Part 2: Indexing and Evaluation

Maria Beili Mena
Sergi Rodríguez
Martí Vilaró

The GitHub TAG for this part of the project is IRWA-2025-part-2 and its corresponding URL is: https://github.com/MariaBeili/IRWA-2025-G_003/tree/main/project_progress/part_2

PART 1: Indexing

1. Overview

The objective of this stage was to design the indexing and ranking functions for the search engine. For this three functions were coded in the file *indexing.py*:

- ➔ *create_index_tfidf*
- ➔ *rank_documents*
- ➔ *search_tfidf*

Furthermore, five queries were written as examples to test the search engine. To test the queries another support function *process_query* was coded in the file *query_preparation.py*.

2. create_index_tfidf

This function receives the collection of documents after being processed into *ProcessedDocument* types. The function loops through all documents to create the inverted index, and calculates the TF, DF, and IDF. It also returns a dictionary to map the PID to titles of the documents.

To make the process efficient a temporary inverted index is created for every document. Once it is finished, it is merged with the global inverted index.

3. rank_documents

This function receives a list of query terms, a set of documents to search within, and the precomputed index, IDF, and TF values. It builds a query vector and document vectors based on the TF-IDF model to calculate similarity scores between the query and each document.

The function loops through all query terms, updating the vectors accordingly, and computes the cosine similarity using the dot product. Finally, it sorts the documents by their similarity scores in descending order and returns the ranked list of document IDs.

4. search_tfidf

This function receives a user query along with the precomputed index, TF, and IDF values. It processes the query into individual terms and retrieves the set of documents that contain all those terms.

For each query term, the function collects the documents where it appears and computes the intersection to apply the AND condition. Once the matching documents are found, it ranks them using the TF-IDF model through the *rank_documents* function and returns the final ordered list of relevant documents.

5. process_query

This function processes the queries similar to how the text variables were processed for the documents. The reason why a different function was used was because the document data was in standard English, so some common processing practices were skipped. These are present in this function and they normalize and clean text by removing accents and non-ASCII characters.

This also lets us add other more advanced query processing practices in the future like misspell correction, or query expansion.

6. Five query examples

Query 1: ARBO cotton track pants for men

Query 2: Multicolor track pants combo ECKO

Query 3: Black solid women track pants

Query 4: Elastic waist cotton blend track pants

Query 5: Self design multicolor track pants

The queries were chosen to test high term frequency relevance words like York, cotton, track pants, solid, multicolor, and combo.

The index terms in each query were combined to test different document variables (brand, color, fabric, gender, style). This is important because the search engine filters intersecting the index terms. So mixing two terms of the same document variable can lead to empty result lists.

Furthermore they are also practical queries that could commonly be found in an e-commerce website.

PART 2: Evaluation

1. Overview

The goal of this section is to measure the effectiveness of the ranking provided in the last task of part 1. The evaluation process has been divided into:

1. Using the provided ground truth (validation_labels.csv) for predefined queries
2. Defining our own queries, manually building a ground truth, and analyzing the system's

All seven metrics were implemented in *evaluation.py*: **Precision@K**, **Recall@K**, **F1-Score@K**, **Average Precision@K**, **MAP**, **MRR**, and **NDCG**.

2. Function Implementation

For each function, the input is composed of the retrieved documents and the relevant one. Some functions also consider the variable K, which is an arbitrary value used to compute the evaluation metric for the first K elements.

We have taken into account all the possible inputs to make sure that it doesn't result in an error.

3. Evaluation 1: Provided Queries

For this part, we used the two predefined queries:

- query 1: women full sleeve sweatshirt cotton
- query 2: men slim jeans blue

The search results were with metrics calculated at k = 10 and round to 3 decimals:

Table 1		
Metric	Query 1	Query 2
P@K	0.100	0.100
R@K	1.000	0.600
F1@K	0.182	0.171
AP@K	0.200	0.333
RR	0.200	0.333
NDCG	0.397	0.307

Overall MAP: 0.267

Overall MRR: 0.267

4. Evaluation 2: Own Queries

In this part, we worked as an “expert judge” to evaluate my system using queries and our own ground truth.

4.1 Generation

First, we defined 5 new queries in run_search.py.

Then, we ran these queries and saved them in search_results.txt. After this, we manually inspected these results and created our own ground truth file, my_queries_validation_labels.csv, assigning a binary relevance label between 1 or 0 to each PID.

Our judgement was for *ARBO cotton track pants for men* and *Elastic waist cotton blend track pants*, the top results were generic "*Solid Men Black Track Pants*" or "*Solid Women Grey Track Pants*". We judged these as 0 (irrelevant) because they missed the specific, important terms ("ARBO", "cotton", "elastic waist"). The queries *Black solid women track pants* and *Self design multicolor track pants* returned exact title matches, which we judged as 1 (relevant). The query *Multicolor track pants combo ECKO* returned 0 results.

We then ran our evaluation script my_query_evaluation.py to calculate the metrics for our 5 queries using our new ground truth file.

Table 2

Query	P@10	R@10	F1@10	AP@10	RR	NDCG@10
Query 1	0.00	0.000	0.000	0.000	0.000	0.000
Query 2	0.000	0.000	0.000	0.000	0.000	0.000
Query 3	0.500	1.000	0.667	1.000	1.000	1.000
Query 4	0.000	0.000	0.000	0.000	0.000	0.000
Query 5	0.625	1.000	0.769	1.000	1.000	1.000

Overall MAP: 0.400

Overall MRR: 0.400

5. Explanation and Analysis

The results from both tables show our system's performance is extremely polarized: it either succeeds perfectly or fails completely.

In the first evaluation (Table 1) shows that query 1 R@K is 1.00 (it found all relevant docs) but P@K is 0.100 (only 1 of the 10 results were relevant). So, the system retrieves relevant items but doesn't rank them properly. Also, we can check this for the MAP (0.267) and MRR (0.267).

The second evaluation (Table 2) confirms previous results. Query 3 and Query 5 worked really well because they matched the product titles exactly. But when the query became more specific like query 1 and query 4, the precision and every other metric fell to 0.000. What happened is that the system ignored the important specific terms and returned only very generic results instead. Same explanation for overall MAP and MRR.

6. System Limitations and Improvements

From our analysis we can weaknesses:

1. **All text fields are treated the same:** The failure of queries like "ARBO cotton track pants for men" and "Elastic waist cotton blend track pants" really shows the problem. They returned things like "Solid Men Black Track Pants", which are too generic. This happens because the system merges all textual fields (title, description, brand, product_details) into a single field called search_text. Then, TF-IDF treats every match as equally important

   We can fix this if we calculate separate TF-IDF scores for each field. Then I'd give more weight to the title, because it usually contains the most relevant words.

2. **Too strict:** Another issue is that our search function uses a strict AND logic. That means a document must include all query terms to be retrieved.

   We can fix this problem by switching from *AND* to OR logic. Then, all documents containing at least one query term will be retrieved. TF-IDF will automatically give higher scores to the ones matching more terms. This should improve recall and make the system less strict.

3. **We propose** to upgrade to BM25 Ranking because another important limitation is that TF-IDF doesn't consider document length affecting the score just because the query contains more words, even if the context is not relevant.