UPF – Information Retrieval and Web Analytics

Final Project Part 3

Jhonatan Barcos Gambaro (u198728)

Daniel Alexander Yearwood Agames (u214976)

Part 3: Ranking & Filtering

1. Delivery information

Firstly, relevant information regarding the delivery of the project is recorded.

Link Github: https://github.com/laxhoni/IRWA_Final_Project

TAG: part-3-submission:

https://github.com/laxhoni/IRWA_Final_Project/releases/tag/part-3-submission

2. Introduction

In this third part of the project we focus on ranking and filtering. Starting from the index and search engine implemented in Parts 1 and 2, we implement and compare several ranking functions (TF-IDF, BM25, a hybrid score, and a Word2Vec + cosine ranking). The main goal is to study how these models reorder the same set of candidate documents and to discuss their advantages and limitations on our fashion products dataset.

3. TF-IDF Ranking

First, we have compiled all the necessary base implementation from parts 1 and 2 of the project. In particular, we would like to highlight the following functions:

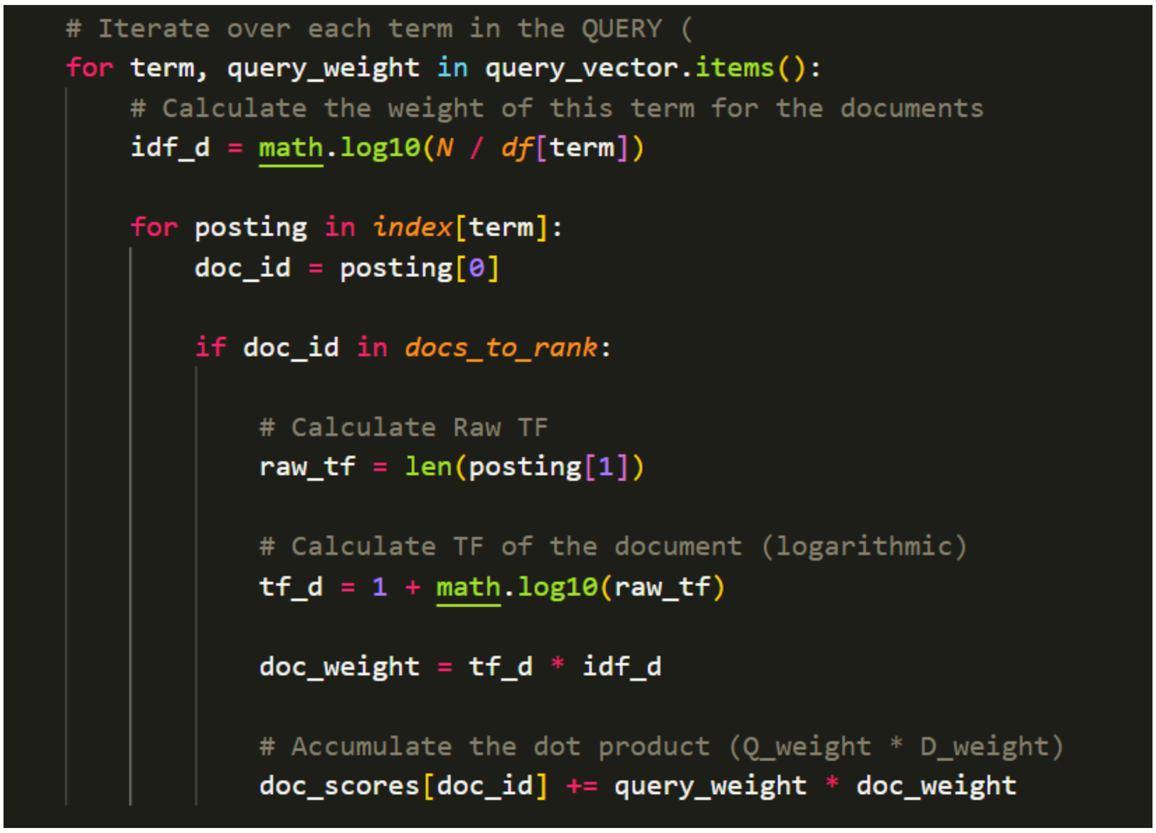
- **find_candidate_docs()**

  Which is identical to our search function from part 2, which implements AND logic and returns a set of candidate documents to implement the ranking on them.

