学　号 20376310

北京航空航天大學
BEIHANG UNIVERSITY

深度学习与自然语言处理

大作业 1

验证齐普夫定律

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	自动化
学生姓名	杨佳木
学 号	**20376310**

2024 年 4 月

1.摘要

第一部分：通过提供的中文的语料库，验证齐普夫（Zipf's Law）定律。

第二部分：通过阅读 Entropy of English，根据信息熵公式计算中文语料库（以词和字为单位）的信息熵。

2.第一部分

2.1 引言

齐夫定律是由哈佛大学的语言学家乔治·金斯利·齐普夫（George Kingsley Zipf）于 1949 年发表的实验定律。它可以表述为：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。所以，频率最高的单词出现的频率大约是出现频率第二位的单词的 2 倍，而出现频率第二位的单词则是出现频率第四位的单词的 2 倍。

本实验的计算方法如下

1. 通过对提供的十六个中文语料库分别进行 jieba 分词，统计每个词出现词频并将其排序。
2. 将排序结果转换为对数图，通过验证其对数图近似为线性关系，验证齐普夫定律。
3. 将十六个中文语料库分词结果合并，绘制对数图并计算其线性相关系数，通过比较线性相关系数的绝对值与 0.9

的关系，若大于，则证明线性相关性强，从而验证齐普夫定律

2.2 实验验证

