NLP-1 计算中文平均信息熵

罗志钊 PT2100565

问题介绍

首先阅读文章：Entropy_of_English_PeterBrown 链接： https://share.weiyun.com/5zGPyJX

作业内容： 参考上面的文章来计算中文(分别以词和字为单位)的平均信息熵。

基础知识

信息熵

信息熵用于表示信息的不确定性，熵值越大，则信息的不确定程度越大。数学公式为：

$$H(x) = \sum_{x \in X} P(x) log(\frac{1}{P(x)}) = -\sum_{x \in X} P(x) log(P(x))$$

并规定：0 log（0） = 0。

联合信息熵可以用于估计二元模型和三元模型，数学公式为：

$$\begin{aligned} H(X|Y) &= -\sum_{y \in Y} P(y) log(P(x|y)) \\ &= -\sum_{y \in Y} P(y) \sum_{x \in X} P(x) log(P(x|y)) \\ &= -\sum_{y \in Y} \sum_{x \in X} P(x) P(y) log(P(x|y)) \\ &= -\sum_{y \in Y} \sum_{x \in X} P(x, y) log(P(x|y)) \end{aligned}$$

jieba

jieba最流行的应用是分词，包括介绍页面上也称之为“结巴中文分词”，但除了分词之外，jieba还可以做关键词抽取、词频统计等。

GitHub链接：https://github.com/fxsjy/jieba

N-Gram模型

给定一个句子序列：

$$S = W_1, W_2, \ldots, W_K$$

对于一元模型，它的概率数学公式可以表示为：

$$P(S) = P(W_1, W_2, \ldots, W_K) = p(W_1)P(W_2|W_1)\ldots P(W_K|W_1, W_2, \ldots, W_{K-1})$$

在这里，一元模型用于得出字的信息熵。

对于二元模型，它的概率数学公式可以表示为：

$$P(S) = P(W_1)P(W_2|W_1)P(W_3|W_2)\ldots P(W_i|W_{i-1})\ldots P(W_n|W_{n-1})$$

二元模型用于得出二字词的信息熵。

对于三元模型，它的概率数学公式可以表示为：

$$P(S) = P(W_1)P(W_2|W_1)P(W_3|W_1, W_2)\ldots P(W_i|W_{i-2}W_{i-1})\ldots P(W_n|W_{n-2}, W_{n-1})$$

三元模型用于得出三字词的信息熵。

本文核心内容即利用这三个模型，对金庸16本小说来分析中文字和词的信息熵。

实验内容

数据预处理

预处理分为两部分

1. 对文件格式的处理
2. 对符号和乱码的处理

针对第一点，由于下载的16本金庸小说存在utf-8和gb18030两种编码格式，针对于本次是对中文信息熵的处理，所以统一转码为gb18030。同时，将16本小说内容进行合并。

对于第二点，由于我们是处理中文且无需考虑句与句之间的关联，所以去除标点符号即可。

```
r1 = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:： ;<=>?@，。?★、…【】《》？“”‘’！ [\\]^_`{|}~]+''''
```

同时，经过笔者对小说内容的查阅，发现存在换行符、乱码和重复条目的内容会影响处理的质量，于是进一步对这些内容进行处理：

```
filecontext = filecontext.replace("\n", '')
filecontext = filecontext.replace(" ", '')
filecontext = filecontext.replace("牋", '')
filecontext = filecontext.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", '')
```

语料的基本信息

总语料库是金庸的十六本小说，利用jieba进行分词。

```
for line in article:
   for x in jieba.cut(line):
            wordsS.append(x)
            wordsL += 1
   wordTf(wordsTf, wordsS)
   wordsS = []
   lineC += 1
```

总字数和分词个数统计如下

序号	小说名	字数	分词个数
1	三十三剑客图	53682	31551
2	书剑恩仇录	438338	255908
3	侠客行	317099	190545
4	倚天屠龙记	830439	487540
5	天龙八部	1040707	623475
6	射雕英雄传	794124	480018
7	白马啸西风	64061	42207
8	碧血剑	420068	246405
9	神雕侠侣	835634	502809
10	笑傲江湖	833372	490987
11	越女剑	14803	9293
12	连城诀	199412	122177
13	雪山飞狐	119513	73383
14	飞狐外传	378777	224513
15	鸳鸯刀	32552	20658
16	鹿鼎记	1047468	629266
17	汇总	7420049	4430735

