MLlab4 实验报告

大数据学院 和泳毅 PB19010450

一、 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。

二、数据集

800个中文文本文档。

三、 算法原理

Latent Dirichlet Allocation, LDA 模型由Blei, David M.、Ng, Andrew Y.、Jordan于2003年提出，用来推测文档的主题分布。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，从而通过分析一些文档抽取出它们的主题分布后，便可以根据主题分布进行主题聚类或文本分类。

LDA中的 3 个概念：

- **词**(word)：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，**不计顺序**。文档对象在话题模型中是“词袋”概念。
- **话题**(topic)：每篇文档都有特定的一些话题，根据这些话题产生。

1. 词袋模型

LDA 采用词袋模型。所谓词袋模型，是将一篇文档，我们仅考虑一个词汇是否出现，而不考虑其出现的顺序。在词袋模型中，“我喜欢你”和“你喜欢我”是等价的。与词袋模型相反的一个模型是n-gram，n-gram考虑了词汇出现的先后顺序。有兴趣的读者可以参考其他书籍。

2.多项式分布

二项分布是n重Bernoulli试验成功次数的离散概率分布。即$X \sim B(n,p)$：

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

而多项式分布，是二项分布扩展到多维的情况. 多项分布是指单次试验中的随机变量的取值不再是0-1的，而是有多种离散值可能（$1,2,3...,k$）.概率密度函数为：

$$P\left(x_1, x_2, \ldots, x_k; n, p_1, p_2, \ldots, p_k\right)=\frac{n!}{x_1! \ldots x_k!} p_1^{x_1} \ldots p_k^{x_k}.$$

3.Gamma函数、Beta分布、Dirichlet分布及其性质

（1）Gamma函数

Gamma函数的定义：

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

分部积分后，可以发现Gamma函数如有这样的性质：

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

Gamma函数可以看成是阶乘在实数集上的延拓，具有如下性质：

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

（2）Beta分布

Beta分布的定义：对于参数$\alpha>0,\beta>0$,取值范围为$[0,1]$的随机变量x的概率密度函数为:

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

其中

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

（3）Dirichlet分布

Dirichlet分布的概率密度函数为：

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

其中

$$B(\alpha)=\frac{\prod_{i=1}^{k}\Gamma\left(\alpha^i\right)}{\Gamma\left(\sum_{i=1}^{k}\alpha^i\right)},\sum_{i=1}^{k}x^i=1.$$

根据Beta分布、二项分布、Dirichlet分布、多项分布的公式，可以验证Beta分布是二项式分布的共轭先验分布，而Dirichlet分布是多项分布的共轭分布。

如果$p\sim Beta(t|\alpha,\beta)$，计算其期望有：

$$
\begin{aligned}
&E(p) = \int_0^1 t * \mathrm{Beta}(t \mid \alpha, \beta) dt \\
&= \int_0^1 t * \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} t^{(\alpha-1)}(1-t)^{\beta-1} dt \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 t^{\alpha}(1-t)^{\beta-1} dt.
\end{aligned}
$$

上述右式对应到概率分布 $Beta(t|\alpha+1,\beta)$，对于这个分布有

$$
\int_0^1 \frac{\Gamma(\alpha+\beta+1)}{\Gamma(\alpha+1)\Gamma(\beta)} t^{\alpha}(1-t)^{\beta-1} dt = 1,
$$

把上式带入$E(p)$中有

$$
E(p) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \cdot \frac{\Gamma(\alpha+1)\Gamma(\beta)}{\Gamma(\alpha+\beta+1)} = \frac{\alpha}{\alpha+\beta}.
$$

这说明，对于Beta分布的随机变量，其均值可以用$\frac{\alpha}{\alpha+\beta}$来估计。Dirichlet分布也有类似的结论，如果$p \sim Dir(t|\alpha)$, 同样可以证明

$$
E(p) = \left(\frac{\alpha^1}{\sum_{i=(1)}^K \alpha_i}, \frac{\alpha^{(2)}}{\sum_{i=1}^K \alpha_i}, \cdots, \frac{\alpha^{(K)}}{\sum_{i=1}^K \alpha_i} \right).
$$

4.MCMC 和 Gibbs Sampling

在现实应用中，我们很多时候很难精确求出精确的概率分布，常常采用近似推断方法。近似推断方法大致可分为两大类：第一类是采样(Sampling), 通过使用随机化方法完成近似；第二类是使用确定性近似完成近似推断，典型代表为变分推断(variational inference).

