作业1: 中文的平均信息熵计算

谭天一	ZY2203511	billytan@buaa.edu.cn

1. 实验介绍

阅读论文《An Estimate of an Upper Bound for the Entropy of English》，分别以字和词为单位计算中文的平均信息熵。语料库为16部小说：《白马啸西风》《碧血剑》《飞狐外传》《连城诀》《鹿鼎记》《三十三剑客图》《射雕英雄传》《神雕侠侣》《书剑恩仇录》《天龙八部》《侠客行》《笑傲江湖》《雪山飞狐》《倚天屠龙记》《鸳鸯刀》《越女剑》。

2. 实验过程

实验分为文本加载、文本预处理、字/词频统计、信息熵计算等步骤。

2.1 文本加载

文本加载函数负责读取作为语料库的16部小说，并将其合并为一个字符串`content_mix`。由于Mac电脑的文件夹下自带难以删除的.Ds_Store文件，因此特意设置只读取txt文件。

```
def load_data(path):
    # 这个程序项目由Mac构建，文件路径采用正斜杠'/'，在Windows系统运行请改成'\\'
    content_mix = ''
    novels = os.listdir(path)
    for novel in novels:
        if 'txt' in novel:  # Mac文件夹中含有.Ds_store，应该滤除
            novel_path = path + '/' + novel
            with open(novel_path, 'r', encoding='gb18030') as file:
                content = file.read()
                content_mix += content
                content_mix += '\n'
            file.close()
    return content_mix
```

2.2 文本预处理

首先从给定路径中加载停词列表cn_stopwords.txt，将其读取为列表`stopwords`。基于停词列表对文本加载得到的`content_mix`进行处理，删掉停词列表中包含的关键词。由于停词列表中不包含回车符`'\n'`，因此手动扩展列表`stopwords`，加入回车符。

```python
def data_process(content, stopwords_path):
    with open(stopwords_path, 'r', encoding='utf-8') as file:
        stopwords = file.read().split('\n')
    file.close()
    stopwords.append('\n')
    for stopword in stopwords:
        content = content.replace(stopword, '')
    return content
```

2.3 字/词频统计

一元模型不考虑其他词的影响。

通过get函数对字/词进行计数，然后构建词典`freq_one`，格式为`{word: frequency}`。借助lambda表达式依据词典的`frequency`对词典的内容降序排列。

字/词频统计函数同时适用于以字为单位和以词为单位，当以字为单位时，输入应为`<str>`类型的文本，当以词为单位时，输入应为`<list>`类型的已完成分词的文本

```python
def freq_count_one(content):
    freq_one = {}
    for word in content:
        freq_one[word] = freq_one.get(word, 0) + 1
    freq_one = dict(sorted(freq_one.items(), key=lambda x: x[1], reverse=True))
    return freq_one
```

二元模型考虑前一个词和本词之间的关系。

```python
def freq_count_two(content):
    freq_two = {}
    for i in range(len(content) - 1):
        freq_two[content[i], content[i + 1]] = freq_two.get((content[i], content[i + 1]), 0) + 1
    freq_two = dict(sorted(freq_two.items(), key=lambda x: x[1], reverse=True))
    return freq_two
```

三元模型同理。

```python
def freq_count_three(content):
    freq_three = {}
    for i in range(len(content) - 2):
        freq_three[content[i], content[i + 1], content[i + 2]] = freq_three.get(
            (content[i], content[i + 1], content[i + 2]), 0) + 1
    freq_three = dict(sorted(freq_three.items(), key=lambda x: x[1], reverse=True))
    return freq_three
```

2.4 信息熵计算

一元模型下，信息熵的计算公式为$H(x) = -\sum_{x \in X} P(x) \log P(x)$。

```
def entropy_calculate_one(frequency1):
    entropy = 0
    content_length = sum(frequency1.values())
    for word, freq in frequency1.items():
        prob = freq / content_length
        entropy -= prob * math.log(prob, 2)
    return entropy
```