建立模型

根据一元、二元和三元模型对文段的处理进行获取，笔者获取文段的总字数、分词个数以及对应的运行时间接着建立对应的模型，得到不同模型对应的信息熵。

实验数据如下：

```
1
Corpus word count: 7420049
Word number: 4430735
Time: 15.84973 s
Chinese information entropy of unitary model: 12.01312 bit/word
2
Corpus word count: 7420049
Word number: 4430735
Time: 17.60917 s
Chinese information entropy for binary models: 6.89151 bit/word
3
Corpus word count: 7420049
Word number: 4430735
Time: 22.99487 s
Chinese information entropy of ternary model: 2.4166 bit/word

Process finished with exit code 0
```

具体模型代码见附录。

结论

根据金庸16本小说得出的中文信息熵如下所示：

序号	模型名称	总字数/个	分词个数/个	运行时间/s	信息熵b/w
1	一元模型	7420049	4430735	15.84973	12.01312
2	二元模型	7420049	4430735	17.60917	6.89151
3	三元模型	7420049	4430735	22.99487	2.4166

从中我们得出，**中文单字的信息熵为12.01312；中文二字词的信息熵为6.89151；中文三字词的信息熵为2.4166。**

中文信息熵中的一元模型信息熵最高，三元模型信息熵最低。而英文单词的一元信息熵一般在3.5附近，二元信息熵在3附近。从中可以得出以下结论：

1. 中文的信息熵高于英文的信息熵。
2. 随着一个字前有限个字进行约束，组成词约束越多（即词越长），信息熵越低。因为固定的词能减少字或短词打乱语段的次数，使得语段整体变得有序，减少了不确定性。因而我们可以利用这一点，加入对分析语段的约束来降低语段的信息熵，降低分析成本，得出相对可靠的结论。

附录

1-gram/一元模型

```
def unigramC(article,count):
    before = time.time()
    lineC = 0
    wordsTf = {}
    wordsS = []
    wordsL = 0

    for line in article:
        for x in jieba.cut(line):
            wordsS.append(x)
            wordsL += 1
        wordTf(wordsTf, wordsS)
        wordsS = []
        lineC += 1

    print("1")
    print("Corpus word count:", count)
    print("Word number:", wordsL)
    entropy = []
    for uni_word in wordsTf.items():
        entropy.append(-(uni_word[1] / wordsL) * math.log(uni_word[1] / wordsL, 2))
    print("Chinese information entropy of unitary model:", round(sum(entropy), 5),
"bit/word")
    after = time.time()
    print("Time:", round(after - before, 5), "s")
```

2-gram/二元模型

```
def bigramC(article, count):
    before = time.time()
    wordsS = []
    wordsL = 0
    lineC = 0
    wordsTf = {}
    bigramTf = {}

    for line in article:
        for x in jieba.cut(line):
            wordsS.append(x)
            wordsL += 1

        wordTf(wordsTf, wordsS)
        bigramTfG(bigramTf, wordsS)
```

```
            wordsS = []
            lineC += 1

    print("2")
    print("Corpus word count:", count)
    print("Word number:", wordsL)

    bigramL = sum([dic[1] for dic in bigramTf.items()])
    entropy = []
    for bigramW in bigramTf.items():
        jp_xy = bigramW[1] / bigramL  # 计算联合概率p(x,y)
        cp_xy = bigramW[1] / wordsTf[bigramW[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


    after = time.time()
    print("Time:", round(after - before, 5), "s")
    print("Chinese information entropy for binary models:", round(sum(entropy), 5),
"bit/word")
```

3-gram/三元模型

```
def trigramC(article,count):
    before1 = time.time()
    wordsS = []
    wordsL = 0
    lineC = 0
    wordsTf = {}
    trigramTf = {}

    for line in article:
        for x in jieba.cut(line):
            wordsS.append(x)
            wordsL += 1

        bigramTfG(wordsTf, wordsS)
        trigramTfG(trigramTf, wordsS)

        wordsS = []
        lineC += 1

    print("3")
    print("Corpus word count:", count)
    print("Word number:", wordsL)

    trigramL = sum([dic[1] for dic in trigramTf.items()])

    entropy = []
    for trigramW in trigramTf.items():
```

```
            jp_xy = trigramW[1] / trigramL  # 计算联合概率p(x,y)
            cp_xy = trigramW[1] / wordsTf[trigramW[0][0]]  # 计算条件概率p(x|y)
            entropy.append(-jp_xy * math.log(cp_xy, 2))  # 计算三元模型的信息熵
    after1 = time.time()
    print("Time:", round(after1 - before1 , 5), "s")
    print("Chinese information entropy of ternary model:", round(sum(entropy), 5),
"bit/word")
```

参考文献

1. https://blog.csdn.net/weixin_50891266/article/details/115723958?spm=1001.2101.3001.6661.1&utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc_relevant_default&utm_relevant_index=1
2. https://blog.csdn.net/qq_40412713/article/details/115742092?spm=1001.2101.3001.6650.2&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-2.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-2.pc_relevant_default&utm_relevant_index=5