苏 州 大 学 实 验 报 告

院、系	计算机科学与技术	年级 专业	2020 软件工程	姓名	高歌	学号	2030416018
课程名称	信息检索综合实践					成绩	
指导教师	贡正仙	同组实验者	无	实验日期	2023 年 6 月 14 日		

实 验 名 称 　实验 8 信息检索综合实践

一. 实验目的

综合本学期学到的有关信息检索的全部知识，实现从爬取网页、解析、索引到查询的综合实践任务。

二. 实验内容

具体要求

1. 将南京大学计算机科学技术学院 NLP 组所有老师的信息保存到硬盘

   地址：http://nlp.nju.edu.cn/homepage/people.html

   可以用爬虫进行信息的原始收集，如果无法实现用爬虫，可以手工下载多个相关页面

2. 基于爬取的网页，做一个完整的检索系统

   采用前面讲过的网页正文提取，数据预处理，建立倒排索引，建立 TF/IDF 文档向量

   实现输入检索词（教师姓名/教学/论文/项目）后，能把教师相关的信息展现给查询者下载 en.txt，

加分项

- 细致的数据处理
- 除了 TF-IDF 的其它信息检索技术（词向量：word2vec　glove bert ）
- 对比，分析
- 使用爬虫
- 友好的用户界面
- 效率高
- ……

爬取基本方法和注意事项

- 以主页作为起点，根据其中包含的链接，递归爬取，直到没有新的页面可以爬到
- 要记录已经爬取的网页，避免重复爬取（死循环）
- 为了避免爬不是目标对象的网页，可以检查网址，例如南大网页都包含 nju.edu.cn
- 为了避免给机构的服务器太大的压力，同学们可以考虑爬 10 个网页就停止；爬的时候，间隔合适的时间（随机数几秒）。

三. 实验步骤和结果

注：代码使用 TypeScript 编写，运行时使用 ts-node。使用 Prettier 与 ESLint 作为代码格式化工具，代码风格遵从 TypeScript ESLint Recommended 标准。建立索引时使用了 JS 上的 NLP 库 Cmpromise 进行英文词形还原；数据预处理时使用了 LangChain.js 调用了 OpenAI 提供的 GPT-3.5 API 将中文翻译为英文。

（一）实验步骤

1. 本实验逻辑稍有些复杂，这里对目录结构做一个简单解释：

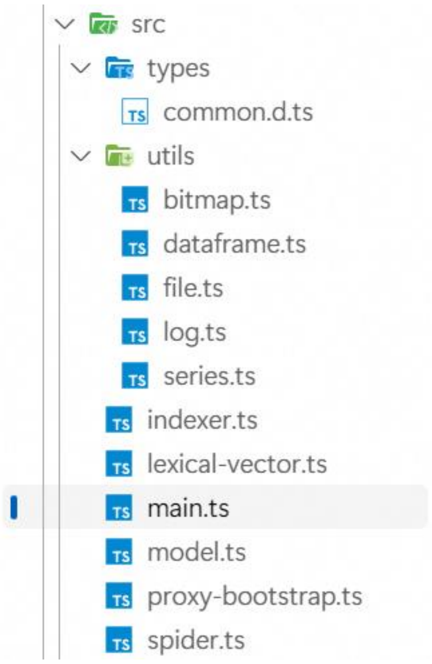


图 1 目录结构

其中，`types/`目录下包含了一些必要的类型定义，实际上仅包含了有关索引字典的类型定义；

`utils/`目录下包含了一些工具函数，`bitmap.ts` 包含了与位图相关的函数，用来辅助集合操作如去重等，`file.ts` 包含了一些与文件相关的工具函数，`log.ts` 包含了一些与日志相关的工具函数，而 `dataframe.ts` 包含了一个用于本次实验的 `DataFrame` 数据类型定义（类似于 Python 中 Pandas 的 DataFrame），`series.ts` 则供 `dataframe.ts` 使用，作为其内部数据的存储方式；

而 `indexer.ts` 包含了构建与读取索引文件所需的相关函数，`lexical-vector.ts` 中包含本次实验中与计算词汇向量有关的几个函数，`model.ts` 中包含了与调用 OpenAI API 并进行翻译的相关代码，`proxy-bootstrap.ts` 中包含了代理相关配置，`spider.ts` 中包含了与爬虫相关的代码；

`main.ts` 则为主入口文件。

其中，`utils/` 目录下的所有文件都与上次实验完全一致，而 `indexer.ts` 和 `lexical-vector.ts` 只做了一些微小的修改，会在之后说明。

接下来将一一解释剩余文件中的代码逻辑。

1. 先解释一下 `indexer.ts` 中的小改动。在上次实验中，数据是已经分好词的，因此直接用空格分隔就可以了。但本次实验获得的是原始数据，因此需要处理分词。

不过，实际上在 Compromise 库中这非常简单，只需要以下改动：

```
/**
 * Generate the index from corpus.
 * @param corpus The corpus.
 * @returns The index map.
 */
export const generateIndex = async (
  documents: string[],
  { lemmatize: doesLemmatize = true }: GenerateIndexOptions = {},
): Promise<IndexMap> => {
  const indexMap = new Map<string, Record<number, number[]>>();

  for (const [index, document] of documents.entries()) {
    const docID = index + 1;

    const words = (
      nlp(document)
        .terms()
        .not('#Value')
        .json()
        .map(
          (t: { terms: Array<{ normal: string }> }) => t.terms[0].normal,
        ) as string[]
    )
      .map(doesLemmatize ? lemmatize : R.identity)
      .filter((w) => /^[a-z-]+$/.test(w));

    ...
```

改动的部分用黄色标注了出来。可以看到只需要使用 Compromise 提供的 `nlp()`这个函数就可以得到句子中的全部词语了，这里还做了一下词形还原。

这便是 `indexer.ts` 中的全部改动了。

2. 然后介绍一下 `lexical-vector.ts` 中的一些改动。在这里只是将上一个实验中与计算相似度有关的函数稍加修改加入了这个文件中。该函数如下所示：

```
export const calculateSimilaritiesOf = (document: string) => ({
  on: (
    indexMap: SimplifiedIndexMap,
    df: SparseDataFrame<number>,
  ): Array<[number, number]> => {
    const words = (
      nlp(document)
        .terms()
        .not('#Value')
        .json()
```

```
        .map(
          (t: { terms: Array<{ normal: string }> }) => t.terms[0].normal,
        ) as string[]
    )
      .filter((w) => /^[a-z]+$/.test(w))
      .map(lemmatize);

    ...

    return df.columnNames
      .map((docID) => {
        const vec = df.col[docID]!;

        const vecLen = Math.sqrt(
          vec.accumulate((acc, value) => acc + value ** 2, 0),
        );

        const dotProd = Object.entries(searchVec).reduce(
          (acc, [index, value]) => acc + value * (vec.iat[Number(index)] ?? 0),
          0,
        );

        return [Number(docID), dotProd / (searchVecLen * vecLen)] as [
          number,
          number,
        ];
      })
      .filter(([, similarity]) => similarity > 0)
      .sort((a, b) => b[1] - a[1]);
  },
});
```

