Lantent Dirichlet Allocation Report

PB19030861 王湘峰

实验要求

- 实现LDA模型，并且输出训练好的LDA模型中每个主题下概率最高的15个单词
- 不得直接调用开源库中的LDA

实验原理

隐狄利克雷分配模型（Latent Dirichlet Allocation, LDA）是话题模型（topic model）的典型代表。

LDA 模型的基本原理

LDA 的基本单元：

- 词（word）：待处理数据中的基本离散单元
- 文档（document）：待处理的数据对象，由词组成，词在文档中**不计顺序。**数据对象只要能用"词袋"（bag-of-words）表示就可以使用主题模型
- 话题（topic）：表示一个概念，具体表示为一系列相关的词，以及它们在该概念下出现的概率

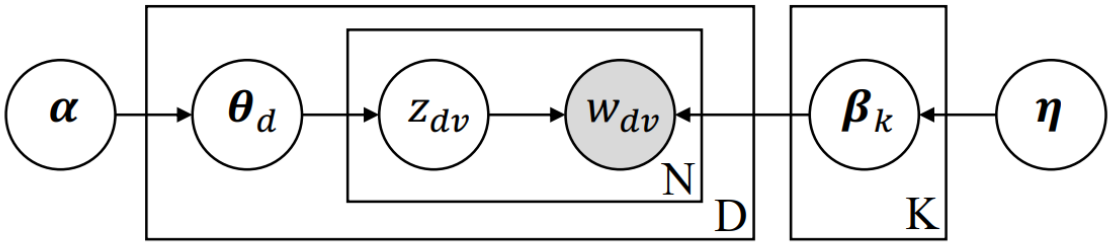
LDA 模型的描述：

记号约定：假定数据集中共含 K 个话题和 D 篇文档 $W=\{w_1 \dots w_D\}$，词来自 V 词的词典。每篇文档（第 i 篇文档 w_i) 用长度为 N_i 的的单词序列表示 $w_i=\{w_{i,1} \dots w_{i,N_i}\}$.

在LDA模型中，认为一篇文档是先给出主题，然后围绕这个主题遣词造句生成的。具体而言，一篇文档 w 是按下面的概率模型生成的：

- 确定每个主题中词的概率：从参数为 η 的**Dirichlet分布** $Dir(\eta)$ 中采样，生成主题 k 的词语分布 $\beta_k \in [0,1]^V$，表示属于主题 k 的词是词典中某个词的概率；
- 给出文档主题：从参数为 α 的**Dirichlet分布** $Dir(\alpha)$ 中采样，得到某篇文档的主题分布 $\theta \in [0,1]^K$，表示文档中每个词属于每个主题的概率；
- 从**多项式分布** $P(1;\theta)$ 采样，指派文档中第 j 个词的主题 $z_j \in \{0,1\}^K$ 若 $z_j^k=1$，表示它属于主题 k 。这里虽然称为多项式分布，但实际上这个多项式分布的 $n=1$，每个词直接根据概率 θ 指派一个主题，所以我们也可以直接用 z_j 表示其所属的主题 $z_j \in \{1,\dots,K\}$
- 对于第 j 个词，根据指派的主题 z_j，从多项式分布 $P(1;\beta_{z_j})$ 中，最终采样生成词语 w_j .

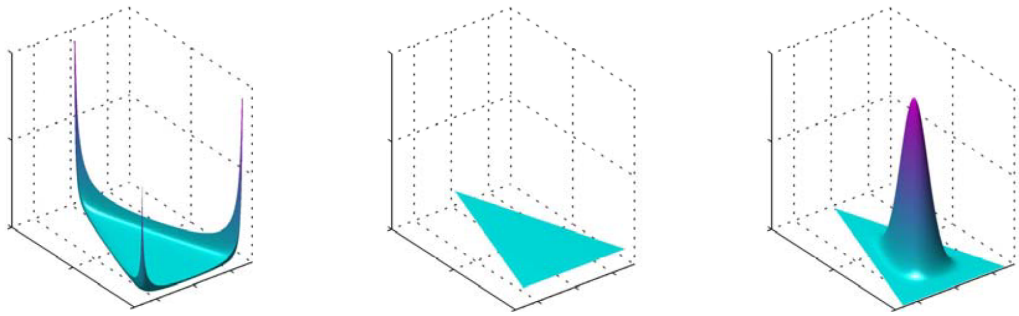
根据模型中文档的生成过程，变量的依赖关系如示意图。在这个图中，w 唯一的观测变量，而 η,α 是先验变量，我们需要求出的是隐变量 θ,β，代表着每篇文档的主题分布，每个主题的词频分布。



