第八课 Spark ML LDA主题聚类算法

目录

- LDA主题模型算法
- ML LDA模型参数详解
- ML实例

目录

LDA

LDA主题模型算法

- LDA由David M. Blei，Andrew Y. Ng，Michael I. Jordan于2003年提出，是一种主题模型，它可以将文档集中每篇文档的主题以概率分布的形式给出。同时，它是一种典型的词袋模型，即一篇文档是由一组词构成，词与词之间没有先后顺序的关系。此外，一篇文档可以包含多个主题，文档中每一个词都由其中的一个主题生成。
- LDA的这三位作者在原始论文中给了一个简单的例子。比如假设事先给定了这几个主题：Arts、Budgets、Children、Education，然后通过学习的方式，获取每个主题对应的词语。

LDA主题模型算法

- 然后以一定的概率选取上述某个主题，再以一定的概率选取那个主题下的某个单词，不断的重复这两步，最终生成如下图所示的一篇文章。

"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

LDA概率统计基础

1. Gamma 函数

$\Gamma(x)$函数：

$$\Gamma(x)=\int_0^{\infty} t^{x-1}e^{-t}\mathrm{d}t$$

通过分部积分的方法，可以推导出这个函数有如下的递归性质：

$$\Gamma(x+1)=x\Gamma(x)$$

于是很容易证明，$\Gamma(x)$函数可以当成是阶乘在实数集上的延拓，具有如下性质：

$$\Gamma(n)=(n-1)!$$

LDA概率统计基础

2. 二项分布（binomial distribution）

二项分布是从伯努利分布推进的。伯努利分布，又称两点分布或 0-1 分布，是一个离散型的随机分布，其中的随机变量只有两类取值，非正即负{+，−}。而二项分布即重复 n 次的伯努利实验，记为 $X \sim b(n, p)$。简言之，只做一次实验，是伯努利分布；重复做了 n 次，就是二项分布。二项分布的概率密度函数为：

$$p(K = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

LDA概率统计基础

3. 多项分布，是二项分布扩展到多维的情况

多项分布是指单次实验中的随机变量的取值不再是0-1的，而有多种离散值可能（1，2，3，…，k）。比如投掷6个面的骰子实验，N次实验结果服从K=6的多项分布。其中：

$$\sum_{i=1}^{k} p_i = 1, \quad p_i > 0$$

多项分布的概率密度函数为：

$$p(x_1, x_2, \ldots, x_k; n, p_1, p_2, \ldots, p_k) = \frac{n!}{x_1! \ldots x_k!} p_1^{x1} \ldots p_k^{xk}$$

LDA概率统计基础

4. Beta 分布，是二项分布的共轭先验分布

给定参数 $\alpha > 0$ 和 $\beta > 0$，取值范围为[0,1]的随机变量 x 的概率密度函数：

$$f(x;\alpha,\beta) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

其中：

$$\frac{1}{B(\alpha,\beta)} = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)}$$

LDA概率统计基础

5. Dirichlet 分布，是 Beta 分布在高维度上的推广

Dirichlet 分布的密度函数形式跟 Beta 分布的密度函数如出一辙：

$$f(x_1, x_2, ..., x_k; \alpha_1, \alpha_2, ..., \alpha_k) = \frac{1}{B(\alpha)} \prod_{i=1}^{k} x_i^{\alpha^i - 1}$$

其中：

$$B(\alpha) = \frac{\prod_{i=1}^{k} \Gamma(\alpha^i)}{\Gamma(\sum_{i=1}^{k} \alpha^i)}, \quad \sum x_i = 1$$

至此，我们可以看到二项分布和多项分布很相似，Beta 分布和 Dirichlet 分布很相似，Beta 分布是二项分布的共轭先验概率分布，而狄利克雷分布（Dirichlet 分布）是多项分布的共轭先验概率分布。

LDA数学模型

1. 一元模型

对于文档 $w=(w_1,w_2,...,w_N)$，用 $p(w_n)$ 表示词 w_n 的先验概率，生成文档 w 的概率为：

$$p(w)=\prod_{n-1}^{N}p(w_n)$$

其图模型（图中被涂色的 w 表示可观测变量，N 表示一篇文档中总共 N 个单词，M 表示 M 篇文档）如图 11-3 所示。

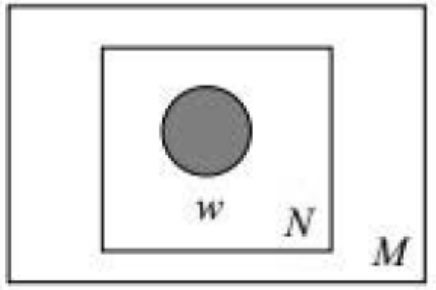


图 11-3　一元模型

一元模型假设文本中的词服从多项分布，而我们已经知道多项分布的先验分布为 Dirichlet 分布。

LDA数学模型

2. 混合一元模型

该模型的生成过程是：给某个文档先选择一个主题 z，再根据该主题生成文档，该文档中的所有词都来自一个主题。假设主题有 $z_1,\ldots,z_k$，生成文档 w 的概率为：

$$p(w)=p(z_1)\prod_{n=1}^{N}p(w_n\mid z_1)+\ldots+p(z_k)\prod_{n=1}^{N}p(w_n\mid z_k)=\sum_{z}p(z)\prod_{n=1}^{N}p(w_n\mid z)$$

其图模型（图中被涂色的 w 表示可观测变量，未被涂色的 z 表示未知的隐变量，N 表示一篇文档中总共 N 个单词，M 表示 M 篇文档）如图 11-4 所示。

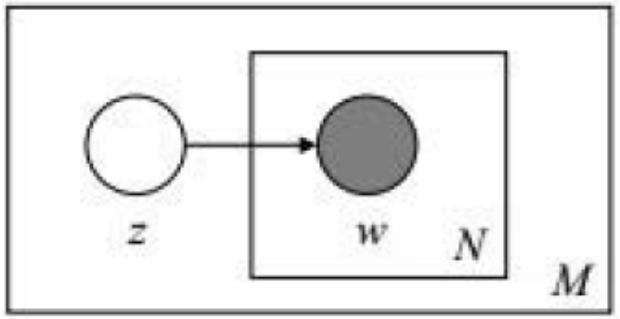


图 11-4　混合一元模型

LDA数学模型

3. PLSA 模型

在混合一元模型中，我们假定一篇文档只由一个主题生成。可实际中，一篇文章往往有多个主题，只是这多个主题各自在文档中出现的概率大小不一样。按照如下步骤得到“文档-词项”的生成模型：

1）按照概率 $p(d_i)$ 选择一篇文档 d_i 。

2）选定文档 d_i 后，从主题分布中按照概率 $p(z_k \mid d_i)$ 选择一个隐含的主题类别 z_k 。

3）选定 z_k 后，从词分布中按照概率 $p(w_j \mid z_k)$ 选择一个词 w_j 。

LDA数学模型

所以，PLSA 中生成文档的整个过程便是选定文档生成主题，确定主题生成词。文档 d 和单词 w 自然是可被观察到的，但主题 z 却是隐藏的，如图 11-5 所示（图中被涂色的 d、w 表示可观测变量，未被涂色的 z 表示未知的隐变量，N 表示一篇文档中总共 N 个单词，M 表示 M 篇文档）。

