100131)

[1.5 本章小结 4](#_Toc453100132)

[第2章 相关知识 5](#_Toc453100133)

[2.1 贝叶斯推断 5](#_Toc453100134)

[2.2 Dirichlet分布 5](#_Toc453100135)

[2.3 变分法 6](#_Toc453100136)

[2.4 EM算法 6](#_Toc453100137)

[2.5 MCMC和Gibbs Sampling 7](#_Toc453100138)

[2.5.1 随机模拟方法 7](#_Toc453100139)

[2.5.2 马尔可夫链 7](#_Toc453100140)

[2.5.3 MCMC 8](#_Toc453100141)

[2.5.4 Gibbs Sampling 9](#_Toc453100142)

[2.6 文本分析中的参数估计 10](#_Toc453100143)

[2.6.1 最大似然估计MLE 10](#_Toc453100144)

[2.6.2 最大后验估计MAP 10](#_Toc453100145)

[2.6.3 贝叶斯估计 11](#_Toc453100146)

[2.7 本章小结 12](#_Toc453100147)

[第3章 LDA文本主题生成模型 13](#_Toc453100148)

[3.1 简单的文本潜在变量模型 13](#_Toc453100149)

[3.1.1 Unigram Model 13](#_Toc453100150)

[3.1.2 Mixture of unigrams 14](#_Toc453100151)

[3.1.3 主题模型PLSA 15](#_Toc453100152)

[3.2 LDA文本主题生成模型 16](#_Toc453100153)

[3.3 LDA推断模型参数的过程 18](#_Toc453100154)

[3.3.1 EM方法 20](#_Toc453100155)

[3.3.2 模拟法 22](#_Toc453100156)

[3.3.3 本节小结 23](#_Toc453100157)

[3.4 本章小结 24](#_Toc453100158)

[第4章 LDA算法的并行实现 25](#_Toc453100159)

[4.1 并行相关知识 25](#_Toc453100160)

[4.1.1 Spark简介 25](#_Toc453100161)

[4.1.2 GraphX并行图计算框架 28](#_Toc453100162)

[4.2 LDA并行方案 31](#_Toc453100163)

[4.2.1 潜在图结构 31](#_Toc453100164)

[4.2.2 基于GraphX的并行LDA实现 33](#_Toc453100165)

[4.3 基于图的困惑度并行计算方法 37](#_Toc453100166)

[4.4 文本预处理的并行实现 38](#_Toc453100167)

[4.5 并行LDA算法的不足之处 38](#_Toc453100168)

[4.6 本章小结 40](#_Toc453100169)

[第5章 实验及结果分析 41](#_Toc453100170)

[5.1 实验说明 41](#_Toc453100171)

[5.1.1 硬件配置 41](#_Toc453100172)

[5.1.2 软件配置 41](#_Toc453100173)

[5.1.3 数据来源 41](#_Toc453100174)

[5.2 困惑度实验 42](#_Toc453100175)

[5.2.1 困惑度 42](#_Toc453100176)

[5.2.2 确定迭代次数的困惑度实验 42](#_Toc453100177)

[5.2.3 确定主题数目的困惑度实验 44](#_Toc453100178)

[5.3 并行LDA主题聚类结果展示 46](#_Toc453100179)

[5.4 集群并行性能实验 47](#_Toc453100180)

[5.4.1 Speed-up实验 47](#_Toc453100181)

[5.4.2 Size-up实验 47](#_Toc453100182)

[5.4.3 Scale-up实验 48](#_Toc453100183)

[5.5 本章小结 49](#_Toc453100184)

[结 论 51](#_Toc453100185)

[致 谢 52](#_Toc453100186)

[参考文献 53](#_Toc453100187)

第1章 绪 论

## 1.1 论文背景与意义

随着信息技术与互联网的高速发展，人们所面对的网络信息爆炸式增长，面对杂乱无章的海量文本信息，如何快速、准确地从中提取有用的知识就成为当前自然语言处理领域的一大热点研究问题。LDA是一种无监督机器学习算法，可以用来从语料库中提取潜在主题信息，它根据文本的生成过程提出自己的概率图模型，然后基于观测到的词汇数据估计隐含变量。LDA主题模型的基本思想是文档可以由潜在主题随机混合而成，而每一个主题可以描述为词之上的分布，因此LDA主题模型也是一个多层概率模型。

作为一种文本主题生成模型，LDA能通过多轮迭代把特征向量集合按主题分类，可以用来识别大规模文档集或语料库中潜藏的主题信息，然而传统的LDA算法是串行地依次为语料库中的每一个词汇分配主题，且需要反复迭代处理，因而传统的LDA算法在实际应用中处理大规模文档集时速度较慢。

考虑到LDA算法学习模型的过程可并行的特点，将LDA算法并行化，并与当前流行的基于内存分布式计算的大数据处理平台Spark进行结合就变成一项很有意义的工作。

Spark[1]起源于加州大学伯克利分校的AMPLab实验室，它作为Hadoop MapReduce的替代方案，兼容HDFS、Hive和HBase等分布式存储层，可完美融入现有Hadoop生态系统，再加上它基于内存计算以及适合迭代的特点使得Spark一经提出就广受关注。目前Spark已成为Apache软件基金会的顶级项目，Spark开源社区也极其活跃，这对Spark的快速发展奠定了良好的基础。

本文主要是在海量文本数据背景下，针对LDA算法在Spark平台的并行化展开研究与实现。LDA模型由Blei等在2003首次提出[2]，在原始论文中，Blei等使用变分推断求解模型，并首先给出了一个基于变分法的C语言实现版本。2004年，Griffiths等提出了LDA参数估计的Gibbs采样方法[3]，这是一种精度更高，但是速度更慢的方法。尽管上述两种方法都可以估计出模型参数，但是由于它们都是串行算法，在处理大数据时就显得捉襟见肘。

综上，已有的串行LDA推断方法在面对大数据集时性能表现都不尽如人意，因此，对LDA算法并行化的研究就显得很有意义，基于LDA算法多轮迭代的特点将其并行地实现在Spark平台上将是本文研究的主要目标。

## 1.2 相关研究现状

### 1.2.1 LDA的多种推断技术

Blei等在2003年提出的潜在狄利克雷分布（Latent Dirichlet Allocation, LDA）[2]和Hofmann在1999年提出的概率潜在语义分析（Probabilistic Latent Semantic Analysis, PLSA）[4]已然成为众所周知的用于高维计数数据（例如，用“词袋”表示的文本数据或者用特征计数表示的图片）的潜在变量模型。

学术界已提出多种推断技术用于求解模型，其中，Griffiths等在2010年提出折叠吉布斯采样（Collapsed Gibbs Sampling, CGS）[5]，这是一种基于MCMC的随机模拟方法，它将模型参数通过积分消除；Blei等在LDA原始论文中使用变分贝叶斯推断（Variational Bayesian Inference, VB）来求解模型[2]，变分贝叶斯推断是一种变分推断方法，它基于Jesson不等式获得似然函数下界，通过调整参数来不断拟合；Teh等在2006年提出了一种变分贝叶斯的改进方法——折叠变分贝叶斯推断（Collapsed Variational Bayesian Inference, CVB）[6]，它通过积分消除模型参数，精度显著优于VB推断；此外，Hofmann在PLSA原始论文中使用最大似然估计（Maximum Likelihood Estimation, ML）来推断模型参数[4]；Chien等在2008年提出了一种ML估计的改进方法——最大后验估计（Maximum a Posteriori Estimation, MAP）[7]，它通过在估计参数的函数中加入先验分布，计算由贝叶斯公式得到的后验概率的最大值，使得参数更能反映真实情况。

### 1.2.2 LDA的并行研究

LDA推断技术主要有两种，分别是基于变分推断的变分法和基于Gibbs采样的模拟法，所以，目前对LDA并行化的研究也主要集中在变分贝叶斯方向和Gibbs采样方向。

在变分法方面，Nallapati等在2007年首次提出了LDA基于变分EM参数学习的并行化实现方法[8]，他们通过分析变分EM算法的迭代过程发现，EM算法的计算瓶颈主要体现在E步骤，并且E步骤计算各个文档的变分参数和是文档独立的，于是，他们提出一种静态负载均衡的方法，将语料库均匀地分成若干子集，然后分配到各个CPU核或者各个从节点进行并行计算，实现了对E步骤计算的并行加速。

然而，Nallapati等提出的基于等分数据集的并行变分EM算法的并行效果并不理想，主要原因在于LDA变分推断算法的复杂度与文档长度的平方成正比，数据集等分并不意味着各个从节点的计算量相同，这不能保证所有从节点同时结束计算，并且该方法聚合各个文档的充分统计量时依然是串行计算，这降低了算法的并行程度。

针对 Nallapati等提出的基于静态负载均衡的并行方案的不足之处，李文波等提出了一种基于动态负载均衡的并行算法[9]。算法的基本思想是，各个从节点通过竞争获取数据集的一个片段，处理完毕立即申请下一个片段，直到文档集处理完毕。该算法不仅提高了并行的效率，还对存储空间进行缩减，并优化访问。

在模拟法方面，David Newman等在2007年提出了首个基于Gibbs采样的LDA并行化实现方案[10]，称为近似分布式LDA（Approximate Distributed LDA, AD-LDA）。AD-LDA的主要思想是，将大的文档集分成许多子集，从而可以在子数据集上并行进行Gibbs采样，每采样一次就对中间结果进行一次合并与分发，以降低并行计算所带来的误差。

不过，由于每采样一次都需要将各个子数据集的参数进行合并计算再分发，这会带来比较大的空间和时间上的开销。针对已有算法的不足之处，邱卓林提出了一种可应用于并行框架上的无冲突数据分割方法[11]，并实现了新的分区函数，进一步减少迭代过程中数据交换所花费的时间。

此外，并行LDA算法也开始与各种并行框架进行结合。2010年，图并行框架GraphLab将Async-LDA加入到机器学习库中[12]；2009年，Wang采用MPI和MapReduce分别实现了AD-LDA，并命名为PLDA[13]；2013年，Wen等实现了基于GPU和Hadoop的CVB算法的并行化[14]。

## 1.3 本文的研究工作

本文是在大数据背景下基于Spark平台对LDA算法的并行化进行研究与实现，主要的研究工作可以分为如下四个方面：

首先，针对已有的工作，进行了一些综述介绍，包括LDA推导过程所需要的理论知识、经典的机器学习算法——EM算法、自然语言处理的一些方法、已有的几种简单文本潜在变量模型、LDA文本主题生成模型、LDA的几种推断方法以及Spark平台和GraphX并行图计算框架等并行相关的知识。

接着，对LDA的几种推断方法进行了一个比较，并选择了MAP EM算法作为推断模型参数的方法。由于用EM算法学习LDA模型存在一个潜在的图结构，于是将语料库转化为图，将对文档和词的操作转化为对顶点和边的操作，并将并行LDA算法实现在了Spark GraphX之上。

此外，针对并行LDA算法基于GraphX图的实现方式，给出了一种基于图的LDA模型困惑度计算方法，它有效地利用了原有图的词顶点，构造了一个基于测试集的新图，然后在图上使用消息聚合操作计算模型困惑度。

最后，选取了一些经典的文本数据集，通过并行文本预处理将其转化为词频矩阵，然后通过计算测试集的困惑度寻找最优的主题数目和迭代次数，接着展示了并行LDA算法的主题聚类结果，并在Spark云平台上对并行LDA算法进行了性能分析，完成了集群加速比等多个实验。

## 1.4 本文章节安排

本文按照主要的研究工作可以划分为如下五个章节：

第1章 绪论。介绍了论文的研究背景与意义、LDA相关的研究现状以及本文的研究工作和章节安排。

第2章 相关知识。介绍与LDA相关的理论知识，包括概率论和数理统计、EM机器学习算法、MCMC和Gibbs Sampling，以及文本分析中常用的参数估计方法。

第3章 LDA文本主题生成模型。首先介绍了几种简单的文本潜在变量模型，然后通过分析这些模型的不足之处引出了LDA主题模型，最后详细阐述了LDA推断模型参数的过程。

第4章 LDA算法的并行实现。首先介绍了Spark云平台和GraphX并行图计算框架，接着详细阐述了LDA算法基于Spark GraphX的并行化实现，然后给出了基于图的LDA模型困惑度的计算方法，并简要说明了文本预处理的并行实现，最后指出了本并行LDA算法的不足之处。

第5章 实验及结果分析。首先简单介绍了集群实验环境，然后进行了困惑度实验，接着展示了并行LDA算法的主题聚类结果，最后在集群完成并行性能相关的三个实验，并对实验结果进行分析。

