北京郵電大學

作业 2：
信息检索系统实验报告

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：2022211305

学号：2022211683

姓名：张晨阳

2025 年 6 月 6 号

目录

1. 系统概览 1
2. 爬虫工作 3
   2.1. ZENODO 回应包 3
   2.2. 设计数据结构 4
   2.3. 爬虫设计 5
3. 文本预处理与清洗 7
4. 建立索引结构 9
   4.1. 倒排索引的基本思想 9
   4.2. 构建 TF-IDF 语义模型 9
   4.3. 构建稀疏文档向量表示 11
   4.4. 索引与模型存储 11
5. 搜索与匹配算法 12
   5.1. 查询预处理与表示 12
   5.2. 向量空间检索与相似度计算 12
   5.3. 匹配关键词与上下文摘要提取 13
   5.4. 人工反馈与查询微调机制 13
   5.5. 检索返回结构 14
6. 人机交互界面设计 15
   6.1. 前端功能设计与布局 15
   6.2. 后端接口与逻辑实现 15
   6.3. 图像查询扩展设计（OCR） 16
7. 系统优化与创新功能 17
   7.1. 人工评价反馈与查询自适应机制 17
   7.2. 多媒体信息检索（图像 OCR 支持） 17
   7.3. 字段分权建模与权重调节 17
   7.4. 匹配关键词上下文展示 18
8. 总结与体会 19

1. 系统概览

本项目旨在设计并实现一个面向英文学术文献的信息检索系统。

我们通过自建爬虫从 ZENODO 网站爬取了 500 篇英文科研论文的元数据信息。最终保留了 496 篇质量较高的文档用于后续处理与检索。

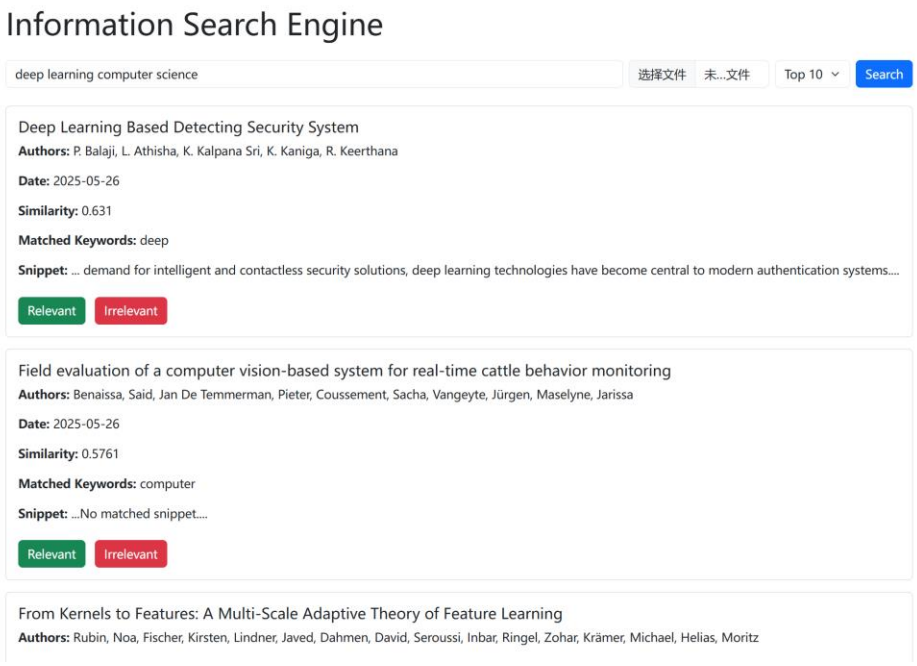
系统实现了以下核心功能：

- **文本预处理与规范化：**包括 HTML 清洗、分词、词形还原（lemmatization）、停用词过滤、自定义冗余词处理等，保证索引质量。
- **倒排索引构建：**基于文档的标题、关键词与摘要信息构建稀疏倒排索引表，提升搜索效率与关键词定位能力。
- **向量空间模型：**采用 TfidfVectorizer 将每篇文档表示为向量，结合余弦相似度（cosine similarity）进行文档与查询之间的匹配评分。
- **用户评分反馈机制：**在搜索结果展示后，用户可以对每一条结果进行手动评价（relevant / irrelevant），系统记录用户反馈以用于后续查询优化。
- **界面设计与交互功能：**基于 Flask + HTML + Bootstrap 技术栈开发简洁的前端界面，用户可以通过输入查询、上传图片（OCR 识别）、评分反馈等交互形式使用系统。