Dirichlet 分布是多项式分布的**共轭先验分布**，这两个分布模型在文档生成模型中的应用是非常自然的想法——因为后验分布和先验分布具有相同形式，只是参数有所不同，这意味着当我们获得新的观察数据时，我们就能直接通过更新参数，获得新的后验分布。参数为 α 的Dirichlet分布的密度函数为

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{K}\alpha_i)}{\Pi_{i=1}^{K}\Gamma(\alpha_i)}\theta_1^{\alpha_1-1}\dots\theta_k^{\alpha_k-1}$$

下图从左至右分别为 $\alpha = 0.1, \alpha = 1, \alpha = 10$ 的Dirichlet分布图像



参数 α, η 的求解

根据这个概率模型，可以写出隐变量 z, θ, β 和观测变量 w 的联合分布密度

$$p(\mathbf{w}, \mathbf{z}, \theta, \beta \mid \alpha, \eta) = p(\theta \mid \alpha)p(\beta \mid \eta)\prod_{j=1}^{N} p\left(z_j \mid \theta\right)p\left(w_j \mid \beta_{z_j}\right)$$

这里我们知道的是文档样本 $\mathbf{w}$ ，我们的目的是估计出参数 α 和 η ，对隐变量 θ 和 β 积分，对 z 求和可以得到边缘概率 $p(\mathbf{w}|\alpha, \eta)$，通过对$p(\mathbf{w}|\alpha, \eta)$ 的极大似然估计可以求出 α 和 η 。但实际上由于 $p(\mathbf{w}|\alpha, \eta)$ 不易计算，所以难以直接求解极大似然估计，实践中常用变分推断求近似解。

参数 α, η 的先验给出

在本次实验中，α, η 不是通过估计得出的，而是**直接给定**一个先验值，这样做法简单并且同样能取得不错的实验效果。根据我们对于一般情况一篇具有主题文章的常识，它会对某一些词语如助词、介词等必用的但无意义的词语、主题相关的词语具有明显的偏好，而与主题无关的生僻词则几乎不会出现。在Dirichlet话题模型，这个向量是一个主题的词频。根据常识，主题的词频一般都有明显偏好。再观察上面的Dirichlet分布的形状，可以看出，当alpha大于1时，Dirichlet分布中更倾向于产生更均匀的向量；当alpha小于1时，Dirichlet分布更倾向于产生个别值较大的向量。所以我们可以大致推测出各个话题词频的分布，应该具有以下特征：

- 话题总是偏好一些对应话题词，而不相关的话题词不常提及：参数 η 可以取适当的小于1的值；各个文章同样也有明显的话题偏好，参数 α 的值也都小于1。

事实上（同种语言的）所有话题的都会共同偏好一些助词、介词等，参数 η 对应这些词一般更小（这些词的词频高的概率总是大），但这相对难以考查，并且我们在预处理时，会删除一些无意义的词汇，一定程度消除了这些词的影响。为了方便我们直接取对称Dirichlet的参数值，即 α, η 各个维度都相等。

隐变量 θ, β 的求解

若 α, η 已确定，可以通过吉布斯采样(Gibbs Sampling)方法估计出模型中的隐变量。

$$p(\mathbf{z}, \theta, \beta \mid \mathbf{W}, \alpha, \eta) = \frac{p(\mathbf{W}, \mathbf{z}, \theta, \beta \mid \alpha, \eta)}{p(\mathbf{W} \mid \alpha, \eta)}$$

由于分母上的 $p(\mathbf{W} \mid \alpha, \eta)$ 难以求得，实践中常用吉布斯采样或变分推断求得。本实验采用Gibbs采样求解

吉布斯采样算法思想:

- 对隐变量 θ, φ积分，得到边缘概率 $p(Z \mid W, \alpha, \beta)$;
- 对后验概率进行吉布斯抽样，得到分布 $p(Z \mid W, \alpha, \beta)$ 的样本集合;
- 利用这个样本集合对参数 θ 和 φ 进行参数估计.
- 对隐变量 θ, φ 积分，得到边缘概率 $p(Z \mid W, \alpha, \beta)$;

  $$p(Z \mid W, \alpha, \beta) \propto \prod_{m=1}^{M} \frac{\mathrm{B}(\alpha + \sigma_m)}{\mathrm{B}(\alpha)} \prod_{k=1}^{K} \frac{\mathrm{B}(\beta + \delta_k)}{\mathrm{B}(\beta)}.$$

