



学 期 2021-2022 (2)

北京航空航天大学
BEIHANG UNIVERSITY

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	电子信息
学生姓名	孙茗逸
学号	ZY2103113
指导老师	秦曾昌

















2022 年 4 月

1 引言

1.1 问题描述

第一步：阅读参考文献 *An Estimate of an Upper Bound for the Entropy of English*

第二步：利用上述文章的方法计算 16 本小说的中文信息熵并分析

名称	修改日期	大小	种类
 白马啸西风.txt	2011年10月15日 上午12:05	149 KB	纯文本文稿
 碧血剑.txt	2011年10月15日 上午12:06	985 KB	纯文本文稿
 飞狐外传.txt	2011年10月14日 下午11:56	889 KB	纯文本文稿
 连城诀.txt	2011年10月14日 下午11:56	474 KB	纯文本文稿
 鹿鼎记.txt	2011年10月14日 下午11:58	2.5 MB	纯文本文稿
 三十三剑客图.txt	2011年10月14日 下午11:58	126 KB	纯文本文稿
 射雕英雄传.txt	2011年10月15日 上午12:00	1.9 MB	纯文本文稿
 神雕侠侣.txt	2011年10月15日 上午12:00	1.9 MB	纯文本文稿
 书剑恩仇录.txt	2011年10月15日 上午12:01	1 MB	纯文本文稿
 天龙八部.txt	2011年10月15日 上午12:01	2.5 MB	纯文本文稿
 侠客行.txt	2011年10月15日 上午12:02	751 KB	纯文本文稿
 笑傲江湖.txt	2011年10月15日 上午12:03	2 MB	纯文本文稿
 雪山飞狐.txt	2011年10月15日 上午12:03	273 KB	纯文本文稿
 倚天屠龙记.txt	2011年10月15日 上午12:04	1.9 MB	纯文本文稿
 鸳鸯刀.txt	2011年10月15日 上午12:04	76 KB	纯文本文稿
 越女剑.txt	2011年10月15日 上午12:04	36 KB	纯文本文稿

图一：16 本小说

1.2 中文信息熵

1.2.1 信息熵

1948 年，香农从热力学当中借鉴提出信息熵的概念，解决了对信息量化度量的问题。其定义为：

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

1.2.2 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：

$$p(s) = p(\omega_1, \omega_2, \omega_3, \omega_4 \cdots \omega_n)$$

利用条件概率公式：

$$p(\omega_1, \omega_2, \omega_3, \omega_4 \cdots \omega_n) = p(\omega_1)p(\omega_1|\omega_2) \cdots p(\omega_n|\omega_1, \omega_2, \cdots \omega_n)$$

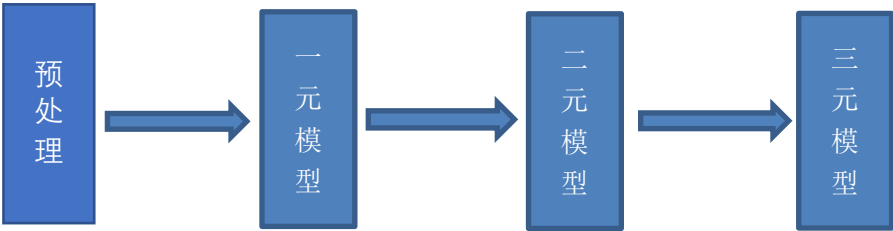
当计算 $p(\omega_1)$ ，仅存在一个参数；计算 $p(\omega_1|\omega_2)$ ，存在两个参数，以此类推，难易计算，所以马尔可夫提出一种假设：假设 ω_i 出现的概率只与前面 N-1 个词相关，当 N=2 时，