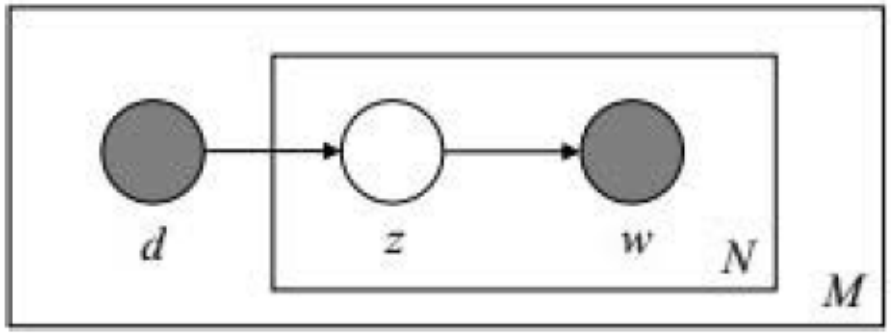


图 11-5　PLSA 模型

LDA数学模型

在图 11-5 中，文档 d 和词 w 是我们得到的样本，可观测得到，所以对于任意一篇文档，其 $p(w_j \mid d_i)$ 是已知的。

从而可以根据大量已知的文档-词项信息 $p(w_j \mid d_i)$，训练出文档-主题 $p(z_k \mid d_i)$ 和主题-词项 $p(w_j \mid z_k)$，如下公式所示：

$$p(w_j \mid d_i) = \sum_{k=1}^{K} p(w_j \mid z_k) p(z_k \mid d_i)$$

故得到文档中每个词的生成概率为：

$$\begin{aligned} p(d_i, w_j) &= p(w_j \mid d_i) \\ &= p(d_i) \sum_{k=1}^{K} p(w_j \mid z_k) p(z_k \mid d_i) \end{aligned}$$

4. LDA 模型

LDA 就是在 PLSA 的基础上加层贝叶斯框架，即 LDA 就是 PLSA 的贝叶斯版本。LDA 在 PLSA 的基础上，为主题分布和词分布分别加了两个 Dirichlet 先验。

LDA 模型中一篇文档生成的方式是：

1）按照先验概率 $p(d_i)$ 选择一篇文档 d_i。

2）从狄利克雷分布（即 Dirichlet 分布）α 中取样生成文档 d_i 的主题分布 θ_i。换言之，主题分布 θ_i 由超参数为 α 的 Dirichlet 分布生成。

3）从主题的多项分布 θ_i 中取样生成文档 d_i 第 j 个词的主题 $z_{i,j}$。

4）从狄利克雷分布（即 Dirichlet 分布）β 中取样生成主题 $z_{i,j}$ 对应的词语分布 $\varnothing_{z_{i,j}}$。换言之，词语分布 $\varnothing_{z_{i,j}}$ 由参数为 β 的 Dirichlet 分布生成。

5）从词语的多项分布 $\varnothing_{z_{i,j}}$ 中采样，最终生成词语 $w_{i,j}$。

LDA数学模型

- LDA的概率模型图模型（其中，阴影圆圈表示可观测的变量，非阴影圆圈表示隐变量，箭头表示两变量间的条件依赖性，方框表示重复抽样，方框右下角的数字代表重复抽样的次数）：

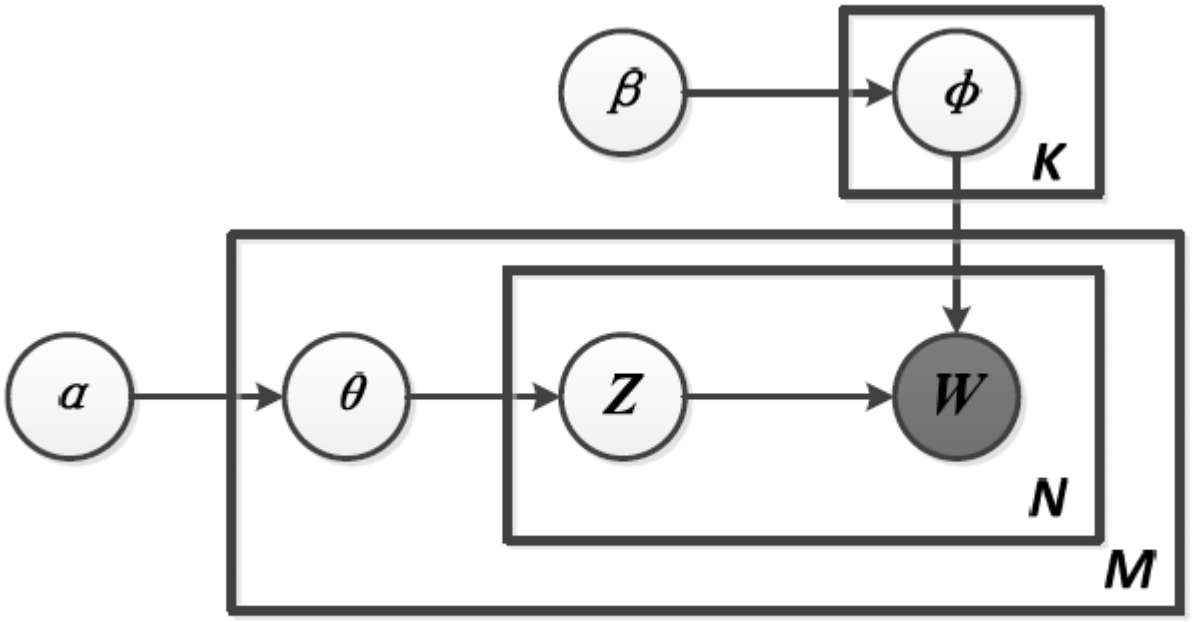


图 11-6　LDA 模型

LDA数学模型

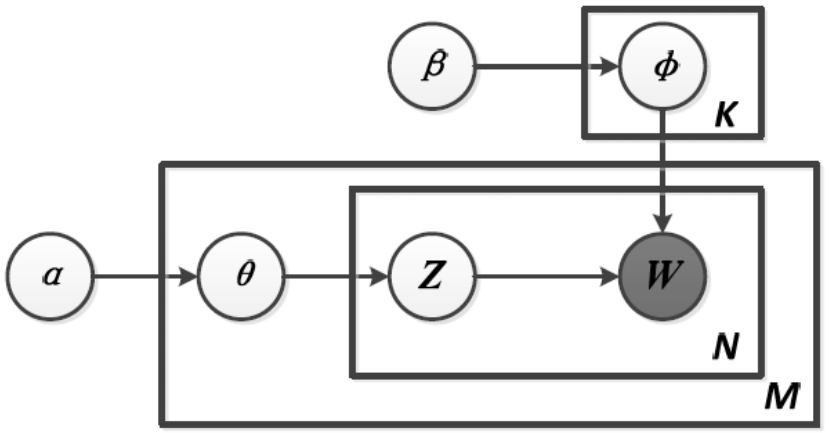


图 11-6　LDA 模型

对于图 11-6 的 LDA，只有 W 是观察到的变量，其他都是隐变量或者参数。其中，ϕ 表示词分布，θ 表示主题分布，α 是主题分布 θ 的先验分布（即 Dirichlet 分布）的参数，β 是词分布 θ 的先验分布（即 Dirichlet 分布）的参数，N 表示文档的单词总数，M 表示文档的总数。

所以，对于一篇文档 d 中的每一个单词，LDA 根据先验知识 α 确定某篇文档的主题分布 θ，然后从该文档所对应的多项分布（主题分布）θ 中抽取一个主题 z，接着根据先验知识 β 确定当前主题的词语分布 ϕ，随后从主题 z 所对应的多项分布（词分布）ϕ 中抽取一个单词 w。之后将这个过程重复 N 次，就产生了文档 d。