为提升信息检索效果与用户体验，系统还实现了以下扩展功能：

- **关键词加权机制：**在文档向量构建过程中，分别对标题、关键词、摘要字段设置不同权重，实现更精细化的语义控制。
- **上下文摘要片段提取：**在搜索结果中标注并高亮匹配关键词周围的上下文片段，提升结果可读性与解释性。
- **人工评分驱动的向量微调机制：**系统记录用户的 relevant / irrelevant 评价结果，并通过调节查询词向量的方式动态优化后续搜索性能。
- **图片搜索支持（OCR）：**用户可以上传图片，系统使用 OCR 技术自动提取图片中的文本，并作为查询内容发起搜索，实现多模态信息检索能力。

系统主要使用界面如下：


Information Search Engine
deep learning computer science
选择文件
未...文件
Top 10
Search
Deep Learning Based Detecting Security System
Authors: P. Balaji, L. Athisha, K. Kalpana Sri, K. Kaniga, R. Keerthana
Date: 2025-05-26
Similarity: 0.631
Matched Keywords: deep
Snippet: ... demand for intelligent and contactless security solutions, deep learning technologies have become central to modern authentication systems....
Relevant
Irrelevant
Field evaluation of a computer vision-based system for real-time cattle behavior monitoring
Authors: Benaissa, Said, Jan De Temmerman, Pieter, Coussement, Sacha, Vangeyte, Jürgen, Maselyne, Jarissa
Date: 2025-05-26
Similarity: 0.5761
Matched Keywords: computer
Snippet: ...No matched snippet....
Relevant
Irrelevant
From Kernels to Features: A Multi-Scale Adaptive Theory of Feature Learning
Authors: Rubin, Noa, Fischer, Kirsten, Lindner, Javed, Dahmen, David, Seroussi, Inbar, Ringel, Zohar, Krämer, Michael, Helias, Moritz

2. 爬虫工作

本系统的元数据来自于 ZENODO 的 500 篇最近上传的文章，保证覆盖的领域足够全面，重合度够低。采用 ZENODO 提供的 API 进行文章获取，实现过程的关键步骤如下：

2.1. ZENODO 回应包

通过查阅 ZENODO 提供的 API 文档，我们了解到其接受的请求格式与返回的包的格式分别为：

- **请求格式：**

```
base_url = "https://zenodo.org/api/records"
params = {
    "q": search_query,
    "size": max_results,
    "sort": sort_by,
}
```

其中网站提供的基础 API 为 https://zenodo.org/api，我们使用的/records 表示访问公开的记录，对于这个 url，我们提供了以下参数进行获取限制：

1. q：该参数表示提供的搜索关键词
2. size：该参数表示进行搜索的返回的最大记录数限制
3. sort：该参数表示返回记录的排序限制

- **返回格式：**

```
{
    "conceptrecid": "542200",
    "created": "2020-05-19T11:58:41.606998+00:00",
    "files": [],
    "id": 542201,
    "links": {
        "bucket": "https://zenodo.org/api/files/568377dd-daf8-4235-85e1-
a56011ad454b",
        "discard":
"https://zenodo.org/api/deposit/depositions/542201/actions/discard",
        "edit":
"https://zenodo.org/api/deposit/depositions/542201/actions/edit",
```

```
        "files":
"https://zenodo.org/api/deposit/depositions/542201/files",
        "html": "https://zenodo.org/deposit/542201",
        "latest_draft":
"https://zenodo.org/api/deposit/depositions/542201",
        "latest_draft_html": "https://zenodo.org/deposit/542201",
        "publish":
"https://zenodo.org/api/deposit/depositions/542201/actions/publish",
        "self": "https://zenodo.org/api/deposit/depositions/542201"
    },
    "metadata": {
        "prereserve_doi": {
            "doi": "10.5072/zenodo.542201",
            "recid": 542201
        }
    },
    "modified": "2020-05-19T11:58:41.607012+00:00",
    "owner": 12345,
    "record_id": 542201,
    "state": "unsubmitted",
    "submitted": false,
    "title": ""
}
```


