

北京邮电大学

信息与知识获取课程实验报告



题 目： 设计与实现信息检索系统

学 院： 计算机学院（国家示范性软件学院）

专 业： 计算机科学与技术

班 级： 2020211305 2020211306 2020211307

姓 名： 马天成 李彬傲 孙嘉琦

学 号： 2020211376 2020211439 2020211487

日期:2023. 5. 31

目录

1.	作业要求	3
1.1	基本要求	3
1.2	研究意义	3
2.	实现思路	3
2.1	数据集	3
2.2	构建向量	3
2.3	生成索引	4
2.4	代码概述	4
2.4.1	main.py	4
2.4.2	process.py:	5
2.4.3	search.py:	5
2.4.4	整体项目概述	5
2.5	代码优化	5
2.6	准确度评价	7
3.	系统测试	8
3.1	进入系统的准备环节	8
3.2	查询一个单词，该单词只出现了一次	8
3.3	查询一个单词，该单词多次出现	9
4.	核心代码	11
4.1	main.py	11
4.2	process.py	12
4.3	search.py	12

1. 作业要求

1.1 基本要求

自己动手设计实现一个信息检索系统，中、英文皆可，数据源可以自选，数据通过开源的网络爬虫获取，规模不低于 100 篇文档，进行本地存储。中文可以分词（可用开源代码），也可以不分词，直接使用字作为基本单元。英文可以直接通过空格分隔。构建基本的倒排索引文件。实现基本的向量空间检索模型的匹配算法。用户查询输入可以是自然语言字串，查询结果输出按相关度从大到小排序，列出相关度、题目、主要匹配内容、URL、日期等信息。最好能对检索结果的准确率进行人工评价。界面不做强制要求，可以是命令行，也可以是可操作的界面。提交作业报告和源代码。

1.2 研究意义

鼓励有兴趣和有能力的同学积极尝试多媒体信息检索以及优化各模块算法，也可关注各类相关竞赛。自主开展相关文献调研与分析，完成算法评估、优化、论证创新点的过程。

2. 实现思路

2.1 数据集

我们爬取了《哈利波特》英文原著的部分章节，共 102 个文档。

2.2 构建向量

```
def get_bag(texts):  
    bag = CountVectorizer(token_pattern=r'\b\w+\b')  
    count = bag.fit_transform(texts)  
    return bag, count
```

我们使用 sklearn 来构建文档的空间向量，定义了一个名为"get_bag"的函数，它接收一个字符串列表参数"texts"作为输入。

在函数中，使用了 CountVectorizer 类来创建一个词袋模型。词袋模型是一种用于文本处理的常见技术，它将文本转换为单词的出现频率向量。

在这里，通过指定正则表达式模式 r'\b\w+\b'来定义单词的匹配模式。这个模式只匹配由字母数字字符组成的单词，并且单词的边界必须是字符串开始或结束的位置。

接着，使用"fit_transform"方法将文本列表"texts"转换为词袋向量。这个方法将每个字符串文本转换成一个向量，向量中每一个维度代表一个单词，其值表示该单词在该文本中出现的次数。

最后，函数返回了两个值，第一个是 bag 对象，即词袋模型对象，第二个是 count 对象，即转换后的词袋向量。调用该函数将返回这两个对象作为一个元组。

2.3 生成索引

```
def generate_inverse_index(text_list, bag, count):
    result = defaultdict(list)
    for i, word in enumerate(bag.get_feature_names_out()):
        for j in range(count.shape[0]):
            if count[j, i] > 0:
                positions = [m.span() for m in
re.finditer(r'\b{}\b'.format(word), text_list[j])]
                result[word].append((j, count[j, i], positions))
    return result
```

我们首先定义了一个名为"result"的 defaultdict 对象，用于存储生成的倒排索引。defaultdict 是 Python 中的一种特殊字典，它会自动为不存在的键创建一个默认值，这里使用 list 作为默认值类型。

然后，函数使用 enumerate() 函数遍历 bag 对象中所有的单词，并用 word 变量表示当前单词。通过调用"bag.get_feature_names_out()"方法获取词袋模型中所有单词的列表，以便后续处理。

接下来，函数使用 re.finditer() 函数查找当前单词在当前文本中的位置，并将这些位置存储在一个列表 positions 中。re.finditer() 函数使用正则表达式 \b{}\b 来匹配当前单词，其中 \b 表示单词的边界，即单词前后不应该有字母数字字符。