在很多任务中，我们关心某些概率分布并非因为对这些概率分布本身感兴趣，而是要基于他们计算某些期望，并且还可能进一步基于这些期望做出决策。概率图模型中最常用的采样技术是马尔可夫链脸蒙特卡罗(Markov chain Monte Carlo, MCMC).

给定连续变量$x \in X$的概率密度函数$p(x)$, 若有函数$f: X \mapsto R$，则可以计算$f(x)$的期望

$$
E_p[f(X)] = \int_x f(x)p(x)dx.
$$

若x不是单变量而是一个高维多元变量x, 且服从一个非常复杂的分布，则对上式求积分通常很困难。为此，MCMC先构造出服从p分布的独立同分布随机变量$x_1, x_2, \cdots, x_N$, 再得到上式的无偏估计

$$
\hat{E}_p[f(X)] = \frac{1}{N} \sum_{i=1}^N f(x_i).
$$

然而，若概率密度函数$p(x)$很复杂，则构造服从p分布的独立同分布样本也很困难。MCMC方法的关键在于通过构造“平稳分布为p的马尔可夫链”来产生样本：若马尔科夫链运行时间足够长，即收敛到平稳状态，则此时产出的样本X近似服从分布p。也就是说，MCMC方法先设法构造一条马尔科夫链，使其收敛至平稳分布恰为待估计参数的后验分布，然后通过这条马尔科夫

链来产生符合后验分布的样本，并基于这些样本来进行估计。这里马尔科夫链转移概率的构造至关重要，不同的构造方法将产生不同的MCMC算法。

Metropolis-Hastings(简称MH)算法是MCMC的重要代表。它基于“拒绝采样”（reject sampling）来逼近平稳分布p。算法如下：

输入：先验概率$Q(x^*|x^{t-1})$
过程：
(1) 初始化x^0
(2) for $t = 1, 2, \ldots$:
(3) 根据$Q(x^*|x^{t-1})$采样出候选样本x^*
(4) 根据均匀分布从$(0, 1)$范围内采样出阈值u
(5) if $u \leq A(x^*|x^{t-1})$:
(6) $x^t = x^*$
(7) else :
(8) $x^t = x^{t-1}$
(9) return $x^1, x^2, \ldots$
输出：采样出的一个样本序列

为了达到平稳状态，只需将接受率设置为

$$A\left(x^* \mid x^{t-1}\right)=\min \left(1, \frac{p\left(x^* Q\left(x^{t-1} \mid x^*\right)\right)}{p\left(x^{t-1}\right) Q\left(x^* \mid x^{t-1}\right)}\right).$$

吉布斯采样(Gibbs sampling)有时被视为MH算法的特例，它也使用马尔科夫链读取样本，而该马尔科夫链的平稳分布也是采用采样的目标分布$p(x)$。具体来说，假定$x = x_1, x_2, \cdots, x_N$, 目标分布为$p(x)$, 在初始化$x$的取值后，通过循环执行以下步骤来完成采样：

1. 随机或以某个次序选取某变量x_i；
2. 根据x中除x_i外的变量的现有取值，计算条件概率$p(x_i|X_i)$，其中$X_i = x_1, x_2, \cdots, x_{i-1}, x_{i+1}, x_N$；
3. 根据$p(x_i|X_i)$对x_i采样，用采样值代替原值。

