

学 期 2022-2023

****

深度学习与自然语言处理

计算中文平均信息熵

|  |  |
| --- | --- |
| 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4月

目录

[一，作业要求 2](#_Toc100343310)

[二，实验背景 2](#_Toc100343311)

[2.1信息熵 2](#_Toc100343312)

[2.2 N-Gram语言模型 2](#_Toc100343313)

[三，实验流程 3](#_Toc100343314)

[3.1数据的读取与预处理 3](#_Toc100343315)

[3.2模型词频计算 3](#_Toc100343316)

[3.3平均信息熵计算 4](#_Toc100343317)

[四，实验结果 4](#_Toc100343318)

[五，总结 4](#_Toc100343319)

[附录： 5](#_Toc100343320)

、

# 作业要求

首先阅读文章：《Entropy\_of\_English\_PeterBrown》参考上面的文章来计算中文(分别以词和字为单位)的平均信息熵。

# 二，实验背景

## 2.1信息熵

熵，泛指某些物质系统状态的一种量度，某些物质系统状态可能出现的程度，熵的概念是由德国物理学家[克劳修斯](https://baike.baidu.com/item/%E5%85%8B%E5%8A%B3%E4%BF%AE%E6%96%AF/1254238)于1865年所提出，在热力学当中使用广泛，熵的本质是一个系统“内在的混乱程度”。1948年，[香农](https://baike.baidu.com/item/%E9%A6%99%E5%86%9C)将统计物理中熵的概念，引申到信道通信的过程中，从而开创了”[信息论](https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E8%AE%BA/302185" \t "_blank)“这门学科。香农定义的“熵”又被称为“[香农熵](https://baike.baidu.com/item/%E9%A6%99%E5%86%9C%E7%86%B5/1649961)”或“[信息熵](https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E7%86%B5/7302318)”，如式2.1所示，而针对联合分布的随机变量，当两个变量相互独立的情况下，根据联合概率分布公式，其联合信息熵如式2.2所示，数学上可证明[信息熵](https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E7%86%B5/7302318)具有连续性，对称性，极值性，可加性等。

 (2.1)

其中i为概率空间中所有可能的样本，pi表示该样本的出现几率，K是和单位选取相关的任意常数。

 （2.2）

## 2.2 N-Gram语言模型

N-Gram是一种基于统计语言模型的算法。它的基本思想是将文本里面的内容按照字节进行大小为n的滑动窗口操作，形成了长度是n的字节片段序列。假设句子T是有词序列w1,w2,w3...wn组成，用公式表示N-Gram语言模型如式2.3所示，但实际情况我们并不采用这种方法直接计算，这样肢解计算容易导致参数空间过大，数据稀疏严重等问题。

 （2.3）

为了解决上述问题，引入马尔科夫假设进行简化，假设随意一个词出现的概率只与它前面出现的有限的一个或者几个词有关，这样我们就得到前面的条件概率计算简化如式2.4所示，当k为0时就是一元模型，即与他前面的0个词相关，这也就是按照字为单位的平均信息熵，当k=1时为二元模型，即与他前面一个词相关。依此类推，可以得到三元，四元等等。本次作业要求按字和词为单位的平均信息熵，故分别采用一元，二元，三元模型求平均信息熵。

# 三，实验流程

## 3.1数据的读取与预处理

实验第一步，需读取出文本信息，同时对信息做处理，得到可以使用的语料库，同时计算出整个语料库的字数，方便后续使用，在本次作业中，数据处理放在一个类class pre\_process()当中，其中主要包括 getCorp(self, fileroute)函数，做的工作主要是将一本一本小说提取出来，每一本分别做去多余字，去标点，去各种符号的工作，利用os模块操作文件路径，最后将每一本处理好的小说放入到corpus数组当中，同时计算总字数。­处理后的结果如图3.1所示，可见符号，回车，空格等影响后续计算的符号均以删除，该语料库满足要求，可以正常使用。

