第3章

文本表示模型

一般来说，文本挖掘算法不能直接在原始文本形式上处理。因此，在预处理阶段，将文本转化为更易计算机识别的信息，即对文本进行形式化处理。这一形式化的结果称为文本表示，不同的文本表示模型有不同的特点，所以根据文本的特点和文本处理的要求选择合适的文本表示模型是非常重要的。目前比较常见的文本表示模型包括向量空间模型、概率模型、主题模型等。

3.1 文本预处理

我们得到的数据（如利用爬虫从网页上爬取的数据）包含了丰富的描述数据，因为数据具有多样性，因此一般是非结构化数据，而直接对非结构化的文本数据分析有一定的限制和困难。尤其是一些低质量的数据进入系统将导致昂贵的操作费用和系统漫长的响应时间，并且对从数据集中抽取的模式的正确性和导出规则的准确性产生巨大的影响，更严重的会使决策支持系统产生错误的分析结果，误导决策。所以，在进行数据挖掘之前，需要对数据进行预处理，将数据处理成适合文本挖掘的格式。

3.1.1 原始数据处理

在一些搜索引擎应用系统中，由爬虫发现的文档或由信息源提供的文档，通常都不是纯文本，它们的格式多种多样，如 HTML、XML、Adobe PDF、Microsoft Word 或 Microsoft PowerPoint 等格式的文本。还有些数据来源于文档数据库，它用于管理大量的文档及与这些文档相关的结构化数据。结构化数据包括文档的元数据，以及从文档中抽取出来的其他信息，如超链接和锚文本（Anchor Text，与超链接关联的文本）。

1. 去除数据中非文本部分

这一步主要是针对用爬虫收集的语料数据，由于爬取的内容中有很多 HTML 的标签，需要去掉。少量的非文本内容可以直接用 Python 的正则表达式(re)去除，复杂的则可以用 Beautiful Soup 去除。去除掉这些非文本的内容后，我们就可以进行真正的文本预处理了。Beautiful Soup 是 Python 的一个库，最主要的功能就是从网页爬取我们需要的数据。Beautiful Soup 将 HTML 解析为对象进行处理，全部页面转换为字典或数组，相对于正则表达式的方式，可以大大简化处理过程。

例 3.1　Python 正则提取字符串里的中文。

```
# - * - coding: UTF - 8 - * -
import re
#过滤掉除中文以外的字符
str = "hello,world!! % [545]你好 234 世界..."
str = re.sub("[A - Za - z0 - 9\!\ % \[\]\,\.]", "", str)
print(str)
#提取字符串里的中文,返回数组
pattern = "[\u4e00 - \u9fa5] + "
regex = re.compile(pattern)
results = regex.findall("adf 中文 adf 发京东方")
print(results)
```

程序运行结果如图 3.1 所示。

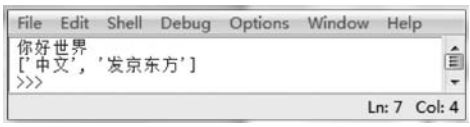


图 3.1　汉字提取运行结果

2. 经典数据集

Python NLTK 提供了非常多经典的数据集，很多数据集都是手工标注而成。NLTK 是由宾夕法尼亚大学计算机与信息科学系使用 Python 语言实现的一种自然语言工具包，其收集的大量公开数据集、模型上提供了全面、易用的接口，涵盖了分词、词性标注(Part-of-Speech Tag，POS-Tag)、命名实体识别(Named Entity Recognition，NER)、句法分析(Syntactic Parse)等各项自然语言处理(Natural Language Processing，NLP)领域的功能。

THUCNews 是根据新浪新闻 RSS 订阅频道 2005—2011 年间的历史数据筛选过滤生成，包含 74 万篇新闻文档(2.19 GB)，均为 UTF-8 纯文本格式。

3. 编码处理

1) 乱码问题

若开启的文档或下载的网页有乱码现象，通常是因为发送端与接收端的字符编码

(Character Encoding)设定不同所致。所以,在遇到乱码时,掌握文件(网页)的编码与Python环境的字符编码,确定这两个地方字符编码一致以后就没有问题了。

例 3.2 Python乱码处理的例子。

(1) Python不能将汉字编码直接输出汉字,需要转换成Unicode,然后用print()函数输出,代码如下。

```
str = b'\xc7\xeb\xca\xb9\xd3\xc3\xca\xda\xc8\xa8\xc2\xeb\xb5\xc7 \xc2\xbc\xa1\xa3\xcf\
xea\xc7\xe9\xc7\xeb\xbf\xb4'
print(str.decode('gbk'))
```

程序运行结果如图3.2所示。

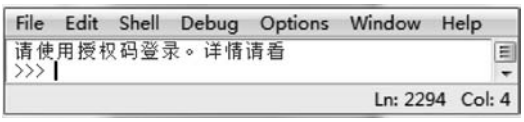


图3.2 汉字编码转换成Unicode输出

(2) 将百分比编码的序列解码为Unicode字符。

```
import urllib.parse
c = 'cardId = 110110110110&mobile = 13123456789&realName = %E6%9D%8E%E9%9B%B7'
res = urllib.parse.unquote(c)
print(res)
```

程序运行结果如图3.3所示。

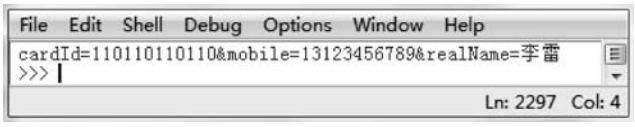


图3.3 百分比编码的序列解码为Unicode字符

2) 字符编码种类

字符编码主要可以分为地区性编码与国际性编码。地区性编码为各地区自行发展出来的编码方式,各地皆有专属的字符编码,以中文为例,惯用繁体字的地区使用的编码是大五码(Big5),惯用简体字的地区使用的是汉字内码扩展规范(GBK)。在现今网络无国界,交流越来越频繁的情况下,因字符编码不同导致的乱码问题影响会越来越大。

国际性编码,即为解决地区性编码所导致的混乱问题所发展出的解决之道。最典型的例子为UTF-8(8-bit Unicode Transformation Format),它是由万国码(Unicode)延伸出来的编码方式,但是与Unicode属不同的编码方式。这个编码系统的发展可以收录更多的字符,因此可以处理更大数量的字符,统一了所有国家的不同编码,使UTF-8通用于繁体中文与简体中文。

3.1.2　文本预处理简述

文本预处理是文本分类的初始阶段，是将原始的文本表示成计算机可识别的特定的数学模型。

1. 文本预处理的一般过程

中文文本预处理的一般过程有分段、分句、分词、词性标注、停用词过滤、权重设置等，如图 3.4 所示。

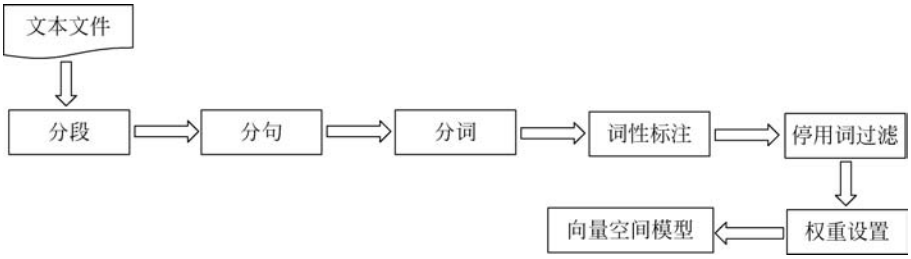


图 3.4　文本预处理的一般过程

(1) 对输入的文本进行删除开始、结尾以及段与段之间的空格等操作。
(2) 识别换行符号，对文本进行分段，并用分段标记标识段落。
(3) 识别断句符号，如“。”“！”“？”“；”等，并用断句符号标记句子。
(4) 对断句出来的句子用分词算法进行分词，并进行词性标注。
(5) 处理完成之后，采用一定的策略对之前得到的词汇进行特征词选取。
(6) 设置特征词汇的权重。
(7) 形成文本的向量空间模型。

2. 文本分段和分句

文本分段一般采用识别分段符号的方法，在 ASCII 码中，分段(换行)符是“10”和“13”。在 HTML 网页中出现的大段文字，通常采用< p > </p >标签进行分段规划，即段落的开始用< p >标签，段落的结束用</p >标签。

与分段相似，文本的断句也使用识别断句符号的方法。常用的中文断句标点符号如下。
(1) 中文常用句末标点符号，如“。”“！”“？”“：”“……”“；”等；
(2) 中文右匹配标点符号，如“ ” ”“)”“】”“》”等也视为断句符号；
(3) 英文句末标点符号，如“.”“!”“?”“:”“;”等；
(4) 英文右匹配符号，如“'”“"”“)”等。

例 3.3　使用 re.split()函数，可以进行一次性分隔、断句。

```
import re
pattern = r',|\.|/|;|\'|-|\[|\]|<|>|\?|:|"|\{|\}|\~|!|@|#|\$|%|\^|&|\(|\)|-|=
|\_|\+|,|.|、|; |'|'|【|】|«|»|·|!| |…|(|)'
```

```
test_text = '人民网北京 10 月 27 日电 (记者 刘洁妍)第十三届全国人民代表大会常务委员会第十四次会议 26 日表决通过«全国人民代表大会常务委员会关于授权澳门特别行政区对横琴口岸澳方口岸区及相关延伸区实施管辖的决定»。'
result_list = re.split(pattern, test_text)
print(result_list)
```

