计算汉字信息熵

黄渲淇
2320634327@qq.com

Abstract

信息熵是信息论中的一个重要概念，用于描述随机变量的不确定度或信息量大小。在信息论中，信息量的大小与事件发生的概率成反比，即发生概率越小的事件所含信息量越大。信息熵是对所有可能事件的信息量的加权平均值，其中权值是各事件发生的概率。信息熵越大，表示随机变量的不确定度越高，所包含的信息量也就越大。基于信息熵的概念，本文基于 jieba 分词，使用 python 语言进行编程完成了汉字信息熵的计算，并对结果进行分析。

Introduction

在自然语言处理领域中，信息熵是一种重要的度量方式。

以二进制编码为例，如果一个事件只有两种可能的结果，比如抛一次硬币的结果可以是正面或反面，那么它的信息熵可以表示为：

$$H(X) = -P(x = \text{正面}) * \log_2{}^{P(x=\text{正面})} - P(x = \text{反面}) * \log_2{}^{P(x=\text{反面})}$$

其中 p(x=正面)表示正面朝上的概率，p(x=反面)表示反面朝上的概率，log2 表示以 2 为底的对数。如果硬币是均匀的，即正反面出现的概率都是 0.5，那么信息熵为 1，表示这个事件的信息量是最大的，因为我们在进行一次抛硬币的时候，需要得到的信息最多，我们不能预测硬币的结果；如果硬币有可能是偏重的，比如正面出现的概率是 0.8，反面出现的概率是 0.2，那么信息熵为 0.72，表示这个事件的信息量较小，因为我们可以大致预测硬币的结果。

对于一个文本来说，可以将每个单词或字符视为一个随机变量，计算它们的信息熵来衡量文本的信息量大小。例如，对于一段中文文本，可以将每个汉字视为一个随机变量，计算它们的信息熵来衡量文本的信息量大小。计算方法与二进制编码类似，只是需要换成中文字符集。汉字信息熵越大，表示这个文本所包含的信息量越大，反之则越小。 汉字信息熵在自然语言处理中有着广泛的应用。例如，可以将汉字信息熵作为文本特征之一，用于文本分类、聚类和信息检索等任务中。在文本分类中，汉字信息熵可以用来衡量每个类别的文本的信息量大小，选择信息量最大的特征作为分类器的输入；在文本聚类中，可以将文本按照汉字信息熵大小进行聚类，从而将信息量相似的文本归为一类；在信息检索中，可以使用汉字信息熵来计算文本之间的相似度，从而实现文本的检索和排序。 总之，信息熵是一个非常有用的概念，它可以用来衡量信息源的信息量大小，也可以用来衡量数据的不确定性和随机性。在自然语言处理中，汉字信息熵是一个重要的度量方式，它可以用来作为文本特征之一，用于文本分类、聚类和信息检索等任务中。

信息熵的计算公式：

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

其中，H(X)表示随机变量 X 的信息熵； P(x)表示随机变量 X 取值为 x 的概率； log2 表示以 2 为底的对数。 需要注意的是，信息熵的单位通常是比特（bit），表示所需的二进制位数，用来编码随机变量的信息。如果使用自然对数（以 e 为底），则单位是纳特（nat），表示所需的自然对数位数。

除却一元的信息熵计算，我们将信息熵拓展到多元，例如二元信息熵。二元信息熵是指一个由两个随机变量组成的系统的信息熵，可以用来衡量这个系统的不确定性和随机性。二元信息熵的计算方法与一元信息熵相似，只是需要考虑两个随机变量的联合分布以及它们的边缘分布。二元系统 X 和 Y 为例，其二元信息熵的计算公式如下：

$$H(X,Y) = -\sum\sum P(x,y)*\log_2{}^{P(x|y)}$$

其中，H(X,Y)表示二元系统 X 和 Y 的信息熵； P(x,y)表示二元系统 X 和 Y 取值分别为 x 和 y 的联合概率； log2 表示以 2 为底的对数。

多元汉字信息熵其对应的概率为按顺序出现该词语中除最后一个字外所有字的基础上，再出现最后一个字的条件概率公式：

$$H(X_1,X_2,X_3...X_n) = -\sum_{x_1,x_2,x_3...x_n \in X} P(x_1,x_2,x_3...x_n)\log_2{}^{P(x_n|x_1,x_2...x_{n-1})}$$