LDA数学模型

- **EM算法**
- LDA采用EM方法的步骤如下：
- (1) 给矩阵wk和kj随机赋值，其中wk是每个主题中每个单词出现的次数，kj是每个文档中每个主题出现的次数，虽然这些次数还只是随机数，我们还是可以根据这些次数，利用Dirichlet分布计算出每个主题中每个单词最可能出现的概率，以及每个文档中每个主题最可能出现的概率，也就是给（文档，主题）和（主题，词语）矩阵初始化；

- (2) 对于文档中的一个单词，计算出是由哪个主题产生的，因为可能有多个主题都会产生这个单词，那么它到底是属于哪个主题呢？这时就要用到极大似然估计了。计算出每个主题产生这个单词的概率：

$$p(z|w,d) = p(w|z) * p(z|d)$$

然后找出概率最大的那个主题，认为这个单词就是这个主题产生的，这在**EM方法中属于E-STEP**；

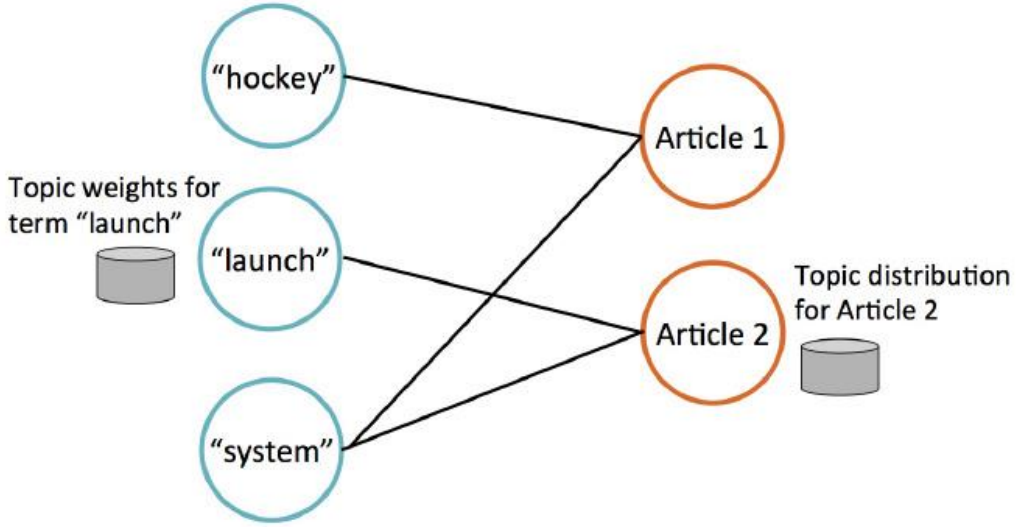
- (3) 由于确定了这个单词是哪个主题产生的，相当于Dirichlet分布中a的值发生了改变，于是计算出新的概率矩阵（（文档，主题）和（主题，词语）矩阵），这在**EM方法中属于M-STEP**。
- 重复步骤(2)和(3)，就可以得到最终的概率矩阵（（文档，主题）和（主题，词语）矩阵）。

LDA数学模型

采用 EM 算法训练 LDA 模型，因为它简单并且快速收敛，并且 EM 训练 LDA 有一个潜在的图结构，所以采用 GraphX 构建 LDA 模型。

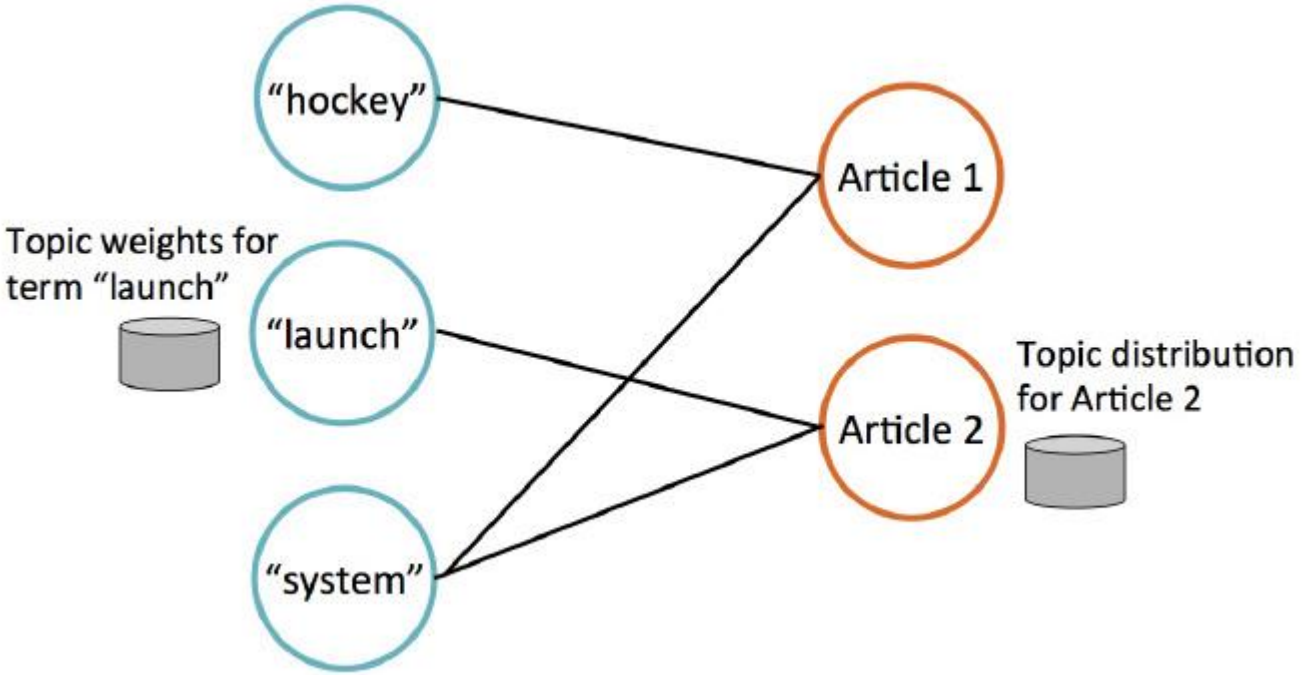
1. GraphX 构建 LDA 模型

LDA 主要有两类数据：词和文档。我们把这些数据存成一个对偶图（如图 11-7 所示），左边是词节点，右边是文档节点。每个词节点存储一些权重值，表示这个词语和哪个主题相关；类似地，每篇文章节点存储当前文章讨论主题的估计。