“稍作修改”指的是这里进行了按相似度从大到小的排序，并且过滤掉了相似度为 0 的情况。在上一次实验中数据量很大，因此几乎不会出现相似度为 0 的情况，因而疏忽了，这次补上。

3. 然后是与 OpenAI API 有关的一些配置。`proxy-bootstrap.ts` 比较简单就不展开了，只介绍一下 `model.ts` 中的内容。其全部内容如下：

```
import { OpenAI } from 'langchain/llms/openai';

// Bootstrap the proxy, as OpenAI cannot be accessed in China due to GFW.
import './proxy-bootstrap.js';

const OPENAI_API_KEY = '<your_openai_api_key>';

export const model = new OpenAI({
  openAIApiKey: OPENAI_API_KEY,
```

```
  temperature: 0,
  maxTokens: 1024,
});

export const translate = async (
  text: string,
  type: 'course-name' | 'project-name' | 'paper-title' = 'paper-title',
) => {
  const prompt = `将下面这个${
    type === 'course-name'
      ? '课程名称'
      : type === 'project-name'
      ? '项目名称'
      : '论文标题'
  }翻译成英文，不要包含任何多余字符`;

  const result = await model.call(`${prompt}: ${text}`);

  return result
    .replace(/[\u4e00-\u9fa5]/g, '')
    .replace(/\n+/g, ' ')
    .trim();
};
```

逻辑还是比较简单的，并不复杂。其导出了一个 `translate` 函数，传入一个待翻译的中文字符串，和一个可选的类型（课程名称、项目名称或论文标题），使用 GPT-3.5 将其翻译成英文。

由于 AI 返回的结果比较不可控，因此需要做一些处理工作，如过滤结果中多余的中文字符（比如，AI 经常返回“这是你需要的内容”作为前缀，但是我们不需要这个前缀）、过滤 AI 为了美观多加的空行、空格等。

4. 然后是最重要的 `spider.ts` 中的配置。

首先包含这么一个函数，用于获取主页中所有老师个人信息的 URL：

```
const HOME_URL = 'http://nlp.nju.edu.cn/homepage/people.html';

export const getTeacherURLs = async () => {
  const rawHTML = await fetch(HOME_URL).then((res) => res.text());
  const html = rawHTML
    .split('<!--teacher-->', 2)[1]
    .split('<!--PhD student-->', 2)[0];
  const $ = cheerio.load(html);
  return $('h4.pt15.mb10')
    .map((_, elem) => {
      const $elem = $(elem);
      const $a = $elem.find('a');
      const url = $a.attr('href')!;
```

```
      const name = $a.text().replaceAll(' ', '');
      return { name, url };
    })
    .toArray()
    .filter(({ url }) => url !== '#');
};
```

这里使用了 Node.js 上的 HTML 解析库 Cheerio，它提供了类似 jQuery 的语法用于筛选 HTML 信息，可以将其理解为 Python 中的 beautifulSoup.

然后，为获取每个老师的详细信息抽象了一个函数：

```
interface ExtractOptions {
  /**
   * The CSS selector used to split sections.
   */
  splitter: string;
  /**
   * Extract the title from the section texts.
   * @param $elem The title element.
   * @param index The index of the element.
   * @returns The title, or `false` to skip this section.
   */
  titleExtractor?: (
    $elem: cheerio.Cheerio<cheerio.AnyNode>,
    index: number,
  ) => string | false;
  /**
   * Process the content of the section.
   * @param title The title of the section.
   * @param $elem The content element.
   * @param splitter The splitter.
   * @param $ The cheerio instance.
   * @param currentURL The URL of the page.
   * @returns The processed content. If it is a promise, it will be awaited.
   */
  contentProcessor?: (
    title: string,
    $elem: cheerio.Cheerio<cheerio.AnyNode>,
    splitter: string,
    $: cheerio.CheerioAPI,
    currentURL: string,
  ) => string[] | Promise<string[]>;
}

/**
 * Extract the information from a teacher's page.
 *
```

```
 * @param url The URL of the page.
 * @param options The options.
 */
export const extractPageInfo = async (
  url: string,
  { contentProcessor, splitter, titleExtractor }: ExtractOptions,
) => {
  const html = await fetch(url).then((res) => res.text());
  const $ = cheerio.load(html);

  // Get texts after splitter, and convert to { "<title>": ["<text1>",
"<text2>", ...], ... }
  const info: Record<string, string[]> = {};
  const promises = $(splitter)
    .map(async (i, elem) => {
      const $elem = $(elem);
      const title = titleExtractor
        ? titleExtractor($elem, i)
        : $elem.text().trim();
      if (!title) return;
      const text = (
        contentProcessor
          ? await contentProcessor(title, $elem, splitter, $, url)
          : // Get all text before next splitter
            $elem.nextUntil(splitter).text().split('\n')
      )
        .map((line) => line.trim())
        .filter((line) => line);
      info[title] = text;
    })
    .toArray();
  await Promise.all(promises);
  return info;
};
```

