

学 期 2021-2022 (2)



深度学习与自然语言处理第一次大作业

中文信息熵的计算

院（系）名称	自动化科学与电气工程学院
--------	--------------

专业名称	模式识别
------	------

学生姓名	殷健凯
------	-----

学号	SY2103130
----	-----------

指导老师	秦曾昌
------	-----

2022 年 4 月

1 问题描述

在参考文章的基础上，对给定 16 篇语料库，分为字、词，一元、二元以及三元模型进行中文信息熵的计算。

2 信息熵

1948 年，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：

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

信息熵的性质有：

- 单调性，发生概率越高的事件，其携带的信息量越低；
- 非负性，信息熵可以看作为一种广度量，非负性是一种合理的必然；
- 累加性，即多随机事件同时发生存在的总不确定性的量度是可以表示为各事件不确定性的量度的和，这也是广度量的一种体现

3 统计语言模型

假定 S 表示某个有意义的句子，由一连串特定顺序排列的词 $\omega_1, \omega_2, \omega_3, \dots, \omega_n$ 组成这里 n 是句子长度。现在我们想知道 S 在文本中出现的可能性，即：

$$p(s) = p(\omega_1, \omega_2, \omega_3, \dots, \omega_n)$$

利用条件概率公式：

$$p(\omega_1, \omega_2, \omega_3, \dots, \omega_n) = p(\omega_1)p(\omega_2|\omega_1)\dots p(\omega_n|\omega_1, \omega_2, \dots, \omega_{n-1})$$

当计算 $p(\omega_1)$ ，仅存在一个参数；计算 $p(\omega_2|\omega_1)$ ，存在两个参数，以此类推，难易计算，所以马尔可夫提出一种假设：假设出现的概率只与前面 $N-1$ 个词相关，当 $N=2$ 时，就是二元模型， $N=3$ 就是三元模型。 N 元模型的数学表达如下所示：

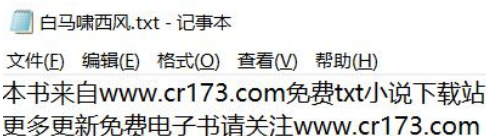
$$P(\omega_i|\omega_{i-n+1}^{i-1}) = P(\omega_i|\omega_{i-n+1}\dots\omega_{i-1})$$

即当前这个词 ω_i 依赖于前面 $N-1$ 个词，上述 $N=1$ 为与前面单词都没有关系； $N=2$ 表示与前面一个单词有关； $N=3$ 表示与前面两个词有关。选取不同模型最终结果也会不同。

4 中文熵计算实验

4.1 数据预处理

(1) 去除广告：



白马啸西风.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
本书来自www.cr173.com免费txt小说下载站
更多更新免费电子书请关注www.cr173.com

(2) 将换行符，Tab 等去除：

```
replacements = ["\t", "\n", "\u3000", "\u0020", "\u00A0", " "]
```

```
for replacement in replacements:
```

```
    file = file.replace(replacement, "")
```

(3) 将逗号替换为句号，并以句号分割每一句话：

```
file = file.replace(",", ".")
```

```
file = file.split(".")
```

(4) 对每句话进行取中文操作：

```
pattern = re.compile(r'^\u4e00-\u9fa5$')
```

```
chinese = re.sub(pattern, "", file)
```

预处理后的结果如下所示（以白马啸西风.txt 为例，节选部分结果）：

[白马啸西风得得得,'得得得得得得得','得得得在黄沙莽莽的回疆大漠之上','尘沙飞起两丈来高','两骑马一前一後的急驰而来','前面是匹高腿长身的白马','马上骑著个少妇','怀中搂著个七八岁的小姑娘','後面是匹枣红马','马背上伏著的是个高瘦的汉子','那汉子左边背心上却插著一枝长箭','鲜血从他背心流到马背上','又流到地下','滴入了黄沙之中','他不敢伸手拔箭','只怕这枝箭一拔下来','就会支持不住']