- **create_index_part3()**

  An adaptation of our create_index_tfidf() function from part 2. Initially, this only stored the normalised TF, but for this implementation, in which we will use the BM25 algorithm, we need raw data (raw TF) and the length of the document (doc_len), so these characteristics have been added to our new function.

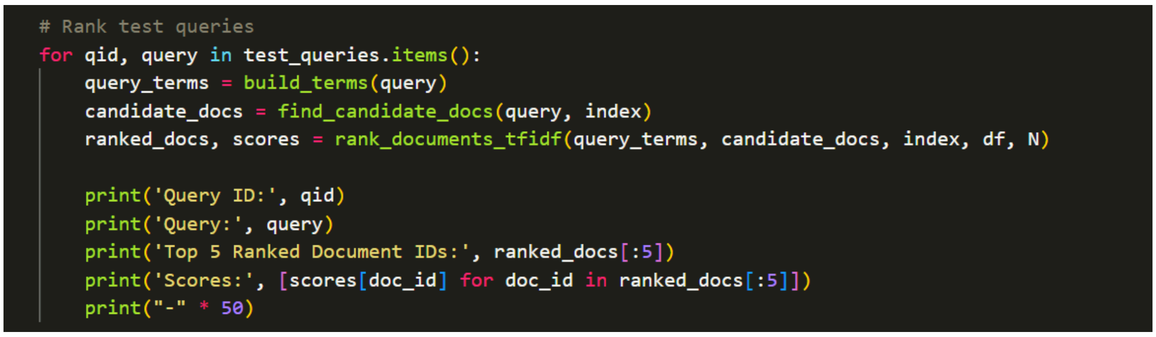
Finally, the rank_documents_tfidf() function has been implemented, which ranks candidate documents using the TF-IDF model calculated as the dot product.



```python
    # Iterate over each term in the QUERY (
    for term, query_weight in query_vector.items():
        # Calculate the weight of this term for the documents
        idf_d = math.log10(N / df[term])

        for posting in index[term]:
            doc_id = posting[0]

            if doc_id in docs_to_rank:

                # Calculate Raw TF
                raw_tf = len(posting[1])

                # Calculate TF of the document (logarithmic)
                tf_d = 1 + math.log10(raw_tf)

                doc_weight = tf_d * idf_d

                # Accumulate the dot product (Q_weight * D_weight)
                doc_scores[doc_id] += query_weight * doc_weight
```

