FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV-IR

Semester Project

Information Retrieval System

`Towards Data Science`

Daniel Schnurpfeil (dschnurp@students.zcu.cz)

May 2, 2023

Contents

1 Assignment

The aim of the semester project is to learn how to implement a complex IR system using ready-made pre-processing libraries. A by-product will be a deeper understanding of indexers, retrieval systems and lectures.

Basic functionality:

- indexing the data downloaded for the exercise (1st scored exercise Crawler) and any other data in the same format. Both sets of data can be indexed independently. ✔️
- querying from the GUI or CLI (command line interface) and when querying it is possible to select the index and the search model (vector space model vs. Boole model). The search results include the total number of documents that match the query. ✔️ 🤔
- pre-processing (stopwords remover, stemmer) ✔️
- creation of in-memory inverted index ✔️
- tf-idf model, cosine similarity ✔️
- search with logical operators AND ✔️
- documentation (programmer and user) ✔️

Improvements:

- File-based index ✔️
- later data indexing - adding new data to an existing index ❌
- sanitizing html tags ✔️
- detection of query language and indexed documents ❌
- search for phrases (even stop words) ❌
- search around the word ❌
- multiple scoring models ❌
- web content indexing - I enter a website, the program downloads the data and indexes it directly into the existing index ✔️
- further pre-processing of normalization ✔️
- GUI/web interface ✔️
- keyword suggestion ❌

- support for multiple document fields (e.g. date, from to) ✔️
- highlighting search text in the results preview ✔️
- custom implementation of query parsing without using an external library ✔️
- implementation of another model (use of semantic spaces, doc2vec, Transformers - BERT), etc ❌

2 Implementation

When the assignment is clear, I can begin with possible solution proposals for implementation and then describe implementation details. The first thing to choose is a website to index. I have chosen the website which contains articles about data science. This website is for English speaking audience and it is part of medium.com portal, so the IR system can be extended by more topics than just data scinece.

2.1 Used Programming Language

Due to personal preferences and by previous agreement there are two possible programming languages for implementation the IR system. First is Java and second is Python. I have chosen Python due to personal preferences to this programming language. The project implementation is suitable version 3.10 of Python interpreter. Usable dependencies are specified in the list bellow and were used for implementation.

- **numpy** (version 1.24.2) – library for working with N-dimensional vectors and operations among them
- **beautifulsoup4** (version 4.11.2), **lxml** (version 4.9.2) – used for extracting html tags
- **requests** (version 2.28.2) – library for acquiring content from web
- **nltk** (version 3.8.1) – pre-processing library used for stemming and stopwords
- **pandas** (version 1.5) – used for loading CSV documents
- **Django** (version 4.1.7) – web application framework

2.2 IR System Architecture

IR system is divided to several parts. The first part consists of scripts which are required to prepare indexed documents such as crawler or pre-processor. This part can be seen on the bottom of the Figure 1. The second part is web application based on Model View Controller (MVC) architecture. Both part are described in next sections.