4.2 统计字频和词频

对每个字和每个词出现的次数进行统计,为后续计算熵做准备,其中统计词的次数使用到了 jieba 包中的分词功能。

```
def cha_fre(file,n):
    """
    统计字频
    :param file:给定文章
    :param n: n 元
    :return:
    """
    adict = {}
    if n == 1:
        for line in file:
            for i in line:
                if i in adict:adict[i] += 1
                else:adict[i] = 1
    elif n == 2:
        for line in file:
            for i in range(len(line)-1):
                if (line[i]+line[i+1]) in adict:adict[line[i]+line[i+1]] += 1
                else:adict[line[i]+line[i+1]] = 1
    else:
        for line in file:
            for i in range(len(line)-2):
                if (line[i]+line[i+1]+line[i+2]) in adict:adict[line[i]+line[i+1]+line[i+2]] += 1
                else:adict[line[i]+line[i+1]+line[i+2]] = 1
    return adict

def word_fre(file,n):
    """
    统计词频
    :param file:给定文章
    :param n: n 元
    :return:
    """
    adict = {}
    if n == 1:
        for line in file:
```

```

words = list(jieba.cut(line))
for i in range(len(words)):
    if tuple(words[i:i+1]) in adict:adict[tuple(words[i:i+1])] += 1
    else:adict[tuple(words[i:i+1])] = 1
elif n == 2:
    for line in file:
        words = list(jieba.cut(line))
        for i in range(len(words)-1):
            if tuple(words[i:i+2]) in adict:adict[tuple(words[i:i+2])] += 1
            else:adict[tuple(words[i:i+2])] = 1
else:
    for line in file:
        words = list(jieba.cut(line))
        for i in range(len(words)-2):
            if tuple(words[i:i+3]) in adict:adict[tuple(words[i:i+3])] += 1
            else:adict[tuple(words[i:i+3])] = 1
return adict

```

统计**字**频的结果如下所示（白马啸西风.txt，节选）：

一元：{'白': 128, '马': 231, '啸': 4, '西': 54, '风': 72, '得': 369, '在': 582, '黄': 31, '沙': 78, '莽': 6, '的': 1561, '回': 139, '疆': 27, '大': 397, '漠': 54, '之': 290, '上': 429, '尘': 4, '飞': 31, '起': 225, '两': 188, '丈': 24, '来': 571, '高': 79, '骑': 24, '一': 1315, '前': 100, '後': 142}

二元：{'白马': 76, '马啸': 1, '啸西': 1, '西风': 2, '风得': 1, '得得': 10, '得在': 2, '在黄': 1, '黄沙': 12, '沙莽': 2, '莽莽': 3, '莽的': 1, '的回': 3, '回疆': 27, '疆大': 1, '大漠': 17, '漠之': 9, '之上': 11, '尘沙': 2, '沙飞': 4, '飞起': 3, '起两': 1, '两丈': 1, '丈来': 1, '来高': 1, '两骑': 1, '骑马': 6}

三元：{'白马啸': 1, '马啸西': 1, '啸西风': 1, '西风得': 1, '风得得': 1, '得得得': 6, '得得在': 1, '得在黄': 1, '在黄沙': 1, '黄沙莽': 2, '沙莽莽': 2, '莽莽的': 1, '莽的回': 1, '的回疆': 1, '回疆大': 1, '疆大漠': 1, '大漠之': 6, '漠之上': 3, '尘沙飞': 2, '沙飞起': 1}

统计**词**频的结果如下所示（白马啸西风.txt 节选）：

一元：{('白马'), 75, ('啸'), 1, ('西风'), 2, ('得'), 168, ('在'), 494, ('黄沙'), 12, ('莽莽'), 3, ('的'), 1484, ('回疆'), 22, ('大漠'), 17, ('之上'), 11, ('尘沙'), 2, ('飞'), 9, ('起'), 18, ('两丈'), 1, ('来'), 145, ('高'), 9, ('两'), 7, ('骑马'), 5, ('一前'), 1, ('一'), 96, ('後'), 102, ('急驰'), 1}

二元：{('白马', '啸'), 1, ('啸', '西风'), 1, ('西风', '得'), 1, ('得', '得'), 10, ('得', '在'), 1, ('在', '黄沙'), 1, ('黄沙', '莽莽'), 2, ('莽莽', '的'), 1, ('的', '回疆'), 1, ('回疆', '大漠'), 1, ('大漠', '之上'), 3, ('尘沙', '飞'), 1, ('飞', '起'), 1, ('起', '两丈'), 1, ('两丈', '来'), 1, ('来', '高'), 1}

三元：{('白马', '啸', '西风'), 1, ('啸', '西风', '得'), 1, ('西风', '得', '得'), 1, ('得', '得', '得'), 6, ('得', '得', '在'), 1, ('得', '在', '黄沙'), 1, ('在', '黄沙', '莽莽'), 1, ('黄沙', '莽莽', '的'), 1, ('莽莽', '的', '回疆'), 1, ('的', '回疆', '大漠'), 1, ('回疆', '大漠', '之上'), 1, ('尘沙', '飞', '起'), 1}