  其中 $\mathrm{B}(\cdot)$ 为 Beta 函数，$\sigma_{mk} = \sum_{n=1}^{N_m} \mathbb{I}(z_{mn} = k)$ 表示第 m 篇文档中第 k 个话题的词的频数，$n_{kv} = \sum_{m=1}^{M} \sum_{n=1}^{N_m} \mathbb{I}(w_{mn} = v)\mathbb{I}(z_{mn} = k)$ 表示所有文档中第 k 个话题下词 w_v 出现的频数。
- 对后验概率进行吉布斯抽样，得到分布 $p(Z \mid W, \alpha, \beta)$ 的样本集合;

吉布斯采样的过程如下:

- 首先遍历所有文档中的所有词，为其各随机分配一个主题 $z_{d,j} \in \{1 \ldots K\}$ ，表示第 d 篇文档中的第 j 个词属于主题 $z_{d,j}$;并统计和维护变量 $n_d^{(k)}, m_k^{(t)}$ 其中 $n_d^{(k)}$ 表示文档 d 中主题 k 出现的次数；$m_k^{(t)}$ 表示词典中词 t 在主题 k 中出现的次数.
- 重复以下迭代：
  - 对所有文档的所有词遍历。对于属于文档 d 中的词 t ,取出该词根据LDA中主题的概率分布采样出新主题

    $$p\left(z_{d,j} = k \mid \mathbf{Z}_{\neg d,j}, \mathbf{W}, \alpha, \beta\right) \propto \frac{n_{d,k}^{\neg d,j} + \alpha_k}{\sum_{k=1}^{K} n_{d,k} + \alpha_k} \cdot \frac{m_{k,w_{d,j}}^{\neg d,j} + \beta_{w_{d,j}}}{\sum_{v=1}^{V} m_{k,v}^{-d,j} + \beta_v}$$

  - 迭代完成后，输出主题-词分布概率 β 和文档-主题分布概率 θ

    $$\beta_{k,t} \approx \frac{m_{k,t} + \eta_t}{\sum_{t=1}^{V} m_{k,t} + \eta_t}$$

    $$\theta_{d,k} \approx \frac{n_{d,k} + \alpha_k}{\sum_{k=1}^{K} n_{d,k} + \alpha_k}$$

    其中 m_k 表示所有属于主题 k 的词的个数，n_d 表示文档 d 的总词数。

核心代码讲解

导入必要的库

```
import numpy as np
import pandas as pd
import jieba
import jieba.posseg as psg
import re
from time import time
```

解释：numpy可以将矩阵运算变得很方便，且底层大部分用C实现，效率高；pandas可用于读入excel文件；jieba库用于将中文文本分割成一个个的词语；re是用于正则匹配，同样在分割中文时用到；time库用来计时。

全局函数word_cut

```
def word_cut(text, stopword_list):
    jieba.initialize()
    stop_list = []
    flag_list = ['n', 'nz', 'vn']
    for line in stopword_list:
        line = re.sub(u'\n|\\r', '', line)
        stop_list.append(line)

    word_list = []
    # jieba分词
    seg_list = psg.cut(text)
    for seg_word in seg_list:
        word = re.sub(u'[^\u4e00-\u9fa5]', '', seg_word.word)
        find = 0
        if word in stop_list or len(word) < 2:
            continue
        if seg_word.flag in flag_list:
            word_list.append(word)
    return word_list
```

解释：函数的输入为待切分的文本和一个停用词文件（名），输出为切分后的词的列表。切分的对象为动词、名词和动名词。

LDA类

初始化

```
class LDATopicModel:
    # 无意义标点
    meaningless_symbol_list = ['.', '$', '#', '[', ']', '(', ')', '|', '*', ':',
'=', '/',
                               '>', '<', '+', '{', '}', ',', '?', '&', '-', '@',
"'", '%', '^', ', ',
                               '。', '《', '》', ': ', '; ', '“', '”', '‘', '？']

    def __init__(self, raw_texts, k_topics):
        self.raw_data = raw_texts
        self.k_topics = k_topics
        self.beta = None
        self.eta = None
```

```
        self.theta = None
        self.data_set_mapped = None
        self.vocabulary_list = None
        self.vocabulary_idx = None
```