程序运行结果如图 3.5 所示。

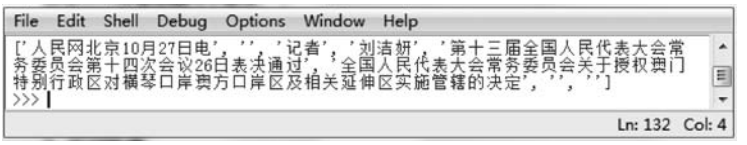


图 3.5 re.split()函数断句示例

3. 分词处理

中文与英文最大的区别在于中文的词汇可以由一个或是两个以上邻近的字组成，而词汇与词汇之间不像英文有明显的间隔。此时我们就需要利用分词将文档中的句子分成词汇。

4. 词性标注

词性标注是指为分词结果中的每个单词标注一个正确的词性，即确定每个词是名词、动词、形容词或其他词性。在汉语中，词性标注比较简单，因为汉语词汇词性多变的情况比较少见，大多词语只有一个词性，或者出现频次最高的词性远远高于第二位的词性。由于在词性标注中早期出现的错误会在后续处理中被放大，因此会直接影响到后续文本挖掘的效果。

目前常用的词性标注方法主要分为 3 种：基于规则的方法、基于统计的方法以及规则和统计相结合的方法。表 3.1 给出部分汉语词性对照表。

表 3.1 汉语词性对照表(部分)

代码	名 称	说 明	举 例
n	名词	取英语名词 noun 的第 1 个字母	希望/v 双方/n 在/p 市政/n 规划/vn
v	动词	取英语动词 verb 的第 1 个字母	举行/v 老/a 干部/n 迎春/vn 团拜会/n
a	形容词	取英语形容词 adjective 的第 1 个字母	最/d 大/a 的/u
d	副词	取 adverb 的第 2 个字母，因其第 1 个字母已用于形容词	两侧/f 台柱/n 上/分别/d 雄踞/v 着/u
c	连词	取英语连词 conjunction 的第 1 个字母	全军/n 和/c 武警/n 先进/a 典型/n 代表/n
p	介词	取英语介词 prepositional 的第 1 个字母	往/p 基层/n 跑/v。/w
q	量词	取英语 quantity 的第 1 个字母	不止/v 一/m 次/q 地/u 听到/v，/w
m	数词	取英语 numeral 的第 3 个字母，n 和 u 已有他用	科学技术/n 是/v 第一/m 生产力/n

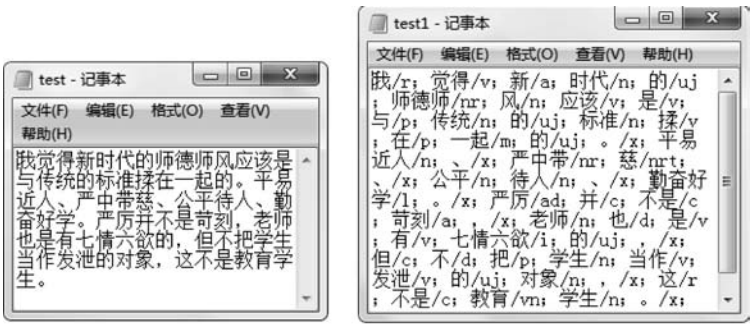
续表

代码	名　称	说　　明	举　　例
r	代词	取英语代词 pronoun 的第 2 个字母，因 p 已用于介词	有些/r 部门/n
u	助词	取英语助词 auxiliary 的第 2 个字母，因 a 已用于形容词	工作/vn 的/u 政策/n
t	时间词	取英语 time 的第 1 个字母	当前/t 经济/n 社会/n 情况/n
w	标点符号		生产/v 的/u5G/nx、/w8G/nx 型/k 燃气/n 热水器/n
k	后接成分		权责/n 明确/a 的/u 逐级/d 授权/v 制/k
e	叹词	取英语叹词 exclamation 的第 1 个字母	嗬/e！/w
vn	动名词	指具有名词功能的动词。动词和名词的代码并在一起	股份制/n 这种/r 企业/n 组织/vn 形式/n，/w
nx	字母专名		ATM/nx 交换机/n
ad	副形词	直接作状语的形容词	一定/d 能够/v 顺利/ad 实现/v。/w

例 3.4　对中文文本文件中的内容进行分词及词性标注。

```
import jieba
import jieba.posseg as pseg
p = open(r'D://python3sy/test.txt', 'r', encoding = 'gbk')
q = open(r'D://python3sy/test1.txt', 'w', encoding = 'gbk')
for line in p.readlines():
    words = pseg.cut(line)
    for word, flag in words:
        q.write(str(word) + "/" + str(flag) + "; ")
    q.write('\n')
    q.close()
```

源文本文件和标注后结果如图 3.6 所示。



(a) 源文本文件　　(b) 结果文本文件

图 3.6　词性标注前后的文本文件

5. 停用词过滤

在对文本进行分词之后，文本就变成了一系列词集的表示。但是文本中的词并不是出现频率越高，代表性就越强。事实上，如果一个词项在文档集中出现过于频繁(英语里诸如 a、the、or 等使用频率很高的词)，则对文档的区分是没有意义的，我们称之为停用词。停用词对于文本所表达的内容几乎没有任何贡献，即对于文本分类没有太大作用。因此，有必要将这些停用词从原始文本中过滤掉，这个过程称为停用词过滤。

停用词过滤通常有两种方法：一种方法是基于统计的，统计每个词项在文档集中出现的文档数，如果超过总数量的某个百分比(如 80%)，就将这个词项作为停用词过滤；另一种是通过建立一个停用词表来实现，这个列表包含所有的停用词，如哈工大停用词词库、四川大学机器学习智能实验室停用词库、百度停用词表等各种停用词表，如表 3.2 所示。

表 3.2　停用词表(部分)

啊	吧	鄙人	不成	不怕	趁着	从	的	多少	反过来说
阿	吧哒	彼	不单	不然	乘	从而	的话	而	反之
哎	把	彼此	不但	不如	冲	打	等	而况	非但
哎呀	罢了	边	不独	不特	除	待	等等	而且	非徒
哎哟	被	别	不管	不惟	除此之外	但	地	而是	否则
唉	本	别的	不光	不问	除非	但是	第	而外	嘎
俺	本着	别说	不过	不只	除了	当	叮咚	而言	嘎登
俺们	比	并	不仅	朝	此	当着	对	而已	该
按	比方	并且	不拘	朝着	此外	到	对于	尔后	赶
按照	比如	不比	不论	趁	此间	得	多	反过来	个

使用停用词表过滤停用词的过程很简单，就是一个查询过程，对每个词汇，看其是否位于停用词表中，如果是，则从词汇中删除。

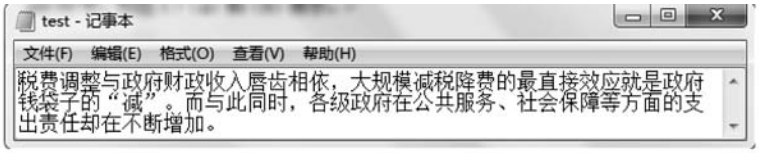
停用词过滤一方面可以降低特征维度，提高文本分类算法的效率和速度，节省计算资源；另一方面可以准确地表示文本。

例 3.5　读出源文本文件，去除停用词，完成操作后将结果写入目标文件。

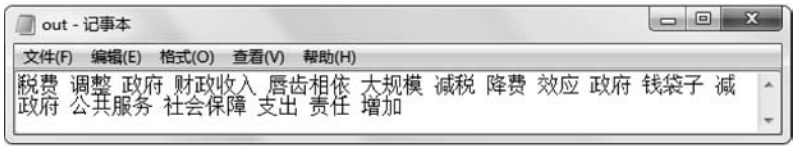
```
from collections import Counter
import jieba
#创建停用词列表
def stopwordslist(filepath):
    stopwords = [line.strip() for line in open('D://python3sy/stopword.txt', 'r').
readlines()]
    return stopwords
#对句子进行分词
def seg_sentence(sentence):
    sentence_seged = jieba.cut(sentence.strip())
    stopwords = stopwordslist('D://python3sy/stopword.txt')
```

```
    #这里加载停用词的路径
    outstr = ''
    for word in sentence_seged:
        if word not in stopwords:
            if word != '\t':
                outstr += word
                outstr += " "
    return outstr
inputs = open('D://python3sy/test.txt', 'r')      #加载要处理的文件的路径
outputs = open('D://python3sy/out.txt', 'w')      #加载处理后的文件路径
for line in inputs:
    line_seg = seg_sentence(line)                 #这里的返回值是字符串
    outputs.write(line_seg)
    outputs.close()
    inputs.close()
```

原始文本数据在路径 D://python3sy/下的文本文件 test.txt 中，去除停用词后的数据在文本文件 out.txt 中，如图 3.7 所示。



(a) 原始文本文件



(b) 去除停用词后的结果

图 3.7　去除停用词的原始文本和结果数据

6. 特征词选择与权重设置

经过前面的分词得到的文本词汇，再进行文本特征词的选择后，一个文本就由自然语言描述转化为由一系列的特征词描述。由于特征词对于进一步分类的影响不同，有些特征词区分能力强，而有些则弱，因此需要对特征词进一步加权处理，以尽量提高具有类别区分能力的特征词的权重，降低缺乏类别区分能力的特征词的权重。

目前最常用的特征词加权算法是 2.4 节讲述的基于统计的 TF-IDF 算法。TF-IDF 算法最早是由 Salton 和 Buckley 于 1988 年提出并用于信息检索领域的，后来被应用于文本分类和聚类等数据挖掘中的特征词加权。