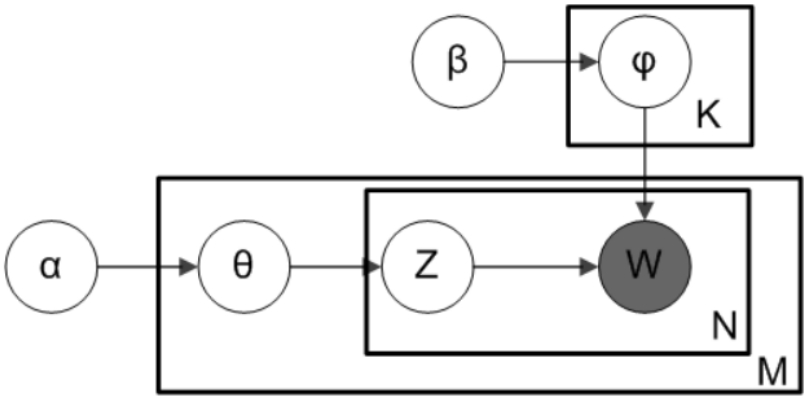
5.LDA

假定数据集中共含 K 个话题和 M 篇文档，词来自含 V 个词的字典：

- V个词的词典$\mathcal{V} = \{w_1, w_2, \ldots, w_V\}$；
- M篇文档$\mathbf{W} = \{\mathbf{w_1}, \mathbf{w_2}, \cdots, \mathbf{w_M}\}$，第$i$篇文档用长度为$N_i$的单词序列$w_i = \{w_{i,1}, \cdots w_{i,N_i}\}$表示。文档可以表示为话题的分布，即用长度为 K 概率话题向量表示。第 m 篇文档的概率话题向量为 $\theta_m \in [0, 1]^K$, 其中 $\theta_{mk} = P(z_k|w_m)$ 表示第 m 篇文档中话题 z_k 的概率。
- K个话题$\mathcal{Z} = \{z_1, z_2, \ldots, z_K\}$。话题可以表示为词的分布，即用长度为 V 的概率词向量表示。第 k 个话题的概率词向量为 $\varphi_k \in [0, 1]^V$, 其中 $\varphi_{kv} = P(w_v|z_k)$ 表示第 k 个话题中单词 w_v 的概率。
- 隐变量 $\mathbf{Z} = \{\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_M}\}$, 与文档 $\mathbf{W}$ 对应，其中 $z_{i,j}$ 表示单词 $w_{i,j}$ 的话题。

具体而言，一篇文档w是按下面的概率模型生成的：

1. 确定每个主题中词的概率：从以 β 为参数的Dirichlet分布中随机采样一个词分布 φ_k；
2. 给出文档主题：从以 α 为参数的Dirichlet分布中随机采样一个词分布 θ_m；
3. 根据θ_m 进行话题指派，得到文档 w_i 中的第 j 个词的话题 $z_{i,j}$；
4. 根据话题 $z_{i,j}$ 所对应的词分布 φ_k 随机采样生成词 $w_{i,j}$。



上图描述了LDA的变量关系，其中文档集 $\mathbf{W}$ 是唯一的已观测变量，它依赖于话题指派矩阵 $\mathbf{Z}$，以及话题所对应的词分布矩阵 φ; 同时话题指派矩阵 $\mathbf{Z}$ 依赖于话题分布矩阵 θ。而 φ 依赖于狄利克雷分布的参数 β, θ 依赖于狄利克雷分布的参数 α。

由上述变量关系，LDA 模型对应的概率分布为

$$p(\mathbf{W}, \mathbf{Z}, \varphi, \theta \mid \alpha, \beta)=\prod_{m=1}^{M} p(\theta_m \mid \alpha) \prod_{k=1}^{K} p(\varphi_k \mid \beta)\left(\prod_{n=1}^{N_m} p(w_{m,n} \mid z_{m,n}, \varphi_k) p(z_{m,n} \mid \theta_m)\right),$$

其中$p(\theta_m \mid \alpha), p(\varphi_k \mid \beta)$ 分别为以 α 和 β 为参数的 K 维和 V 维Dirichlet分布。

现给定训练数据 $\mathbf{W}=\{\mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_M}\}$, LDA的模型参数可以通过极大似然法估计，即寻找 α 和 β 以最大化对数似然

$$LL(\alpha, \beta)=\sum_{m=1}^{M} \ln p(\mathbf{w}_m \mid \alpha, \beta).$$

但由于 $p(\mathbf{w}_m \mid \alpha, \beta)$ 不易计算，上式难以直接求解，因此实践中常采用变分法来求取近似解。

若模型已知，即参数 α 和 β 已确定，则根据 $\mathbf{W}$ 来推断文档集所对应的话题结构（即推断 $\theta, \varphi, \mathbf{Z}$）可通过求解

$$p(\mathbf{Z}, \varphi, \theta \mid \mathbf{W}, \alpha, \beta)=\frac{p(\mathbf{W}, \mathbf{Z}, \varphi, \theta \mid \alpha, \beta)}{p(\mathbf{W} \mid \alpha, \beta)}.$$

