GithubRepository:
https://github.com/RaulMartin01/IRWA-2025-RaulMartin-NoelPedrosa-AdriaPorta.git

Report IRWA Part 2

1: Part 1: Indexing

Objective

The goal of this phase was to build an inverted index and a TF-IDF-based retrieval system which was capable of returning the most relevant fashion products for the given text queries. For this whole part 2 of the project we had to use the dataset preprocessed in Part 1.

Dataset Input

We used the cleaned dataset fashion_products_clean.csv, which contains 28,080 documents after preprocessing (title, description, brand, category, etc.). Each document represents a product from an e-commerce catalog, as we knew from the previous part.

Approach

The indexing approach was divided into four main stages: text preparation, inverted index creation, TF-IDF modeling, and similarity-based search.
In the first step, we merged several key textual attributes, specifically, clean_title, clean_description, combined_info, and product_details_text into a single searchable field. This more unified representation now ensures that all relevant product information, such as the category, brand, seller, and descriptive details, contribute to improving search relevance across the whole dataset

Next, we built an inverted index using Python's `defaultdict(set)` data structure. For every document, the text was tokenized using `nltk.word_tokenize()`, and the tokens were cleaned after that by converting them to lowercase, removing stopwords, and excluding punctuation marks. Each remaining term was then mapped to a list of product identifiers (pid) in which it appears. The resulting inverted index allowed efficient word-to-document lookup and the total number of unique terms was 5.131.

The third stage consisted of creating a TF-IDF model using the `TfidfVectorizer` from scikit-learn. This model numerically represents each document according to the relative importance of its words across the corpus. The main configuration parameters were: `max_features = 15000`, `stop_words = 'english'`, and `lowercase = True`. After training, the vectorizer produced a TF-IDF matrix with shape (28,080 × 5,066), which means that 28,080 product documents and 5,066 distinct terms were represented there. Each matrix cell had the weight of a given term within a specific product description.

Finally, we implemented a similarity and search mechanism based on cosine similarity. The system computes similarity scores between the TF-IDF vector of a user's query and all product vectors in the dataset. Documents are then ranked according to these scores, and the top-k, which in our case k=10, most relevant items are retrieved using the `search_query()` function.

Results of part 1

The search function had to be tested with various example queries, in order to check that the indexing and the retrieval process worked correctly. The results were the following:

- "women blue cotton t-shirt", which returned *Printed Women Round Neck Multicolor T-Shirt* with a similarity score of 0.666.
- "men black jeans slim fit", which retrieved *Slim Men Black Jeans* with a similarity score of 0.442.
- "cotton round neck sweatshirt", which matched *Full Sleeve Color Block Men Sweatshirt* with a similarity score of 0.485.
- "women red dress long sleeve", which produced *Women Solid Pure Cotton Ethnic Dress* with a similarity score of 0.394.

With these results we verify that the search function does its work perfectly by ranking higher the documents with higher similarity scores with the query.

Here's a cleaned-up **Part 2: Evaluation** section you can drop into the report, consistent with your notebook outputs and with the "numeric-only" instruction for 2.2.

2. Part 2: Evaluation

This part evaluates the retrieval system built in Part 1. We first implemented the standard IR metrics, then applied them to the two predefined queries using the file validation_labels.csv, and finally created our own ground truth for the five custom queries defined earlier.

2.1 Metric implementation

We implemented the seven metrics required in the statement:

- Precision@K (P@K): proportion of retrieved documents in the top *K* that are relevant.
- Recall@K (R@K): proportion of all relevant documents that appear in the top *K*.
- F1-Score@K: harmonic mean of precision and recall at *K*, used to balance the two.

- Average Precision@K (AP@K): averages the precision values at the ranks where relevant documents appear; rewards ranking relevant items higher.
- Mean Average Precision (MAP): mean of AP over all queries; single-number summary of effectiveness.
- Mean Reciprocal Rank (MRR / RR): looks at the rank of the first relevant item; higher when the first relevant document appears early.
- NDCG@K: accounts for the position of relevant documents using a logarithmic discount; good to evaluate overall ranking quality, not just the first hit.

All metrics were implemented on top of the ranked list produced by our TF–IDF + cosine similarity search function, and relevance was taken from the corresponding ground-truth file (either the provided validation_labels.csv or the manual one we generated in 2.3).

2.2 Evaluation with provided relevance judgments

In this subsection we evaluated the system only on the two queries specified in the assignment, using the relevance labels from validation_labels.csv:

1. women full sleeve sweatshirt cotton
2. men slim jeans blue

The notebook reports only numeric results, rounded to three decimals, as required.

Results at K = 10

- Query 1
  - P@10 = 0.000
  - R@10 = 0.000
  - F1@10 = 0.000
  - AP@10 = 0.000
  - RR = 0.000
  - NDCG@10 = 0.000
- Query 2

  - P@10 = 0.000
  - R@10 = 0.000
  - F1@10 = 0.000
  - AP@10 = 0.000
  - RR = 0.000
  - NDCG@10 = 0.000
- MAP (@10) = 0.000

Because in both queries the first relevant item appears only around rank 15, all metrics at K=10 are 0.