## 1.5 本章小结

本章首先介绍了论文的研究背景与意义，阐述了大数据背景下并行LDA算法对处理海量文本数据所带来的积极作用，接着从LDA的推断技术和LDA的并行研究两个方面进行相关研究现状的介绍，最后概述了本文的主要研究工作和章节安排。

第2章 相关知识

本章主要介绍了一些和LDA相关的知识，包括一些概率论和数理统计的知识、EM机器学习算法和自然语言处理领域的一些方法。由于LDA推断过程涉及的知识点比较多，因此在介绍LDA之前回顾一下相关知识是有必要的。

## 2.1 贝叶斯推断

贝叶斯推断是一种应用于不确定条件下的统计推断方法，它的主要思想是通过观测到的数据来推断各种假设的概率，也就是计算特定猜测的后验概率。贝叶斯推断是贝叶斯定理的一种运用，它与其他推断方法有很大的不同之处，它先通过主观判断估计一个值，然后根据实际结果进行调整，因此，贝叶斯推断所需要的计算量比较大，在计算机得到大规模应用之前并没有得到人们的重视。随着计算机科学的迅猛发展以及人们对大数据处理的强烈需求，贝叶斯推断正日益显现出它巨大的威力。

贝叶斯推断基于贝叶斯定理，贝叶斯定理是全概率公式的逆应用，用于在某事件已经发生的条件下，计算各种原因使之发生的可能性大小，因此贝叶斯定理有时也被称为逆概率定理，它的计算公式为：

(2-1)

式中表示事件*B*发生的条件下事件*A*发生的概率。

贝叶斯定理也可以表示成先验概率和后验概率的关系，如式2-2所示。

(2-2)

如果用似然函数来刻画样本信息，那么式2-2就描述了后验分布和样本信息与先验分布的乘积之间成比例的关系，这也就是贝叶斯推断的含义，先按照先验分布估计一个值，然后加入实验结果，不断修正，使得推断的概率趋近于真实值。从数据知识的角度看，贝叶斯推断也可以用下面的过程来表示。

(2-3)

## 2.2 Dirichlet分布

Dirichlet公式的一般形式为：



(2-4)

式中和都是*K*维向量；是Dirichlet分布的参数；而可以看成是一个多项分布。所以Dirichlet分布也被叫做分布的分布，这是因为它可以按照概率去生成一个多项分布。

回顾Beta分布的公式定义，可以发现，Dirichlet分布是Beta分布在高纬度上的拓展，并且Beta分布和二项分布是共轭分布，所以可以得到，Dirichlet分布和多项分布是共轭关系。此处共轭的意思就是，当数据符合多项分布的时候，参数的先验分布和后验分布都能保持Dirichlet分布的形式，它们的关系为：

(2-5)

式中服从多项分布，整个式子也符合式2-3所示的贝叶斯推断过程。

Dirichlet分布和多项分布成共轭关系这个重要条件可以大大简化推导LDA的过程，这是因为，当已知参数的先验分布和数据的知识，根据式2-5，可以直接写出参数的后验分布。

## 2.3 变分法

变分法，即变分贝叶斯方法[15]，通常也被称为变分（贝叶斯）推断，它是一类用于贝叶斯估计和机器学习领域中近似计算复杂积分的技术。变分法主要应用于复杂的统计模型中，这样的模型一般包括三类变量，分别是：观测变量（data），未知参数（parameters）和潜变量（latent variables）。在贝叶斯推断中，未知参数和潜变量统称为不可观测变量（unobserved variables）。应用变分贝叶斯方法的主要目的是近似不可观测变量的后验概率，以便通过这些概率作出统计推断。

变分法的具体思想是对一个特定的模型，根据Jesson不等式给出观测变量的边缘似然函数的变分下界，然后通过调整变分参数，使得变分下界单调递增，不断拟合似然函数，当变分下界达到本地最大值，也就得到了变分参数的局部最优解，也即得到了不可观测变量的后验概率。

变分贝叶斯方法提供了一种局部最优，但具有确定解的近似方法，另外，变分贝叶斯方法也通过一组相互依赖的方程式进行不断的迭代来获得最优解。

## 2.4 EM算法

EM算法全称Expectation-maximization 算法[16]，即期望最大化算法，是机器学习领域的经典算法，它是一种从不完全数据或有数据丢失的数据集（存在隐含变量）中求解模型参数的最大似然估计方法。

它的基本思想是，利用Jesson不等式，获得模型似然函数L的下界，然后固定模型参数，计算隐含变量的后验概率，这会使得下界上升至与似然函数L在此点处相等，同时也会改变下界的形状；接着，固定隐含变量的后验概率，调整模型参数使下界达到最大值，然后以此时的参数值作为新的值；接着，继续固定，计算隐含变量的后验概率，然后这样不断迭代直到收敛到似然函数L的最大值处的。

从上述的EM迭代过程可以看出，下界是在不断地提高的，所以求得的极大似然估计也在单调递增，于是最终下界会达到最大似然估计的最大值，这也正是EM算法迭代收敛的依据。

一般的EM算法如算法1所示。

|  |
| --- |
| **算法1** EM算法 |
| 1. 初始化模型参数。  2. 迭代以下步骤直到收敛：  （1）E步骤：根据参数初始值或者上一次迭代的模型参数来计算隐含变量的后验分布，然后更新隐含变量的估计值。  （2）M步骤：将似然函数最大化以更新模型参数。 |

## 2.5 MCMC和Gibbs Sampling

### 2.5.1 随机模拟方法

随机模拟（或者统计模拟）方法也被称为蒙特卡罗方法（Monte Carlo Simulation），它能生成特定概率分布的样本，然后进行统计推断得到模型参数。对用确定性算法无法解决的问题，蒙特卡罗方法常常让人们看到希望。

要想采用随机模拟方法解决问题，就必须根据给定的概率分布，用计算机生成它的样本。一般而言，通过线性同余发生器可以相对容易地生成均分分布的样本，这是因为通过线性同余发生器生成的伪随机数在各项统计指标上都和均匀分布的理论计算结果非常接近，于是这样的伪随机序列就被当成真实的随机数使用。

此外，平时常见的概率分布，包括指数分布、Gamma分布、t分布、F分布、Beta分布和Dirichlet分布等，都可以基于的样本生成。然而，当概率分布的形式很复杂，或者是个高维分布的时候，基于的样本来得到的样本就会变得很困难。此时就需要一些更加复杂的随机模拟方法来生成样本，而MCMC（Markov Chain Monte Carlo）方法[17]和Gibbs Sampling方法就是两种比较常用的方法，由于这两种算法都是基于马氏链的平稳分布性质，所以需要先对马氏链有一个基本的了解。

### 2.5.2 马尔可夫链

马尔可夫链，即马氏链，它的数学定义比较简单，如下所示：

(2-6)

即马氏链上下一个状态转移的概率只由当前状态决定。

任意的初始状态分布在马尔可夫链上经过有限次状态转移之后都将会收敛于平稳分布，这被称为马氏链的收敛定理，是所有MCMC方法的理论基础。于是，从一个具体的初始状态开始，沿着马氏链按照概率转移矩阵做跳转，这样可以得到一个转移序列，假设，第n步时马氏链收敛，则由马氏链的收敛性质可知，都是平稳分布的样本。

### 2.5.3 MCMC

由于马氏链能收敛到平稳分布，那么构造一个转移矩阵的平稳分布为需要采样的概率分布的马氏链，然后从一个任意的初始状态开始，沿着马氏链转移，得到一个转移序列，如果，第n步时马氏链收敛，那么收敛之后得到的状态都将是的样本。

这样就用一个很巧妙的方法得到了给定概率分布的样本，这个方法最早由Metropolis在1953年提出，当时Metropolis为了研究粒子系统的平稳性质，从物理学中常见的玻尔兹曼分布采样问题中得到了灵感，首次提出了基于马氏链的蒙特卡罗方法，即Metropolis算法。Metropolis算法是首个普适的采样方法，并由此启发了一系列MCMC方法，被人们视为随机模拟技术腾飞的起点。

由上一节的定理可以发现，马氏链的收敛性质主要由转移矩阵决定，所以基于马氏链进行采样的关键问题是如何构造一个转移矩阵，使得它的平稳分布恰为需要采样的分布。我们主要使用细致平稳条件来得到转移矩阵，这可以构造符合要求的马氏链。细致平稳条件指，当非周期马氏链的转移矩阵*P*和分布满足

(2-7)

时，是马氏链的平稳分布。

假设现在有一个转移矩阵为*Q*的马氏链(*q*(*i*,*j*)表示从状态*i*转移到状态*j*的概率)，一般情况下，细致平稳条件是不会成立的，所以这里引入，并希望

(2-8)

成立，一个最简单的方法就是按照对称性，取

(2-9)

这样，式2-8就成立了，于是就将一个普通的马氏链改造为了满足细致平稳条件的马氏链，并且改造后的马氏链的平稳分布恰好就是需要采样的分布。

在改造马氏链的过程中引入了接受率，这样它的物理意义就可以理解为，在原来的马氏链上，当状态*i*以*q*(*i*,*j*)的概率转移到状态*j*时，马氏链有的概率接受这个转移，于是改造之后的马氏链的转移概率为。MCMC方法的转移过程如图2-1所示。

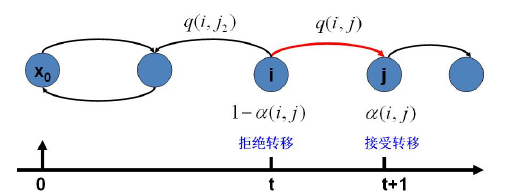


图2-1 MCMC方法转移过程[18]

把上面的推导过程整理一下就得到了如下所示的用于采样的MCMC算法。

|  |
| --- |
| **算法2** MCMC算法 |
| 1. 初始化马尔可夫链初始状态。 |
| 1. 对，循环下面的过程进行采样： |
| （1）第个时刻马氏链的状态为，采样。 |
| （2）从均分分布采样。 |
| （3）如果，则接受转移，即。 |
| （4）否则不接受转移，即。 |

### 2.5.4 Gibbs Sampling

对于高维的情况，由于MCMC方法的接受率通常小于1，这导致了MCMC方法的效率不够理想。于是寻找一个可以使得接受率的转移矩阵，这将有效地解决高维分布样本生成的问题。

在*n*维空间坐标轴中，对于*n*维概率分布，如果按照如下方式定义转移矩阵*Q*：

（1）当*n*维状态在马氏链上进行转移时，沿着这根坐标轴转移的概率为，即固定其他维度，只沿着某一个维度进行转移；

（2）其他不沿着单根坐标轴所进行的跳转，转移概率都为0。

会发现*n*维概率分布和转移矩阵*Q*满足细致平稳条件，于是就构造了一个平稳分布为的*n*维空间马氏链。而这个算法就被称为Gibbs Sampling算法，由物理学家Gibbs首次提出。此时由于接收率，*n*维空间马氏链上的状态每次转移都将成功，于是马氏链收敛的速度得到提高。

将上面的推导过程整理如下，就得到如算法3所示的*n*维Gibbs Sampling算法。

|  |
| --- |
| **算法3** Gibbs Sampling算法 |
| 1. 随机初始化*n*维状态。  2. 对，循环下面的过程进行采样： |
| （1）。  （2） |
| （3）。 |

## 2.6 文本分析中的参数估计

以LDA为代表的文本语言模型是当今自然语言处理研究的热点问题。这类语言模型一般都是基于文本的生成过程提出自己的概率图模型，然后利用观察到的语料数据估计模型参数，这是比较典型的贝叶斯推断过程。有了语言模型和相应的模型参数，就可以进行文本特征降维、文本主题分析等许多重要应用。本小节主要介绍应用于离散域的通用参数估计方法，由于本文主要讨论LDA模型，所以我们专门介绍了自然语言处理领域中常用的文本分析的三类参数估计方法，分别是最大似然估计MLE、最大后验估计MAP和贝叶斯估计[19]。

### 2.6.1 最大似然估计MLE

将前文介绍的贝叶斯公式（式2-1）重写为：

(2-10)

它可以表示成先验概率和后验概率的关系，如式2-2所示，而最大似然估计就是要寻找使得似然函数取最大值的参数值，似然函数如下所示。



(2-11)

为了简化运算，将似然函数取对数，将连乘转化为累加，就得到了对数似然函数。于是最大似然估计问题可以表述如下：



(2-12)