Methodology

1: 数据读取

读取停词表：

```
with open('./cn_stopwords.txt', 'r', encoding='utf-8') as f:
    stopwords = set([line.strip() for line in f])
```

停词表文件为 UTF-8 编码格式，需要对编码格式进行注意；

读取所给的文档库里的所有小说：

```
corpus_path = './datasets'
filenames = os.listdir(corpus_path)
docs = []
for filename in filenames:
    with open(os.path.join(corpus_path, filename), 'r', encoding='ansi') as f:
        text = f.read()
        # 分词并过滤停用词
        words = [word for word in text if word not in stopwords]
        docs.append(words)
```

上图为以字为单位时采用的读取文档方式，以字为单位我们就不对文档进行分词处理，当以词为单位时需要对文档进行分词操作。文档库内文档皆采用 ANSI 编码，在读取时需要进行更改。同时在整合文本时我们进行了对于停用词的删除。

2: 使用采用 jieba 分词进行中文文档的分词处理

```
words = [word for word in jieba.cut(text) if word not in stopwords]
```

jieba 分词是一款基于 Python 语言开发的中文分词工具，由于其高效、准确和易用性受到广泛的欢迎。jieba 分词采用了基于前缀词典和动态规划的分词算法，可以有效地处理中文文本中的分词问题。jieba 模式有三种，分别是精确模式、全模式和搜索引擎模式。搜索引擎模式：在这种模式下，jieba 会根据词语的位置和频率对文本进行分词，同时去除停用词和词典中不存在的词语。这种模式适用于对文本进行搜索引擎优化（SEO）和文本摘要等任务，故我们采用**搜索引擎模式**。

3: 字，词频率统计

在停用词去除，分词等预处理完成后，我们进行字，词频率的暴力统计。

```
if '\u4e00' <= char <= '\u9fa5':
```

用以判断是否为中文字/词组。

一元字/词组统计:

```
for char in text:
    if '\u4e00' <= char <= '\u9fa5':
        char_count[char] = char_count.get(char, 0) + 1
```

二元字/词组统计:

```
for i in range(len(text) - 1):
    if '\u4e00' <= text[i] <= '\u9fa5' and '\u4e00' <= text[i + 1] <= '\u9fa5':
        bigram_count[(text[i], text[i + 1])] = bigram_count.get((text[i], text[i + 1]), 0) + 1
        bigram_len+=1
```

三元字/词组统计:

```
for i in range(len(text) - 2):
    if '\u4e00' <= text[i] <= '\u9fa5' and '\u4e00' <= text[i + 1] <= '\u9fa5' and '\u4e00' <= text[i + 2] <= '\u9fa5':
        trp_count[((text[i], text[i + 1]), text[i + 2])] = trp_count.get(((text[i], text[i + 1]), text[i + 2]), 0) + 1
        trp_len += 1
```

4.信息熵计算

一元信息熵计算

```
# 计算每个汉字出现的概率
char_prob = {}  char_prob: {}
for char in char_count:
    char_prob[char] = char_count[char] / len(text)

# 计算信息熵
entropy = 0
for prob in char_prob.values():
    entropy -= prob * math.log(prob, 2)
```

二元信息熵计算

```
bigram_count = {}
bigram_len = 0
char_count = {}
for char in text:
    if '\u4e00' <= char <= '\u9fa5':
        char_count[char] = char_count.get(char, 0) + 1

for i in range(len(text) - 1):
    if '\u4e00' <= text[i] <= '\u9fa5' and '\u4e00' <= text[i + 1] <= '\u9fa5':
        bigram_count[(text[i], text[i + 1])] = bigram_count.get((text[i], text[i + 1]), 0) + 1
        bigram_len+=1

# 计算每个汉字二元词组出现的概率
bigram_prob = {}
entropy = []
for bigram in bigram_count.items():
    #bigram_prob[bigram] = bigram_count[bigram] / (bigram_len - 1)
    jp_xy = bigram[1] / bigram_len  # 计算联合概率p(x,y)
    cp_xy = bigram[1] / char_count[bigram[0][0]]  # 计算条件概率p(x|y)
    entropy.append(-jp_xy * math.log(cp_xy, 2))  # 计算二元模型的信息熵

# 计算信息熵
return round(sum(entropy), 6)
```

