

《数据科学与工程算法基础》实践报告

报告题目： 使用簇重要性计算进行抽取式文本摘要

姓 名： 张雯怡

学 号： 10195501425

完成日期： 2021.12.24

提供中英文摘要

摘要 [中文]:

本文主要实现的是通过关键词爬取百度搜索上的语料信息，对每段文本进行文本摘要提取。通过合理构造子模函数，可以将抽取式文本摘要问题转化为集合覆盖问题。本文使用用"簇"（**cluster**）表示关键词的聚集，将分值最高的簇的句子合并构成文本的自动摘要。

Abstract [英语]

The main realization of this paper is to crawl the corpus information on Baidu search through the keywords, and extract the text summary of each paragraph of text. By reasonably constructing submodular functions, the problem of extracted text summarization can be transformed into a set covering problem. This paper uses "cluster" to represent the aggregation of keywords, and combines the sentences of the cluster with the highest score to form the automatic summary of the text.

一、 项目概述（阐明该项目的科学和应用价值，以及相关工作进展并描述项目的主要内容）

文本摘要通过对原网页生成若干个不同的摘要，可以有效增加搜索引擎选择不同站点内相同网页的可能性。

本文采用了抽取式的方法实现文本摘要。

二、 问题描述（问题定义）

抽取式文本摘要：根据文本内容，选择若干关键词，并选择若干段句子，使得选择的所有句子最大覆盖所选择的关键词。

三、 方法（问题解决步骤和实现细节）

以“自然语言处理”为例，

1. 爬虫

爬取百度新闻下含有“自然语言处理”的新闻标题、来源、时间、url，并通过其 url 捕捉新闻正文内容，含翻页。

将得到的内容进行处理，包括去除空行、tag、image、video 以及 JavaScript、stylesheet 和(<script>....</script> and <style>....</style> <!--xxx -->)等非文本内容的信息，以防干扰摘要。

将结果输出在指定文件夹下的文本文件中，如下图所示：

脑 > 桌面 > dase算法 > newscn

名称	修改日期	类型	大小
1	2021/12/24 10:44	文本文档	20 KB
2	2021/12/24 10:29	文本文档	7 KB
3	2021/12/24 10:29	文本文档	9 KB
4	2021/12/24 10:30	文本文档	12 KB
5	2021/12/24 10:30	文本文档	9 KB
6	2021/12/24 10:30	文本文档	6 KB
7	2021/12/24 10:30	文本文档	11 KB
8	2021/12/24 10:30	文本文档	6 KB
9	2021/12/24 10:30	文本文档	6 KB
10	2021/12/24 10:30	文本文档	6 KB

1 - 记事本

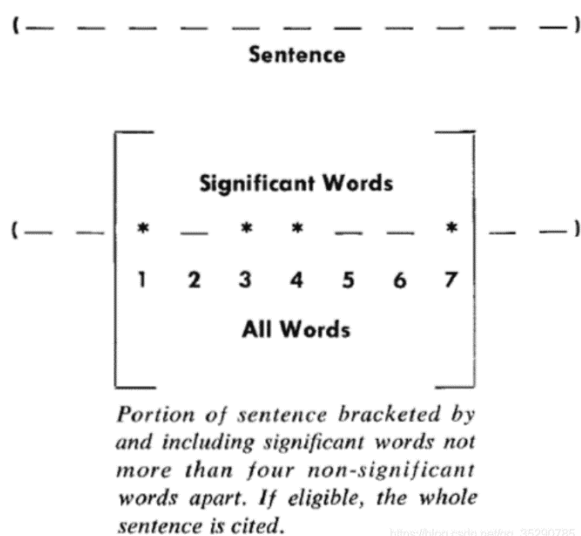
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

鸥玛软件:公司与山东大学合作在自然语言处理、机器学习、智能识别...同花顺财经昨天17:03同花顺(300033)金融研究中心12月23日讯,有投资者向鸥玛软件(300033)提问,公司是否参与了2021年度金猿策划活动——2021大数据产业创新技术突破榜单及奖项”评选。通过深度学习、自然语言处理以及预训练语言模型等前沿人工智能技术的运用,实现数据推理。计算机视觉是人工智能的“眼睛”,因为它能够比人类更快地识别数字图像中的视觉模式、对象、场景和活动。自然语言处理是指识别和理解口语的技术的数据源入手,智慧芽...百度快照容联云AI商业大会:在商业场景下 自然语言处理如何联动决策智能...驱动之家12月13日在前不久容联云举办的「AI有心 决策有智」...