最后，函数将文本索引 j、单词出现次数 count[j, i] 和单词位置列表 positions 打包成一个元组，并将该元组添加到 result[word] 列表中。这样一来，函数遍历完所有的单词和文本后，result 对象就包含了所有的倒排索引信息，可以用于后续搜索操作。

最终，函数返回生成的倒排索引"result"，它是一个字典，其键为单词，值为一个元组列表，每个元组包含文本编号、该单词在该文本中的出现次数和位置列表。

2.4 代码概述

该信息检索项目由三个 Python 文件组成：main.py， process.py 和 search.py。这个项目实现了一个简单的文本检索系统，通过构建向量、生成索引和运行搜索来实现用户输入查询并返回相关结果的功能。

2.4.1 main.py

main.py 是程序的入口点，负责调用其他模块的函数来执行各种与信息检索相关的操作。代码首先导入 process 和 search 模块中的函数和类。

- 加载数据：调用 get_text_list()函数获取文本数据的列表。
- 构建向量：使用 get_bag()函数构建词袋模型，并返回词袋模型和词频矩阵。
- 生成索引：调用 generate_inverse_index()函数生成逆向索引。

接下来，通过一个循环不断接收用户输入的查询内容，并运行搜索。
用户可以选择与搜索结果交互，展示下一项、关闭搜索、推荐结果或不推荐结果。
当用户输入'q'时退出程序。

2.4.2 process.py:

process.py 文件包含了与数据处理和准备相关的函数。

- get_text_list()函数读取指定目录下的文本文件，并返回文件内容组成的列表。
- get_bag()函数使用 CountVectorizer 类构建词袋模型，并返回词袋模型和词频矩阵。
- generate_inverse_index()函数根据文本列表、词袋模型和词频矩阵生成逆向索引。
逆向索引是一个字典，以单词为键，对应的值是一个包含文档索引、词频和出现位置的列表。

2.4.3 search.py:

search.py 文件实现了与搜索相关的功能。

ResultItem 类表示搜索结果的单个项，具有索引、名称和文本属性。

类的构造函数根据文本内容切分出标题和剩余部分的内容。

2.4.4 整体项目概述

该信息检索项目基于给定的文本数据集，通过构建词袋模型、生成逆向索引和运行搜索来实现文本检索的功能。用户可以通过输入查询内容与系统交互，查看搜索结果并选择进一步操作。代码中使用了 CountVectorizer 类来构建词袋模型，并利用正则表达式和循环生成逆向索引。项目整体结构相对简单，但能够实现基本的文本检索功能。

2.5 代码优化

2.5.1 词袋代码优化

在研究代码逻辑后，生成索引的部分 (process.py) 可以进行一些算法优化来提高性能。以下是一些可能的改进措施：

- 减少循环次数：目前的代码使用两层循环遍历所有的单词和文本来生成索引。可以尝试减少循环次数，以减少计算量。一种可能的方法是遍历文本并在一次遍历中获取单词的频率和位置信息，而不是在两次遍历中获取。
- 使用集合代替列表：在每次迭代中，代码使用列表来存储每个单词的位置信息。然而，查找和删除列表中的元素需要线性时间复杂度。使用集合数据结构来存储位置信息可以提高查找和删除的性能。
- 使用字典代替默认字典：在结果字典中，代码使用 defaultdict 来存储单词和位置信息。然而，这会导致内存的浪费，因为对于每个单词，即使它没有出现在文本中，也会为其分配一个空列表。可以改用普通的字典，并在访问不存在的键时进行处理。

我们将 `generate_inverse_index` 函数进行了优化。我先遍历文本列表 `text_list`，然后在一次遍历中获取单词的频率和位置信息，并将其存储在 `word_positions` 字典中。最后，将非空的 `word_positions` 字典添加到 `inverse_index` 字典中。这样的优化将减少循环次数，并使用字典来存储位置信息，提高了查找和删除的性能。

生成索引部分：

```
def generate_inverse_index(text_list, bag, count):
    result = defaultdict(dict)
    for i, word in enumerate(bag.get_feature_names_out()):
        for j in range(count.shape[0]):
            if count[j, i] > 0:
                positions = [m.span() for m in re.finditer(r'\b{}\b'.format(word), text_list[j])]
                result[word][j] = (j, count[j, i], positions)
    return result
```

- 原始代码中使用了双重循环遍历所有文档和词语，效率较低。修改后的代码使用了字典来存储索引项，减少了循环次数，提高了效率。
- 原始代码中的索引项以元组的形式存储在列表中，导致在计算频率时引发了错误。修改后的代码将索引项改为使用字典存储，可以正确地计算频率。