Char_count{}为一元统计值，bigram_count{}为二元统计值，二元分布需要计算联合分布和条件概率，以上为具体代码实现。

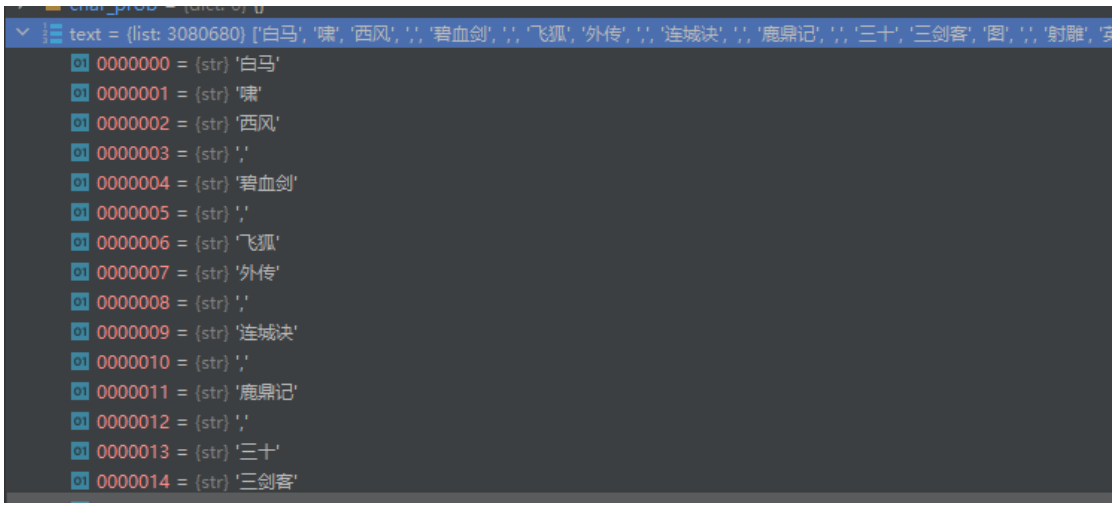
三元信息熵计算

```
# 定义计算汉字三元词组信息熵的函数
def calc_chinese_trp_entropy(text):
    # 统计每个汉字三元词组出现的次数
    bigram_count = {}
    bigram_len = 0
    trp_count = {}
    trp_len = 0
    for i in range(len(text) - 2):
        if '\u4e00' <= text[i] <= '\u9fa5' and '\u4e00' <= text[i + 1] <= '\u9fa5' and '\u4e00' <= text[i + 2] <= '\u9fa5':
            trp_count[((text[i], text[i + 1]), text[i + 2])] = trp_count.get(((text[i], text[i + 1]), text[i + 2]), 0) + 1
            trp_len += 1
    for i in range(len(text) - 1):
        if '\u4e00' <= text[i] <= '\u9fa5' and '\u4e00' <= text[i + 1] <= '\u9fa5':
            bigram_count[(text[i], text[i + 1])] = bigram_count.get((text[i], text[i + 1]), 0) + 1
            bigram_len += 1

    # 计算每个汉字三元词组出现的概率
    entropy = []
    for bigram in trp_count.items():
        #bigram_prob[bigram] = bigram_count[bigram] / (bigram_len - 1)
        jp_xy = bigram[1] / trp_len  # 计算联合概率p(x,y)
        cp_xy = bigram[1] / bigram_count[bigram[0][0]]  # 计算条件概率p(x|y)
        entropy.append(-jp_xy * math.log(cp_xy, 2))  # 计算二元模型的信息熵

    # 计算信息熵
    return round(sum(entropy), 6)
```