权值的计算需要视具体情况而定，至今没有普遍适用的“最优公式”。国内外相关工作者在这方面也做了大量的工作，并取得了一些较好的成果，如第 2 章讲述的综合文本特征词选择的方法。

3.2 向量空间模型

向量空间模型(Vector Space Model,VSM)是由哈佛大学的 Gerard Salton 等专家于 20 世纪 60 年代提出的,已经在文本分类、自动标引、信息检索等许多领域得到了广泛的应用。

3.2.1 向量空间模型的概念

向量空间模型(VSM)的基本思想是把文本表示成向量空间中的向量,采用向量之间的夹角余弦作为文本间的相似性度量。向量维对应文本特征词在文档集中的权值,这种表示形式也称为词袋(Bag of Words)。为了将文本向量化,首先把文本的内容简单地看作它含有的基本语言单位(字、词、词组或短语)所组成的集合,这些基本的语言单位统称为特征项。假设文本集是特征项的集合 $D=\{w_{ij}\mid w_{ij}$ 是特征项,$1\leqslant i\leqslant|D|,1\leqslant j\leqslant n\}$,则对于 D 中的某文本 $\boldsymbol{D}_i$,可以用特征项集表示为 $\boldsymbol{D}_i=(w_{i1},w_{i2},\cdots,w_{in})$,其中 $1\leqslant i\leqslant|D|$,本节将文本集 D 和某文本 $\boldsymbol{D}_i$ 都看作特征项(词)的集合,其关系为 $\boldsymbol{D}_i\subseteq D$。然后根据各个项 w_{ij} 在文本中的重要性为其赋予一定的权重 t_{ij},此时,文本 $\boldsymbol{D}_i$ 就可以标记为 $\boldsymbol{D}_i=(w_{i1},t_{i1};w_{i2},t_{i2};\cdots;w_{in},t_{in})$。

所谓向量空间模型(VSM),是指给定一个文档集 $D=(w_1,t_1;w_2,t_2;\cdots;w_n,t_n)$,$D$ 符合以下两条约定。

(1) 各个特征项 $w_k(1\leqslant k\leqslant n)$互异(即没有重复)。

(2) 各个特征项 w_k 无先后顺序关系(即不考虑文档的内部结构)。

在以上两个约定下,可以把特征项看成一个 n 维坐标系,而权重 $t_1,t_2,\cdots,t_n$ 为相应的坐标值。因此,一个文本 $\boldsymbol{D}_i=(w_{i1},t_{i1};w_{i2},t_{i2};\cdots;w_{in},t_{in})$就表示为 n 维空间中的一个向量,称为文本集 D 的向量空间模型,如图 3.8 所示。

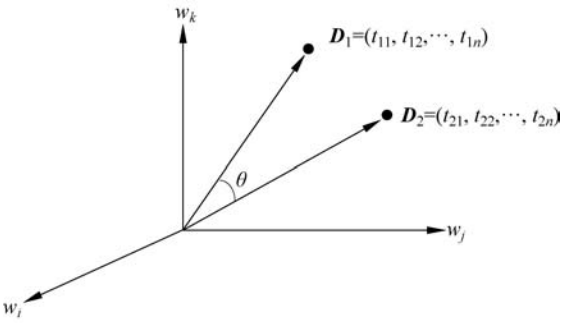


图 3.8 向量空间模型

图 3.8 中,$\theta(0\leqslant\theta\leqslant\pi)$为文本向量 $\boldsymbol{D}_1$ 和 $\boldsymbol{D}_2$ 的夹角。

3.2.2 文本向量的相似度

在文本挖掘的过程中,我们经常需要知道两个文本向量间差异的大小,进而来评价文本向量的相似度和类别。相似度是描述两个文本向量之间相似程度的一种度量。任意两

个文档 $\boldsymbol{D}_i=(w_{i1},w_{i2},\cdots,w_{in})$ 和 $\boldsymbol{D}_j=(w_{j1},w_{j2},\cdots,w_{jn})$ 之间的相似度指两个文档内容的相关程度(Degree of Relevance)。

1. 向量内积

当文本被表示成空间向量时,可以借助向量之间的某种距离来表示文本之间的相似程度,目前常用的方法是使用向量内积来计算,即

$$\mathrm{Sim}(\boldsymbol{D}_i,\boldsymbol{D}_j)=\sum_{k=1}^{n}t_{ik}t_{jk} \tag{3-1}$$

其中,t_{ik} 和 t_{jk} 分别为词项 w_{ik} 和 w_{jk} 的权重。内积代数值越大,相似度越大。

如果两个向量离得越近,那么这两个向量的夹角就越小,直到这两个向量平行且模相同时,这两个向量才完全相等。考虑到向量的归一化,内积也可以使用两个向量夹角 θ 的余弦值来表示,即

$$\mathrm{Sim}(\boldsymbol{D}_i,\boldsymbol{D}_j)=\cos\theta=\frac{\sum_{k=1}^{n}t_{ik}t_{jk}}{\sqrt{\sum_{k=1}^{n}t_{ik}}\sqrt{\sum_{k=1}^{n}t_{jk}}} \tag{3-2}$$

从式(3-1)和式(3-2)可以看出,对于向量空间模型,存在两个重要因素,即文本特征词的选择和文本特征词的权重计算,其中文本特征词的权重计算对文本分类效果影响很大。

例 3.6　计算文档 $\boldsymbol{D}_i=(0.2,0.4,0.11,0.06)$ 和 $\boldsymbol{D}_j=(0.14,0.21,0.026,0.34)$ 的相似度。

程序源代码如下。

```
import numpy as np
Di = [0.2,0.4,0.11,0.06]
Dj = [0.14,0.21,0.026,0.34]
dist1 = np.dot(Di,Dj)/(np.linalg.norm(Di) * np.linalg.norm(Dj))
print("余弦相似度: sin(Di,Dj) = \t" + str(dist1))
```

程序运行结果如图 3.9 所示。

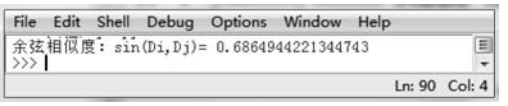


图 3.9　文档 $\boldsymbol{D}_i$ 和 $\boldsymbol{D}_j$ 的相似度

该程序段中用到了内积函数 dot()和范式函数 linalg.norm()。

2. 向量间的距离

度量两个文本向量关系的另一个重要指标是文本向量距离。文本向量距离与文本向量相似度之间有着密切的关系。两个文本向量的距离越大,其相似度越低;反之,两个文

本向量的距离越小，其相似度越大。

两个文本向量 $\boldsymbol{D}_i=(w_{i1},w_{i2},\cdots,w_{in})$ 和 $\boldsymbol{D}_j=(w_{j1},w_{j2},\cdots,w_{jn})$ 之间的相似度记为 $\mathrm{Sim}(\boldsymbol{D}_i,\boldsymbol{D}_j)$，其文本向量距离为 $\mathrm{Dis}(\boldsymbol{D}_i,\boldsymbol{D}_j)$，那么可以定义一个满足以上条件的简单的转换关系为

$$\mathrm{Sim}(\boldsymbol{D}_i,\boldsymbol{D}_j)=\frac{\alpha}{\mathrm{Dis}(\boldsymbol{D}_i,\boldsymbol{D}_j)+\alpha} \tag{3-3}$$

其中，α 是一个可调节的参数。α 的含义是当相似度为 0.5 时的文本向量距离值。

常用文本向量距离的表达式如下。

1）绝对值距离

$$\mathrm{Dis}(\boldsymbol{D}_i,\boldsymbol{D}_j)=\sum_{k=1}^{n}|w_{ik}-w_{jk}| \tag{3-4}$$

2）欧几里得距离

$$\mathrm{Dis}(\boldsymbol{D}_i,\boldsymbol{D}_j)=\sqrt{\sum_{k=1}^{n}(w_{ik}-w_{jk})^2} \tag{3-5}$$

3）切比雪夫距离

$$\mathrm{Dis}(\boldsymbol{D}_i,\boldsymbol{D}_j)=\max_{k}|w_{ik}-w_{jk}| \tag{3-6}$$

在很多情况下，直接计算文本向量的相似度比较困难，通常可以先计算文本向量的距离，然后再转换成文本向量的相似度。所以有时我们只谈论文本向量的距离，而没有提及文本向量的相似度，因为这二者是可以互相转换的。

3.2.3 向量模型的 Python 实现

Python 可以使用 jieba.analyse 提取句子级的特征词，格式如下。

```
jieba.analyse.extract_tags(sentence,topK=5,withWeight=True, allowPOS=())
```

参数说明：sentence 是需要提取的字符串，必须是 str 类型，不能是 list 类型；topK 为提取前多少个关键字；withWeight 为是否返回每个关键词的权重；allowPOS 是允许提取的词性，默认为 allowPOS=‘ns’,‘n’,‘vn’,‘v’，即提取地名、名词、动名词、动词。