由于这是一个求关于[http://latex.codecogs.com/gif.latex?\theta](http://www.codecogs.com/eqnedit.php?latex=\theta)的函数的最大值的问题，所以通常的做法是对[http://latex.codecogs.com/gif.latex?\theta](http://www.codecogs.com/eqnedit.php?latex=\theta)求导，然后令导函数为0，解之得极值点，即为最大似然估计得到的模型参数值。

### 2.6.2 最大后验估计MAP

最大后验估计相比于最大似然估计，不同点在于估计参数[http://latex.codecogs.com/gif.latex?\theta](http://www.codecogs.com/eqnedit.php?latex=\theta)的函数中加入了一个参数的先验分布[http://latex.codecogs.com/gif.latex?p(\theta)](http://www.codecogs.com/eqnedit.php?latex=p(\theta))，于是最大后验估计就是要寻找使得由贝叶斯公式计算出的整个后验概率最大的参数值，如式2-13所示。



(2-13)

由于与参数[http://latex.codecogs.com/gif.latex?\theta](http://www.codecogs.com/eqnedit.php?latex=\theta)无关，因此最大后验估计等价于要寻找使得分子最大的参数值。与最大似然估计相比，这里得到的最大后验估计值多了一个先验分布的对数。在实际应用中，人们常用先验分布来描述已经知道或者普遍接受的规律。例如在扔硬币的试验中，抛出正面或者反面的概率应该服从一个概率分布，由于抛出正面的次数服从二项分布，而Beta分布又是二项分布的分布，所以它的先验分布很好的一个选择就是Beta分布，而由我们的认知，抛硬币正反面的概率一般都为0.5，所以先验分布Beta分布在0.5处取得最大值。先验分布的参数称为超参数(hyperparameter)，如式2-14所示。

(2-14)

求解最大后验估计的方法类似于MLE，将关于[http://latex.codecogs.com/gif.latex?\theta](http://www.codecogs.com/eqnedit.php?latex=\theta)的函数进行求导，然后令导函数等于0，解之即得MAP估计参数。

和最大似然估计的结果对比可以发现结果中多了先验分布的伪计数，这是因为先验分布在起作用，这会使得模型参数估计值更加趋近于真实值。

### 2.6.3 贝叶斯估计

贝叶斯估计是贝叶斯推断的应用，是MAP的进一步拓展，此时不直接估计参数值，而是让参数服从一个概率分布，即计算参数的后验概率。

于是现在需要计算由贝叶斯公式给出的后验概率，而由式2-3数据知识角度表述的贝叶斯推断过程以及2.2小节所描述的共轭分布的一些性质，可以以如下的方式巧妙地计算出参数的后验分布。

(2-15)

(2-16)

接着，求满足用Beta分布或者Dirichlet分布表示的后验分布的参数的期望，以此作为参数的估计值，计算表明，通过贝叶斯估计得到的参数的期望值比MLE和MAP方法得到的估计值都更为接近真实值。

## 2.7 本章小结

本章主要对LDA的相关知识进行了一个比较系统的介绍，从LDA推导过程需要用到的一些概率数理统计知识入手，介绍了基于贝叶斯定理的贝叶斯推断，Dirichlet分布和多项分布共轭的性质以及一种用于求解复杂积分的技术——变分法；接着还介绍了一种经典的用于从不完全数据集中求解模型参数的机器学习算法——EM算法；最后还介绍了自然语言处理领域的一些方法，包括MCMC和Gibb Sampling，以及文本分析中常用的三种参数估计方法，这些都为下一章具体介绍LDA模型和推断模型参数打好了基础。

第3章 LDA文本主题生成模型

本章首先介绍了几种更为简单的文本潜在变量模型—Unigram Model，Mixture of unigrams和PLSA主题模型；通过分析这些模型的不足之处，引出了LDA文本文档主题生成模型，并对其详细介绍；最后详细阐述了LDA模型推断模型参数的过程。

## 3.1 简单的文本潜在变量模型

### 3.1.1 Unigram Model

在一元模型中，每一篇文档的每一个词都由一个多项分布独立地生成，它使用下面的方法生成一个文档：

|  |
| --- |
| For each of the *N* words : |
| Choose a word . |

其中符号的含义如表3-1所示。

表3-1 Unigram模型符号含义

|  |  |
| --- | --- |
| **符号** | **含义** |
| ***N*** | 要生成的文档的词的个数 |
|  | 生成的第*n*个词*w* |
|  | 词*w*的分布 |

于是生成一篇文档的概率为：

(3-1)

式中可以通过语料进行统计得到，比如给定一本书，统计各个词在书中出现的概率。

这种方法通过训练语料获得一个词的概率分布函数，然后根据这个概率分布函数每次生成一个词，使用这个方法*M*次生成*M*个文档。其图模型如图3-1所示。



图3-1 Unigram图模型

### 3.1.2 Mixture of unigrams

Unigram模型的缺点就是生成的文本没有主题，过于简单，mixture of unigrams模型对其进行了改进，该模型使用下面的方法生成一个文档：

|  |
| --- |
| 1. Choose a topic . |
| 1. For each of the *N* words :   Choose a word . |

其中符号的含义如表3-2所示。

表3-2 Mixture of unigrams模型符号含义

|  |  |
| --- | --- |
| **符号** | **含义** |
| ***z*** | 所生成文档的主题 |
|  | 主题的概率分布 |
|  | 给定主题时词的概率分布 |
| ***N*** | 同表3-1 |
|  | 同表3-1 |

其中可以看成一个*K*×*V*的矩阵，*K*为主题的个数，*V*为词汇表词的个数，每行表示这个主题对应的词的概率分布，即主题*z*所包含的各个词的概率，通过这个概率分布按一定概率生成每个词。

在这种混合模型中，生成文档的过程为：首先选定一个主题*z*，然后根据条件多项概率，独立地生成*N*个词。生成一篇文档的概率如式3-2所示。

(3-2)

使用*M*次这个方法生成*M*份不同的文档。其图模型如图3-2所示。



图3-2 Mixture of unigrams图模型

从图3-2可以看出，*z*在*w*所在的长方形外面，表示*z*生成一份*N*个词的文档时主题*z*只生成一次，即只允许一个文档只有一个主题，这不太符合常规情况，通常一个文档可能包含多个主题。

### 3.1.3 主题模型PLSA

上述两个模型都是很简单的模型，模型中的假设都过于简单，这与人类写文章产生每一个词的过程差距比较大，于是探讨更好的模型。

通过观察人类写文章的过程，可以得到，一篇文章通常是由多个主题构成，而每一个主题又可以用与该主题相关的频率最高的一些词来描述。

以上这种直观的想法首先由Hofman于1999年提出[3]，Hoffman认为一篇文档（Document）可以由多个主题（Topic）混合而成，而每个Topic都是词汇上的概率分布，文章中的每个词都是由一个固定的Topic生成的。

PLSA使用下面的方法生成一篇文档：

|  |
| --- |
| 1. Choose a document .  2. For each of the *N* words :  （1）Choose a topic . |
| （2）Choose a word . |

其中符号的含义如表3-3所示。

表3-3 PLSA模型符号含义

|  |  |
| --- | --- |
| **符号** | **含义** |
|  | 要生成的文档 |
|  | 文档的先验概率 |
|  | 第*n*个词的所属主题 |
|  | 给定文档时主题的概率分布 |
| ***N*、、** | 同表3-2 |

在PLSA模型中，生成文档的过程为：首先选定一个文档，然后在该文档中遍历每一个词块（token），按照给定文档时的主题概率分布生成一个主题，再按给定主题时词的概率分布生成词，这样重复*N*次就得到了一篇文档。

第*m*篇文档中的每个词的生成概率如式3-3所示。

(3-3)

使用这个方法*M*次，可以得到包含*M*个文档的语料库，其图模型如图3-3所示，其中深色圆圈表示观测变量，浅色圆圈表示隐含变量。



图3-3 PLSA图模型

由于文档之间是独立可交换的，文档中的词也是独立可交换的，于是可以很容易求出一篇文档或者整个语料库的生成概率。求解PLSA模型参数，可以使用EM算法求得局部最优解。

## 3.2 LDA文本主题生成模型

按照贝叶斯学派的观点，任何未知参数都应该是随机变量，对于上述的PLSA模型，文档-主题分布和主题-词分布都应该服从一个先验分布。于是Blei在模型参数前加上先验分布将PLSA过程改造为一个贝叶斯过程[2]。由于和都是多项分布，所以可以选择Dirichlet分布作为它们的共轭先验分布，于是就得到了LDA（Latent Dirichlet Allocation）模型。

LDA是一个语料库的概率生成模型，它的基本观点是，文档可以由潜在主题随机混合而成，而每一个主题可以描述为词之上的分布。

本文用一些通用的符号来开始描述LDA模型，LDA采用标准的词袋方法来表示语料库，语料库*D*中的每一篇文档被表示为*W*项的词频计数向量，其中，*W*指的是词汇表中词的数目。语料库中的每一篇文档*j*由*K*个主题混合建模而成，每一个主题*k*是一个词汇表之上的分布。每一个主题-词分布是从参数为*η*的狄利克雷分布采样得到，而每一个文档-主题分布是从参数为*α*狄利克雷分布采样得到。对于语料库中的每一个词块*i*，它的主题分配是从采样的到，指定的词是从采样得到。

LDA模型可以使生成的文档包含多个主题，该模型使用下面的方法生成语料库*D*中的每一篇文档***w***：

|  |
| --- |
| 1. Choose Poisson(*ζ*).  2. Choose .  3. Choose *.*  4. For each of the *N* tokens :  （1）Choose a topic .  （2）Choose a word . |

其中符号的含义如表3-4所示。

表3-4 LDA模型符号含义

|  |  |
| --- | --- |
| **符号** | **含义** |
| **、** | Dirichlet分布的参数 |
| ***N*** | 所生成文档词的数目 |
| **、** | 指定文档的主题分布 |
| **、** | 指定主题的词汇分布 |
|  | 第个词块 |
|  | 第个词块所属的主题 |
|  | 第个词块所生成的词 |

使用这个方法*M*次，可以得到*M*篇文档，LDA的概率图模型如图3-4所示。



图3-4 LDA概率图模型

其中，深色部分为观测变量；其余浅色部分的变量均为不可观测变量，和是未知参数；是潜变量。

LDA的概率图模型可以进行物理过程分解，这将得到两个Dirichlet-Multinomial共轭结构，分别为：

（1） ：这表示在生成第*j*篇文档的时候，先从参数为的Dirichlet分布按照一定概率得到一个文档-词分布，然后再按一定概率生成第*i*个词的所属主题。

（2） ：首先从参数为的Dirichlet分布按照一定概率生成*K*个主题-词分布，接着在生成第*j*篇文档的第*i*个词的时候，从中选择主题编号为的主题-词分布生成词。

由于LDA生成模型是词袋模型，因而文档与文档之间是独立的，词与词之间也是独立的，于是有些物理过程可以调换顺序，因此在LDA模型中，M篇文档对应M个独立的如（1）所示的共轭结构，K个主题对应K个独立的如（2）所示的共轭结构。

综上，LDA（隐含狄利克雷分布）文本主题生成模型，是一个三层概率模型，它包含文档、主题和词三层结构。文档到主题服从多项分布，主题到词服从多项分布。所谓生成模型，就是说，我们认为一篇文章的每个词都是通过“以一定概率选择了某个主题，并从这个主题中以一定概率选择某个词”这样一个过程得到。

LDA采用了词袋（bag of words）的方法，这种方法将每一篇文档视为一个词频向量，从而将文本信息转化为易于建模的数字信息。但是词袋方法没有考虑词与词之间的顺序，这简化了问题的复杂性，同时也为模型的改进提供了契机。

## 3.3 LDA推断模型参数的过程

LDA是一种非监督机器学习技术，是一种主题聚类模型，它能通过多轮迭代把特征向量集合按主题分类，可以用来识别大规模文档集（document collection）或语料库（corpus）中潜藏的主题信息。目前，广泛运用在文本主题聚类中，是当前主题聚类领域最火、最有力的模型之一。LDA接受文本文档作为输入，输出推断出的主题以及输入语料库中每篇文档的主题分布。

LDA的模型参数主要有两个，分别为文档-主题分布和主题-词分布，联系3.2节所介绍的参数含义，可以发现估计出模型参数，也就完成了用LDA模型推断文档主题的任务。于是，进行LDA文本主题生成的主要工作就转化为估计模型参数和。

以LDA为代表的文本语言模型是当今统计自然语言处理研究的热点问题。这类语言模型一般都是对文本的生成过程提出自己的概率图模型，然后利用观察到的语料数据对模型参数做估计。

下面列出文档-主题分布，如下所示：

(3-4)

