【NLP作业-01】计算中文的平均信息熵

杨思捷 ZY2103533

1. 引言

本文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。并在给定的数据库（金庸小说）中计算三种语言模型的信息熵。

信息在信息论中的定义是指用符号传送的报道，而报道的内容是接收符号者预先不知道的。因而信息的存在是消除了不确定性。关于信息的量化度量的定量描述问题，由香农提出的信息熵概念所解决。

2. 信息熵

考虑上抛一枚质量硬币落下时朝上是正面或者反面这样一个事件，我们只能通过发送电位信号传递这个事件，显然使用一个电位信号即可描述这个事件，即1表示正面朝上，0表示反面朝上。而上抛两枚质量均匀的硬币有四种结果，传递这个结果需要两个电位信号，三枚硬币有八种结果，传递这个事件需要三个电位信号。

从上述的例子中引出信息熵H的定义，若信源有n种取值：$X \in \{X_1, X_2, \dots X_n\}$，对应概率为：$p \in \{p_1, p_2, \dots p_n\}$，且各种信源出现的取值彼此独立，则信息熵的定义如下：

$$H(X) = -E[\log p] = -\sum p_i \log p_i$$

其中信息熵的单位为比特，对数一般取2为底。

考虑下面两个信号源。信号源S_1有四种取值，其对应的概率分别为0.7,0.1,0.1,0.1，且四种取值之间相互独立。信号源S_2有四种取值，其对应的概率分别为0.97,0.01,0.01,0.01且四种取值之间相互独立。代入上式，得到两个信号源的信息熵$H(S_1) = 1.357$ bit和$H(S_2) = 0.2419$ bit。则可知信号源1的不确定性比信号源2大。信息熵描述了信息的不确定度，信息熵越大，不确定性越大。

信息熵给出了信息的不确定度，同时该不确定度也代表着数据压缩的理论极限。即无损传递信号源S的信息时至少需要$H(S)$ bit进行描述，任何小于$H(S)$ bit的压缩算法都会对信息造成损失。换句话说，我们可以通过某种编码方式对数据进行压缩从而降低传输和储存的成本。

3. 统计语言模型

假定S表示某一个有意义的句子，n为S的长度，由一连串特定顺序排列的词$w_1, w_2, \dots, w_l$组成，l为语料库中存在的单词个数。现在想知道句子S在文本中出现的概率$P(S)$时，需要通过某个模型进行计算。不妨把$P(S)$展开为$P(S) = P(w_1, w_2, \dots, w_n)$，根据条件概率有

$$P(w_1, w_2, \dots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\dots P(w_n|w_1, w_2, \dots, wn-1)$$

其中$P(w_1)$表示第一个词w_1出现的概率；$P(w_2|w_1)$是在已知第一个词是w_1的情况下，第二个词是w_2的概率，以此类推。显然当句子过长时，$P(w_n|w_1, w_2, \dots, wn-1)$难以计算。

利用马尔科夫假设，任意一个词w_i出现的概率只同它前面的词$w_{i-N}, \dots, w_{i-1}$有关。

则有$N = 1$时，统计模型称为一元模型，每个词出现的概率都是独立的，则有

$$P(w_1, w_2, \dots, w_n) = P(w_1)P(w_2)P(w_3)\dots P(w_n)$$

当$N = 2$时，统计模型称为二元模型，每个词出现的概率与前面两个词相关，则有

$$P(S) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\dots P(w_i|w_{i-2}, w_{i-1})\dots P(w_n|w_{n-2}, w_{n-1})$$

4. 程序实现

给定的数据集为金庸先生的16本小说，对其分别以字和词为单位进行信息熵计算。以字为单位的处理流程为，先预处理去除非打印字符和非中文字符以及标点符号，以句号和逗号为分割，保留句子结构的情况下分别利用一元模型、二元模型处理语料库计算得到信息熵。以词为单位的处理流程为，使用分词库获得分词的统计频率，并利用一元模型、二元模型处理语料库计算得到信息熵。

预处理部分的代码如下，其目的是去除特殊字符和无法打印的字符

```
re_preprocess = re.compile('[a-zA-Z0-9’'#$%&\'()*+,-./:：;<=>?@?★、…【】《》？
“”‘’！[\\]^_`{|}~]+')

# 预处理函数，将原始文本处理为断句断好的列表
def getCorpus(text_raw):
    text_raw = re_preprocess.sub("",text_raw)
    punctuationL =["\t","\n","\u3000","\u0020","\u00A0"," "]
    for i in punctuationL:
        text_raw = text_raw.replace(i,"")
    text_raw = text_raw.replace("，",",。")
    corpus = text_raw.split("。")
    return corpus
```

预处理完成后需要进行词频统计，这部分的代码如下

```
# 获得字的统计字典
def getCharacterFrequency(corpus,n=1):
    cf_dict = {}
    for line in corpus:
        for i in range(len(line)+1-n):
            key = "".join(line[i:i+n])
            cf_dict[key] = cf_dict.get(key,0) + 1
    return cf_dict

# 获得词的统计字典
def getWordFrequency(corpus,n=1):
    cf_dict = {}
    for line in corpus:
        words = list(jieba.cut(line))
        for i in range(len(words)+1-n):
            key = tuple(words[i:i+n])
            cf_dict[key] = cf_dict.get(key,0) + 1
    return cf_dict
```