然而由于分母难以计算，上式难以直接求解，因此在实践中常采用Gibbs采样或变分法进行近似推断。

本实验使用Gibbs采样完成LDA模型的近似推断：

- 首先遍历所有文档中的所有词，为其各随机分配一个主题$z_{i,j} \in \mathcal{Z}$，表示第i篇文档中第j个词属于主题$z_{i,j}$。
- 对隐变量θ, φ积分，得到边缘概率$p(\mathbf{Z} \mid \mathbf{W}, \alpha, \beta)$：

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

其中$\mathrm{B}(\cdot)$为 Beta 函数，$n_{i,k} = \sum_{j=1}^{N_i} \mathbb{I}(z_{i,j} = k)$表示第$i$篇文档中第$k$个话题的词的频数，$m_{k,v} = \sum_{i=1}^{M} \sum_{j=1}^{N_i} \mathbb{I}(w_{i,j} = v)\mathbb{I}(z_{i,j} = k)$表示所有文档中第$k$个话题下词$w_v$出现的频数。

- 即重复以下迭代：对所有文档的所有词遍历。对于属于文档中的词，取出该词，根据LDA中主题的概率分布采样出新主题：

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

- 迭代完成后，输出对文档-主题分布θ和主题-词分布φ的估计：

$$\theta_{i,k} \approx \frac{n_{i,k} + \alpha_k}{\sum_{k=1}^{K} n_{i,k} + \alpha_k},$$
$$\varphi_{kv} \approx \frac{m_{k,v} + \beta_v}{\sum_{v=1}^{V} m_{k,v} + \beta_v}.$$

四、核心代码讲解

1.中文文本处理

```python
def word_cut(text):
    """
    将一个中文文本处理为词语列表
    :param text: 中文文本
    :return: 词语列表
    """
    jieba.load_userdict(dic_file)  # 可以在jieba词典中添加新词
    jieba.initialize()
    try:
        stopword_list = open(stop_file, encoding='utf-8')
    except:
        stopword_list = []
        print("error in stop_file")
    stop_list = []
    flag_list = ['n', 'nz', 'vn']  # 设定只需要名词、专有名词、动名词
    # 读取设定的停用词
```

```
    for line in stopword_list:
        line = re.sub(u'\n|\\r', '', line)
        stop_list.append(line)

    word_list = []
    # 分词
    seg_list = psg.cut(text)
    for seg_word in seg_list:
        word = re.sub(u'[^\u4e00-\u9fa5]', '', seg_word.word)
        find = 0
        # 不记录停用词以及小于两个字的词
        if word in stop_list or len(word) < 2:
            continue
        if seg_word.flag in flag_list:
            word_list.append(word)
    return word_list

def text_handling(data):
    """
    对n个中文文本处理，并统计总词语频数
    :param data: n个中文文本
    :return: 词语列表，词语频数
    """
    freq = {}
    word_list = []
    for i in range(len(data)):
        word_list.append([])
        word_list[-1] = word_cut(data[i])
        for word in word_list[i]:
            if word in freq.keys():
                freq[word] += 1
            else:
                freq[word] = 1
    return word_list, freq
```

使用jieba库进行中文文本词语提取，并且只提取名词、专有名词、动名词。设置新词库和停用词库的功能。

2.类初始化

```
def __init__(self, data, topic_num, alpha, beta):
    """
    data: 所有文档
    text_num: 文档个数
    topic_num: 主题个数
    alpha: Dirichlet分布参数
    beta: Dirichlet分布参数
    """
    self.data = data
    self.text_num = len(data)
    self.topic_num = topic_num
    self.alpha = alpha
    self.beta = beta
```

