基于中文语料库的齐夫定律验证及中文平均信息熵计算

ZY2303217 郑茗畅
zy2303217@buaa.edu.cn

Abstract

本文首先介绍了Zipf's Law（**齐夫定律**）和N元语言统计模型估计中文信息熵的方法；然后基于给定的16篇金庸武侠小说语料库，对Zipf's Law进行验证；同时在阅读Peter Brown的《An Estimate of an Upper Bound for the Entropy of English》一文，学习并理解信息熵的概念和计算信息熵的方法后，分别以词和字为单位，利用一元、二元以及三元模型计算中文的平均信息熵。

Introduction

1. Zipf's Law

Zipf's Law（**齐夫定律**）是由美国哈佛大学的语言学家乔治·金斯利·齐夫于1949年提出的词频分布定律。他在对莎士比亚等作家的作品进行数理统计后建立了这个实验定律[1]。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。

齐夫定律可以表述为：如果把一篇较长文章中每个词出现的频次统计起来，按照高频词在前、低频词在后的递减顺序排列，并用自然数给这些词编上等级序号，即频次最高的词等级为1，频次次之的词等级为2，……，频次最小的词等级为D。若用f表示频次，r表示等级序号，则有$f \times r = C$，其中C为常数。以f为纵坐标、r为横坐标的图像为一条双曲线。如果等级r和频次f都取对数，则有$\ln(f) + \ln(r) = \ln(C)$，此时以$\ln(f)$为纵坐标、$\ln(r)$为横坐标的图像变成一条直线[2]。

齐夫定律的适用范围不局限于词频分布领域，相反，它是描述一系列实际现象的特点非常到位的经验定律之一。它认为，如果我们按照大小或者流行程度给某个大集合中的各项进行排序，集合中第二项的比重大约是第一项的一半，而第三项的比重大约是第一项的三分之一，以此类推。换句话来说，一般来讲，排在第k位的项目其比重为第一项的$\frac{1}{k}$。其在情报学、地理学、经济学、信息科学等领域也有了广泛的应用，且取得了不少可喜成果。

2. 信息熵

1948年信息论之父克劳德·香农提出了“信息熵”的概念，解决了对信息的量化度量问题。信息熵通常是用来衡量一个事物的有序程度。熵值越大，表明该事物所处的状态越稳定；熵值越小，则表明该事物所处的状态越不稳定[3]。此外，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。信息熵的定义公式为：

$$H(x) = -\sum p(x) log_b(p(x))$$

其中$p(x)$为每种对应信源发生的概率分布，b为对数所使用的底，通常为2，此时熵的单位是bit；当$b = e$，熵的单位是mat；当$b = 10$，熵的单位是Hart。且规定：

$$0 \log(0) = 0$$

信息熵的性质有：①单调性：发生概率越高的事件，其携带的信息量越低；②非负性：收到一个信源所获得的信息量应为正值；③累加性：多事件同时发生存在的总不确定性可以表示为各事件不确定性的和；④极大值性：当概率分布的所有可能事件均等概率出现时，信息熵达到极大值。满足极大熵的概率分布是均匀分布；⑤条件熵特性：条件熵衡量已知某个随机变量的取值情况下，另一个随机变量的不确定性或信息量的平均量。条件熵的值始终大于等于零，并且在两个随机变量相互独立时达到最小值。

3. N-gram语言模型

N-gram语言模型是一种输入为一个句子，输出为该句子的概率的语言模型。N-gram语言模型引入了马尔可夫假设，这是一种独立性假设，此处指某一个词语出现的概率只由其前面的n-1个词语所决定，而与其他词都无关[4]。

假定S表示某个有意义的句子，其由一连串特定顺序排列的词$\omega_1, \omega_2, \cdots, \omega_n$组成，n为句子长度，则S在文本中出现的可能性为：

$$p(S) = p(\omega_1, \omega_2, \cdots, \omega_n)$$

利用条件概率公式得到：

$$p(\omega_1, \omega_2, \cdots, \omega_n) = p(\omega_1)p(\omega_2|\omega_1) \cdots p(\omega_n|\omega_1, \omega_2, \cdots, \omega_{n-1})$$

其中$p(\omega_1)$表示第一个词ω_1出现的概率；$p(\omega_2|\omega_1)$是在已知第一个词为ω_1的前提下，第二个词ω_2出现的概率，后面以此类推。