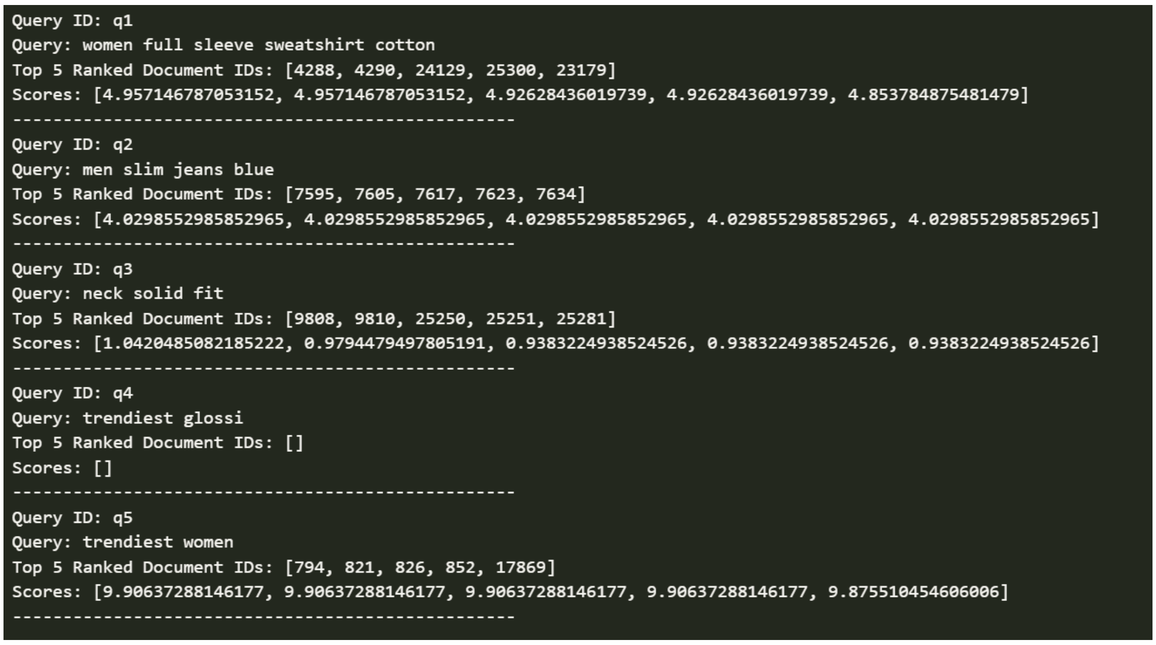
It is worth noting how this function calculates the dot product (numerator of Cosine Similarity) by omitting the final division by the vector norms to maximise the performance of our algorithm.

Once our ranking function has been implemented, we have iterated over our test_queries, defined in part 2 of the project and retrieved to test our ranking functions.



```python
# Rank test queries
for qid, query in test_queries.items():
    query_terms = build_terms(query)
    candidate_docs = find_candidate_docs(query, index)
    ranked_docs, scores = rank_documents_tfidf(query_terms, candidate_docs, index, df, N)

    print('Query ID:', qid)
    print('Query:', query)
    print('Top 5 Ranked Document IDs:', ranked_docs[:5])
    print('Scores:', [scores[doc_id] for doc_id in ranked_docs[:5]])
    print("-" * 50)
```

Once the ranking has been executed on test_queries, we obtain the following result:



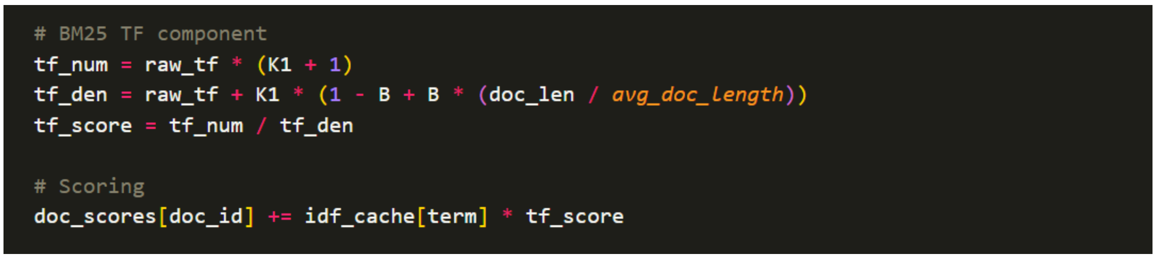
```
Query ID: q1
Query: women full sleeve sweatshirt cotton
Top 5 Ranked Document IDs: [4288, 4290, 24129, 25300, 23179]
Scores: [4.957146787053152, 4.957146787053152, 4.92628436019739, 4.92628436019739, 4.853784875481479]
--------------------------------------------------
Query ID: q2
Query: men slim jeans blue
Top 5 Ranked Document IDs: [7595, 7605, 7617, 7623, 7634]
Scores: [4.0298552985852965, 4.0298552985852965, 4.0298552985852965, 4.0298552985852965, 4.0298552985852965]
--------------------------------------------------
Query ID: q3
Query: neck solid fit
Top 5 Ranked Document IDs: [9808, 9810, 25250, 25251, 25281]
Scores: [1.0420485082185222, 0.9794479497805191, 0.9383224938524526, 0.9383224938524526, 0.9383224938524526]
--------------------------------------------------
Query ID: q4
Query: trendiest glossi
Top 5 Ranked Document IDs: []
Scores: []
--------------------------------------------------
Query ID: q5
Query: trendiest women
Top 5 Ranked Document IDs: [794, 821, 826, 852, 17869]
Scores: [9.90637288146177, 9.90637288146177, 9.90637288146177, 9.90637288146177, 9.875510454606006]
--------------------------------------------------
```

