技术名称	信息检索实验 1
小组成员	吴静

实验 成绩	

华中师范大学计算机科学系

实验报告书

课程名称：信息检索技术

主讲教师：王成济

学　　号：2023214461

姓　　名：吴静

组　　号：_____

年　　月　　日

标题用二号宋简体，正文用三号仿宋，一级标题用三号黑体，二级标题用三号楷体加粗，三级标题用三号仿宋加粗。行间距 28 磅。

标题不缩进，正文首行缩进 2 字符。

目录

1 选题介绍

本实验旨在利用结巴中文分词算法构建一个稀疏检索系统原型，并基于曹文轩小说数据集进行实验验证。选择该主题的原因在于中文分词是信息检索领域的基础且关键的一步，尤其对于中文文本的处理至关重要。曹文轩的小说数据集具有丰富的词汇多样性和语言表达形式，能够充分测试中文分词算法的处理能力，并为检索系统的准确性测试提供明确的标准。同时，适中的篇幅也便于实验操作和结果分析。

2 自己在小组中承担的工作、以及小组分工

个人作业

3 项目实现过程，以及遇到的困难和收获

1. 环境配置

```
pip3 install jieba
```

2. 数据集描述与统计分析

a. 数据集概述

本实验采用曹文轩的五部经典小说作为数据集，包括《青铜葵花》《细米》《根鸟》《草房子》和《红瓦黑瓦》。这些作品均为中国当代儿童文学的代表作品，语言优美，情节丰富，具有典型的中文文学特色。选择这些作品作为数据集的原因如下：

- 这些小说具有丰富的词汇多样性和语言表达形式，能够充分测试中文分词算法的处理能力。曹文轩的作品以细腻的心理描写和诗意的语言著称，包含了大量的成语、俗语和文学性表达，为分词系统提供了具有挑战性的测试材料。
- 每部小说都有独特的人物、地名和情节设置，这为检索系统的准确性测试提供了明确的标准。例如，"桑桑"是《草房子》的主人公，"根鸟"是《根鸟》的主人公，这种明确的对应关系便于设计和评估检索算法的准确性。
- 这些作品的篇幅适中，既能提供足够的文本量用于构建有效的索引，又不会因为数据量过大而影响实验的可操作性和结果的可解释性。

b. 原始数据统计分析

> 参考
>
> python 中文分词工具:结巴分词 jieba_jieba 参数

> -CSDN 博客
>
> https://zhuanlan.zhihu.com/p/502691059

运行 data_preprocessing.py 得到原始数据分析结果

c. 原始数据统计结果

小说	文件大小(字节)	字符数	占比(%)
青铜葵花	276,554	138,467	16.8
草房子	319,811	160,738	19.5
根鸟	226,895	113,718	13.8
细米	130,954	65,534	8.0
红瓦黑瓦	663,736	331,912	40.3

总计	1,617,950	810,369	100.0

从统计数据可以看出，《红瓦黑瓦》是篇幅最长的作品，占整个数据集的 40.3%，这可能会在检索结果中产生一定的偏向性。《草房子》和《青铜葵花》的篇幅相对较为均衡，分别占 19.5%和 16.8%。《细米》是篇幅最短的作品，仅占 8.0%，这种不均衡的分布为测试检索算法在不同文档长度下的表现提供了有价值的实验条件。

数据集的总字符数超过 81 万，为构建有效的中文检索系统提供了充足的文本量。这个规模既能够产生丰富的词汇分布，又便于在实验环境中进行快速的索引构建和查询处理。

3. 结巴分词处理与统计分析

> 参考
>
> Python 第三方库 jieba(中文分词)入门与进阶(官方文档) - 交流_QQ_2240410488 - 博客园
>
> 【结巴分词资料汇编】结巴中文分词源码分析(2) - 伏草惟存 - 博客园
>
> https://zhuanlan.zhihu.com/p/701661969
>
> python 中文分词工具:结巴分词 jieba_jieba 参数-CSDN 博客

> https://zhuanlan.zhihu.com/p/502691059
>
> https://zhuanlan.zhihu.com/p/701661969

