中文平均信息熵的计算

ZY2103518 吕晔

问题：首先阅读文章：Entropy_of_English_PeterBrown。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。

1 信息熵的概念

1.1 随机变量的自信息

当对一个随机事件进行观察时，这个随机事件给观察者带来了多少信息量，直觉上可以对信息给出如下两个定义：信息量等于传输该信息所用的代价；两个相同信源产生的信息量两倍于单个信源产生的信息量。

香农提出，信息是对不确定性的消除。消除的不确定性越多，获得的信息量就越大。假设有两个事件分别为：1 天气预报于夏天预报下雪。2 天气预报于冬天预报下雪。一般来说，对于第一个事件，人们往往会觉得不可思议，因为夏天下雪是小概率事件，天气预报消除了更多的不确定性，因此事件 1 中包含了较大的信息量；而对于第二个事件，人们则不会感到惊奇，因为冬天下雪为大概率事件，符合常识，事件 2 提供的信息量相对较少。

如果将信息视为消除随机事件不确定性过程中引入的变化的话，那么信息的量就与随机事件的概率紧密相关。设函数 $I(a_1)$ 表示观测到随机事件 a_1 发生所带来的信息量，按照上面的思路进行推导，随机事件的自信息（在文本中可以将自信息等价地视为信息量）应该符合下面的四条性质：

1. 概率越大的随机事件提供的自信息越小，概率越小的随机事件提供的自信息越大，即若 $P(a_1) > P(a_2)$，则 $I(a_1) < I(a_2)$。
2. 概率为 1 的随机事件所能提供的信息量为 0，因为没有任何的不确定性被消除，即若 $P(a_1) = 1$，则 $I(a_1) = 0$。
3. 概率为 0 的随机事件所能提供的信息量为无穷，即若 $P(a_1) = 0$，则 $I(a_1) = \infty$。
4. 当两个独立的随机事件同时出现的时候，他们所提供的自信息的量为这两个事件各自自信息的量的算数和，，即若 a_1, a_2 相互独立，则 $I(a_1, a_2) = I(a_1) + I(a_2)$

根据直觉进行观察和猜测，发现函数 I 可以为对数函数，即 $I(a_i) = \log \frac{1}{P(a_i)}$，它满足以上四条自信息的性质。

1.2 离散随机变量的信息熵

在对随机事件的自信息进行定义后，人们很自然地就希望将这个概念推广到随机系统上。一个随机事件等价于从随机系统的输出中观测到某一个取值的事件；对随机系统所能够提供的总信息量进行统计时，若每一个随机事件的自信息为$I(a_i)$，人们很自然地就能想到使用该系统中所有随机事件自信息的统计平均代表该随机系统的总体信息量，即$E_p I(a_i) = \sum_i p(a_i) I(a_i) = -\sum_i P(a_i) \log p(a_i)$。假设一个随机系统可以用一个离散随机变量 X 进行建模，那么这个随机系统总体信息量可以用信息熵 $H(X)$ 来表示，$H(X) = -\sum_{x \in X} p(x) \log p(x)$。上面对于自信息以及信息熵的定义与形式都是基于观察与猜测得到的。实际上，这样的定义不仅是合理的，而且是唯一的。香农提出如果要对一个随机系统的信息总量（即信息熵）进行度量，那么用于度量的函数表达必须满足以下三个性质：

1. 连续性，即当随机系统的概率分布发生了微小的变化的时候，随机系统的总体信息量不应该发生显著的变化，二者前后应该是连续的。

2. 等概时单调增性，即当随机系统是在集合上等概率分布时，随着集合元素的个数增加，信息熵度量函数应该具有单调增的特性。

3. 可加性，即一个随机系统的信息熵应该具有可加的性质；用语言对可加性进行表述，即若先对随机系统的一部分进行观察，再对随机系统的剩下部分进行观察，这两次观察得到的总体信息量应当与随机系统进行一次彻底的观察得到的信息量相同。