针对提供的每个文本进行分析，并绘制其对数词频图如图 1 所示

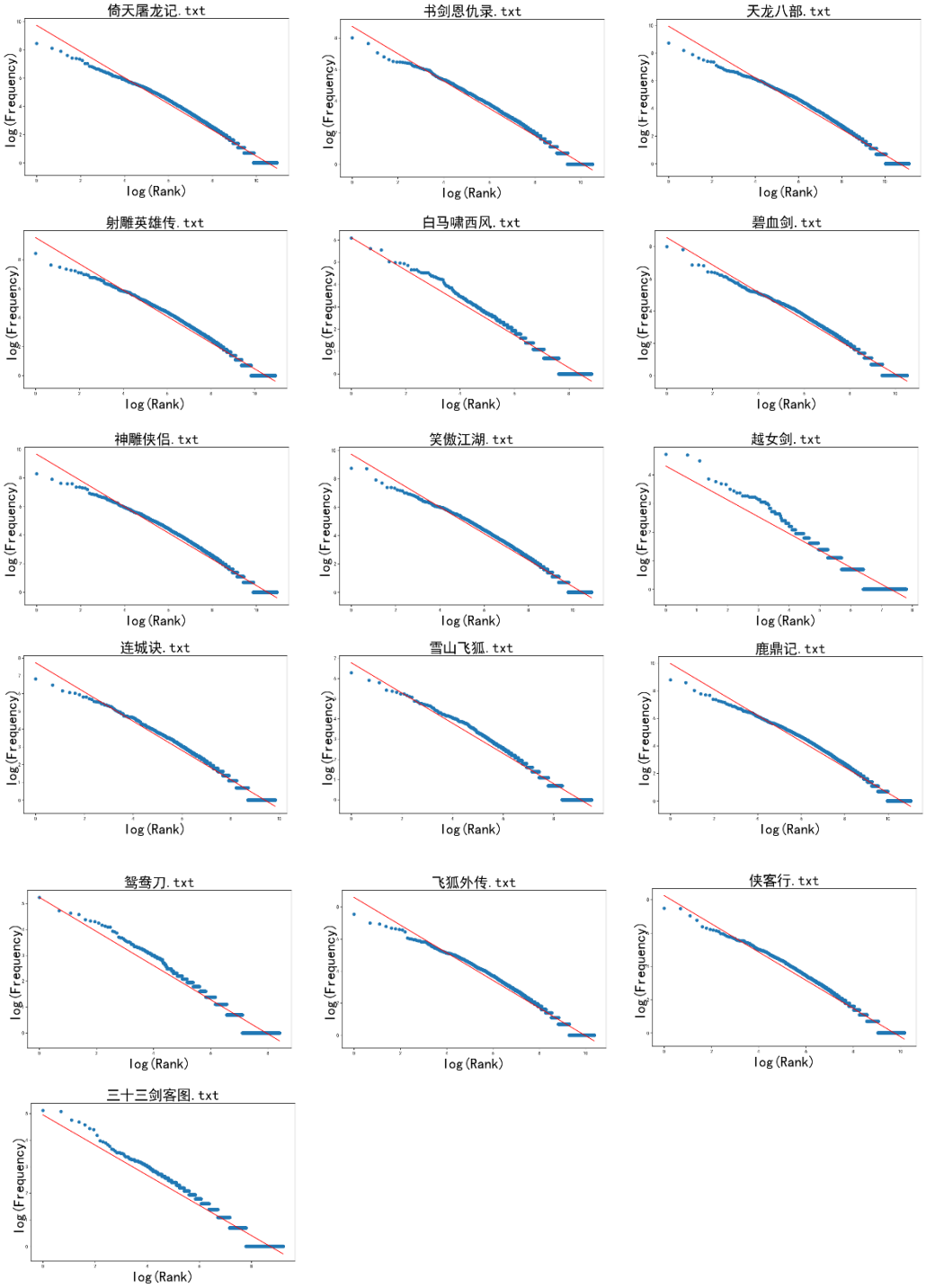

图 1 十六个文本信息对数词频图

将提供的十六个文本合并，统计其词频并按照由高到低排序，并绘制对数词频图如图 2 所示：

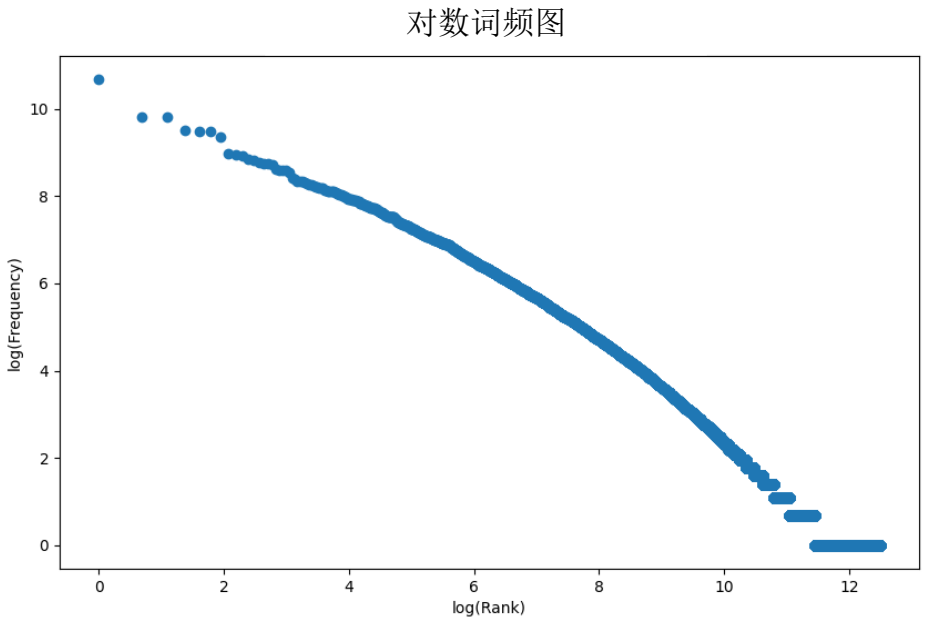

图 2 对数词频图

进行一元线性回归，得到拟合的对数直线，如图 3 所示

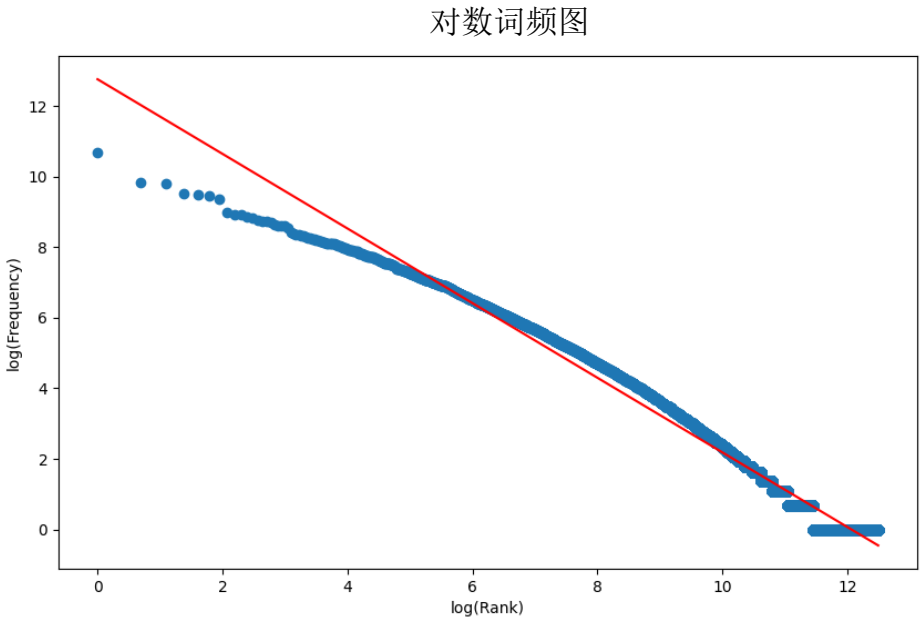

图 3 经过拟合的对数直线

得到的线性方程为：

$$y = -1.0575x + 12.7604 \quad (1)$$

线性相关系数为

$$|r| = 0.96706 \quad (2)$$

认为线性相关性强，则验证齐普夫定律。

3. 第二部分

3.1 文献阅读

参考《An Estimate of an Upper Bound for the Entropy of English》，该论文提出了一种估计英文字符熵的方法，并通过该方法在布朗语料库上进行了验证。通过该方法，作者得出英文字符熵的上限为 1.75bit。该方法的创新点在于

1.使用了更大的英语文本样本;以前的估计是基于最多几百封信的样本。

2.使用语言模型来近似字符串的概率;以前的估计采用的是真人受试者。

3.在所有可打印的 ASCII 字符中做预测。

3.1 信息熵计算