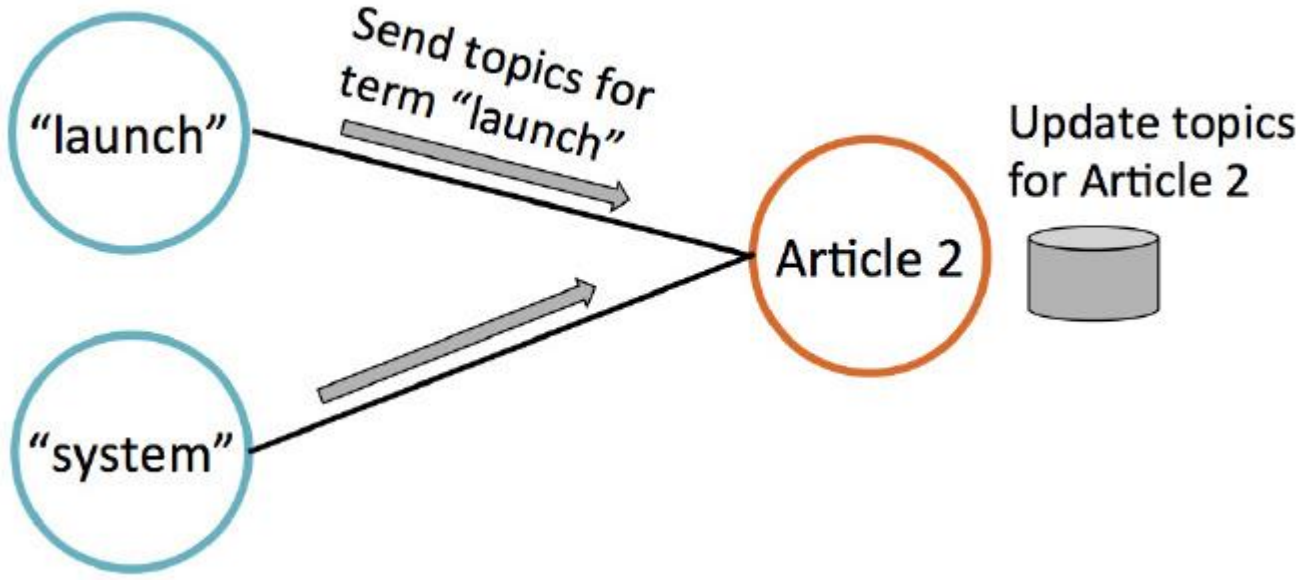
LDA数学模型

- 每当一个词出现在一篇文章中，图中就有一个边连接对应的词节点和文章节点。例如，在图中，文章1包含词语“hockey” 和“system”

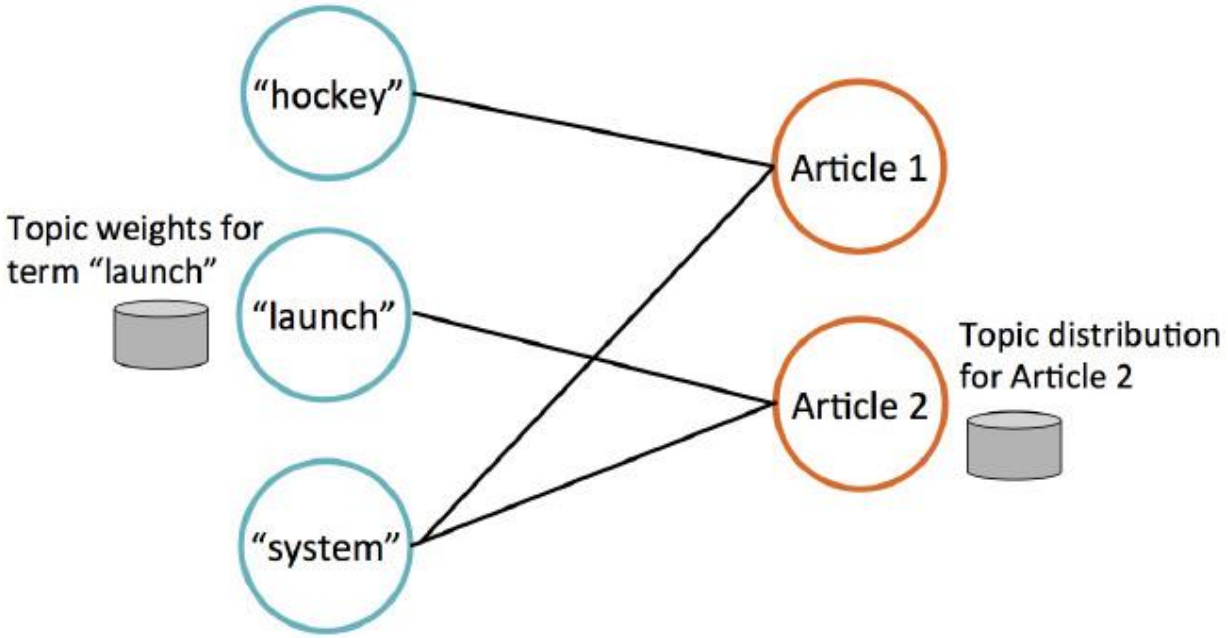


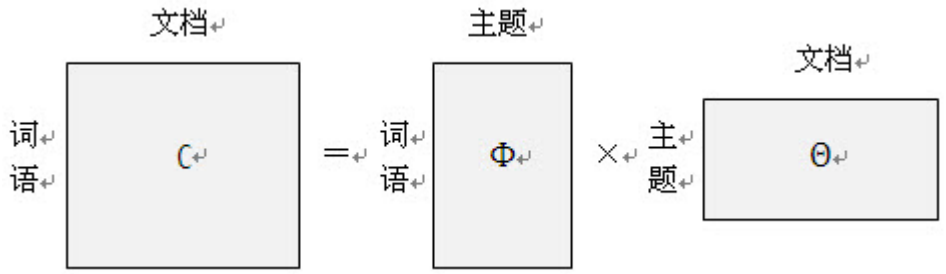
LDA数学模型

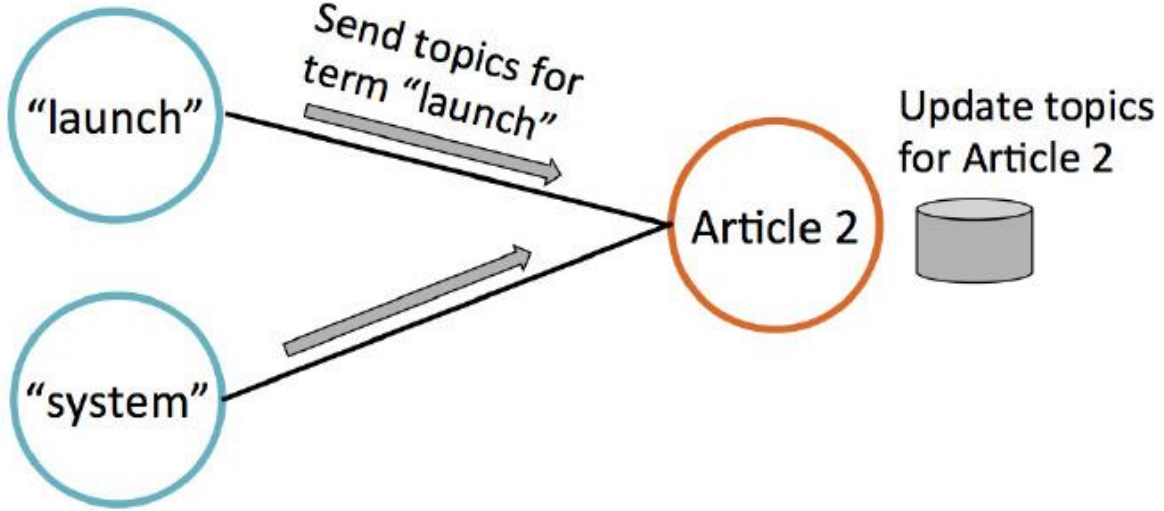
- 这些边也展示了这个算法的流通性。每轮迭代中，每个节点通过收集邻居数据来更新主题权重数据。下图中，文章2通过从连接的词节点收集数据来更新它的主题估计。



LDA数学模型


"hockey"
"launch"
"system"
Article 1
Article 2
Topic weights for term "launch"
Topic distribution for Article 2


文档
主题
文档
词
语
C
=
词
语
Φ
×
主
题
Θ


"launch"
Send topics for term "launch"
Article 2
Update topics for Article 2
"system"

目录

ML 参数讲解

ML 参数 LDA

ATAGURU 炼数成金

Parameters

A list of (hyper-)parameter keys this algorithm can take. Users can set and get the parameter values through setters and getters, respectively.

final val **checkpointInterval**: IntParam
Param for set checkpoint interval (>= 1) or disable checkpoint (-1).

final val **docConcentration**: DoubleArrayParam
Concentration parameter (commonly named "alpha") for the prior placed on documents' distributions over topics ("theta").

final val **featuresCol**: Param[String]
Param for features column name.

final val **k**: IntParam
Param for the number of topics (clusters) to infer.

final val **maxIter**: IntParam
Param for maximum number of iterations (>= 0).

final val **optimizer**: Param[String]
Optimizer or inference algorithm used to estimate the LDA model.

final val **seed**: LongParam
Param for random seed.

final val **subsamplingRate**: DoubleParam
For Online optimizer only: optimizer = "online".

final val **topicConcentration**: DoubleParam
Concentration parameter (commonly named "beta" or "eta") for the prior placed on topics' distributions over terms.

final val **topicDistributionCol**: Param[String]
Output column with estimates of the topic mixture distribution for each document (often called "theta" in the literature).

