形状, 圆圈

描述已自动生成

学 期 2021-2022（2）

**卡通人物

中度可信度描述已自动生成**

**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 | 熊方书 |
| 学号 | ZY2103309 |
| 指导老师 | 秦曾昌 |

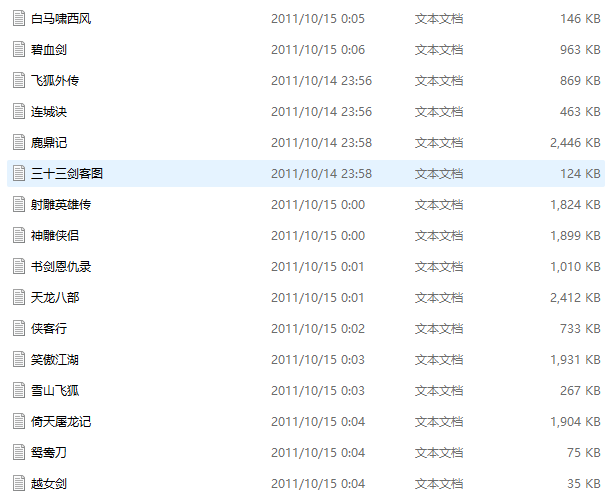
2022年 4月

# 引言

## 问题描述

第一步：阅读 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



图一：16本小说

## 中文信息熵

**1.2.1 信息熵**

信息是个很抽象的概念，人们常说信息很多，或者信息很少，但却很难说清楚信息到底有多少。直到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：

**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，，，…. 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：

利用条件概率公式：

当计算，仅存在一个参数；计算，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，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。

# 中文信息熵实验设计

## 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



## 代码设计

使用Python语言完成算法和整个实验流程。代码输入输出如下所示：

详细代码见附录。

# 结果分析与总结

## 实验结果

表 1 不同模型下的指标统计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |
| Unigram | 7420081 | 44307687 | 1.67467 | 12.01312 | 27.49733 |
| Bigram | 7420081 | 44307687 | 1.67467 | 6.8915 | 30.65374 |
| trigram | 7420081 | 44307687 | 1.67467 | 2.41661 | 37.43707 |

# 代码

import jieba

import math

import time

import os

import re

class GetData():

# 1 初始化

def \_\_init\_\_(self, root):

self.root = root

def ergodic(self):

return GetData.getCorpus(self, self.root)

def getCorpus(self, root):

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(root)

count=0

for file in listdir:

path = os.path.join(root, 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):

GetData.AllFiles(self, path)

return corpus,count

# 词频统计，方便计算信息熵

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))

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")

if \_\_name\_\_ == '\_\_main\_\_':

data = GetData("./data1/")

corpus,count = data.ergodic()

calculate\_unigram(corpus, count)

calculate\_bigram(corpus,count)

calculate\_trigram(corpus,count)

注：代码参考 https://github.com/y852000/NLP\_Chinese-entroy