2. 文本摘要

使用 jieba 将文本分词得到 topN 的 words，基于 topN words 对语料中语句进行打分，而 score_sentences 是抽取语句生成摘要的重点。进一步计算得分均值和标准差过滤非重要句子。

Score 计算方式：用“簇”（cluster）来表示关键词的聚集。



上图就是 Luhn 原始论文的插图，被框起来的部分就是一个“簇”。

只要关键词之间的距离小于"门槛值",它们就被认为处于同一个簇之中。Luhn 建议的门槛值是 4 或 5,本文代码中使用的是 5,也就是说,如果两个关键词之间有 5 个以上的其他词,就可以把这两个关键词分在两个簇。

对于两个连续的单词,利用单词位置索引,通过距离阈值计算簇。

簇重要性分值计算公式:

$$\text{簇的重要性} = \frac{(\text{包含的关键词数量})^2}{\text{簇的长度}}$$

https://blog.csdn.net/qq_35290785

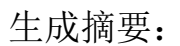
以前图为例,其中的簇一共有 7 个词,其中 4 个是关键词。因此,它的重要性分值等于 $(4 * 4) / 7 = 2.3$ 。

然后,对每个簇打分,每个簇类的最大分数是对句子的打分,只要找出包含分值最高的簇的句子(比如前 5 句),把它们合在一起,就构成了这篇文章的自动摘要。

3. 关键词词频

最后统计这个特定关键词在摘要中的出现频数,以此来简单判断与文本和关键词的相关程度。

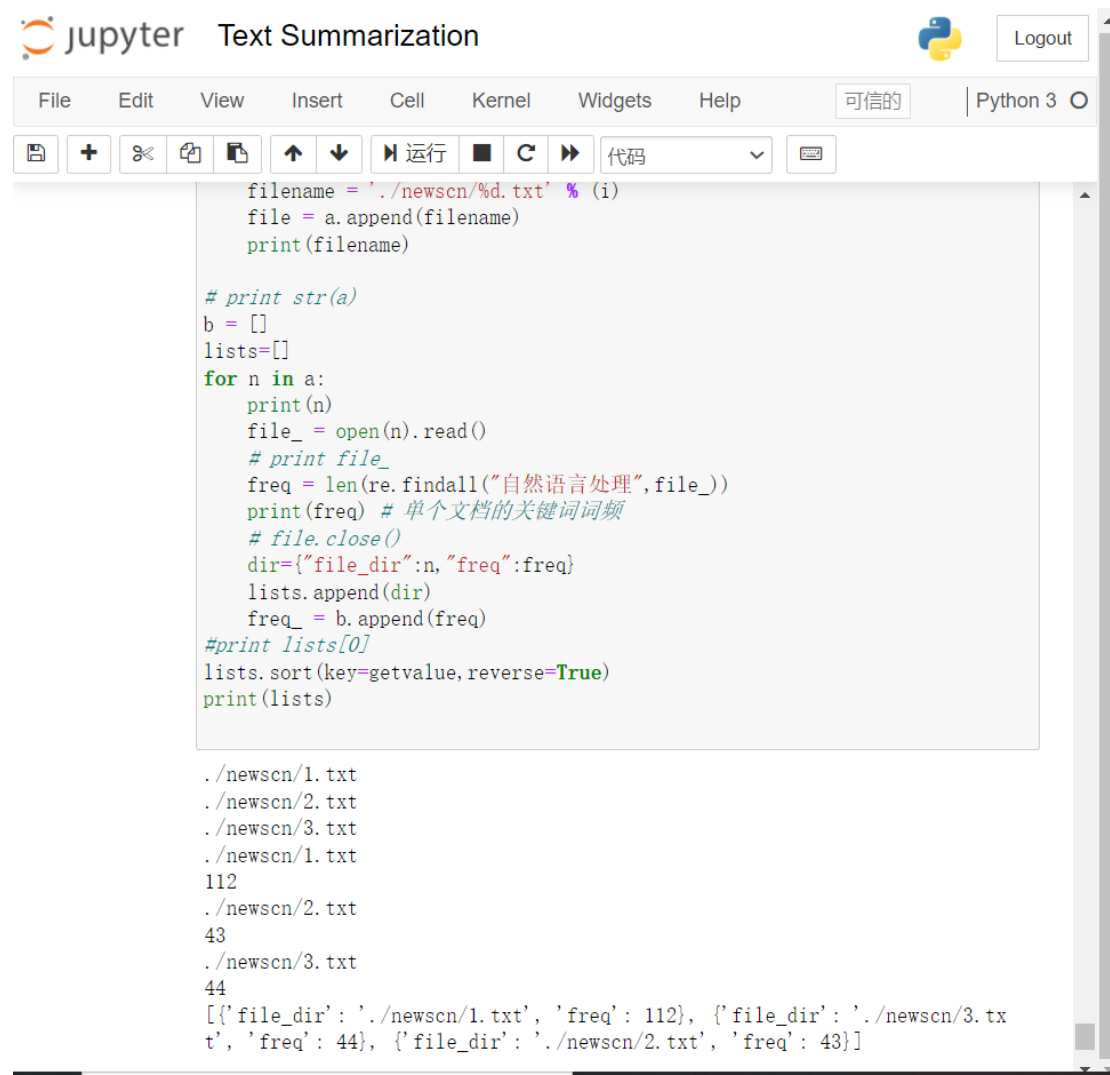
爬虫所得语料:



同花顺财经昨天17:03同花顺(300033)金融研究中心12月23日讯,有投资者向鹏玛软件(301185)提问,从新闻公告中得知公司在自然语言处理、机器学习、智能识别、人机交互及虚拟现实等人工智能技术方面联合山东大学攻关研究、深度合作。

统计每个文本下关键词的词频，并按照从高到低对其词频和文本地址进行排序，简单说明其摘要与关键词的相关性。

这里以 3 个文本摘要为例：



Jupyter Text Summarization

```
filename = './newscn/%d.txt' % (i)
file = a.append(filename)
print(filename)

# print str(a)
b = []
lists=[]
for n in a:
    print(n)
    file_ = open(n).read()
    # print file_
    freq = len(re.findall("自然语言处理",file_))
    print(freq) # 单个文档的关键词词频
    # file.close()
    dir={"file_dir":n,"freq":freq}
    lists.append(dir)
    freq_ = b.append(freq)
#print lists[0]
lists.sort(key=getvalue,reverse=True)
print(lists)

./newscn/1.txt
./newscn/2.txt
./newscn/3.txt
./newscn/1.txt
112
./newscn/2.txt
43
./newscn/3.txt
44
[{'file_dir': './newscn/1.txt', 'freq': 112}, {'file_dir': './newscn/3.txt', 'freq': 44}, {'file_dir': './newscn/2.txt', 'freq': 43}]
```

五、 结论（对使用的方法可能存在的不足进行分析，以及未来可能的研究方向进行讨论）

1. 使用簇重要性计算所抽取出来的句子尽可能多地覆盖了关键词集，而忽略了语义信息，因此生成的语句解释性差，更是较难评估。因此除抽取式文本摘要之外，尚有生成式文本摘要的方法解决这一问题。
2. 语料文本来源于互联网，可以引入常用停用词词典。