例 3.7 简单的文本向量模型。

```
import jieba
import jieba.analyse
import collections
sen1='各级党委和政府必须适应我国发展进入新阶段、社会主要矛盾发生变化的必然要求，紧紧扭住新发展理念推动发展，把注意力集中到解决各种不平衡不充分的问题上'
sen2='要树立全面、整体的观念，遵循经济社会发展规律，重大政策出台和调整要进行综合影响评估，切实抓好政策落实，坚决杜绝形形色色的形式主义、官僚主义。'
sen3='要确保脱贫攻坚任务如期全面完成，集中兵力打好深度贫困歼灭战，政策、资金重点向“三区三州”等深度贫困地区倾斜，落实产业扶贫、易地搬迁扶贫等措施，严把贫困人口退出关，巩固脱贫成果。'
```

```
sen4 = '要建立机制,及时做好返贫人口和新发生贫困人口的监测和帮扶。要打好污染防治攻坚战,坚持方向不变、力度不减,突出精准治污、科学治污、依法治污,推动生态环境质量持续好转。'
sen5 = '要重点打好蓝天、碧水、净土保卫战,完善相关治理机制,抓好源头防控。我国金融体系总体健康,具备化解各类风险的能力。要保持宏观杠杆率基本稳定,压实各方责任。'
for item in [sen1,sen2,sen3,sen4,sen5]:
    keywords = jieba.analyse.extract_tags(item, topK = 10, withWeight = True, allowPOS = ('n','nr','ns'))
    counter = collections.Counter(keywords)
    print(counter) print(" ================================================== ")
```

程序运行结果如图 3.10 所示。

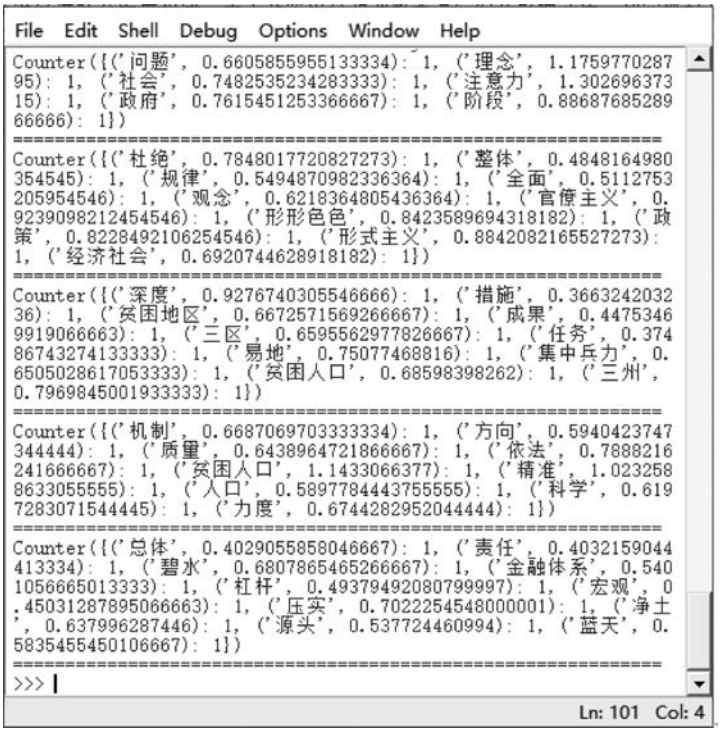


图 3.10　向量模型的结果

例 3.8　利用 Python 计算中文文本的相似度。在 D://python3sy/下创建两个文本文件 data1.txt 和 data2.txt 作为语料库,再创建一个文本文件 data3.txt 作为需要对比的文件,分别计算 data3.txt 与 data1.txt 和 data2.txt 的相似度。

基本思路：jieba 进行分词,整理为指定格式,gensim 库将要对比的文档通过 doc2bow 转换为稀疏向量,再通过 models 中的 TF-IDF 将语料库进行处理,特征值和稀疏矩阵相似度建立索引,最后得到相似结果。

```
import jieba
from gensim import corpora,models,similarities
from collections import defaultdict
#用于创建一个空的字典,在后续统计词频可清理频率少的词语
#1. 读取文档
doc1 = "D://python3sy/data1.txt"
doc2 = "D://python3sy/data2.txt"
d1 = open(doc1,encoding = 'GBK').read()
d2 = open(doc2,encoding = 'GBK').read()
#2. 对要计算的文档进行分词
data1 = jieba.cut(d1)
data2 = jieba.cut(d2)
#3. 对分词完的数据整理为指定格式
data11 = ""
for i in data1:
    data11 += i + " "
data21 = ""
for i in data2:
     data21 += i + " "
documents = [data11,data21]
texts = [[word for word in document.split()] for document in documents]
#4. 计算词语的频率
frequency = defaultdict(int)
for text in texts:
      for word in text:
          frequency[word] += 1
#5. 对频率低的词语进行过滤
texts = [[word for word in text if frequency[word]> 2] for text in texts]
#6. 通过语料库将文档的词语建立词典
dictionary = corpora.Dictionary(texts)
dictionary.save("./dict.txt")            #可以将生成的词典进行保存
#7. 加载要对比的文档
doc3 = "D://python3sy/data3.txt"
d3 = open(doc3,encoding = 'GBK').read()
data3 = jieba.cut(d3)
data31 = ""
for i in data3:
    data31 += i + " "
#8. 将要对比的文档通过 doc2bow 转换为稀疏向量
new_xs = dictionary.doc2bow(data31.split())
#9. 对语料库进一步处理,得到新语料库
corpus = [dictionary.doc2bow(text)for text in texts]
#10. 将新语料库通过 TF - IDF model 进行处理,得到 TF - IDF
tfidf = models.TfidfModel(corpus)
#11. 通过 token2id 得到特征数
featurenum = len(dictionary.token2id.keys())
#12. 稀疏矩阵相似度,从而建立索引
index = similarities.SparseMatrixSimilarity(tfidf[corpus],num_features = featurenum)
```

```
#13. 得到最终相似结果
sim = index[tfidf[new_xs]]
print("")
print("文本 data3.txt 与 data1.txt、data2.txt 相似度分别为: ")
print(sim)
```

程序运行结果如图 3.11 所示。

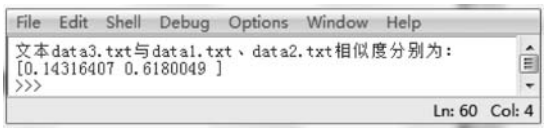


图 3.11　文本相似度运行结果

3.3　概率模型

在文本数据挖掘技术中，概率模型在描述文本数据方面有着广泛的应用，可以使用概率模型解决诸如对数据的预测、数据总体描述的问题。概率模型的最典型算法是朴素贝叶斯算法。

3.3.1　概率模型概述

在概率模型出现之前，文本挖掘主要停留在向量空间模型与统计语言模型上。向量空间作为一种文本的数值表示，其特点是概念简单，易于实现。但由于用向量空间表示的文档不可避免地会出现高维性、稀疏性等问题，这不仅带来了计算的负担，而且也加剧了文档数据中的歧义问题。理想的文本表示模型，应当可以实现文本在(潜)语义层的相似度计算。也就是即使两个文档没有相同的词汇，如果这两个文档的词汇属于同一语义范畴，仍然可以被赋予较高的相似度。为了自动抽取这种潜在语义空间，需要实现文本的低维潜在语义表示。潜在意义分析(Latent Semantic Indexing，LSI)的出现较好地实现了这个目标，是继向量空间表示后的一个改进。随着文本挖掘概率模型研究的深入，该方法被进一步赋予概率模型的解释与提升。

对于从很大的总体中抽取出的数据，或者可以被看作是从很大总体中抽取出的数据(如将要处理的文本数据看作从一个文档集中抽取出来的一个子集)，通过潜在的分布或密度函数描述它们是一种很基本的策略，也就是说在分析数据的时候可以假定这些总体中的数据是服从于某一个概率分布的。例如，经常使用标准正态分布假设一些现实世界中的一些事情的发生情况等。

很多预测问题经常会遇到这样的情况：对于一个未知的或待估计的变量，使用其他变量对其进行预测。文本数据挖掘中很多的建模问题都属于这一类。还有许多的建模问题是“描述性”的，目标是给出对数据的描述或总结。如果现有数据是完整的(如某一类文本的全部)，那么就不存在任何推理概念，目标就是简化描述。另外，如果现有的数据是一个样本或带有误差的测量值(因而如果采集多次数据，可能会得到略微不同的值，如对于

某一事件发展过程的连续报道等),那么建模的目的实质上是一种推理——推理出“真实”或至少是比较好的模型结构。所以,在一般情况下,可以把要分析的数据假定为由一个潜在的概率函数产生的。也就是说,数据空间中的点服从于某一个概率模型。

3.3.2 概率建模方法

在各种概率模型中,最重要的就是推断(Inference)问题,其中包含了两方面的内容。

(1) 在给定独立同分布的可测样本集 $\mathcal{X}$,估计出模型的一系列最佳参数 ϑ。

(2) 在给定独立同分布的可测样本集 $\mathcal{X}$,计算一个新可测样本 x^* 的概率 $p(x^*|\mathcal{X})$。

下面分别具体介绍在以上假设下的 3 个典型框架的参数推断方法。

1. 基于最大似然估计的概率模型

最大似然估计方法(Maximum Likelihood Estimation,MLE)求解模型参数的依据如下。

$$L(\vartheta \mid \mathcal{X}) = p(\vartheta \mid \mathcal{X}) = p\left(\bigcap_{x\in\mathcal{X}}\{X = x \mid \vartheta\}\right) = \prod_{x\in\mathcal{X}} p(x \mid \vartheta) \tag{3-7}$$

对于全体可测数据集 $\mathcal{X}$ 用所有样本的联合概率进行建模。其中,$L(\vartheta|\mathcal{X})$就是似然函数,最终的表达式是参数的条件概率乘积形式。由于概率值是 0～1 的一个很小的数值,连续乘法操作可能导致计算机的数值误差,因此实际应用中采用对数化的形式求解。

$$\vartheta^*_{\mathrm{ML}} = \arg\max_{\vartheta} \log_2\left(\prod_{x\in\mathcal{X}} p(x \mid \vartheta)\right) = \arg\max_{\vartheta} \sum_{x\in\mathcal{X}} \log_2 p(x \mid \vartheta) \tag{3-8}$$