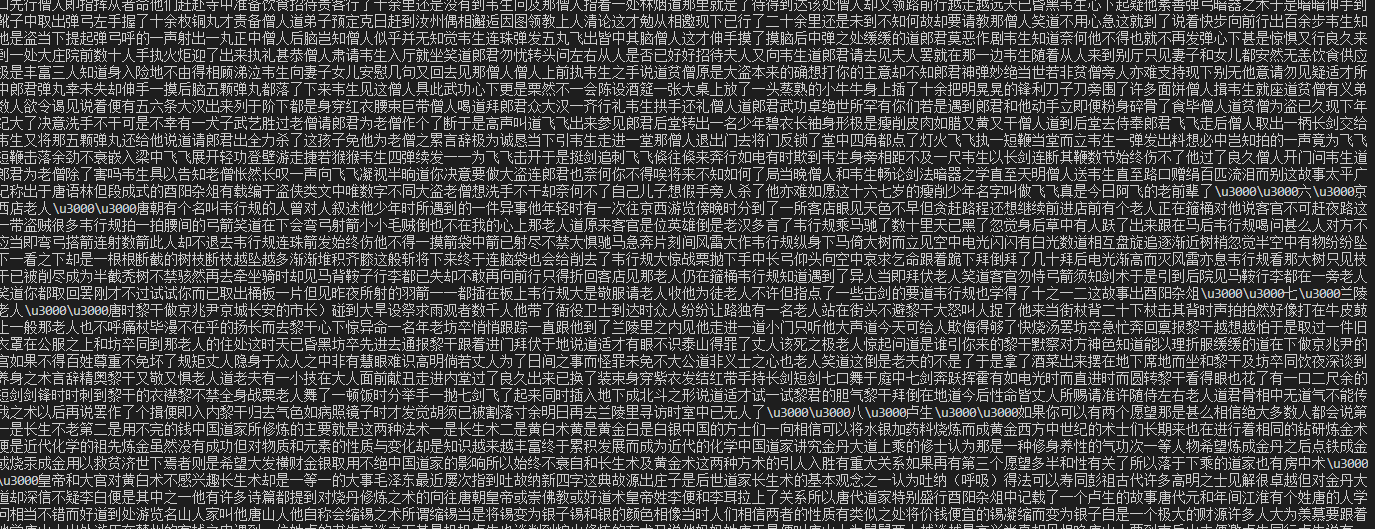


图3.1处理后文本

## 3.2模型词频计算

分别计算一元模型，二元模型，三元模型词频，为了提高程序可读性，设置了三个函数 get\_fsigram\_tf(tf\_dic, words)，get\_secgram\_tf(tf\_dic, words)，get\_trigram\_tf(tf\_dic, words)分别计算模型的词频，得到词频后方便下一步信息熵的计算。

## 3.3平均信息熵计算

分别使用一元，二元，三元模型计算信息熵，在程序的编写上这三种方法十分相似，即循环得到的语料库corpus，采用jieba模块做分词处理，将处理结果送入3.2的词频计算函数中计算词频，最后根据不同的模型选择不同的计算公式，计算出平均信息熵，至此，计算出不同模型下的结果。

# 四，实验结果

通过第三章的的计算，得到整体实验的结果，如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 模型选择 | 总字数 | 分词个数 | 平均信息熵 |
| 1 | 一元模型 | 7420148 | 4430794 | 12.01 |
| 2 | 二元模型 | 7420148 | 4430794 | 6.89 |
| 3 | 三元模型 | 7420148 | 4430794 | 2.42 |

通过上述结果可以看出，平均信息熵随着单个分词的不同而发生较大变化，当一元模型，也就是按字计算时平均信息熵较大，而三元模型信息熵较小，这符合我们的认知，在按字分时比较混乱，无法得到正确的意思，而随着词的数量的上升，意义就越来越明确，则平均信息熵也越小，同时耗费的时间也就越长

# 五，总结

本次作业我更加深入了解了自然语言处理，在此之前没有做过这种大量文件处理的工作，这次是第一次尝试，借鉴了别人的框架，学习并思考了这种情况的处理方式，我认为只要我坚持学下去就会越来越熟练，同时能够在这方面做的更好。

## 附录：

import jieba

import math

import os

import re

#本代码整体架构参考了CSDN上的文章《深度学习与自然语言处理实验——中文信息熵的计算》，同时增加了一些自己的理解

#在之前并未做过类似大批量处理文件的操作，通过学习参考别人的代码做出尝试

#首先，初始化做一个类

class pre\_process():

    def \_\_init\_\_(self, fileroute):

        self.fileroute = fileroute

    def giv\_process(self):

        return pre\_process.getCorp(self, self.fileroute)

    def getCorp(self, fileroute):     #定义一个函数，主要目的是处理数据库，包括删除标点，多余字等等工作，得到语料库，为后面的计算提供条件

        #这里定义几个必要量

        rdelate = u'[a-zA-Z0-9’!"#$%&\'()\*+,-./:：;<=>?@，。?★、…【】《》？“”‘’！[\\]^\_`{|}~]+'  # 定义了要删掉的一些符号，保证文章中只剩下中文，便于后续处理

        allfile\_list = os.listdir(fileroute)    #列出路径下所有文件，储存在allfile\_list当中，后面使用

        corpus = []   #定义一个空的数组，将未来的语料库放入进去

        count=0   #定义一个计数器用于计算文字数量，便于平均数字量的计算

        for file in allfile\_list:   #将小说一本一本提取出来，每一本分别处理

            path  = os.path.join(fileroute, file)   #该函数用来整合路径，输出即为当前文件的路径

            if os.path.isfile(path):  #做一个判断，只有路径正确才进行下一步处理，否则进入下一个路径

                with open(os.path.abspath(path), "r", encoding='ansi') as file:

                    file\_read = file.read()    #导入文件

                    file\_read = re.sub(rdelate, '', file\_read)   #把多余的符号删除，符号上面有过定义

                    file\_read = file\_read.replace("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", '')    #本身缺少这句话，打开文件后发现该段话与正文无关，故删除

                    file\_read = file\_read.replace("\n", '') #下面两句话删除一些空格和回车，让整个语料库连接起来，方便操作

                    file\_read = file\_read.replace(" ", '')