To better understand system behaviour, we also computed the metrics with a larger cutoff (K = 100):

- Query 1
  - Precision@100 = 0.030
  - Recall@100 = 0.231
  - AP@100 = 0.055
- Query 2
  - Precision@100 = 0.040
  - Recall@100 = 0.400
  - AP@100 = 0.062
- MAP (@100) = 0.059

So, with a deeper cutoff the system does retrieve part of the relevant products (3/13 for query 1 and 4/10 for query 2), but too low in the ranking, which is exactly what the metrics show.

(Explanations and interpretation go here in the report, not in the notebook.)

2.3 Expert-labeled evaluation (custom queries)

The assignment also asks us to “act as expert judges” for the five queries defined in Part 1. To make this reproducible, we did automatic but explicit judging like this:

1. Take each of our 5 custom queries:
    - women blue cotton tshirt
    - men black jeans slim fit
    - cotton round neck sweatshirt
    - women red dress long sleeve
    - men leather jacket brown
2. Use our Boolean AND helper over the inverted index to find the documents that actually contain all the query terms.
3. Mark those documents as relevant = 1.
4. Take the top 100 TF–IDF ranked documents for that query and build a validation-like table
   (query_id, pid, label).
5. Evaluate the ranker (the same TF–IDF system) against that ground truth.

This produced the following numeric results at K = 10 (these are the ones from your notebook):

- Query 1 (women blue cotton tshirt)
    - P@10 = 0.800
    - R@10 = 0.129
    - F1@10 = 0.222
    - AP@10 = 0.643
    - RR = 0.333
    - NDCG@10 = 0.641
- Query 2 (men black jeans slim fit)
    - P@10 = 0.000

    - R@10 = 0.000
    - F1@10 = 0.000
    - AP@10 = 0.000
    - RR = 0.000
    - NDCG@10 = 0.000
- Query 3 (cotton round neck sweatshirt)
    - P@10 = 0.900
    - R@10 = 0.265
    - F1@10 = 0.409
    - AP@10 = 0.989
    - RR = 1.000
    - NDCG@10 = 0.934
- Query 4 (women red dress long sleeve)
    - P@10 = 0.000
    - R@10 = 0.000
    - F1@10 = 0.000
    - AP@10 = 0.000
    - RR = 0.000
    - NDCG@10 = 0.000
- Query 5 (men leather jacket brown)
    - P@10 = 0.200
    - R@10 = 0.667
    - F1@10 = 0.308
    - AP@10 = 0.417

    - RR = 0.333
    - NDCG@10 = 0.437
- MAP (over the 5 custom queries) = 0.410

So, in our own relevance setup the system performs well for clearly-described, well-represented products (queries 1 and 3), and poorly for queries where:

- the query terms are rare,
- the dataset uses slightly different wording,
- or the Boolean AND makes relevance too strict.

2.3.b Brief discussion of metrics

- Precision@K told us whether the very top of the ranking was clean. For the two provided queries it was 0.000, meaning that the first relevant item was not in the first 10 positions.
- Recall@K showed that, when we looked deeper (K=100), the system was in fact retrieving some relevant products, just not early enough.
- RR and NDCG made the "first relevant at rank ~15" problem very visible: when a relevant document appears late, both of these drop to almost 0.
- MAP was useful to summarize all queries with one number. On the course-provided labels, MAP was very low (0.059 at K=100); on our manual labels, MAP was much higher (0.410), which is expected because our relevance definition was better aligned with the TF–IDF representation.

2.3.c System limitations and improvement ideas

From the experiments above we can point out a few issues:

- Relevant items too low in the ranking. For the two teacher-provided queries, relevant products were typically found around rank 15 or later.
  Fix: use BM25 instead of plain TF–IDF; boost title; or add a pre-filter by category/gender before ranking.

- Exact-term dependence. Our index and TF–IDF vectorizer rely on exact or very close wording. Variants like “sweatshirt” vs “hoodie”, “slim” vs “skinny”, or small color/style differences can make relevant items look less similar.
  Fix: add stemming/lemmatization and limited synonym expansion.

- All fields weighted equally. We merged everything into combined_info, so title, description, and extra text contribute the same.
  Fix: build field-aware representation (e.g. 2× weight for title, 1× for description).

- Very strict relevance in 2.3. Using AND over all query terms is good for building a clear ground truth, but it can mark as non-relevant items that the user would actually accept.
  Fix: use AND for judging but allow OR / soft matching for ranking (hybrid Boolean + ranked retrieval).

- No query reformulation. Some of the bad results are because the query is too long or includes low-value terms (“full sleeve”, “cotton”).
  Fix: do simple query cleaning (remove very common modifiers) before retrieval.