可以看出，相对于字频来说，无论是几元模型，词频都更贴近真实意思。

4.3 计算信息熵

基于字和词的多元模型计算公式是一致的，如下所示：

一元： $H(x) = -\sum p(x) \log p(x)$

二元： $H(x) = -\sum p(x, y) \log p(x|y)$

三元： $H(x) = -\sum p(x, y, z) \log p(x|y, z)$

```
def cal_cha_entropy(file,n):
    if n == 1:
        frequency = cha_fre(file,1)
        sums = np.sum(list(frequency.values()))
        entropy = -np.sum([i*np.log2(i/sums) for i in frequency.values()])/sums
    elif n == 2:
        frequency1 = cha_fre(file,1)
        frequency2 = cha_fre(file,2)
        sums = np.sum(list(frequency2.values()))
        entropy=-np.sum([v*np.log2(v/frequency1[k[:n-1]])for k,v in frequency2.items()])/sums
    else:
        frequency2 = cha_fre(file,2)
        frequency3 = cha_fre(file,3)
        sums = np.sum(list(frequency3.values()))
        entropy=-np.sum([v*np.log2(v/frequency2[k[:n-1]])for k,v in frequency3.items()])/sums
    return entropy

def cal_word_entropy(file,n):
    if n == 1:
        frequency = word_fre(file,1)
        sums = np.sum(list(frequency.values()))
        entropy = -np.sum([i*np.log2(i/sums) for i in frequency.values()])/sums
    elif n == 2:
        frequency1 = word_fre(file,1)
        frequency2 = word_fre(file,2)
        sums = np.sum(list(frequency2.values()))
        entropy=-np.sum([v*np.log2(v/frequency1[k[:n-1]])for k,v in frequency2.items()])/sums
    else:
        frequency2 = word_fre(file,2)
        frequency3 = word_fre(file,3)
        sums = np.sum(list(frequency3.values()))
        entropy=-np.sum([v*np.log2(v/frequency2[k[:n-1]])for k,v in frequency3.items()])/sums
    return entropy
```

分别计算 16 个文本的信息熵，结果如下（单位为比特/字）：

	字 一元	字 二元	字 三元	词 一元	词 二元	词 三元
白马啸西风	8.911	4.495	1.656	10.226	4.193	0.878
碧血剑	9.446	5.742	2.405	11.725	5.281	1.225
飞狐外传	9.308	5.632	2.456	11.512	5.262	1.305
连城诀	9.171	5.296	2.194	11.027	4.934	1.131
鹿鼎记	9.281	5.831	2.947	11.432	5.971	1.899
三十三剑客图	9.668	4.790	1.047	11.668	3.215	0.372
射雕英雄传	9.439	5.915	2.802	11.809	5.709	1.551
神雕侠侣	9.373	5.921	2.881	11.711	5.777	1.654
书剑恩仇录	9.466	5.626	2.433	11.496	5.297	1.299
天龙八部	9.404	5.977	2.946	11.716	5.891	1.765
侠客行	9.152	5.480	2.394	11.174	5.193	1.349
笑傲江湖	9.206	5.747	2.866	11.384	5.822	1.808
雪山飞狐	9.201	5.078	1.813	11.099	4.401	0.869
倚天屠龙记	9.393	5.874	2.812	11.741	5.754	1.605
鸳鸯刀	9.033	4.175	1.169	10.477	3.329	0.545
越女剑	8.824	3.601	0.961	10.072	2.751	0.428

5 对比实验

为了验证“信息熵越大携带的信息越有用”这句话，整理了“哈工大停用词词库”、“四川大学机器学习智能实验室停用词库”、“百度停用词表”，共 1598 个停用词，部分如下所示：

啊
阿
哎
哎呀
哎哟
唉
俺
俺们
按
按照
吧
吧哒
把
罢了
被
本

调用程序，将 16 个文本中都去除了停词，降低了每句话携带的“无用信息”。理论上也提高了每句话的信息熵。为了验证这个想法，进行了实验，实验结果如下：

	字 一元	字 二元	字 三元	词 一元	词 二元	词 三元
白马啸西风	9.260	3.878	1.397	11.178	2.901	0.537
碧血剑	9.776	5.374	2.025	12.990	4.011	0.596
飞狐外传	9.618	5.270	2.112	12.752	4.043	0.605
连城诀	9.558	4.839	1.858	12.343	3.533	0.449
鹿鼎记	9.633	5.515	2.524	12.742	5.010	1.100
三十三剑客图	10.003	4.269	0.840	12.596	1.791	0.146
射雕英雄传	9.769	5.627	2.424	13.183	4.565	0.674
神雕侠侣	9.677	5.642	2.532	12.986	4.739	0.839
书剑恩仇录	9.776	5.266	2.097	12.852	4.201	0.686
天龙八部	9.791	5.561	2.430	13.204	4.733	0.816
侠客行	9.472	5.074	2.027	12.404	3.982	0.667
笑傲江湖	9.206	5.747	2.866	11.384	5.822	1.808
雪山飞狐	9.500	4.587	1.546	12.117	3.140	0.419
倚天屠龙记	9.709	5.561	2.430	13.038	4.651	0.820
鸳鸯刀	9.238	3.653	1.039	11.129	2.240	0.347
越女剑	8.913	3.085	0.981	10.429	1.846	0.414

6 结论

通过对比一元字和词的信息熵可以发现，所有词的都大于字的，说明词携带的信息要高于字。

通过对比实验发现，将停词删除后，所有一元模型信息熵均有所提升，说明信息熵的模型确实可以衡量一句话、一本书的信息量大小。