这个函数虽然看起来很小，但功能比较复杂，这里详细解释一下。

首先，观察到所有老师的个人信息页都可按特定的分隔分为几个小节，如：

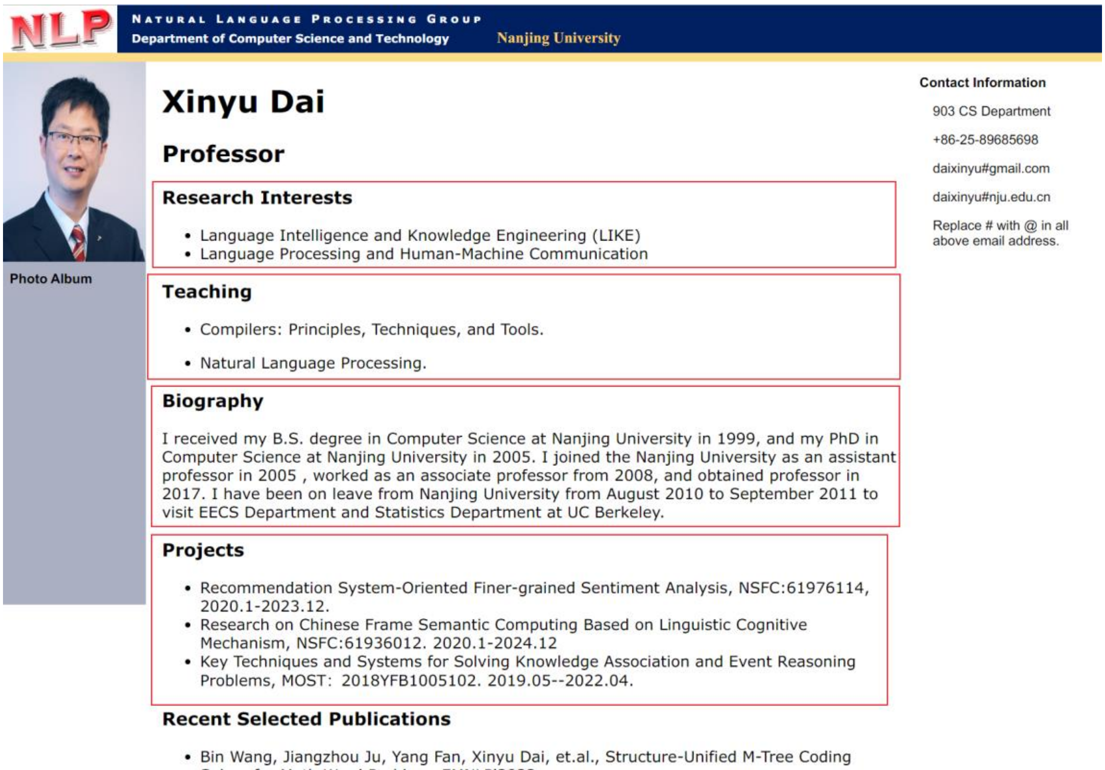


图 2 小节（section）示例

在这里，我将其称为一个个“Section”.

每个 Section 的标题都使用了同样格式的 HTML 元素，如对于这位老师，就是`<h3>`：

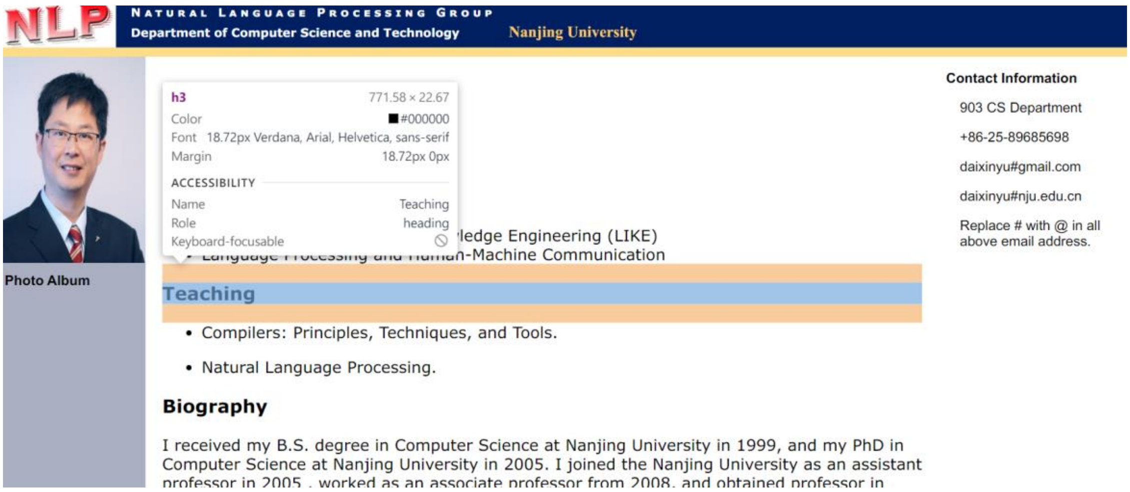


图 3 Section 标题

因此，可以使用一个 CSS Selector 来表示这种“分隔”，这就是 `ExtractOptions['splitter']`.

例如，我可以这样获取一个教师的个人信息，其中的内容需要按分割为若干 Section：

```
await extractPageInfo(url, { splitter: 'h3' })
```

然后我就可以获得类似这样的 JSON：

```
{
  "Teaching": [
    "Compilers: Principles, Techniques, and Tools.",
    "Natural Language Processing."
  ],
  "Projects": [
```

```
    "Recommendation System-Oriented Finer-grained Sentiment Analysis, NSFC:61976114, 2020.1-2023.12.",
    ...
  ],
  ...
}
```

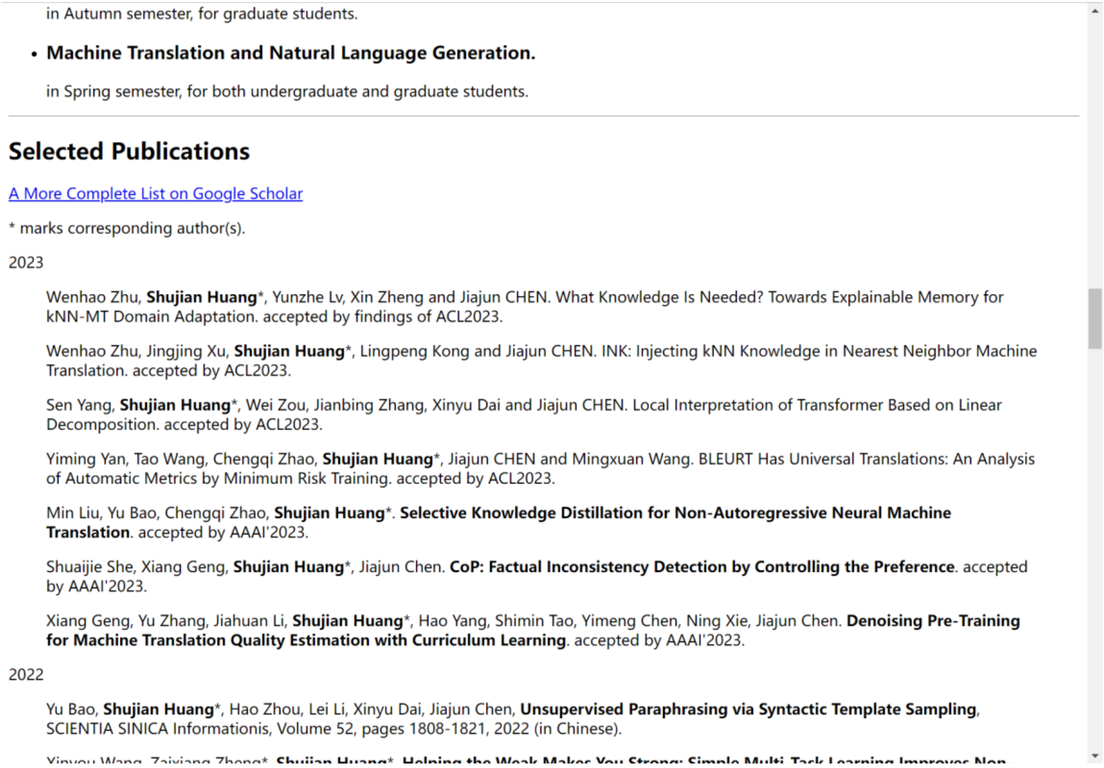
不过这并不完全是我们想要的。首先，我们希望标题（如这里的"Teaching"）能够符合统一的格式。例如有些老师的教学部分使用标题"Teaching"，有些又使用"Courses"，而有些中甚至包含链接等特殊字符，这显然是我们需要处理掉的，并且需要输出为统一的格式。