解释：无意义符号是用来将标点符号进行剔除的

预处理

```
def preprocess(self, drop_n_freq=30):
        num_docs = len(self.raw_data)
        all_words = list()
        raw_docs = list()
        try:
            stopword_list = open(stop_file, encoding='utf-8').read().split('\n')
        except:
            stopword_list = []

        for i in range(num_docs):
            doc_text = self.raw_data[i]
            for s in LDATopicModel.meaningless_symbol_list:
                doc_text = doc_text.replace(s, '')
            raw_docs.append(list())

            for word in word_cut(doc_text, stopword_list):
                if len(word) >= 2:
                    raw_docs[-1].append(word)
            all_words.extend(raw_docs[-1])

        unique_words = list(set(all_words))
        word_frequency = dict()
        for wd in unique_words:
            word_frequency[wd] = 0
        for wd in all_words:
            word_frequency[wd] += 1
        unique_words.sort(key=lambda wd: word_frequency[wd], reverse=True)
        self.vocabulary_list = unique_words[drop_n_freq:]

        self.vocabulary_idx = dict()
        for t, wd in enumerate(self.vocabulary_list):
            self.vocabulary_idx[wd] = t
        vocabulary_set = set(self.vocabulary_list)

        self.data_set_mapped = []
        for raw_doc in raw_docs:
            self.data_set_mapped.append([])
            for word in raw_doc:
                if word in vocabulary_set:
                    self.data_set_mapped[-1].append(self.vocabulary_idx[word])
```

解释：预处理会生成一个table，每一行代表一个文档，里面是词语。但是后面在统计词语的时候如果直接用字典来存储，则每个单词的查找需要花费 $O(lg|V|)$ 的时间，这里通过给每个单词一个整数编号来通过下标进行索引，将时间复杂度降到了 $O(1)$ 最终储存在了self.data_set_mapped变量中。

训练模型

```
def fit(self, num_iterations):
    # 训练模型
    num_docs = len(self.data_set_mapped)
    num_vocabulary = len(self.vocabulary_list)
    # 初始化超参数
    alpha = 2 * np.ones(self.k_topics)
    eta = 0.01 * np.ones(num_vocabulary)

    # 随机初始分配主题
    nd = np.zeros((num_docs, self.k_topics), dtype=np.int64)
    mk = np.zeros((self.k_topics, num_vocabulary), dtype=np.int64)
    z = [np.random.randint(0, self.k_topics, len(self.data_set_mapped[i])) for i
in range(0, num_docs)]
    new_z = [np.zeros(len(self.data_set_mapped[i]), dtype=np.int64) for i in
range(0, num_docs)]
    # 初始化nd和mk
    for d, doc in enumerate(self.data_set_mapped):
        for j, word in enumerate(doc):
            nd[d, z[d][j]] += 1
            mk[z[d][j], word] += 1
    # 吉布斯采样
    for iteration in range(num_iterations):
        print('Iteration %d' % (iteration + 1))
        denominator = np.sum(mk, axis=1) + np.sum(eta)
        # 更新 z
        for d, doc in enumerate(self.data_set_mapped):
            for j, word in enumerate(doc):
                prob_d = (mk[:, word] + eta[word]) * (nd[d, :] + alpha) /
denominator
                prob_d = prob_d / np.sum(prob_d)
                if not (prob_d > 0.).all():
                    print(prob_d)
                new_z[d][j] = np.argmax(np.random.multinomial(1, prob_d))
        # 更新隐变量
        for d, doc in enumerate(self.data_set_mapped):
            for j, word in enumerate(doc):
                mk[z[d][j], word] -= 1
                mk[new_z[d][j], word] += 1
                nd[d, z[d][j]] -= 1
                nd[d, new_z[d][j]] += 1
                z[d][j] = new_z[d][j]
    # 更新 theta 和 beta
    self.beta = mk + np.broadcast_to(eta, (self.k_topics, num_vocabulary))
    self.beta /= np.transpose(np.broadcast_to(np.sum(self.beta, axis=1),
(num_vocabulary, self.k_topics)))
    self.theta = nd + np.broadcast_to(alpha, (num_docs, self.k_topics))
    self.theta /= np.transpose(np.broadcast_to(np.sum(self.theta, axis=1),
(self.k_topics, num_docs)))
```

解释：函数的主体部分是按照前文的计算公式计算 $\beta_{k,t}$ 和 $\theta_{d,k}$，其本质是吉布斯采样。这里为了加快程序的速度，对每次更新时直接在原变量上进行加减（类似动态规划）

生成每个主题下的高频词汇

```
def topics_words(self, n):
    topic_words_map = np.argsort(-self.beta, axis=1)
    top_words = [[] for i in range(self.k_topics)]
    for i in range(self.k_topics):
        for j in range(n):
            top_words[i].append(self.vocabulary_list[topic_words_map[i, j]])
    return top_words
```