Note that in the case of Q4 'trendiest glossi', it does not return any candidate documents or their scores since, as we saw in part 2, there are no results for documents that meet both terms, as they have a very low frequency and AND logic is being implemented.

4. BM25 Ranking

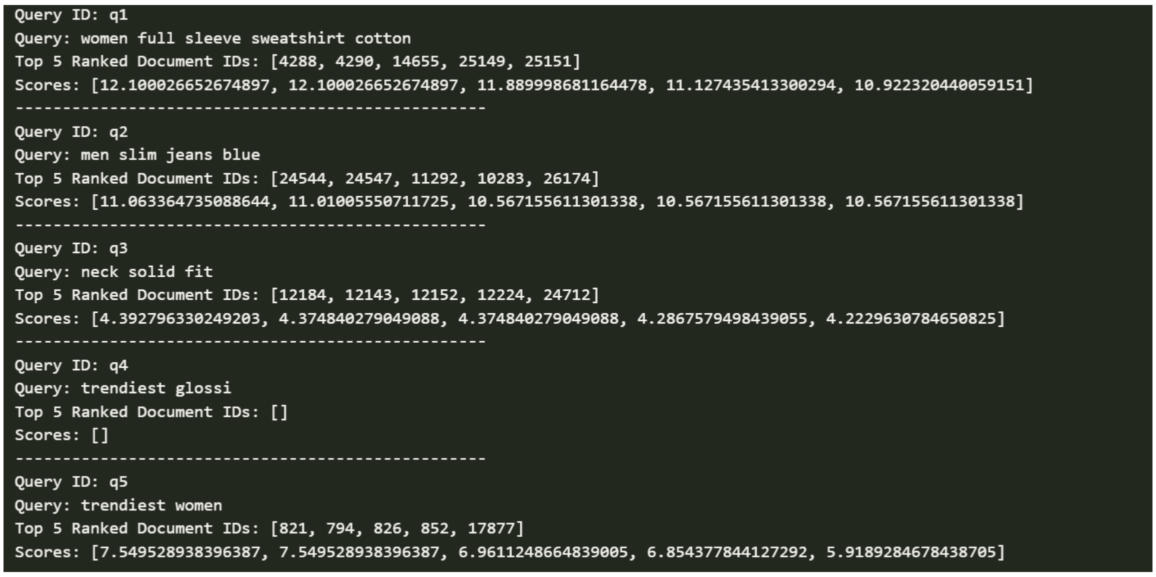
We implemented the algorithm BM25, which improves upon TF-IDF in two key ways:

- TF Saturation: Controlled by the parameter K=1 This prevents a term appearing 10 times from being 10 times more relevant than one appearing once.
- Length Normalization: Controlled by the parameter B=0.75. This penalizes documents that are much longer than the average (avg_doc_length).



```
# BM25 TF component
tf_num = raw_tf * (K1 + 1)
tf_den = raw_tf + K1 * (1 - B + B * (doc_len / avg_doc_length))
tf_score = tf_num / tf_den

# Scoring
doc_scores[doc_id] += idf_cache[term] * tf_score
```