代码:

spider.py

```
# coding:utf-8
# 关键词是自然语言处理，能翻页

import re
import urllib.request
import chardet
from bs4 import BeautifulSoup as bs
import requests

# 提取网页正文，放入 txt 中
def remove_js_css(content):
    """ remove the the javascript and the stylesheet and the comment content
    (<script>....</script> and <style>....</style> <!-- xxx -->) """
    content = content.decode('utf-8')
    r = re.compile(r'''<script.*?</script>''', re.I | re.M | re.S)
    s = r.sub('', content)
    r = re.compile(r'''<style.*?</style>''', re.I | re.M | re.S)
    s = r.sub('', s)
    r = re.compile(r'''<!--.*?-->''', re.I | re.M | re.S)
    s = r.sub('', s)
    r = re.compile(r'''<meta.*?>''', re.I | re.M | re.S)
    s = r.sub('', s)
    r = re.compile(r'''<ins.*?</ins>''', re.I | re.M | re.S)
    s = r.sub('', s)
    return s

def remove_empty_line(content):
    """remove multi space """
    r = re.compile(r'''^\s+$''', re.M | re.S)
    s = r.sub('', content)
    r = re.compile(r'''\n+''', re.M | re.S)
    s = r.sub('\n', s)
    return s

def remove_any_tag(s):
    s = re.sub(r'''<[^>]+>''', '', s)
    return s.strip()
```

```

def remove_any_tag_but_a(s):
    text = re.findall(r'''<a[^\r][^>]*>(.*?)</a>''', s, re.I | re.S | re.S)
    text_b = remove_any_tag(s)
    return len(''.join(text)), len(text_b)

def remove_image(s, n=50):
    image = 'a' * n
    r = re.compile(r'''<img.*?>''', re.I | re.M | re.S)
    s = r.sub(image, s)
    return s

def remove_video(s, n=1000):
    video = 'a' * n
    r = re.compile(r'''<embed.*?>''', re.I | re.M | re.S)
    s = r.sub(video, s)
    return s

def sum_max(values):
    cur_max = values[0]
    glo_max = -999999
    left, right = 0, 0
    for index, value in enumerate(values):
        cur_max += value
        if (cur_max > glo_max):
            glo_max = cur_max
            right = index
        elif (cur_max < 0):
            cur_max = 0

    for i in range(right, -1, -1):
        glo_max -= values[i]
        if abs(glo_max < 0.00001):
            left = i
            break
    return left, right + 1

def method_1(content, k=1):
    if not content:
        return None, None, None, None
    tmp = content.split('\n')
    group_value = []

```

```

    for i in range(0, len(tmp), k):
        group = '\n'.join(tmp[i:i + k])
        group = remove_image(group)
        group = remove_video(group)
        text_a, text_b = remove_any_tag_but_a(group)
        temp = (text_b - text_a) - 8
        group_value.append(temp)
    left, right = sum_max(group_value)
    return left, right, len('\n'.join(tmp[:left])),
len('\n'.join(tmp[:right]))

def extract(content):
    content = remove_empty_line(remove_js_css(content))
    left, right, x, y = method_1(content)
    return '\n'.join(content.split('\n')[left:right])

# 输入 url, 将其新闻页的正文输入 txt
def extract_news_content(web_url, file_name):
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36"
    }

    request = requests.get(web_url, headers=headers)
    # opener = urllib.request.build_opener()
    # html = opener.open(request).read()
    html = request.content
    infoencode = chardet.detect(html)['encoding'] # 提取网页的编码
    if html != None and infoencode != None: # 提取内容不为空, error.或者用
else
        html = html.decode(infoencode, 'ignore')
        soup = bs(html, 'lxml')
        content = soup.renderContents()
        content_text = extract(content) # 提取新闻网页中的正文部分, 化为无
换行的一段文字
        content_text = re.sub(" ", " ", content_text)
        content_text = re.sub(">", ">", content_text)
        content_text = re.sub(""", "\"", content_text)
        content_text = re.sub("<[^>]+>", "", content_text)
        content_text = re.sub("\n", "", content_text)
        file = open(file_name, 'a') # append
        file.write(content_text)
        file.close()
        print("输入成功! ")