式中表示文档*j*生成主题*k*的概率，它的计算公式为：

(3-5)

式中，表示文档*j*中属于主题*k*的词的数目；表示文档*j*中所有词的数目。

当确定了每个词的所属主题，也即确定了潜变量（latent variable）的取值，就得到了一个文档的主题分布。

类似地，列出主题-词分布，如下所示：

(3-6)

式中表示主题*k*生成词*w*的概率，它的计算公式为：

(3-7)

式中，表示词项*w*属于主题*k*的词的数目；表示所有属于主题*k*的词的总数。

同样地，当确定了每个词的所属主题，就得到了主题-词分布，也就得到了主题。

回顾LDA生成模型可以知道，文档中的每一个词只由一个主题生成，并且，模型中唯一的潜变量就是词所属的主题，当潜变量确定，并且模型类型已知，就可以用最大似然方法估计模型参数，于是用LDA模型生成主题的工作进一步转化为确定词所属主题。通过观测变量去推测生成过程是一个逆概率过程，这是一个典型的贝叶斯问题。

于是，为了使用LDA必须解决的关键推理问题就变为计算给定文档的潜变量的后验分布，如下所示：

(3-8)

然而，不幸的是，这个后验分布难以被精确计算[2]，于是可以采用近似估计的方法。

根据LDA生成文档的过程，可以发现一个词属于某个主题的概率正比于LDA以某个主题生成它的概率，具体表述如式3-9所示。

(3-9)

于是，可以得到一个计算隐含变量后验概率的式子，利用当前的模型参数和可以计算文档*j*中词*w*属于各个主题*k*的概率，然后采样这个分布，来为词重新分配主题，进而更新模型参数，不断迭代，直到收敛。

这里也体现了聚类的思想，先随机的为词分配主题，然后将词按照所属主题聚成*K*个类，然后估计各个分布的参数，接着按照各个分布的参数，重新将词进行聚类，这样不断迭代直到各个分布基本稳定，收敛时也就确定了词所属主题，进而得到了模型参数。

这里总结一下LDA的推断过程，见算法4。

|  |
| --- |
| **算法4** LDA模型参数的推断 |
| 1. 为每一篇文档的每一个词随机分配一个主题，这样模型参数也得到初始化。 |
| 1. 利用模型参数和计算，然后采样这个分布，为词更新所属主题。 |
| 1. 更新模型参数，不断迭代，直到收敛，就确定了隐含变量，并输出模型参数。 |

学术界主要的推断方法也基本遵循了这个推断过程，包括EM方法和模拟法，它们的主要区别可以概述如下：

（1）为词分配主题时所依据的采样公式不同：EM方法依据的是MAP或变分贝叶斯推断方法得到的词主题更新式；而模拟法依据的是Gibbs采样公式。

（2）更新词主题和更新模型参数的顺序不同：EM方法是在E步骤中，为所有词依据更新式重新分配主题，然后在M步骤中更新模型参数；而模拟法是每更新一个词的主题就更新模型参数。

### 3.3.1 EM方法

已知模型类型，从观测数据中估计模型参数，首先可以考虑最大似然估计，但是由于有隐含变量的存在，所以需要采用EM算法。EM算法是一种从不完全数据集中求解模型参数的最大似然方法，它分为E步骤和M步骤。如下所示：

（1）E-step：计算隐含变量的后验概率。

（2）M-step：极大似然方法估计模型参数。

如本节之前所描述，隐含变量的后验分布难以直接精确计算，于是可以采用近似计算的方法。E-step计算隐含变量的后验概率又有两种近似方法，分别是变分推断和MAP。为了叙述方便，给出如下两个定义：

（1）变分EM：E步骤用变分推断来求解隐含变量的后验分布。

（2）MAP EM：E步骤用MAP来求解隐含变量的后验分布。

#### 3.3.1.1 变分推断计算隐含变量后验分布

本文结合LDA原始论文来说明变分推断，原始论文中的生成概率图模型如图3-5所示。



图3-5 原始论文中的LDA概率图模型

变分推断，也叫变分贝叶斯推断，是一种近似计算复杂积分的方法，它利用Jensen不等式，获得观测变量的边缘似然函数的下界，（在LDA中，是指文档对数似然函数的下界），然后调整下界，拟合似然函数。为了让下界变得易于处理，原文在原来的图模型上做了一些简单的修改，注意到，导致隐含变量后验分布难以计算的原因是*θ*和*β*之间存在耦合，而导致耦合出现的原因又是因为*θ*，*z*，*w*之间存在边，于是，抛弃这些边和*w*顶点，并赋予生成的简化图模型两个自由变分参数，和。

消除耦合，引入变分参数之后得到图模型如图3-6所示。



图3-6 应用变分分布来近似LDA后验分布的图模型表示

其中是狄利克雷参数；是多项参数；表示第*n*个词属于不同主题的概率，也就是隐含变量。

通过Jensen不等式获得的文档对数似然函数下界可以表示为，似然函数和下界的关系如下所示：

(3-10)

式中，是文档对数似然函数。

用和去最大化下界，可以得到迭代更新式，如下所示：

(3-11)

(3-12)

收敛时，可以得到和的局部最优值，此时，下界达到本地最大值，同时，也求解出了隐含变量的后验分布。

E-step中使用变分推断来计算隐含变量后验分布的EM算法也称为变分EM算法，这在LDA原始论文中得到了应用。

变分EM算法如算法5所示。

|  |
| --- |
| **算法5** 变分EM算法 |
| 1. 初始化：随机初始化模型参数。 |
| 1. 迭代直到收敛： |
| （1）E-step：使用变分推断对每个文档求和，使文档对数似然函数的下界达到本地最大值，然后进行累加，得到整个语料库的总的变分下界。 |
| （2）M-step：用最大化总的变分下界，得到新的。 |

变分EM的整体过程可以看成是在语料库似然函数*L*上的坐标上升（coordinate ascent），最终会到达似然估计的最大值，这也是变分EM的收敛依据。

#### 3.3.1.2 MAP计算隐含变量后验分布

MAP，最大后验估计，寻找使由贝叶斯公式计算出的整个后验概率达到最大的参数值，它的推导过程类似于MLE，不同点在于估计参数的函数中允许加入一个先验*p*(参数)。这就使得参数的最大后验估计值与最大似然估计值相比，多了伪计数。

用最大似然估计得到的文档*j*中的词*w*属于主题*k*的概率，也即更新词主题的更新式为：

(3-13)

式中，的含义与前述相同。

而MAP相比于MLE的不同之处在于，它为模型参数增加了狄利克雷先验，即、，于是参数的MAP估计值比ML估计值多了伪计数和。

于是可以推导出在MAP方法下的更新式为式3-14。

(3-14)

Asuncion等在技术报告中[20]指出在变分贝叶斯推断方法下的更新式为：

(3-15)

这也揭示了VB更新式和MAP更新式之间的关系。

### 3.3.2 模拟法

如本节之前所述，求解模型参数，本质上，是要确定词属于哪个主题，而词属于哪个主题，是采样隐含变量后验分布得到的，但是很不幸，这个分布不好计算。于是，寻找一种方法，可以不计算分布，但是依然可以生成它的样本，也就是不求解后验概率，却可以用它来给词分配主题。考虑到Gibbs Sampling，可以生成高维分布的样本，而文本也是高维的，词数就是它的维度数。

所以可以用Gibbs采样随机模拟隐含变量的后验分布的样本，这相当于采样隐含变量的后验分布，为每一篇文档的每一个词分配主题，确定了隐含变量，也就确定了模型参数。

Gibbs Sampling，马尔可夫链蒙特卡罗（MCMC）随机模拟法的一种，使用Gibbs采样分配主题的基本思想是：每一轮迭代中，都为每一篇文档的每一个词，按照Gibbs Sampling采样公式更新主题，直到模型收敛，或者达到最大迭代次数。

Gibbs采样方法推断模型参数的伪代码见算法6。

|  |
| --- |
| **算法6** Gibbs Sampling推断LDA模型参数 |
| // 初始化阶段   1. 将全局计数变量置为零； 2. for all words in all documents do 3. 为词随机分配主题； 4. 按照词所属主题，更新相应的全局计数变量； 5. end for   // Gibbs 采样burn-in阶段   1. while 没有达到最大迭代次数 do 2. for all words in all documents do 3. 减全局计数变量；// 消除当前词所属主题的影响 4. 依据Gibbs采样公式进行采样； 5. 增全局计数；// 按照词新分配的主题，更新相应的全局计数变量 6. end for 7. end while   // 迭代结束，生成模型参数   1. 根据参数估计式计算模型参数和； |

它的收敛依据是任意初始状态在马尔科夫链上经过有限次状态转移之后都将收敛于平稳分布，它构造转移矩阵的依据是马氏链的细致平稳条件。

Asuncion等它们的技术报告[20]中指出，MCMC的推断技术也可以用于LDA，依据折叠Gibbs采样公式（折叠指的是和.被积分消除），主题分配的采样按照如下的方式依次进行：



(3-16)

式中， 表示去除词块*ij*的计数。

### 3.3.3 本节小结

MAP、VB和CGS方法得到的词主题更新式（即词属于各个主题的概率，也就是潜变量的后验分布），虽然推理的方法不同，但是殊途同归，得到的数学公式却形式一致，都形如：

(3-17)

这是因为它们的物理意义相同，都是在*K*条生成词的路径上进行采样，而数学只是一种揭示物理世界客观规律的手段。在*K*条路径上采样生成词的过程如图3-7所示。



图3-7

## 3.4 本章小结

本章首先从几个简单的文本潜在变量模型开始介绍，然后通过分析这些已有简单

模型的不足之处引出了LDA文本主题生成模型，接着对LDA模型进行了详细的介绍，最后详细阐述了LDA的两种模型参数推导方法——EM方法和Gibbs采样法，并对不同方法推导出的数学公式形式一致的现象给出了解释。

第4章 LDA算法的并行实现

通过前文对LDA模型参数推断过程的介绍，可以发现LDA模型通过为词分配主题来推断文档主题。由于串行LDA算法是依次为每一篇文档的每一个词分配主题，如果能将其改造成同时为整个语料库中的每个词分配主题，就实现了LDA的并行化。观察LDA的两种推断过程——EM方法和模拟法，可以发现模拟法在更新词主题之后立即更新模型参数，并且它的采样公式需要用到其它词的所属主题，这导致并行模拟法将会造成不可避免的精度损失，而EM方法就全然没有这样的问题，于是，我们选择EM方法来做LDA的并行化。

对比变分EM和MAP EM，可以发现MAP EM的求解过程更为简单，在工程上更易于实现，于是我们选择MAP EM算法来求解模型。由于用EM算法学习LDA模型存在一个潜在的图结构，用Spark GraphX并行LDA就变成一个很自然的选择。

本章首先进行一些并行相关的知识介绍，包括Spark云平台和GraphX并行图计算框架；接着，给出了LDA基于Spark GraphX的并行实现方案；之后，给出了基于图的困惑度计算方法和文本预处理的并行实现；最后，指出了本算法的不足之处。

## 4.1 并行相关知识

### 4.1.1 Spark简介

本小节主要就当前流行的内存并行计算框架Spark进行简要介绍。

Spark是近几年兴起的一种基于内存计算的大数据处理平台，它站在巨人的肩膀上，借鉴了许多前人工作的经验，通过RDD实现数据的分布式内存存储，通过DAG执行模式实现应用的并行执行。Spark针对Hadoop MadReduce的已有缺点，将中间结果缓存在内存中，极大提高了迭代计算的性能，并通过提供大量的算子，解决了MapReduce操作单一的弊端。

Spark作为新一代的大数据计算框架，势必会取代Hadoop MapReduce，而Hadoop YARN则将朝着通用集群操作系统的方向发展。此外，Spark用HDFS作为保存数据的持久化层，这也意味者，Hadoop的HDFS已成为分布式领域事实上的数据存储标准。

#### 4.1.1.1 Spark生态系统

Spark生态系统旨在提供一站式解决方案（“One stack to rule them all”），它的全称为伯克利数据分析栈（BDAS）。BDAS以Spark作为核心组件，包括了支持结构化数据SQL查询与分析的查询引擎Spark SQL，提供了Hive命令接口的Shark，提供了用户友好的机器学习功能的MLbase和底层的分布式机器学习库Mlib，基于BSP模型的并行图计算框架GraphX，流计算框架Streaming等。这些子项目在Spark上层提供了更高层、更丰富的计算框架。BDAS项目结构图如图4-1所示。