2 实验过程

2.1 数据预处理

首先对文本进行预处理，删除文本中非中文字符、隐藏符号和标点符号。

```
def getCorpus(self, rootDir):
    corpus = []
    r1 = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:; ;<=>?@, 。?★、…【】《》？“”‘’！ [\\]^_`{|}~]+'  # 用户也可以在此进行自定义过滤字符
    listdir = os.listdir(rootDir)
    count=0
    for file in listdir:
        path = os.path.join(rootDir, file)
        if os.path.isfile(path):
            with open(os.path.abspath(path), "r", encoding='ansi') as file:
                filecontext = file.read();
                filecontext = re.sub(r1, '', filecontext)
                filecontext = filecontext.replace("\n", '')
                filecontext = filecontext.replace(" ", '')
                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", '')
                #seg_list = jieba.cut(filecontext, cut_all=True)
                #corpus += seg_list
                count += len(filecontext)
                corpus.append(filecontext)
        elif os.path.isdir(path):
            TraversalFun.AllFiles(self, path)
    return corpus, count
```

2.2 计算平均信息熵

分别应用一元模型、二元模型、三元模型三种模型分别计算小说集中的平均信息熵。得到如下运行结果。

```
2022/03/26  20:20            24,496 Untitled Diagram.drawio
2022/04/13  11:23    <DIR>          新建文件夹
               5 个文件         45,634 字节
               5 个目录 375,623,925,760 可用字节

(ChineseEn) C:\Users\dell\OneDrive\研究生\课程\深度学习与自然语言处理\第一次大作业\NLP_Chinese-entroy-main>python "sever
al models.py"
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\dell\AppData\Local\Temp\jieba.cache
Loading model cost 0.578 seconds.
Prefix dict has been built successfully.
语料库字数: 7420081
分词个数: 4430767
平均词长: 1.67467
基于词的一元模型的中文信息熵为: 12.01312 比特/词
运行时间: 32.07425 s
语料库字数: 7420081
分词个数: 4430767
平均词长: 1.67467
二元模型长度: 4430751
基于词的二元模型的中文信息熵为: 6.8915 比特/词
运行时间: 36.50828 s
语料库字数: 7420081
分词个数: 4430767
平均词长: 1.67467
三元模型长度: 4430735
基于词的三元模型的中文信息熵为: 2.41661 比特/词
运行时间: 47.56856 s

(ChineseEn) C:\Users\dell\OneDrive\研究生\课程\深度学习与自然语言处理\第一次大作业\NLP_Chinese-entroy-main>
```

2.3 实验结果

模型	语料库字数	分词个数	平均词长	中文信息熵	运行时间
一元模型	7420081	4430767	1.67467	12.01312	32.07465
二元模型	7420081	4430767	1.67467	6.8915	36.50828
三元模型	7420081	4430767	1.67467	2.41661	47.56856