其中 metadata 字段保存了我们需要的一些重要信息，例如关键字、编号、摘要原文、标题等，也是我们进行检索所需要的信息。

2.2. 设计数据结构

对于一篇摘要，该程序需要访问它的如下信息：

- 标题
- 作者
- 记录编号
- 关键字
- 发布日期
- 摘要原文
- 文件（可选）

据此，我们定义了如下数据结构进行数据记录：

```
class ZenodoRecord:
    def __init__(self, record_id, title, authors, keywords,
publication_date, description, files):
        self.record_id = record_id
        self.title = title
        self.authors = authors
        self.keywords = keywords
        self.publication_date = publication_date
        self.description = description
        self.files = files
```

2.3. 爬虫设计

由于我们使用 api 进行文章获取，故我们只需要利用 api 提出请求，并根据返回的 response 中的内容进行信息分析即可。

- **提取函数**

观察 response 包，我们可以发现我们需要的内容均在 metadata 与 files 中，鉴于我们需要的是摘要信息，我们只对 metadata 中的摘要信息进行提取，提取函数设计如下：

```
 1. # 提取元数据
 2. title = record.get('metadata', {}).get('title', 'N/A')
 3. authors = [author.get('name', 'N/A') for author in
record.get('metadata', {}).get('creators', [])]
 4. keywords = record.get('metadata', {}).get('keywords', [])
 5. publication_date = record.get('metadata',
{}).get('publication_date', 'N/A')
 6. description = record.get('metadata', {}).get('description', 'N/A')
 7. files = record.get('files', [])
 8. # 创建 ZenodoRecord 对象
 9. zenodo_record = ZenodoRecord(
10.     record_id=record_id,
11.     title=title,
12.     authors=authors,
13.     keywords=keywords,
14.     publication_date=publication_date,
15.     description=description,
16.     files=files
17. )
18. # 保存元数据
```

```
base_filename = os.path.join(output_dir, str(record_id))
save_metadata_to_txt(zenodo_record, f"{base_filename}.txt")
```

- **保存函数**

我们将提取到的概要信息保存至记录 id 同名的 txt 文件中，保存格式如下所示：

```
Title: Code for large aperture scintillometry source area processing: scintools
Authors: Saunders, Beth, Morrison, William, Grimmond, Sue, Hertwig, Denise, Lean, Humphrey, Bohnenstengel, Sylvia
Keywords: Large aperture scintillometry, source area
Publication date: 2024-08-02
Description:
Code used to derive source areas for large aperture scintillometer observations.
This upload corresponds to version 0.2 of the scintools GitHub repository: <a href="https://github.com/Urban-Meteorology-Reading/scintools">https://github.com/Urban-Meteorology-Reading/scintools</a>
This code is used in Saunders et. al 2024: “Methodology to evaluate numerical weather predictions using large aperture scintillometry sensible heat fluxes: demonstration in London”, DOI: <a title="https://doi.org/10.1002/qj.4837" href="https://doi.org/10.1002/qj.4837" target="_blank" rel="noopener">https://doi.org/10.1002/qj.4837</a>
Also included is ‘digital_surface_models_London.zip’, containing digital surface models of the area considered in Saunders et al. 2024: Building height (height_surface_4m.tif), terrain height (height_terrain_4m.tif) and vegetation height (height_veg_4m.tif). Classification of the land-cover over the area is also included (LandUseMM_7classes_32631.tif).
```

3. 文本预处理与清洗

为保证后续信息检索系统的准确性与效率，我们对原始从 ZENODO 爬取的文献文本数据进行了系统化的预处理与清洗操作。本阶段的目标是将格式混乱、语义不统一的原始文本转换为结构化、可向量化的标准形式，为倒排索引构建和向量建模提供可靠基础。

预处理主要包含以下步骤：

（1）字段提取与格式规整

通过正则表达式 (re) 对 .txt 文件进行字段提取，若标题或摘要字段缺失或为 N/A 则直接跳过该文档，以确保后续文本处理具备有效语义基础。

（2）HTML 标签清除

Description 字段中常包含 HTML 格式标签，因此使用 BeautifulSoup 对其进行 HTML 清洗，仅保留纯文本内容。

```
soup = BeautifulSoup(raw_html, "html.parser")
text = soup.get_text(separator=" ", strip=True)
```