显然当句子长度过长时，$p(\omega_n|\omega_1, \omega_2, \cdots, \omega_{n-1})$的可能性太多，无法估算。假设该句子具有马尔可夫性，即任意一个词ω_i出现的概率只同它前面的词ω_{i-1}有关，S的概率变为：

$$p(S) = p(\omega_1)p(\omega_2|\omega_1) \cdots p(\omega_n|\omega_{n-1})$$

其对应的统计语言模型为2-gram模型。也可以假设一个词由前面N-1个词决定，即N元模型。当N=1时，每个词出现的概率与其他词无关，为一元模型，S对应的概率变为：

$$p(S) = p(\omega_1)p(\omega_2) \cdots p(\omega_n)$$

一般N不超过3。

Methodology

Zipf's Law验证

根据上文用f表示频次，r表示等级序号，以ln (f)为纵坐标、ln (r)为横坐标的图像为一条直线的现象，设计出基于给定的16篇金庸武侠小说语料库，对Zipf's Law进行验证的方法：针对语料库中的文本，首先对其进行预处理，然后使用jieba分词和去除停用词，统计词频并排序，最后绘制以ln (f)为纵坐标、ln (r)为横坐标的图像，观察是否为一条直线。如果图像为一条直线，说明验证Zipf's Law成立；反之，说明Zipf's Law不成立。

中文平均信息熵计算

根据上文信息熵定义与N-gram语言模型公式，设计出基于给定的16篇金庸武侠小说语料库，计算中文平均信息熵的方法：针对语料库中的文本，首先对其进行预处理，然后分别以词和字为单位，利用一元、二元以及三元模型公式计算中文的平均信息熵。

一元模型的信息熵计算公式为：

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

二元模型的信息熵计算公式为：

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

三元模型的信息熵计算公式为：

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

Experimental Studies

Zipf's Law验证

Zipf's Law验证的算法流程如下：

①数据预处理：给定的16篇金庸武侠小说语料库中包含无用的广告信息以及符号，其中部分符号在给定的中文停词列表中已经包括在内，例如标点符号、阿拉伯数字等符号，可在去除停用词时进行去除；但广告信息、英文字母、部分空白符号等符号在给定的中文停词列表中并没有包括，因此需要在数据预处理时进行去除；

```
with open(novel_file_name,'r',encoding='ANSI') as f:
        novel_text = f.read()
        novel_text = novel_text.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', '')
        novel_text = novel_text.replace('新语丝电子文库(www.xys.org)','')
        novel_text = novel_text.replace('新语丝电子文库','')
        novel_text = novel_text.replace('Last Updated: Saturday, November 16, 1996','')
        novel_text = novel_text.replace(u'\u3000',u'').replace('\n','').replace('\r','').replace(" ","")
        novel_text = novel_text.replace('[','').replace(']','')
        novels.append(novel_text)
        f.close()
```

②使用jieba算法库进行分词并去除停用词；

```
words = []

for novel in novels:
    seg_list = jieba.cut(novel)
    filtered_words = [word for word in seg_list if word not in stopwords]
    words.extend(filtered_words)
```

③进行词频统计；

```
# 统计词频并排序
word_freq = {}
for word in words:
    if word in word_freq:
        word_freq[word] += 1
    else:
        word_freq[word] = 1

sorted_word_freq = sorted(word_freq.items(), key=lambda x: x[1], reverse=True)
```

④绘图验证Zipf's Law。

```
# 绘制词频与词序排名的关系图
word_ranks = list(range(1, len(sorted_word_freq) + 1))
word_freqs = [freq for _, freq in sorted_word_freq]
plt.figure(2)
plt.plot(word_ranks, word_freqs)
plt.xlabel('Word Rank')
plt.ylabel('Word Frequency')
```

```
8.  plt.xscale('log')
9.  plt.yscale('log')
10. plt.title('Zipf\'s Law')
11. plt.show()
```

