文本处理

这是一篇有关**文本处理**的说明文档。对应的程序为**textprocessingEn.py**。包括以下几方面。

- 编程思路
- 函数说明
- 结果展示
- 参考资料

编程思路

目标是要实现一部小说的处理，包括词频统计，验证zipf-law，查找文本中的特定词语或者类型，本文选择的是小说中的人物名字，最后绘制词云以便更方便地看出词频的大小。
本文使用的模块为os、wordcloud、re、matplotlib
首先是从网上下载一本英文小说（本文为Harry Potter，共7部），因为是多个文件，因此在处理的时候加入了交互，可以输入数字选择处理其中的一本。
得到了小说名字之后，就可以开始着手处理了。这里每一次处理都对应了一个函数，即词（字）频统计、验证zipf-law、名字统计以及词云绘制。
需要注意的是，验证zipf-law和名字统计都将数据写到了文本文件中，因此相应的函数调用一次即可。**程序中使用的写入方法保证每次都是覆盖写入的**，虽然执行多次也没什么问题，但明显冗余，浪费性能。
最后依次调用相应的函数处理即可。

函数说明

定义了一个类，里面封装了需要的函数。

__init()__

```
def __init__(self):
    pass
```

将类实例化的时候并不需要初始化参数，因此pass掉就可以了。

read_file(filelist)

```
def read_file(self, filelist):
    """
    A function for reading a file,or files if you want.
    filename: a list that represents a txt file.
    :return: file name
    """
```

如函数文档所示，需要传递一个包含文件名的列表，以便选择具体的文本进行处理。
获得该列表的方法参考如下：

```
filepath = r"./Harry Potter Env/"
file_name_list = os.listdir(filepath)
# file_name_list = [os.path.join(filepath, file) for file in file_name_list]
# complete path
# print(file_name_list)
```

两种方式都可以，注释掉的是包含文件名完整路径的列表，考虑到交互的简洁性，使用的则是**只有文件名**的列表。

get_content(file_name, mode='rb')

```
def get_content(self, file_name, mode="rb"):
    """
    A function for processing the txt file
    :param file_name: the name of a file
    :return: content
    """
```

该函数用来读取文本文件，返回经过编码的文本内容。一般来说，读取模式是*"rb"*，得到的对象是**byte**类型的。

num_letter(content)

该函数实现的就是字频以及词频的统计。需要用正则表达式筛选掉其中的空白字符（包括空格，换行，制表等）。

zipf_law(content)

得到验证zipf_law需要的数据。即先统计出整个文本中的单词，然后对应标上序号，本文选择从小到大排列，最后将数据写入到**zipf.txt**文件中。
数据写入代码如下:

```
with open('zipf.txt', 'w') as f2:
    for each in result:
        f2.write(str(each)+"\n")                    # The 2nd is done
```

num_name(content)

该函数用来**统计人名**，当然了，不包含重复的。经过分析发现，人名基本跟在 `said` 后面，到标点符号截止。那么正则表达式可以写为 `re.compile(r"\s*said (.*?)[.,;:?!]")`。但这里有个问题，统计出来的人名有的还会有副词，还没想到好的方法筛选掉。
最后也是将数据写入到文本呢文件中。

draw_chart(filename,mode='rb')

需要注意这里传入的参数是存储数据文件的文件名（包含后缀），而不是读取出来的文档内容。

```
def draw_chart(self, filename, mode="rb"):
    """
    Draw the result by using matplotlib.pyplot
    :param filename: A file name
    :param mode: defalut , normally "rb"
    :return: None
    """
```

zipf-law验证函数，由前面的结果可知，数据存储在 `zipf.txt` 中，现在需要对词频进行绘图，但是打开发现，词频的形式是这样的：

(1,

也就是说，需要用到的数据夹在 `(` 和 `,` 之间，于是，正则表达式为： `re.compile(r"\((.?),")`，然后再调用 `findall` 函数就能提取出所有的词频，最后绘图就不再赘述了。

daw_cloud(filename)

```
def daw_cloud(self, filename):
    """
    the cloud picture drew by wordcloud
    :param filename: name of file
    :return: None
    """
```