（3）大小写统一 + 分词与词形还原

统一将所有文本转为小写，并借助 spaCy 的 en_core_web_sm 模型对文本进行：

- 分词（Tokenization）
- 词形还原（Lemmatization）

例如：

```
“using optimizations and proposed models” → “use optimization propose model”
```

（4）去除停用词与冗余学术词

我们基于 NLTK 提供的英文停用词表（stopwords.words('english')）以及自定义的学术冗余词集合（如 using, result, paper, example, et al. 等）进行了双重过滤：

```
STOP_WORDS = set(stopwords.words('english'))

# 添加自定义停用词
CUSTOM_STOP_WORDS = {
    'using', 'use', 'used', 'based', 'of', 'result', 'results', 'in',
```

```
    'on', 'to', 'analysis', 'paper', 'also', 'shown', 'different',
'new',
    'and', 'the', 'example', 'provide', 'present', 'show', 'propose',
    'may', 'within', 'however', 'therefore', 'thus', 'among', 'etc',
'et', 'al',
    'including', 'respectively', 'could', 'would', 'one', 'two', 'three'
}

if token.lemma_.lower() not in STOP_WORDS and token.lemma_.lower() not
in CUSTOM_STOP_WORDS
```

这一步显著减少了语义无关或信息量低的词汇对后续建模造成的干扰。

（5）结构化输出保存

最终，我们将每篇文档提取并清洗得到的字段保存为 JSON 格式，字段包括：

- filename: 文档名
- title: 原始标题
- clean_title: 清洗后的标题（用于向量化）
- authors: 作者列表
- keywords: 原始关键词（暂未分词）
- pub_date: 发布时间
- raw_description: 原始摘要内容
- clean_description: 清洗后的摘要内容（用于索引与搜索）

共计处理完成 **496 篇文档**，所有文档均具有可用标题与摘要内容，清洗后保留了文本语义中的核心关键词，为后续的倒排索引构建、向量模型训练与搜索匹配提供了坚实基础。

4. 建立索引结构

为了实现高效的文档检索与相关性计算，我们在预处理文档的基础上构建了两个关键索引结构：

- **倒排索引（Inverted Index）**：用于支持关键词到文档的快速反向映射；
- **文档向量矩阵（Doc Vectors）**：用于进行基于向量空间模型的相似度计算。

整个过程主要包括 **TF-IDF 权重建模、关键词选择、倒排索引构建**以及**稀疏向量表示生成**四个步骤。

4.1. 倒排索引的基本思想

倒排索引是一种将“词项 → 文档列表”的映射结构，用于快速查询包含某个关键词的所有文档。相比正排索引（文档 → 词项列表），倒排索引大大降低了查询开销，特别适用于全文检索任务。

本系统中的倒排索引采用如下结构：

```
{
  "machine": {
    "10496129.txt": 0.128,
    "15517614.txt": 0.092
  },
  "learning": {
    "10496129.txt": 0.211,
    "15517614.txt": 0.133
  }
}
```

其中，键是关键词，值为包含该关键词的文档及其对应的 TF-IDF 权重（保留小数点后 6 位），用于后续相似度计算。

4.2. 构建 TF-IDF 语义模型

为了提升搜索的语义质量，仅使用高信息量的词项构建倒排索引。系统使用 sklearn.feature_extraction.text.TfidfVectorizer 模块对文档集合进行处理，计算每个词项的 TF-IDF（Term Frequency–Inverse Document Frequency）得分。

其核心公式为：

- **TF** (Term Frequency):

$$TF_{t,d} = \frac{f_{t,d}}{\sum_k f_{k,d}}$$

其中 $f_{t,d}$ 表示词项 t 在文档 d 中的出现次数。

- **DF** (Inverse Document Frequency):

$$\mathrm{IDF_t} = \log\left(\frac{\mathrm{N}}{1+\mathrm{n_t}}\right) + 1$$

其中 N 为文档总数，n_t 为包含词项 t 的文档数，加 1 避免分母为 0。

- **TF-IDF** 得分:

