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

Report IRWA Part 3

Overview

In this part of the project we implemented a retrieval pipeline that combines strict conjunctive filtering (AND semantics) with several ranking functions. The system uses the preprocessed text fields clean_title and clean_description, concatenates them into a single full_text field, and tokenizes them with a simple regex-based tokenizer. We then build an inverted index over full_text and use it to enforce that a document is a candidate only if it contains all query terms.

On top of this conjunctive candidate set we implemented four ranking methods:

- TF-IDF + cosine similarity
- BM25
- A custom score combining BM25 with product metadata (average_rating, discount_percent, out_of_stock)
- A Word2Vec-style semantic method using pretrained GloVe embeddings

We evaluated these methods on the following queries:

- women blue cotton tshirt
- men black jeans slim fit
- cotton round neck sweatshirt
- women red dress long sleeve
- men leather jacket brown
- kids white sneakers

The key empirical observation is that only two queries, "women blue cotton tshirt" and "cotton round neck sweatshirt", have any documents that satisfy the conjunctive filter. The other four queries retrieve zero documents for all ranking methods. This is a combined effect of the strict AND requirement and the fact that the underlying text was normalized in earlier preprocessing. The analysis below is based on the rankings produced for these queries and on the empty results for the remaining ones.

1. Conjunctive pipeline and ranking functions (TF-IDF, BM25, custom score)

Implementation

We first created a full_text field by concatenating clean_title and clean_description, filling missing values with empty strings. We used a simple tokenizer that lowercases the text and extracts alphabetic tokens via a regular expression. Using these tokens, we built an inverted index mapping each token to the set of document indices in which it appears.

The conjunctive filter is implemented by the function and_filter(query):

- The query is tokenized using the same tokenizer as the documents.
- For each token that appears in the inverted index, we retrieve its postings list.
- If any query token is missing from the index, we immediately return an empty candidate set.
- Otherwise we take the intersection of all postings lists to obtain the set of documents that contain every query term.
- Each ranking method only scores and ranks this candidate set.

TF-IDF ranking with cosine similarity

We used scikit-learn's TfidfVectorizer on full_text, with our tokenizer and English stopwords. The resulting matrix represents each document in TF-IDF space. For a given query, we:

- Apply and_filter(query) to get the candidate document indices.
- Transform the raw query string into its TF-IDF vector using the same vectorizer.
- Extract the TF-IDF vectors of the candidate documents.
- Compute cosine similarity between the query vector and each candidate document vector.
- Sort candidates by decreasing similarity and keep the top 20.

Observed behavior:

- When there are candidates (for "women blue cotton tshirt" and "cotton round neck sweatshirt"), TF-IDF gives high scores to documents whose titles closely match the query terms, for example blue t-shirts for women and cotton sweatshirts with round necks.
- The top-ranked documents tend to be those with short, focused titles where the query terms are prominent.

- TF-IDF scores are relatively compressed; differences between top documents are small, which can make ranking ties more frequent.

BM25 ranking

We used rank_bm25's BM25Okapi implementation over the same tokenized full_text collection. For a given query, we:

- Apply and_filter(query) to get the candidate indices.
- Tokenize the query.
- Compute BM25 scores for all documents, then restrict them to the candidate indices.
- Sort by decreasing score and take the top 20.

Observed behavior:

- BM25 and TF-IDF largely agree on which documents are relevant for the two non-empty queries, but BM25 provides better separation in scores.
- BM25 handles repeated query terms and longer descriptions more gracefully, so that documents are not over-penalized for length or over-rewarded for excessive term repetition.
- For "women blue cotton tshirt", BM25 and TF-IDF both rank the same blue t-shirt documents highly, but BM25 spreads out the scores more clearly.

Custom score (YourScore)

The custom scoring function starts from BM25 and incorporates three numerical fields:

- average_rating
- discount_percent
- out_of_stock

For each candidate document:

- We compute the BM25 score for the query.
- We extract the average_rating and discount_percent, filling missing values with zeros.
- We convert out_of_stock to an integer and treat it as a penalty term.
- We normalize rating to [0,1] by dividing by 5 and discount_percent to [0,1] by dividing by 100.
- The final score is a weighted sum of BM25, normalized rating, normalized discount, and a negative term for out_of_stock. The weights used were w_text = 1.0, w_rating = 0.3,

w_discount = 0.1 and w_stock = 0.5.

Observed behavior:

- For the two queries with results, YourScore typically keeps the same family of top documents as BM25, but reorders them according to rating, discount, and stock status.
- High-rated and highly discounted products move up in the ranking, while out-of-stock items are penalized and move down.
- For example, products with high average_rating (such as certain blue t-shirts) appear higher with YourScore than with pure TF-IDF or BM25, even when their lexical similarity is similar to that of other candidates.
- This leads to rankings that are more realistic for an e-commerce setting, where text relevance is necessary but not sufficient.

Comparison of the three methods

- All three methods operate on exactly the same candidate set given by the AND filter. They differ only in how they rank those candidates.
- TF-IDF provides a simple lexical baseline but is less robust to document length and term frequency saturation.
- BM25 improves upon TF-IDF by moderating the impact of term frequency and document length, and generally yields more stable and interpretable rankings.
- The custom score YourScore preserves BM25's lexical strengths and enhances rankings with product metadata, producing results that better reflect user preferences for rating, discount, and availability.
- The empty results for four of the six queries are shared by all three methods, because the conjunctive filter eliminates all documents before scoring when at least one query term does not occur in the indexed vocabulary in the required form.