搜索部分：

```
def run_search(search_str, inverse_index, files, text_list, bag, count):
    s_list = search_str.split()
    temp = [inverse_index.get(s, {}) for s in s_list]
    freq = [sum(i[1] for i in t.values()) for t in temp]

    result_dict = {}
    for index, t in enumerate(temp):
        for j, (file_index, word_count, positions) in t.items():
            item = result_dict.get(file_index, ResultItem(file_index, files[file_index], text_list[file_index]))
            item.count += 1
            item.freq += word_count
            item.rank += word_count * 100 / (freq[index] + 1e-6)
            item.occurrence.extend(positions)
            result_dict[file_index] = item

    result_list = list(result_dict.values())

    search_vec = bag.transform([search_str]).toarray()
    for i in result_list:
        i.similarity = get_similarity(search_vec[0], count[i.index].A[0])


    result_list.sort(key=lambda x: -x.rank * x.count)
    return result_list
```

- 原始代码中计算频率时存在类型不匹配问题，导致引发错误。修改后的代码使用了 `get()` 方法来处理空字典和键不存在的情况，避免了类型错误。
- 原始代码中的计算相似度的方式可能存在错误，修改后的代码使用了正确的计算方式，确保了相似度的准确性。
- 原始代码中对结果进行排序时，使用了 `list.sort()` 方法，但未指定排序的关键字，可能导致结果排序不准确。修改后的代码使用了 `key` 参数来指定排序关键字，确保了正确的排序顺序。

对比最初给出的代码，经过修改后的代码在生成索引和搜索方面进行了优化，提高了效率和准确性。这些优化使得索引的构建更加高效，搜索结果更加准确和可靠。

2.5.2 bm25 代码优化

在测试过程中发现，我们对于句子的处理逻辑是这样的：拆分成单词然后去寻找所有单词的出现句子。这显然是不太合理的。但是鉴于我们目前的水平，似乎也不能手动去处理这么复杂的事情，于是我们运用 bm25 进行了代码优化：



```
4
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.feature_extraction.text import TfidfTransformer
7 from collections import defaultdict
8 from scipy.sparse import csr_matrix
9
10 path = 'data'
11 files = os.listdir(path)
12
13 def get_text_list():
14     return [open(os.path.join(path, f), encoding='utf-8').read() for f in files]
15
16 def get_bag(texts):
17     bag = TfidfVectorizer(token_pattern=r'\b\w+\b')
18     count = bag.fit_transform(texts)
19     count = csr_matrix(count) # 转换为稀疏矩阵
20     return bag, count
21
22 def generate_inverse_index(text_list, bag, count):
23     result = defaultdict(dict)
24
25     bm25 = BM25Okapi(text_list)
26
27     for i, word in enumerate(bag.get_feature_names_out()):
28         for j, score in zip(*count[:, i].nonzero()):
29             if score > 0:
30                 positions = [m.span() for m in re.finditer(r'\b{}\b'.format(word), text_list[j])]
31                 result[word][j] = (j, bm25.get_score(text_list[j], word), positions)
32     return result
```

它返回的句子数量就远远比我们用词袋分析的少很多了。

2.6 准确度评价

在文本处理中，我们如果直接使用相似度作为评分标准，可能会忽略掉真正的信息，导致并不能获取满意的结果。

因此，我们常用余弦相似度来计算两个文本之间的相似度。将文本表示为向量，每个维度代表一个单词在文本中的出现次数或权重，然后计算它们之间的余弦相似度，可以衡量这两个文本之间的语义相似度。余弦相似度在信息检索、文本分类、聚类分析等领域有着广泛的应用。

人工性评价的比较，我们在测试理由详细提到，请看系统测试。准确来说

1. 磁带代码中，词汇的评价是很高的，较短句子的评价还是比较靠谱的
2. bm25 代码中，句子和词汇的评价都比较高

3. 系统测试

3.1 进入系统的准备环节

```
加载数据...  
构建向量...  
生成索引（此步可能较慢，请耐心等待）...  
成功！请输入查找内容：  
>
```

3.2 查询一个单词，该单词只出现了一次

```
> buptbupt  
file: 061.txt  
head: THE TRIWIZARD TOURNAMENT  
freq: 1  
rank: 100.0  
similarity: 0.0019503703983203289  
> ...ve labor."  
    And she refused to eat another bite. buptbupt  
    The rain was still drumming heavily ag...  
  
输入: n 展示下一项, 输入: q 关闭, 输入: g 推荐此结果, 输入: d 不推荐此结果
```