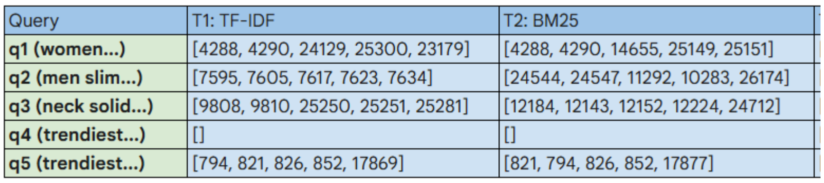
Once our ranking function has been implemented, we have iterated over our test_queries, defined in part 2 of the project and retrieved to test our ranking functions. Once the ranking has been executed on test_queries, we obtain the following result:



```
Query ID: q1
Query: women full sleeve sweatshirt cotton
Top 5 Ranked Document IDs: [4288, 4290, 14655, 25149, 25151]
Scores: [12.100026652674897, 12.100026652674897, 11.889998681164478, 11.127435413300294, 10.922320440059151]
--------------------------------------------------
Query ID: q2
Query: men slim jeans blue
Top 5 Ranked Document IDs: [24544, 24547, 11292, 10283, 26174]
Scores: [11.063364735088644, 11.01005550711725, 10.567155611301338, 10.567155611301338, 10.567155611301338]
--------------------------------------------------
Query ID: q3
Query: neck solid fit
Top 5 Ranked Document IDs: [12184, 12143, 12152, 12224, 24712]
Scores: [4.392796330249203, 4.374840279049088, 4.374840279049088, 4.2867579498439055, 4.2229630784650825]
--------------------------------------------------
Query ID: q4
Query: trendiest glossi
Top 5 Ranked Document IDs: []
Scores: []
--------------------------------------------------
Query ID: q5
Query: trendiest women
Top 5 Ranked Document IDs: [821, 794, 826, 852, 17877]
Scores: [7.549528938396387, 7.549528938396387, 6.9611248664839005, 6.854377844127292, 5.9189284678438705]
```

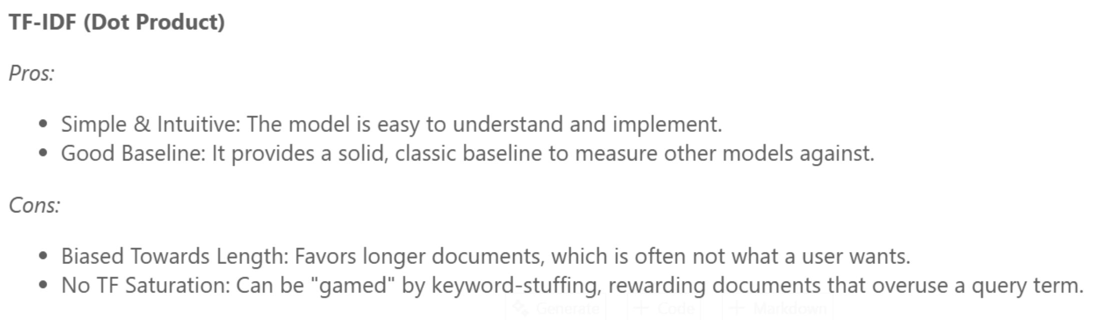
5. TF-IDF and BM25 Comparison

Firstly, we have compiled the results defined above regarding the ranking of the top documents ordered from highest to lowest score for the different implementations:



Query	T1: TF-IDF	T2: BM25
q1 (women...)	[4288, 4290, 24129, 25300, 23179]	[4288, 4290, 14655, 25149, 25151]
q2 (men slim...)	[7595, 7605, 7617, 7623, 7634]	[24544, 24547, 11292, 10283, 26174]
q3 (neck solid...)	[9808, 9810, 25250, 25251, 25281]	[12184, 12143, 12152, 12224, 24712]
q4 (trendiest...)	[]	[]
q5 (trendiest...)	[794, 821, 826, 852, 17869]	[821, 794, 826, 852, 17877]

Next, by interpreting the results, we can establish the differences, pros and cons of these ranking models.