因此，允许传入一个函数 ExtractOptions['titleExtractor']，用于表示如何处理标题。例如，这是陈家骏老师的 titleExtractor：

```
titleExtractor: ($elem, i) => {
  if (i < 2) return false;
  const titleMap: Record<string, string | undefined> = {
    教学: 'teaching',
    研究项目: 'projects',
    发表论文: 'publications',
  };
  const rawTitle = $elem
    .find('strong')
    .text()
    .split('： ', 2)[0]
    .split(':', 2)[0]
    .trim();
  return titleMap[rawTitle] ?? rawTitle;
}
```

可以看到，除了简单将标题映射到统一的名称外（注意陈家骏老师的个人主页是中文的），这里还舍弃了最前面的两个 Section，并且还将标题中的冒号给去掉了。这样一个回调函数可以提供最大的自由度，便于复用一些复杂的逻辑。

然后，对于 Section 的内容，有时也不能简单地按换行符分隔。比如下面这位老师的论文 Section 中出现了链接和年份，这是我们不需要的：



in Autumn semester, for graduate students.

- **Machine Translation and Natural Language Generation.**

in Spring semester, for both undergraduate and graduate students.

Selected Publications

A More Complete List on Google Scholar

* marks corresponding author(s).

2023

Wenhao Zhu, **Shujian Huang***, Yunzhe Lv, Xin Zheng and Jiajun CHEN. What Knowledge Is Needed? Towards Explainable Memory for kNN-MT Domain Adaptation. accepted by findings of ACL2023.

Wenhao Zhu, Jingjing Xu, **Shujian Huang***, Lingpeng Kong and Jiajun CHEN. INK: Injecting kNN Knowledge in Nearest Neighbor Machine Translation. accepted by ACL2023.

Sen Yang, **Shujian Huang***, Wei Zou, Jianbing Zhang, Xinyu Dai and Jiajun CHEN. Local Interpretation of Transformer Based on Linear Decomposition. accepted by ACL2023.

Yiming Yan, Tao Wang, Chengqi Zhao, **Shujian Huang***, Jiajun CHEN and Mingxuan Wang. BLEURT Has Universal Translations: An Analysis of Automatic Metrics by Minimum Risk Training. accepted by ACL2023.

Min Liu, Yu Bao, Chengqi Zhao, **Shujian Huang***. **Selective Knowledge Distillation for Non-Autoregressive Neural Machine Translation**. accepted by AAAI'2023.

Shuaijie She, Xiang Geng, **Shujian Huang***, Jiajun Chen. **CoP: Factual Inconsistency Detection by Controlling the Preference**. accepted by AAAI'2023.

Xiang Geng, Yu Zhang, Jiahuan Li, **Shujian Huang***, Hao Yang, Shimin Tao, Yimeng Chen, Ning Xie, Jiajun Chen. **Denoising Pre-Training for Machine Translation Quality Estimation with Curriculum Learning**. accepted by AAAI'2023.

2022

Yu Bao, **Shujian Huang***, Hao Zhou, Lei Li, Xinyu Dai, Jiajun Chen, **Unsupervised Paraphrasing via Syntactic Template Sampling**, SCIENTIA SINICA Informationis, Volume 52, pages 1808-1821, 2022 (in Chinese).

图 4 不够整洁的 Section 内容

因此，也允许传入一个函数 `ExtractOptions['contentProcessor']`，用于表示如何处理其中的内容。这个函数需要返回一个字符串数组。例如，对于上面这位老师，其 `contentProcessor` 如下：

```
contentProcessor: (title, $elem, splitter, $) => {
  if (title === 'teaching')
    return $elem
      .next()
      .find('li > h3')
      .map((_, elem) => $(elem).text().trim().replace(/.$/, ''))
      .get();

  if (title === 'publications')
    return $elem
      .next()
      .next()
      .nextUntil('p:has(> a)')
      .text()
      .replace(/20[0-9][0-9]\n/g, '')
      .split('\n');

  return $elem.nextUntil(splitter).text().split('\n');
}
```

如上所示，在这里针对不同 Section 进行了不同的过滤。

另外，陈家骏老师的个人信息一部分包含在单独的连接中，这需要比较复杂的 `contentProcessor` 定义：


陈家骏

南京大学 计算机科学与技术系，教授，博士生导师
毕业于南京大学计算机软件专业，获学士、硕士和博士学位
电子邮箱：chenjj at nju dot edu dot cn
电话：025-89683672
办公地点：南京大学仙林校区计算机 科学与技术系大楼904

教学：

1. 程序设计（用C++实现 ）

- 讲稿1（基础篇）
- 讲稿2（高级篇）
- 教材：《程序设计教程——用C++语言编程》（第3版），机械工业出版社，2015年6月出版
  - 勘误
  - 例题源码

2. 自然语言处理

- 传统方法
- 统计方法

研究方向：

1. 自然语言处理
2. 软件工程，并发面向对象程序设计

研究项目：

1. 自然语言处理
2. 软件工程

发表论文：

1. 自然语言处理
2. 软件工程

图 5 外部链接

这也是为什么 `contentProcessor` 可以接受这么多参数——因为需要处理这样的外链情况。陈家骏老师的 `contentProcessor` 非常复杂，这里只给出其中一部分：

```
contentProcessor: async (title, $elem, splitter, $, currentURL) => {
  ...

  if (title === 'projects' || title === 'publications') {
    const links = $elem
      .next()
      .find(title === 'projects' ? 'li > p > a' : 'li > p > strong > a')
      .map((_, elem) => `${$(elem).text()}|${$(elem).attr('href')}`)
      .toArray()
      .map((t) => ({ name: t.split('|', 2)[0], href: t.split('|', 2)[1] }));
    // Add base URL to relative links
    for (let i = 0; i < links.length; i++) {
      const link = links[i];
      if (link.href.startsWith('http')) continue;
      links[i].href = new URL(link.href, currentURL).href;
    }

    const texts = [];
    for (const link of links) {
      const html = await fetch(link.href).then((res) => res.text());
      const $ = cheerio.load(html);

      if (title === 'projects') {
```

```
      if (link.name === '自然语言处理') {
        texts.push(
          ...$('table > tbody > tr')
            .map((_, elem) =>
              $(elem)
                .text()
                .trim()
                .replaceAll('\n', ',')
                .replace(/\s+/g, ''),
            )
            .get(),
        );
      } else {
        ...
```