```

```

# 抓取百度新闻搜索结果:中文搜索, 前 10 页, url: key=关键词
def search(key_word):
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36"
    }

    search_url =
'http://news.baidu.com/ns?word=key_word&tn=news&from=news&cl=2&rn=20&ct
=1'

    search_url = search_url.replace('key_word', key_word)
    print(search_url)
    req = requests.get(search_url, headers=headers)
    #print(req.text)
    real_visited = 0
    http_url=None
    url = "http://www.baidu.com"
    visited_url_list = []
    for count in range(2): # 前 2 页
        if count>0:
            req=requests.get(http_url)
            html = req.content
            soup = bs(html, 'lxml')
            content = soup.findAll( "div", {"class":"result-op"}) # resultset
object
            #print(content)
            next=soup.findAll("a",{"class":"n"})
            #print(next)
            dddd=len(next) - 1
            if dddd<0:
                print("未搜索到")
                return
            ssss=next[dddd].get('href')
            http_url=url+ssss
            print(http_url)
            num = len(content)
            print(str(num))
            for i in range(num):
                p_str = content[i].find('a') # if no result then nontype object
                contenttitle = p_str.renderContents()
                contenttitle = contenttitle.decode('utf-8', 'ignore') # need
it
                contenttitle = re.sub("<[^>]+>", "", contenttitle)
                contentlink = str(p_str.get("href"))

```

```

print(contentlink)
# 存放顺利抓取的 url, 对比
visited_url = open('./visited.txt', 'r') # 是否已经爬过
visited_url_list = visited_url.readlines()
visited_url.close() # 及时 close
exist = 0
if contentlink in visited_url_list:
    exist = 1
else: # 如果未被访问 url
    real_visited += 1
    file_name = './newscn/%d.txt' % (real_visited)
    extract_news_content(contentlink, file_name) # 写入文件
    visited_url_list.append(contentlink)
    visited_url = open('./visited.txt', 'a') # 标记为已访问,
永久存防止程序停止后丢失
    visited_url.write(contentlink + u'\n')
    visited_url.close()
    if len(visited_url_list) >= 120:
        print("解析下一页")
    if len(visited_url_list) >= 120:
        print("解析下一页")

if __name__ == '__main__':
    #key_word = input('input key word:')
    search("自然语言处理")

```

summarize.py

```
# coding:utf-8
# 文本摘要方法有很多，主要分为抽取式和生成式，应用比较多的是抽取式，也比较简单，
# 就是从文本中抽取重要的句子或段落。本方法主要是利用句子中的关键词的距离，主要思想
# 和参考来自阮一峰的网络日志
http://www.ruanyifeng.com/blog/2013/03/automatic\_summarization.html
import nltk
import numpy
import jieba
import codecs

# N=100#单词数量
# CLUSTER_THRESHOLD=5#单词间的距离
# TOP_SENTENCES=5#返回的 top n 句子

#分句
def sent_tokenizer(texts):
    start=0
    i=0#每个字符的位置
    sentences=[]
    punt_list='.!?。! ? ' #',.!?;~, 。! ? : ; ~'.decode('utf8')
    token=""
    for text in texts:
        if text in punt_list and token not in punt_list: #检查标点符号下一个
            #字符是否还是标点
            sentences.append(texts[start:i+1])#当前标点符号位置
            start=i+1#start 标记到下一句的开头
            i+=1
        else:
            i+=1#若不是标点符号，则字符位置继续前移
            token=list(texts[start:i+2]).pop()#取下一个字符
    if start<len(texts):
        sentences.append(texts[start:])#这是为了处理文本末尾没有标点符号的
        #情况
    return sentences

#摘要
def summarize(text):
    N = 100
    TOP_SENTENCES = 5
    #stopwords=load_stopwordslist('./stopwords_cn.txt')
    sentences=sent_tokenizer(text)
    words=[w for sentence in sentences for w in jieba.cut(sentence) if
len(w)>1 and w!='\t']
```

```

wordfre=nltk.FreqDist(words)
topn_words=[w[0] for w in sorted(wordfre.items(),key=lambda
d:d[1],reverse=True)][1:N]
scored_sentences=_score_sentences(sentences,topn_words)
#approach 1,利用均值和标准差过滤非重要句子
avg=numpy.mean([s[1] for s in scored_sentences])#均值
std=numpy.std([s[1] for s in scored_sentences])#标准差
mean_scored=[(sent_idx,score) for (sent_idx,score) in scored_sentences
if score>(avg+0.5*std)]
#approach 2, 返回 top n 句子
top_n_scored=sorted(scored_sentences,key=lambda
s:s[1])[-TOP_SENTENCES:]
top_n_scored=sorted(top_n_scored,key=lambda s:s[0])
top_n_summary = [sentences[idx] for (idx, score) in top_n_scored]
mean_scored_summary = [sentences[idx] for (idx, score) in mean_scored]
di={"top_n_summary":top_n_summary,
"mean_scored_summary":mean_scored_summary}
return di

#句子得分
def _score_sentences(sentences,topn_words):
    CLUSTER_THRESHOLD = 5
    scores=[]
    sentence_idx=-1
    for s in [list(jieba.cut(s)) for s in sentences]:
        sentence_idx+=1
        word_idx=[]
        for w in topn_words:
            try:
                word_idx.append(s.index(w))#关键词出现在该句子中的索引位置
            except ValueError:#w 不在句子中
                pass
        word_idx.sort()
        if len(word_idx)==0:
            continue
        #对于两个连续的单词，利用单词位置索引，通过距离阈值计算族
        clusters=[]
        cluster=[word_idx[0]]
        i=1
        while i<len(word_idx):
            if word_idx[i]-word_idx[i-1]<CLUSTER_THRESHOLD:
                cluster.append(word_idx[i])
            else:
                clusters.append(cluster[:])
                cluster=[word_idx[i]]
                i+=1
        clusters.append(cluster[:])