二元模型下，信息熵的计算公式为$H(X \mid Y) = -\sum_{x \in X, y \in Y} P(x, y) \log P(x \mid y)$，其中$P(x \mid y)$为二元词组的频数与以该二元词组的第一个词为首的二元词组的频数的比。而分母，也就是以该二元词组的第一个词为首的二元词组的频数，即可看作该二元词组的第一个词在一元模型下的频数，程序中可以直接以此代替。三元模型同理。

```
def entropy_calculate_two(frequency2, frequency1):
    entropy = 0
    content_length = sum(frequency2.values())
    for word1, word2 in frequency2:
        prob = frequency2[(word1, word2)] / content_length
        prob_conditional = frequency2[(word1, word2)] / frequency1[word1]
        entropy -= prob * math.log(prob_conditional, 2)
    return entropy
```

三元模型下，信息熵的计算公式为$H(X \mid Y, Z) = -\sum_{x \in X, y \in Y, z \in Z} P(x, y, z) \log P(x \mid y, z)$。

```
def entropy_calculate_three(frequency3, frequency2):
    entropy = 0
    content_length = sum(frequency3.values())
    for word1, word2, word3 in frequency3:
        prob = frequency3[(word1, word2, word3)] / content_length
        prob_conditional = frequency3[(word1, word2, word3)] / frequency2[(word1,
word2)]
        entropy -= prob * math.log(prob_conditional, 2)
    return entropy
```

3. 实验结果

以字为单位	信息熵
一元模型	9.86566397937682
二元模型	6.957636263565023
三元模型	3.511847456690315

以词为单位	信息熵
一元模型	13.488345992329663
二元模型	6.15746702884708
三元模型	1.2965720270876406

4. 总结

随着关联的字词增加，信息熵逐渐变小。当用单字进行统计的时候，由于事实上中文存在很多词语，所以二元模型和三元模型的信息熵仍然较大。而以词为单位进行统计的时候，由于中文的词与词之间的关联不是很紧密，所以二元模型和三元模型的信息熵迅速变小。

本次实验仍然存在一些不足。可能由于停词列表中的关键词不够丰富，导致文本预处理程序对非中文符号的剔除不够彻底，在字/词频统计时统计出了大量奇奇怪怪的符号，可能影响计算结果的精度。

5. 参考文献

1. Brown P F, Della Pietra S A, Della Pietra V J, et al. An estimate of an upper bound for the entropy of English[J]. Computational Linguistics, 1992, 18(1): 31-40.
2. 在完成本次作业时曾阅读过网友“红衣青蛙”的博客https://blog.csdn.net/shzx_55733/article/details/115744123