$$\mathrm{TFIDF_{t,d}} = \mathrm{TF_{t,d}} \times \mathrm{IDF_t}$$

系统选取 TF-IDF 总权重最高的前 800 个关键词作为核心关键词集，并使用它们构建倒排索引与向量表示。

为了更好地建模词汇在语料中的分布与重要性，在此基础上，我们引入了一个**创新设计**：对不同字段赋予不同权重，以强调标题与关键词的重要性。

字段加权策略：

字段	权重
标题 title	8.0
关键词 keywords	4.0
摘要 description	1.0

实现方式为：将每个字段重复若干次组合为统一的训练文本，传入 TF-IDF 模型中：

```
def apply_weighted_text(doc, w_title=8.0, w_keywords=4.0, w_description=1.0):
    def repeat_text(text, weight):
        repeat_count = int(weight)
        partial = int((weight - repeat_count) * 10)
        return (" ".join([text] * repeat_count)) + (" " + text) * (partial // 10)

    title_text = repeat_text(doc['clean_title'], w_title)
    keyword_text = repeat_text(doc['keywords'], w_keywords)
    desc_text = repeat_text(doc['clean_description'], w_description)
    return f"{title_text} {keyword_text} {desc_text}"
```

然后对所有文档组成的加权语料（corpus）统一建模：

```
vectorizer = TfidfVectorizer(max_df=0.9, min_df=2)
tfidf_matrix = vectorizer.fit_transform(corpus)
```

- max_df=0.9: 过滤在 90%以上文档中出现的常见词
- min_df=2: 过滤仅出现一次的低频词

4.3. 构建稀疏文档向量表示

在搜索匹配中，我们使用向量空间模型（VSM）进行相似度计算。为此，我们将每篇文档编码为一个固定长度的稀疏向量，每个维度对应一个选定关键词。

生成逻辑如下：

- 每个向量维度表示一个关键词的 TF-IDF 值；
- 未出现的关键词默认为 0；
- 结果存储为 doc_vectors.json，结构如下：

```
{

  "10496129.txt": [0.1, 0.0, 0.03, ..., 0.0],

  ...
}
```

4.4. 索引与模型存储

最终，索引结构与模型以如下格式持久化存储：

文件名	说明
keys.json	选定关键词列表（共 800 个）
inverted_index.json	倒排索引（关键词到文档）
doc_vectors.json	文档向量（用于计算相似度）
vectorizer.pkl	TF-IDF 模型（用于查询向量化）

5. 搜索与匹配算法

本信息检索系统采用基于向量空间模型（Vector Space Model, VSM）的搜索匹配算法，结合倒排索引与 TF-IDF 向量表示实现用户查询的高效检索。同时系统进一步引入了人工反馈调整机制，使得检索结果可随用户偏好不断优化。

5.1. 查询预处理与表示

用户查询输入首先经过以下处理步骤：

1. **HTML 清洗**：使用 BeautifulSoup 去除无关 HTML 标签；
2. **小写转换与词元化**：使用 spaCy 提取有效单词；
3. **词形还原（Lemmatization）**：统一不同形式词项；
4. **停用词剔除**：移除英文常用停用词和自定义冗余词汇。

例如，输入：

```
"Using machine learning methods in analysis of COVID results"
```

预处理后变为：

```
["machine", "learning", "method", "covid"]
```

预处理后的 token 被拼接成字符串后传入已有的 TfidfVectorizer 模型进行转换，得到原始查询向量：

```
raw_query_vector = vectorizer.transform(["machine learning method
covid"])
```

随后，我们将该查询向量映射到系统保留的 800 个关键词空间中，构成最终维度一致的稀疏查询向量：

$$\vec{q} = [q_1, q_2, \dots, q_{800}]$$

其中每一维 q_i 表示查询中第 i 个关键词的 TF-IDF 权重。

5.2. 向量空间检索与相似度计算

所有文档已在索引构建阶段转化为相同维度的文档向量：

$$\overrightarrow{d_j} = [d_{j1}, d_{j2}, \dots, d_{j800}]$$

系统通过计算查询向量与所有文档向量之间的余弦相似度（Cosine Similarity）进行相关性排名：

$$\text{sim}(\vec{q}, \vec{d_j}) = \frac{\vec{q} \cdot \vec{d_j}}{|\vec{q}| \cdot |\vec{d_j}|}$$

Python 实现：

```
sims = cosine_similarity([query_vector], doc_vectors_matrix)[0]
```