在信源中，考虑的不是某一单个符号发生的不确定性，而是要考虑这个信源所有可能发生情况的平均不确定性。若信源符号有n种取值：$U_1, \dots, U_n$，对应概率为：$P_1, \dots, P_n$，且各种

符号的出现彼此独立。这时，信源的平均不确定性应当为单个符号不确定性 $-logP_i$ 的统计平均值，可称为信息熵，即：

$$H(U) = E[-logP_i] = -\sum_{i=1}^{n} -P_i logP_i \tag{3}$$

基于信息熵的理论，在中文文本中进行应用。假定 S 表示某一个有意义的句子，由一连串特定顺序排列的词 $w1, w2, \cdots, wn$ 组成，n为句子的长度。现在想知道 S在文本中出现的可能性，即 $P(S)$。此时设计信息熵模型来估算，把$P(S)$ 展开表示为 $P(S) = \mathrm{P}(w1, w2, \cdots, wn)$。利用贝叶斯定理，$S$这个序列出现的概率等于每一个词出现的条件概率相乘，于是 $\mathrm{P}(w1, w2, \cdots, wn)$ 可展开为：

$$\mathrm{P}(w1, w2, \cdots, wn) = P(w1)P(w2|w1)P(w3|w1, w2) \ldots P(wn|w1, w2, w3, \ldots, wn-1) \tag{4}$$

当句子过长时，$P(wn|w1, w2, w3, \ldots, wn-1)$难以计算，在实际的文本中，字、词、一元词组、二元词组和三元词组的出现概率大致等于句子在文本中的出现概率。