绘制不同小说词频与词序排名的关系图如图1所示，绘制总体词频与词序排名的关系图如图2所示。

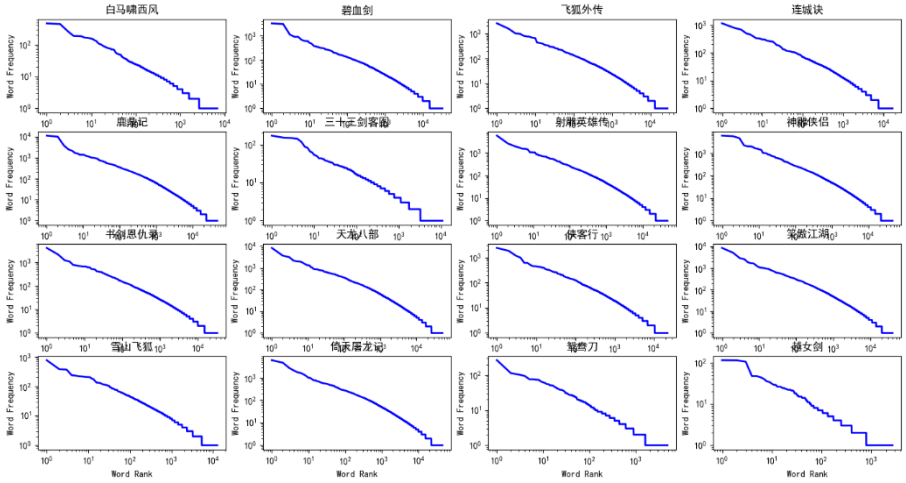

Figure 1 不同小说的验证效果图

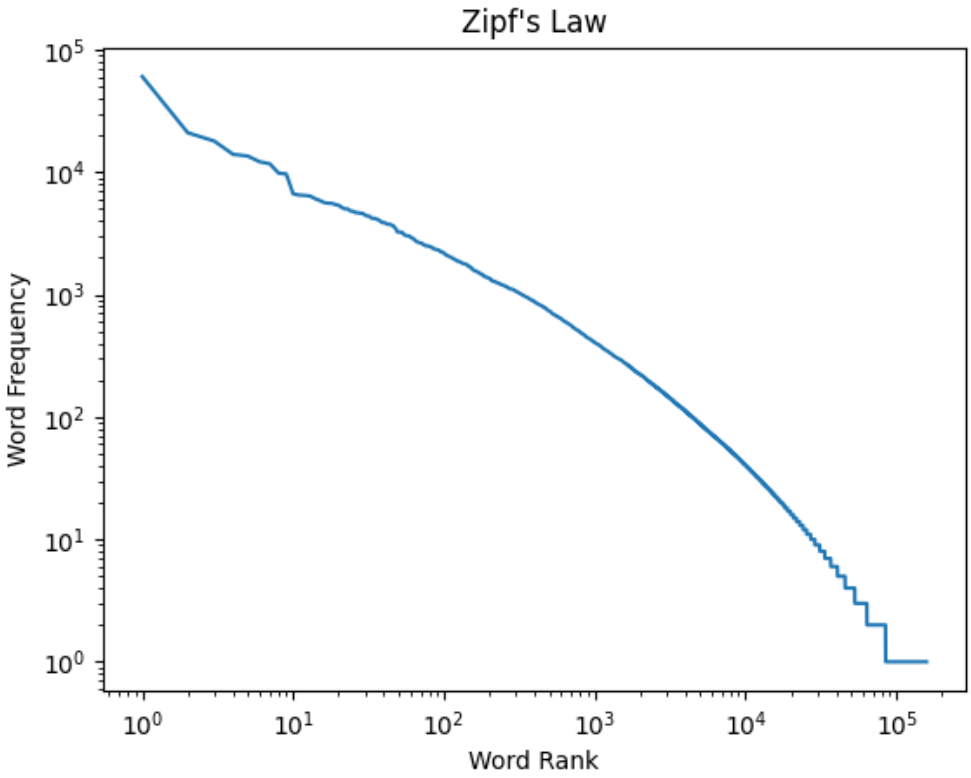

Figure 2 总体验证效果图

中文平均信息熵计算

中文平均信息熵计算的算法流程如下：

①数据预处理：给定的16篇金庸武侠小说语料库中包含无用的广告信息以及符号，其中部分符号在给定的中文停词列表中已经包括在内，例如标点符号、阿拉伯数字等符号，可在去除停用词时进行去除；但广告信息、英文字母、部分空白符号等符号在给定的中文停词列表中并没有包括，因此需要在数据预处理时进行去除；

```
with open(target, "r", encoding='gbk', errors='ignore') as f:
            self.data = f.read()
            self.data = self.data.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', '')
            self.data = self.data.replace('新语丝电子文库(www.xys.org)','')
            self.data = self.data.replace('新语丝电子文库','')
            self.data = self.data.replace('Last Updated: Saturday, November 16, 1996','')
            self.data = self.data.replace(u'\u3000',u'').replace('\n','').replace('\r','').replace(" ","")
            self.data = self.data.replace('[','').replace(']','')
            f.close()
```

②分别针对每本小说及总体使用jieba算法库进行分词并去除停用词

```
# 分词
        for words in jieba.cut(self.data):
            if (words not in self.stop_word) and (not words.isspace()):
                self.split_word.append(words)
                self.split_word_len += 1
```

③分别针对每本小说及总体计算一元模型、二元模型、三元模型的词频表

一元模型词频表：

```
def get_unigram_tf(self, word):
        # 得到单个词的词频表
        unigram_tf = {}
        for w in word:
            unigram_tf[w] = unigram_tf.get(w, 0) + 1
        return unigram_tf
```

