金庸小说中文信息熵

——自然语言处理的深度学习报告(第一次)

崔潇雅

ZY2203502

1 摘要

信息熵常被用来作为一个系统的信息含量的量化指标，从而可以进一步用来作为系统方程优化的目标或者参数选择的判据。本文首先介绍了利用基于词的一元模型、二元模型、三元模型估计中文信息熵的计算方法，并基于 16 本金庸小说在分词和字符模式下计算三种统计语言模型的中文信息熵，分词模式 1-3 元平均信息熵分别为 **12.2825bit、3.8996bit、0.5104125bit**，字符模式 1-3 元信息熵平均分别为 **9.5505bit、5.1293bit、1.7403bit**。

2 论文理论基础

2.1: 信息熵

“熵”是由香农在 1948 年首次引入到信息论中, 用于衡量一个离散随机变量的不确定性。通过阅读论文可知，设 $\mathrm{X}=\{...X_{-2}，X_{-1}，X_0，X_1，X_{-2}...\}$是离散随机变量，P（$X_i$）表示事件$X_i$的发生概率, 其熵值被定义为:

$$\mathrm{H(X)}=\mathrm{H(P)}=\lim_{n\to\infty}-E_P\,log\,P(X_0|X_{-1},X_{-2},...)$$

当底数选用 2 时，熵值单位为 bit，此时公式表达为：

$$\mathrm{H(P)}=\lim_{n\to\infty}-E_P\,log\,P(X_0|X_{-1},X_{-2},...)=\lim_{n\to\infty}-\frac{1}{n}E_P\,log\,P(X_1X_2...X_n)$$

如果 P 是遍历的，期望为 1。此时无法精确获取 P，通过建立 P 的平稳随机过程模型 M 来估计 H(P)上界，公式表达为：

$$\mathrm{H(P,M)}=\lim_{n\to\infty}-\frac{1}{n}log\,P(X_1X_2...X_n)$$

从文本压缩的角度理解信息熵，对于$X_1X_2...X_n$任意编码方式，$l(X_1X_2...X_n)$为编码需要的比特数，都有：

$$E_{pl}(X_1X_2...X_n)\geq-E_{pl}logP(X_1X_2...X_n)$$

可知， 是对从 P 中提取的长字符串进行编码所需的每个符号的平均位数的下限，每个符号编码时需要的位数越多，即熵越高，说明混乱程度越高，单个字符携带的信息量越大。

计算中文信息熵首先统计文本中每个汉字出现频次与频率再计算其熵值：

$$H(X) = -\sum P(X_i)\log P(X_i)$$

2.2: 语言模型（N-Gram）

在马尔可夫假设下，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，这样我们前面得到的条件概率的计算可以简化如下:

$$P(W_i|W_1\cdots W_{i-1}) = P(W_i|W_{i-k}\cdots W_{i-1})$$

K=0 时，一元模型(unigram)：

$$P(W_1, W_2, \cdots, W_k) = \prod_i P(W_i)$$

K=1 时，二元模型(bigram）：

$$P(W_1, W_2, \cdots, W_k) = \prod_i P(W_i|W_{i-1})$$

K=2 时，三元模型(trigram)：

$$P(W_1, W_2, \cdots, W_k) = \prod_i P(W_i|W_{i-2}，\ W_{i-1})$$

本文分别计算 1、2、3 元模型，计算 16 本小说单独信息熵。

3 实验流程

3.1 预处理获取停词库

```python
# 获取停词库
def get_stop_word(stop_word_file):
    stop_word_list = []
    with open(stop_word_file, "r", encoding="utf-8") as file_to_read:
        for line in file_to_read:
            stop_word_list.append(line.strip())
    return stop_word_list
# 获取标点符号库
def get_punctuation_word(punctuation_file):
    punctuation_list = []
    with open(punctuation_file, "r", encoding="utf-8") as file_to_read:
        for line in file_to_read:
            punctuation_list.append(line.strip())
    return punctuation_list
```

3.2 结巴分词并去掉停词、标点

```python
def get_cleaned_word_list(sentence, stop_word_list, punctuation_list):
    word_list = jieba.lcut(sentence)
    cleaned_word_list = []
    for word in word_list:
        if word in stop_word_list or word in punctuation_list:
            continue
        cleaned_word_list.append(word)
    return cleaned_word_list
def get_cleaned_charater_list(sentence, stop_word_list, punctuation_list):
    cleaned_charater_list = []
    for word in sentence:
        if word in stop_word_list or word in punctuation_list:
            continue
        cleaned_charater_list.append(word)
    return cleaned_charater_list
```