ML 参数

```
class LDA private (
  private var k: Int,
  private var maxIterations: Int,
  private var docConcentration: Double,
  private var topicConcentration: Double,
  private var seed: Long,
  private var checkpointInterval: Int,
  private var ldaOptimizer: LDAOptimizer) extends Logging {
  /**
   * k：主题数量
   * maxIterations：迭代次数
   * docConcentration：超参 alpha
   * topicConcentration：超参 beta
   * seed：随机种子
   * checkpointInterval：检查间隔
   * ldaOptimizer：优化方法 "em" "online"
   *
   */
```

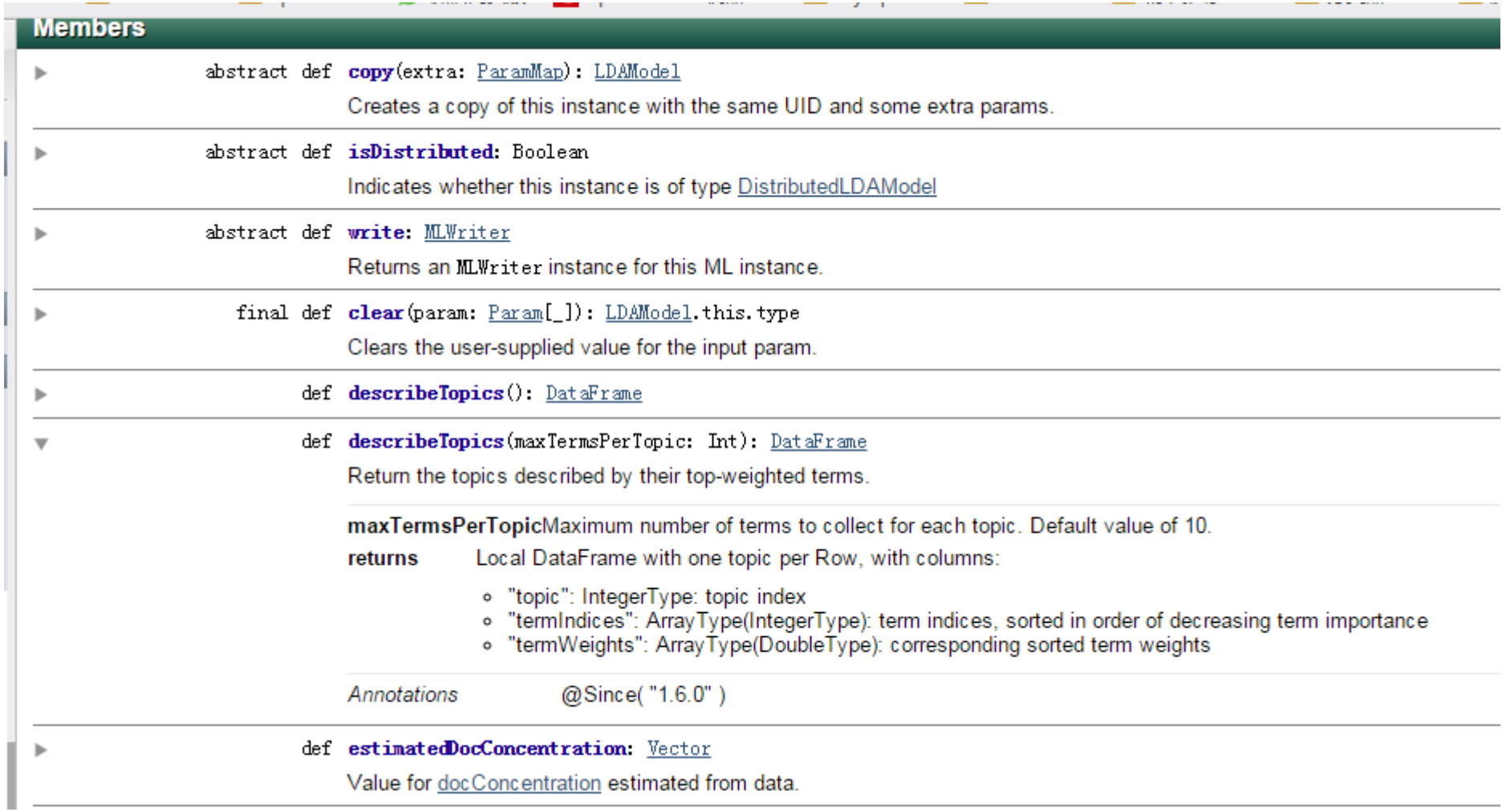
```
def getDocConcentration: Double = this.docConcentration

/**
 * 超参 alpha
 *
 * 这是一个对称的 Dirichlet 分布的参数，值越大意味着越平滑（更正规化）
 *
 * 如果设置为-1，程序会自动设置
 *  (default = -1 = automatic)
 *
 * 对于特定优化方法的参数设置：
 *     - EM
 *     - 必须  > 1.0
 *     - 默认 = (50 / k) + 1。其中 50/k 是在 LDA 中的通用设置，+1 是根据《On smoothing and inference
 *       for topic models》文中的推荐值
 *     - Online
 *     - 必须   >= 0
 *     - 默认 = (1.0 / k)，参照以下文档：
 *       [[https://github.com/Blei-Lab/onlineldavb]]
```

```
def getTopicConcentration: Double = this.topicConcentration

/**
 * 超参 beta
 *
 * 这是一个对称的 Dirichlet 分布的参数
 *
 * 注意：在 LDA 的原始论文中，对于主题的词分布通常叫 beta，
 * 但是在许多后来的论文中叫 phi，比如在《On smoothing and inference for topic models》论文中
 *
 * 如果设置为 -1，程序会自动设置
 *
 * 对于特定优化方法的参数设置：
 *     - EM
 *     - 必须  > 1.0
 *     - 默认 = (50 / k) + 1。其中 50/k 是在 LDA 中的通用设置，+1 是根据《On smoothing and inference
 *       for topic models》文中的推荐值
 *     - Online
 *     - 必须   >= 0
 *     - 默认 = (1.0 / k)，参照以下文档：[[https://github.com/Blei-Lab/onlineldavb]]
```

ML 参数 LDAModel



Members

abstract def **copy**(extra: ParamMap): LDAModel

Creates a copy of this instance with the same UID and some extra params.

abstract def **isDistributed**: Boolean

Indicates whether this instance is of type DistributedLDAModel

abstract def **write**: MLWriter

Returns an MLWriter instance for this ML instance.

