DL-NLP 第一次实验——信息熵的计算

一、问题描述

首先阅读文章：Entropy of English PeterBrown

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。

二、实验原理

1. 信息熵

信息熵的概念最早由香农于 1948 年借鉴热力学中的“热熵”的概念提出，旨在表示信息的不确定性。熵值越大，则信息的不确定程度越大。其数学公式可以表示为：

$$H(x) = -\sum_{x \in X} p(x)\log(p(x))$$

针对联合分布的情况，其联合信息熵为：

$$H(X|Y) = -\sum_{\substack{y \in Y \\ x \in X}} p(x,y)\log(p(x|y))$$

该公式可用于计算二元或三元等模型。

2. 语言模型

对于自然语言相关的问题，比如机器翻译，最重要的问题就是文本的序列有时候不是符合我们人类的使用习惯，语言模型就是用于评估文本序列符合人类语言使用习惯程度的模型。

当前的语言模型是以统计学为基础的统计语言模型，统计语言模型是基于预先人为收集的大规模语料数据，以真实的人类语言为标准，预测文本序列在语料库中可能出现的概率，并以此概率去判断文本是否“合法”，是否能被人所理解。

N-Gram 是一种基于统计语言模型的算法。它的基本思想是将文本里面的内容按照字节进行大小为 N 的滑动窗口操作，形成了长度是 N 的字节片段序列。

每一个字节片段称为 gram，对所有 gram 的出现频度进行统计，并且按照事先设定好的阈值进行过滤，形成关键 gram 列表，也就是这个文本的向量特征空间，列表中的每一种 gram 就是一个特征向量维度。

该模型基于这样一种假设，第 N 个词的出现只与前面 N-1 个词相关，而与其它任何词都不相关，整句的概率就是各个词出现概率的乘积。这些概率可以通过直接从语料中统计 N 个词同时出现的次数得到。常用的是二元的 Bi-Gram 和三元的 Tri-Gram。

三、实验步骤与结果分析

本次实验中，分别以字和词为单位，以一元、二元和三元模型，针对所给小说进行信息熵计算，将三元信息熵的结果与所给文献中的三元模型英文信息熵进行对比。

1. 文档初步预处理

删除压缩包中的无关文档和文件，仅留下 16 个文档文件，将每个文档中的“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”删除。本步骤为手工完成。

2. 文档内容读取及预处理

```
1.  def entropy_calculate(path, is_chara):
2.      files = os.listdir(path)
3.      data = []
4.      replace = '[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;「<=>?@，。?★、…【】《》？“”‘’！[\\]^_`{|}~]+\n\u3000 '
5.      for file in files:
6.          with open(path + '/' + file, 'r', encoding='ANSI') as f:
7.              t = f.read()
8.              for i in replace:
9.                  t = t.replace(i, '')  #删除文档中含有的 replace 中的字符
10.             if is_chara:  #判断是按词计算还是按字计算
11.                 data.append(t)
12.             else:
13.                 c = jieba.lcut(t)
14.                 data.append(c)
15.         f.close()
```

在读取文档时，将其中非中文的部分全部删除掉，其中的 replace 部分参考参考文献 2。

3. 字词的计数和信息熵的计算

```
#一元模型计数
    uni_chara = {}
    uni_count = 0
    for i in data:
        for j in range(len(i)):
            uni_chara[i[j]] = uni_chara.get(i[j], 0) + 1
            uni_count += 1
#二元模型计数
    bi_chara = {}
    bi_count = 0
    for i in data:
        for j in range(len(i)-1):
            bi_chara[(i[j], i[j+1])] = bi_chara.get((i[j], i[j+1]), 0) + 1
            bi_count += 1
#三元模型计数
    tri_chara = {}
    tri_count = 0
    for i in data:
        for j in range(len(i)-2):
            tri_chara[(i[j], i[j+1], i[j+2])] = tri_chara.get((i[j], i[j+1], i[j+2]), 0) + 1
            tri_count += 1
#熵的计算
    uni_entropy = (sum(-(chara[1])*math.log2(chara[1]/uni_count) for chara in uni_chara.items()))/uni_count
    bi_entropy = (sum(-(chara[1])*math.log2((chara[1]/bi_count)/(uni_chara[chara[0][0]]/uni_count)) for chara in bi_chara.items()))/bi_count
    tri_entropy = (sum(-(chara[1])*math.log2((chara[1]/tri_count)/(bi_chara[(chara[0][0], chara[0][1])]/uni_count)) for chara in tri_chara.items()))/tri_count
```

以字典形式进行计数，将字或词与出现次数以键值对的形式存在字典里，按照公式计算信息熵。

4. 结果与分析

将以上几部分综合为完整程序，计算得到：

按字计算信息熵：一元模型　　二元模型　　三元模型

9.539612077329478 6.723863066662528 3.94464184025448

按词计算信息熵：一元模型　　二元模型　　三元模型

12.176332250579346 6.945766121610492 2.3056611827335547

与文献中英文的三元平均信息熵 1.75 相比，显然中文的平均信息熵要更大，即中文携带的信息量更大。

在 N-gram 模型中，随着 N 取值变大，文本的信息熵则越小，这是因为 N 取值越大，通过分词后得到的文本中词组的分布就越简单，某些固定搭配的出现较为集中，而不常见的搭配出现概率很低，计算得到的信息熵就会变小。

四、参考文献

1. https://docs.qq.com/pdf/DUUR2Z1FrYUVqU0ts

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