三元信息熵计算类比二元。

Experimental Studies

预处理部分——分词结果



词频统计

一元词数统计结果


```
{'白马': 294, '啸': 8, '西风': 11, '碧血剑': 9, '飞狐': 40, '外传': 36, '连城诀': 32, '鹿鼎记': 16, '三十': 97, '三剑客': 5, '图': 126, '射雕': 53, '英雄传': 40, '神雕侠侣': 49, '书剑': 26, '恩仇录': 25, '天龙八部': 22, '侠客行': 34, '笑傲江湖': 57, '雪山飞狐': 57, '倚天': 68, '屠龙记': 40, '鸳鸯刀': 50, '越女剑': 20, '本书': 111, '免费': 58, '小说': 132, '下载站': 28, '更': 2375, '更新': 30, '电子书': 28, '请': 3782, '关注': 52, '旧': 92, '插图': 11, '绣像': 3, '我国': 36, '向来': 676, '传统': 17, '喜欢': 860, '读旧': 1, '中': 14060, '可惜': 836, '美术': 3, '水准': 2, '文学': 19, '差得': 40, '实在': 1036, '太远': 83, '木版画': 1, '雕刻': 17, '木版': 2, '印出来': 2, '往往': 260, '画得': 15, '粗俗': 32, '刻得': 10, '简陋': 38, '极少数': 2, '例外': 12, '版画': 6, '悠久': 2, '历史': 75, '最古': 3, '作品': 5, '汉代': 4, '肖形印': 1, '印章': 3, '上刻': 64, '龙虎': 10, '鸾鸟': 6, '图印': 1, '印在': 6, '绢': 14, '纸': 58, '成为': 295, '精美': 10, '巧丽': 1, '图形': 141, '成长': 6, '隋唐': 4, '时': 3921, '佛画': 1, '盛于宋': 1, '元': 51, '明末': 23, '登峰造极': 29, '最大': 127, '艺术家': 1, '陈洪绶': 1, '老莲': 1, '清代': 6, '普遍': 5, '发展': 12, '年画': 1, '盛行于': 1, '民间': 29, '咸丰': 2, '年间': 82, '任胃长': 4...
```


Table 1: this is the table 1

二元词数统计结果


```
{('白马', '啸'): 2, ('啸', '西风'): 2, ('飞狐', '外传'): 29, ('三十', '三剑客'): 2, ('三剑客', '图'): 5, ('射雕', '英雄传'): 40, ('书剑', '恩仇录'): 25, ('倚天', '屠龙记'): 39, ('越女剑', '本书'): 1, ('小说', '下载站'): 28, ('更', '更新'): 28, ('更新', '免费'): 28, ('免费', '电子书'): 28, ('电子书', '请'): 28, ('请', '关注'): 28, ('旧', '小说'): 6, ('小说', '插图'): 1, ('插图', '绣像'): 1, ('绣像', '我国'): 1, ('我国', '向来'): 1, ('向来', '传统'): 2, ('传统', '喜欢'): 1, ('喜欢', '读旧'): 1, ('读旧', '小说'): 1, ('小说', '喜欢'): 2, ('喜欢', '小说'): 1, ('小说', '中'): 14, ('中', '插图'): 1, ('插图', '可惜'): 1, ('可惜', '插图'): 1, ('插图', '美术'): 1, ('美术', '水准'): 1, ('水准', '小说'): 1, ('小说', '文学'): 1, ('文学', '水准'): 1, ('水准', '差得'): 1, ('差得', '实在'): 1, ('实在', '太远'): 4, ('太远', '插图'): 1, ('插图', '木版画'): 1, ('木版画', '雕刻'): 1, ('雕刻', '木版'): 1, ('木版', '印出来'): 1, ('印出来', '往往'): 1, ('往往', '画得'): 1, ('画得', '粗俗'): 1, ('粗俗', '刻得'): 1, ('刻得', '简陋'): 1, ('简陋', '极少数'): 1, ('极少数', '例外'): 1, ('例外', '我国'): 1, ('我国', '版画'): 1, ('版画', '悠久'): 1, ('悠久', '历史'): 1, ('历史', '最古'): 1, ('最古', '版画'): 1, ('版画', '作品'): 1, ('作品', '汉代'): 1, ('汉代', '肖形印'): 1, ('肖形印', '印章'): 1, ('印章', '上刻'): 1, ('上刻', '龙虎'): 1, ('龙虎', '鸾鸟'): 1, ('鸾鸟', '图印'): 1, ('图印', '印在'): 1, ('印在', '绢'): 1, ('绢', '纸'): 1, ('纸', '成为'): 1, ('成为', '精美'): 1, ('精美', '巧丽'): 1, ('巧丽', '图形'): 1, ('图形', '版画'): 1, ('版画', '成长'): 1, ('成长', '隋唐'): 1, ('隋唐', '时'): 1, ('时', '佛画'): 1, ('佛画', '盛于宋'): 1, ('盛于宋', '元'): 1, ('元', '明末'): 1, ('明末', '登峰造极'): 1, ('登峰造极', '最大'): 1, ('最大', '艺术家'): 1, ('艺术家', '陈洪绶'): 1, ('陈洪绶', '老莲'): 1, ('老莲', '清代'): 1, ('清代', '版画'): 1, ('版画', '普遍'): 1, ('普遍', '发展'): 1, ('发展', '年画'): 1, ('年画', '盛行于'): 1, ('盛行于', '民间'): 1, ('民间', '咸丰'): 1, ('咸丰', '年间'): 1, ('年间', '任胃长'): 1, ('任胃长', '认为'): 1, ('认为', '我国'): 1, ('我国', '传统'): 4, ('传统', '版画'): 1, ('版画', '最后'): 1, ('最后', '一位'): 4...
```

