Astrid Alins, Miquel Bisbe, Cinta Carot

Part III: Ranking & Filtering

Github link: https://github.com/astridalins/IRWA_final_project.git

Tag name: **IRWA-2025-part-3**

Important: For this assignment, we only consider conjunctive queries (AND). This means that a document is included in the results only if it contains every word from the query.

PART 1: 3 different ways of ranking

You're asked to provide 3 different ways of ranking:

- ***TF-IDF + cosine similarity: Classical scoring, which we have also seen during the practical labs***
- ***BM25***
- ***Your Score: Here, the task is to create a new score. (Be creative , think about what factors could make a document more relevant to a query and include them in your formula.)***

Explain how the ranking differs when using TF-IDF and BM25, and think about the pros and cons of using each of them. Regarding your own score, justify the choice of the score (pros and cons).

For all the implemented ranking methods (TF-IDF, BM25, and our custom score), the calculation is performed on a **combined set of the document's title and description**. This approach allows us to correctly handle **conjunctive (AND) queries**, ensuring that only documents containing **all the query terms** are included in the results, regardless of whether the terms appear in the title or the description. By combining both fields, we increase the coverage of relevant documents and avoid missing important information that might be present only in one part of the document. This ensures that the ranking reflects truly relevant and complete documents for each query, improving the quality of the retrieved results.

1.1 TF-IDF + cosine similarity

This ranking method uses the classical vector space model, where both the query and each document are represented as TF-IDF vectors. After tokenizing the query, we compute the document frequency (DF) and corresponding IDF values for all query terms.

To satisfy the assignment's AND semantics, we intersect the posting lists of all query terms and keep only documents that contain every term (either in title or description). If any term is missing, the result set becomes empty.

For each remaining document, we build a TF-IDF vector based on the term frequencies extracted from the inverted index. The query vector uses a uniform TF of 1. Relevance is then estimated using cosine similarity, which measures the angular distance between the query vector and each document vector while normalizing for document length.

Finally, documents are sorted by their similarity scores in descending order, producing a ranked list of relevant results under this traditional IR scoring model.

1.2 BM25

BM25 is a ranking formula used to estimate how relevant a document is to a given search query. It's a variation of the classic TF-IDF but with some adjustments to improve its performance. BM25 considers how often query terms appear in a document (term frequency), how rare those terms are across all documents (inverse document frequency), and the length of the document compared to the average document length. Documents that contain the query terms frequently, have rare terms, and are of moderate length tend to get higher BM25 scores and are ranked higher in the search results. It's designed to give good results for a wide range of queries without needing complex tuning.

1.3 Your Score: Hybrid Ranking Function

In addition to the classical TF-IDF and BM25 approaches, we designed a custom scoring function that combines the strengths of both methods while introducing an additional "rarity" factor to reward documents containing less frequent (and thus potentially more informative) terms.

Definition

Our proposed score, called the **Hybrid Score**, is defined as:

$$\text{Score}(d, q) = \alpha \cdot \text{TF-IDF}(d, q) + (1 - \alpha) \cdot \text{BM25}(d, q) + \lambda \cdot \text{Rarity}(q)$$

where:

- **TF-IDF(d, q)** captures the importance of query terms within each document.

- **BM25(d, q)** adjusts for document length and term saturation, providing a more balanced weighting.
- **Rarity(q)** measures how uncommon the query terms are across the corpus, giving extra credit to documents that contain rare but potentially more relevant words.
- **α** controls the balance between TF-IDF and BM25 (in our implementation, α = 0.6).
- **λ** is a small weighting factor (λ = 0.05) for the rarity term.

Motivation

The main motivation behind this score is to **combine frequency-based relevance with document-level normalization and a notion of term uniqueness**.
In our project, queries are short and focused, and the corpus contains documents of varying lengths. TF-IDF alone can overemphasize longer documents, while BM25 can sometimes undervalue documents with rare but important terms. By merging both and adding a rarity adjustment, we aim to achieve a more stable and semantically meaningful ranking.

Detailed Comparison of the Three Factors

Each component of the Hybrid Score contributes a distinct aspect of relevance to the final ranking. **TF-IDF** emphasizes the importance of query terms that appear frequently within a document but are rare across the corpus, making it especially effective for identifying documents that strongly focus on the main topic. However, it does not normalize for document length and can therefore favor longer texts.

BM25, on the other hand, refines this by introducing term frequency saturation and document length normalization, ensuring that long documents with repeated words do not dominate the ranking. This makes BM25 more balanced and robust for corpora with documents of varying sizes.

Finally, the **Rarity** factor adds an additional layer of discrimination by rewarding documents that contain uncommon yet potentially informative terms, which might otherwise be undervalued by standard frequency-based methods. Together, these three elements provide complementary perspectives on relevance — local term importance (TF-IDF), structural normalization (BM25), and global term uniqueness (Rarity) — resulting in a hybrid score that captures both the statistical and semantic nuances of document retrieval.

Why this score fits our project

In the context of our information retrieval system, which deals with **conjunctive queries (AND)** and documents that may vary considerably in length and vocabulary, the hybrid score provides a more robust way to compare document relevance:

- It maintains the interpretability of TF-IDF and BM25 while combining their complementary strengths.
- It adapts well to the diversity of document lengths in our dataset.
- It favors documents containing distinctive vocabulary, which is often more informative for specific queries.
- It remains computationally efficient and easy to implement using our existing inverted index structure.