可以看到，这里处理了获取外部链接信息的任务。

介绍完毕该函数的作用后，应该比较清晰了。然后为每个老师单独定义它们的这几个参数：

```
export const extractOptions: Record<string, ExtractOptions | undefined> = {
  陈家骏: {
    splitter:
      'p:has(> strong:has(> span)), p:has(> span:has(> b:has(> strong:has(> span))))',
    titleExtractor: ...,
    contentProcessor: ...,
  },

  戴新宇: {
    splitter: 'h3',
    titleExtractor: ...
```

类似这样就可以了。

这看起来似乎有些笨拙，但实际上已经是一个比较合适的粒度了。一个更为通用的、不需要单独处理每个网页的爬虫需要一些非常复杂的边界情况处理，或者可以使用目前现有的 LLM 进行处理——事实上，我也试验了这种方式，但效果不太理想，每次给出的结果差距过大，不太适合复用。

5. 最后在 `main.ts` 中编写主函数即可。

首先是一些常量及日志函数定义。和上次实验基本没差。

```
import fs from 'fs/promises';
import path from 'node:path';

import {
  generateIndex,
  loadIndex,
  saveIndex,
  transformIndexMap,
} from './indexer.js';
import {
```

```
  calculateSimilaritiesOf,
  generateTFIDFVectorMatrix,
} from './lexical-vector.js';
import { translate } from './model.js';
import { extractOptions, extractPageInfo, getTeacherURLs } from './spider.js';
import { fileExists, folderExists } from './utils/file.js';
import { logged } from './utils/log.js';

import type { SimplifiedIndexMap } from './types/common';
import type { SparseDataFrame } from './utils/dataframe';

const DATA_DIR = './data';
const DATA_NAME = 'data.json';
const DATA_JSON_PATHNAME = path.join(DATA_DIR, DATA_NAME);

type Keys = 'teaching' | 'projects' | 'publications';
const KEYS = ['teaching', 'projects', 'publications'] as const;

const loggedSaveJSON = logged({
  message: 'JSON saved',
  fn: (data: unknown) => ({
    toFile: async (pathname: string) =>
      await fs.writeFile(pathname, JSON.stringify(data, null, 2)),
  }),
  depth: 1,
});
const loggedReadCorpus = logged({
  message: 'Corpus read',
  fn: fs.readFile,
});
const loggedLoadIndex = logged({
  message: 'Index loaded',
  fn: loadIndex,
});
const loggedGenerateIndex = logged({
  message: 'Index generated',
  fn: generateIndex,
});
const loggedSaveIndex = logged({
  message: 'Index saved',
  fn: saveIndex,
  depth: 1,
});
const loggedTransformIndexMap = logged({
  message: 'Index map transformed',
```

```
  fn: transformIndexMap,
});
const loggedGenerateTFIDFVectorMatrix = logged({
  message: 'TF-IDF vector matrix generated',
  fn: generateTFIDFVectorMatrix,
});
const loggedReadDocuments = logged({
  message: 'Documents read',
  fn: async (pathname: string) => {
    const content = await fs.readFile(pathname, 'utf-8');
    return content.split('\n');
  },
});
```

然后编写函数获取教师信息，并保存和建立（读取）索引：

```
/**
 * Fetch teacher info from the Internet. (would translate Chinese to English using ChatGPT-3.5)
 * @returns The teacher infos.
 */
const fetchTeacherInfos = async () => {
  const teacherURLs = await getTeacherURLs();
  const teacherInfos: Array<{
    name: string;
    teaching: string[];
    projects: string[];
    publications: string[];
  }> = [];
  for (const { name, url } of teacherURLs) {
    const { projects, publications, teaching } = await logged({
      message: `Fetched info of "${name}"`,
      fn: async () =>
        await extractPageInfo(url, extractOptions[name] ?? { splitter: 'h3' }),
    })();
    teacherInfos.push({
      name,
      teaching: teaching ?? null,
      projects: projects ?? null,
      publications: publications ?? null,
    });
  }

  // Translate Chinese to English
  const translationTypeMap = {
    teaching: 'course-name',
    projects: 'project-name',
```

```
    publications: 'paper-title',
  } as const;

  await logged({
    message: 'Translated Chinese to English',
    fn: async () => {
      for (const info of teacherInfos) {
        for (const key of KEYS) {
          if (info[key] === null) continue;
          info[key] = await Promise.all(
            info[key].map(async (line) =>
              /[\u4e00-\u9fa5]/.test(line)
                ? await translate(line, translationTypeMap[key])
                : line,
            ),
          );
        }
      }
    },
  })();

  return teacherInfos;
};

/**
 * Get index map from cached file or generate it from corpus.
 * @param indexPathname The pathname of the cached index map file.
 * @param corpusPathname The pathname of the corpus file.
 * @returns The index map.
 */
const getIndexMap = async (indexPathname: string, corpusPathname: string) => {
  if (await fileExists(indexPathname))
    return await loggedLoadIndex(indexPathname, { simplified: true });

  const corpus = await loggedReadCorpus(corpusPathname, 'utf-8');
  const documents = corpus.split('\n');
  const indexMap = await loggedGenerateIndex(documents);
  await loggedSaveIndex(indexMap).toFile(indexPathname);
  return loggedTransformIndexMap(indexMap);
};
```

这两个函数比较简单，不多赘述，只是简单地组合一些函数。

然后在主函数中，首先爬取信息并翻译成英文，然后保存（若未缓存）。

```
const main = async () => {
```

```
  /* Fetch teacher info from the Internet if not exists */
  if (!(await fileExists(DATA_JSON_PATHNAME))) {
    const teacherInfos = await fetchTeacherInfos();
    // Save to JSON
    await loggedSaveJSON(teacherInfos).toFile(DATA_JSON_PATHNAME);
    // Save to text files
    await logged({
      message: 'Saved to text files',
      fn: async () => {
        for (const info of teacherInfos) {
          const folderPath = path.join(DATA_DIR, info.name);
          if (!(await folderExists(folderPath))) fs.mkdir(folderPath);

          for (const key of KEYS) {
            const filePath = path.join(folderPath, `${key}.txt`);
            if (info[key] === null) continue;
            await fs.writeFile(filePath, info[key].join('\n'));
          }
        }
      },
    })();
    console.log();
  }

  ...
}
```

