Universitat Pompeu Fabra Barcelona

Pau Reig Vaqué - u215101
Marc Rodríguez Vitolo - u215105
Clara Serra Borràs - u204635

REPORT
Part 3: Ranking & Filtering

1. Three different ways of ranking: TF-IDF and cosine similarity, BM25 and our Score

In this part of the project, we have implemented three different ranking strategies to evaluate how the choice of scoring function affects the final ordering of documents retrieved under conjunctive (AND) querying. The three ranking methods implemented were: TF-IDF with cosine similarity, BM25, and a custom score.

For the first approach, TF-IDF combined with cosine similarity, we have reused the functions created and the ranking created in the part 2 of the project. Each document and each query is represented as a TF-IDF vector, where term frequency captures how often a word appears within a document, and inverse document frequency down-weights very common words across the corpus. Once TF-IDF vectors are created, we compute the cosine similarity between each document vector and the query vector. This measures how aligned the two vectors are, which corresponds to how similar their term distributions are.

To implement BM25 we have used the rank_bm25 Python library. First, we have transformed the dataframe column, which contains the key words of the document, into a matrix to have a list of documents containing the list of terms. This list of tokenized documents was then used to instantiate a BM25Okapi model, which pre-computes the necessary corpus statistics. The core of our retrieval is the get_top_n function. This function takes the BM25 model, the dataframe, a query string, and the desired number of results n. To enforce conjunctive (AND) query semantics, the function then iterates through all documents, identifying only those that contain every single query term. The function then extracts the top n documents with the highest BM25 scores from this filtered set, providing a ranked list of the most relevant products that satisfy the query.

As introduced in the theory, we know that TF-IDF is based on these two principles, inverse document frequency and term frequency. The BM25 ranking formula builds upon these principles and adds a third factor, document length normalization, which adjusts scores to account for differences in document length. This factor aims to reduce the bias towards longer documents that might naturally contain more query terms. To analyze the practical effect of this difference, we applied both ranking methods to the five testing queries defined in part 2. After that, we have observed notable differences in the ranking orders produced by the two algorithms, reflecting how document length influences the results.

By comparing the top-5 documents returned for each query, we observed that some queries produce overlaps between the two ranking methods (“women winter jacket”, “sports shoes running”, “casual footwear black”), while others show no overlap at all (“cotton t-shirt men”, “formal shirt slim fit”). This indicates that the two search engines can produce substantially different results depending on the query. These differences arise mainly from the characteristics of BM25. It reduces the excessive influence of repeated terms, preventing documents that repeat several terms very often from dominating the ranking, and its document length normalization favors smaller documents.

Pau Reig Vaqué - u215101
Marc Rodríguez Vitolo - u215105
Clara Serra Borràs - u204635

Each method has its own pros and cons. The main advantage of TF-IDF with cosine similarity is its simplicity, it is easy to implement, intuitive to understand, and does not require document length information. However, its main drawback is the absence of explicit document length normalization. As a result, longer documents may receive higher scores even if they are not truly more relevant. In addition, because term frequency increases linearly, documents that repeat a query term many times can be disproportionately boosted in the ranking.

In contrast, BM25 applies document length normalization, which prevents long documents from receiving disproportionately high scores and ensures a fairer comparison between documents of different lengths. This generally leads to more robust and reliable ranking results. However, BM25 is a more complex model than TF-IDF, and its scoring is less interpretable because it does not rely on a vector model.

Finally, we have designed a custom scoring (that we have named “pepito score”) that uses tf-idf and other features of the dataset to compute the ranking. This new score is a linear combination of tf-idf with cosine similarity, the discount and the average rating. We also applied a penalization for not being in stock. The factors can be modified but we have assigned some default values as follows: 40% to the cosine similarity score and both 30% to discount and average rating. The penalty for not being in stock reduces the score to a 10% of the original score. The way it works is very similar to the tf-idf functions. The structure is the same but after computing the cosine similarity we input the values into the new formula so we get the new score. We chose to use these fields as we considered them to be the most relevant when people are searching for products. We decided to keep the price aside, because cheaper does not always mean more relevant, so we focused only on the discount, as discounted items will always be attractive. We also gave weight to the rating, so well rated products get pushed to the top. The penalty for not being out of stock is applied so the ranking show products that are available, and will show products out of stock only in the case that it does not find better items available.