三元词数统计结果

信息熵计算

以字为单位信息熵计算

	一元	二元	三元
Inf	4.717697	9.191179	3.647261
白马啸西风	9.912360	0.361702	6.020877
碧血剑	9.619711	4.283907	0.032258
飞狐外传	8.984227	5.604615	0.648430
连城诀	9.410856	5.371277	1.869467
鹿鼎记	9.404501	5.984745	1.830657
三十三剑客图	9.194147	6.100858	2.283771
射雕英雄传	7.842424	5.967590	2.361850
神雕侠侣	9.562418	4.073626	2.207551
书剑恩仇录	9.154738	5.678616	1.221750
天龙八部	9.289717	6.072215	1.803091
侠客行	7.576116	5.857965	2.311292
笑傲江湖	9.029551	3.113173	2.371777
雪山飞狐	9.100415	5.082941	0.866909
倚天屠龙记	9.441353	4.796672	1.655784
鸳鸯刀	8.077478	5.571576	1.312636
越女剑	4.717697	9.191179	1.873082
All 文本	9.549075	7.022063	3.500738

以词为单位信息熵

	一元	二元	三元
Inf	3.25255	0.0	0.0
白马啸西风	12.35921	1.812049	0.090574
碧血剑	12.45575	4.152879	0.500141
飞狐外传	11.36478	4.012654	0.519701
连城诀	12.27453	4.701541	0.646318
鹿鼎记	12.22015	4.848591	0.672178
三十三剑客图	11.88821	4.608831	0.534064
射雕英雄传	8.81126	2.916233	0.359809
神雕侠侣	12.48856	3.97712	0.432002
书剑恩仇录	11.79710	4.791778	0.643233
天龙八部	12.06764	4.854165	0.799899
侠客行	8.413602	1.770923	0.249675
笑傲江湖	11.23763	3.60526	0.372712
雪山飞狐	11.28166	3.083038	0.293971
倚天屠龙记	12.24990	4.051713	0.464051
鸳鸯刀	9.10370	2.176664	0.233856
越女剑	11.69710	5.024732	0.842603
All 文本	12.740208	6.541074	1.181922

Conclusions

汉字信息熵是一种用于衡量汉字信息量大小的指标。汉字信息熵越大，表示该汉字所包含的信息量越大，反之则越小。汉字信息熵的大小可以反映出汉字的复杂程度、使用频率等信息。 在信息论中，熵是指随机变量不确定性的度量，即某个事件发生的概率越小，其熵值就越大。对于汉字来说，每个汉字在汉语中的使用频率不同，有些汉字使用频率较高，如“的”、“是”等，而有些汉字使用频率较低，如“龘”、“亡”等。使用频率高的汉字信息熵较小，使用频率低的汉字信息熵较大。另一方面，可以看到三元词组包含的信息熵小于二元词组，二元词组小于一元，这是显然可以看出的当对一个事物描述越具体的时候起信息量越大，意义更明确。同时词的信息熵小于字的信息熵，亦是因为词组相较于单个字更能有确切的意思。