式(3-8)在满足连续可微的条件下,直接采用求导优化就可以求解,如果不满足连续可微的条件,就要采用其他更复杂的近似算法求解模型(如 EM 算法)。在获得模型参数后,就可以对新样本 x^* 出现的概率进行预测。

$$p(x^* \mid \mathcal{X}) = \int_{\vartheta\in\Theta} p(x^* \mid \vartheta) p(\vartheta \mid \mathcal{X}) \mathrm{d}\vartheta \tag{3-9}$$

其中,Θ 是 ϑ 的参数空间。此时,代入求出的最大似然估计参数 $\vartheta^*_{\mathrm{ML}}$ 到式(3-9)中,由于 $\vartheta^*_{\mathrm{ML}}$ 已为常数,所以不参与积分运算,可得近似解。

$$p(x^* \mid \mathcal{X}) \approx \int_{\vartheta\in\Theta} p(x^* \mid \vartheta^*_{\mathrm{ML}}) p(\vartheta \mid \mathcal{X}) \mathrm{d}\vartheta = p(x^* \mid \vartheta^*_{\mathrm{ML}}) \tag{3-10}$$

2. 基于最大后验的概率模型

最大似然估计方法对于数据的建模全部依据当前观测的情况,所以容易产生拟合现象。为解决此问题,基于最大后验概率(Maximum A Posteriori,MAP)的参数估计方法引入了先验概率 $p(\vartheta)$进行了弥补。由此,其目标函数变为

$$\vartheta^*_{\mathrm{MAP}} = \arg\max_{\vartheta} p(\vartheta \mid \mathcal{X}) \tag{3-11}$$

然后,通过使用贝叶斯定理可以进一步推导得出含有 $p(\vartheta)$的表达式。

$$\begin{aligned} \vartheta^*_{\mathrm{MAP}} &= \arg\max_{\vartheta} \frac{p(\mathcal{X} \mid \vartheta) p(\vartheta)}{p(\mathcal{X})} \\ &= \arg\max\left\{\sum_{x\in\mathcal{X}} \log_2 p(x \mid \vartheta) + \log_2 p(\vartheta)\right\} \end{aligned} \tag{3-12}$$

从式(3-12)中可以发现，在贝叶斯方法框架下，此时的待估计 ϑ 本身已经成为一个随机变量。最后的参数求解与对新观测的数据的 MLE 类似。

$$p(x^* \mid \chi)=\int_{\vartheta\in\Theta} p(x^* \mid \vartheta^*_{\mathrm{MAP}})p(\vartheta \mid \chi)\mathrm{d}\vartheta = p(x^* \mid \vartheta^*_{\mathrm{MAP}}) \tag{3-13}$$

3. 基于贝叶斯网络的概率模型

虽然 MAP 方法考虑到了先验概率的影响，但是从对于新样本的预测过程来看，$\vartheta^*_{\mathrm{MAP}}$ 仅仅还是一个点估计，并没有真正地将其当成具有某种分布的随机变量显示的形式处理。为此，贝叶斯估计方法进一步对 MAP 方法进行了扩展，真正把待估参数的先验分布考虑进来。

$$p(\vartheta \mid \chi)=\frac{p(\chi \mid \vartheta)p(\vartheta)}{p(\chi)}=\frac{p(\chi \mid \vartheta)p(\vartheta)}{\int_{\vartheta\in\Theta} p(\chi \mid \vartheta)p(\vartheta)\mathrm{d}\vartheta} \tag{3-14}$$

通过式(3-14)可以发现，此时模型的求解要考虑到参数空间的全部信息。由此，特别是分母中的积分运算给模型参数的求解带来较大的困难。一般的最大似然参数估计策略将很难运用，为此需要更复杂的参数估计方法(如变分法、马尔可夫链蒙特卡洛方法等)。进一步，对于新观测样本的预测形式也随之变为

$$p(x^* \mid \chi)=\int_{\vartheta\in\Theta} p(x^* \mid \vartheta^*)p(\vartheta \mid \chi)\mathrm{d}\vartheta=\int_{\vartheta\in\Theta} p(x^* \mid \vartheta^*)\frac{p(\chi \mid \vartheta)p(\vartheta)}{p(\chi)}\mathrm{d}\vartheta \tag{3-15}$$

3.3.3　文本信息检索中的概率模型

信息检索(Information Retrieval)是指信息按一定的方式组织起来，并根据用户的需要找出有关信息的过程和技术。概率模型基于如下思想：根据用户的检索 q，可以将文档集 D 中的所有文档分为两类，一类与检索需求 q 相关(集合 R)，另一类与检索需求不相关(集合 $\overline{R}$)。在同一类文档中，各个索引项具有相同或相近的分布；而属于不同类的文档中，索引项应具有不同的分布。因此，通过计算文档中所有索引项的分布，就可以判定该文档与检索的相关度。

1. 概率信息检索的基本假设前提和理论

(1) 相关性独立原则。文档对一个检索式的相关性与文档集合中的其他文档是独立的。

(2) 词的独立性。标引词和检索式中词与词之间是相互独立的。

(3) 文档相关性是二值的，即只有相关和不相关两种。

(4) 概率排序原则。该原则认为，如果一个检索系统对用户的每个检索提问的反应是将文档集合中的文档按相关性递减的顺序排列，那么系统的总体效果将是最好的。

(5) 贝叶斯(Bayes)定理，用公式表示为

$$p(R \mid D_i)=\frac{p(D_i \mid R)p(R)}{p(D_i)} \tag{3-16}$$

其中，R 表示相关文档集合；D_i 为某一文档；$p(R \mid D_i)$表示文档 D_i 与查询相关的概率。

2. 概率信息检索的基本过程

对于检索 q，任意文档 D_i 与其相关和不相关的概率分别表示为 $p(R|D_i)$和 $p(\overline{R}|D_i)$。根据贝叶斯公式可得

$$p(R \mid D_i)=\frac{p(D_i \mid R)p(R)}{p(D_i)} \tag{3-17}$$

$$p(\overline{R} \mid D_i)=\frac{p(D_i \mid \overline{R})p(\overline{R})}{p(D_i)} \tag{3-18}$$

式(3-17)中的后两项只与检索需求有关，而与每个文档 D_i 无关，可以不计算，则将计算 $p(R|D_i)$转化为计算 $p(D_i|R)$。同理，对 $p(\overline{R}|D_i)$的计算也将转化为对 $p(D_i|\overline{R})$的计算。

由于索引项的数目很大，因此常常在计算中引入一些假设，以简化计算。对应不同的假设，就形成了 3 种不同的经典概率模型，分别是二元独立模型(Binary Independent Model)、二元一阶相关模型(Binary First Order Dependent Model)和二维泊松分布模型(2-Poisson Independent Model)。

1) 二元独立模型

二元独立模型对文档中索引项的分布做了以下两个假设。

(1) 二元属性取值假设。任意一个文档 D_i 可以表示为 $D_i(x_{i1},x_{i2},\cdots,x_{ik},\cdots,x_{im})$，其中二元随机变量 x_{ik} 表示索引项 w_k 是否在该文档中出现，如果出现，$x_{ik}=1$；否则 $x_{ik}=0$。

(2) 索引项独立性假设。在一个文档中，任意一个索引项的出现与否不会影响到其他索引项的出现，它们之间相互独立。

根据以上两个假设，有

$$p(D_i \mid R)=p(x_{i1},x_{i2},\cdots,x_{im} \mid R)=\prod_{k=1}^{m} p(x_{ik} \mid R) \tag{3-19}$$

$$p(D_i \mid \overline{R})=p(x_{i1},x_{i2},\cdots,x_{im} \mid \overline{R})=\prod_{k=1}^{m} p(x_{ik} \mid \overline{R}) \tag{3-20}$$

至此，我们可以定义文档 D_i 与检索 q 的相关度为

$$\mathrm{Sim}(q,D_i)=\frac{p(R \mid D_i)}{p(\overline{R} \mid D_i)} \tag{3-21}$$

相关度越大，表示文档 D_i 与检索 q 越相关。

根据贝叶斯定理，将式(3-17)和式(3-18)代入式(3-21)，有

$$\mathrm{Sim}(q,D_i)\approx\frac{p(D_i \mid R)}{p(D_i \mid \overline{R})} \tag{3-22}$$

假设各特征项 $x_{i1},x_{i2},\cdots,x_{im}$ 相互独立，则式(3-22)可改写为

$$\mathrm{Sim}(q,D_i)=\frac{\prod\limits_{g(D_i)=1} p(x_{ik} \mid R)\prod\limits_{g(D_i)=0} p(\overline{x}_{ik} \mid R)}{\prod\limits_{g(D_i)=1} p(x_{ik} \mid \overline{R})\prod\limits_{g(D_i)=0} p(\overline{x}_{ik} \mid \overline{R})} \tag{3-23}$$

其中，$g(D_i)=1$ 或 0 分别表示与索引项相关文档和无关文档；$p(x_{ik}|R)=|R_k|/|R|$，$|R_k|$为包含 x_{ik} 的相关文档数，$|R|$为相关文档的总数；$p(x_{ik}|\bar{R})=(n_k-|R_k|)/(N-|R|)$，$n_k$ 为包含 x_{ik} 的无关文档数，N 为文档的总数。

