

学 期 2022-2023

****

深度学习与自然语言处理

基于LDA的文本建模

|  |  |
| --- | --- |
| 院（系）名称 | 自动化科学与电气工程学院 |
| 专业名称 | 电子信息 |
| 学生姓名 | 苏士鹏 |
| 学号 | ZY2103306 |

2022年5月

目录

[一，作业要求 2](#_Toc102753867)

[二，LDA模型 2](#_Toc102753868)

[三，实验过程 3](#_Toc102753869)

[3.1 数据预处理 3](#_Toc102753870)

[3.2模型训练 3](#_Toc102753871)

[3.3模型测试 3](#_Toc102753872)

[四，实验结果 3](#_Toc102753873)

[五，总结分析 4](#_Toc102753874)

[附录 5](#_Toc102753875)

# 作业要求

从给定的语料库中均匀抽取200个段落（每个段落大于500个词）， 每个段落的标签就是对应段落所属的小说。利用LDA模型对于文本建模，并把每个段落表示为主题分布后进行分类。验证与分析分类结果。

# LDA模型

LDA（Latent Dirichlet Allocation）是一种文档主题生成模型，也称为一个三层贝叶斯概率模型，包含词、主题 和文档三层结构。而主题模型（Topic Model）能够识别在文档里的主题，并且挖掘语料里隐藏信息，并且在主题聚合、从非结构化文本中提取信息、特征选择等场景有广泛的用途。

用一句简单易懂的话来形容LDA就是，在LDA眼里，主题到词服从多项式分布，文档到主题服从多项式分布，所以在LDA模型中，文档生成方式如图2.1所示，1，对每一篇文档，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，重复上述步骤直到遍历整个文档。

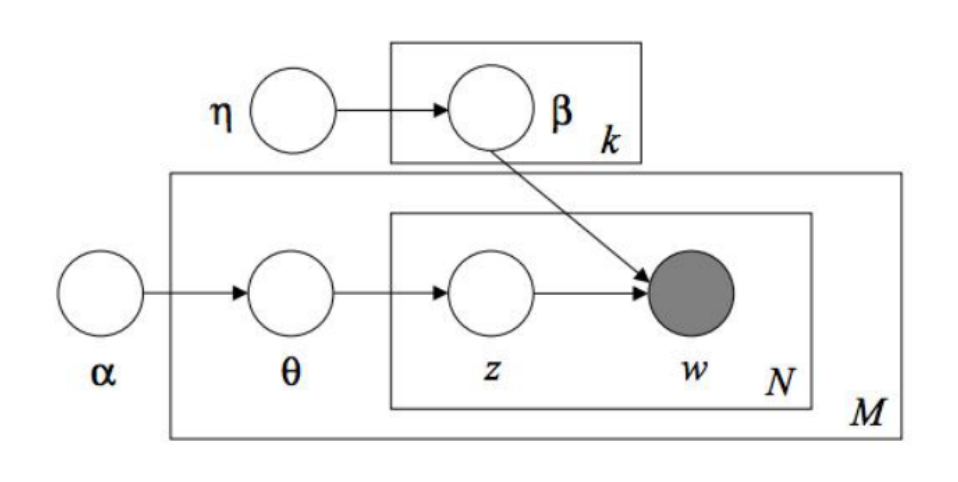


图2.1文档生成过程

因此可以得到整个模型中所有可见变量以及隐藏变量的[联合分布](https://baike.baidu.com/item/%E8%81%94%E5%90%88%E5%88%86%E5%B8%83)如式2.1所示

 （2.1）

最终得到一篇文章词分布的[最大似然估计](https://baike.baidu.com/item/%E6%9C%80%E5%A4%A7%E4%BC%BC%E7%84%B6%E4%BC%B0%E8%AE%A1)如式2.2所示

 （2.2）

# 实验过程

## 3.1 数据预处理

题目要求均匀抽取200个段落（每个段落大于500个词）， 每个段落的标签就是对应段落所属的小说。而语料库内有16本小说，无法完全均匀选择段落，这里每一篇小说选取12个段落，每个段落选521字，这样字数满足要求，段落为192段落，基本与要求吻合，分布较为均匀。