解释：储存主题-单词的概率变量为 β ，故对 β 的每一行按照频率降序排列，然后取前 k 个即可。

主函数

```
if __name__ == '__main__':
    stop_file = 'stopwords.txt'
    raw_data = pd.read_excel('data.xlsx')
    model = LDATopicModel(raw_texts=raw_data['content'], k_topics=8)
    start = time()
    model.preprocess(drop_n_freq=0)
    end = time()
    print('预处理用时{}s'.format(end - start))
    start = time()
    model.fit(num_iterations=60)
    end = time()
    print('训练用时{}s'.format(end - start))
    for i, words in enumerate(model.topics_words(n=15)):
        print('Topic %d:' % (i + 1), end='')
        for word in words:
            print(word, end=' ')
        print('')
    print('真实的主题有：体育 娱乐 彩票 房产 教育 游戏 科技 股票')
```

解释：主函数在训练模型的同时统计了运行时间的信息，输出为每个主题下概率最大的15个词汇。

实验中的困难与解决

中文的分词问题

在原论文中，训练的数据为英文文本。英文天然具有分割成单词的优势，但是中文的话每个字自己可以有含义，与其他字结合也会有别的含义。对于一句中文是否分割，在哪分割以及如何分割都是问题。为此参考了实验文档中给出的B站视频 https://www.bilibili.com/video/BV1LQ4y1Q7xv . 最终获得了不错的效果。

运行速度问题

一开始进行训练的时候以字典作为存储，但是字典的底层实现是红黑树，每次索引需要花费 $O(lg|V|)$ 的时间（V为语料库中全部单词的数量）.后来在一篇知乎文章的启发下知道了为单词建立索引（本质上是Hash散列表），将复杂度降到 $O(1)$ 。对于吉布斯采样过程，如果使用

库对函数进行加速的话，甚至可以将每次训练时间降到0.2秒的量级，但是由于实现较为复杂没有采用（目前的方案可以在100秒左右训练60次）。

实验结果展示

以下为 $\alpha = 2, \eta = 0.01$ 迭代60次的情况下的结果

```
Iteration 60
训练用时144.76516723632812s
Topic 1:专家 网友 老师 压力 分析 走势 黄金 新浪 股票 大盘 银行 成本 机会 趋势 整理
Topic 2:学生 大学 学校 专业 教育 移民 国家 孩子 能力 记者 留学生 网站 国际 费用 家长
Topic 3:电影 主持人 票房 影片 观众 演员 手机 故事 角色 娱乐 合作 电影节 现场 主演 女性
Topic 4:经济 公司 政府 企业 投资 研究 技术 海选 市场 行业 人类 科学家 文章 政策 基金
Topic 5:游戏 电子竞技 项目 玩家 世界 作品 国际 冠军 总决赛 网络 全球 比赛 网站 全国 互联网
Topic 6:主队 赔率 主场 数据 比赛 客场 公司 奇才 联赛 本场 足彩 客胜 球队 优势 助攻
Topic 7:比赛 球队 火箭 球员 篮板 新浪 奖金 训练 俱乐部 体育讯 绿城 机会 记者 内线 湖人
Topic 8:项目 市场 发展 建筑 生活 空间 地产 投资 文化 设计 户型 房子 新浪 产品 别墅
真实的主题有：体育 娱乐 彩票 房产 教育 游戏 科技 股票
```

股票
教育
娱乐
科技
游戏
彩票
体育
房产

第一个主题出现了**股票，大盘，走势**等，故为股票；

第二个主题出现了**教育，学生，大学，留学**等，故为教育；

第三个主题出现了**娱乐，电影，手机，主演**等，故为娱乐

第四个主题出现了**研究，技术，科学家**等，故为科技；

第五个主题出现了**游戏，电子竞技，玩家**等，故为游戏；

第六个主题出现了**足彩，赔率，球队**等，故为彩票；

第七个主题出现了**篮板，体育讯，湖人**等，故为体育；

第八个主题出现了**建筑，地产，户型，房子，别墅**等，故为房产。

可以看出每个主题都拟合的比较好，LDA模型的效果还是很不错的。

总结

本次实验的原理是目前上大学以来最复杂的，涉及的数学推导涵盖单多变量微积分，概统，随机过程等，但是经过学习和推敲逐渐理解了模型，最终完成的代码也没有想象的多。根据结果来看，LDA模型对主题的分类确实有着不错的效果。通过这次实验，我也学习很多的知识，锻炼了代码能力和搜索信息的能力，受益匪浅。