Project Part 4 Report: RAG, UserInterface, and WebAnalytics

Information Retrieval And Web Analysis 2025

Date: 5/12/2025
Group Identifier: G_007
Group Member 1: Marc Bosch Manzano u215231
Group Member 2: Chris Matienzo Chilo u198726
Group Member 3: Àlex Roger Moya u199765

This file was edited with Markdown language and exported with the Markdown Preview Enhanced extension (by Yiyi Wang) from VS Code.

GitHub repository link

The GitHub repository with the code of the labs is the following: https://github.com/u215231/Information-Retrieval-and-Web-Analytics-Project.git. Here, you have to navigate to the folder where the Part 4 solution is located: `IRWA-2025-part-4`. In this folder, an information retrieval search application can be found, where a `README.md` explains how to execute it. The general `README.md` in the root of the GitHub repository also explains a short description how to proceed in all parts of the project, including Part 4.

Index

1. Introduction

In this fourth part of the project, instead of focusing on the retrieving documents task, we were focused on building a functional web. This web includes a Retrieval-Augmented Generation (RAG) that summarizes the results information, and lastly, an analytics system that stores the user's interactions that visit the webpage.

The idea of this project was to mix this workflow in a cohesive way, that way when the user does a search:

- System processes the search.
- Returns the indexed information.
- Generated a summary of the results.
- Registers every interaction.

This way, the application not only answers, but also retains valuable information for the web analysis.

2. User Interface (UI)

For the web interface we tried granting the user with the most relevant and coherent information for their search.

2.1. General UI structure

The application is organized in three main pages:

- **Home (`/`)**: The home page, where the user enters the query.
- **Results (`/search`)**: Shows the retrieved documents, ordered according to the ranking defined by the search engine. Here it is also where a summary of the results is presented, generated through a RAG system. A maximum number of documents are retrieved; this amount has to be defined in an environment variable called `MAX_DOCUMENTS_DISPLAYED` in the `.env` file.

- **Document Details (`/doc_details`)**: When clicking a result, the user accesses a more detail view of the product, including:
    - Category
    - Title
    - Brand
    - Rating
    - Price
    - Discount
    - Description
    - Image ot the product

2.2. User interaction flow

The user poses a query and receives a set of documents accompanied by a generated summary (RAG). When a user accesses a document (`/doc_details`), it is registered as a click in the analytics system (Stats section), to later show the metrics of the user's history. User can return to the Results section by pressing the `Back to results` option.

2.3. Search engine implementation

We have implemented the search engine taking advantage of algorithms we developed in previous parts, and modifying parts of the provided algorithms.

First, we have modified the `web_app.py` file. We have added a dataframe containing the `fashion_products_dataset_processed_review.csv` , which is an updated version of the processed file in Part 1 of the project. The file was generated using the processing.py file from myapp/search/solution directory. In this directory can be found the source code files of the ranking library we developed in Parts 1, 2, and 3.

Then, we have modified search_engine.py, which can also be found in myapp/search, implementing the `tfidf_search` method, which is an adaptation of `get_scores_df` from the `TFIDF` class developed in Part 3. This method returns a document list similar to the default `dummy_search` that was provided.

3. Retrieval-Augmented Generation (RAG)

Once the search flow and interface were built, we incorporated the Retrieval-Augmented Generation module, which acts as a bridge between the engine results and a more natural response for the user. The idea is that the user does not have to read the full descriptions (results), but rather generates a clear synthesis of the difference between the products and whether they really fit their search.

We have implemented the RAG by creating a Groq Cloud account on this web page: https://groq.com. Then, we have created a Groq API Key and we have setted it the `.env` file in `IRWA-2025-part-4` directory. Then, we have executed the web and we have obtained the RAG generated summary for the retrieved documents.

The algorithms that execute the RAG system can be found in rag.py file from myapp/generation.

4. Web Analytics

The third part of this project has consisted of an analytics system capable of observing how users interact with the search engine, tracking the queries made, which documents are of most interest, which server routes receive the most traffic, and how a session evolves from the moment the user arrives until they leave the web.

The file analytics_data.py, from myapp/analytics, stores the following information in memory:

1. **Request**: Registers every HTTP request that reaches the server, such as route, IP, browser, and timestamp. This table allows identifying which pages are visited the most and from which devices.
2. **Sessions**: Stores a unique identifier per session and its initial timestamp. In this way, it is possible to track a user's activity during their stay on the web.
3. **Queries**: Saves each query written by the user, including the number of terms and an incremental identifier. This allows analyzing what terms are used.
4. **Clicks**: Captures each click made on a result document, indicating which query it belongs to.
5. **FactClicks**: An accumulated count per document that has been clicked. This is useful for detecting product relevance.

4.1. Application Integration

All these tables are updated via calls from `web_app.py.` Whenever an event occurs, the application delegates the registration to `analytics_data.py` (class AnalyticsData).

On the main page, the user session is generated or recognized. In each request, a record is added with the visited route and the detected browser. Every time a search is performed, the query is stored as a new entry. When the user clicks on a result, the document counter is incremented and the event is saved.

4.2. Additional Metric: Query Length Analysis

In addition to the necessary fields to register queries and clicks, we save additional information that we consider useful for later analysis. For example, for each query, we store the number of terms (`num_terms`). This data allows studying the average length of searches and allows us to distinguish between generic queries ("jacket") and more specific ones ("black leather jacket slim fit").

5. Analytics Dashboard and Visualization

The analytics system is presented to users through a dashboard (`/dashboard` route) that visualizes key metrics and user behavior patterns.

5.1. Dashboard Components

The dashboard (`dashboard.html`) displays four main visualizations for data exploration:

1. **Number of Views Over Time** (`/plot_number_of_views`)
    - Tracks page visits across all routes
    - Shows traffic patterns and peak usage periods
    - Helps identify when users are most active
2. **Browser Statistics** (`/plot_browsers`)
    - Displays which browsers users employ (Chrome, Firefox, Safari, etc.)
    - Useful for ensuring cross-browser compatibility
    - Supports responsive design decisions
3. **Operating System Statistics** (`/plot_os`)
    - Shows distribution of user operating systems (Windows, macOS, Linux, etc.)
    - Informs mobile vs. desktop traffic ratios
    - Guides platform-specific optimization efforts

4. **Hourly Traffic Analysis** (`/plot_time`)
    - Analyzes traffic distribution by hour of day
    - Identifies peak usage hours
    - Supports server resource planning

5.2. Query Statistics Table

A structured table displays the most frequently searched terms:

- **Query Terms**: The exact search query entered by users
- **Times Searched**: Count of how many times each query has been performed
- Sorted by frequency to highlight popular search patterns

5.3. Additional Statistics Pages

Stats Page (`/stats` route): Provides detailed view of:

- **Document Clicks Ranking**: Top-clicked products with click counts
- **Request Logs**: Complete record of all HTTP requests with timestamp, route, IP, and browser info
- **Session Data**: User session tracking with session IDs and timestamps
- **Query Details**: All queries performed with term count and query ID
- **Click Events**: Detailed click history linking queries to documents
- **Dwell Time Metrics**: Time spent on document detail pages

6. Conclusions

In this project part, we have developed a search engine application following a template framework. This application has allowed us to understand how the ranking process of documents done in the back-end can be presented elegantly on the front-end. We have also included all that we have developed in previous parts in this project part -Python source code files, processed data files, notebooks, and reports. Finally, we have learned how to generate statistics for the application and Artificial Intelligence (AI) generated comments for the retrieved comments (RAG).