This choice of features to use also implies some cons, mainly with the usage of the rating. First of all if new items would like to be added, as they don’t have ratings, it might need to use another type of score so they are not pushed back in the ranking. And also if products have just 1 rating of 5 stars, it might get a better score than popular items with a more averaged rating that will probably not be as high. Despite these disadvantages we decided to include it anyway as it might be relevant, and for now, we don’t have a way to see how many times this item has been sold or how popular it is.

Universitat Pompeu Fabra Barcelona

Pau Reig Vaqué - u215101
Marc Rodríguez Vitolo - u215105
Clara Serra Borràs - u204635

2. Word2vec and cosine ranking score

To implement the Word2vec + cosine similarity ranking approach, we have followed several steps. First, we have trained a Word2vec model using the cleaned_title_description_extra_fields column of the DataFrame, which contains the tokenized and cleaned text for each document. Then, we have generated a single vector representation for each document in the DataFrame by averaging the Word2vec embeddings of all its cleaned terms, and we have obtained a dictionary where the keys are PIDs and the values are their corresponding vectors. After that, we have defined the function rank_documents_word2vec_cosine, which computes the cosine similarity between the query vector and the document vectors, and ranks only those documents that contain all query terms. In addition, we have defined the function search_word2vec_cosine, which preprocesses the query, generates its Word2vec vector representation, filters the DataFrame so that it retains only the documents containing all query terms in the document dictionary, and then calls the ranking function. Finally, we have executed the five test queries using this ranking approach, and we displayed the top 20 results (PID) returned for each query.

3. Can you imagine a better representation than word2vec?

Word2vec provides a useful way of representing text as dense vectors and works reasonably well in our case. However, we think that there are several alternatives that can capture semantic information more accurately. One limitation of Word2vec is that it generates embeddings at the word level and represents an entire document by simply averaging its word vectors. For instance, Word2vec cannot distinguish between different usages of the same word depending on the surrounding context.

We think that a better approach for representing full documents could be Doc2vec, which extends Word2vec by learning embeddings not only for words but also for entire documents. The main advantage of Doc2vec is that it directly models information on document level during training. Therefore, it is trained to predict the words within its specific document, allowing it to capture the overarching topic more effectively. However, Doc2vec also has certain drawbacks: it requires more training data and it is more computationally expensive. In practice, this could mean that achieving stable and high-quality document vectors may require more effort than with a basic Word2vec model.

Another alternative could be Sentences2vec, which generates fixed length vectors for entire sentences using neural architectures that take context into account. Unlike Word2vec, models like BERT-based Sentence Transformers create contextualized embeddings, meaning the representation of a word changes based on the other words in the sentence. This allows them to understand that the word "light" has a different meaning in "light jacket" versus "light blue shirt."

Overall, both Doc2vec and Sentence2vec offer more expressive representations than averaging Word2vec embeddings. They incorporate contextual and structural information that Word2vec cannot capture, potentially leading to more accurate relevance ranking. However, these advantages come at the cost of increased complexity, computational load, and training requirements, so the best choice depends on the characteristics of the dataset and the goals of the retrieval system.

Universitat Pompeu Fabra Barcelona

Pau Reig Vaqué - u215101
Marc Rodríguez Vitolo - u215105
Clara Serra Borràs - u204635

In our project we think that Sentence2vec would be the best option because our product titles and descriptions are short and well-structured pieces of text, so contextual meaning would be highly valuable for distinguishing between similar fashion products.

Another significant consideration is how these models handle words that were not present in their training data. If a user's query contains a new term not found in our dataset, the model cannot generate a vector for it, effectively ignoring that word. This can lead to a failure in capturing the user's full intent. A practical solution to this issue is to preprocess rare or unseen words using tokenization methods.

GitHub URL: https://github.com/mmarcrv/irwa-search-engine-G005
TAG: IRWA-2025-part-3
AI Usage: We used ChatGPT and Gemini to review our draft explanation of pre-processing decisions and to help us to make some code lines.