3.数据预处理

```
def preprocess(self, freq, drop_num=0):
    """
    去除高频词语，并将词语序列映射为词典索引序列（降低复杂度）
    :param freq: 词语频数
    :param drop_num: 需要去除的高频词语个数
    """
    # 合并所有文档的单词序列
    all_words = []
    for i in range(self.text_num):
        all_words = all_words + self.data[i]
    # 去除重复单词
    unique_words = list(set(all_words))
    # 提供去除高频词语功能
    unique_words.sort(key=lambda wd: freq[wd], reverse=True)
    self.word_list = unique_words[drop_num:]
    # 将词语映射为词典索引
    self.word_idx = {}
    for t, wd in enumerate(self.word_list):
        self.word_idx[wd] = t
    # 重新处理所有文档，将词语序列映射为词典索引序列
    self.word_map = []
    for word_line in self.data:
        self.word_map.append([])
        for word in word_line:
            if word in self.word_list:
                self.word_map[-1].append(self.word_idx[word])
```

lda数据预处理，提供去除高频词语功能，并为了降低时间复杂度，提高性能，将词语序列映射为词典索引序列。

4.训练

```
def fit(self, epochs):
    """
    训练过程
    :param epochs: 迭代次数
    """
    word_num = len(self.word_list)
    # 设置超参
    alpha = self.alpha * np.ones(self.topic_num)
    beta = self.beta * np.ones(word_num)
    # 初始化参数
    nd = np.zeros((self.text_num, self.topic_num), dtype=np.int)
    mk = np.zeros((self.topic_num, word_num), dtype=np.int)
    z = [np.random.randint(0, self.topic_num, len(self.word_map[i])) for i in range(0, self.text_num)]
    new_z = [np.zeros(len(self.word_map[i]), dtype=np.int) for i in range(0, self.text_num)]
    # 统计变量
    for i, wordlist in enumerate(self.word_map):
        mk, nd = self.Gibbs_0(np.array(wordlist), mk, nd, i, np.array(z[i]))

    # 吉布斯采样
    for epoch in range(1, epochs + 1):
        if epoch % 5 == 0:
            print('--第', epoch, '次迭代--')

        temp = (np.sum(mk, axis=1) + np.sum(beta)) * (np.sum(nd) + np.sum(alpha))
        for i, wordlist in enumerate(self.word_map):
            new_z[i] = list(self.Gibbs_1(np.array(wordlist), temp, mk, beta, alpha, nd, i, np.array(new_z[i])))

        for i, wordlist in enumerate(self.word_map):
            mk, nd, z[i] = self.Gibbs_2(np.array(wordlist), mk, nd, np.array(z[i]), i, np.array(new_z[i]))

    # 参数输出
    self.phi = mk + np.broadcast_to(beta, (self.topic_num, word_num))
    self.phi = self.phi / np.transpose(np.broadcast_to(np.sum(self.phi, axis=1), (word_num, self.topic_num)))
    self.theta = nd + np.broadcast_to(alpha, (self.text_num, self.topic_num))
    self.theta = self.theta / np.transpose(np.broadcast_to(np.sum(self.theta, axis=1), (self.topic_num, self.text_num)))
```

为了提高性能，将所有复杂的迭代部分写为numpy数组，使用numba用LLVM编译技术来解释字节码，实现对迭代的加速。其中加速部分写为类内静态函数：

```
@staticmethod
@jit(nopython=True)
def Gibbs_0(wordlist, mk, nd, i, z):
    for j, word in enumerate(wordlist):
        nd[i, z[j]] += 1
        mk[z[j], word] += 1
    return mk, nd

@staticmethod
@jit(nopython=True)
def Gibbs_1(wordlist, temp, mk, beta, alpha, nd, i, new_z):
    for j, word in enumerate(wordlist):
        p = (mk[:, word] + beta[word]) * (nd[i, :] + alpha) / temp
        p = p / np.sum(p)
        if not (p > 0.).all():
            print(p)
        new_z[j] = np.argmax(np.random.multinomial(1, p))
    return new_z

@staticmethod
@jit(nopython=True)
def Gibbs_2(wordlist, mk, nd, z, i, new_z):
    for j, word in enumerate(wordlist):
        mk[z[j], word] -= 1
        mk[new_z[j], word] += 1
        nd[i, z[j]] -= 1
        nd[i, new_z[j]] += 1
        z[j] = new_z[j]
    return mk, nd, z
```