- 分词处理步骤：

  a. 文本预处理阶段：读取原始文本文件，处理编码问题，确保文本能够正确解析。原始文件采用 GBK 编码，在读取时指定了正确的编码格式，避免了字符乱码问题。

  b. 分词执行阶段：使用 jieba.cut()函数对每部小说的完整文本进行分词处理。该函数返回一个生成器，将其转换为列表以便后续处理和统计。分词过程中，结巴算法会自动识别词汇边界，将连续的汉字序列切分为有意义的词汇单元。

  c. 后处理阶段：对分词结果进行清理和标准化处理，包括去除单字符词汇（主要是标点符号和助词）和去除空白字符等。

运行 `jieba_segmentation.py`

得到详细的分词统计数据

小说	原始字符数	分词数量	分词密度
青铜葵花	138,467	95,694	0.691
草房子	160,738	110,904	0.690
根鸟	113,718	77,735	0.684
细米	65,534	45,746	0.698
红瓦黑	331,912	230,239	0.694

瓦			
总计	810,369	560,318	0.692

分词密度在各部小说中保持相对稳定，平均为 0.692，这表明结巴分词算法在处理不同作品时具有良好的一致性。这个比例意味着平均每 1.45 个汉字构成一个词汇单元，符合现代汉语的一般语言规律。

- 此外
  - 词汇总量与多样性：经过去重和过滤处理后，整个数据集共产生了 29,218 个不同的词汇，总词数为 219,794 个。这个词汇量体现了曹文轩作品的语言丰富性，同时也为构建有效的检索索引提供了充足的特征维度。
  - 高频词汇分析：词频统计显示，出现频率最高的 20 个词汇反映了小说的主要内容特征。其中，"没有"（1,739 次）、"一个"（1,662 次）等功能词占据了高频位置，这符合中文文本的一般规律。主要人物名称如"根鸟"(1,478 次）、"桑桑"（1,116 次）、"葵花"（1,065 次）、"青铜"（1,007 次）、"细米"（648 次）等在高频词汇中占据重要位置。
  - 平均词长分析：统计结果显示，平均词长为 2.14 个字符，这个数值符合现代汉语词汇的一般特征。大部分中文

词汇由 2-3 个汉字组成，这个统计结果验证了分词算法的有效性。

- 专有名词识别效果：通过抽样检查，我们发现结巴分词在识别小说中的人名、地名等专有名词方面表现良好。例如，"油麻地"、"大麦地"等地名，"马水清"、"梅纹"等人名都得到了正确的识别和切分。

4. 稀疏检索算法设计与实现

`sparse_retrieval_system.py`

1. 稀疏检索系统架构

> 参考
>
> 打造高效全文检索系统:Python 全文检索框架实例_python 全文检索-CSDN 博客
>
> https://cloud.tencent.com/developer/article/1757069

```
Python
class SparseRetrievalSystem:
    def __init__(self):
        self.documents = {}
```

```
        self.inverted_index = defaultdict(dict)
        self.doc_lengths = {}
        self.doc_count = 0
        self.term_doc_freq = Counter()
```

- 文档存储模块：管理原始文档内容，为每个文档分配唯一标识符，并维护文档元数据信息。每部小说被视为一个独立的文档，使用文件名作为文档标识符。
- 倒排索引模块：构建词汇到文档的映射关系，记录每个词汇在各个文档中的出现频率。倒排索引的数据结构为嵌套字典，外层键为词汇，内层键为文档标识符，值为该词汇在对应文档中的词频。
- 统计信息模块：维护全局统计信息，包括文档总数、词汇表大小、文档长度分布、词汇文档频率等。
- 查询处理模块：处理用户查询，包括查询分词、相关性计算、结果排序等功能。支持多种相关性计算算法的切换和参数调整。

2. TF-IDF 算法实现

> 参考
>
> 来，带你从 TF-IDF 说起搞懂 BM25_tfidf bm25-CSDN 博客
>
> 1.14 - 信息检索:TF-IDF/BM25，原理+代码 - 橘子葡萄火

> 龙果 - 博客

```
Python
def calculate_tf_idf(self, term, doc_id):
    if term not in self.inverted_index or doc_id not
in self.inverted_index[term]:
        return 0.0
    tf = self.inverted_index[term][doc_id]
    df = self.term_doc_freq[term]
    tf_score = tf / self.doc_lengths[doc_id]
    idf_score = math.log(self.doc_count / (df + 1))
    return tf_score * idf_score
```

