Università della Svizzera italiana

Information Retrieval **2024**

Student: Costanza Rodiguez Gavazzi, Agnese Zamboni, Davide Frova

Due date: Friday, 20 December 2024, 11:59 PM

1. Overview

Our retrieval system is about charities. We collected data from two websites: Global Giving and Charity Navigator.

Our work has been split evenly between team members. Davide took care of the frontend, Costanza of the scraping and data retrieval, and Agnese of the backend, indexing, and query augmentaiton.

1. The Features

We decided to implement the following features:

- **Filtering (simple feature)**: in addition to being able to search by title, the user is able to filter the results based on 3 additional attributes. In our case, the user can filter the charities by:
  - causes: one or more causes
  - country: one or more countries
  - continent: one or more continent

  The possible options are a list of the existing fields in the charities we obtained in the data retrieval. This way the user can select one or more options from existing ones. The user interface also offers autocompletion to make the filtering exerience easier.
- **Results Snippets (simple feature)**: the results are presented in the form of result snippets in a kind of "Google style", with query terms highlighted.
- **User Relevance Feedback (complex feature)**: the user can provide a positive or negative feedback on each result to mark them as relevant or irrelevant. We implemented TF-IDF query expansion, keeping track of all the preferences specified for a query, and resetting the feedback whenn the user inserts a new query.

2. Frontend Design and Implementation

The initial design research and UI mockup creation have been completed using Figma. Then, the design has been implemented using Next.js and Material UI to have a modern, minimalistic, user-friendly, "Google Style" look.

1. The landing page

The landing page is minimalistic, containing only the search bar and the filters. The design is inspired by Google, having a wide search bar that invites the user to insert a query. The white colors are also inspired by Google's design, with a strong shading contrast highlighting the search bar. The search bar is centered and the search button is large and minimalistic to make the searching experience more simple. This page contains as little text as possible to avoid overwhelming the user, but there is an invitation to search charities, and an example of a query to inspire the user's imagination.

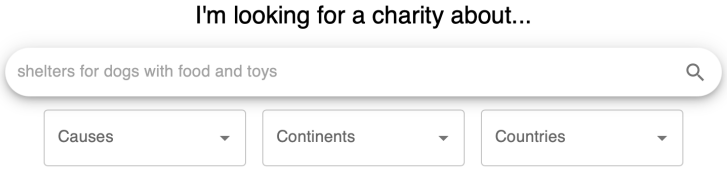

Figure 1: The landing page

2. The results page

The results page is also inspired by Google's design, by always showing the search bar and the filters at the top of the page to allow the user to refine the query at any time. The search bar always has a reasonable size and shows the query to which the results correspond to.

The results are shown as an ordered ranked list, where the charity which is more relevant to the query is shown first. The ranked list of charities is bounded at 1000 results, to avoid sending too much data to the frontend and ensure that the user is not scrolling forever.

The result snippets contais information about the charity in a "Google Style" desing, showing the charity's logo and name clearly. Clicking on the result snippet will take the user to the official web page of the charity. Each snippet also contains some lines of the document's text, showing the query terms highlighted. This allows the user to get a feeling of the content of the document in relation to the query. This was part of the second required feature we implemented. The text content is kept short (2-3 lines) to avoid overwhelming the user and allowing each snippet to have a reasonable size.

Lastly, each snippet has a feedback section, also kept minimalistic, that prompt the user to give relevance feedback on the charity. Clicking on the thumb up or thumb down icon will refresh the result page, and apply the relevance feedback, performing wuery augmentation and displaying more relevant results.