final def **clear**(param: Param[_]): LDAModel.this.type

Clears the user-supplied value for the input param.

def **describeTopics**(): DataFrame

def **describeTopics**(maxTermsPerTopic: Int): DataFrame

Return the topics described by their top-weighted terms.

maxTermsPerTopic Maximum number of terms to collect for each topic. Default value of 10.

returns Local DataFrame with one topic per Row, with columns:

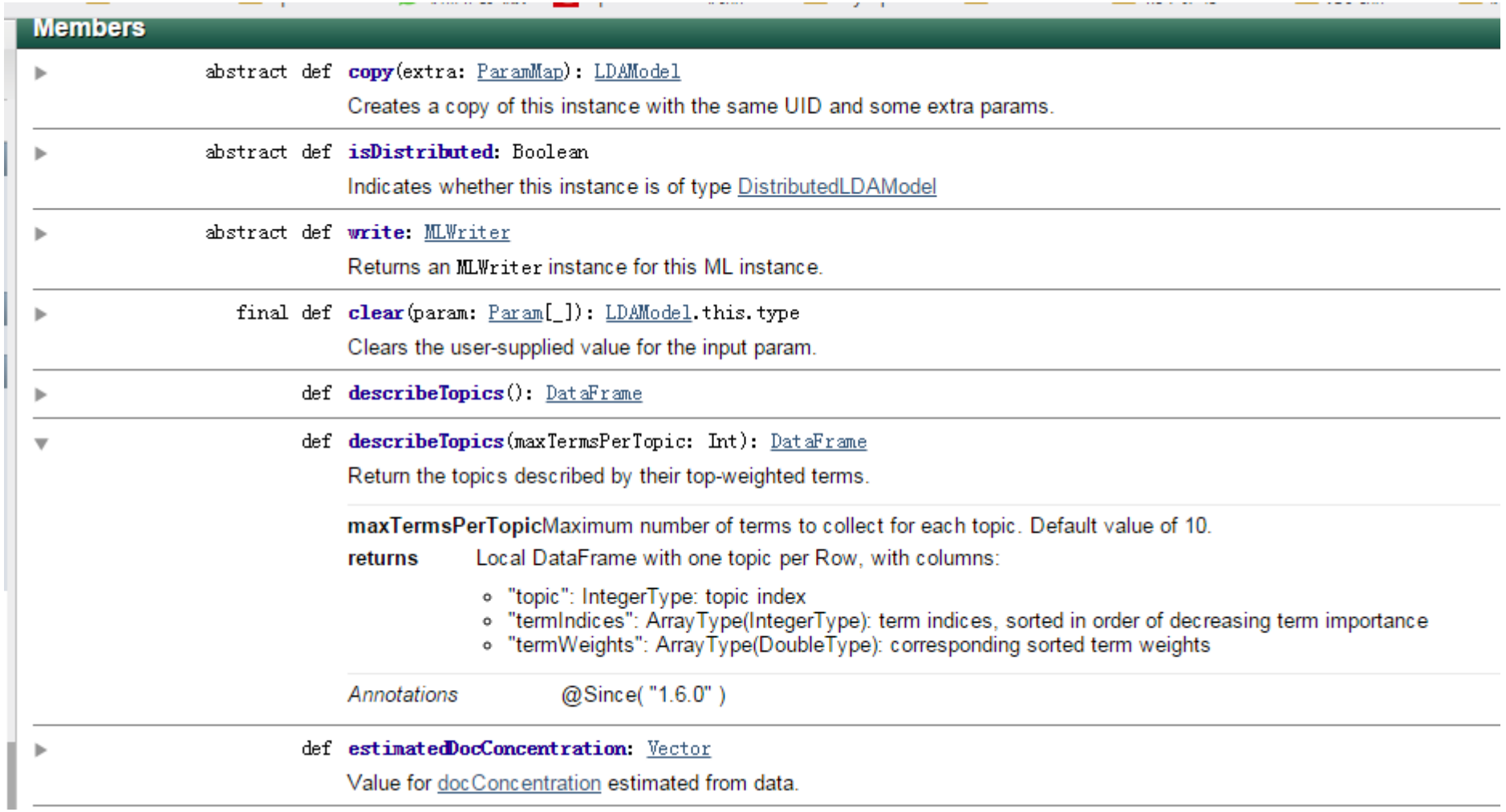
- "topic": IntegerType: topic index
- "termIndices": ArrayType(IntegerType): term indices, sorted in order of decreasing term importance
- "termWeights": ArrayType(DoubleType): corresponding sorted term weights

Annotations @Since("1.6.0")

def **estimatedDocConcentration**: Vector

Value for docConcentration estimated from data.

ML 参数 LDAModel



Members

abstract def **copy**(extra: ParamMap): LDAModel

Creates a copy of this instance with the same UID and some extra params.

abstract def **isDistributed**: Boolean

Indicates whether this instance is of type DistributedLDAModel

abstract def **write**: MLWriter

Returns an MLWriter instance for this ML instance.

final def **clear**(param: Param[_]): LDAModel.this.type

Clears the user-supplied value for the input param.

def **describeTopics**(): DataFrame

def **describeTopics**(maxTermsPerTopic: Int): DataFrame

Return the topics described by their top-weighted terms.

maxTermsPerTopic Maximum number of terms to collect for each topic. Default value of 10.

returns Local DataFrame with one topic per Row, with columns:

- "topic": IntegerType: topic index
- "termIndices": ArrayType(IntegerType): term indices, sorted in order of decreasing term importance
- "termWeights": ArrayType(DoubleType): corresponding sorted term weights

Annotations @Since("1.6.0")

def **estimatedDocConcentration**: Vector

Value for docConcentration estimated from data.

ML 参数 LDAModel

Sets the parent of this model (Java API).

def **setTopicDistributionCol**(value: String): LDAModel.this.type

final val **supportedOptimizers**: Array[String]

Supported values for Param optimizer.

def **toString**(): String

def **topicsMatrix**: Matrix

Inferred topics, where each topic is represented by a distribution over terms. This is a matrix of size vocabSize x k, where e

WARNING: If this model is actually a DistributedLDAModel instance produced by the Expectation-Maximization ("em") optir vocabSize x k).

Annotations @Since("2.0.0")

def **transform**(dataset: Dataset[_]): DataFrame

Transforms the input dataset.

def **transform**(dataset: Dataset[_], paramMap: ParamMap): DataFrame

Transforms the dataset with provided parameter map as additional parameters.

def **transform**(dataset: Dataset[_], firstParamPair: ParamPair[_], otherParamPairs: ParamPair[_]*): DataFrame

Transforms the dataset with optional parameters

def **transformSchema**(schema: StructType): StructType

val **uid**: String

An immutable unique ID for the object and its derivatives.

val **vocabSize**: Int

ML 参数 LDAModel

11.3.3 LDA 模型

训练完成后，生成 LDA 主题模型，LDA 主题模型包括 Local 模式的 LDA 模型及分布式的 LDA 模型。LDA 主题模型包含方法：主题分布、主题有序排序、文章分布等。

LDAModel 的源码解析如下。

```
/**
 * :: Experimental ::
 *
 * 隐含 Dirichlet 分布（LDA）模型
 *
 * 包含本地和分布式模型
 */
@Experimental
abstract class LDAModel private[clustering] {
```

ML 参数 LDAModel

```
/**
 * 主题分布矩阵
 */
def topicsMatrix: Matrix

/**
 * 每个主题的词权重排序
 * @param maxTermsPerTopic  每个主题的最大词汇数量
 * @return  数组形式返回分布结果，是按照词权重递减排序的主题数组的词权重
*/
def describeTopics(maxTermsPerTopic: Int): Array[(Array[Int], Array[Double])]
```