```

```

        cluster=[word_idx[i]]
        i+=1
    clusters.append(cluster)
    #对每个族打分，每个族类的最大分数是对句子的打分
    max_cluster_score=0
    for c in clusters:
        significant_words_in_cluster=len(c)
        total_words_in_cluster=c[-1]-c[0]+1
        score=1.0*significant_words_in_cluster*significant_words_in_
cluster/total_words_in_cluster
        if score>max_cluster_score:
            max_cluster_score=score
    scores.append((sentence_idx,max_cluster_score))
    return scores;

if __name__=='__main__':
    text = []
    for i in range(1):
        filename = './newscn/%d.txt' % (i+1)
        file = text.append(filename)
        # print filename
    #
    print(text)
    for item in text:
        print(item)
        file_ = open(item).read()
        print(file_)
        dict = summarize(file_)
        file = open(item, 'a+')
        file.write("\n\n 摘要: \n")
        print('-----approach 1-----')
        for sent in dict['top_n_summary']:
            file.write(sent)
            print(sent)
        file.close()

        #print('-----approach 2-----')
        #for sent in dict['mean_scored_summary']:
        #    print(sent)
# coding:utf-8
# 文本摘要方法有很多，主要分为抽取式和生成式，应用比较多的是抽取式，也比较简单，
就是从文本中抽取重要的句子或段落。本方法主要是利用句子中的关键词的距离，主要思想