这里需要注意下。这里涉及了本实验的一个核心思路。就是将**每个老师**的**每个类别**的信息都作为一个独立的文档集（如，“陈家骏”->"teaching"，表示该老师的教学内容）。每个文档集是一个.txt 文件，其中按行存储相应信息。搜索时，会搜索每个老师每个类别文档集中相似度最高的几个文档并展示出来。

这是具体的文件结构，这应该能更清晰地展示思路：

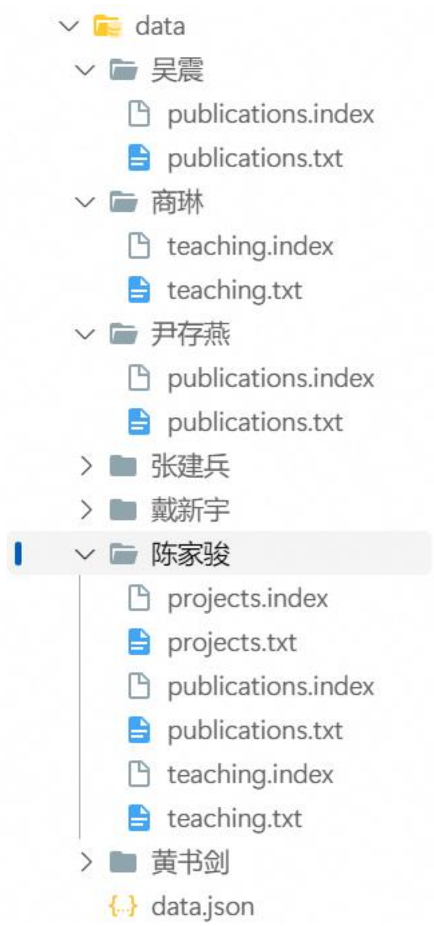


图 6 缓存的文档结构

然后，这是其中一个文档集：

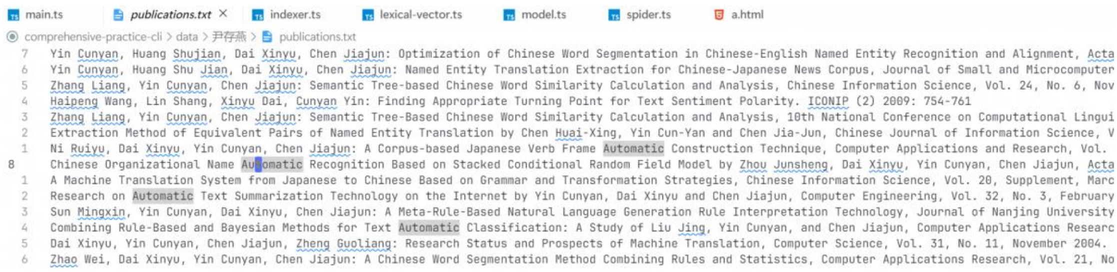


图 7 其中一个文档集

这样，搜索的时候就会匹配每个老师每个分类的内容，更好地展示结果。

然后是主函数的剩余部分，基本上就是按照这个思路写下去，没太多可以说的。主要思路已经在上一次实验中描述清楚了。

```
const main = async () => {
  ...

  /* Read teacher info from cached files, and generate TF-IDF vector matrix for similarity calculation */
  const indexMaps: Record<string, { [key in Keys]?: SimplifiedIndexMap }> = {};
  const tfidfMatrices: Record<
    string,
    { [key in Keys]?: SparseDataFrame<number> }
```

```
> = {};
const allDocuments: Record<string, { [key in Keys]?: string[] }> = {};

// Read indexMaps and tfidfMatrices from cached files
const folders = (await fs.readdir(DATA_DIR)).filter(
  (name) => name !== DATA_NAME,
);
for (const name of folders) {
  const filenames = await fs.readdir(path.join(DATA_DIR, name));
  for (const filename of filenames) {
    if (!filename.endsWith('.txt')) continue;

    const key = filename.split('.', 2)[0] as Keys;
    console.log(`Reading ${name}'s ${key}...`);

    const corpusPathname = path.join(DATA_DIR, name, filename);
    const indexPathname = path.join(DATA_DIR, name, `${key}.index`);

    const indexMap = await getIndexMap(indexPathname, corpusPathname);
    if (!indexMaps[name]) indexMaps[name] = {};
    indexMaps[name][key] = indexMap;

    const tfidfMatrix = loggedGenerateTFIDFVectorMatrix(indexMap);
    if (!tfidfMatrices[name]) tfidfMatrices[name] = {};
    tfidfMatrices[name][key] = tfidfMatrix;

    const documents = await loggedReadDocuments(corpusPathname);
    if (!allDocuments[name]) allDocuments[name] = {};
    allDocuments[name][key] = documents;
  }
  console.log();
}

/* Wait for user input and calculate similarities */
process.stdout.write('\nEnter a sentence to search: ');
process.stdin.on('data', async (data) => {
  for (const name of folders) {
    for (const key of KEYS) {
      if (!indexMaps[name][key]) continue;

      const indexMap = indexMaps[name][key]!;
      const tfidfMatrix = tfidfMatrices[name][key]!;
      const documents = allDocuments[name][key]!;

      const similarities = calculateSimilaritiesOf(data.toString().trim()).on(
```

```
          indexMap,
          tfidfMatrix,
        );

        const similaritiesToShow = similarities.slice(0, 5);
        if (similaritiesToShow.length === 0) continue;

        console.log(`\n${name}'s ${key}:`);
        similaritiesToShow.forEach(([docID, score], index) => {
          console.log(
            `${index + 1}. Similarity: ${score}\n` +
              `   Document ID: ${docID}\n` +
              `   Document: ${documents[docID - 1]}`,
          );
        });
      }
    }

    process.stdout.write('\nEnter a sentence to search: ');
  });
};
```

这部分代码很简单，其循环读取用户输入的待搜索文档，通过 `calculateSimilaritiesOf` 计算其与所有文档的相似度，从大到小排序后取每个老师的每个分类的前 5 个（若大于 5 个）分别打印相似度、文档 ID 及文档内容。