为了简化计算，对式(3-23)右边取对数并忽略恒定不变的因子，得

$$\mathrm{Sim}(D_i,q)\approx\sum_{k=1}^{m}\lambda_{ik}\left[\log_2\frac{p(x_{ik}\mid R)}{1-p(x_{ik}\mid\bar{R})}\right]+\left[\log_2\frac{1-p(x_{ik}\mid\bar{R})}{p(x_{ik}\mid\bar{R})}\right] \tag{3-24}$$

其中，λ_{ik} 为相关因子。

如果能够预先得到一定数量的带有相关性标记的文档，则式(3-24)中需要确定的概率可以通过最大似然估计方法来确定。

2) 二元一阶相关模型

索引项独立性假设只是为了数学上计算处理方便，并不符合实际情况。可以看到，一些索引项在文档中的出现往往并不是相互独立的，而是存在某种关系，如某些索引项经常会同时出现在一个文档中。因此，要想获得更好的检索结果，就必须考虑各个索引项之间的相互依赖关系，这就是建立二元相关模型的背景。二元一阶相关模型不承认假设(2)，承认假设(1)是为了保证文档表示的一致性，从而对 $p(D_i|R)$和 $p(D_i|\bar{R})$的计算与二元独立模型不同。这里我们主要研究 $p(D_i|R)$的计算，即在相关文档中各个索引项的分布。同理也可以计算出 $p(D_i|\bar{R})$。

为了实际地表示文档中各个索引词的相互关系，我们可以假设在相关文档中，各个索引项之间存在统计相关性。统计相关性不同于逻辑相关性，它是两个或多个索引项在文档中出现频率之间所表现出的一种相关性，而并不考虑各个索引项在文档中出现的先后次序。根据统计相关性有

$$\begin{aligned}p(D_i\mid R)&=p(x_{i1},x_{i2},\cdots,x_{im}\mid R)=p(x_{i1}\mid R)p(x_{i2}\mid x_{i1},R)\cdots\\&\quad p(x_{im}\mid x_{i1},x_{i2},\cdots,x_{im-1},R)\end{aligned} \tag{3-25}$$

式(3-25)尽管可以准确地表示各个索引项之间的相关性，然而它包含的参数数目非常大。一种简化计算的假设是：假设对于每一个索引项 w_k，有且只有一个索引项 $w_{k(j)}$，使索引项 w_k 与其余索引项之间条件独立，该假设称为一阶相关性假设。根据该假设，我们必须找到分布 $p(D_i|R)$ 的一个近似分布 $p'(D_i|R)$，满足如下分解性质。

$$\begin{aligned}p'(D_i\mid R)&=p(x_{i1},x_{i2},\cdots,x_{k(j)},\cdots,x_{im}\mid R)\\&=p(x_{i1}\mid x_{ik(1)},R)\ p(x_{i2}\mid x_{ik(2)},R)\cdots p(x_{im}\mid x_{ik(m)},R)\end{aligned} \tag{3-26}$$

该近似分布 $p'(D_i|R)$与分布 $p(D_i|R)$越接近越好，这可以通过它们之间的交叉熵来表示。Chow 等给出了一个最大生成树(Maximum Spanning Tree)算法，并证明了该算法可以求得符合上述要求的近似分布 $p'(D_i|R)$。至此，可以将 $p(D_i|R)$近似地表示为

$$\begin{aligned}p(D_i\mid R)&\propto\log_2 p(D_i\mid R)\approx\log_2 p'(D_i\mid R)\\&=\sum_{j=1}^{m}[x_{ij}\log_2 r_{ij}+(1-x_{ij})\log_2(1-r_{ij})]+\\&\quad\sum_{j=1}^{m}\left[x_{ik(j)}\log_2\frac{1-s_{ij}}{1-r_{ij}}+x_{ij}x_{k(j)}\log_2\frac{s_{ij}(1-r_{ij})}{r_{ij}(1-s_{ij})}\right]+C\end{aligned} \tag{3-27}$$

其中，$s_{ij}=p'(x_{ij}=1|x_{ik(j)}=1,R)$；$r_{ij}=p'(x_{ij}=1|x_{ik(j)}=0,R)$。

3）二维泊松分布模型

二维泊松分布模型的基本思想来源于如下的实验观察，文档中的特征词可分为两类：一类特征词与表达文档的主题相关，称为内容词(Content-Bearing Words)；另一类只完成一些语法功能，称为功能词(Functional Words)。统计实验发现：功能词在文档中的分布与内容词不同，前者出现的频率比较稳定，其波动情况可以近似为泊松分布，即如果用 x 表示某个功能词在文档中的出现频率，则

$$p(x)=\frac{u^x}{x!}e^{-x} \tag{3-28}$$

其中，u 为该分布的均值，表示该功能词的平均出现频率。

可见内容词在文档中的出现频率在一定意义上反映了一个文档的主题。因此，提出二维泊松分布假设：根据一个内容词，可以将文档从主题上分为两类，同时该内容词在两类文档中的出现频率也会很不相同，一类文档的主题与该内容词相关，那么该内容词在其中的出现频率应该比较高，其波动特征可以用一个泊松分布表示；而另一类文档的主题与该内容词不相关，所以该内容词在其中的出现频率应该比较低，其波动特征也可以用一个泊松分布表示。总之，一个内容词在文档中的出现频率 x 可以表示为如下两个泊松分布的加权组合。

$$p(x)=\pi\frac{u^x}{x!}e^{-x}+(1-\pi)\frac{v^x}{x!}e^{-x} \tag{3-29}$$

其中，u 和 v 分别为内容词在两类文档中出现频率的均值；π 表示任意一个文档属于第一类的概率，该假设称为二维泊松分布假设。只要将所有的索引项看作是内容词(其实，在实际的检索中，索引项一般都是内容词)，它们也满足二维泊松分布模型。与二元独立模型相比，二维泊松分布模型的不同在于不承认假设(1)，其余都相同，所以可以与二元独立模型一样定义如下相关性排序函数。

$$\mathrm{Sim}(q,D_i)\approx\frac{p(D_i\mid R)}{p(D_i\mid\overline{R})}=\frac{p(x_{i1},x_{i2},\cdots,x_{im}\mid R)}{p(x_{i1},x_{i2},\cdots,x_{im}\mid\overline{R})} \tag{3-30}$$

根据二维泊松分布假设，得

$$\mathrm{Sim}(q,D_i)\approx\frac{\prod_{k=1}^{m}u_{ik}^{x_{ik}}e^{-x_{ik}}/x_{ik}!}{\prod_{k=1}^{m}v_{ik}^{x_{ik}}e^{-x_{ik}}/x_{ik}!} \tag{3-31}$$

对其取对数，并去掉不变量，可得

$$\mathrm{Sim}(q,D_i)\approx\sum_{k=1}^{m}(v_{ik}-u_{ik})+\sum_{k=1}^{m}x_{ik}\log_2(u_{ik}-v_{ik}) \tag{3-32}$$

对于二维泊松分布模型，一般采用力矩法(Method of Moments)计算估计该模型的所有参数。

3.3.4 概率模型的 Python 实现

概率建模最基础的级别是简单的概率分布。以语言建模为例，概率分布就是人们常

说的每个单词出现频率的分布。Pomegranate 是 Python 上的图模型与概率建模工具包，为 k-means、混合模型、隐马尔可夫模型、贝叶斯网络模型、朴素贝叶斯/贝叶斯分类器等模型提供模型凝合、结构化学习和推断过程的修正，并重点关注与处理数据缺失值。它抽象了概率图模型的底层细节，可以方便地基于 API 进行上层应用建模。Pomegranate 的重心是从训练模型的定义中抽象出其复杂性，允许用户专注于为自己的应用选择合适的模型，而不用受到对底层算法理解不足的限制。Pomegranate 的这一重心包括从数据集中收集充分的统计数据，作为一种训练模型的策略。该方法使用了很多有用的学习策略，如 out-of-core 学习、小批量学习和半监督学习，用户无须考虑如何分割数据或修改算法，算法自己处理这些任务。Pomegranate 用 Cython 构建以加速计算，同时内置多线程并行处理方法，Pomegranate 可匹配甚至优于其他类似算法的实现。

例 3.9　利用 Pomegranate 库建模。

```
# - * - coding: UTF - 8 - * -
from pomegranate import *
if __name__ == '__main__':
guest = DiscreteDistribution({'A': 1./3, 'B': 1./3, 'C': 1./3})
prize = DiscreteDistribution({'A': 1./3, 'B': 1./3, 'C': 1./3})
monty = ConditionalProbabilityTable([['A', 'A', 'A', 0.0],['A', 'A', 'B', 0.5],
['A', 'A', 'C', 0.5],['A', 'B', 'A', 0.0],['A', 'B', 'B', 0.0],['A', 'B', 'C', 1.0],
['A', 'C', 'A', 0.0],
['A', 'C', 'B', 1.0],['A', 'C', 'C', 0.0],['B', 'A', 'A', 0.0],['B', 'A', 'B', 0.0],
['B', 'A', 'C', 1.0],['B', 'B', 'A', 0.5],['B', 'B', 'B', 0.0],['B', 'B', 'C', 0.5],
['B', 'C', 'A', 1.0],['B', 'C', 'B', 0.0],
['B', 'C', 'C', 0.0],['C', 'A', 'A', 0.0],['C', 'A', 'B', 1.0],['C', 'A', 'C', 0.0],
['C', 'B', 'A', 1.0],
['C', 'B', 'B', 0.0],['C', 'B', 'C', 0.0],['C', 'C', 'A', 0.5],['C', 'C', 'B', 0.5],
['C', 'C', 'C', 0.0]],
[guest, prize])
s1 = Node(guest, name = "guest")
s2 = Node(prize, name = "prize")
s3 = Node(monty, name = "monty")
model = BayesianNetwork("Monty Hall Problem")
model.add_states(s1, s2, s3)
model.add_edge(s1, s3)
model.add_edge(s2, s3)
model.bake()
print("")
print(model.probability([['A', 'A', 'A'],['A', 'A', 'B'],['C', 'C', 'B']]))
print(model.predict([['A', 'B', None],['A', 'C', None],['C', 'B', None]]))
print(model.predict([['A', 'B', None],['A', None, 'C'],[None, 'B', 'A']]))
```