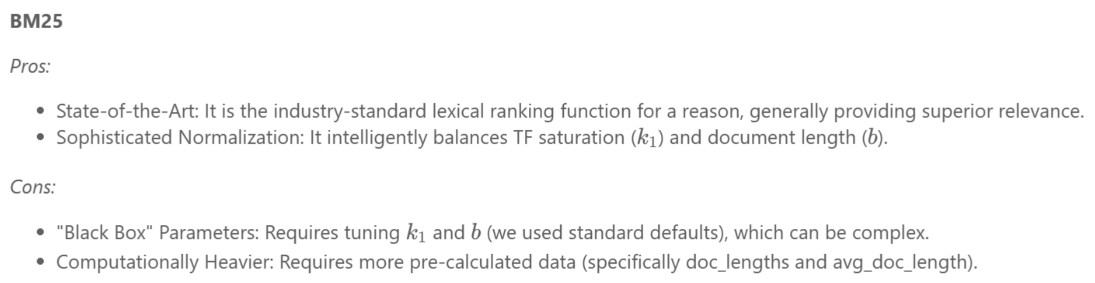
TF-IDF (Dot Product)

Pros:

- Simple & Intuitive: The model is easy to understand and implement.
- Good Baseline: It provides a solid, classic baseline to measure other models against.

Cons:

- Biased Towards Length: Favors longer documents, which is often not what a user wants.
- No TF Saturation: Can be "gamed" by keyword-stuffing, rewarding documents that overuse a query term.



BM25

Pros:

- State-of-the-Art: It is the industry-standard lexical ranking function for a reason, generally providing superior relevance.
- Sophisticated Normalization: It intelligently balances TF saturation (k_1) and document length (b).

Cons:

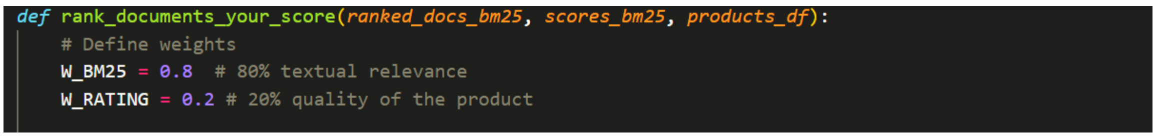
- "Black Box" Parameters: Requires tuning k_1 and b (we used standard defaults), which can be complex.
- Computationally Heavier: Requires more pre-calculated data (specifically doc_lengths and avg_doc_length).

6. Hybrid Ranking ("your score")

Prior to implementing this new ranking, we reflected on the key factors that make a ranking relevant when searching for information or products on the internet, and we concluded that one of the pillars on which we base our ranking is the quality of the results.

Following this idea, we have decided to implement a hybrid algorithm with the results of the BM25 ranking (which is more powerful than TFIDF) and the average_rating variable from our data. This allows us to combine these variables in a linear combination to find a new ranking according to different weights.

First, we defined the weights, i.e., the importance we give to each variable. In this case, the selected weights are as follows:



```python
def rank_documents_your_score(ranked_docs_bm25, scores_bm25, products_df):
    # Define weights
    W_BM25 = 0.8   # 80% textual relevance
    W_RATING = 0.2 # 20% quality of the product
```

Next, we found the rating for that product, normalised it, and calculated and stored the final score.

```
    # Normalize rating
    norm_rating = rating / MAX_RATING

    # Calculate final score
    final_score = (W_BM25 * norm_bm25) + (W_RATING * norm_rating)

    # Store final score
    your_scores[doc_id] = final_score
```

Note that we have also normalised our score, so our maximum score is 1.00.