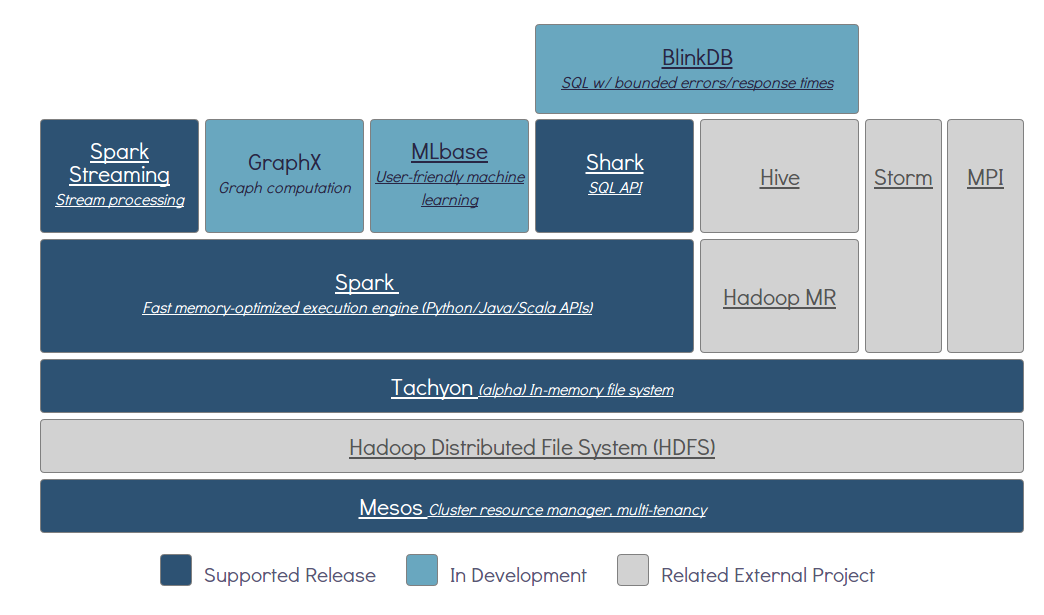


图4-1 BDAS项目结构图[1]

#### 4.1.1.2 Spark核心概念

1. Spark架构

Spark采用Master-Slave模型，主节点运行Master进程，从节点运行Worker进程。Master负责整个集群的正常运行，Worker作为计算节点。Spark架构图如图4-2所示。

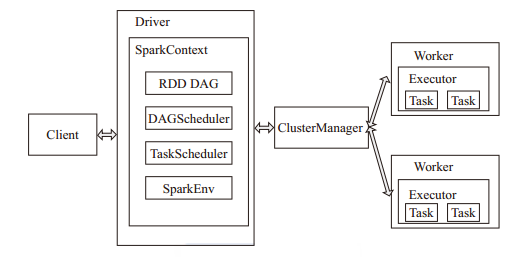


图4-2 Spark架构图[1]

图4-2的名词解释见表4-1。

表4-1 Spark架构图的名词概念解释

|  |  |
| --- | --- |
| **名词** | **解释** |
| **Client** | 客户端，提交应用 |
| **Driver** | 控制一个应用的执行，运行应用的main函数，创建SparkContext |
| **SparkContext** | 整个应用的上下文，控制应用的生命周期 |
| **RDD** | Spark基本计算单元 |
| **DAG** | 有向无环图 |
| **DAGScheduler** | 将作业（Job）按照RDD的宽依赖关系划分Stage，形成基于Stage的DAG，并提交Stage给TaskScheduler |
| **TaskScheduler** | 分发任务给Executor |
| **Executor** | 负责任务的执行 |

2. RDD

RDD是Spark的核心数据结构，通过对RDD的依赖关系形成spark的调度顺序，通过对RDD的操作形成整个Spark程序。

RDD类似于Scala集合，这使得我们可以像编写单机程序一样编写分布式程序。本质上，RDD相当于数据的一个元数据结构，存储Partition、Block、Node之间的映射关系，以及RDD之前的依赖转换关系。

3. 应用执行流程

Spark支持四种执行模式，如表4-2所述。

表4-2 Spark执行模式

|  |  |
| --- | --- |
| **执行模式** | **介绍** |
| **本地单机模式（Local）** | 所有Spark进程都运行在同一个JAVA虚拟机中 |
| **集群单机模式（Standalone）** | 使用Spark自带的任务调度框架 |
| **基于Mesos** | Mesos是一个类似于YARN的流行的开源计算框架 |
| **基于YARN** | 即Hadoop 2，它是一个与Hadoop关联的集群计算和资源调度框架 |

Spark应用的执行流程为：用户提交一个应用程序（APP）；Driver运行APP的main函数并创建SparkContex；接着Driver以RDD的Action算子将RDD Graph划分为作业（Job）；接着，DAG Scheduler将Job按照RDD的宽依赖关系划分成Stage，形成基于Stage的DAG，并提交Stage给TaskScheduler；TaskScheduler将Stage中并行的任务分发到Executor中；Executor启动Task进程执行任务，每个Task计算一个分片的数据。

#### 4.1.1.3 Spark工作机制

1. 调度机制

Spark有4个级别的调度，分别是：Application、Job、Stage和Task。

（1）Application调度：FIFO（先入先出）。

（2）Job调度：FIFO和FAIR（时间片轮转）。

（3）Stage调度：由DAG Scheduler来调度，使用基于Stage的DAG，从最后执行的Stage为根进行广度优先遍历，找到最开始执行的Stage。

2. 容错机制

Spark主要有两种容错机制，分别是：Lineage机制和CheckPoint机制。

（1）Lineage机制：这是一种粗粒度的记录更新来实现容错的机制。它通过记录RDD间的Transformation操作，当RDD的某些分区数据丢失，便重新运算整个RDD，以恢复丢失的数据分区。

（2）CheckPoint机制：这是一种将RDD写入磁盘做检查点的容错机制。当DAG中的Lineage过长，重算开销太大（如在PageRank中），于是可以从检查点的RDD开始重做Lineage，以减少开销。

3. 通信机制

Spark在模块间通信使用的是AKKA框架，AKKA是一个基于Scala语言开发的库，可以用来编写Actor应用。AKKA Actor采用树形结构来组织Actor，Actor之间的通信是异步的，每个Actor都有一个信箱。Actor之间，发送通过“！”符号发送消息，接收通过receive方法中的case模式匹配接收和处理消息。Spark中的Client、Master和Worker都是一个Actor。

AKKA的优势和特性：并行和分布式、可靠性、高性能、去中心和良好的可拓展性。

Spark通信模型如图4-3所示。

### 4.1.2 GraphX并行图计算框架

Spark GraphX是一个基于Spark平台的分布式图处理框架[21]，它提供各种用于图计算和图挖掘的简洁易用的接口，使得人们对于分布式图处理的需求得以满足。

众所周知，社交网络中人与人之间存在很多关系链，例如微博的关注和被关注， 微信的好友和群聊关系等，这些都是大数据产生的地方，通过图计算可以进行社区发现和数据挖掘。Spark GraphX由于底层是基于Spark来处理的，所以天然就是一个分

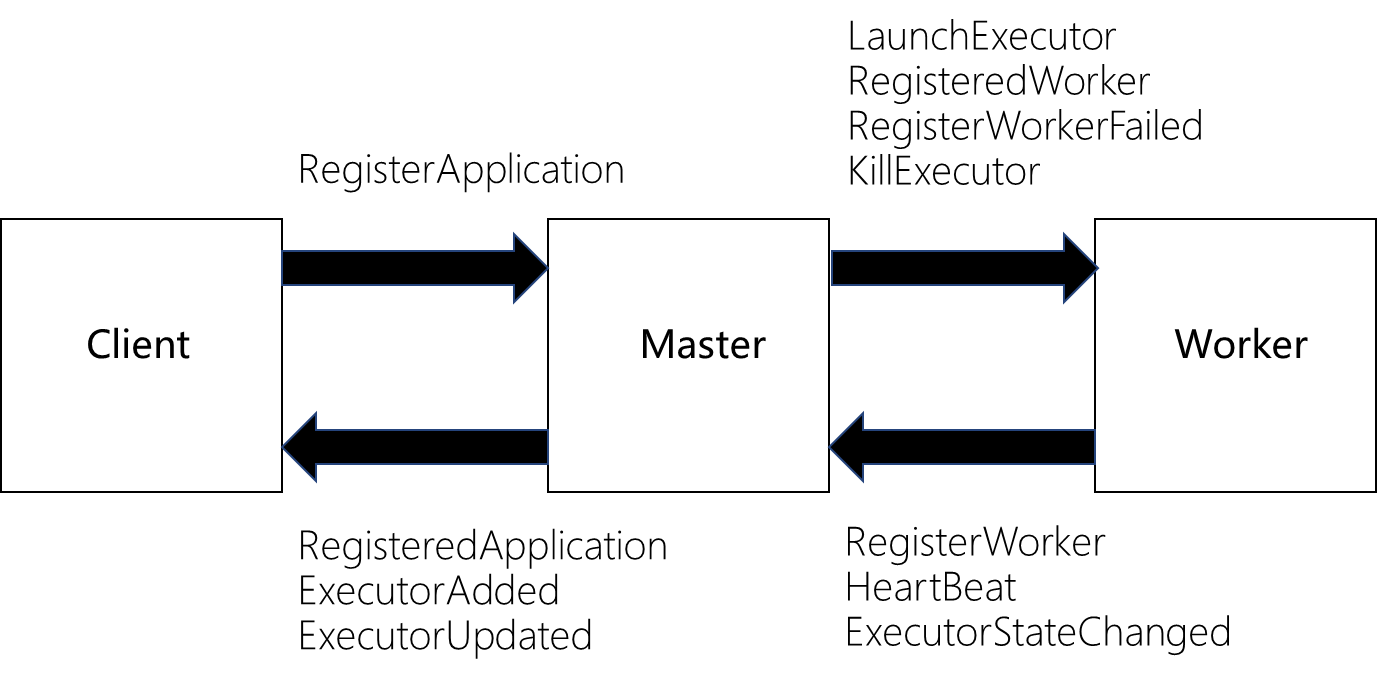


图4-3 Spark通信模型

布式的图处理系统。图的分布式或者并行处理其实是把图拆分成很多的子图，然后对这些子图进行分阶段迭代并行计算。

#### 4.1.2.1 GraphX实现分析

GraphX的核心抽象是弹性分布式属性图（Resilient Distributed Property Graph），这是一种点和边都带属性的有向多重图。它扩展了Spark RDD的抽象，有Table和Graph两种视图，而只需要一份物理存储。两种视图都有自己独有的操作符，从而获得了灵活操作和执行效率。如图4-4所示。

GraphX的底层设计有以下几个关键点。

首先，对Graph视图的所有操作，最终都会转换成其关联的Table视图的RDD操作来完成。这样对一个图的计算，最终在逻辑上，等价于一系列RDD的转换过程。因此，GraphX最终具备了RDD的3个关键特性：Immutable、Distributed和Fault-

Tolerant，其中最关键的是Immutable（不变性）。逻辑上，所有图的转换和操作都产生了一个新图；物理上，GraphX会有一定程度的不变顶点和边的复用优化，对用户透明。

其次，两种视图底层共用的物理数据，由RDD[VertexPartition]和RDD[EdgePart-

ition]这两个RDD组成。点和边实际都不是以表Collection[tuple]的形式存储的，而是由VertexPartition和EdgePartition在内部存储一个带索引结构的分片数据块，以加速不同视图下的遍历速度。不变的索引结构在RDD转换过程中是共用的，降低了计算和存储开销。

最后，图的分布式存储采用点分割模式，使用partitionBy方法，由用户指定不同的划分策略（PartitionStrategy）。划分策略会将边分配到各个EdgePartition，顶点Master

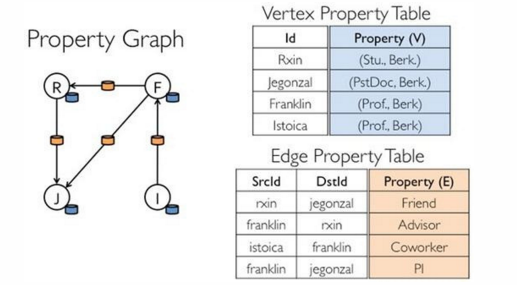


图4-4 GraphX的两种视图表示[22]

分配到各个VertexPartition，EdgePartition也会缓存本地边关联点的Ghost副本。划分策略的不同会影响到所需要缓存的Ghost副本数目，以及每个EdgePartition分配的边的均衡程度，需要根据图的结构特征选取最佳策略。目前有EdgePartition1d、EdgePartition2d、RandomVertexCut和CanonicalRandomVertexCut这四种策略。