（二）实验结果

运行 `npm run dev`，使用 ts-node 执行 `./src/main.ts`。下面展示输出结果。

首先是第一次运行时爬取网页内容的日志：

```
Fetched info of "陈家骏" in 1526ms.
Fetched info of "戴新宇" in 728ms.
Fetched info of "黄书剑" in 335ms.
Fetched info of "尹存燕" in 426ms.
Fetched info of "张建兵" in 714ms.
Fetched info of "吴震" in 188ms.
Fetched info of "商琳" in 452ms.
Translated Chinese to English in 27206ms.
JSON saved in 3ms.
Saved to text files in 15ms.
```

图 8 第一次运行时爬取网页内容的日志

下为第二次及之后运行时的输出：

```
Reading 吴震's publications...
Index loaded in 2ms.
TF-IDF vector matrix generated in 3ms.
Documents read in 1ms.

Reading 商琳's teaching...
Index loaded in 0ms.
TF-IDF vector matrix generated in 1ms.
Documents read in 1ms.

Reading 尹存燕's publications...
Index loaded in 1ms.
TF-IDF vector matrix generated in 1ms.
Documents read in 1ms.

Reading 张建兵's publications...
Index loaded in 2ms.
TF-IDF vector matrix generated in 1ms.
Documents read in 0ms.
Reading 张建兵's teaching...
Index loaded in 1ms.
TF-IDF vector matrix generated in 0ms.
Documents read in 1ms.

Reading 戴新宇's projects...
Index loaded in 1ms.
TF-IDF vector matrix generated in 1ms.
Documents read in 1ms.
Reading 戴新宇's publications...
Index loaded in 6ms.
TF-IDF vector matrix generated in 6ms.
Documents read in 0ms.
Reading 戴新宇's teaching...
Index loaded in 1ms.
TF-IDF vector matrix generated in 0ms.
Documents read in 1ms.
```

```
Reading 陈家骏's projects...
Index loaded in 3ms.
TF-IDF vector matrix generated in 3ms.
Documents read in 1ms.
Reading 陈家骏's publications...
Index loaded in 25ms.
TF-IDF vector matrix generated in 48ms.
Documents read in 1ms.
Reading 陈家骏's teaching...
Index loaded in 1ms.
TF-IDF vector matrix generated in 2ms.
Documents read in 1ms.

Reading 黄书剑's publications...
Index loaded in 10ms.
TF-IDF vector matrix generated in 11ms.
Documents read in 1ms.
Reading 黄书剑's teaching...
Index loaded in 0ms.
TF-IDF vector matrix generated in 1ms.
Documents read in 1ms.


Enter a sentence to search: █
```

图 9 第二次及之后运行时的输出

然后展示几个示例，首先搜索“tag”：

```
Enter a sentence to search: tag

吴震's publications:
1. Similarity: 0.2999608685830437
   Document ID: 1
   Document: Grid Tagging Scheme for Aspect-oriented Fine-grained Opinion Extraction Zhen Wu, Chengcan Ying, Fei Zhao, Zhifang Fan, Xinyu Dai, Rui Xia EMNLP Findings, 202
0

戴新宇's publications:
1. Similarity: 0.2348259300415764
   Document ID: 10
   Document: Zhen Wu, Xinyu Dai, Rui Xia, Pairwise Tagging Framework for End-to-End Emotion-Cause Pair Extraction, FCS, 2023:17 (2), 1-10.
2. Similarity: 0.2338615834939517
   Document ID: 27
   Document: Zhen Wu, Chengcan Ying, et. al., Grid Tagging Scheme for Aspect-oriented Fine-grained Opinion Extraction, Findings of EMNLP’2020.
3. Similarity: 0.2310919668329996
   Document ID: 44
   Document: Di Shang, Xin-Yu Dai, Yi Li, Shujiang Huang, Jiajun Chen, Tagging Chinese Microblogger via Sparse Feature Selection, IJCNN'2016.

陈家骏's publications:
1. Similarity: 0.31061612597002897
   Document ID: 84
   Document: Feng-Yu Qiu, Wei-Yi Ge, Xin-Yu Dai. Code Recommendation with Natural Language Tags and Other Heterogeneous Data. accepted by CSAI'2017.
2. Similarity: 0.2684995388527889
   Document ID: 52
   Document: Zhen Wu, Chengcan Ying, Fei Zhao, Zhifang Fan, Xinyu Dai, Rui Xia. Grid Tagging Scheme for Aspect-oriented Fine-grained Opinion Extraction, Findings of EMNLP
’2020.

Enter a sentence to search: █
```