Once our ranking function has been implemented, we have iterated over our test_queries, defined in part 2 of the project and retrieved to test our ranking functions. Once the ranking has been executed on test_queries, we obtain the following result:

```
Query ID: q1
Query: women full sleeve sweatshirt cotton
Top 5 Ranked Document IDs: [4288, 4290, 14655, 25149, 25015]
Scores: ['0.9880', '0.9880', '0.9781', '0.9277', '0.9221']
--------------------------------------------------
Query ID: q2
Query: men slim jeans blue
Top 5 Ranked Document IDs: [10303, 24544, 24547, 10401, 10348]
Scores: ['0.9641', '0.9520', '0.9481', '0.9481', '0.9361']
--------------------------------------------------
Query ID: q3
Query: neck solid fit
Top 5 Ranked Document IDs: [21243, 21273, 14713, 13512, 24730]
Scores: ['0.9449', '0.9449', '0.9357', '0.9348', '0.9331']
--------------------------------------------------
Query ID: q4
Query: trendiest glossi
Top 5 Ranked Document IDs: []
Scores: []
--------------------------------------------------
Query ID: q5
Query: trendiest women
Top 5 Ranked Document IDs: [821, 794, 17877, 17869, 826]
Scores: ['0.9600', '0.9600', '0.8192', '0.7498', '0.7376']
--------------------------------------------------
```

Below we compare the results obtained with our new hybrid ranking implementation with the previous ones:

Query	T1: TF-IDF	T2: BM25	T3: Your Score
q1 (women...)	[4288, 4290, 24129, 25300, 23179]	[4288, 4290, 14655, 25149, 25151]	[4288, 4290, 14655, 25149, 25015]
q2 (men slim...)	[7595, 7605, 7617, 7623, 7634]	[24544, 24547, 11292, 10283, 26174]	[10303, 24544, 24547, 10401, 10348]
q3 (neck solid...)	[9808, 9810, 25250, 25251, 25281]	[12184, 12143, 12152, 12224, 24712]	[21243, 21273, 14713, 13512, 24730]
q4 (trendiest...)	[]	[]	[]
q5 (trendiest...)	[794, 821, 826, 852, 17869]	[821, 794, 826, 852, 17877]	[821, 794, 17877, 17869, 826]

Next, by interpreting the results, we can establish the differences, pros and cons of our hybrid ranking model.

Hybrid Ranking

Pros:

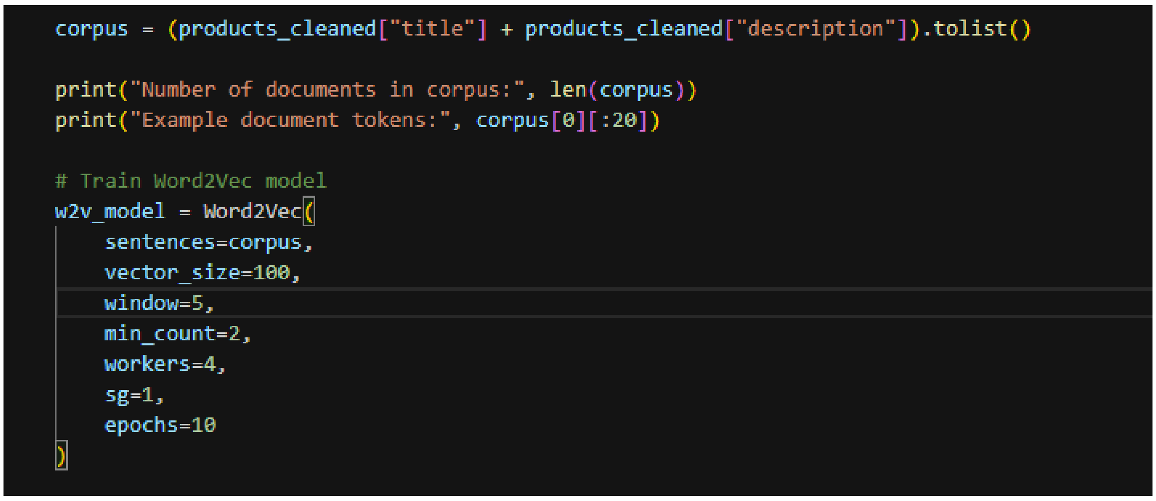
- Smarter Tie-Breaker: When textual relevance is similar, the product with the better rating wins.
- Better User Experience: Aligns with real-world user intent (users want good, relevant products, not just textually relevant ones).
- Leverages Part 1 Work: Directly uses the average_rating field we identified and cleaned in Part 1.