#### 4.1.2.2 GraphX存储模式

Graphx借鉴PowerGraph，使用的是Vertex-Cut（点分割）方式存储图，用如下所示的三个RDD存储图数据信息：

（1）VertexTable(id, data)：id为顶点ID；data为顶点存储的属性。

（2）EdgeTable(pid, src, dst, data)：pid为分区ID；src为源顶点ID；dst为目的顶点ID；data为边存储的属性。

（3）RoutingTable(id, pid)：id为顶点ID；pid为分区ID。

GraphX点分割存储如图4-5所示。

#### 4.1.2.3 计算模式

1. 图计算模式

目前基于图的并行计算框架已经有很多，比如来自Google的Pregel、来自Apache开源的图计算框架Giraph和HAMA以及最为著名的GraphLab，其中Pregel、HAMA和Giraph非常类似，都是基于BSP（Bulk Synchronous Parallell）模式。

Bulk Synchronous Parallell，即整体同步并行，它将计算分成一系列的超步（superstep）的迭代。从纵向上看，它是一个串行模式，而从横向上看，它是一个并行的模式，每两个超步之间设置一个栅栏（barrier），即整体同步点，确定所有并行的计算都完成后再启动下一轮超步。

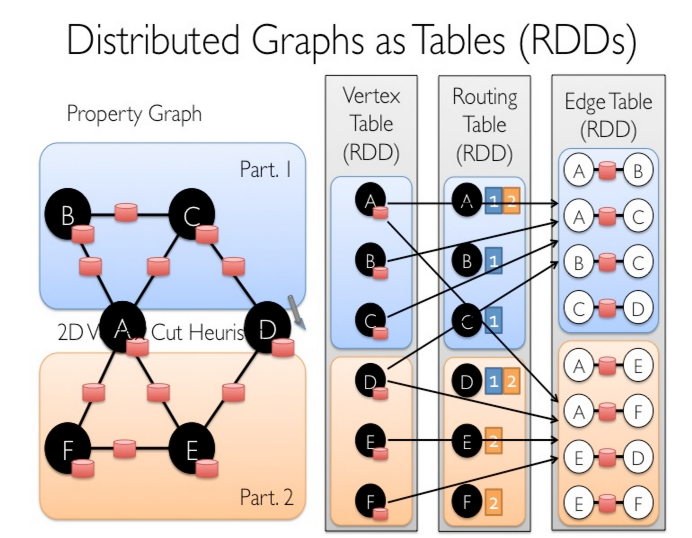


图4-5 GraphX点分割存储示意图[22]

每一个超步包含如下的三部分内容：

（1）计算：每一个处理单元利用上一个超步传过来的消息和本地的数据进行本地计算。

（2）消息传递：每一个处理单元计算完毕后，将消息传递给与之关联的其它处理单元。

（3）整体同步点：用于整体同步，确定所有的计算和消息传递都完成后，进入下一个超步。

2. GraphX计算模式

GraphX最重要的操作就是聚合消息ggregateMessages)，aggregateMessages是GraphX最重要的API之一，是1.2版本添加的新函数，用于替换mapReduceTriplets。目前mapReduceTriplets最终也是使用兼容的aggregateMessages。

它的计算过程为：

（1）Map：应用于每一个Triplet上，生成一个或者多个消息，消息以Triplet关联的两个顶点中的任意一个或两个为目标顶点；

（2）Reduce：应用于每一个Vertex上，将发送给该顶点的消息合并起来。

aggregateMessages最后返回的是一个VertexRDD[A]，包含每一个顶点聚合之后的消息（类型为A），没有接收到消息的顶点不会包含在返回的VertexRDD中。

## 4.2 LDA并行方案

### 4.2.1 潜在图结构

我们在3.4节已经介绍了LDA的多种推断技术，并给出了MAP、VB和CGS方法得到的词主题更新式，同时还对这些更新式在数学上形式一致的现象给出了解释，这是因为它们表示的物理意义是相同的，具体见3.4小节。

本文选用MAP EM算法来推断模型参数，这是因为并行EM过程不会造成精度损失，并且MAP方法下得到的更新式在工程上更易于实现。

于是串行EM算法如算法7所示。

|  |
| --- |
| **算法7** 串行LDA EM算法 |
| 输入：  第*j*篇文档的词向量（未作词频统计）  超参数  超参数  主题数目  全局数据：  记录第*j*篇文档中属于第*k*个主题的词的数目  记录第*k*个主题拥有词*w*的数目  记录第*j*篇文档拥有词的数目  记录第*k*个主题拥有词的数目  输出：  参数估计 文档-主题分布矩阵  参数估计 主题-词分布矩阵   1. 将全局计数器置零；//1-10为初始化过程 2. for all documents do 3. for all words i in do 4. ； 5. ； 6. ； 7. ； //w为第j篇文档的第i个词 8. ； 9. end for 10. end for 11. while没有达到最大迭代次数 do //执行迭代过程   // E-step：基于MAP计算隐含变量的后验分布，并为词分配主题   1. for all documents do 2. for all words i in do 3. 计算后验分布：； 4. 为词分配主题：； 5. end for 6. end for   // M-step：极大似然方法估计模型参数   1. 按照词的所属主题，更新； 2. end while // 迭代结束   // 输出模型参数   1. 遍历所有文档和主题，计算：； 2. 遍历所有主题和词汇表中的词，计算：； |

LDA中文档里的每个词都属于一个主题，LDA的训练过程是，一轮迭代中，为每篇文档里的每一个词重新选择主题，选择的依据是MAP方法计算的隐含变量后验分布。

LDA模型主要有两类数据，分别是文档和出现在文档中的词，现在把语料库中的每一篇文档和词汇表中的每一个词都用一个顶点来表示，然后用一条由文档指向词的边把文档和文档中出现的词都连接起来，就得到了一个图结构。在上面叙述的EM算法中，可以发现所有对文档和词的操作都可以转换为对边顶点三元组的遍历操作，所以用EM算法学习LDA模型存在一个潜在的图结构。

在LDA实现算法里，表示一篇文档*j*中属于各个主题*k*的词的数目，表示词项*w*属于各个主题*k*的词的数目，这两个计数变量刚好和LDA模型的两类数据——文档和词一一对应，于是可以用文档顶点存储、用词顶点存储。由于在LDA图结构中，每条由文档指向词的边都是独一无二的，所以词在文档中出现的次数，也就是词频，可以存储在图的边上。

通过上述的操作，将整个语料库转化为图，它把、存储在文档顶点和词顶点上，把词频信息存储在边上，这样，整个文档的聚类结果矩阵、模型矩阵和语料库词频矩阵都表达在图结构中。而LDA实现算法的核心——为每篇文档的每个词重新分配主题也被巧妙的转化为在图中的边顶点三元组上的操作，这样，就把LDA算法处理过程表达为对边的遍历处理过程，又由机器学习的多轮迭代性质，将其简单高效的实现在GraphX之上。

### 4.2.2 基于GraphX的并行LDA实现

由于我们采用EM算法来推断LDA模型，并且将其实现在Spark GraphX之上，所以暂且称其为GraphX EM LDA。

基于GraphX的并行LDA实现主要包括两个步骤，分别是初始化图和迭代并行图计算。

#### 4.2.2.1 GraphX EM LDA初始化

我们对语料库进行文本预处理（预处理的并行实现方式见4.3节），将文档集转化为词频矩阵，词频矩阵的每一行代表一篇文档的词频向量。

LDA的输入数据为词频矩阵RDD[(docID：Long, termCount：Vector)]，其存储格式如表4-3所示。

表4-3 词频矩阵存储格式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **文档** **词** | **Term0** | **Term1** | **Term2** | … |
| **Doc0** | 1 | 3 | 0 |  |
| **Doc1** | 5 | 4 | 3 |  |
| … |  |  |  |  |

为了对文档顶点和词顶点统一编号，将文档顶点和词顶点的顶点ID进行重新分配。文档顶点ID编号从0递增，词顶点编号从-1递减。表4-3中词频矩阵转换为下表所示。

表4-4 顶点ID重新分配之后得到的词频矩阵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **文档** **词** | **-1** | **-2** | **-3** | … |
| **0** | 1.0 | 3.0 | 0.0 |  |
| **1** | 5.0 | 4.0 | 3.0 |  |
| … |  |  |  |  |

根据文档到词的关系可以生成GraphX边，如图4-6所示。



图4-6 由输入词频矩阵得到的GraphX图结构

其中边的格式为[(文档顶点ID，词顶点ID，词频)]，如表4-5所示。

表4-5 GraphX边

|  |  |  |
| --- | --- | --- |
| (0, -1, 1.0) | (0, -2, 3.0) | 无 |
| (1, -1, 5.0) | (1, -2, 4.0) | (1, -3, 3.0) |

构建GraphX边的伪代码如算法8所示。

|  |
| --- |
| **算法8** 构建GraphX边 |
| *M* 语料库中文档数目  *V* 词汇表词的数目  *D* 词频矩阵  // 在每一篇文档中，创建一条由文档指向词频计数不为零的词的边 |
| 1. for document *j* from [0, *M*-1] do 2. for word *i* in doc *j*, (*i* from [-1, -*V*] && *D*[*j*][*i*] != 0) do 3. generate an edge (*j*, *i*, *D*[*j*][*i*]) as an element of EdgeRDD; 4. end for 5. end for |

由GraphX边构建GraphX顶点的伪代码见算法9。

|  |
| --- |
| **算法9** 构建GraphX顶点 |
| 1. for all edge in EdgeRDD do // 调用flatMap函数，返回RDD[(vertexID, sum)] |
| 1. 随机初始化*K*维向量*gamma*，并归一化；// 随机初始化隐含变量的后验分布 |
| 1. 用边的值进行加权，得到*K*维向量*sum*；// 为词随机分配主题 |
| 1. 生成(*srcID*, *sum*)和(*dstID*, *sum*)两个临时顶点； 2. end for |
| 1. 将RDD[(vertexID, sum)]按顶点ID聚合，ID相同的*sum*向量叠加，作为顶点数据； |

接着，用得到的顶点和边构建GraphX图，并指定分区策略，使得边按文档顶点分组，这样一篇文档的文档顶点和词顶点都会存储到同一台机器上，即按文档分布式存储，这样做的好处是每个EdgePartition所需要缓存的本地边关联点的Ghost副本较少，且各个EdgePartition分配的边数目较为均衡。构建GraphX图的代码如下：

|  |
| --- |
| val graph = Graph(docTermVertices, edges) //构建GraphX图  .partitionBy(PartitionStrategy.EdgePartition1D) //指定分布策略 |

#### 4.2.2.2 GraphX EM LDA迭代并行图计算

GraphX EM LDA初始化完毕之后，就将整个语料库构造成了GraphX图，接下来就可以在图上进行并行迭代计算。

由于EM迭代中，E步骤的计算隐含变量后验分布和为词分配主题都是在操作（文档顶点，词顶点，词频）三元组，而M步骤中极大似然方法估计模型参数则是在文档顶点和词顶点上做叠加运算，所以EM迭代过程可以用GraphX中的聚合消息函数aggregateMessages来比较好地实现。

下面给出了GraphX EM LDA的迭代过程伪代码，如算法10所示。

|  |
| --- |
| **算法10** GraphX EM LDA迭代过程 |
| ：超参数  ：超参数  *W*：词汇表中词的数目  *K*：主题数目  ：*K*维向量，文档顶点数据  ：*K*维向量，词顶点数据  ：*K*维向量，所有词顶点向量的叠加  *srcID*：文档顶点*ID*  *dstID*：词顶点*ID*  *msg*：*K*维向量，边三元组产生的消息 |
| 1. while 未达到最大迭代次数 do  2. = 所有词顶点向量值叠加；  // E-step：aggregateMessages Map阶段  3. for all edge in EdgeRDD do // 遍历整个GraphX图  4. for do // 计算隐含变量的后验分布  5. ；  6. end for  7. ； // 为词分配主题  8. sendToSrcID(*msg*)； // 发送消息到源顶点  9. sendToDstID(*msg*)； // 发送消息到目的顶点  10. end for  // M-step：aggregateMessages Reduce阶段  11. 顶点将收到的消息进行叠加，作为该顶点新的向量值；  12. end while |

伪代码中的第3-11步是由GraphX的aggregateMessages方法来实现的。

其中第3-10步是EM算法的E步骤，主要是为每个出现在文档中的词计算它属于各个主题的概率，并为词分配主题。第3-10步对应着aggregateMessages函数的Map阶段，它由边三元组生成*K*维消息，并将*msg*乘以词频然后发往两端顶点，消息生成与发送的过程是并行的。*K*维消息的生成依据为MAP方法得到的词主题更新式，本质上，*K*维消息就是隐含变量的后验分布，将消息乘以词频就相当于为词分配主题。