计算信息熵部分的代码如下

```
# 计算单词信息熵
def calChineseWordEntropy(corpus,n=1):
    entropy = -1
    if n > 1:
        cf_dict_n = getWordFrequency(corpus,n)
        cf_dict_n1 = getWordFrequency(corpus,n-1)
        all_sum = np.sum(list(cf_dict_n.values()))
        entropy = -np.sum([v*np.log2(v/cf_dict_n1[k[:n-1]]) for k,v in
cf_dict_n.items()])/all_sum
    else:
        cf_dict = getWordFrequency(corpus)
        all_sum = np.sum(list(cf_dict.values()))
        entropy = -np.sum([i*np.log2(i/all_sum) for i in
cf_dict.values()])/all_sum
    return entropy
```

其中需要注意的地方是取以2为底的对数需要使用`np.log2`，`np.log`函数是对数函数。

而联合概率部分是统计`n-1`的词频来计算条件概率，代码对应这一行。

```
entropy = -np.sum([v*np.log2(v/cf_dict_n1[k[:n-1]]) for k,v in
cf_dict_n.items()])/all_sum
```

5. 计算结果

根据字进行分割的计算结果如下：

小说名称	小说字数	一元模型（比特/字）	二元模型（比特/字）	三元模型（比特/字）
三十三剑客图.txt	53489	9.676	4.7893	1.0441
书剑恩仇录.txt	435862	9.4723	5.6225	2.431
侠客行.txt	309825	9.1539	5.4799	2.3933
倚天屠龙记.txt	818444	9.3954	5.8735	2.8108
天龙八部.txt	1021452	9.4068	5.9765	2.945
射雕英雄传.txt	772792	9.4521	5.9067	2.7911
白马啸西风.txt	59338	8.8738	4.447	1.6622
碧血剑.txt	416627	9.4599	5.7381	2.3994
神雕侠侣.txt	827768	9.3512	5.8795	2.8581
笑傲江湖.txt	824779	9.2083	5.7467	2.8651
越女剑.txt	13695	8.8313	3.5989	0.9573
连城诀.txt	194534	9.1738	5.2946	2.1924
雪山飞狐.txt	117213	9.1722	5.0381	1.8225
飞狐外传.txt	375501	9.3131	5.6312	2.4538
鸳鸯刀.txt	30490	8.9825	4.1464	1.1987
鹿鼎记.txt	1024303	9.2954	5.8235	2.9397

根据词进行分割的计算结果如下：

小说名称	小说词数	一元模型（比特/词）	二元模型（比特/词）	三元模型（比特/词）
三十三剑客图.txt	31588	11.668	3.2173	0.3719
书剑恩仇录.txt	254181	11.7032	5.2904	1.2962
侠客行.txt	183706	11.1757	5.192	1.3481
倚天屠龙记.txt	476375	11.743	5.7522	1.6035
天龙八部.txt	605732	11.7187	5.8898	1.764
射雕英雄传.txt	458298	11.7886	5.7006	1.5749
白马啸西风.txt	37146	10.063	4.1669	1.054
碧血剑.txt	243605	11.7371	5.2732	1.2201
神雕侠侣.txt	496421	11.5653	5.734	1.782
笑傲江湖.txt	483510	11.3861	5.8202	1.807
越女剑.txt	8094	10.0791	2.7441	0.4268
连城诀.txt	117600	11.0302	4.9313	1.1303
雪山飞狐.txt	71269	10.9072	4.4152	1.0716
飞狐外传.txt	221806	11.5139	5.2609	1.3044
鸳鸯刀.txt	18423	10.2486	3.3928	0.7514
鹿鼎记.txt	606484	11.4349	5.952	1.8992

6. 结论

通过上述的分析计算可以看出

1. 对于字和词的信息熵比较可以看出，
2. 在一元词组模型中，可以明显的发现，信息熵的大小与文字总字数用一定的关系，文字字数多时，信息熵比较大，**计算全部文字总平均信息熵约为11.23**，也侧面可以说明熵本身是一种对于混乱程度的描述。
3. 在多元模型中，平均信息熵时对多元词组条件信息熵的描述，直观的讲是，在确定上一个词或者上几个词之后，对于最后一个词信息熵的描述。除了与统计的总字数有关之外，可以发现一个特例：《三十三剑客图》。虽然《三十三剑客图》与《白马啸西风》字数相仿，比《越女剑》、《鸳鸯刀》字数多，但是多元模型中平均信息熵却更少。其原因是《三十三剑客图》文章中讲的是多个故事，每个故事长度较短，关联性不高，故导致多元模型平均信息熵小。
4. 结果表明金庸小说文章的一元词组模型中文平均信息熵约为11bit/词，香农最早计算出单个英文单词的信息熵约为**2.62**，可以从直观上说明**中文携带的信息量更大**。但是中英文在做词组分割时标准也很难统一，对于中文来说，词组的分割可能更加复杂。
5. 一元模型平均信息熵与多元模型反应的信息关联不大，但多元模型之间的趋势基本一致。对于大于三元的模型计算量比较大，而反映的信息又与三元模型相仿，所以一般计算到三元模型信息熵就足够反映一篇文章的信息熵了

7. 参考资料

https://blog.csdn.net/qq_37098526/article/details/88633403

https://blog.csdn.net/weixin_42663984/article/details/115718241