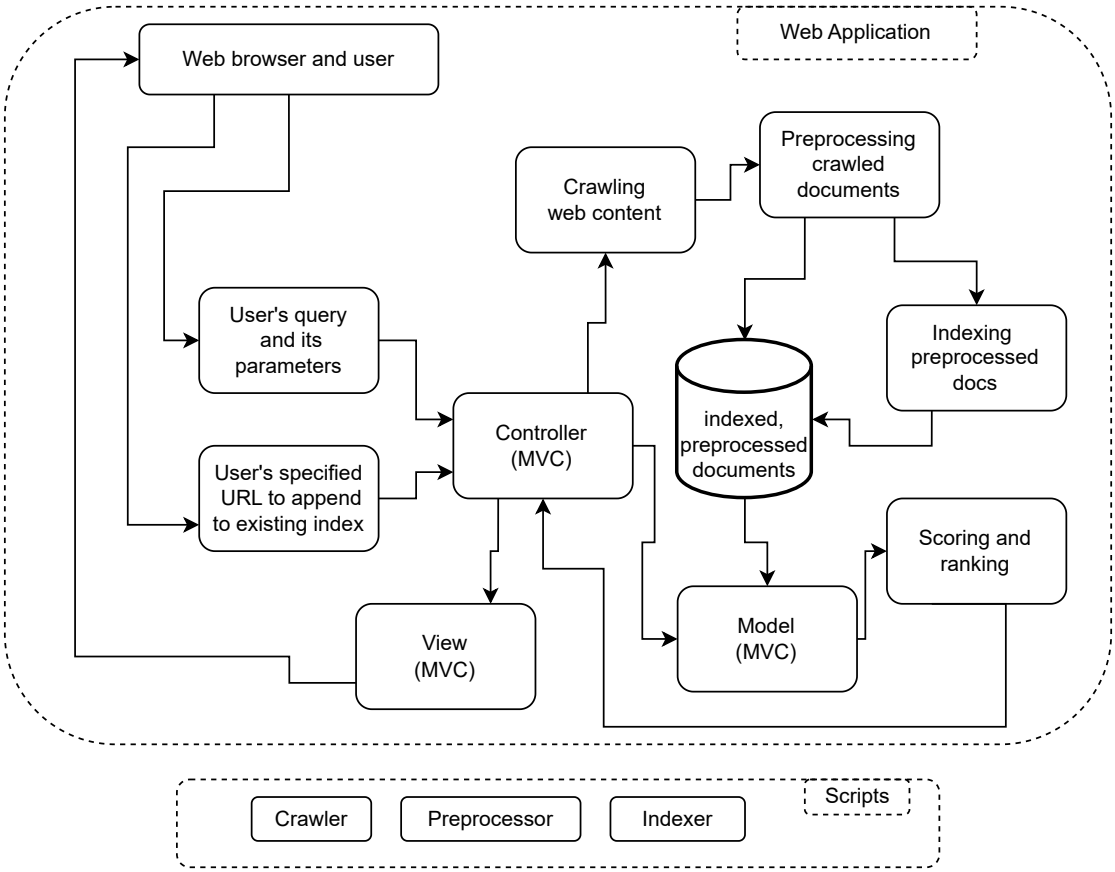

Figure 1: Architecture of implemented IR system

2.3 Crawler

The crawler class is designed mainly for Towards Data Science website and provides methods for sending "polite" requests, extracting data from `robots.txt` file, article urls from `sitemap.xml` and also retrieving data from articles (`<title>` and `<p>`) html elements. Articles consist of `index number`, `hash` for detecting possible duplicates, `dates` when the articles were written, `source URL`, `titles`, `authors`, `contents` and `website name`.

```
...
 7363)-6615916101173972302
2020-11-26|https://towardsdatascience.com/fedspeak-how-to-build-a-nlp-
↪  pipeline-to-predict-central-bank-policy-changes-
↪  a2f157ca0434|FedSpeak - How to build a NLP pipeline to predict
↪  central bank policy changes | by Yuki Takahashi | Towards Data
↪  Science
Towards Data Science Member-only Save This blog describes how I analysed
↪  central bank policy by means ...
```

Snippet 1: Sample data from crawled content in plaintext (txt format)

2.4 Pre-processor

For pre-processing is used **nltk** library with Porter Stemmer and English stopwords. Then I filtered common sentences at the beginning and the end of articles. These common sentences does not carry any specific information. Thanks to Numpy library and some sort of invested time to optimization the pre-processing program can be faster than without using the Numpy library. Hierarchy of output data is similar to crawled contents, but it is in CSV format and *Towards Data Science* as page name is removed as common sentence.

```
...
7361;-6615916101173972302;2020-11-26;Yuki
↪ Takahashi;https://towardsdatascience.com/fedspeak-how-to-build-a-
↪ nlp-pipeline-to-predict-central-bank-policy-changes-
↪ a2f157ca0434;fedspeak - build nlp pipelin predict central bank
↪ polici chang ;save thi blog describ analys central bank polici mean
↪ ...
```

Snippet 2: Sample data from pre-processed content in CSV

2.5 Indexer

The indexer script creates inverted index like structure for titles and contents that are indexed separately. For each term in dictionary is stored dictionary that consist of document IDs, term frequency in each document, TF-IDF for each document and inverted document frequency. This structure is stored in JSON format.

```
..., {
  "christina": {
    "doc_id": [0, 1757, 2044, 3032, 6243],
    "term_frequency": [1, 2.386294361119891, 1, 1, 1],
    "inverted_doc_frequency": [9.801775927129302],
    "tf-idf": [9.801775927129302, 23.389922623869342, 9.801775927129302,
               9.801775927129302, 9.801775927129302]
  }
}, ...
```

Snippet 3: Sample data from file-based index Titles in JSON

2.6 Search techniques

Search can be preformed on titles or contents from file based index as it is mentioned before. Ranked search performs operation in following notation:

- ltc.ltc

  - l – term frequency is logarithmic
  - t – document frequency is inverted document frequency
  - c – cosine similarity

BOOL search model uses logical AND for search and is available only for titles.