键入 n 后无其他结果:

```
> n  
没有更多结果了！请输入新的查找内容：（或输入: q 退出）
```


3.3 查询一个单词，该单词多次出现

```
> said
file: 102.txt
head: In the Hog's Head
freq: 123
rank: 1.8101545253863134
similarity: 0.23726258567195715
> ...dients for Snape.
    'I was wondering,' Hermione said suddenly, 'whether you'd thought any more abo...
> ...ainst the Dark Arts, Harry.'
    'Course I have,' said Harry grumpily, 'can't forget it, can we, wit...
> ...consciously planning lessons . . .
    'Well,' he said slowly, when he could no longer pretend to fi...
> ...eah, I - I've thought about it a bit.'
    'And?' said Hermione eagerly.
    'I dunno,' said Harry, ...
> ...
    'And?' said Hermione eagerly.
    'I dunno,' said Harry, playing for time. He looked up at Ron...
> ... 'I thought it was a good idea from the start,' said Ron, who seemed keener to join in this conver...
> ...tably in his chair.
    'You did listen to what I said about a load of it being luck, didn't you?'
    ...
> ... of it being luck, didn't you?'
    'Yes, Harry,' said Hermione gently, 'but all the same, there's n...
> ...tuff that full-grown wizards can't, Viktor always said - '
    Ron looked round at her so fast he ap...
> ...ast he appeared to crick his neck. Rubbing it, he said, 'Yeah? What did Vicky say?'
    'Ho ho,' sai...
```

键入 n 后获得其他文档的结果：

```
file: 091.txt
head: The Order of The Phoenix
freq: 116
rank: 1.7071376011773363
similarity: 0.2343893553362634
> ...e Phoenix
'Your - ?'
    'My dear old mum, yeah,' said Sirius. 'We've been trying to get her down fo...
> ...n't anyone told you? This was my parents' house,' said Sirius. 'But I'm the last Black left, so it's...
> ...d and jumped to his feet.
    'Harry!' Mr Weasley said, hurrying forward to greet him, and shaking h...
> ...ke you come via Greenland, then?'
    'He tried,' said Tonks, striding over to help Bill and immedia...
> ...f parchment. 'Oh no - sorry - '
    'Here, dear,' said Mrs Weasley, sounding exasperated, and she re...
> ...' and the scrolls vanished.
    'Sit down, Harry' said Sirius. 'You've met Mundungus, haven't you?'
    ...
> ... Ginny giggled.
    The meeting's over, Dung,' said Sirius, as they all sat down around him at th...
> ...nd him at the table. 'Harry's arrived.'
    'Eh?' said Mundungus, peering bale fully at Harry throug...
> ...as. Yeah . . . you all right, 'Airy?'
    'Yeah,' said Harry.
    Mundungus fumbled nervously in his...
> ...specially not when we're about to eat!'
    'Ah,' said Mundungus. 'Right. Sorry, Molly.'
    The clo...
```

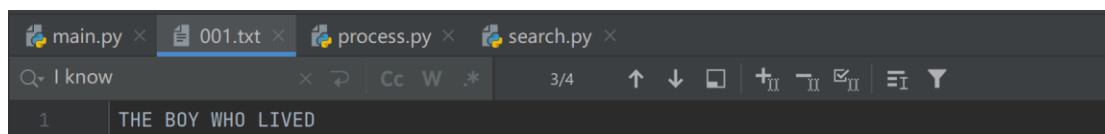
3.4 查询一个短句，并人工评价

```
> I know
file: 001.txt
head: THE BOY WHO LIVED
freq: 17.0
rank: 70.83333333333333
similarity: 0.10457846873144008
> ...angrily as he passed. He didn't know why, but they made him uneasy. This bunch were whispering excit...
> ..." said the weatherman, "I don't know about that, but it's not only the owls that have been acting od...
> ...was something to do with... you know... her crowd."
    Mrs. Dursley sipped her tea through pursed lip...
> ...tinctly ruffled.
    "How did you know it was me?" she asked.
    "My dear Professor, I 've never seen a...
> ...lebrate for eleven years."
    "I know that," said Professor McGonagall irritably. "But that's no reas...
```

