学　　期 2023-2024 第二学期

北京航空航天大學
BEIHANG UNIVERSITY

Deep NLP

第一次大作业

院（系）名称	自动化科学与电气工程学院
专业名称	控制工程
学生姓名	乔彪
学号	ZY2303706

2024 年 4 月

一、问题描述

1.通过中文语料库来验证 Zipf's Law。

2.阅读 Entropy Of English，计算中文(分别以词和字为单位) 的平均信息熵。

二、问题解释

1. Zipf's Law

齐夫定律（Zipf's Law）可以表述为：在自然语言的语料库里，一个单词出现的频率与它在频率表里的排名成反比。具体而言，齐夫定理可以表述为：一个单词出现的频率 f 与它在频率表中的排名 r 的乘积大致是一个常数，即 $f \times r = k$，这里的 k 是一个常数。

2.信息熵

信息熵（Information Entropy）是信息论中的一个基本概念，用于衡量随机变量或系统的不确定性。它是由克劳德·香农（Claude Shannon）在 1948 年提出的，是现代通信理论的基础之一。信息熵可以看作是信息含量的度量，即在获得一个随机变量的值之前，我们期望获得多少信息。

对于一个随机变量X，其信息熵$H(X)$的定义如下：

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

对于联合分布的随机变量(X, Y)，在在两变量相互独立的情况下，其联合信息熵为：

$$H(X|Y) = -\sum_{y \in Y} \sum_{x \in X} p(x, y) \log_2 p(x|y)$$

对数函数的底数通常取 2，这样信息熵的单位就是比特（bit）。

3.n-gram 模型

n-gram 模型是一种统计语言模型，用于自然语言处理中的各种任务，如文本预测、拼写检查、机器翻译等。n-gram 模型基于这样一个假设：一个词的出现只与其前面的 n-1 个词相关，而与更前面的词无关。这里的“n”表示上下文的长度，即模型考虑的词的个数。它通过计算 n 个连续词或标记的频率来估计概率分布。例

如，在 bigram 模型中，给定一个词序列$w_1, w_2, \ldots, w_n$，模型会计算每个 bigram w_i, w_{i+1}的概率$p(w_{i+1}|w_i)$。

三、代码实现

（一）代码主函数

```
if __name__ == '__main__':

    punctuation = get_useless("cn_punctuation.txt")
    stopwords = get_useless("cn_stopwords.txt")

    texts = dict()
    get_texts(texts,'中文语料库')

    wordslists = get_wordslists(texts,punctuation)

    prove_law(wordslists, stopwords)

    table_word = []
    fullwords = {'所有小说':get_fullwords(wordslists)}
    list_table(table_word,wordslists)
    list_table(table_word,fullwords)
    save_table(table_word,'基于词的N元模型的中文信息熵','word.png')

    str_lists = get_strlists(wordslists)
    table_character = []
    list_table(table_character,str_lists)
    fullcharacters = {'所有小说':combine_str(str_lists)}
    list_table(table_character,fullcharacters)
    save_table(table_character,'基于字的N元模型的中文信息熵','character.png')
```

（二）代码详细介绍

1.读取标点符号和停顿字

```
punctuation = get_useless("cn_punctuation.txt")
stopwords = get_useless("cn_stopwords.txt")
```

分别读取 cn_punctuation.txt 和 cn_stopwords.txt，以列表的形式分别储存在 punctuation 和 stopwords 中。

```
def get_useless(list):
    with open(list, 'r', encoding='utf-8') as file:
        useless = set([line.strip() for line in file.readlines()])
    return useless
```

2.读取文件夹内的所有小说

```
texts = dict()
get_texts(texts,'中文语料库')
```

读取“中文预料库”文件夹中所有的小说并进行初步的预处理，包括删除所有的换行、空格、以及全角英文。最后以字典的形式存储在 texts 中，该字典的 key 为每个小说的名称，value 为每个小说预处理后的中文字符串。

```
def get_texts(texts,rootDir):
    listdir = os.listdir(rootDir)
    for file in listdir:
        path = os.path.join(rootDir, file)
        if os.path.isfile(path) and os.path.splitext(file)[1].lower() == '.txt':
            with open(os.path.abspath(path), "r", encoding='ansi') as file:
                filename = os.path.basename(file.name)
                if re.search(r'[\u4e00-\u9fa5]', filename):
                    filecontext = file.read()
                    filecontext = filecontext.replace('\n','')
                    full_width_english = re.compile(r'[\uFF01-\uFF5E]+')
                    filecontext = full_width_english.sub('',filecontext)
                    texts[filename] = filecontext.replace('\u3000','')
        elif os.path.isdir(path):
            get_texts(texts, path)
```