- 词频（TF）计算：采用归一化的词频计算方法，公式为 `TF(t,d) = f(t,d) / |d|`，其中 `f(t,d)` 表示词汇 t 在文档 d 中的出现次数，`|d|` 表示文档 d 的总词数。
- 逆文档频率（IDF）计算：采用标准的 IDF 计算公式 `IDF(t) = log(N / df(t))`，其中 N 表示文档总数，`df(t)` 表示包含词汇 t 的文档数量。IDF 值反映了词汇的区分能力，出现在较少文档中的词汇具有更高的区分价值。
- TF-IDF 分数计算：最终的 TF-IDF 分数通过 TF 和 IDF 的乘积获得，即 `TF-IDF(t,d) = TF(t,d) × IDF(t)`。对于多

词查询，文档的总相关性分数为查询中所有词汇的 TF-IDF 分数之和。

3. BM25 算法实现

> 参考
>
> 深入理解 TF-IDF、BM25 算法与 BM25 变种：揭秘信息检索的核心原理与应用-CSDN 博客
>
> 从 TF-IDF 到 BM25，BM25+，一文彻底理解文本相关度 - JadePeng - 博客园

```Python
def calculate_bm25(self, term, doc_id, k1=1.2, b=0.75):
    if term not in self.inverted_index or doc_id not in self.inverted_index[term]:
        return 0.0
    tf = self.inverted_index[term][doc_id]
    df = self.term_doc_freq[term]
    doc_len = self.doc_lengths[doc_id]
    avg_doc_len = sum(self.doc_lengths.values()) / len(self.doc_lengths)
```

```
    idf = math.log((self.doc_count - df + 0.5) / (df + 0.5) + 1)
    tf_component = (tf * (k1 + 1)) / (tf + k1 * (1 - b + b * (doc_len / avg_doc_len)))
    return idf * tf_component
```

- 饱和函数设计：BM25 引入了饱和函数来处理词频的非线性增长效应。其 TF 组件的计算公式为：`TF_component = (f(t,d) × (k1 + 1)) / (f(t,d) + k1 × (1 - b + b × (|d| / avgdl)))`，其中 k1 控制词频饱和度，b 控制文档长度归一化的程度，avgdl 为平均文档长度。
- 参数优化：采用经典的参数设置：k1=1.2，b=0.75。这些参数在大量实验中被证明能够在多数情况下提供良好的检索性能。k1=1.2 意味着词频的影响会在较高频率时趋于饱和，b=0.75 表示文档长度归一化起到重要但不是决定性的作用。
- IDF 改进：BM25 采用了改进的 IDF 计算公式：`IDF(t) = log((N - df(t) + 0.5) / (df(t) + 0.5))`，这种计算方式能够更好地处理低频词汇，避免了传统 IDF 在处理稀有词汇时可能出现的数值不稳定问题。

4. 索引构建

```
Python
```

```
def add_document(self, doc_id, content):
    self.documents[doc_id] = content
    words = list(jieba.cut(content))
    words = [word.lower().strip() for word in words
if len(word.strip()) > 1]
    word_freq = Counter(words)
    self.doc_lengths[doc_id] = len(words)
    for term, tf in word_freq.items():
        self.inverted_index[term][doc_id] = tf
    unique_terms = set(words)
    for term in unique_terms:
        self.term_doc_freq[term] += 1
    self.doc_count += 1
```

5. 查询处理

```
Python
def search(self, query, algorithm='tf_idf',
top_k=5):
    query_terms = list(jieba.cut(query))
    query_terms = [term.lower().strip() for term in
query_terms if len(term.strip()) > 1]
    if not query_terms:
```

```
        return []
    doc_scores = defaultdict(float)
    for term in query_terms:
        if term in self.inverted_index:
            for doc_id in
self.inverted_index[term]:
                if algorithm == 'tf_idf':
                    score =
self.calculate_tf_idf(term, doc_id)
                elif algorithm == 'bm25':
                    score =
self.calculate_bm25(term, doc_id)
                else:
                    score =
self.inverted_index[term][doc_id]
                doc_scores[doc_id] += score
    sorted_docs = sorted(doc_scores.items(),
key=lambda x: x[1], reverse=True)
    return sorted_docs[:top_k]
```

6. 系统初始化+示例查询

`system_initialization.py`

4 个人在本项目的收获

技术上，从解决 GBK 编码乱码到优化结巴分词后处理，掌握了中文文本预处理关键环节;通过实现 TF-IDF 与 BM25 算法，理解了稀疏检索核心逻辑，还能对比算法差异；理论认知上，倒排索引、文档长度归一化等概念从抽象变具象，也意识到数据集特性对系统设计的影响。 同时，我提升了问题排查能力与模块化编码思维，更建立“数据驱动优化”意识，为后续技术实践打下坚实基础。