3.3 获取 N 元语言模型

```
def getNmodel(phrase_model, n, words_list):
    if n == 1:
        for i in range(len(words_list)):
            phrase_model[words_list[i]] = phrase_model.get(words_list[i], 0) + 1
    else:
        for i in range(len(words_list) - (n - 1)):
            if n == 2:
                condition_t = words_list[i]
            else:
                condition = []
                for j in range(n-1):
                    condition.append(words_list[i + j])
                condition_t = tuple(condition)
            phrase_model[(condition_t, words_list[i+n-1])] = phrase_model.get((condition_t, words_list[i+n-1]), 0) + 1
    return phrase_model
```

3.4 获取 N 元信息熵

```
def getNentropy(n, clean_zh_file_content):
    if n == 1:
        phrase_model = getNmodel({}, 1, clean_zh_file_content)
        model_lenth = len(clean_zh_file_content)
        entropy = sum(
            [-(phrase[1] / model_lenth) * math.log(phrase[1] / model_lenth, 2) for phrase in phrase_model.items()])
    elif n>1:
        phrase_model_pre = getNmodel({}, n-1, clean_zh_file_content)
        phrase_model = getNmodel({}, n, clean_zh_file_content)
        phrase_n_len = sum([phrase[1] for phrase in phrase_model.items()])
        entropy = 0
        for n_phrase in phrase_model.items():
            p_xy = n_phrase[1] / phrase_n_len
            p_x_y = n_phrase[1] /  phrase_model_pre[n_phrase[0][0]]
            entropy+=(-p_xy * math.log(p_x_y, 2))
    return entropy
```

3.5 画图

```
def draw_img(imgs_folder, zh_file_entropy, type="word"):
    x_axis = [key[:-4] for key in zh_file_entropy.keys()] # 书名为x轴
    entropy_one = [value[0] for key,value in zh_file_entropy.items()] # 信息熵为y轴
    entropy_two = [value[1] for key,value in zh_file_entropy.items()]
    entropy_three = [value[2] for key, value in zh_file_entropy.items()]
    entropy = []
    entropy.append(entropy_one)
    entropy.append(entropy_two)
    entropy.append(entropy_three)

    # 解决图片中中文乱码解决
    plt.rcParams['font.family'] = ['Arial Unicode MS', 'Microsoft YaHei', 'SimHei', 'sans-serif']
    plt.rcParams['axes.unicode_minus'] = False
    # 遍历画图
    for index in range(len(entropy)):
        for i in range(len(x_axis)):
            plt.bar(x_axis[i], entropy[index][i], width=0.5)
        if type=="word":
            plt.title(str(index+1)+"元信息熵词分析")
        else:
            plt.title(str(index + 1) + "元信息熵字分析")
        # 设置x轴标签名
        plt.xlabel("书名")
        # 设置y轴标签名
        plt.ylabel("信息熵")
        # 显示
        plt.xticks(fontsize=7)
        plt.show()
        plt.savefig(os.path.join(imgs_folder,str(index)+".jpg"))
```

4 实验结果

4.1 以词为单位

表 1　金庸小说单本信息熵（分词模式）

#	小说名	1-gram	2-gram	3-gram
1	三十三剑客图	12.2867	3.9907	0.5137
2	书剑恩仇录	12.7270	4.1355	0.4987
3	侠客行	12.2867	3.9907	0.5137
4	倚天屠龙记	12.8937	4.6847	0.6430
5	天龙八部	13.0182	4.8388	0.6648
6	射雕英雄传	13.0447	4.5945	0.5340
7	白马啸西风	11.1953	2.8796	0.3540
8	碧血剑	12.8819	3.9640	0.4335
9	神雕侠侣	12.3760	4.9079	0.7905
10	笑傲江湖	12.5235	4.8371	0.7963
11	越女剑	10.5012	1.7368	0.2346
12	连城诀	12.2068	3.5889	0.3692
13	雪山飞狐	12.1779	3.0655	0.2908
14	飞狐外传	12.6263	4.0400	0.4612
15	鸳鸯刀	11.1362	2.1465	0.2323
16	鹿鼎记	12.6378	4.9925	0.8363
17	平均值	12.2825	3.8996	0.5104