二元模型词频表：

```
def get_bigram_tf(self, word):
        # 得到二元词的词频表
        bigram_tf = {}
        for i in range(len(word) - 1):
            bigram_tf[(word[i], word[i + 1])] = bigram_tf.get(
                (word[i], word[i + 1]), 0) + 1
        return bigram_tf
```

三元模型词频表：

```
def get_trigram_tf(self, word):
        # 得到三元词的词频表
        trigram_tf = {}
        for i in range(len(word) - 2):
            trigram_tf[(word[i], word[i + 1], word[i + 2])] = trigram_tf.get(
                (word[i], word[i + 1], word[i + 2]), 0) + 1
```

```
        return trigram_tf
```

④分别针对每本小说及总体计算一元模型、二元模型、三元模型的信息熵

一元模型信息熵：

```
def calc_entropy_unigram(self, word):
        # 计算一元模型的信息熵
        word_tf = self.get_unigram_tf(word)
        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()])
        return entropy
```

二元模型信息熵：

```
def calc_entropy_bigram(self, word):
        # 计算二元模型的信息熵
        # 计算二元模型总词频
        word_tf = self.get_bigram_tf(word)
        last_word_tf = self.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)
        return entropy
```

三元模型信息熵：

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

计算信息熵结果如表1所示：

Table 1：信息熵计算结果

	一元字	二元字	三元字	一元词	二元词	三元词
白马啸西风	9.2209	4.0907	1.2126	11.1235	2.9143	0.3707
碧血剑	9.7546	5.6755	1.7956	12.8847	3.9623	0.4308
飞狐外传	9.6257	5.5736	1.8673	12.6241	4.0442	0.4613

连城诀	9.5138	5.0915	1.6401	12.2058	3.5907	0.3687
鹿鼎记	9.6583	6.0200	2.4097	12.6320	4.9929	0.8381
三十三剑客图	10.0067	4.2845	0.6507	12.5345	1.8089	0.0909
射雕英雄传	9.7370	5.9746	2.2019	13.0246	4.6030	0.5391
神雕侠侣	9.6560	6.0089	2.2855	12.7393	4.6858	0.6367
书剑恩仇录	9.7402	5.6093	1.8677	12.7131	4.1501	0.4991
天龙八部	9.7806	6.1155	2.3522	13.0171	4.8399	0.6636
侠客行	9.4349	5.3800	1.8197	12.2874	3.9926	0.5125
笑傲江湖	9.5117	5.8607	2.3638	12.5218	4.8425	0.7961
雪山飞狐	9.5008	4.8016	1.3033	12.0564	3.0658	0.2907
倚天屠龙记	9.7026	5.9874	2.2784	12.8894	4.6881	0.6445
鸳鸯刀	9.2108	3.6565	0.8961	11.0977	2.1562	0.2432
越女剑	8.7811	3.1099	0.8423	10.4661	1.7387	0.2438
总体	9.9460	7.0259	3.4974	13.5775	6.5212	1.1761

Conclusions

Zipf's Law验证

根据各小说及总体绘制的以ln (f)为纵坐标、ln (r)为横坐标的图像可见，频率与词频排名的对数存在近似反比的关系，曲线近似为一条直线；且随着语料库规模的增大，曲线更加趋近于一条直线。上述结果验证了Zipf's Law的成立。

中文平均信息熵计算

由信息熵计算结果可知，使用一元模型、二元模型、三元模型分别对各小说及总体计算信息熵时，以词和字为单位的结果均有相似规律。随着N元模型的N值增加，信息熵逐渐减小。这是因为用于计算信息熵的词组的词增加，冗余度减小，使用的特定场景越小，出现在文章中的不确定性越小，因此信息熵减小。

以词为单位采用N-gram模型计算的信息熵在N为1的时候值较高，而N为2、3时值较低，并且低于以字为单位的N为2、3时信息熵。这是因为在使用1-gram模型时考虑其前后上下文较少，无意义词重复率高，打乱了文章的秩序，从而使得信息熵增大；而在使用2、3-gram模型的时候基本考虑了语法结构，更好地考虑了上下文，因此得到的熵值较低。

References

[1] 曹聪孙.齐夫定律和语言的“熵”[J].天津师大学报,1987,(04):80-85+73.

[2] 游荣彦.Zipf定律与汉字字频分布[J].中文信息学报,2000,(03):60-65.

[3] 吴萍萍.基于信息熵加权的Word2vec中文文本分类研究[J].长春师范大学学报,2020,39(02):28-33.

[4] Brown P F, Della Pietra S A, Della Pietra V J, et al. An estimate of an upper bound for the entropy of English. Computational Linguistics, 1992, 18(1): 31-40.