```


和参考来自阮一峰的网络日志

http://www.ruanyifeng.com/blog/2013/03/automatic_summarization.html

```
import nltk
import numpy
import jieba
import codecs

# N=100#单词数量
# CLUSTER_THRESHOLD=5#单词间的距离
# TOP_SENTENCES=5#返回的 top n 句子

#分句
def sent_tokenizer(texts):
    start=0
    i=0#每个字符的位置
    sentences=[]
    punt_list='.!?. ! ? ' #',.!?;~, 。 ! ? : ; ~'.decode('utf8')
    token=""
    for text in texts:
        if text in punt_list and token not in punt_list: #检查标点符号下一个字符是否还是标点
            sentences.append(texts[start:i+1])#当前标点符号位置
            start=i+1#start 标记到下一句的开头
            i+=1
        else:
            i+=1#若不是标点符号，则字符位置继续前移
            token=list(texts[start:i+2]).pop()#取下一个字符
    if start<len(texts):
        sentences.append(texts[start:])#这是为了处理文本末尾没有标点符号的情况
    return sentences

#摘要
def summarize(text):
    N = 100
    TOP_SENTENCES = 5
    #stopwords=load_stopwordslist('./stopwords_cn.txt')
    sentences=sent_tokenizer(text)
    words=[w for sentence in sentences for w in jieba.cut(sentence) if len(w)>1 and w!='\t']
    wordfre=nltk.FreqDist(words)
    topn_words=[w[0] for w in sorted(wordfre.items(),key=lambda d:d[1],reverse=True)][:N]
    scored_sentences=_score_sentences(sentences,topn_words)
    #approach 1,利用均值和标准差过滤非重要句子
```

```

    avg=numpy.mean([s[1] for s in scored_sentences])#均值
    std=numpy.std([s[1] for s in scored_sentences])#标准差
    mean_scored=[(sent_idx,score) for (sent_idx,score) in scored_sentences
if score>(avg+0.5*std)]
    #approach 2, 返回 top n 句子
    top_n_scored=sorted(scored_sentences,key=lambda
s:s[1])[-TOP_SENTENCES:]
    top_n_scored=sorted(top_n_scored,key=lambda s:s[0])
    top_n_summary = [sentences[idx] for (idx, score) in top_n_scored]
    mean_scored_summary = [sentences[idx] for (idx, score) in mean_scored]
    di={"top_n_summary":top_n_summary,
"mean_scored_summary":mean_scored_summary}
    return di

#句子得分
def _score_sentences(sentences,topn_words):
    CLUSTER_THRESHOLD = 5
    scores=[]
    sentence_idx=-1
    for s in [list(jieba.cut(s)) for s in sentences]:
        sentence_idx+=1
        word_idx=[]
        for w in topn_words:
            try:
                word_idx.append(s.index(w))#关键词出现在该句子中的索引位置
            except ValueError:#w 不在句子中
                pass
        word_idx.sort()
        if len(word_idx)==0:
            continue
        #对于两个连续的单词，利用单词位置索引，通过距离阈值计算族
        clusters=[]
        cluster=[word_idx[0]]
        i=1
        while i<len(word_idx):
            if word_idx[i]-word_idx[i-1]<CLUSTER_THRESHOLD:
                cluster.append(word_idx[i])
            else:
                clusters.append(cluster[:])
                cluster=[word_idx[i]]
            i+=1
        clusters.append(cluster)
        #对每个族打分，每个族类的最大分数是对句子的打分
        max_cluster_score=0

```

```

        for c in clusters:
            significant_words_in_cluster=len(c)
            total_words_in_cluster=c[-1]-c[0]+1
            score=1.0*significant_words_in_cluster*significant_words_in_
cluster/total_words_in_cluster
            if score>max_cluster_score:
                max_cluster_score=score
            scores.append((sentence_idx,max_cluster_score))
        return scores;

if __name__=='__main__':
    text = []
    for i in range(1):
        filename = './newscn/%d.txt' % (i+1)
        file = text.append(filename)
        # print filename
    #
    print(text)
    for item in text:
        print(item)
        file_ = open(item).read()
        print(file_)
        dict = summarize(file_)
        file = open(item, 'a+')
        file.write("\n\n 摘要: \n")
        print('-----approach 1-----')
        for sent in dict['top_n_summary']:
            file.write(sent)
            print(sent)
        file.close()

    #print('-----approach 2-----')
    #for sent in dict['mean_scored_summary']:
    #    print(sent)

```

frequence.py

```
import re
def getvalue(direct):
    return direct["freq"]
a = []
for i in range(1,2):
    filename = './newscn/%d.txt' % (i)
    file = a.append(filename)
    print(filename)

# print str(a)
b = []
lists=[]
for n in a:
    print(n)
    file_ = open(n).read()
    # print file_
    freq = len(re.findall("自然语言处理",file_))
    print(freq) # 单个文档的关键词词频
    # file.close()
    dir={"file_dir":n,"freq":freq}
    lists.append(dir)
    freq_ = b.append(freq)
#print lists[0]
lists.sort(key=getvalue,reverse=True)
print(lists)

# txt = 'xxhello danh 但是 ello 但是 sdfsd 但是 hf 速度'
# list = re.findall("但是", txt)
# print len(list)
```