2.7 Django Web Application

Application consists of 3 windows. The first window is search input, the second is web content indexing and the third shows results. The look can be seen at User Reference section at the end of this documentation.

3 Conclusion

Despite the window for indexing web content is quite unstable (buttons are not disabled and it is slow), is assignment done. I spent a lot of time with web application design and optimization of pre-processor. All approx times are described below.

Spent time:

- 9h ≈ crawler (with clean code)
- 15h ≈ pre-processor and its optimization (with clean code)
- 25h ≈ web application, indexer (with clean code)
- 12h ≈ this documentation

Information Retrieval System of Towards Data Science

- before start: `pip install -r requirements.txt`

Components

Web Application

- usage: `python ./web_app/manage.py runserver`

Simple Crawler

- crawling website: Towards Data Science posts(articles) read from sitemap.xml and for each post saving title and content in `<p>...</p>` by using simple xpath expressions
- usage: `python main_crawler.py`
- or with custom parameters:

```
usage: main.py [-h] [-u MAIN_SITE_URL] [-o OUTPUT_DIR] [-p PREPARED_URLS]

SImple Crawler.

options:
  -h, --help            show this help message and exit
  -u MAIN_SITE_URL, --main_site_url MAIN_SITE_URL
                        main site that contains file robots.txt...
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        path to output dir where crawled_data directory is
                        created...
  -p PREPARED_URLS, --prepared_urls PREPARED_URLS
                        crawl prepared urls? True/False
```

- prefetch data from this app on my onedrive: here
- extract to "./crawled_data"
- if needed, dataset can be easily extended
- parallelization can be added as well but due to politeness of the crawler is not implemented

NLTK preprocessor

- usage: `python main_preprocessor.py`

```
usage: main_preprocessor.py [-h] -i INPUT_FILE_PATH [-o MAKE_CSV_ONLY]

preprocessor using NLTK lib
```

```
options:
  -h, --help            show this help message and exit
  -i INPUT_FILE_PATH, --input_file_path INPUT_FILE_PATH
  -o MAKE_CSV_ONLY, --make_csv_only MAKE_CSV_ONLY
                        reformat to csv only? True/False
```

Indexer (inverted index creator)

- usage: ▶ `python main_indexer.py`

```
usage: main_indexer.py [-h] -i INPUT_FILE_PATH [-t INDEX_TITLES] [-c INDEX_CONTENTS]

Simple indexer

options:
  -h, --help            show this help message and exit
  -i INPUT_FILE_PATH, --input_file_path INPUT_FILE_PATH
  -t INDEX_TITLES, --index_titles INDEX_TITLES True/False
  -c INDEX_CONTENTS, --index_contents INDEX_CONTENTS True/False
```