附录

```
import math
import jieba
import os

# 读取停用词表
with open('./cn_stopwords.txt', 'r', encoding='utf-8') as f:
    stopwords = set([line.strip() for line in f])
# 读取中文文档库中的所有文档
corpus_path = './datasets'
filenames = os.listdir(corpus_path)
docs = []
for filename in filenames:
    with open(os.path.join(corpus_path, filename), 'r', encoding='ansi') as f:
        text = f.read()
        # 分词并过滤停用词
        words = [word for word in jieba.cut(text) if word not in stopwords]
        docs.append(words)
# 将所有文档的分词结果合并为一个列表
all_words = []
for doc in docs:
    all_words.extend(doc)
# 去重
unique_words = set(all_words)

# 定义计算汉字信息熵的函数
def calc_chinese_entropy(text):
    # 统计每个汉字出现的次数
    char_count = {}
    for char in text:
        if '\u4e00' <= char <= '\u9fa5':
            char_count[char] = char_count.get(char, 0) + 1

    # 计算每个汉字出现的概率
    char_prob = {}
    for char in char_count:
        char_prob[char] = char_count[char] / len(text)

    # 计算信息熵
    entropy = 0
    for prob in char_prob.values():
        entropy -= prob * math.log(prob, 2)
```

```
    return entropy


# 定义计算汉字二元词组信息熵的函数
def calc_chinese_bigram_entropy(text):
    # 统计每个汉字二元词组出现的次数
    bigram_count = {}
    bigram_len = 0
    char_count = {}
    for char in text:
        if '\u4e00' <= char <= '\u9fa5':
            char_count[char] = char_count.get(char, 0) + 1

    for i in range(len(text) - 1):
        if '\u4e00' <= text[i] <= '\u9fa5' and '\u4e00' <= text[i + 1] <= '\u9fa5':
            bigram_count[(text[i], text[i + 1])] = bigram_count.get((text[i], text[i + 1]),
0) + 1
            bigram_len+=1

    # 计算每个汉字二元词组出现的概率
    bigram_prob = {}
    entropy = []
    for bigram in bigram_count.items():
        #bigram_prob[bigram] = bigram_count[bigram] / (bigram_len - 1)
        jp_xy = bigram[1] / bigram_len  # 计算联合概率 p(x,y)
        cp_xy = bigram[1] / char_count[bigram[0][0]]  # 计算条件概率 p(x|y)
        entropy.append(-jp_xy * math.log(cp_xy, 2))  # 计算二元模型的信息熵

    # 计算信息熵
    return round(sum(entropy), 6)

# 定义计算汉字三元词组信息熵的函数
def calc_chinese_trp_entropy(text):
    # 统计每个汉字三元词组出现的次数
    bigram_count = {}
    bigram_len = 0
    trp_count = {}
    trp_len = 0
    for i in range(len(text) - 2):
        if '\u4e00' <= text[i] <= '\u9fa5' and '\u4e00' <= text[i + 1] <= '\u9fa5' and
'\u4e00' <= text[i + 2] <= '\u9fa5':
            trp_count[((text[i], text[i + 1]), text[i + 2])] = trp_count.get(((text[i],
text[i + 1]), text[i + 2]), 0) + 1
```

```
            trp_len += 1
    for i in range(len(text) - 1):
        if '\u4e00' <= text[i] <= '\u9fa5' and '\u4e00' <= text[i + 1] <= '\u9fa5':
            bigram_count[(text[i], text[i + 1])] = bigram_count.get((text[i], text[i + 1]),
0) + 1
            bigram_len += 1

    # 计算每个汉字三元词组出现的概率
    entropy = []
    for bigram in trp_count.items():
        #bigram_prob[bigram] = bigram_count[bigram] / (bigram_len - 1)
        jp_xy = bigram[1] / trp_len  # 计算联合概率 p(x,y)
        cp_xy = bigram[1] / bigram_count[bigram[0][0]]  # 计算条件概率 p(x|y)
        entropy.append(-jp_xy * math.log(cp_xy, 2))  # 计算二元模型的信息熵

    # 计算信息熵
    return round(sum(entropy), 6)


#计算所有文档的信息熵
for i, doc in enumerate(docs):
    entropy = calc_chinese_entropy(doc)
    print("文档{}的一元信息熵为：{}".format(i+1, entropy))
for i, doc in enumerate(docs):
    entropy = calc_chinese_bigram_entropy(doc)
    print("文档{}的二元信息熵为：{}".format(i+1, entropy))
for i, doc in enumerate(docs):
    entropy = calc_chinese_trp_entropy(doc)
    print("文档{}的三元信息熵为：{}".format(i+1, entropy))

# 计算整个文库的信息熵和二元词组信息熵
char_entropy = calc_chinese_entropy(all_words)
bigram_entropy = calc_chinese_bigram_entropy(all_words)
trp_entropy = calc_chinese_trp_entropy(all_words)
print("整个文库的汉字信息熵为：", char_entropy)
print("整个文库的汉字二元词组信息熵为：", bigram_entropy)
print("整个文库的汉字三元词组信息熵为：", trp_entropy)
```