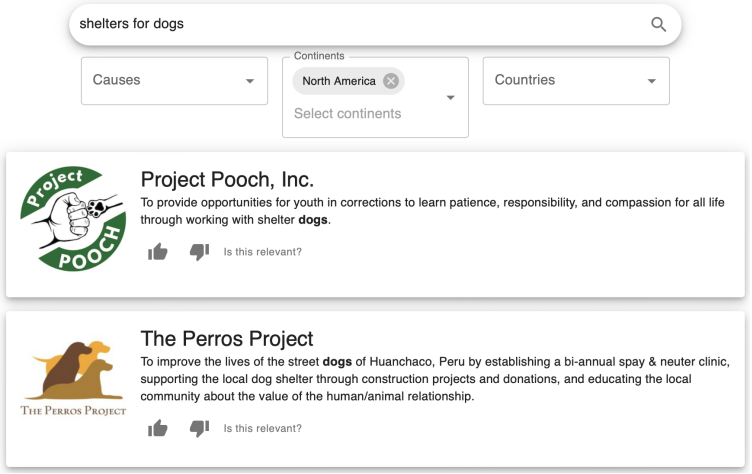

Figure 2: The results page

3. The filters

The filters are always present under the search bar, and allow the user to select one or more option from the possible values. Autocompletion is supported when picking an option, to make the filtering experience less time-consuming. The selected options are shown to the user, allowing to easily see the and remove the filters currently set. Clicking on the search icon will search results relevant to the query and filter them according to the chosen filters.

As metioned previously, the options of each filter are taken from the actual values that the fields take in all the charities in the data. We decided to do this so that the user can see what options are available in order to inspire them and give them an overview of what's available. The list of options is computed in the backend once, and retrieved statically from the frontend as a JSON object.

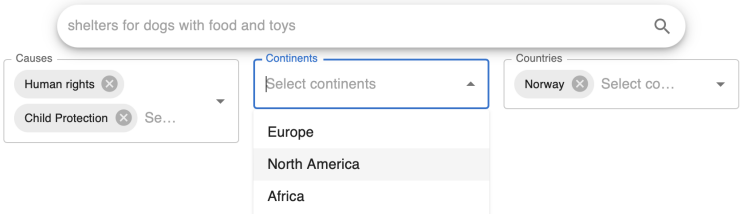

Figure 3: The filters

3. Data Collection and Processing

We obtained the data from two websites, using a mix of leveraging their API's and applying scraping methods. The two websites are Global Giving and Charity Navigator.

The data has been obtained using the Jupyter Notebooks in the folder `scraping`. The resulting JSON files have then been moved into the backend to allow static data retrieval when building the index.

1. Global Giving

Global Giving provides structured data on charitable organizations through its API in XML format. We utilized this API to collect comprehensive information about each charity, ensuring a standardized method of data retrieval.

1.1. Data Retrieval

To gather data, we accessed the Global Giving API and downloaded XML files containing details of various organizations. The use of their API eliminated the need for web scraping, allowing us to work directly with structured data. The XML files provided the following information about each charity:

- Basic details such as the charity's name and location.
- Mission statements and website URLs.
- Themes representing the causes they work on.
- Countries where the organizations operate.

1.2. Data Processing

After retrieving the data, we developed a parser using Python's `ElementTree` library to extract information from the XML format. To enhance the dataset:

- Geographical information was added using `pycountry` and `pycountry_convert`, allowing us to classify organizations by continent based on their headquarters.
- Field names were standardized to align with the data structure of Charity Navigator, ensuring consistency across platforms.
- The final dataset was converted to JSON format for easier storage and readability.

2. Charity Navigator

Charity Navigator offers data through its GraphQL API. Unlike Global Giving, Charity Navigator does not provide organization logos, requiring additional processing to locate this information.

2.1. Data Retrieval

Using the GraphQL API, we tailored queries to retrieve:

- Organization details, including names and locations.
- Causes, mission statements and website URLs.

The GraphQL API's flexibility allowed us to avoid redundant data requests and efficiently handle nested data structures. 10,000 records were retrieved in batches of 10 to not flood the server.