3.小说分词

```
wordslist = get_wordslists(texts,punctuation)
```

对每个小说进行分词，并删除所有的非中文字符，以字典的形式保存在 wordslist 中，其 key 为小说名称，value 为每篇小说中的所有分词。

```
def get_wordslists(texts,punctuation):
    wordslists = dict()
    for text_name,text in texts.items():
        words = jieba.lcut(text)
        words_noPunctuation = [word for word in words if word not in punctuation and word.isalpha() and not word.isascii()]
        wordslists[text_name] = words_noPunctuation
    return wordslists
```

4.验证 Zipf's Law

```
prove_law(wordslist, stopwords)
```

这里开始验证齐夫定理。首先删除分词中的所有停顿词，然后将剩余分词整合到一个列表（fullwords）中，统计所有分词的频次和排名，绘制双对数图并保存。

```
def prove_law(wordslists,stopwords):
    fullwords = []
    for words_name,words in wordslists.items():
        words_noStopwords = [word for word in words if word not in stopwords and word.isalpha() and not word.isascii()]
        fullwords = fullwords + words_noStopwords
    fullranks,fullfrequencies = get_ranks(fullwords)
    plt.loglog(fullranks,fullfrequencies, 'o', 10, 10)
    plt.xlabel('Rank')
    plt.ylabel('Frequency')
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    plt.title('Zipf\'s law')
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.savefig('齐夫定理.png')
    plt.close()
```

get_ranks 函数用于获得排名和频次。

```
def get_ranks(words):
    word_counts = Counter(words)
    words_sorted = word_counts.most_common()
    ranks = np.arange(1, len(words_sorted) + 1)
    frequencies = np.array([word[1] for word in words_sorted])
    return ranks,frequencies
```

5.计算基于词的 N 元模型的中文信息熵

```
table_word = []
list_table(table_word,wordslist)
fullwords = {'所有小说': get_fullwords(wordslist)}
list_table(table_word,fullwords)
save_table(table_word,'基于词的N元模型的中文信息熵','word.png')
```

首先建立一个空列表 table_word ，用于接收所有的结果数据以便展示。然后计算词表 wordslist 中每个小说各自的基于词的一元、二元、三元模型的中文信息熵，并以四元的列表形式保存，其第一个元素为小说名称，剩下四个元素分别为基于词的一元、二元、三元模型的中文信息熵。这个四元列表会添加到 table_word 中。然后将所有分词整合在一起，以类似的方法计算整个语料库的基于词的一元、二元、三元模型的中文信息熵，并以四元的列表形式保存，然后添加到 table_word 中。最后，绘制并保存表格。

将信息熵计算结果添加到列表中：

```
def list_table(table,corpuslist):
    for key,value in corpuslist.items():
        uni_entropy, bi_entropy, tri_entropy = cal_entropy(value)
        table.append([key,uni_entropy, bi_entropy, tri_entropy ])
```

计算语料库的中文信息熵：

```
def cal_entropy(words):
    unigram_tf = dict()
    bigram_tf = dict()
    trigram_tf = dict()
    uni_entropy = 0
    bi_entropy = 0
    tri_entropy = 0
    get_tf(unigram_tf,words)
    words_len = len(words)
    for uni_word in unigram_tf.items():
        uni_entropy += -(uni_word[1] / words_len) * math.log(uni_word[1] / words_len, 2)
    get_bigram_tf(bigram_tf,words)
    bigram_len = sum([dic[1] for dic in bigram_tf.items()])
    for bi_word in bigram_tf.items():
        bi_entropy += -(bi_word[1] / bigram_len) * math.log((bi_word[1] / unigram_tf[bi_word[0][0]]), 2)
    get_trigram_tf(trigram_tf,words)
    trigram_len = sum([dic[1] for dic in trigram_tf.items()])
    for tri_word in trigram_tf.items():
        tri_entropy += -(tri_word[1] / trigram_len) * math.log((tri_word[1] / bigram_tf[tri_word[0][0]]), 2)
    return uni_entropy,bi_entropy,tri_entropy
```