程序运行结果如图 3.12 所示。

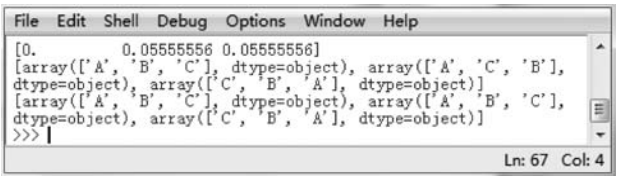


图 3.12 Pomegranate 库建模运行结果

3.4 概率主题模型

主题模型(Topic Model)是一种概率生成模型,主要包括概率潜在语义索引和潜在狄利克雷分布(Latent Dirichlet Allocation,LDA)。主题模型的应用广泛,涉及很多方面,尤其是在自然语言处理中。主题模型在机器学习和自然语言处理等领域是用来在一系列文档中发现抽象主题的一种统计模型。

3.4.1 概率主题模型概述

主题模型起源于 1990 年 S. C. Deerwester 等提出的潜在语义索引(Latent Semantic Indexing,LSI)的工作,其原理是利用奇异值分解技术实现文本维度的压缩,使压缩后的潜在语义空间能够反映不同特征词之间的语义关系,但 LSI 不是概率模型,因而 LSI 并不算是主题模型。在 LSI 的基础之上,1999 年 T. Hofmann 提出概率潜在语义索引(Probabilistic LSI,PLSI),该模型通过引入概率模型,显式地对文本及其隐含主题进行建模,是第一个真正意义上的主题模型,但它仍不是完整的概率模型,其参数只与训练文本相关,很难直接用于对新文本进行建模。2003 年,D. M. Blei 等又在 PLSI 的基础上提出了 LDA 模型,该模型集成了 PLSI 的优点,同时也克服了 PLSI 的理论缺陷,被广泛应用于诸多领域。

主题模型是对文字隐含主题进行建模的方法,主题可以定义为文档集中具有相同词境的词的集合模式。例如,将“健康”“病人”“医院”“药品”等词汇集合成“医疗保健”主题,将“农场”“玉米”“小麦”“棉花”“播种机”“收割机”等词汇集合成“农业”主题。主题模型克服了传统信息检索中文档相似度计算方法的缺点,并且能够在海量互联网数据中自动寻找出文字间的语义主题。主题模型自动分析每个文档,统计文档内的词语,根据统计的信息来断定当前文档含有哪些主题,以及每个主题所占的比例各为多少。

3.4.2 PLSA 概率主题模型

PLSA(Probabilistic Latent Semantic Analysis)利用概率手段对文档和词汇的共现现象进行建模,引入了表示文档主题的隐含变量模拟文档中每个词汇的生成过程。

1. PLSA 模型原理

给定文档集合 $D=\{D_1,D_2,\cdots,D_N\}$ 和其中包含的词汇集合 $W=\{w_1,w_2,\cdots,w_M\}$,可

以构造一个 $N\times M$ 的文档-词频矩阵 $\mathbf{df}_{ij}=\{n_{ij}\mid n_{ij}=n(D_i\mid w_j),1\leqslant i\leqslant N,\ 1\leqslant j\leqslant M\}$，其中，$n(D_i\mid w_j)$表示词 w_j 在文档 D_i 中出现的频率。矩阵的行为文档向量，列为词向量。

具体建模过程可以用如下概率图进行表达，对于文档集中的每个词 w_j 的产生，首先以概率 $p(D_i)$选择一个文档 D_i，然后根据 $p(z_k\mid D_i)$选择一个潜在的主题 z_k，最后通过 $p(w_j\mid z_k)$生成最终的词 w_j。PLSA 概率主题模型示意图如图 3.13 所示。

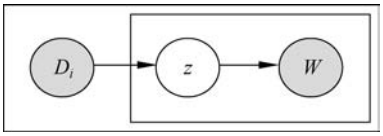


图 3.13　PLSA 主题模型示意图

为了进行模型求解，需要得到其联合概率形式如下。

$$p(D_i,w_j)=p(D_i)p(w_j\mid D_i) \tag{3-33}$$

$$p(w_j\mid D_i)=\sum_{k=1}^{k}p(z_k\mid D_i)p(w_j\mid z_k) \tag{3-34}$$

其中，$p(D_i)$表示文档 D_i 出现的概率；$p(z_k\mid D_i)$表示文档在潜在主题上的概率分布；$p(w_j\mid z_k)$表示潜在主题在词上是概率分布。

2. 基于 EM 算法的 PLSA 模型求解

在得到 PLSA 模型的联合概率表示后，利用最大似然法对模型求解。但由于其似然函数无法直接利用求导的数值解法进行最优化估计，所以这里采用 EM(Expectation-Maximization)算法求解。

EM 算法提供的是一种通用计算框架，主要适用于对含有隐含变量以及缺失值参数估计的情形。其本质是对于最大似然函数进行直接的参数估计较为困难，从而转化为一种参数迭代求解的方式。EM 算法主要包括两个步骤，一个是 E 步骤，主要实现对隐含变量的估计，依据的原理是 Jensen 不等式，使其最大似然函数在已估其他参数下的数值变为最大下界。对应的 PLSA 模型 E 步骤如下。

$$p(z_k\mid D_i,w_j)=\frac{p(z_k)p(D_i\mid z_k)p(w_j\mid z_k)}{\sum_{l=1}^{K}p(z_l)p(D_i\mid z_l)p(w_j\mid z_l)} \tag{3-35}$$

另一个是 M 步骤，由 E 步骤得到的隐含变量的估计值，重新对其他参数进行最大似然计算。对应的 PLSA 模型 M 步骤如下。

$$p(w_j\mid z_k)=\frac{\sum_{i=1}^{N}n(D_i,w_j)p(z_k\mid D_i,w_j)}{\sum_{m=1}^{M}\sum_{i=1}^{N}n(D_i,w_m)p(z_k\mid D_i,w_m)} \tag{3-36}$$

$$p(z_k\mid D_i)=\frac{\sum_{j=1}^{M}n(D_i,w_j)p(z_k\mid D_i,w_j)}{\sum_{i=1}^{N}\sum_{m=1}^{M}n(D_i,w_m)p(z_k\mid D_i,w_m)} \tag{3-37}$$

$$p(z_k)=\frac{\sum_{i=1}^{N}\sum_{j=1}^{M}n(D_i,w_j)p(z_k\mid D_i,w_j)}{\sum_{i=1}^{N}\sum_{j=1}^{M}n(D_i,w_j)} \tag{3-38}$$

对于式(3-35)～式(3-38)反复进行迭代直至收敛,就可以得到最终的 $p(z_k\mid D_i)$和 $p(w_j\mid z_k)$分布。

但是在 PLSA 模型在对文档集的主题类别表示式中,仍有 $K\times N$ 的矩阵,其中 K 是潜在主题的数目,可以由用户设定; N 是文档集 D 中所含文档数目。因此,随着文档集数目的增加,该矩阵的大小也在线性增加,存在过度拟合问题。

3. PLSA 模型分析

PLSA 模型的主要优缺点如下。

1) 主要优点

(1) 可以对大规模文档进行很好的低维表示。

(2) 在一定程度上,可以处理一词多义(多义词)与多词一义问题(同义词)。

(3) 有坚实的概率统计理论基础。

2) 主要缺点

(1) 模型估计的参数和文档的规模呈线性增长。

(2) 无法有效地对新文档进行相应的模型嵌入。

(3) 容易产生过度拟合。

3.4.3 LDA 概率主题模型

LDA 概率主题模型是基于贝叶斯模型的主题模型,挖掘文档中所隐含的主题信息,使用户或读者快速了解文档的信息。之所以称之为隐含,是因为主题在这个模型中是不必求出的变量,隐藏在主题-特征词概率分布和文档-主题概率分布中,使用狄利克雷分布求解最终概率分布,确定潜在的主题。LDA 的主要目的就是通过无指导的学习方式,从大量的文档中挖掘出隐含的主题。

1. LDA 概率主题模型概述

LDA 概率主题模型有 3 层结构,从上到下分别为文档层、主题层、特征词层。实质就是利用文本的特征词的共现特征挖掘文本的主题,层次非常清晰,其结构如图 3.14 所示。

LDA 概率主题模型是一种文档主题生成模型,也称为一个 3 层贝叶斯概率模型。所谓生成模型,就是我们认为一篇文章的每个特征词都是通过“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语”这样一个过程得到的。

