Beer Search Engine

Luca Di Bello, Enrico Benedettini

December 9, 2023

1 Dataset construction and crawling

1.1 Data sources

For our Search Engine, we looked up the most possible complete websites. We considered the best option to opt for websites that were not too complicated to scrap, but that were at the same time able to give us enough data to build an effective search engine.

So, the decision for us to pick the websites to pull the data was based on **two main parameters**:

1. **Structure of the content**: How easy it is to access the content according to the structure, whether the website is loading the content dynamically or statically, and how easy it is to parse the content.
2. **Data Quality**: How precise and detailed the information provided by the website is and how many records we could take out of it to build a good information retrieval system.
3. **Data Availability**: How easy it is to access the data and how many records we can pull out of the website.

Among all the candidates, we have chosen the following websites to pull the data from:

- **WineVybe**: database with thousands of records of beers with Producer and Beer's name, type, Alcohol By Volume (ABV%), tasting notes, closure and packaging type.
- **RateBeer**: website that contains more than 100 thousand different beers with complete description, taste notes, alcohol bv, price, packaging, critic score, and brewer details.
- **BeerMe**: a database containing 11 thousand records of beers with brewer and beer's name, style, score, production date, and location.

Initially, other websites such as (BeerAdvocate and Untappd) were considered, but they were discarded as applying scraping techniques was not possible due to the dynamic nature of the content and the authentication required to access the data loaded dynamically via APIs.

1.2 Data structure and storage

The dataset used to build the retrieval reverse index is stored in a JSONL file (JSON Lines), where each line is a JSON object containing the information collected for a specific beer. The JSONL format has been chosen since it is a convenient way to store structured data that may be processed one record at a time, which perfectly fits our needs.

Each JSON object follows a fixed structure to facilitate easy parsing of data. It contains the following fields:

- **docno**: A unique identifier for each beer.
- **name**: The name of the beer.
- **description**: A detailed description of the beer.
- **image_url**: URL link to an image of the beer.
- **price**: An object containing the price details of the beer, which includes:
  - **amount**: The numerical value of the price.
  - **currency**: The currency in which the price is expressed.
- **style**: The style or category of the beer (e.g., lager, ale).
- **critic_score**: An object representing the scores given by critics, which includes:
  - **max**: The maximum possible score.

  - **actual**: The actual score received.
- **brewer**: Information about the brewer, including:
  - **name**: The name of the brewer.
  - **city**: The city where the brewer is located.
  - **country**: An object containing the country details, with fields for the country code and name.
  - **state**: An object containing the state name.
- **alcohol_bv**: The alcohol by volume percentage in the beer.
- **tasting_notes**: Notes regarding the taste and flavor profile of the beer.
- **closure**: Information about the type of closure used for the beer packaging (e.g., cork, screw cap).
- **packaging**: The type of packaging used for the beer (e.g., bottle, can).

1.3 Data scraping

The data scraping process was done using the **Scrapy** framework, which is a Python library that allows to create ad-hoc spiders to crawl websites and extract the data in a matter of minutes. From the three websites, we have been able to extract more than **20'000 records** of beers.

Each website has been scraped using a different spider, which leverages different techniques to extract the data. The spiders are described in detail in the following sections.

1.3.1 WineVybe

Work in progess

To-do list:

- Gallery of images, based on WordPress's WooCommerce plugin
- Very straightforward to scrape, but the data is a bit shuffeled and not very clean (tabluar data does not follow the same structure on all pages, even if the same fields are present)
- The data is clean, and for most of the beers we have all the fields we need, in addition to the image URL and a description (length varies from missing to lengthy, but in most cases is present)
- Unfortunately, due to the CORS policies set to the CDN that hosts the images, we cannot access the images directly from the browser. Since downloading images to our server would be a waste of resources, we decided to leave the image URL as is, and let our web UI show a placeholder image instead.

1.3.2 RateBeer

Work in progess

To-do list:

- Very famous websites, with a lot of well-structured and high-quality data
- Unfortunately the website is not very easy to scrape as all content is loaded dynamically via JavaScript at runtime. For this reasons, we have not been able to use the standard crawling techniques presented during the course, but we have been forced to use alternative techniques referred in the official documentation of Scrapy (learn more here).
- As outlined in the documentation, we have direclty used the API endpoints leveraged by the website to load the data dynamically. This has allowed us to get the data we needed, but it has also forced us to reverse engineer the API endpoints and the data format used to load the data dynamically. This has been a bit time consuming, but we have been able to get the data we needed in a matter of hours.
- The technical details of the API endpoints and the specific requests we have used to get the data will not be discussed here, but they can be found in the code of the spider (see `./crawler/spiders/beerme.py`).

1.3.3 BeerMe

Work in progess

To-do list:

- Enrico, questo è tutto tuo :)

2 Indexing

For the indexing of the documents, we followed the guidelines indicating to use **PyTerrier**. After the completion of the crawling of the data, we realized we needed a set of strings to be able to create an index, which we didn't have yet ready for index creation.

Our decision was thus to move on to put together all the data for each record into a single string by simply taking all the values of the various fields and concatenating them with a blank space in between. We decided to go with this structure as indexers usually have to deal with plain text, and thus it's reasonable to unify the data in this way as that's how data would usually look after scraping them.

After that, we moved on to the index creation. There, a matter on which weighting model to use was raised. The final decision was to opt for the **BM25** model as it's considered the best option seen so far.

3 Implementation

3.1 Backend

3.2 Frontend

3.2.1 Lighthouse benchmark

3.3 Deployment

4 Evaluation

4.1 Evaluation metrics

4.2 System Usability Scale (SUS)

4.3 User Experience (UX) questionnaire