Overall, this hybrid approach enhances retrieval quality by balancing **term frequency, document normalization, and term rarity**, making it particularly suitable for our project's goal of ranking documents accurately in response to conjunctive queries.

1.4 TF-IDF vs. BM25 Ranking

- **TF-IDF** excels in simplicity and interpretability but lacks sophisticated frequency saturation.
- **BM25** provides better handling of document length variation and prevents over-emphasis of high-frequency terms through non-linear term frequency scaling.

In practice, BM25 generally delivers more robust rankings across diverse document collections, while TF-IDF remains valuable for its transparency and computational efficiency.

In our case, when comparing the results of our functions for the query *"print shirt"*, the difference in ranking behavior becomes clear. TF-IDF cosine returns many documents with identical scores (1.0), including numerous items that do not explicitly emphasize "print" but match the terms in a minimal way. This happens because cosine normalization flattens score differences when documents contain the query terms at least once, making long and short documents appear equally relevant.

In contrast, BM25 produces a sharper and more discriminative ranking: it prioritizes documents where "print" and "shirt" appear with higher term frequency and within more informative contexts (e.g., "Printed Women Round Neck Blue T-Shirt"), while penalizing longer documents and giving less weight to terms that appear only incidentally. Consequently, BM25 retrieves results that are semantically more aligned with the intent of the query, especially for product-style datasets where keyword prominence matters.

Astrid Alins, Miquel Bisbe, Cinta Carot

PART 2: Implement **word2vec + cosine** ranking score.

Return a top-20 list of documents for each of the 5 queries defined in the Part 2 of your project, using search and word2vec + cosine similarity ranking.

To represent a piece of text using word2vec, we create a single vector that represents the entire text. This vector has the same number of dimensions as the word vectors and is calculated by averaging the vectors of all words in the text.

In this part of the project, we implemented a **document retrieval pipeline** using **Word2Vec embeddings** combined with **cosine similarity** to rank documents according to query relevance. The pipeline follows **AND semantics**, meaning that only documents containing all query terms are considered as candidates, as required by the assignment.

Implementation

The implementation consists of several stages, as shown in **Code**. First, auxiliary functions were defined to handle the conversion of text to vectors (text_to_w2v_vector) and to compute cosine similarity between two vectors (cosine_similarity). The text_to_w2v_vector function represents a piece of text as the **average of all its word vectors**, producing a single fixed-size vector regardless of text length. Words missing from the model default to zero vectors, ensuring robust behavior. Cosine similarity is then used to measure the semantic closeness between the query vector and document vectors.

Next, we constructed the **Word2Vec model** itself. A corpus was created by combining title_tokens and desc_tokensfrom all documents. The model was trained using a 100-dimensional skip-gram configuration with a context window of 5 (Word2Vec in **Code**). This allows words appearing in similar contexts to have similar embeddings, capturing semantic relationships that go beyond exact string matches.

The main ranking function, rank_query_word2vec, implements the retrieval pipeline:

1. **AND semantics:** The query is split into tokens (without preprocessing), and only documents containing all query terms are considered.
2. **Vector representation:** The query and each candidate document are converted into Word2Vec vectors.
3. **Scoring:** Cosine similarity is calculated between the query vector and each document vector.
4. **Sorting and deduplication:** Documents are sorted by similarity score, and duplicates based on titles are removed, returning the top 20 results.

Overall, the **Word2Vec + cosine similarity approach** captures both exact term matches and semantic relationships, producing a more flexible and meaningful ranking of documents

compared to strict lexical matching. The combination of AND semantics with semantic scoring ensures that retrieved results are both relevant and contextually appropriate, as reflected in the top-20 results for all test queries.

PART 3: Can you imagine a better representation than word2vec? Justify your answer.

While Word2Vec provides dense vector representations that capture semantic similarity between individual words, it has some important limitations when used for document retrieval tasks. In particular, Word2Vec represents each word independently and ignores word order and context. As a result, it cannot fully capture the meaning of longer texts such as sentences or documents.

A better alternative could be Doc2Vec (Paragraph Vector), which extends Word2Vec to represent entire documents as vectors rather than averaging word embeddings.

Advantages of Doc2Vec:
It captures the context and semantics of a whole document, not just isolated words.
It allows documents with similar meaning but different wording to have similar representations.
It's better suited for information retrieval tasks, since we can directly compare query and document embeddings.

However, Doc2Vec also has drawbacks:
It requires more training data and computational resources.
It can be slower to train and harder to fine-tune.
The learned representations may not generalize well to unseen documents unless the model is carefully trained.

Another improvement could be Sentence2Vec or Transformer-based models (like BERT, SBERT), which produce contextual embeddings — meaning that the same word can have different representations depending on the sentence. These methods often achieve much better retrieval performance because they understand the meaning of the whole query in context.

In summary:
Compared to Word2Vec, models like Doc2Vec or Sentence-BERT provide richer, context-aware representations that can significantly improve retrieval quality, at the cost of higher computational complexity.

In the context of our project, Doc2Vec could be particularly suitable because our goal is to retrieve entire documents based on their semantic similarity to user queries. Since each document in our corpus contains multiple sentences and complex ideas, representing them as single embeddings allows for a more meaningful comparison than averaging individual word vectors, as done in Word2Vec. This would likely lead to more accurate retrieval results, especially when different documents use distinct wording to express similar concepts.