4.2 以字为单位

表 2　金庸小说单本信息熵（字符模式）

#	小说名	1-gram	2-gram	3-gram
1	三十三剑客图	10.0092	4.2807	0.6533
2	书剑恩仇录	9.7594	5.5962	1.8615
3	侠客行	9.4366	5.3777	1.8194
4	倚天屠龙记	9.7076	5.9823	2.2759
5	天龙八部	9.7842	6.1143	2.3514
6	射雕英雄传	9.7538	5.9656	2.1951
7	白马啸西风	9.2251	4.0887	1.2116
8	碧血剑	9.7578	4.5358	1.7942
9	神雕侠侣	9.5507	6.0300	2.3680
10	笑傲江湖	9.5176	5.8547	2.3611
11	越女剑	8.7855	3.1083	0.8419
12	连城诀	9.5163	5.0897	1.6390
13	雪山飞狐	9.5019	4.8010	1.3031
14	飞狐外传	9.6313	5.5680	1.8650
15	鸳鸯刀	9.2108	3.6565	0.8961
16	鹿鼎记	9.6606	6.0188	2.4089
17	平均值	9.5505	5.1293	1.7403

4.3 绘图

4.3.1 词分析

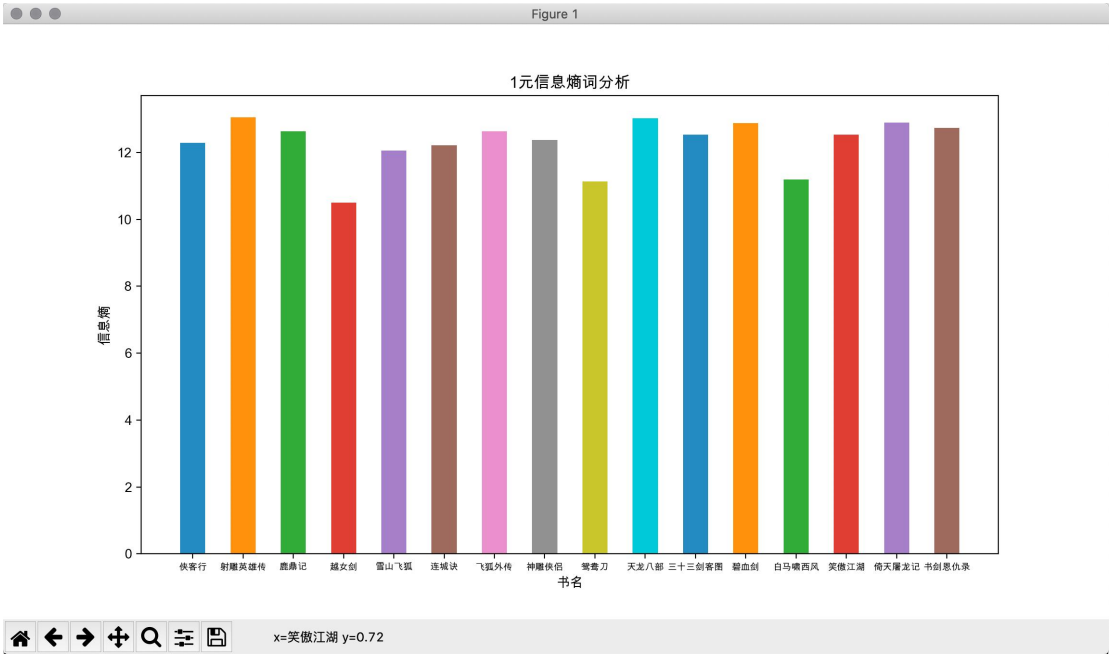

图 1　1-gram-byword

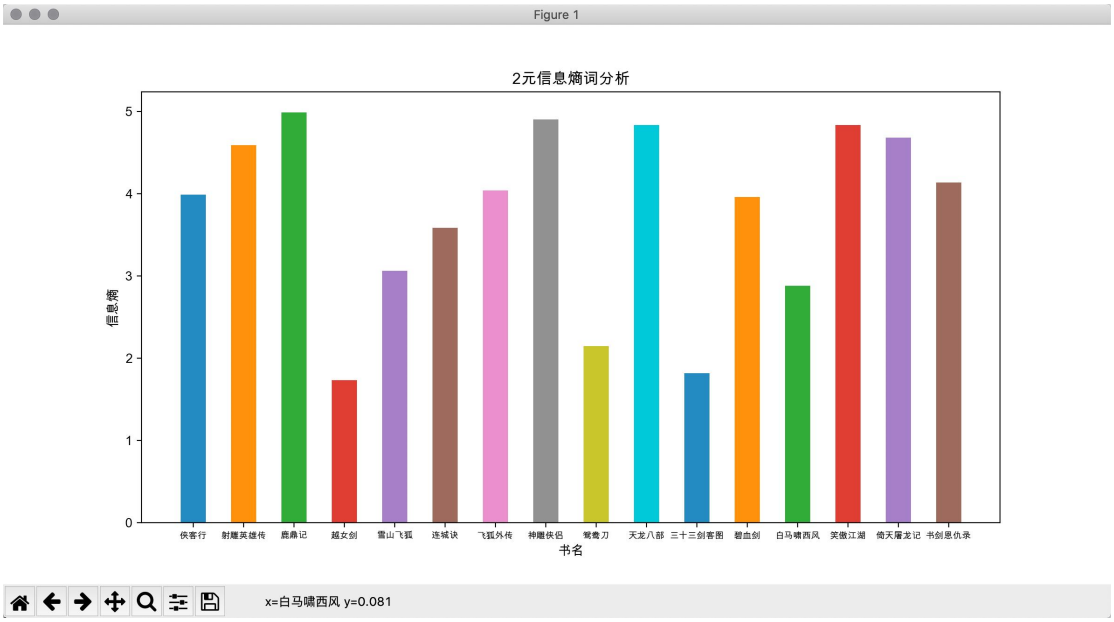

图 2　2-gram-byword

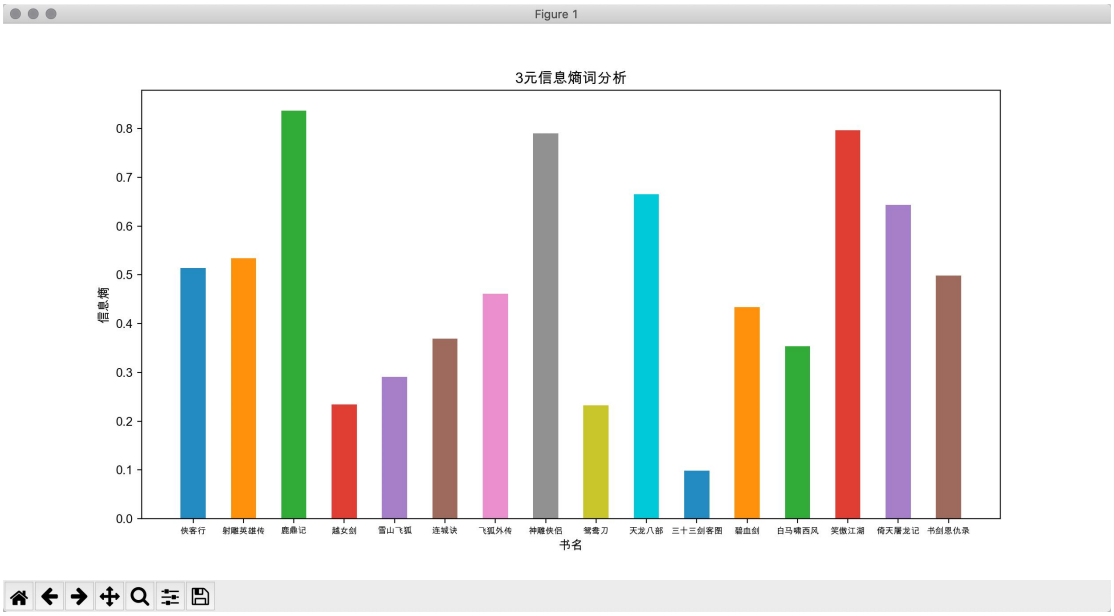

图 3　3-gram-byword

4.3.2 字分析

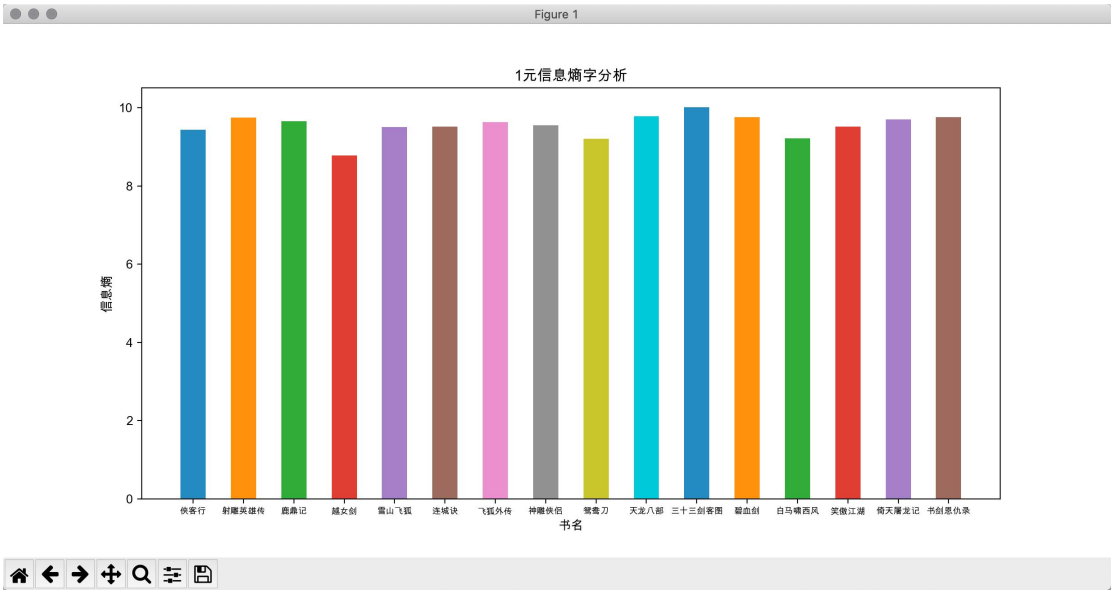