3 实验代码

```python
import jieba
import math
import time
import os
import re
class TraversalFun():

    # 1 初始化
    def __init__(self, rootDir):
        self.rootDir = rootDir

    def TraversalDir(self):
        return TraversalFun.getCorpus(self, self.rootDir)

    def getCorpus(self, rootDir):
        corpus = []
```

```
        r1 = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;<=>?@, 。?★、…【】《》？“”‘’！[\\]^_`{|}~]+'  # 用户也可以在此进行自定义过滤字符
        listdir = os.listdir(rootDir)
        count=0
        for file in listdir:
            path  = os.path.join(rootDir, file)
            if os.path.isfile(path):
                with open(os.path.abspath(path), "r", encoding='ansi') as file:
                    filecontext = file.read();
                    filecontext = re.sub(r1, '', filecontext)
                    filecontext = filecontext.replace("\n", '')
                    filecontext = filecontext.replace(" ", '')
                    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", '')
                    #seg_list = jieba.cut(filecontext, cut_all=True)
                    #corpus += seg_list
                    count += len(filecontext)
                    corpus.append(filecontext)
            elif os.path.isdir(path):
                TraversalFun.AllFiles(self, path)
        return corpus,count

# 词频统计，方便计算信息熵
def get_tf(tf_dic, words):

    for i in range(len(words)-1):
        tf_dic[words[i]] = tf_dic.get(words[i], 0) + 1

def get_bigram_tf(tf_dic, words):
    for i in range(len(words)-1):
        tf_dic[(words[i], words[i+1])] = tf_dic.get((words[i], words[i+1]), 0) + 1

def get_trigram_tf(tf_dic, words):
    for i in range(len(words)-2):
        tf_dic[((words[i], words[i+1]), words[i+2])] = tf_dic.get(((words[i], words[i+1]), words[i+2]), 0) + 1

def cal_unigram(corpus,count):
     before = time.time()
    split_words = []
    words_len = 0
    line_count = 0
    words_tf = {}
    for line in corpus:
        for x in jieba.cut(line):
            split_words.append(x)
            words_len += 1
        get_tf(words_tf, split_words)
        split_words = []
        line_count += 1

    print("语料库字数:", count)
    print("分词个数:", words_len)
    print("平均词长:", round(count / words_len, 5))
    entropy = []
    for uni_word in words_tf.items():
```

```
70.         entropy.append(-
    (uni_word[1] / words_len) * math.log(uni_word[1] / words_len, 2))
71.     print("基于词的一元模型的中文信息熵为:", round(sum(entropy), 5), "比特/词")
72.     after = time.time()
73.     print("运行时间:", round(after - before, 5), "s")
74.
75. def cal_bigram(corpus, count):
76.     before = time.time()
77.     split_words = []
78.     words_len = 0
79.     line_count = 0
80.     words_tf = {}
81.     bigram_tf = {}
82.
83.     for line in corpus:
84.         for x in jieba.cut(line):
85.             split_words.append(x)
86.             words_len += 1
87.
88.         get_tf(words_tf, split_words)
89.         get_bigram_tf(bigram_tf, split_words)
90.
91.         split_words = []
92.         line_count += 1
93.
94.     print("语料库字数:", count)
95.     print("分词个数:", words_len)
96.      print("平均词长:", round(count / words_len, 5))
97.
98.     bigram_len = sum([dic[1] for dic in bigram_tf.items()])
99.     print("二元模型长度:", bigram_len)
100.
101.        entropy = []
102.        for bi_word in bigram_tf.items():
103.            jp_xy = bi_word[1] / bigram_len  # 计算联合概率p(x,y)
104.            cp_xy = bi_word[1] / words_tf[bi_word[0][0]]  # 计算条件概率p(x|y)
105.            entropy.append(-jp_xy * math.log(cp_xy, 2))  # 计算二元模型的信息熵
106.        print("基于词的二元模型的中文信息熵为:", round(sum(entropy), 5), "比特/词")
107.
108.        after = time.time()
109.        print("运行时间:", round(after - before, 5), "s")
110.
111.    def cal_trigram(corpus,count):
112.        before = time.time()
113.        split_words = []
114.        words_len = 0
115.        line_count = 0
116.        words_tf = {}
117.        trigram_tf = {}
118.
119.        for line in corpus:
120.            for x in jieba.cut(line):
121.                split_words.append(x)
122.                words_len += 1
123.
124.            get_bigram_tf(words_tf, split_words)
125.            get_trigram_tf(trigram_tf, split_words)
```

```
126.
127.             split_words = []
128.             line_count += 1
129.
130.         print("语料库字数:", count)
131.         print("分词个数:", words_len)
132.         print("平均词长:", round(count / words_len, 5))
133.
134.         trigram_len = sum([dic[1] for dic in trigram_tf.items()])
135.         print("三元模型长度:", trigram_len)
136.
137.         entropy = []
138.         for tri_word in trigram_tf.items():
139.             jp_xy = tri_word[1] / trigram_len  # 计算联合概率p(x,y)
140.             cp_xy = tri_word[1] / words_tf[tri_word[0][0]]  # 计算条件概率p(x|y)
141.             entropy.append(-jp_xy * math.log(cp_xy, 2))  # 计算三元模型的信息熵
142.         print("基于词的三元模型的中文信息熵为:", round(sum(entropy), 5), "比特/词")
143.
144.         after = time.time()
145.         print("运行时间:", round(after - before, 5), "s")
146.
147.
148.     if __name__ == '__main__':
149.         tra = TraversalFun("./datasets/")
150.         corpus,count = tra.TraversalDir()
151.         cal_unigram(corpus, count)
152.         cal_bigram(corpus,count)
153.         cal_trigram(corpus,count)
```

4 参考文献

深度学习与自然语言处理实验——中文信息熵的计算 https://blog.csdn.net/weixin_42663984/article/details/11571 8241