2.2. Data Processing

The lack of logos in Charity Navigator's data required us to develop a custom solution for locating and extracting organization logos from their websites. This system:

- Parsed webpage structures using `BeautifulSoup` to identify elements marked as logos.
- Checked metadata and special tags (e.g., `svg`) for logo information.
- Scanned for images commonly used as logos, filtering out irrelevant elements like favicons or menu icons.

Standard Python libraries, including `requests`, `BeautifulSoup`, and `re`, facilitated these operations. Finally, the processed data was converted to JSON format to match the structure of Global Giving's dataset.

4. Backend and Indexing/Retrieval

1. Indexing and Retrieval

For the indexing we used Python-Terrier with the Retriever model "BM25".

1.1. The Documents

The data is collected from the JSON files produced by the scraping, and charities with missing data are removed. In the index, the "text" column in the Pandas DataFrame for each charity will contain a combination of the charity's name and the charity's mission, so that the user can find results based on both the title and the description.

1.2. The Index

The index documents are kept in the folder `backend/index_docs`, and we used BM25 as the retrieval model. After the query is inserted, and the list of relevant documents is obtained, each document is mapped to its corresponding charity object using the "docno" column, and the ranked list of charities is returned to the frontend.

1.3. The Relevance Feedback

For the relevance feedback, a dictionary `feedback` is kept, containing the feedback information for each session. When the user inserts a new query, the session is restored so that the feedback on a previous query doesn't affect the feedback on a new query. For each session the feedback is kept in the form of a list of `(docid, relevant)` pairs.

We perform query augmentation using TF-IDF, considering all the feedback provided on a single query.

The documents on which the user has expressed feedback are separated into relevant documents and non-relevant sets. Using a TF-IDF vectorizer with tokenization and stopword removal, we generated a term-document matrix for both relevant and non-relevant documents. Then, term weights are computed by summing TF-IDF scores from relevant documents and subtracting scores from non-relevant ones. This emphasizes terms strongly associated with relevant documents while downplaying those linked to non-relevant ones. Terms are sorted by their computed weights, and positively weighted terms are added to the original query. Original terms are retained to preserve the query's context.

This approach dynamically refines the query by integrating the feedback of the user, enhancing the relevance of the query.

2. The Backend

Since we used Python-Terrier for the retrieval, we decided to have a backend in Python, and we used Fast API.

The backend contains two routes, one is used for the retrieval based on the query and filters, the other is used for the user relevance feedback.

The retreival of the document and the indexing are performed only once, when the backend is initialized.

2.1. The Routes

- `GET /search?query=<query>&session_id=<session_id>`:
  This route takes the following parameters:
  - `query`: a string representing the query input by the user
  - `session_id`: the id of the session. A session starts when a new query is inserted, and is used to keep track of the relevance feedback for a specific query.
  - `causes`: optional list of strings containing the filtering information of the cause attribute.
  - `continents`: optional list of strings containing the filtering information of the continent attribute.
  - `countries`: optional list of strings containing the filtering information of the country attribute.

  This route takes the index and the documents initialized when the backend is run, and performs a retreival of the documents relevant to the query. Once the ranked list of results is obtained, it is mapped to a list of charity objects that are then filtered according to the filters specified in the request. The resulting list is returned as a response to the request. The result is a list of objects with the following attributes:
  - `score`: the score of the document.
  - `docid`: the id of the document in the index.
  - `charity`: a complex object containing all the information of the charity, like the name, logo, mission, coutnry, etc. . . .

- `POST /feedback/<session_id>/<docid>/<relevant>`:
  This route takes the following parameters:
  - `session_id`: the id of the session.
  - `docid`: the id of the document on which the feedback is expressed.
  - `relevant`: 1 if the user has identified the document as relevant to the query, 0 otherwise.

  This route adds information about the feedback regarding the session specified. For each session there is a list of feedback entries for each `(docid, relevant)` pair. After the feedback for the session has been updated, the query augmentation is performed using TF-IDF considering ALL the documents that the user has specified as being relevant or not relevant to the query.

5. User Evaluation

6. Appendix