5.输出

```
def topics_words(self, word_num):
    """
    输出每个主题下概率最高的的n个词语
    :param word_num: 需要输出每个主题下概率最高的的词语个数
    :return: 词语序列
    """
    topic_word = np.argsort(-self.phi, axis=1)
    top_words = [[]] * self.topic_num
    for i in range(self.topic_num):
        top_words[i] = []
```

```
        for j in range(word_num):
            top_words[i].append(self.word_list[topic_word[i][j]])
    return top_words

def text_topics(self, topic_num):
    """
    输出每个文档下概率最高的的n个主题
    :param topic_num: 需要输出每个文档下概率最高的的主题个数
    :return: 主题序列
    """
    return np.argsort(-self.theta, axis=1)[:, :topic_num]
```

五、实验中遇到的问题及解决方案

- **结果包含无意义词语**

  对文档进行预处理，观察词频统计设置停用词，或者去除一定数量的高频词。

- **运行时间慢**

  优化代码，对迭代部分使用numba加速。

六、实验结果

首先统计全部词语的频数，观察出现次数前20%的词语，将没有意义的词语或与主题无关的词语添加到停用词库。例如在上图原始词云中，“时候”、“情况”、“大家”等词语对于lda学习是没有意义的，可以添加到停用词库。对文本的预处理可以很明显地改善最终的结果。

令主题数为8，$\alpha = 0.3$，$\beta = 0.3$，迭代50次，结果如下：

```
Topic 1 : ['研究', '科学家', '人类', '流感', '技术', '世界', '公司', '人员', '科技', '太空', '航天飞机', '任务', '流感病毒', '国家', '消息']
Topic 2 : ['学生', '大学', '学校', '教育', '专业', '移民', '孩子', '奇才', '留学生', '能力', '国家', '家长', '国际', '课程', '同学']
Topic 3 : ['专家', '走势', '投资', '分析', '黄金', '突破', '市场', '成本', '股票', '公司', '大盘', '海选', '趋势', '风险', '企业']
Topic 4 : ['电影', '主持人', '票房', '影片', '观众', '演员', '作品', '故事', '电影节', '合作', '娱乐', '主演', '记者', '市场', '角色']
Topic 5 : ['项目', '市场', '建筑', '地产', '空间', '户型', '区域', '国际', '房子', '设计', '文化', '别墅', '主持人', '投资', '产品']
Topic 6 : ['比赛', '电子竞技', '总决赛', '冠军', '项目', '游戏', '战队', '赛事', '赛区', '玩家', '现场', '世界', '星际争霸', '决赛', '投票']
Topic 7 : ['比赛', '游戏', '球队', '火箭', '网站', '球员', '手机', '篮板', '互联网', '公司', '网游', '助攻', '体育讯', '玩家', '记者']
Topic 8 : ['主队', '赔率', '主场', '比赛', '客场', '联赛', '公司', '数据', '球队', '足彩', '本场', '客胜', '客队', '平局', '博彩']
```

预处理用时2.65秒，迭代用时3.32秒，总用时5.97秒。时间上的可接受的。

可以看出主题之间的词具有明显的相关性，并且可以很明显地发现主题1-8分别对应**科技、教育、股票、娱乐、房产、游戏、体育、彩票**。结果很好。

并且由于已知各文档真实主题，用 `text_topics` 函数输出每个文档概率最高的主题，对比而得主题准确率为82%。观察错误样本可以发现，大部分错误主题指派发生在主题**体育**与**彩票**之间，两者内容都包含各种体育运动，确实容易出错。

七、总结

本次实验是本课程实验中原理最为复杂的，通过浏览很多相关的文章和博客，才把LDA及相关的吉布斯采样算法理解清楚。虽然最终的代码量不大，但前期学习部分的工作量很大。

可以看到本次实验实现的LDA模型在给定的中文文本上有很好的结果，主题之间的词具有明显的相关性，并且也符合预期的文档主题。但此次实验仅仅是实现了吉布斯采样算法，这是一个近似算法，并且是在已知参数 α, β 的情况下进行的。若想要推断这两个参数，还要使用更为复杂的算法。并且为了简化实验，本实验给定了主题数，在实际情况下还需要使用分析困惑度的方法来确定主题数。另外，在某些场合下，文档是在线生成的，还需要一些在线的算法去进行模型的学习与推断。总之，对于LDA模型还有许多内容需要学习。