附录：全部代码

```python
# @Author: 谭天一
# @Description: 根据参考文献Entropy_of_English_PeterBrown和给定语料库计算中文的一元、二元、三元信息熵
# @Date: 2023-3-27

import jieba
import math
import os


def load_data(path):
    # 这个程序项目由Mac构建，文件路径采用正斜杠'/'，在Windows系统运行请改成'\\'
    content_mix = ''
    novels = os.listdir(path)
    for novel in novels:
        if 'txt' in novel:  # Mac文件夹中含有.Ds_store，应该滤除
            novel_path = path + '/' + novel
            with open(novel_path, 'r', encoding='gb18030') as file:
                content = file.read()
                content_mix += content
                content_mix += '\n'
            file.close()
    return content_mix
```

```
23
24
25  def data_process(content, stopwords_path):
26      with open(stopwords_path, 'r', encoding='utf-8') as file:
27          stopwords = file.read().split('\n')
28      file.close()
29      stopwords.append('\n')
30      for stopword in stopwords:
31          content = content.replace(stopword, '')
32      return content
33
34
35  def freq_count_one(content):
36      """
37      同时适用于以字为单位和以词为单位，
38      当以字为单位时，输入应为<str>类型的文本
39      当以词为单位时，输入应为<list>类型的已完成分词的文本
40      本注释也适用于下方的freq_count_two(content)、freq_count_three(content)
41      """
42      freq_one = {}
43      for word in content:
44          freq_one[word] = freq_one.get(word, 0) + 1
45      freq_one = dict(sorted(freq_one.items(), key=lambda x: x[1], reverse=True))
46      return freq_one
47
48
49  def freq_count_two(content):
50      freq_two = {}
51      for i in range(len(content) - 1):
52          freq_two[content[i], content[i + 1]] = freq_two.get((content[i],
    content[i + 1]), 0) + 1
53      freq_two = dict(sorted(freq_two.items(), key=lambda x: x[1], reverse=True))
54      return freq_two
55
56
57  def freq_count_three(content):
58      freq_three = {}
59      for i in range(len(content) - 2):
60          freq_three[content[i], content[i + 1], content[i + 2]] = freq_three.get(
61              (content[i], content[i + 1], content[i + 2]), 0) + 1
62      freq_three = dict(sorted(freq_three.items(), key=lambda x: x[1],
    reverse=True))
63      return freq_three
64
65
66  def entropy_calculate_one(frequency1):
67      entropy = 0
68      content_length = sum(frequency1.values())
69      for word, freq in frequency1.items():
```

```
        prob = freq / content_length
        entropy -= prob * math.log(prob, 2)
    return entropy


def entropy_calculate_two(frequency2, frequency1):
    entropy = 0
    content_length = sum(frequency2.values())
    for word1, word2 in frequency2:
        prob = frequency2[(word1, word2)] / content_length
        prob_conditional = frequency2[(word1, word2)] / frequency1[word1]
        # P(x|y)可近似看作每个二元词组在语料库中出现的频数与以该二元词组的第一个词为词首的二元词组的频数的比值
        # 分母等价于该二元词组的第一个词在一元词组统计中的频数
        entropy -= prob * math.log(prob_conditional, 2)
    return entropy


def entropy_calculate_three(frequency3, frequency2):
    entropy = 0
    content_length = sum(frequency3.values())
    for word1, word2, word3 in frequency3:
        prob = frequency3[(word1, word2, word3)] / content_length
        prob_conditional = frequency3[(word1, word2, word3)] / frequency2[(word1, word2)]
        # P(x|y,z)可近似看作每个三元词组在语料库中出现的频数与以该三元词组的前两个词为词首的三元词组的频数的比值
        # 分母等价于该三元词组的前两个词在二元词组统计中的频数
        entropy -= prob * math.log(prob_conditional, 2)
    return entropy


if __name__ == '__main__':
    content = load_data('./jyxstxtqj_downcc')
    content = data_process(content, './cn_stopwords.txt')

    # 以字为单位求一元模型的信息熵
    frequency1 = freq_count_one(content)
    entropy1 = entropy_calculate_one(frequency1)
    print('以字为单位，一元模型：', entropy1)

    # 以字为单位求二元模型的信息熵
    frequency2 = freq_count_two(content)
    entropy2 = entropy_calculate_two(frequency2, frequency1)
    print('以字为单位，二元模型：', entropy2)

    # 以字为单位求三元模型的信息熵
    frequency3 = freq_count_three(content)
    entropy3 = entropy_calculate_three(frequency3, frequency2)
```

```
    print('以字为单位，三元模型：', entropy3)

    # jieba分词
    content_cut = jieba.lcut(content)

    # 以词为单位求一元模型的信息熵
    frequency1 = freq_count_one(content_cut)
    entropy1 = entropy_calculate_one(frequency1)
    print('以词为单位，一元模型：', entropy1)

    # 以词为单位求二元模型的信息熵
    frequency2 = freq_count_two(content_cut)
    entropy2 = entropy_calculate_two(frequency2, frequency1)
    print('以词为单位，二元模型：', entropy2)

    # 以词为单位求三元模型的信息熵
    frequency3 = freq_count_three(content_cut)
    entropy3 = entropy_calculate_three(frequency3, frequency2)
    print('以词为单位，三元模型：', entropy3)

```