                    #下面两句话分别是记录语料库字数和修改过后的语料库，也是我们这个函数整个处理的最终需要得到的结果，

                    count += len(file\_read)     #通过循环记录所有文章总字数

                    corpus.append(file\_read)    #通过循环将所有文章导入数组，形成语料库

        return corpus,count

#做一个词频统计函数，三种ngram模型计算方式不同，方便后面计算，这里设计了三个函数，没有采用for的方式做成一个，方便阅读

#同时借鉴了csdn上的计算方法

def get\_fsigram\_tf(tf\_dic, words):   #一元模型词频统计

    for i in range(len(words)-1):

        tf\_dic[words[i]] = tf\_dic.get(words[i], 0) + 1

def get\_secgram\_tf(tf\_dic, words):   #二元模型词频统计

    for i in range(len(words)-1):

        tf\_dic[(words[i], words[i+1])] = tf\_dic.get((words[i], words[i+1]), 0) + 1

def get\_trigram\_tf(tf\_dic, words):   #三元模型词频统计

    for i in range(len(words)-2):

        tf\_dic[((words[i], words[i+1]), words[i+2])] = tf\_dic.get(((words[i], words[i+1]), words[i+2]), 0) + 1

#下面开始做三个函数，分别使用一元，二元，三元模型

def cal\_unigram(corpus,count):

    all\_words = []      #定义新数组，用于储存分词

    words\_count = 0     #记录分词个数，用于后续计算

    words\_tf = {}       #用于储存词频函数计算出的词频

    information\_entropy = []    #将每本小说的信息熵输入其中

    for word in corpus: #循环调用整个库，每次计算一本小说

        for x in jieba.cut(word):   #采用jieba做分词，将改本小说分解

            all\_words.append(x)     #将分词接到all\_words

            words\_count += 1        #每有一个分词即进一次循环，得到总的分词个数

        get\_fsigram\_tf(words\_tf, all\_words)     #通过词频函数计算出词频，后面计算使用

        all\_words = []                          #在一本小说计算完成后重置，准备用于计算下一本小说

    print("字数:", count)                       #输出语料库字数，该值在前面已经计算

    print("平均词长:", count / words\_count)   #输出平均字长

    print("分词数:", words\_count)               #输出分次数

    #计算信息熵，利用公式，可以得到每一本小说的信息熵

    for fri\_word in words\_tf.items():

        information\_entropy.append(-(fri\_word[1] / words\_count) \* math.log(fri\_word[1] / words\_count, 2))  #计算一元信息熵公式

    print("一元模型----平均信息熵:", sum(information\_entropy), "比特/词")   #输出结果，算一个总值

def cal\_bigram(corpus, count):

    all\_words = []

    words\_count = 0

    words\_tf = {}

    secgram\_tf = {}

    information\_entropy = []

    for word in corpus:

        for x in jieba.cut(word):

            all\_words.append(x)

            words\_count += 1

        get\_fsigram\_tf(words\_tf, all\_words)

        get\_secgram\_tf(secgram\_tf, all\_words)

        all\_words = []

    print("字数:", count)

    print("平均词长:", count / words\_count)

    print("分词数:", words\_count)

    secgram\_len = sum([dic[1] for dic in secgram\_tf.items()])

    print("二元模型长度:", secgram\_len)

#这里计算使用公式，同时参考网上代码

    for sec\_word in secgram\_tf.items():

        jp\_xy = sec\_word[1] / secgram\_len                        # 计算联合概率

        cp\_xy = sec\_word[1] / words\_tf[sec\_word[0][0]]           # 计算条件概率

        information\_entropy.append(-jp\_xy \* math.log(cp\_xy, 2))  # 计算二元模型的信息熵

    print("二元模型----平均信息熵:", sum(information\_entropy), "比特/词")

def cal\_trigram(corpus,count):

    all\_words = []

    words\_count = 0

    words\_tf = {}

    trigram\_tf = {}

    information\_entropy = []

    for word in corpus:

        for x in jieba.cut(word):

            all\_words.append(x)

            words\_count += 1

        get\_secgram\_tf(words\_tf, all\_words)

        get\_trigram\_tf(trigram\_tf, all\_words)

        all\_words = []

    print("字数:", count)

    print("平均词长:", round(count / words\_count, 5))

    print("分词数:", words\_count)

    trigram\_len = sum([dic[1] for dic in trigram\_tf.items()])

    print("三元模型长度:", trigram\_len)

#计算信息熵

    for tri\_word in trigram\_tf.items():

        jp\_xy = tri\_word[1] / trigram\_len                        # 计算联合概率

        cp\_xy = tri\_word[1] / words\_tf[tri\_word[0][0]]           # 计算条件概率

        information\_entropy.append(-jp\_xy \* math.log(cp\_xy, 2))  # 计算三元模型的信息熵

    print("三元模型----平均信息熵::", sum(information\_entropy), "比特/词")

#主函数

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

    cons = pre\_process("./dataset")

    corpus,count = cons.giv\_process()

    cal\_unigram(corpus, count)      #一元模型计算

    cal\_bigram(corpus,count)        #二元模型计算

    cal\_trigram(corpus,count)       #三元模型计算