一元模型频次统计：

```
def get_tf(tf_dic,corpus):
    for i in range(len(corpus)):
        tf_dic[corpus[i]] = tf_dic.get(corpus[i], 0) + 1
```

二元模型频次统计：

```
def get_bigram_tf(tf_dic, corpus):
    for i in range(len(corpus)-1):
        tf_dic[(corpus[i], corpus[i+1])] = tf_dic.get((corpus[i], corpus[i+1]), 0) + 1
```

三元模型频次统计：

```
def get_trigram_tf(tf_dic,corpus):
    for i in range(len(corpus)-2):
        tf_dic[((corpus[i], corpus[i+1]), corpus[i+2])] = tf_dic.get(((corpus[i], corpus[i+1]), corpus[i+2]), 0) + 1
```

绘制表格：

```
def save_table(table,title,savefig):
    plt.table(cellText=table, colLabels=['采用文本', 'N=1（比特/词）', 'N=2（比特/词）', 'N=3（比特/词）'], loc='center',
              cellLoc='center')
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.axis('off')
    plt.title(title)
    plt.savefig(savefig)
    plt.close()
```

6.计算基于字的 N 元模型的中文信息熵

```
str_list = get_strlists(wordslist)
table_character = []
list_table(table_character,str_list)
fullcharacters = {'所有小说':combine_str(str_list)}
list_table(table_character,fullcharacters)
save_table(table_character,'基于字的N元模型的中文信息熵','character.png')
```

首先把每个预处理后词表中的分词合并为字符串，以字典的形式储存在 str_list 中，其 key 为小说名称，value 为每个小说预处理后分词合并后的字符串。之后的步骤与计算基于词的 N 元模型的中文信息熵类似。

将每个小说的分词表合并为字符串：

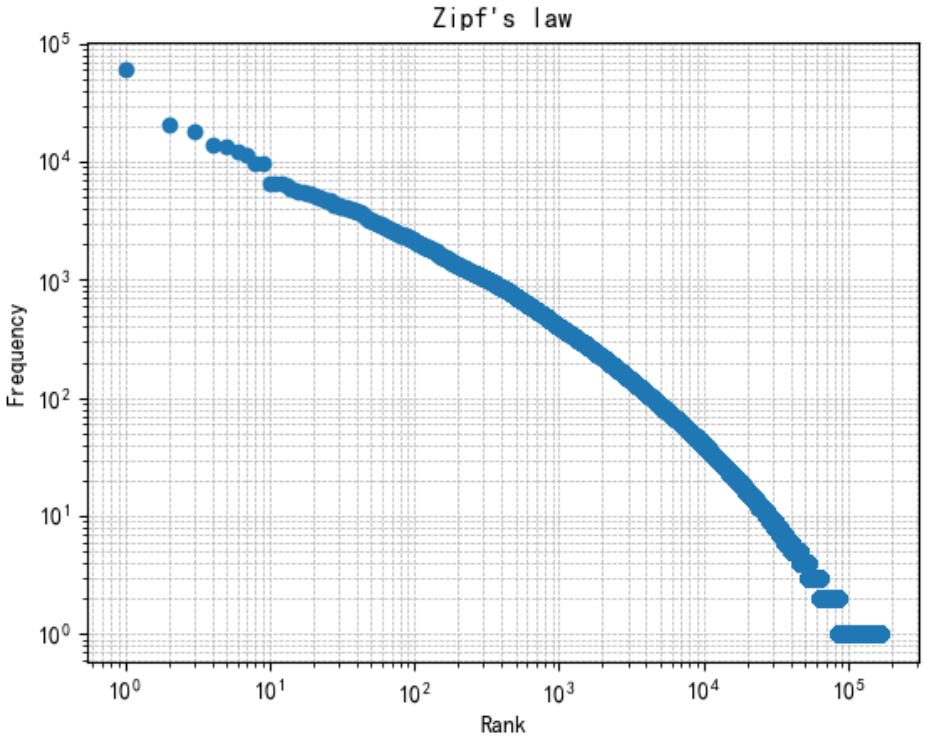
```
def get_strlists(wordslist):
    str_lists = dict()
    for words_name,words in wordslist.items():
        str_lists[words_name] = ''.join(words)
    return str_lists
```

将所以小说的字符串合并为同一个字符串：

```
def combine_str(strlist):
    combined_str = ''
    for key,str in strlist.items():
        combined_str = combined_str +str
    return combined_str
```