就是二元模型，N=3 就是三元模型，本次实验分别使用一元模型、二元模型，三元模型来统计语料库字数，分词个数，平均词长，模型长度，基于模型的中文信息熵和运行时间。

2 中文信息熵实验设计

2.1 流程

- 1) 对数据集进行预处理，删除所有的隐藏符号，标点符号，以及非中文字符
- 2) 分别基于一元模型、二元模型、三元模型计算中文信息熵



2.2 代码设计

使用 Python 语言完成算法和整个实验流程。
详细代码见附录。

3 结果分析与总结

3.1 实验结果

分词模型	语料字数	分词个数	平均词长	信息熵（比特/词）	运行时长(秒)
Unigram	7420081	4430767	1.67467	12.01312	55.05731
Bigram	7420081	4430767	1.67467	6.8915	60.03709
Trigram	7420081	4430767	1.67467	2.41661	74.36535

表 1 不同模型下的指标统计

附录：代码

```
import jieba
import math
import time
import os
import re

class GetData():

    def __init__(self, root):
        self.root = root

    # def ergodic(self):
    #     return self.getCorpus(self, self.root)

    def getCorpus(self):
        corpus = []

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

        listdir = os.listdir(self.root)

        count=0

        for file in listdir:

            path = os.path.join(self.root, file)

            if os.path.isfile(path):

                with open(os.path.abspath(path), "r",
encoding='gb18030') as file:

                    filecontext = file.read();

                    filecontext = re.sub(r1, "",

filecontext)

                    filecontext = filecontext.replace("\n", "")

                    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):

                GetData.AllFiles(self, path)

        return corpus,count

    # 统计词频
    def tf(tf_dic, words):

        for i in range(len(words)-1):

            tf_dic[words[i]] = tf_dic.get(words[i], 0)
+ 1

    def bigram_term_frequency(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 trigram_term_frequency(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 calculate_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
        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():
        entropy.append(-(uni_word[1] / words_len) * math.log(uni_word[1] / words_len, 2))

    print("基于词的一元模型的中文信息熵为:", round(sum(entropy), 5), "比特/词")

    after = time.time()
    print("运行时间:", round(after - before, 5), "秒")

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

```

5))

```
bigram_len = sum([dic[1] for dic in
bigram_tf.items()])
```

```
print("二元模型长度:", bigram_len)
```

```
entropy = []
```

```
for bi_word in bigram_tf.items():
```

```
    jp_xy = bi_word[1] / bigram_len # 计算
联合概率 p(x,y)
```

```
    cp_xy = bi_word[1] /
words_tf[bi_word[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

```
print("基于词的二元模型的中文信息熵
为:", round(sum(entropy), 5), "比特/词")
```

```
after = time.time()
```

```
print("运行时间:", round((after - before), 5),
"秒")
```

```
def calculate_trigram(corpus, count):
```

```
    before = time.time()
```

```
    split_words = []
```

```
    words_len = 0
```

```
    line_count = 0
```

```
    words_tf = {}
```

```
    trigram_tf = {}
```

```
for line in corpus:
```

```
    for x in jieba.cut(line):
```

```
        split_words.append(x)
```

```
        words_len += 1
```

```
        bigram_term_frequency(words_tf,
split_words)
```

```
        trigram_term_frequency(trigram_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))

```
trigram_len = sum([dic[1] for dic in
trigram_tf.items()])
```

```
print("三元模型长度:", trigram_len)
```

```
entropy = []
```

```
for tri_word in trigram_tf.items():
```

```
    jp_xy = tri_word[1] / trigram_len # 计算
联合概率 p(x,y)
```

```
    cp_xy = tri_word[1] /
words_tf[tri_word[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
print("基于词的三元模型的中文信息熵
为:", round(sum(entropy), 5), "比特/词")

after = time.time()
print("运行时间:", round(after - before, 10),
"秒")

data = GetData("/Users/smy/Desktop/HW01/data/")
corpus, count = data.getCorpus()#ergodic()
calculate_unigram(corpus, count)
calculate_bigram(corpus, count)
calculate_trigram(corpus, count)

```

注：代码参考 https://github.com/y852000/NLP_Chinese-entropy