第11步是EM算法的M步骤，通过极大似然方法来更新模型参数，即为潜变量计算充分统计量，在这里指更新文档顶点向量值和词顶点向量值。第11步对应于aggregateMessages函数的Reduce阶段，由于文档顶点得到的是出现在文档中的词属于各个主题的数目，而词顶点得到的是各个文档中它们属于各个主题的数目，所以将顶点收到的消息叠加，并将叠加后的值作为顶点新的向量值，就更新了文档顶点向量值和词顶点向量值。

综上，总结一下GraphX EM LDA的迭代过程：

（1）计算所有词顶点向量值叠加的结果；

（2）调用aggregateMessages方法进行消息的生成、发送和聚合；

（3）将消息聚合后得到的向量值作为顶点数据的更新值。

## 4.3 基于图的困惑度并行计算方法

困惑度是评价LDA模型的重要指标，它指LDA模型确定一个词所属主题的困惑程度。困惑度越小，得到的LDA模型越好。困惑度的计算方法为，将数据集切分成训练集和测试集，接着用训练集建立模型，然后再用测试集计算困惑度，困惑度的计算公式如下：

(4-1)

式中表示测试集；表示整个语料库的对数似然函数；表示整个语料库词块的个数。由于比较容易求解，于是计算困惑度的主要工作就转化为计算语料库的对数似然函数。

本文在2.6.1小节已经介绍过似然函数，现在将似然函数展开，可以得到：

(4-2)

即语料库的对数似然函数等于整个语料库所有词的对数生成概率累加，而词的生成概率又由模型参数和决定，于是词的生成概率如下所示：

(4-3)

而模型参数和又可以从图顶点得到，这样就把求解困惑度和图联系了起来。

基于图的LDA模型困惑度并行求解方法为：

1. 首先依据测试集中文档到词的关系生成边，接着利用由训练集建立的图的词顶点，得到词属于各个主题的概率，然后在遍历边的过程中，将其乘以词频，这相当于为词分配主题，然后发往边两端的顶点，接着将顶点上的*K*维向量进行叠加，就得到了文档顶点和词顶点。其中文档顶点存储的是测试集各个文档的主题分布，词顶点存储的是该词属于各个主题的词的数目。然后用文档顶点和词顶点构造基于测试集生成的新图。

2. 接着利用消息聚合函数aggregateMessages()来进行词块对数生成概率的生成、发送与聚合。在aggregateMessages Map阶段，遍历每一条边，利用新图的模型参数生成每一个词块的对数生成概率，将其乘以词频后发往边的目的顶点，即词顶点；在aggregateMessages Reduce阶段，遍历每一个词顶点，将对数身生成概率叠加，便得到了语料库的对数似然函数，然后就可以求出测试集的困惑度。

## 4.4 文本预处理的并行实现

由于本文实现的LDA算法接受的输入格式为形如RDD[(*docID*：Long, *termCount*：Vector)]的词频矩阵，而输入的数据集格式为词袋形式，所以需要对数据集进行文本预处理，将其转化为满足输入的格式。输入数据集以一个txt文件表示一个文档集，对于每个文档集，*M*是文档的数目，*W*是词汇表中词的数目，*NNZ*是非零计数的数目，即边的数目。输入数据集txt文件的前三行是文档集的元数据，后面是非零计数三元组，它的格式如表4-6所示。

表4-6 数据集格式

|  |  |  |
| --- | --- | --- |
| *M* | | |
| *W* | | |
| *NNZ* | | |
| *docID* | *wordID* | *count* |
| *docID* | *wordID* | *count* |
|  |  |  |
| *docID* | *wordID* | *count* |

去除前三行的元数据后，观察每一行的非零计数三元组（*docID*, *wordID*, *count*），可以发现将整个数据集按照*docID*进行分类，然后再按照*wordID*填充词频计数数组，就得到了词频矩阵，于是我们的并行文本预处理方案如图4-7所示。

## 4.5 并行LDA算法的不足之处

本文实现的GraphX EM LDA使用图顶点存储文档和词的主题统计信息，以图边



图4-7 文本预处理并行实现方案

存储文档和词的关系，以遍历图边的方式训练模型，完美地表达了LDA的训练逻辑。然而，GraphX的aggregateMasseges 方法处理边三元组需要从两端顶点拉取顶点数据，生成消息后需要再把消息发往两端顶点，这里拉取数据和消息汇聚都会引起大量的shuffle过程。在Spark中，shuffle是指两个分区间的数据移动。

shuffle完整过程分为如下四个步骤：

（1）发送方将待发送的数据写入本地磁盘；

（2）数据序列化后经网络传输；

（3）接收方接收流数据，并读取本地磁盘；

（4）接收方反序列化数据。

也就是说，一次shuffle过程就涉及两次序列化、两次读写磁盘，而序列化需要消耗CPU资源，读写磁盘需要消耗大量时间。所以，大量的shuffle过程将会拖累LDA的迭代速度。

此外，在4.2.2.1小节中，由于采取点分割分布式存储的方式构建GraphX图，并指定点分割划分策略，使得边按文档顶点分组，这会使得一个词顶点在多个EdgePartition中有Ghost副本，而词顶点Master则会被划分到VertexPartition中 。基于这样的分布式图存储，当在顶点叠加接受到的消息时将会不可避免地产生大量用于同步的机器之间的通信。

而且，由于基于GraphX的并行方案对每一篇文档中的每一个独一无二的词都建立了一条边，因而当数据集比较大时，将得到一个巨型图，这样做的好处是可以并行的操作每一篇文档中的每一个词，然而这种空间换取时间的方法注定了基于GraphX的并行LDA方案对于存储空间是不友好的，对硬件配置的要求也比较高。

因而，用GraphX实现的EM LDA性能并不完美，可以这么说，GraphX为LDA在Spark上的实现提供了一个完美的结构，而性能却不能令人满意。但是，spark由于其强大的普适性，为了减少数据多平台跨越的烦恼，一般情况下，使用基于Spark GraphX实现的EM LDA训练语料库还是比较好的。

## 4.6 本章小结

本章首先介绍了Spark云平台和GraphX并行图计算框架，接着分析了用EM算法学习LDA模型所存在的潜在图结构，然后详细阐述了基于Spark GraphX的并行LDA算法的实现，接着详细阐述了基于图的LDA模型困惑度计算方法，然后给出了文本预处理的并行实现方案，最后对本LDA并行算法的不足之处进行了说明。

第5章 实验及结果分析

## 5.1 实验说明

### 5.1.1 硬件配置

本次实验使用实验室服务器集群中的9台机器组成的Spark平台，其中1台作为Master节点，余下的8台机器作为Worker节点。9台机器的配置相同，如表5-1所示。

表5-1 集群的机器配置

|  |  |
| --- | --- |
| **CPU型号**  **CPU主频**  **CPU核数**  **内存容量**  **硬盘容量** | Intel(R) Xeon(R) CPU E5506  2.13GHz  4  12GB  550GB |

### 5.1.2 软件配置

实验每台机器的软件配置如表5-2所示。

表5-2 软件配置

|  |  |
| --- | --- |
| **操作系统版本**  **JAVA版本**  **Scala版本**  **Spark版本** | CentOS 6.4  JDK 1.7.0\_60  Scala-2.10.4  Spark-1.5.2 |

### 5.1.3 数据来源

此次试验采用了一些主题模型领域经典的数据集，它们均来自于UCI机器学习库，下载地址为：<http://archive.ics.uci.edu/ml/datasets/Bag+of+Words>。这些数据集都经过分词和去除停用词处理，并过滤掉了出现次数少于10次的低频词汇，然后以词袋形式存储在一个txt文件中，可以完美地应用于主题模型实验。出于版权的原因，这些数据集都不提供文档的名字、类标签和文档级别的元数据。

由于集群计算资源有限，本文选取了其中四个数据集用于本次实验，如表5-3所示。

表5-3 数据集介绍

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **数据集** | **KOS** | **NIPS** | **Enron** | **NYTimes** |
| **文档个数M**  **词汇表中词的数目W**  **非零计数行的数目NNZ** | 3430  6906  353160 | 1500  12419  746316 | 39861  28102  3710420 | 300000  102660  69679427 |

从上表中可以看到，数据集KOS和NIPS属于小数据集，Enron属于中等数据集，NYTimes属于大数据集，我们用KOS、NIPS和Enron三个数据集来进行困惑度实验，用NYTimes数据集来进行并行性能实验。

## 5.2 困惑度实验

### 5.2.1 困惑度

困惑度（Perplexity）是语言模型常用的模型评价指标，它可以粗略地理解为“对于一篇文档，LDA模型有多不确定它是属于某个主题的”，也就是LDA模型确定一个词所属主题的困惑程度。

实验中先将数据集切分成训练集和测试集，二者大小之比为9:1，接着用训练集建立模型，然后再用测试集计算困惑度，计算困惑度的公式如下所示：

(5-1)

在本次实验中，使用基于GraphX图的方法来计算困惑度，具体实现细节见4.3小节

由于基于图的方法对于资源的开销比较大，因而选用中小数据集进行困惑度实验。

对于LDA模型，困惑度越小，意味着模型各项指标越优秀，一般而言，主题数越多，困惑度越小，但是越容易发生过拟合（overfitting），之后主题数继续增多会导致困惑度上升；此外，迭代次数也会影响困惑度，迭代次数越多，困惑度越小。

接下来的两个实验利用困惑度来确定KOS、NIPS和Enron三个数据集的最优迭代次数和主题数，为之后的并行LDA主题聚类结果展示做好准备。

### 5.2.2 确定迭代次数的困惑度实验

如前所述，迭代次数越多，困惑度越小，模型越优秀，但是迭代次数的增大也会导致计算量的增大，于是我们通过实验来确定合适的迭代次数，使得模型足够优秀的同时，计算量还处在可接受的范围之内。

对于KOS、NIPS和Enron三个数据集，设定它们的主题数均为*K*=20，然后将迭代次数从20开始按10递增，由于EM算法随机初始化模型参数，这导致了模型聚类结果具有一定的随机性，于是我们对每个实验都重复进行5次，然后取它们的困惑度平均值。确定KOS、NIPS和Enron三个数据集最优迭代次数的困惑度实验结果分别如图5-1、图5-2和图5-3所示。

图5-1 KOS数据集迭代次数困惑度实验结果

图5-2 NIPS数据集迭代次数困惑度实验结果

从图5-1、图5-2和图5-3的实验结果可以发现，随着迭代次数的增大，困惑度起初会明显减少，然后下降趋势逐渐放缓直至进入平稳状态，根据实验结果图，可以得到，KOS数据集的最佳迭代次数为，NIPS数据集为，Enron数据集为。

图5-3 Enron数据集迭代次数困惑度实验结果

### 5.2.3 确定主题数目的困惑度实验

对于KOS、NIPS和Enron三个数据集，设定它们的迭代次数依次为80、80和70，然后将主题数从20开始按10递增，由于聚类结果存在随机性，我们对每个实验都重复进行5次，然后取它们的困惑度平均值。确定KOS、NIPS和Enron三个数据集最优主题数的困惑度实验结果分别如图5-4、图5-5和图5-6所示。

图5-4 KOS数据集主题数困惑度实验结果

图5-5 NIPS数据集主题数困惑度实验结果

图5-6 Enron数据集主题数困惑度实验结果

从上述实验结果可以发现，随着主题数的增多，困惑度开始减小，当达到最低点时，继续增大主题数，此时由于过拟合，困惑度又会开始上升，于是我们选取最低点为数据集的最佳主题数目。根据实验结果曲线图，可以得到，KOS数据集最佳的主题数为，NIPS数据集为，Enron数据集为。

## 5.3 并行LDA主题聚类结果展示

本小节进行并行LDA算法主题聚类结果展示，对KOS、NIPS和Enron三个数据集按照它们各自的最佳主题数和最佳迭代次数运行LDA进行处理，然后从生成的主题聚类结果中各挑选三个主题进行展示，每个主题配以权重最高的5个词，如表5-4、表5-5和表5-6所示。其中，打引号的主题标题是为了更清晰而手动添加的。

表5-4 KOS数据集部分主题聚类结果展示

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **“伊拉克战争”** | | **“美国金钱选举”** | | **“美国总统大选”** | |
| **term** | **weight** | **term** | **weight** | **term** | **weight** |
| iraq  aar  american  military  iraqi | 0.043  0.036  0.012  0.012  0.010 | campaign  party  democratic  million  money | 0.033  0.020  0.017  0.015  0.014 | november  republicans  house  governor  poll | 0.031  0.024  0.022  0.021  0.021 |