图 10 查询结果 1

可以看到，这里匹配到了吴震、戴新宇和陈家骏老师的论文，并展示了相似度和文档内容。结合文档内容观察，这个相似度比较合理。

然后搜索“artificial intelligence”：

```
Enter a sentence to search: artificial intelligence

商琳's teaching:
1. Similarity: 0.7453559924999299
   Document ID: 4
   Document: Artificial Intelligence (previous)
2. Similarity: 0.14766477605963393
   Document ID: 1
   Document: Computational Intelligence (for graduates)

张建兵's publications:
1. Similarity: 0.233072466082484
   Document ID: 6
   Document: Hong-Jian Xue, Xin-Yu Dai, Jianbing Zhang, Shujian Huang, Jiajun Chen. Deep Matrix Factorization Models for Recommender Systems. In: Proceedings of the 26th
International Joint Conference on Artificial Intelligence (IJCAI'17), Melbourne, Australia, 2017.
2. Similarity: 0.18886739009605508
   Document ID: 7
   Document: Xin-Yu Dai, Jian-Bing Zhang, Shu-Jian Huang, Jia-Jun Chen, and Zhi-Hua Zhou. Structured sparsity with group-graph regularization. In: Proceedings of the 29th
 AAAI Conference on Artificial Intelligence (AAAI'15), Austin, TX, 2015.
3. Similarity: 0.17005293205931318
   Document ID: 5
   Document: Jianbing Zhang, Yixin Sun, Shu-Jian Huang, Cam-Tu Nguyen, Xiaoliang Wang, Xin-Yu Dai, Jiajun Chen, and Yang Yu. AGRA: An analysis-generation-ranking framewor
k for automatic abbreviation from paper titles. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17), Melbourne, Australia, 20
17.

戴新宇's publications:
1. Similarity: 0.23880969660342333
   Document ID: 46
   Document: Deyu Zhou, Haiyang Xu, Xinyu Dai, Yulan He, Unsupervised Storyline Extraction from News Articles. Proceedings of the 24th International Joint Conference on A
rtificial Intelligence (IJCAI'16), 2016.

2. Similarity: 0.2181314137777403
   Document ID: 47
   Document: Shujian Huang, Huifeng Sun, Chengqi Zhao, Jinsong Su, Xin-Yu DAI, Jiajun Chen, Tree-state based Rule Selection Models for Hierarchical Phrase-based Machine T
ranslation. Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'16), 2016.
3. Similarity: 0.21615260557555788
   Document ID: 49
   Document: Xin-Yu Dai, Jian-Bing Zhang, Shu-Jian Huang, Jia-Jun Chen, and Zhi-Hua Zhou. Structured sparsity with group-graph regularization. In: Proceedings of the 29th
 AAAI Conference on Artificial Intelligence (AAAI'15), Austin, TX, 2015.
4. Similarity: 0.20451205215057497
   Document ID: 43
   Document: Hao Zhou, Yue Zhang, Chuan Chen, Shujian Huang, Xin-Yu Dai, and Jiajun Chen. A Neural Probabilistic Structured-Prediction Method for Transition-Based Natural
 Language Processing. Journal of Artificial Intelligence Research (JAIR), Volume 58, pages 703-729.
5. Similarity: 0.2030475238646253
   Document ID: 48
   Document: Guang-Neng Hu, Xin-Yu Dai, Yunya Song, Shu-Jian Huang, Jia-Jun Chen, A Synthetic Approach for Recommendation: Combining Ratings, Social Relations, and Review
s. Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'15), Buenos Aires, Argentina, 2015.

陈家骏's publications:
1. Similarity: 0.28589205006186774
   Document ID: 170
   Document: Attribute Reduction under Three-valued Decision Theory: Jiaxiu, Shanglin and Chen Jiajun, Progress in Chinese Artificial Intelligence: 2009.
2. Similarity: 0.285344425052182
   Document ID: 92
   Document: Hao Zhou, Yue Zhang, Chuan Chen, Shujian Huang, Xin-Yu Dai, and Jiajun Chen. A Neural Probabilistic Structured-Prediction Method for Transition-Based Natural
 Language Processing. Journal of Artificial Intelligence Research (JAIR), Volume 58, pages 703-729.
3. Similarity: 0.26584268955822377
   Document ID: 90
   Document: Hong-Jian Xue, Xin-Yu Dai, Jianbing Zhang, Shujian Huang, Jiajun Chen. Deep Matrix Factorization Models for Recommender Systems. International Joint Conferen
ce on Artificial Intelligence (IJCAI), pages 3203-3209, 2017.
4. Similarity: 0.2622707809275867
   Document ID: 82
   Document: Zhen Wu, Xin-Yu Dai, Cunyan Yin, Shujian Huang, and Jiajun Chen.  Improving review representations with user attention and product attention for sentiment cl
assification.  In Thirty-Second AAAI Conference on Artificial Intelligence, 2018. [paper]
5. Similarity: 0.26202222026057925


   Document ID: 117
   Document: Gao Lei, Dai Xinyu, Huang Shu-Jian, Chen Jia-Jun. Feature Selection and Point Mutual Information Pruning Based Product Attribute Extraction Method*. Pattern
Recognition and Artificial Intelligence, 2015, 28(2): 187-192.

黄书剑's teaching:
1. Similarity: 0.8031236297578404
   Document ID: 3
   Document: Programming for Artificial Intelligence

黄书剑's publications:
1. Similarity: 0.20362665409899847
   Document ID: 48
   Document: Hao Zhou, Yue Zhang*, Chuan Chen, Shujian Huang*, Xin-Yu Dai, and Jiajun Chen. A Neural Probabilistic Structured-Prediction Method for Transition-Based Natur
al Language Processing. Journal of Artificial Intelligence Research (JAIR), Volume 58, pages 703-729, 2017.
2. Similarity: 0.18081124929371237
   Document ID: 13
   Document: Dongqi Wang, Haoran Wei, Zhirui Zhang, Shujian Huang*, Jun Xie, Jiajun Chen. Non-Parametric Online Learning from Human Feedback for Neural Machine Translatio
n. In Proceedings of The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22), pages 11431-11439.
3. Similarity: 0.179030935917048
   Document ID: 29
   Document: Rongxiang Weng, Heng Yu, Shujian Huang*, Shanbo Cheng, Weihua Luo. Acquiring Knowledge from Pre-trained Model to Neural Machine Translation. The Thirty-Fourt
h AAAI Conference on Artificial Intelligence, pages 9266-9273, New York, 2020.
4. Similarity: 0.17731498598955217
   Document ID: 28
   Document: Zewei Sun, Shujian Huang*, Hao-Ran Wei, Xin-yu Dai, Jiajun Chen. Generating Diverse Translation by Manipulating Multi-Head Attention. The Thirty-Fourth AAAI
Conference on Artificial Intelligence, pages 8976-8983, New York, 2020.
5. Similarity: 0.1744519630915622
   Document ID: 20
   Document: Qu Cui, Shujian Huang*, Jiahuan Li, Xiang Geng, Zaixiang Zheng, Guoping Huang, Jiajun Chen. DirectQE: Direct Pretraining for Machine Translation Quality Esti
mation. In Proceedings of Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI2021), pages 12719-12727.

Enter a sentence to search: █
```

图 11 查询结果 2

这次就多得多了，毕竟这里的老师都是做 NLP 的。观察结果，也很符合预期。

四. 实验总结

1. 本次实验中，综合几次实验学习到的爬虫、分词、倒排索引、TF-IDF 相似度、查询等知识，完成了一次从网页中获取信息，并展示查询结果的综合实验。

2. 本次实验对爬虫稍微进行了一些抽象，通过分隔符、标题提取、内容处理几个回调函数统一了网页的处理方式。由于网页内容各不相同，因此暂时只能做到较细粒度的爬取，而难以更加通用。在

此过程中曾经尝试了借助 LLM 等工具（通过 LangChain.js），但是效果并不好，LLM 返回的内容每次差别过大，难以整合到一起，在之后需要的处理并不比手动处理更少。但是，如果要爬取网页的规模进一步增大，使用 LLM 构建一个通用的爬虫工具的确是一个很值得考虑的选择。当然，这也需要投入大量时间处理许多边界情况。

3. 本次实验中使用了 OpenAI 提供的 GPT-3.5 接口将中文翻译成英文。可以看到翻译质量确实要好于 Google 翻译、DeepL 等，这一部分是因为使用 LLM 翻译可以提供翻译的语境，比如“翻译论文标题”，这可以更精准地得到结果。

但是，LLM 的返回结果也往往包含很多噪声，需要做一些处理工作使返回的数据真正可用。而对于传统的翻译工具来说，返回的结果通常是直接可用的。

4. 本次实验的核心思路是将**每个老师**的**每个类别**信息都作为一个文档集，因此搜索时会展示每个老师每个类别中的对应搜索结果，这样更有利于快速找到需要的信息。