四、运行结果

1.验证 Zipf’s Law



2.计算基于词的 N 元模型的中文信息熵

基于词的N元模型的中文信息熵

采用文本	N=1（比特/词）	N=2（比特/词）	N=3（比特/词）
三十三剑客图.txt	11.669104623134052	2.9560004092858465	0.27538286530406003
书剑恩仇录.txt	11.709613493267964	5.033581046181423	1.0415203887426003
侠客行.txt	11.18239225154328	4.9630895579472085	1.1276305682317274
倚天屠龙记.txt	11.748931471612849	5.525342066118218	1.333005615647229
天龙八部.txt	11.724485711389837	5.69191090880628	1.4815600762681556
射雕英雄传.txt	11.825977757414934	5.475939276679227	1.2676611604489585
白马啸西风.txt	10.233896949244375	3.9941560367059505	0.7466346144794723
碧血剑.txt	11.733087279023904	5.006528695581751	0.9952909873122333
神雕侠侣.txt	11.741620701268122	5.549342467430264	1.341265732939682
笑傲江湖.txt	11.39171495629349	5.625156926417476	1.530671816950965
越女剑.txt	10.078724611421235	2.521350214778428	0.3299651114135114
连城诀.txt	11.033078759239611	4.701355195823554	0.9568051114584701
雪山飞狐.txt	11.104670283371444	4.116444331181881	0.6942466251874452
飞狐外传.txt	11.522008171648114	5.00078879559856	1.0542052044548071
鸳鸯刀.txt	10.474975036951895	3.0848853320815826	0.4356393082783876
鹿鼎记.txt	11.441673927697924	5.771410870648666	1.619133900839477
所有小说	12.172768731576907	6.961747805430358	2.300287313557874

3.计算基于字的 N 元模型的中文信息熵

基于字的N元模型的中文信息熵

采用文本	N=1（比特/词）	N=2（比特/词）	N=3（比特/词）
三十三剑客图.txt	9.66862417696748	4.835899866647931	0.9827768080437552
书剑恩仇录.txt	9.466561099009471	5.780634773035317	2.390966384181032
侠客行.txt	9.152722024671247	5.59143520002912	2.368947985366288
倚天屠龙记.txt	9.393314695348417	6.021232587352696	2.8052085885818046
天龙八部.txt	9.404248068432226	6.125016349604318	2.9494540487966923
射雕英雄传.txt	9.439147017933601	6.0619611974970535	2.7603468961344166
白马啸西风.txt	8.912318037067037	4.582907198712382	1.6195390766154505
碧血剑.txt	9.445846392385393	5.869481031415181	2.353359306053134
神雕侠侣.txt	9.372668560122648	6.063134996405713	2.846227010000885
笑傲江湖.txt	9.206355549738763	5.89776200689472	2.8708678115832575
越女剑.txt	8.831324971378029	3.640491254367667	0.9078519017727433
连城诀.txt	9.170997684453623	5.3988026027062785	2.157033365184222
雪山飞狐.txt	9.201964063770724	5.163924125443452	1.7581158377261503
飞狐外传.txt	9.307862668362988	5.753991029569922	2.407003855659806
鸳鸯刀.txt	9.035696055984973	4.216055962488476	1.1161861865891167
鹿鼎记.txt	9.281268393749162	5.993215896608628	2.9531209026966816
所有小说	9.527473313842183	6.72659062622988	3.9512675460117404

五、结果分析

通过绘制语料库中的词频-排名双对数图，我们可以发现输出结果可以拟合为一个直线，这表明语料库中单词的频率和排名可以近似成反比，Zipf’s Law 得到验证。

无论是基于词的 N 元模型信息熵还是基于字的 N 元模型信息熵，随着 N 的增大，要考虑上下文之间的关系更多，不确定性减小了，信息量随之变小，导致信息熵减小了。而对比基于词和基于字的信息熵，N=1 时基于字的 N 元模型信息熵小于基于词的 N 元模型信息熵；N=2 开始部分语料库中基于字的 N 元模型信息熵大于基于词的 N 元模型信息熵；N=3 时基于字的 N 元模型信息熵大于基于词的 N 元模型信息熵。原因可能是当 N 较小时，多个字组成的词不确定性比单个字更大，信息量也更大，随着 N 的增大，基于词要考虑的上下文关系更多，由于语料库的限制，应用范围较小，信息熵反而更低了。