为了能够更好的使用语料库，与作业一一样需要先对语料库做处理，除去中文，英文标点符号以及多余字符，再使用jieba分词得到可以直接使用的语料库。

## 3.2模型训练

模型训练前对模型做初始化，为每篇文章的词赋予一个随机的主题，再计算每个主题被选中的概率，然后进行迭代训练，计算每篇文章选中各个主题的概率与该词语在每个主题中出现的概率相乘，得到该词出现的概率向量，更新每个文章有多少主题词，再更新每个主题的总词数，不断迭代更新，直到每篇文章中的各个主题的概率不再变化，则迭代结束，输出最终训练得到的每篇文章各个主题出现的概率和迭代次数，这样模型的训练过程就已经完成。最终得到，迭代次数为17，训练得到的概率分布放到训练结果.txt中。

## 3.3模型测试

测试过程总体上与训练过程类似，使用和训练过程相似的处理函数，选取不同的单词形成测试段落，每篇文章的词赋予一个随机的主题，再计算每个主题被选中的概率，然后进行迭代训练，计算每篇文章选中各个主题的概率与该词语在每个主题中出现的概率相乘，得到该词出现的概率向量，更新每个文章有多少主题词，再更新每个主题的总词数，不断迭代更新，直到每篇文章中的各个主题的概率不再变化，则迭代结束，最终得到测试集迭代后的主题概率和迭代次数。最终迭代次数为12次，测试集得到的概率分布放到测试结果.txt中。

# 实验结果

分析实验结果，需要根据上一章结果分析选出的词属于哪一本小说，这里就采用简单的欧式距离计算，计算测试集选用的词能否正确对应主题，欧式距离最小的被认为是对应改本小说，这样可以得到每一篇测试集和文章的主题的对应关系，输出的欧式距离的结果放到欧氏距离.txt中，最终得到的结果放到最终结果.txt中，可以看到每一篇测试集数据都能够精准的和相应的文章主题也就是文章名相对应，达到了较好的效果。

# 总结分析

本次实验结果较好，一方面是LDA作为文章主题建模效果较好，另一方面是选择段落的方式正确，使得最终有较为准确的结果，这次作业使用了第一次作业的分词，段落处理，加深了我对自然语言处理的理解，以后会更加努力的学习这方面知识，在以后做的更好。

# 附录

import math

import jieba

import os

import random

import numpy as np

##################################################################################################

##本程序参考了https://blog.csdn.net/shzx\_55733/article/details/116280982?spm=1001.2014.3001.5502文章

##################################################################################################

def deta\_deal(content):  #处理语料库，后面会用到，在第一次作业时已经使用过

    ad = ['本书来自www.cr173.com免费txt小说下载站\n更多更新免费电子书请关注www.cr173.com', '----〖新语丝电子文库(www.xys.org)〗', '新语丝电子文库',

          '\u3000', '\n', '。', '？', '！', '，', '；', '：', '、', '《', '》', '“', '”', '‘', '’', '［', '］', '....', '......',

          '『', '』', '（', '）', '…', '「', '」', '\ue41b', '＜', '＞', '+', '\x1a', '\ue42b']

    for a in ad:

        content = content.replace(a, '')

    return content

def read\_novel(path):  # 读取小说内容，主要是按照题目要求读取段落

    cont = []

    names = os.listdir(path)

    for name in names:

            novel\_name = path + '\\' + name

            con\_text = []

            with open(novel\_name, 'r', encoding='ANSI') as data:

                cont\_data = data.read()

                cont\_data = deta\_deal(cont\_data)     #处理数据

                cont\_data = jieba.lcut(cont\_data)    #结巴分词

                con\_list = list(cont\_data)

                p = int(len(cont\_data)//12          )#16篇文章均匀选出200段，可以先每篇文章均匀选出12段

               #因为12\*16=192，少了8段，为了保证总字数不变，将每一段的字数调整为521字，保证均匀

                for i in range(13):

                    con\_text = con\_text + con\_list[i\*p:i\*p+520]

                cont.append(con\_text)

            data.close()

    return cont, names

def read\_novel\_text(path):  # 读取小说内容，主要是按照题目要求读取段落

    cont = []

    names = os.listdir(path)

    for name in names:

            novel\_name = path + '\\' + name

            con\_text = []

            with open(novel\_name, 'r', encoding='ANSI') as data:

                cont\_data = data.read()

                cont\_data = deta\_deal(cont\_data)     #处理数据

                cont\_data = jieba.lcut(cont\_data)    #结巴分词

                con\_list = list(cont\_data)

                p = int(len(cont\_data)//12          )#16篇文章均匀选出200段，可以先每篇文章均匀选出12段

               #因为12\*16=192，少了8段，为了保证总字数不变，将每一段的字数调整为521字，保证均匀

                for i in range(13):

                    con\_text = con\_text + con\_list[i\*p+521:i\*p+1041]

                cont.append(con\_text)

            data.close()

    return cont, names

if \_\_name\_\_ == '\_\_main\_\_':