一元组模型信息熵的计算公式为：

$$H(X) = -\sum P(x) \log P(x) \tag{5}$$

二元组模型的信息熵计算公式为：

$$H(X|Y) = -\sum P(x, y) \log P(x|y) \tag{6}$$

三元组模型的信息熵计算公式为：

$$H(X|Y, Z) = -\sum P(x, y, z) \log P(x|y, z) \tag{6}$$

文本预处理方法

1）删除开头无用信息

2）删除中文停词及标点符号（利用 cn_stopwords.txt 文件）

3）删除空白符号，比如空格、换行符等。

基于该论文中提出的交叉熵法，在提供的中文语料库中进行计算，通过对 jieba 分词和单个字的信息熵进行计算，得到十六个文本的一元信息熵如表 1 所示

表 1 一元信息熵

语料库	一元词信息熵/bit	一元字信息熵/bit
三十三剑客图	12.45481773269039	10.005023057392192
书剑恩仇录	12.777821441598547	9.746973010743657
侠客行	12.394244132301642	9.434944354682404
倚天屠龙记	13.024288506017223	9.701389804896253
天龙八部	13.224433168615294	9.780112428579546
射雕英雄传	13.142243181611207	9.737277383142489
白马啸西风	11.190455454587655	9.217712907912517
碧血剑	12.929869942575078	9.742780840133838
神雕侠侣	13.023588532218152	9.671986296103249
笑傲江湖	12.631840339843768	9.507912358258327
越女剑	10.272339507762355	8.78928513030255
连城诀	12.296430517033164	9.513325286880987
雪山飞狐	12.150470863072332	9.49674832643194

飞狐外传	12.740121122409821	9.622388134439593
鸳鸯刀	10.990187621829664	9.212399794640092
鹿鼎记	12.908572445288412	9.64840190712427
总语料库	13.583532113495743	9.949032134455893

计算程序如下：

```
def calc_entropy_unigram(word1, is_ci):
    # 计算一元模型的信息熵
    word_tf = get_unigram_tf(word1)
    word_len = sum([item[1] for item in word_tf.items()])
    entropy = sum(
        [-(word[1] / word_len) * math.log(word[1] / word_len, 2) for
word in
        word_tf.items()])
    if is_ci:
        print("基于词的一元模型的中文信息熵为：{}比特/词".format( entropy))
    else:
        print("基于字的一元模型的中文信息熵为：{}比特/字".format( entropy))
    return entropy
```

二元信息熵的计算结果如表 2 所示：

表 2 二元信息熵

语料库	二元词信息熵/bit	二元字信息熵/bit
三十三剑客图	1.6473813862387985	4.2866797528041065
书剑恩仇录	3.93329359411246	5.606639954940954
侠客行	3.7159361115102454	5.3804216590046465
倚天屠龙记	4.418043578802388	5.98799325341187
天龙八部	4.510281815778068	6.115810221421476
射雕英雄传	4.325375417604304	5.971089351750739
白马啸西风	2.706137774201019	4.093436717406064

碧血剑	3.742848286092471	5.6816879888707845
神雕侠侣	4.44720372910001	6.074512536464209
笑傲江湖	4.575876512832927	5.862778023869883
越女剑	1.7132445515388757	3.107162801180037
连城诀	3.3201280962623265	5.091428149708817
雪山飞狐	2.771613316533926	4.805613883781718
飞狐外传	3.769992432234081	5.575373372073747
鸳鸯刀	2.1015602246694196	3.657696972344133
鹿鼎记	4.697453075974613	6.0276713230762144
总语料库	6.251823149587034	7.0249753242588938

计算代码如下：

```
def calc_entropy_bigram( word, is_ci):
            # 计算二元模型的信息熵
          # 计算二元模型总词频
    word_tf = get_bigram_tf(word)
    last_word_tf = get_unigram_tf(word)
    bigram_len = sum([item[1] for item in word_tf.items()])
    entropy = []
    for bigram in word_tf.items():
        p_xy = bigram[1] / bigram_len  # 联合概率p(xy)
        p_x_y = bigram[1] / last_word_tf[bigram[0][0]]  # 条件概率
p(x|y)
        entropy.append(-p_xy * math.log(p_x_y, 2))
    entropy = sum(entropy)
    if is_ci:
        print("基于词的二元模型的中文信息熵为：{}比特/词".format( entropy))
    else:
        print("基于字的二元模型的中文信息熵为：{}比特/词".format( entropy))
    return entropy
```

