IRWA-2025-part-3
Github repository: https://github.com/laiaatomas/IRWA_FinalProject
Laia Tomàs U198723 NIA252294
Quim Ribas U198742 NIA251754

FINAL PROJECT: REPORT PART 3

PART 1: TF-IDF, BM25, Our score

TF-IDF and BM25

The ranking of an information retrieval system depends strictly on the algorithm chosen for this operation. Each of the different approaches has its own advantages and disadvantages.

The *tf-idf* ranks documents by weighting terms according to their frequency in a document and their rarity in the corpus. Each document is represented as a vector where each dimension represents the score of a term in the vocabulary for that document. The similarity to the query is usually computed via cosine similarity. This makes ranking sensitive to the proportion of term usage within each document, but it does not take into account each document size. This length-invariance distorts the influence of each word on the score, since a term on a shorter document might get a higher score than the same term on a long document. However this approach is one of the simplest, fastest and easiest to interpret.

The *bm25* is a probabilistic ranking function that addresses the cons mentioned in the *tf-idf* algorithm by incorporating term-frequency saturation and explicit document-length normalization. This prevents very frequent terms from dominating and handles long documents more effectively. There is no vector representation, but a predefined function to compute the score of a document with respect to a query. As a trade off to a better performance than the *tf-idf*, it requires parameter tuning and is slightly more complex.

Our score

For this part, we designed a custom relevance score that extends the *BM25* scores by incorporating additional product-level attributes. The motivation is that, in a real search engine, textual similarity alone is not sufficient: users also care about ratings, discounts, price, and availability, and especially when searching for fashion products. Our formula boosts the *BM25* score for products that are in stock with a higher rating, higher discount and lower price.
Our formula produces a score for each product in respect to a query, and it is defined as follows

$$OURSCORE(pid) = \alpha * bm25(pid) + \beta * rating(pid) + \gamma * discount(pid) + \delta * (1 - price(pid)) + \epsilon * (1 - outofstock(pid)),$$

where bm25(pid) is the bm25 score for such product, and the other attributes reference the dataset attributes. As outofstock(pid) takes the value 0 when the product is available, we will boost the score only for available products. It is also worth mentioning that all attributes, including the bm25 score and excluding outofstock(pid) (as it is binary) were normalized to a scale from 0 to 1 to ensure comparability among products.

We created the formula above taking into consideration the following assumptions:

- Users typically prefer well-rated products, so higher rating improves ranking.

- Discounts are more attractive to users, so high-discount items receive a boost.
- Lower prices are often more appealing, so we invert and normalize the price.
- Out-of-stock products are penalized, because a product that cannot be bought should not appear near the top of a search result.

It is also worth noting that we chose *BM25* as the base score because it outperforms tf-idf as mentioned earlier.

Pros:

- More realistic results: Combines textual relevance with practical product attributes users actually care about.
- Flexible: We can manually adjust the weights (α,β,γ,δ,ε) depending on our needs. For example we might be more interested in giving a higher weight to the rating or to the low price.
- Avoids bad results: Prevents highly-relevant but out-of-stock or extremely expensive items from appearing at the top.

Cons:

- Requires manual tuning: Choosing good values for α,β,γ,δ,ε can be subjective and therefore, wrong weights might harm ranking quality.
- Assumptions not representative: The assumption, for example, that lower prices are more appealing might not reflect all users' preferences and therefore, the ranking would be less effective for user groups that don't align with such preferences.
- Relies heavily on *BM25*, so weaknesses of *BM25* (like not handling semantics) still remain.
- Ratings might not actually be representative of the product's true quality, since they can be biased.
- Price inversion is simplistic: A very cheap but low-quality product might be ranked too highly.

PART 2: Word2Vec + cosine similarity

Word2Vec is a model that represents words as dense vectors in a continuous space, where semantically similar words are located close together. By averaging the vectors of all words in a document, we can create a single vector representing the overall meaning of the document. To rank documents for a query, we compute a vector for the query in the same way and then measure the similarity between the query and document vectors using cosine similarity (as in the TF-IDF approach). In contrast to the other two traditional ranking algorithms used, this one allows the search to capture semantic relationships. It is efficient for large datasets and works well with short queries, but it ignores word order and syntax, and can be less interpretable.

Regarding the results obtained, we can compare them to the results achieved by the TF-IDF approach, which also uses cosine similarity to quantify the match between a query and a document.

For the TF-IDF results, as one would expect, they rely on exact keyword matches, favoring literal overlaps, while Word2Vec captures semantic similarity, allowing it to retrieve conceptually related items even if the words don't match exactly. In theory, Word2Vec should better handle queries like "round neck black dress" by recognizing "dress" as distinct from "t-shirt." However, in this dataset, the differences are minimal: most queries return very similar results for both methods. This likely reflects limitations such as limited product variety, and sparse descriptions, which reduce the advantage of Word2Vec's semantic understanding

PART 3: Better representation than Word2Vec

Unlike Word2Vec, which generates embeddings only for individual words, Sentence2Vec and Doc2Vec extend this idea to larger textual units: products' titles and descriptions in this case.

Sentence2Vec generates a single embedding for an entire query or sentence, capturing its overall meaning rather than isolated words. This is beneficial for semantic matching and fast to compute; however, it typically ignores word order and syntactic structure, which can lead to shallow representations that fail to capture nuances.

Doc2Vec, on the other hand, produces embeddings for full product titles and descriptions. This allows it to capture global context, topic similarity, and relationships across the entire text, making it more suitable for document-level retrieval tasks. Its disadvantages are increased computational cost and the need for a larger corpus to train stable vectors. In short, while Sentence2Vec and Doc2Vec enable more meaningful semantic comparisons than Word2Vec, they trade off interpretability and speed.

Analyzing the results, Doc2Vec outperforms Word2Vec because it generates embeddings for entire product descriptions, capturing full contextual meaning rather than just individual word semantics. In the first query "women blue casual tshirt," Word2Vec retrieved irrelevant items like black or yellow t-shirts, matching isolated words, whereas Doc2Vec returned items that were also blue and casual, reflecting the overall description context. While Word2Vec is faster and effective when keywords alone are sufficient, Doc2Vec handles synonyms, phrasing variations, and short descriptions better. However, limitations remain, because if product descriptions are very short or inconsistent, even Doc2Vec may struggle to distinguish subtle nuances.