ML 参数 LDAModel

```
/**
 * :: Experimental ::
 *
 * Local 模式的 LDA 模型
 * 该模式仅存储主题分布矩阵
 * 它可用于计算新文档的主题
 *
 * @param topics 主题分布矩阵（矩阵大小为 vocabSize×k）
 */
@Experimental
class LocalLDAModel private[clustering] (
    private val topics: Matrix) extends LDAModel with Serializable {
```

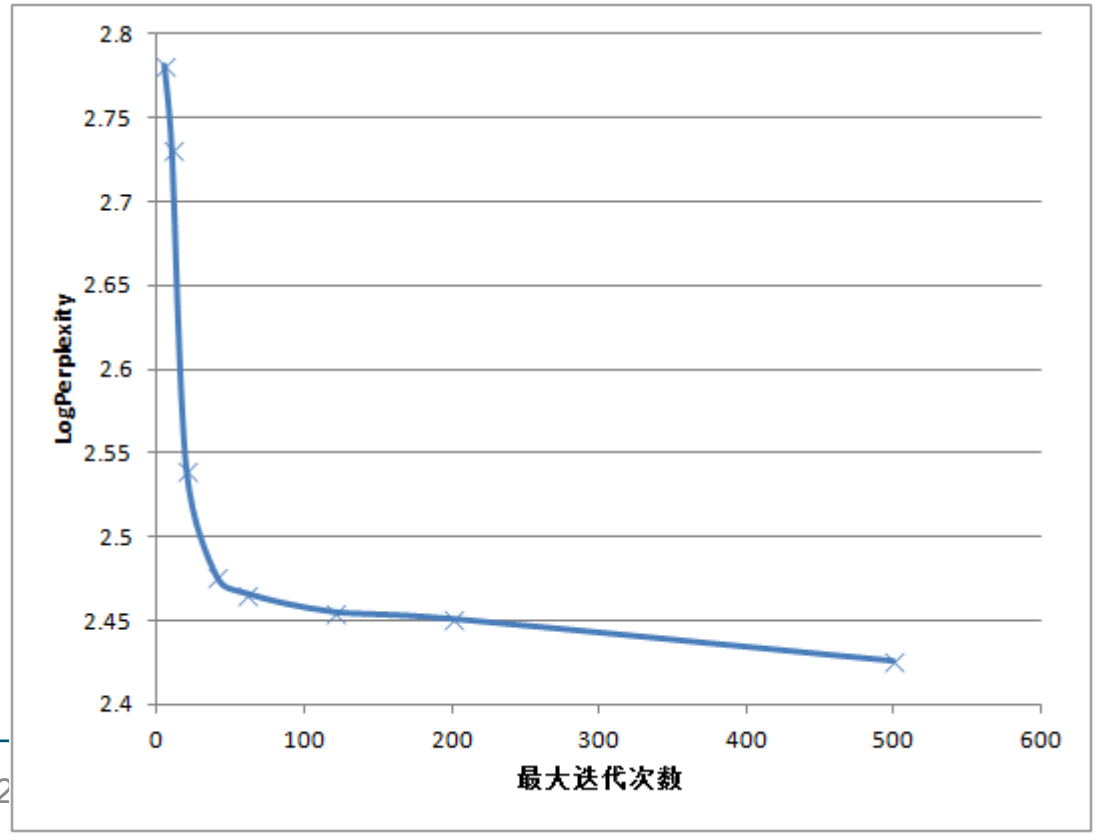
ML 参数 LDAModel

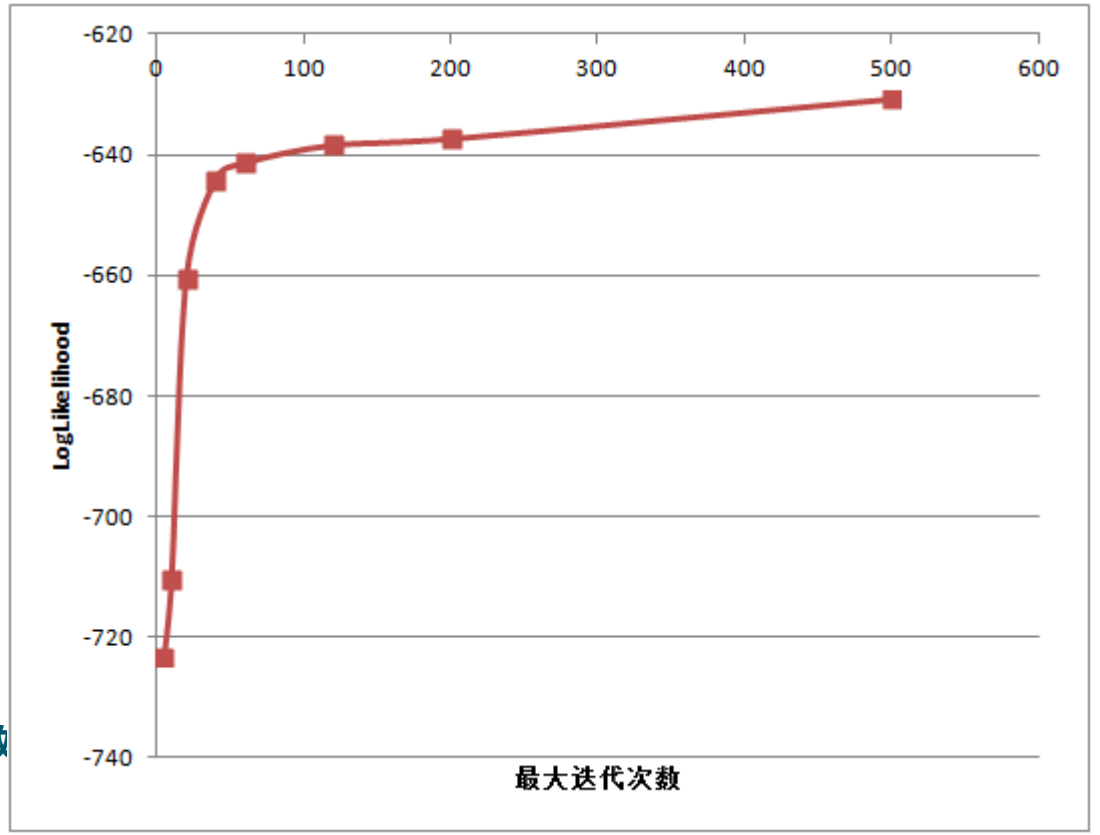
```
/**
 * :: Experimental :: *
 * 分布式的 LDA 模型
 * 该模型存储主题推断过程、全部训练数据集、主题分布
 */
@Experimental
class DistributedLDAModel private (
    private val graph: Graph[LDA.TopicCounts, LDA.TokenCount],
    private val globalTopicTotals: LDA.TopicCounts,
    val k: Int,
    val vocabSize: Int,
    private val docConcentration: Double,
    private val topicConcentration: Double,
    private[spark] val iterationTimes: Array[Double]) extends LDAModel {

  import LDA._
```

ML 参数

- Spark 1.4和1.5引入了一种增量式计算LDA的在线算法，在已训练的LDA模型上支持更多的查询方式，以及支持似然率（likelihood）和复杂度（perplexity）的性能评估。
- https://en.wikipedia.org/wiki/Likelihood_function
- https://en.wikipedia.org/wiki/Perplexity