图 4　1-gram-bychar

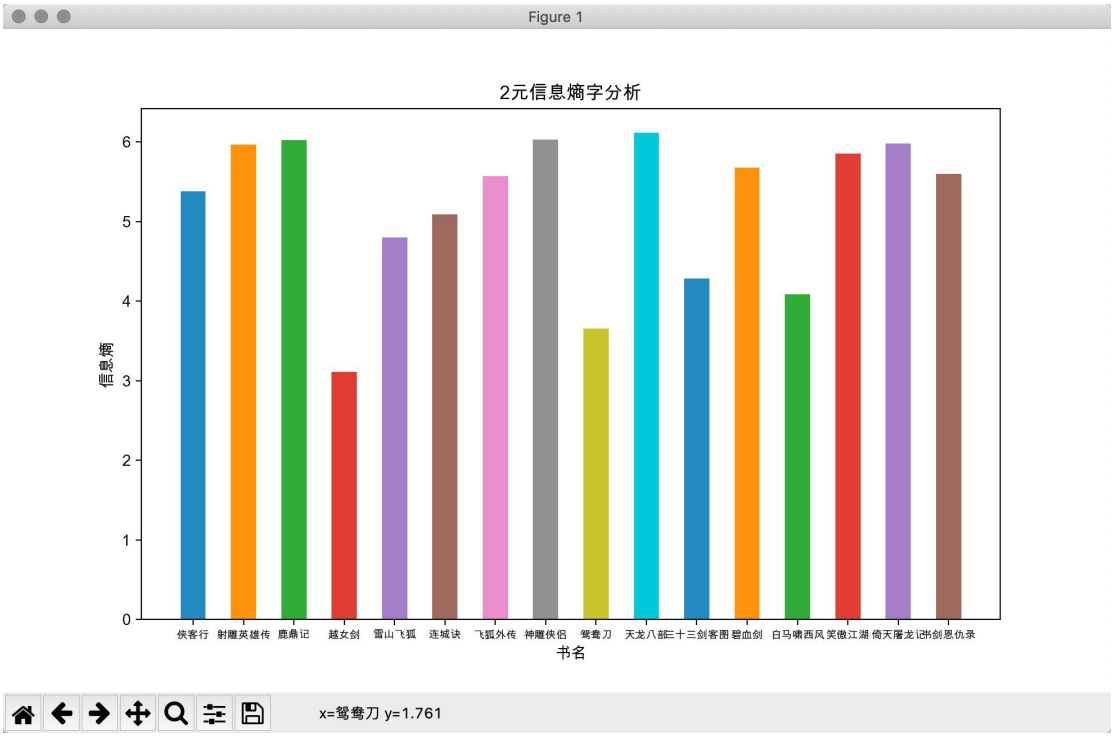

图 5　2-gram-bychar

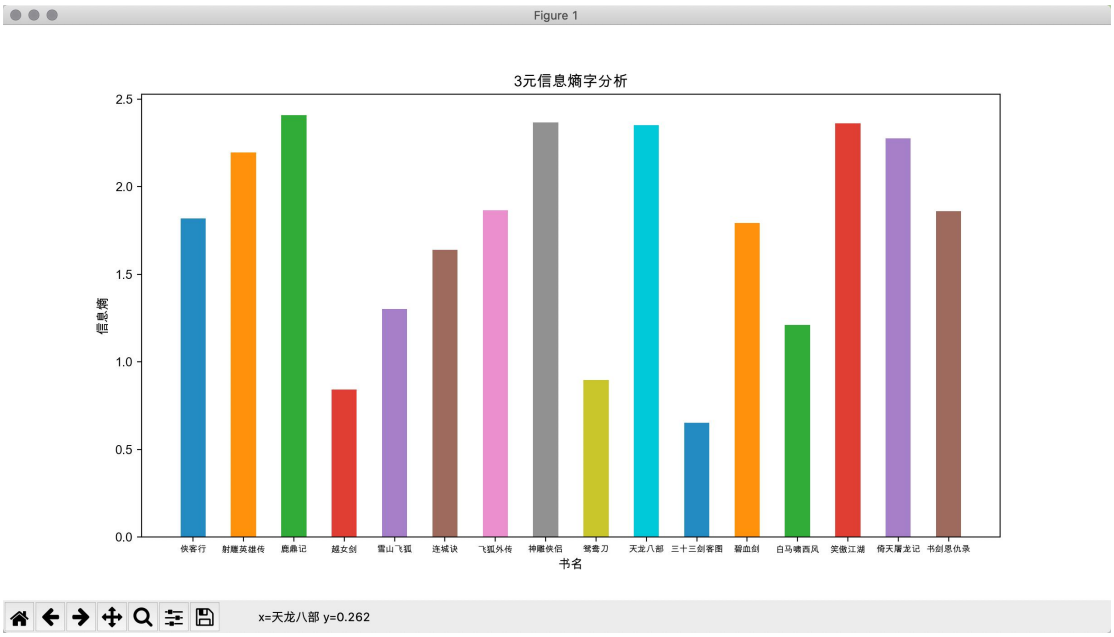

图 6　3-gram-bychar

5 总结

5.1 存在问题

程序在 pycharm 上运行正常，但是在 spyder 上读取文件时就出现中文　乱码，应该是环境变量配置问题，还没有解决。

5.2 N-Gram 模型

在分词模式下，随着 N 从 1、2、3 增大，单本信息熵呈现出下降趋势，说明 N 越大固定的词语越多，词组分布更加简洁，减少了短字词的不确定性打乱文章的机会，也就减少了文本信息熵。在字符模式下，单本信息熵也呈现以上趋势，符合规律。

5.3 中英文信息熵区别

论文中计算出英文单个词信息熵为 1.75bit,而本实验中计算出中文分词模式下一元信息熵平均值为 12.2825，字符模式下一元信息熵为 9.5505，相对于英文一元信息熵都要大，这是因为中文包含的信息更多，复杂程度更高。

5.4 分词模式与字符模式

分词模式 1-3 元平均信息熵分别为 12.2825、3.8996、0.5104125，字符模式 1-3 元信息熵平均分别为 9.5505、5.1293、1.7403。在一元、二元下分词模式信息熵高于字符模式，但是在三元下反而字符模式信息熵高于分析模式。分析是因为一元、二元时不同字符组成词语的方式更加多样，这导致词数目远大于字符数目，即词信息熵更大，而在三元时，词语之间出现较多的固定搭配，而字符的组合这是更加随机，因此字符信息熵更大。

6 参考文献

[1] (12 条消息)【NLP】中文平均信息熵_平均信息熵公式_AngeloG 的博客-CSDN 博客