表5-5 NIPS数据集部分主题聚类结果展示

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **“神经网络”** | | **“图像识别”** | | **“语音识别”** | |
| **term** | **weight** | **term** | **weight** | **term** | **weight** |
| network  neural  system  input  output | 0.071  0.034  0.022  0.017  0.015 | image  feature  images  vector  recognition | 0.021  0.015  0.014  0.013  0.012 | recognition  word  speech  system  training | 0.023  0.022  0.021  0.015  0.013 |

表5-5 Enron数据集部分主题聚类结果展示

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **“商业”** | | **“劳工法案”** | | **“会议”** | |
| **term** | **weight** | **term** | **weight** | **term** | **weight** |
| market  campany  business  stock  trading | 0.017  0.016  0.013  0.010  0.008 | page  court  employees  law  labor | 0.026  0.013  0.013  0.009  0.008 | date  conference  phone  fax  office | 0.031  0.022  0.022  0.021  0.020 |

从上述实验结果展示可以看到，并行LDA算法生成的主题聚类结果表述清晰，且不同主题之间的差异也比较明显，即聚类结果类内相似性较大，类间相似性较小，因此算法效果较好。注意到，每个主题排名靠前的词权重依然不是很高，这是因为LDA模型会为每个词都分配概率，而语料库的词汇表一般都比较大，这使得高频词汇分配的权重看起来并不高。

## 5.4 集群并行性能实验

接下来开始测试并行LDA算法在Spark集群上的性能表现，主要从经典的三个并行指标入手，分别是Speed-up、Size-up和Scale-up，它们分别体现了并行算法的加速性能、承载能力和可拓展性。

### 5.4.1 Speed-up实验

Speed-up（加速比），测试相同数据量条件下，集群节点数增加对同一应用运行时间的影响，它的计算方法是，保持数据大小恒定，然后增加集群节点数，观察运行时间的变化，具体公式如下所示：

(5-2)

式中，是指集群节点数为*m*时应用执行的时间。如果随着的增加保持线性增长，则说明并行算法加速性能良好，在集群中的加速效果理想。

在Speed-up实验中，从NYTimes数据集切分出了100MB，200MB，300MB，400MB规模的子数据集，并将它们用于此次实验，并行算法的Speed-up结果如图5-7所示。

由于选取的节点数为1到8，所以在此次实验中，最理想的加速比为，观察实验结果，可以发现4种规模的数据集得到的加速比曲线都比较接近于理想加速比曲线，并且四条曲线的重合度很高，这说明本文实现的并行LDA算法对不同规模大小的数据集都实现了比较好的加速效果。注意到加速比曲线没有趋向于平缓，这是因为当节点数达到8时，计算能力依然是稀缺资源，也就是说此时继续增大节点数，计算时间还可以减少。

### 5.4.2 Size-up实验

Size-up用于测试并行算法的时间复杂度，它能够在数据集不断增长的情况下反映并行算法的性能，计算Size-up的方法是，保持集群节点数目不变，然后等量增加数据集的大小，观察运行时间的变化，它的计算公式如下：

(5-3)

式中表示基准大小数据集运行的时间；表示*m*倍基准大小数据集运行的

图5-7 基于Spark的并行LDA算法的Speed-up

时间。

本实验采用NYTimes数据集，由于NYTimes数据集大小为957MB，可以构造一个七千万条边的巨型图，因此整个数据集全部使用会对内存造成巨大的负担，计算的时间也会非常长，所以对NYTimes数据集进行切分，取100MB，200MB，300MB，400MB用于Size-up实验，得到的并行LDA算法在Spark集群上的Size-up结果如图5-8所示，为了图示的清晰，这里只展示了节点数为2,4,6,8时的Size-up曲线。

通过Size-up实验，可以发现Size-up曲线尽管线性地增长，但没有超过，这说明本文实现的并行LDA算法具有一定的承载能力。

### 5.4.3 Scale-up实验

Scale-up（可拓展性），测试当数据集大小和集群节点数呈相当水平增长时，并行算法运行所消耗时间的变化，具体评测Scale-up的方法是，在扩大数据集的同时，增

图5-8 基于Spark的并行LDA算法的Size-up

加节点数目，然后观察执行时间的变化。Scale-up的计算公式如式5-4所示。

(5-4)

在Scale-up实验中，依次用100MB数据集、1个节点，200MB数据集、2个节点，300MB数据集、3个节点，400MB数据集、4个节点，对并行算法进行Scale-up指标测试，得到的并行LDA算法的Scale-up结果如图5-9所示。

通过Scale-up实验，可以发现，Scale-up曲线稳定在y=1附近，这表明本文实现的并行LDA算法可拓展性良好。注意到，当数据集规模为200MB，节点数为2时

Scale-up曲线没有明显的下降趋势，当数据集规模为300MB，节点数为3时Scale-up曲线甚至大于1，这可能是因为此时节点数增多，计算能力增强所带来的正面影响大于节点之间通信开销所产生的负面影响，使得执行时间没有增加甚至减少。

## 5.5 本章小结

本章首先对本次实验进行了说明，包括集群软硬件环境和所使用的数据集来源；

图5-9 基于Spark的并行LDA算法的Scale-up

接着通过困惑度实验确定了数据集的最佳迭代次数和最佳主题数目，为接下来的LDA主题聚类结果展示做好准备；在LDA主题聚类结果展示中，我们展示了并行LDA算法推导出的主题，实验结果表明，各个主题表述清晰且差异比较明显；最后进行了集群并行性能实验，通过Speed-up、Size-up和Scale-up三个并行指标对本文实现的并行LDA算法的性能进行衡量，实验结果表明，所实现的并行LDA算法性能良好。

结 论

本文针对大数据背景下人们对海量文本处理的迫切需求，利用传统串行LDA（Latent Dirichlet Allocation）算法在处理大规模文档集时需要反复迭代的特点，选用内存并行计算框架Spark，对串行算法并行化，实现了一种基于Spark平台的并行LDA算法。

首先，在详细介绍LDA文本主题生成模型和比对多种LDA推断方法的基础之上，选择了MAP EM算法作为推断模型参数的方法。由于用EM算法学习LDA模型存在一个潜在的图结构，本文将给出的基于图的并行LDA算法实现在了Spark GraphX之上，该并行方案实现了一种同时为所有文档中所有的词分配主题的方法，使得LDA算法效率大大提高。

其次，针对并行LDA算法基于图的的实现方式，实现了一种基于图的LDA模型困惑度计算方法，它能够有效地利用原有图的词顶点快速地计算出测试集的困惑度；还针对输入数据集的格式，给出了文本预处理的并行实现方案。

最后，选取了主题模型领域的经典文本数据集，针对LDA模型的特点，设计了困惑度实验，用来寻找数据集的最优主题数目和迭代次数；通过并行LDA主题聚类结果展示实验说明了所实现的并行算法聚类得到的结果类间相似性较小，类内相似性较大；此外，还设计了Speed-up、Size-up和Scale-up三个集群并行性能测评实验，通过实验验证了所实现的并行LDA算法具有比较良好的加速性能和可拓展性，同时还有一定的承载能力。

然而，基于GraphX的LDA并行实现方式也存在不足之处，由于它是通过构造一个图来进行并行计算，因而构造图时将产生一些时间开销，对内存的占用也比较高，并且迭代过程中还会引起大量的shuffle过程，Ghost顶点和Master顶点同步也会产生一些通信开销，因而，基于GraphX的LDA并行实现方式还有许多提升空间。

一种可行的LDA并行方案优化方法是每次采样小批量文档进行模型训练，通过多次迭代得到模型参数，从而避免了同时对整个语料库进行处理所带来的问题，这可能是LDA并行方案的一种改进方向。

致 谢

此次毕设的顺利完成离不开我的导师李天瑞教授的细心指导，李老师在学科专业领域上高超的造诣、严谨治学的作风和孜孜以求的学术精神使我印象深刻，在我写论文期间一直鼓励和支持我，帮助我完成最终的论文写作，这些让我永远铭记在心。

感谢李老师门下的研究生学长学姐，因为他们的建议和意见，为我解决毕业设计过程中的很多问题提供了方法。更要感谢李勇学长，感谢他能够乐于与我分享他的资料、心得，能够详细地解答我的疑问，并为我的毕设指明方向。

我也要感谢一起在实验室中做实验的同学们，正是大家相互之间的鼓励与帮助，使得毕设的实验过程得以顺利完满结束。

我还要感谢我的家人，感谢他们一直以来对我的鼓励、理解、支持，他们是我在未来道路上勇敢前行的不竭动力。

大学四年即将画上句号，感谢交大为我提供的学习环境和基础设施，我在这里学习生活了四年，感谢交大的所有老师，感谢我的同学朋友们！

参考文献

1. 高彦杰. Spark大数据处理[M]. 机械工业出版社, 2014.
2. Blei D.M., Ng A.Y., Jordan M.I. Latent Dirichlet Allocation[J]. Journal of Machine Learning Research, 2003, 3: 993-1022.
3. Griffiths T.L., Steyvers M. Finding Scientific Topics[J]. Proceedings of the National Academy of Sciences of the United States of America, 2004, 101: 5228-5235.
4. Hofmann T. Probabilistic Latent Semantic Analysis[C]. Fifteenth Conference on Uncertainty in Artificial Intelligence, 1999: 289-296.
5. Rosen-Zvi M., Griffiths T., Steyvers M., et al. The Author-Topic Model for Authors and Documents[C]. Conference on Uncertainty in Artificial Intelligence, 2010: 487-494.
6. Teh Y.W., Newman D., Welling M. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation[C]. Advances in Neural Information Processing Systems. 2006: 1353-1360.
7. Chien J.T., and Wu M.S. Adaptive Bayesian Latent Semantic Analysis[C]. Audio, Speech, and Language Processing, IEEE Transactions on, 2008: 198-207.
8. Nallapati R., Cohen W., Lafferty J. Parallelized Variational EM for Latent Dirichlet Allocation: An Experimental Evaluation of Speed and Scalability[C]. IEEE Computer Society, 2007: 349-354.
9. 李文波, 孙乐. 一种LDA模型的高效并行求解算法[C]. 全国信息检索学术会议, 2010: 433-440.
10. Porteous I., Newman D., Ihler A., et al. Fast Collapsed Gibbs Sampling for Latent Dirichlet Allocation[C]. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008: 569-577.
11. 邱卓林. 并行LDA算法的研究与实现[D]. 北京邮电大学硕士论文, 2015.
12. Low Y., Gonzalez J.E., Kyrola A., et al. GraphLab: A New Framework For Parallel Machine Learning[J]. Eprint Arxiv, 2014.
13. Wang Y., Bai H., Stanton M., et al. PLDA: Parallel Latent Dirichlet Allocation for Large-Scale Applications[M]. Springer Berlin Heidelberg, 2009: 301-314.
14. Wen L., Rui J., Tingting H.E., et al. Accelerating Hierarchical Distributed Latent Dirichlet Allocation Algorithm by Parallel GPU[J]. Journal of Computer Applications, 2013, 33(12): 3313-3316.
15. Beal M.J. Variational Algorithms for Approximate Bayesian Inference[D]. Doctoral Dissertation, University College London, 2003.
16. Dempster A. Maximum Likelihood from Incomplete Data via the EM Algorithm[J]. Journal of the Royal Statistical Society, 1977, 39(1): 1-38.
17. Ahmed S.E. Markov Chain Monte Carlo Stochastic Simulation for Bayesian Inference[J]. Technometrics, 2008, 50(1): 497-537.
18. 博客园. MCMC(Markov Chain Monte Carlo) and Gibbs Sampling[EB/OL]. <http://www.cnblogs.com/ywl925/archive/2013/06/05/3118875.html>, 2016-3-14.
19. Heinrich G. Parameter Estimation for Text Analysis[J]. Technical Report, 2005.
20. Asuncion A., Welling M., Smyth P., et al. On Smoothing and Inference for Topic Models[C]. Conference on Uncertainty in Artificial Intelligence, 2012: 27-34.
21. Xin R.S., Gonzalez J.E., Franklin M.J., et al. GraphX: A Resilient Distributed Graph System on Spark[C]. International Workshop on Graph Data Management Experiences and Systems, 2013: 1-6.
22. 推酷. Spark入门实战系列——Spark图计算GraphX介绍及实例[EB/OL]. <http://www.tuicool.com/articles/v6je2ei>, 2016-4-10.