ML 参数

```
/**
 * 对数似然: log P(docs | topics, topic distributions for docs, alpha, eta)
 */
lazy val logLikelihood: Double = {
  val eta = topicConcentration
  val alpha = docConcentration
  assert(eta > 1.0)
  assert(alpha > 1.0)
  val N_k = globalTopicTotals
  val smoothed_N_k: TopicCounts = N_k + (vocabSize * (eta - 1.0))
  // 边: 根据 phi_{wk}和 theta_{kj}计算词汇的对数概率
```

ML 参数

```
/**
 * 对数概率：log P(topics, topic distributions for docs | alpha, eta)
 */
lazy val logPrior: Double = {
  val eta = topicConcentration
  val alpha = docConcentration
  // 词顶点：计算 phi_{wk}。用于计算先验概率
  // 文章顶点：计算 theta_{kj}。用于计算先验概率
  val N_k = globalTopicTotals
```

目录

ML 实例

实例代码

```
import org.apache.spark.ml.clustering.{LDA,LDAModel}
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.ml.clustering.{LDA, LDAModel}
import org.apache.spark.sql.SparkSession
```

实例代码

```
// 读取样本
val dataset = spark.read.format("libsvm").load("hdfs://[redacted]/sample_lda_libsvm_data.txt")
dataset.show()
```

```
dataset: org.apache.spark.sql.DataFrame = [label: double, features: vector]
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(11,[0,1,2,4,5,6,...|
|  1.0|(11,[0,1,3,4,7,10...|
|  2.0|(11,[0,1,2,5,6,8,...|
|  3.0|(11,[0,1,3,6,8,9,...|
|  4.0|(11,[0,1,2,3,4,6,...|
|  5.0|(11,[0,1,3,4,5,6,...|
|  6.0|(11,[0,1,3,6,8,9,...|
|  7.0|(11,[0,1,2,3,4,5,...|
|  8.0|(11,[0,1,3,4,5,6,...|
|  9.0|(11,[0,1,2,4,6,8,...|
| 10.0|(11,[0,1,2,3,5,6,...|
| 11.0|(11,[0,1,4,5,6,7,...|
+-----+--------------------+
```

实例代码

```
    // 训练 LDA model.
    val lda = new LDA().setK(10).setMaxIter(10)
    val model = lda.fit(dataset)

    val ll = model.logLikelihood(dataset)
    val lp = model.logPerplexity(dataset)
    println(s"The lower bound on the log likelihood of the entire corpus: $ll")
    println(s"The upper bound on perplexity: $lp")
```

```
lda: org.apache.spark.ml.clustering.LDA = lda_401297d452a3
model: org.apache.spark.ml.clustering.LDAModel = lda_401297d452a3
ll: Double = -841.0235930252268
lp: Double = 3.235480559832786
The lower bound on the log likelihood of the entire corpus: -841.0235930252268
The upper bound on perplexity: 3.235480559832786
```

实例代码

```
    // 主题 topics.
    val topics = model.describeTopics(5)
    println("The topics described by their top-weighted terms:")
    topics.show(false)
```

```
topics: org.apache.spark.sql.DataFrame = [topic: int, termIndices: array<int> ... 1 more field]
The topics described by their top-weighted terms:
+-----+---------------+-----------------------------------------------------------------------------------------------------------+
|topic|termIndices    |termWeights                                                                                                |
+-----+---------------+-----------------------------------------------------------------------------------------------------------+
|0    |[2, 5, 7, 9, 4] |[0.1060644085961625, 0.10570106168102907, 0.1043038961745402, 0.09677466095381614, 0.09348954966398486]  |
|1    |[1, 6, 2, 5, 0] |[0.10185076997486057, 0.09816928141860444, 0.0963245435405093, 0.09533709162757754, 0.09315597570586823] |
|2    |[10, 6, 9, 1, 3]|[0.21830191650160977, 0.1386443612759956, 0.1306310616141132, 0.12280252977822431, 0.08282333095356757]  |
|3    |[0, 4, 8, 5, 7] |[0.10270701955800988, 0.09842848153374084, 0.09815661242068288, 0.09625859107772357, 0.09534802346865436]|
|4    |[9, 6, 4, 0, 3] |[0.10452964425141797, 0.10414908184922392, 0.10103987040611716, 0.0965393332465656, 0.09381347318947432] |
|5    |[1, 10, 0, 6, 7]|[0.10214945363898671, 0.10129059978752218, 0.09513643669169457, 0.09484723290301932, 0.0916799216142901] |
|6    |[3, 7, 4, 5, 2] |[0.11638316687856809, 0.09901763170616922, 0.09795372072050275, 0.09538797685025302, 0.09049487777614593]|
|7    |[4, 0, 2, 7, 9] |[0.10855453653878992, 0.10334275138797878, 0.10034943368693225, 0.09586142922687427, 0.09376962699019654]|
|8    |[0, 7, 8, 9, 10]|[0.11008008210266895, 0.09919723498666952, 0.09810902425250166, 0.09598429155099947, 0.0954932936083263] |
|9    |[9, 6, 8, 7, 2] |[0.10106110088567796, 0.10013295831202408, 0.09769277852783784, 0.09637374364924176, 0.09624837356556025]|
+-----+---------------+-----------------------------------------------------------------------------------------------------------+
```

实例代码

ATAGURU 炼数成金

```
    // 测试结果.
    val transformed = model.transform(dataset)
    transformed.show(false)
```

```
transformed: org.apache.spark.sql.DataFrame = [label: double, features: vector ... 1 more field]
+-----+-----------------------------------------------------------+-----------------------------------------------------------
-----------------------------------------------------------------------------------------+
|label|features                                                   |topicDistribution
                                                                                         |
+-----+-----------------------------------------------------------+-----------------------------------------------------------
-----------------------------------------------------------------------------------------+
|0.0  |(11,[0,1,2,4,5,6,7,10],[1.0,2.0,6.0,2.0,3.0,1.0,1.0,3.0])  |[0.7020410013143723,0.004825993248586696,0.259351294
84885498668,0.004825959112892307,0.004826029452700648,0.0048258990158972954,0.0048259405227369205]  |
|1.0  |(11,[0,1,3,4,7,10],[1.0,3.0,1.0,3.0,2.0,1.0])              |[0.00805122780720418,0.008050678396930818,0.927541143
966754822234,0.008051191076476666,0.008051175133853379,0.008051124921355369,0.008050833928097212]  |
|2.0  |(11,[0,1,2,5,6,8,9],[1.0,4.0,1.0,4.0,9.0,1.0,2.0])         |[0.00419616097037882,0.004196120649162783,0.962235777
59839684049685,0.004196034651063291,0.004196001104210915 5,0.004195859201288365,0.004196016086387109]  |
|3.0  |(11,[0,1,3,6,8,9,10],[2.0,1.0,3.0,5.0,2.0,3.0,9.0])        |[0.003710825536933 9804,0.003710870207561922,0.966602
3710895151237488,0.003710876911576858,0.003710841629857278,0.003710901056842 3407,0.003710891599618 6213]  |
|4.0  |(11,[0,1,2,3,4,6,9,10],[3.0,1.0,1.0,9.0,3.0,2.0,1.0,3.0])  |[0.0040206152028302985,0.00402065678327 41175,0.963813
4020673056295344,0.004020989502400588,0.004020656982046972,0.004020671772191288,0.004020615797176421]  |
|5.0  |(11,[0,1,3,4,5,6,7,8,9],[4.0,2.0,3.0,4.0,5.0,1.0,1.0,1.0,4.0]) |[0.0037113314568483487,0.0037113060146913746,0.424786
```

Thanks

FAQ时间