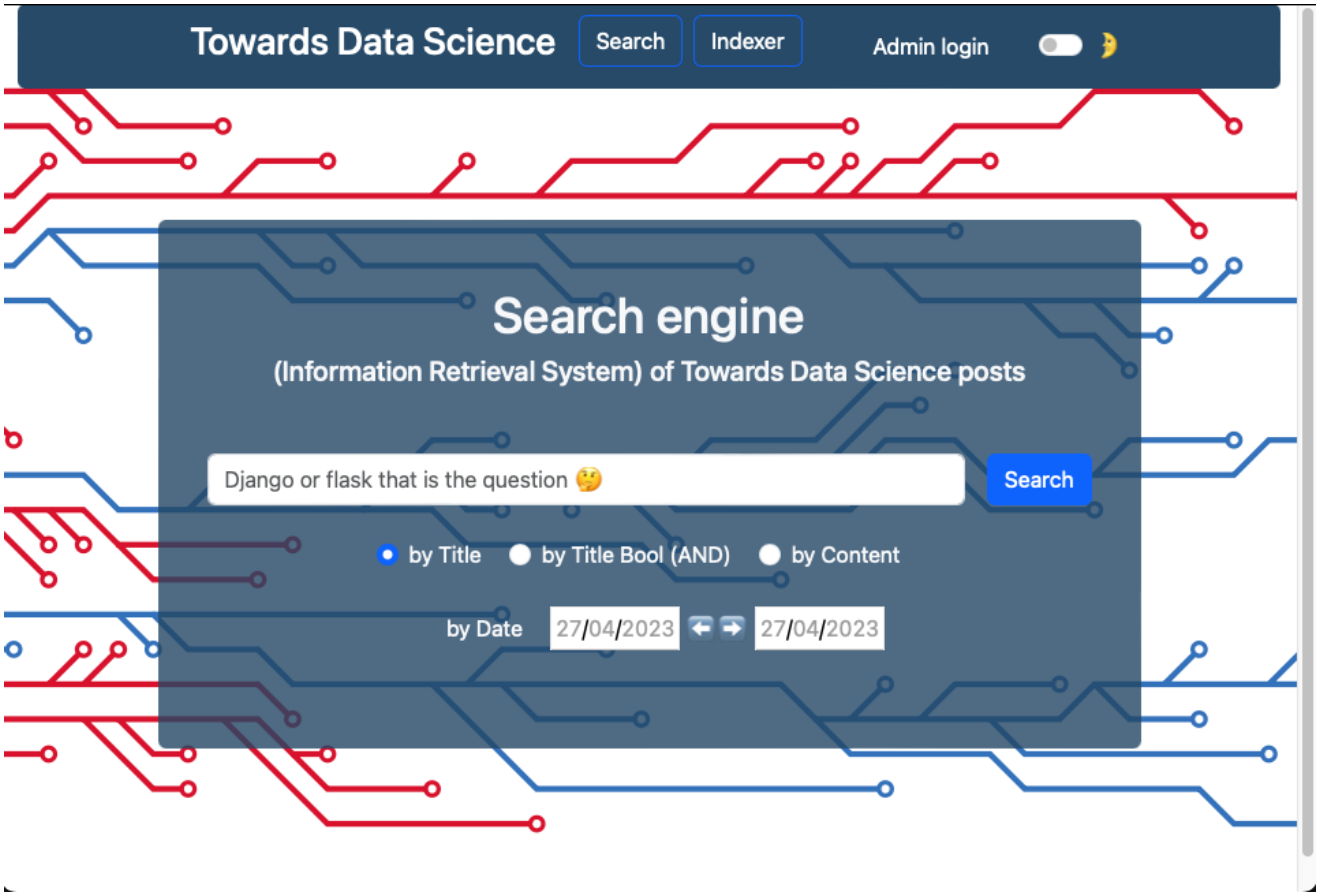

Figure 2: Index site

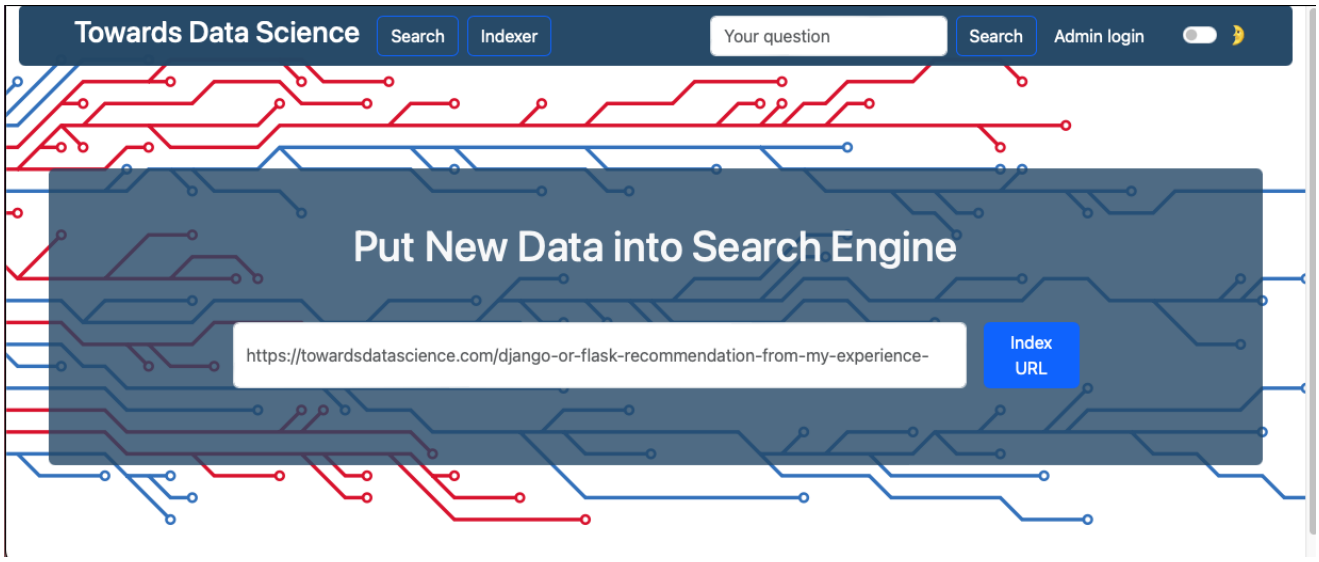