    ##########这里开始是对模型的训练#########

    ##初始化模型

    [data\_txt, files] = read\_novel("dataset")   #按照段落要求读取小说内容

    print(files)

    Topic\_EVE = []  # 统计每个词来自什么主题，也就是每个词来自哪个文章

    Topic\_count = {}  # 每个文章下有多少词

    Topic\_fre0 = {}; Topic\_fre1 = {}; Topic\_fre2 = {}; Topic\_fre3 = {}

    Topic\_fre4 = {}; Topic\_fre5 = {}; Topic\_fre6 = {}; Topic\_fre7 = {}

    Topic\_fre8 = {}; Topic\_fre9 = {}; Topic\_fre10 = {}; Topic\_fre11 = {}

    Topic\_fre12 = {}; Topic\_fre13 = {}; Topic\_fre14 = {}; Topic\_fre15 = {}  #储存16篇文章的词频

    word\_count = []  # 每篇文章中有多少个词

    word\_fre = []    # 每篇文章主题词的词频

    i = 0

    for data in data\_txt:

        topic = []

        docfre = {}

        for word in data:

            a = random.randint(0, len(data\_txt)-1)          # 每个词赋予一个随机初始主题

            topic.append(a)

            if '\u4e00' <= word <= '\u9fa5':

                Topic\_count[a] = Topic\_count.get(a, 0) + 1  # 计算每主题总词数

                docfre[a] = docfre.get(a, 0) + 1            # 计算每篇文章主题词的词频

                exec('Topic\_fre{}[word]=Topic\_fre{}.get(word, 0) + 1'.format(i, i))  # 计算每个的词频

        Topic\_EVE.append(topic)

        docfre = list(dict(sorted(docfre.items(), key=lambda x: x[0], reverse=False)).values())

        word\_fre.append(docfre)

        word\_count.append(sum(docfre))  # 统计每篇文章的总词数

        i += 1

    Topic\_count = list(dict(sorted(Topic\_count.items(), key=lambda x: x[0], reverse=False)).values())

    word\_fre = np.array(word\_fre)

    Topic\_count = np.array(Topic\_count)

    Doc\_count = np.array(word\_count)

    Word\_eve = []        # 选中每个主题的概率

    Word\_eve1 = []       # 迭代后选中每个主题概率

    for i in range(len(data\_txt)):

        doc = np.divide(word\_fre[i], Doc\_count[i])

        Word\_eve.append(doc)

    Word\_eve = np.array(Word\_eve)

    stop = 0  # 迭代停止标志

    loop\_count = 1  # 迭代次数

    while stop == 0:

        i = 0

        for data in data\_txt:

            top = Topic\_EVE[i]

            for w in range(len(data)):

                word = data[w]

                pro = []

                topfre = []

                if '\u4e00' <= word <= '\u9fa5':

                    for j in range(len(data\_txt)):

                        exec('topfre.append(Topic\_fre{}.get(word, 0))'.format(j))  # 计算每个的词频

                    pro = Word\_eve[i] \* topfre / Topic\_count  # 得到该词出现的概率向量

                    m = np.argmax(pro)                       # 认为该词是由上述概率之积最大的那个topic产生的

                    word\_fre[i][top[w]] -= 1                 # 更新每个文档有多少各个主题的词

                    word\_fre[i][m] += 1

                    Topic\_count[top[w]] -= 1                 # 更新每个主题的总词数

                    Topic\_count[m] += 1

                    exec('Topic\_fre{}[word] = Topic\_fre{}.get(word, 0) - 1'.format(top[w], top[w]))  # 更新每个的词频

                    exec('Topic\_fre{}[word] = Topic\_fre{}.get(word, 0) + 1'.format(m, m))

                    top[w] = m

            Topic\_EVE[i] = top

            i += 1

        if loop\_count == 1:  # 计算每篇文章主题概率

            for i in range(len(data\_txt)):

                doc = np.divide(word\_fre[i], Doc\_count[i])

                Word\_eve1.append(doc)

            Word\_eve1 = np.array(Word\_eve1)

        else:

            for i in range(len(data\_txt)):

                doc = np.divide(word\_fre[i], Doc\_count[i])

                Word\_eve1[i] = doc

       # print('训练前主题概率为：',Word\_eve)

       # print('每次迭代后主题概率为',(Word\_eve1))

        if (Word\_eve1 == Word\_eve).all():

            stop = 1

        else:

            Word\_eve = Word\_eve1.copy()

        loop\_count += 1

    print('最终训练结果为：',Word\_eve1)  # 输出训练后选中每个主题概率

    print('迭代次数为：',loop\_count)  # 输出迭代次数

    print('Training is complete')

#########这里开始对训练好的模型做测试##############

    [test\_txt, files] = read\_novel\_text("dataset")

    Doc\_count\_test = []     # 文章总词数

    Doc\_fre\_test = []       # 文章中主题出现频率

    Topic\_All\_test = []     # 主题中关键词频率

    i = 0

    for data in test\_txt:

        topic = []

        docfre = {}

        for word in data:

            a = random.randint(0, len(data\_txt) - 1)  # 测试，为每一个词赋予一个随机主题

            topic.append(a)

            if '\u4e00' <= word <= '\u9fa5':

                docfre[a] = docfre.get(a, 0) + 1  # 计算词频

        Topic\_All\_test.append(topic)

        docfre = list(dict(sorted(docfre.items(), key=lambda x: x[0], reverse=False)).values())

        Doc\_fre\_test.append(docfre)

        Doc\_count\_test.append(sum(docfre))  # 计算每篇文章总次数，并存储

        i += 1

    Doc\_fre\_test = np.array(Doc\_fre\_test)

    Doc\_count\_test = np.array(Doc\_count\_test)

    print('测试文章主题出现概率',Doc\_fre\_test)

    print('每篇文章出现总次数',Doc\_count\_test)

    Doc\_pro\_test = []       # 与上面相似，计算每个主题被选中的概率

    Doc\_pronew\_test = []    # 与上一步类似，计算迭代后每个主题被选中的概率

    for i in range(len(test\_txt)):

        doc = np.divide(Doc\_fre\_test[i], Doc\_count\_test[i])

        Doc\_pro\_test.append(doc)

    Doc\_pro\_test = np.array(Doc\_pro\_test)

    print('每个主题被选中概率',Doc\_pro\_test)

    stop = 0            # 迭代停止标志

    loop\_count = 1      # 迭代次数

    while stop == 0:

        i = 0

        for data in test\_txt:

            top = Topic\_All\_test[i]

            for w in range(len(data)):

                word = data[w]

                pro = []

                topfre = []

                if '\u4e00' <= word <= '\u9fa5':

                    for j in range(len(data\_txt)):

                        exec('topfre.append(Topic\_fre{}.get(word, 0))'.format(j))  # 计算词频

                    pro = Doc\_pro\_test[i] \* topfre / Topic\_count  # 得到概率向量

                    m = np.argmax(pro)

                    Doc\_fre\_test[i][top[w]] -= 1  # 计算文章中主题词的个数

                    Doc\_fre\_test[i][m] += 1

                    top[w] = m

            Topic\_All\_test[i] = top

            i += 1

        if loop\_count == 1:  # 计算新的每篇文章选中主题的概率

            for i in range(len(test\_txt)):

                doc = np.divide(Doc\_fre\_test[i], Doc\_count\_test[i])

                Doc\_pronew\_test.append(doc)

            Doc\_pronew\_test = np.array(Doc\_pronew\_test)

        else:

            for i in range(len(test\_txt)):

                doc = np.divide(Doc\_fre\_test[i], Doc\_count\_test[i])

                Doc\_pronew\_test[i] = doc

       # print('每个主题被选中概率',Doc\_pro\_test)

       # print('迭代后每个主题被选中概率',Doc\_pronew\_test)

        if (Doc\_pronew\_test == Doc\_pro\_test).all():  # 主题概率不变 认为迭代结束

            stop = 1

        else:

            Doc\_pro\_test = Doc\_pronew\_test.copy()

        loop\_count += 1

    print('测试集迭代后主题被选中概率：',Doc\_pronew\_test)

    print('迭代次数',loop\_count)

    print('Test is complete')

    ##开始表示计算结果

    result = []

    for k in range(len(test\_txt)):

        pro = []

        for i in range(len(data\_txt)):

            d = 0

            for j in range(len(data\_txt)):

                d += (Word\_eve[i][j] - Doc\_pro\_test[k][j])\*\*2  # 计算欧式距离。欧式距离小的认为对应文章主题

            pro.append(d)

        m = pro.index(min(pro))

        print('输出欧式距离为：',pro)

        result.append(m)

    print('读入的文件名：',files)

    print('输出结果为：',result)