最终系统返回相似度排名前 Top-N 的文档，并展示标题、作者、发布日期、匹配关键词和内容片段。

5.3. 匹配关键词与上下文摘要提取

为了提升用户体验，系统通过倒排索引定位出查询中与文档匹配的关键词：

```
matched_words = [token for token in query_tokens if token in
inverted_index and doc_id in inverted_index[token]]
```

随后在原始摘要中搜索第一个命中关键词，并返回其前后 60~80 字符作为片段：

```
idx = desc.find(word)
snippet = desc[max(0, idx-60):idx+80]
```

该设计可直观向用户展示相关匹配上下文，增加检索结果的可解释性。

5.4. 人工反馈与查询微调机制

为提升搜索智能性，系统设计了**基于用户反馈的查询向量微调机制**。

用户在界面上对每个文档标注是否 relevant（相关），系统提取标题中的关键词进行标记，并保存在 feedback_keywords.json 中：

```
{
  "machine learning": {
    "positive": {"covid": 2, "diagnosis": 1},
    "negative": {"network": 1}
  }
}
```

在后续搜索时，如遇已记录的查询，系统将根据关键词得分调整查询向量：

- 若关键词曾被标记为“正向相关”，增加其在向量中的权重；
- 若为“负向无关”，则适当降低该词在查询向量中的影响。

数学形式如下：

$$q_i' = q_i + \alpha \cdot \text{pos}_i - \beta \cdot \text{neg}_i \text{ 且保证 } q_i' \geq 0$$

其中：

- q_i：原查询向量某维度；
- $\alpha = 0.3,\ \beta = 0.2$：经验权重系数；
- $\text{pos}_i, \text{neg}_i$：该词在反馈记录中的出现次数。

代码片段：

```
query_vector[idx] += 0.3 * count  # 正向增强
query_vector[idx] -= 0.2 * count  # 负向抑制
```

5.5. 检索返回结构

最终，系统返回 JSON 结构化的结果用于前端展示：

```
{
  "doc_id": "doc123.txt",
  "title": "Deep Learning for Diagnosis",
  "similarity": 0.81,
  "matched_keywords": ["deep", "learning"],
  "snippet": "... deep learning techniques applied to diagnosis ..."
}
```

该结构清晰地展示了匹配过程、关键词命中情况和文档相似度，支持用户快速决策和人工反馈。

6. 人机交互界面设计

本信息检索系统采用 Web 形式的人机交互界面，基于 Flask 框架实现后端接口，结合 HTML＋Bootstrap 搭建响应式用户前端界面，支持文本搜索、图像搜索、相关性评分反馈等多种交互功能。

6.1. 前端功能设计与布局

用户界面由以下模块组成：

- **搜索输入区域：**
  - 文本输入框：可输入任意自然语言查询；
  - 图片上传按钮：支持上传图片，系统会自动提取其中文字作为搜索内容；
  - Top-N 下拉选择框：允许用户选择返回的前 N 个相关文档；
  - 搜索按钮：触发查询请求。
- **搜索结果展示区域：**
  - 每条结果显示文档标题、作者、发布日期、相似度分数、匹配关键词与摘要片段；
  - 每条结果附带“Relevant”和“Irrelevant”按钮，供用户快速进行相关性标注。

该布局使用 Bootstrap 栅格与按钮组件，确保良好的交互体验与跨平台兼容性。

6.2. 后端接口与逻辑实现

系统主要接口如下：

/search：查询处理接口

前端通过 fetch() 以 multipart/form-data 格式提交表单，后端提取文本与图片内容：

```
query = request.form.get("query", "")
image_file = request.files.get("image", None)
```

若上传了图片，则使用 pytesseract OCR 工具提取其中文本并拼接进原始查询：

```
ocr_text = pytesseract.image_to_string(Image.open(temp_file.name))
query += " " + ocr_text
```

最终将综合查询传入检索函数 run_search()，返回相似度排名结果。

/feedback：人工评价反馈接口

每条结果都可通过点击按钮触发 JavaScript 回调函数，将当前标题与评分以 JSON 格式发送至后端：

```
fetch('/feedback', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ query, results: [{ title, relevant }] })
})
```