三元信息熵的计算结果如表 3 所示：

表 3 三元信息熵

语料库	三元词信息熵/bit	三元字信息熵/bit
三十三剑客图	0.0690028273459445	0.6505453691721872
书剑恩仇录	0.4299373414457065	1.8644175706821904
侠客行	0.44403251478539596	1.8194743522174068
倚天屠龙记	0.562703834374333	2.2779474092353755
天龙八部	0.5557278783283545	2.352268338125409
射雕英雄传	0.46387330008677363	2.200807063667061
白马啸西风	0.2659168443735007	1.2123791366637682
碧血剑	0.3790929347201773	1.7977318419054964
神雕侠侣	0.5444993971890495	2.311048126429997
笑傲江湖	0.7248487907310865	2.3640596439125803
越女剑	0.22730698393287693	0.8398331803637774
连城诀	0.29858663540790087	1.6393030666516724
雪山飞狐	0.22753559219418004	1.3035978736600586
飞狐外传	0.3795923320311023	1.8672579270178753
鸳鸯刀	0.18712274961078876	0.8950502764090954
鹿鼎记	0.6565557171727556	2.4131868826837812
总语料库	1.1713483752894525	3.4951545345252544

程序如下：

```
def calc_entropy_trigram(word, is_ci):
        # 计算三元模型的信息熵
        # 计算三元模型总词频
    word_tf = get_trigram_tf(word)
```

```
    last_word_tf = get_bigram_tf(word)
    trigram_len = sum([item[1] for item in word_tf.items()])
    entropy = []
    for trigram in word_tf.items():
        p_xy = trigram[1] / trigram_len  # 联合概率p(xy)
        p_x_y = trigram[1] / last_word_tf[(trigram[0][0],
trigram[0][1])]  # 条件概率p(x|y)
        entropy.append(-p_xy * math.log(p_x_y, 2))
    entropy = sum(entropy)
    if is_ci:
        print("基于词的三元模型的中文信息熵为：{}比特/词".format(entropy))
    else:
            print("基于字的三元模型的中文信息熵为：{}比特/字
".format( entropy))
    return entropy
```

文本信息熵条形图如下所示：

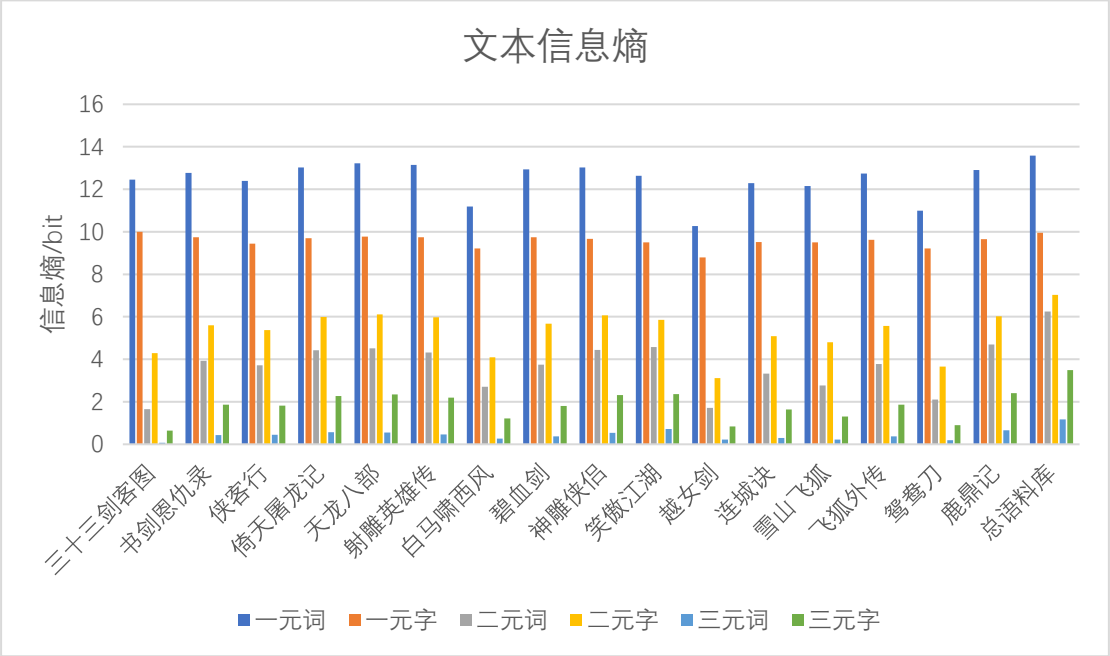

图 4 文本信息熵条形图