Figure 3: Web content indexing possibility

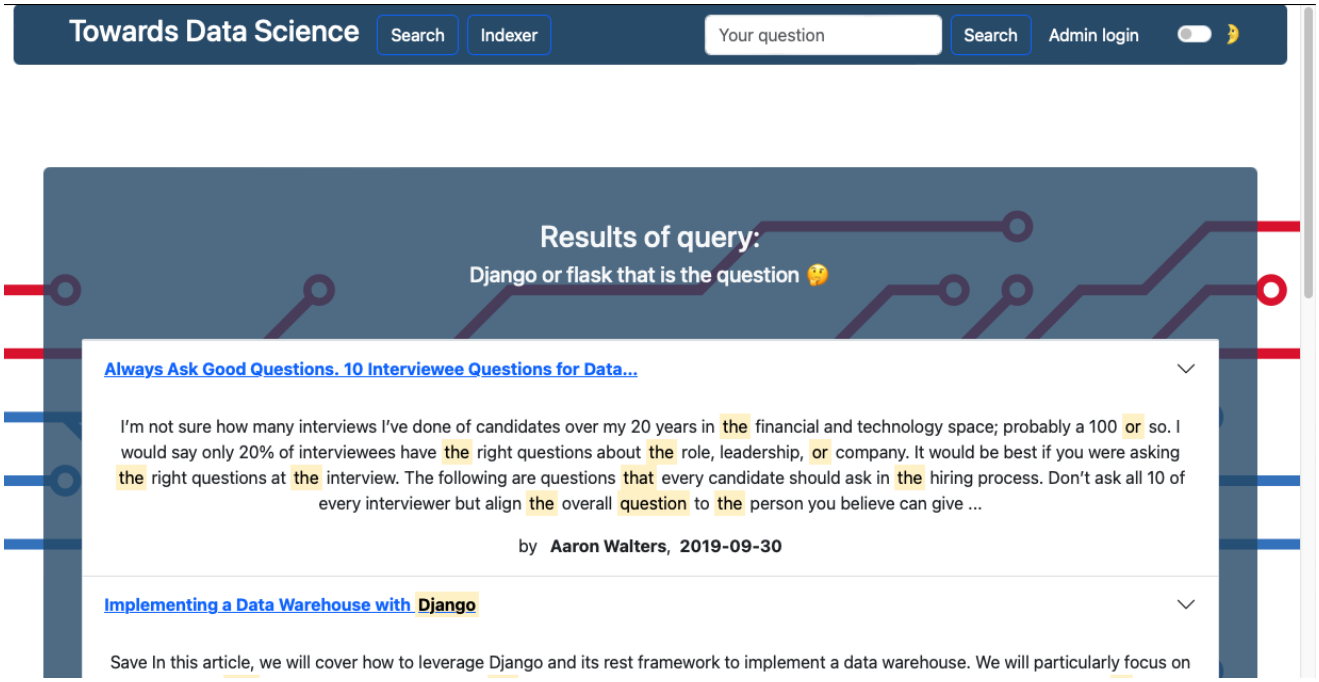

Figure 4: Search Results example