后端调用 save_feedback() 函数保存打分结果，并更新与该查询相关的关键词评分。

6.3. 图像查询扩展设计（OCR）

为扩展查询模式，系统支持将图像输入转化为搜索内容。该模块基于以下技术路径：

1. 使用 HTML <input type="file"> 接受用户上传图片；
2. Flask 后端接收图片后，调用 pytesseract 执行 OCR；
3. 提取出的文字自动拼接到原始文本查询中。

示例流程如下：

- 用户上传一张含有标题“Deep Learning in Diagnosis”的截图；
- 系统识别出文本并拼接到查询；
- 向量检索流程保持一致。

该功能特别适用于移动端拍照搜索、文献截图查询等实际场景，增强系统多模态能力。

7. 系统优化与创新功能

7.1. 人工评价反馈与查询自适应机制

本系统引入了用户反馈驱动的查询向量微调机制。用户在搜索结果中对文档进行“Relevant”或“Irrelevant”标注后，系统会自动分析其内容关键词，并据此记录“积极词汇”和“消极词汇”。

当系统再次遇到相似查询时，会根据历史反馈信息调整对应关键词在查询向量中的权重，从而动态优化相似度计算与文档排序，实现搜索体验的持续自适应改进。这是一种基于轻量监督信号的“弱反馈学习”思路，能够在无须重训模型的前提下改进结果质量。

7.2. 多媒体信息检索（图像 OCR 支持）

在传统文本搜索基础上，本系统增加了对**图像输入查询**的支持。用户可以上传一张含有文字的图像（如文献截图、手写标题等），系统自动提取其中的文字内容，融合进查询流程中，进而完成搜索。

这一功能实现**了图像→文字→搜索**的完整链路，具有广泛的实际应用场景，如：移动端拍照搜索、截图辅助检索、视觉辅助信息提取等，体现了多模态信息处理的初步能力。

7.3. 字段分权建模与权重调节

在构建文档向量表示时，系统引入了**字段加权机制（Field Weighting）**，即针对文档的标题、关键词与摘要等不同信息字段，给予不同的 TF-IDF 权重。

例如，标题字段被赋予更高权重，关键词字段为中等，摘要为低权重，以更好体现其在语义表达中的信息密度差异。

这种基于结构化信息的设计思路，使得搜索结果更符合用户真实意图，也提升了系统对不同文档重要性的识别能力。

7.4. 匹配关键词上下文展示

为了提高搜索结果的可读性与透明性，系统设计了关键词匹配上下文片段展示功能。在每条搜索结果中，系统会自动识别与查询词匹配的关键词，并在原始文档描述中提取其前后文段作为“摘要片段”展示。

用户不仅可以看到哪些关键词被命中，还能快速浏览其出现的语境，从而提升对搜索结果相关性的理解与信任。这一细节优化显著提升了用户对系统结果的感知质量。

8. 总结与体会

本项目旨在构建一个结构清晰、功能完善的信息检索系统，实现从文献爬取、文本处理、索引构建到查询匹配与前端交互的全流程闭环系统设计。整个开发过程涵盖了多项核心信息检索技术，并结合实践需求引入了多种优化机制，使系统在功能性、扩展性与用户体验等方面表现出良好的综合能力。

通过本项目，我深入理解了 TF-IDF 向量化、倒排索引、余弦相似度等经典检索模型的原理，并通过实际编码完成其全流程实现。在数据结构选择、权重设计、性能优化等方面的思考，使我对“理论如何落地”有了更加深刻的认识。

从数据采集到模型计算，再到接口设计与前端构建，项目全过程均需要合理架构与模块划分。我意识到，一个优秀的系统不仅要“能跑起来”，更要结构清晰、易于维护、便于拓展。因此在代码组织、路径管理与模块复用方面也积累了宝贵经验。

在后期的功能扩展中，我不断考虑“如何让系统更好用”。无论是加入 OCR 图像输入、人工反馈增强机制，还是优化关键词匹配展示，这些改进背后都体现了“用户中心”的设计思路。信息检索不只是算法，更是一个与人高度交互的智能系统。

本系统仍有诸多可扩展空间，如引入更强的语义模型（如 BERT）、改进排序学习机制、优化大规模并发处理等。但通过本次项目，我已经打下了坚实的工程与算法基础，并建立了独立完成一个智能系统的信心与方法论。