该函数和前面的绘图函数差不多，只不过需要的数据是单词。单词存储的形式是这样的：

"the"、"I'、'and'、'know"

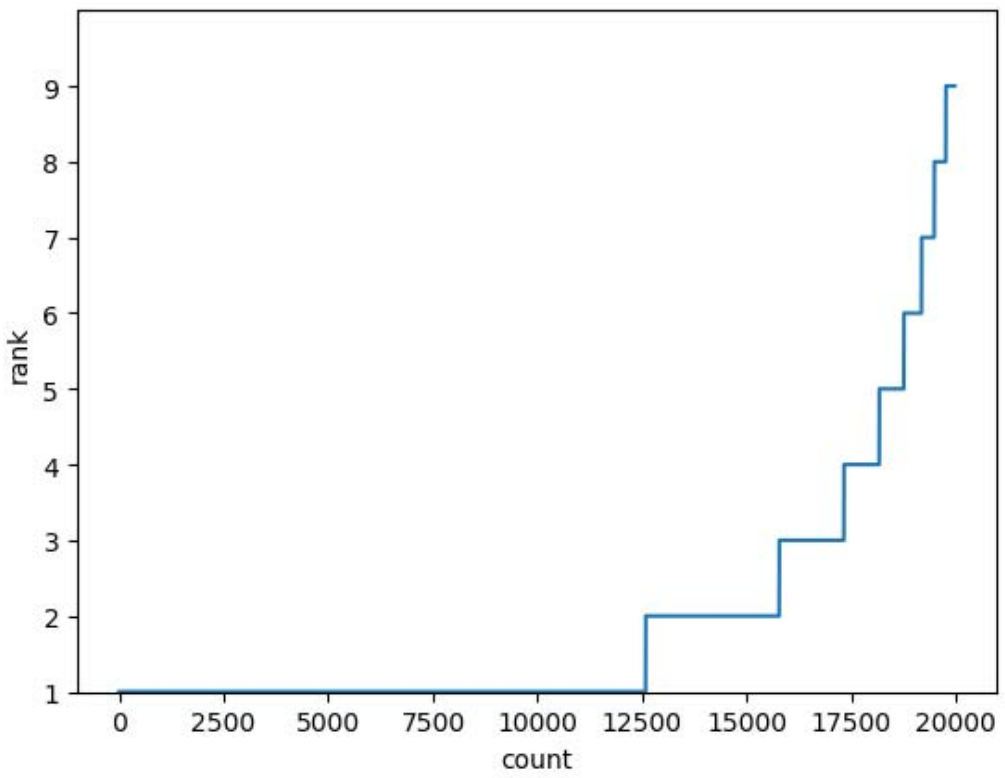
那么可以得到，单词是夹在**引号之间**的。故正则表达式可以这样写： `re.compile(r"[\"'](.*?)[\"']")`，然后再调用 `findall` 函数提取出所有的单词。
最后是词云的绘制，参数如下：

```
background_img = plt.imread('pika.jpg')
            wc = WordCloud(
                background_color='white',
                mask=background_img,
                max_words=2000,
                stopwords=STOPWORDS,
                max_font_size=50,
                random_state=30
            )
            wc.generate(str(result))
            # image_colors = ImageColorGenerator(background_img)
            # wc.recolor(image_colors)
            plt.imshow(wc)
            plt.axis('off')

            plt.show()
```

结果展示

1. 验证zipf-law



   由此可见，词频较高的也只有几个单词而已。
2. 词云

参考资料

1. 教学课件
2. 绘制词云：http://www.cnblogs.com/CQUTWH/p/6129931.html (http://www.cnblogs.com/CQUTWH/p/6129931.html)