2. LDA 概率主题模型原理

LDA 概率主题模型的原理是: 整个文档集是主题的概率分布,每个主题又是特征词

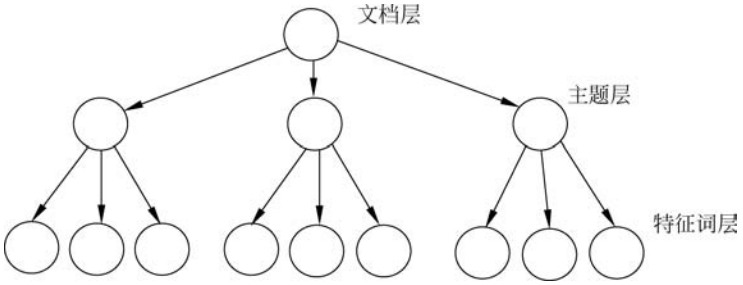


图 3.14　LDA 模型结构示意图

的概率分布。所以，有如下概率公式。

$$p(w_j \mid D_i) = \sum_{k=1}^{K} p(w_j \mid T_k) p(T_k \mid D_i) \tag{3-39}$$

其中，$p(w_j \mid D_i)$表示特征词 w_j 出现在文档 D_i 中的概率，此概率值为特征词的概率与主题特征词概率的乘积，即 w_j 在主题 T_k 中出现的概率与主题 T_k 在文档 D_i 中出现的概率的乘积；K 为主题的个数。

LDA 概率主题模型原理可以用矩阵的形式通俗易懂地表现出来，将整个文档看作是文档特征词矩阵，可以分解成文档主题矩阵和主题特征词矩阵，图 3.15 展示了三者之间的关系。

图 3.15　LDA 的矩阵表示

图 3.15 中，文档-特征词矩阵表示每个文档关于特征词的概率分布，文档-主题矩阵表示每个文档关于主题的概率分布，主题-特征词矩阵表示每个主题关于特征词的概率分布。对于已知的文档，文档-特征词矩阵可以根据第 2 章讲到的 TF-IDF 得到。

3. LDA 生成过程

对于文档集 D 中的每个文档，LDA 模型生成文档过程的思想如下。

(1) 对于每个文档，从主题分布中抽取一个主题。

(2) 从上述被抽到的主题所对应的单词分布中抽取一个单词。

(3) 重复上述过程直至遍历文档中的每一个单词。

具体流程如图 3.16 所示。这里假设生成 N 个文档，包含 K 个主题。

首先是生成主题-特征词多项分布，此多项分布是服从 β 的狄利克雷先验分布。再生成文档-主题分布，此分布也是一个多项分布，服从参数为 α 的狄利克雷先验分布，狄利克雷分布是多维的贝塔分布。贝塔分布的密度函数如下。

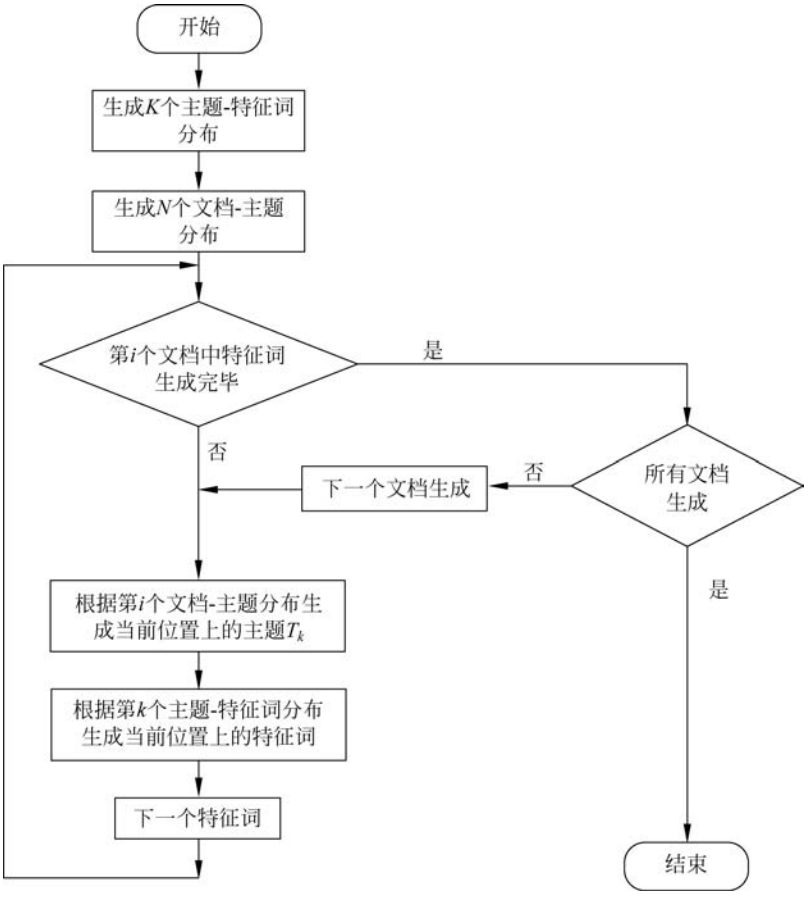


图 3.16　LDA 主题文档生成算法流程图

$$
\begin{aligned}
f(p,\alpha,\beta) &= \frac{p^{\alpha-1}(1-p)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\mathrm{d}u} \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}p^{\alpha-1}(1-p)^{\beta-1} \\
&= \frac{1}{\mathrm{B}(\alpha,\beta)}p^{\alpha-1}(1-p)^{\beta-1}
\end{aligned}
\tag{3-40}
$$

其中，$\mathrm{B}(\alpha,\beta)$表示参数为(α,β)的贝塔分布；p 表示特征词出现在主题中的概率或主题出现在文档中的概率。K 维的狄利克雷分布如式(3-41)所示。

$$
\mathrm{Dirichlet}(\vec{p},\vec{\alpha}) = \frac{\Gamma\left(\sum_{k=1}^{K}\alpha_k\right)}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\prod_{k=1}^{K}p_k^{\alpha_k-1} \tag{3-41}
$$

贝塔分布是狄利克雷分布在二维时的特殊形式。LDA 模型具体的实现就是确定出参数(α,β)，过程如图 3.17 所示。

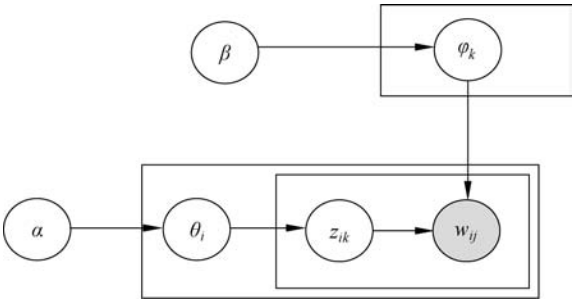


图 3.17　LDA 模型示意图

图 3.17 中，α 的大小体现主题之间的相关性；β 是主题自身的概率分布；θ_i 表示第 i 个文档的主题分布；φ_k 表示第 k 个主题的特征词分布；z_{ik} 表示第 i 个文档的第 k 个主题；w_{ij} 表示第 i 个文档的第 j 个特征词。其中，$\vec{\varphi}\sim\text{Dirichlet}(\vec{\alpha})$，$\vec{\theta}\sim\text{Dirichlet}(\vec{\beta})$。

LDA 主题模型的建模过程，主要是在文本集中训练出主要参数 α 和 β 的大小。训练参数的方法有很多种，主要是由 EM 推断和 Gibbs 抽样学习得到，这里不再赘述。

3.4.4　LDA 概率主题模型的 Python 实现

例 3.10　LDA 简单模型的实现。为了使问题简单化，取 D://python3sy/下的文件 test1.txt，建立主题模型。源代码如下。

```
import codecs                                  #主题模型
from gensim import corpora
from gensim.models import LdaModel
from gensim import models
from gensim.corpora import Dictionary
te = []
fp = codecs.open('D://python3sy/test1.txt','r')
for line in fp:
    line = line.split(',')
    te.append([ w  for w in line ])
print ('输入文本数量: ',len(te))
dictionary = corpora.Dictionary(te)
corpus = [ dictionary.doc2bow(text) for text in te ]
tfidf = models.TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]
print(list(corpus_tfidf))                      #输出词的 tfidf
print(list(corpus))                            #输出文本向量空间
lda = LdaModel(corpus = corpus, id2word = dictionary, num_topics = 20, passes = 100)
doc_topic = [a for a in lda[corpus]]
topics_r = lda.print_topics(num_topics = 20, num_words = 20)
topic_name = codecs.open('topics_result3.txt','w')
for v in topics_r:
```

```
        topic_name.write(str(v) + '\n')
fp2 = codecs.open('documents_result.txt','w')
for t in doc_topic:
    c = []
    c.append([a[1] for a in t])
    print(t)
    m = max(c[0])
        for i in range(0, len(t)):
        if m in t[i]:
            #print(t[i])
            fp2.write(str(t[i][0]) + '  ' + str(t[i][1]) + '\n')
            #输出模型类和概览
            break
```

程序运行结果如图 3.18 所示。

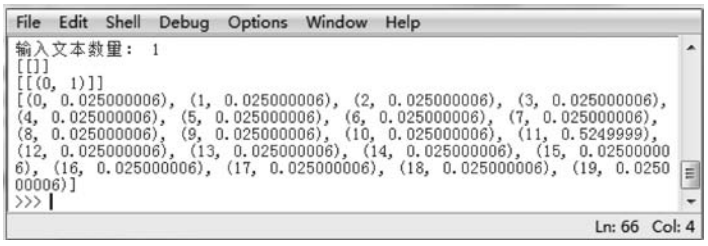


图 3.18　LDA 简单模型建模运行结果

习题 3

3-1　将第 2 章采集的数据利用 Python 进行乱码处理。

3-2　给出一段中文文本，利用 Python 的 re.split()函数进行分隔、断句。

3-3　给出一段中文文本，去除停用词，完成操作后将结果读出来。

3-4　简述 LDA 概率主题模型的生成过程。