2. <u>Word2Vec ranking for the queries</u>

Implementation

For the semantic approach we used gensim’s pretrained GloVe model “glove-wiki-gigaword-100”, which provides 100-dimensional vectors for a large English vocabulary and behaves similarly to Word2Vec in terms of usage. The steps were:

- Load the GloVe model via gensim.downloader.
- Define a function doc_vector(text) that:
    - tokenizes the text,
    - collects the vector for each token that exists in the model vocabulary,
    - returns the average of these vectors, or a zero vector if no tokens are found.
- Precompute doc_embeddings for every document in the collection by applying doc_vector to full_text.
- For a given query:
    - Apply and_filter(query) to get candidate indices.
    - Compute the embedding of the raw query string using doc_vector.
    - Extract the precomputed embeddings of the candidate documents.
    - Compute cosine similarity between the query embedding and each candidate embedding.
    - Sort by decreasing similarity and keep the top 20.

Observed behavior on our queries:

- As with the other methods, Word2Vec is restricted to documents that pass the conjunctive filter. It cannot retrieve documents for queries where the candidate set is empty.
- **For “women blue cotton tshirt”:**
    - All four methods retrieve 20 documents.
    - TF-IDF, BM25, and YourScore agree that a particular product (for example, doc 8766, a printed women blue t-shirt) is the top result, with other blue t-shirts appearing in the top positions.
    - Word2Vec produces a noticeably different ranking: it promotes other blue t-shirts with similar semantics, such as v-neck or polo-neck variants, to the top positions, even if

    their lexical overlap is not maximal.

    - The document that is top-1 for TF-IDF and BM25 still appears high in the Word2Vec ranking, but not necessarily as the top result.

- **For “cotton round neck sweatshirt”:**
    - Again, all four methods retrieve 20 documents.
    - TF-IDF, BM25, and YourScore rank several full-sleeve solid sweatshirts at the top.
    - Word2Vec emphasizes items that are semantically close to sweatshirts and cotton apparel, including some graphic print sweatshirts and packs of t-shirts that share similar stylistic or contextual features in the embedding space.

- **For the remaining four queries:**
    - “men black jeans slim fit”
    - “women red dress long sleeve”
    - “men leather jacket brown”
    - “kids white sneakers”
      all four methods (TF-IDF, BM25, YourScore, and Word2Vec) return no documents. The intersection of postings lists is empty, so Word2Vec never gets any candidates to rank. This shows that the bottleneck is the conjunctive filter rather than the ranking model.

Summary of Word2Vec behavior

- Word2Vec can reorder candidate documents based on semantic similarity, highlighting related products that may not be lexically closest to the query.
- It is still constrained by the AND filter: if the query terms (after tokenization and earlier preprocessing) do not collectively appear in any document, no ranking, including Word2Vec, can return results.
- For queries with candidates, Word2Vec’s ranking differs from the lexical methods in a plausible way, capturing semantic clusters such as different blue t-shirt variants or related cotton garments.

3. <u>Better representations than Word2Vec</u>

Word2Vec-style embeddings provide a useful improvement over purely lexical models, but they also have significant limitations in this setup:

- They treat each word as having a single vector, regardless of context or sense.
- Document representations are simple averages of word vectors, which ignore word order and syntactic structure.
- Important distinctions can be lost, especially for longer documents or more complex queries.

There are several more advanced representation approaches that could overcome these limitations.

Doc2Vec

Pros:

- Learns embeddings directly for entire documents, rather than averaging word embeddings.
- Can capture document-level themes or topics better than simple averaging.
- Makes it easier to compare documents as holistic units.

Cons:

- Requires training on a large, relevant corpus; there is no universal Doc2Vec model comparable to GloVe.
- Training can be unstable and sensitive to hyperparameters and initialization.
- In practice, it is often outperformed by more recent neural architectures.

Sentence embeddings (for example Sentence-BERT)

Pros:

- Provide context-aware embeddings at the level of sentences or short texts.
- Achieve strong performance on semantic similarity and retrieval tasks.
- Many high-quality pretrained models are available and easy to use.
- Handle variations in phrasing better than Word2Vec.

Cons:

- More computationally expensive than simple word embeddings.
- For long product descriptions, it is necessary to combine multiple sentence embeddings or summarize the text.
- Still less expressive than running a full transformer model over entire documents and queries.

Transformer-based models (for example BERT, MiniLM, DistilBERT)

Pros:

- State-of-the-art contextual representations for natural language processing.
- Capture nuance, context, and polysemy by assigning different representations to the same word in different contexts.
- Very strong performance for modern retrieval and ranking tasks, especially when used to encode both queries and documents.

Cons:

- Computationally expensive, especially for large document collections.
- Require more memory and careful engineering to run efficiently in production.
- More complex to integrate into a traditional inverted-index pipeline.

Overall, there are clear alternatives that can represent documents and queries more effectively than Word2Vec in our setting. Doc2Vec improves document-level representation, sentence embeddings provide robust semantic similarity for shorter texts, and transformer-based encoders offer the richest and most flexible contextual representations. If computational resources allow, using a sentence embedding model or a lightweight transformer encoder to represent queries and documents would likely improve the quality of retrieval beyond what is achievable with Word2Vec-style averaging.

Conclusion

The implementation in this part of the project fulfills the requirements of conjunctive retrieval combined with multiple ranking functions. TF-IDF provides a simple lexical baseline; BM25 improves ranking quality through better term frequency and length handling; the custom score incorporates business-relevant metadata to produce more realistic e-commerce rankings; and Word2Vec introduces semantic similarity on top of the same candidate sets.

The experiments with our six queries show both the strengths of these models and the limitations imposed by strict AND filtering and earlier preprocessing. Finally, we have identified more advanced representation methods, such as Doc2Vec, sentence embeddings, and transformer-based models, that could further enhance retrieval performance in future extensions of this system.