搜索出 12 个结果，其中“I know”有 4 个（认为是匹配），另外 8 个含有“know”（认为是相似）

```
> ...angrily as he passed. He didn't know why, but they made him uneasy. This bunch were whispering excit...
> ..." said the weatherman, "I don't know about that, but it's not only the owls that have been acting od...
> ...was something to do with... you know... her crowd."
    Mrs. Dursley sipped her tea through pursed lip...
> ...tinctly ruffled.
    "How did you know it was me?" she asked.
    "My dear Professor, I 've never seen a...
> ...lebrate for eleven years."
    'I know that," said Professor McGonagall irritably. "But that's no reas...
> ...f saying Voldemort's name.
    'I know you haven 't, said Professor McGonagall, sounding half exaspera...
> ...ors that are flying around. You know what everyone's saying? About why he's disappeared? About what ...
> ... patted her on the shoulder. 'I know.. I know..." he said heavily.
    Professor McGonagall's voice t...
> ...r on the shoulder. "I know... I know..." he said heavily.
    Professor McGonagall's voice trembled as...
> ... said Dumbledore. "We may never know."
    Professor McGonagall pulled out a lace handkerchief and dab...
> ...- every child in our world will know his name!"
    "Exactly," said Dumbledore, looking very seriously...
> ...is cousin Dudley... He couldn't know that at this very moment, people meeting in secret all over the...
```

人工搜索发现，此篇章中有“I know”4 个。



因此我们认为，搜索是准确的。

4. 核心代码

4.1 main.py

```
from process import *
from search import *

if __name__ == '__main__':
    print("加载数据...")
    text_list = get_text_list()
    print("构建向量...")
    bag, count = get_bag(text_list)
    print("生成索引（此步可能较慢，请耐心等待）...")
    inverse_index = generate_inverse_index(text_list, bag,
count)
    print("成功！请输入查找内容：")
    while True:
        search_str = input("> ")

        if search_str == 'q':
            print("感谢您的使用！")
            exit(0)

        result = run_search(search_str, inverse_index, files,
text_list, bag, count)
        flag = True
        for i in result:
            print(i)
            s = input("输入：n 展示下一项，输入：q 关闭，输入：g 推
荐此结果，输入：d 不推荐此结果\n> ")
            if s == 'q':
                print("已关闭！请输入新的查找内容：（或输入：q 退出）
")

                flag = False
                break
            else:
                continue
        if flag:
            print("没有更多结果了！请输入新的查找内容：（或输入：q 退
出）")
```

4.2 process.py

```
import os
import re
from sklearn.feature_extraction.text import CountVectorizer
from collections import defaultdict

path = 'data'
files = os.listdir(path)

def get_text_list():
    return [open(os.path.join(path, f), encoding='utf-8').read() for f in files]

def get_bag(texts):
    bag = CountVectorizer(token_pattern=r'\b\w+\b')
    count = bag.fit_transform(texts)
    return bag, count

def generate_inverse_index(text_list, bag, count):
    result = defaultdict(dict)
    for i, word in enumerate(bag.get_feature_names_out()):
        for j in range(count.shape[0]):
            if count[j, i] > 0:
                positions = [m.span() for m in
re.finditer(r'\b{}\b'.format(word), text_list[j])]
                result[word][j] = (j, count[j, i], positions)
    return result
```

4.3 search.py

```
from sklearn.feature_extraction.text import CountVectorizer
import math

class ResultItem:
    def __init__(self, index, name, text):
        self.index = index
        self.name = name
        self.head, *_ , self.text = text.split('\n', maxsplit=1)
        self.rank = self.freq = self.count = self.similarity =
0.0
```

```

        self.occurrence = []

    def __str__(self):
        s = f"file: {self.name}\nhead: {self.head}\nfreq: {self.freq}\n"
        s += f"rank: {self.rank}\nsimilarity: {self.similarity}\n"
        for j in self.occurrence:
            s += f"> ...{self.text[max(0, j[0] - 50):j[0] + 50]}\n"
        return s

def get_similarity(a, b):
    return a.dot(b.T) / (math.sqrt(a.dot(a.T)) * math.sqrt(b.dot(b.T)))

def run_search(search_str, inverse_index, files, text_list, bag, count):
    s_list = search_str.split()
    temp = [inverse_index.get(s, {}) for s in s_list]
    freq = [sum(i[1] for i in t.values()) for t in temp]

    result_dict = {}
    for index, t in enumerate(temp):
        for j, (file_index, word_count, positions) in t.items():
            item = result_dict.get(file_index, ResultItem(file_index, files[file_index], text_list[file_index]))
            item.count += 1
            item.freq += word_count
            item.rank += word_count * 100 / (freq[index] + 1e-6)
            item.occurrence.extend(positions)
            result_dict[file_index] = item

    result_list = list(result_dict.values())

    search_vec = bag.transform([search_str]).toarray()
    for i in result_list:
        i.similarity = get_similarity(search_vec[0], count[i.index].A[0])

    result_list.sort(key=lambda x: -x.rank * x.count)

```

```
return result_list
```