Cons:

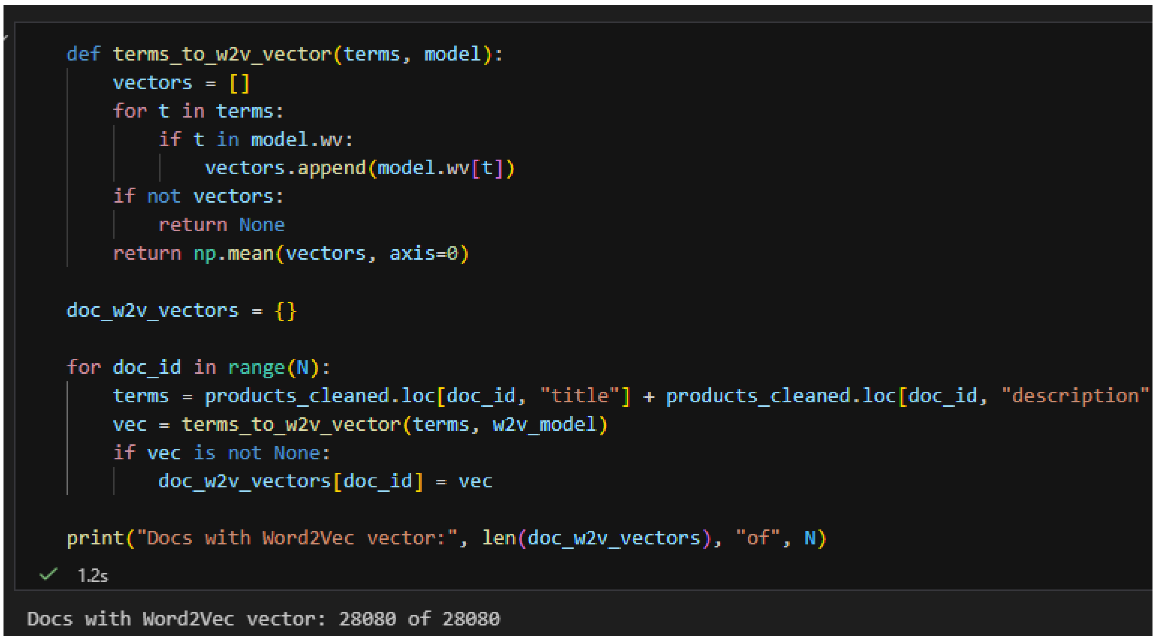
- Popularity Bias: This is the main drawback. New products with 0 ratings are unfairly penalized, making it very difficult for them to ever appear in the top results.
- Arbitrary Weights: The 80/2s0 split is an educated guess. The optimal weights would need to be found experimentally (e.g., via a Grid Search).

7. Word2vec + cosine ranking score

In this part we trained a Word2Vec model on our own corpus, using the preprocessed tokens from the product titles and descriptions. This gives us a dense vector representation for each word in the vocabulary. Then, for every product we built a single document embedding by averaging the Word2Vec vectors of all its tokens (title + description). In this way, each document is represented by one fixed-size vector.



```python
corpus = (products_cleaned["title"] + products_cleaned["description"]).tolist()

print("Number of documents in corpus:", len(corpus))
print("Example document tokens:", corpus[0][:20])

# Train Word2Vec model
w2v_model = Word2Vec(
    sentences=corpus,
    vector_size=100,
    window=5,
    min_count=2,
    workers=4,
    sg=1,
    epochs=10
)
```

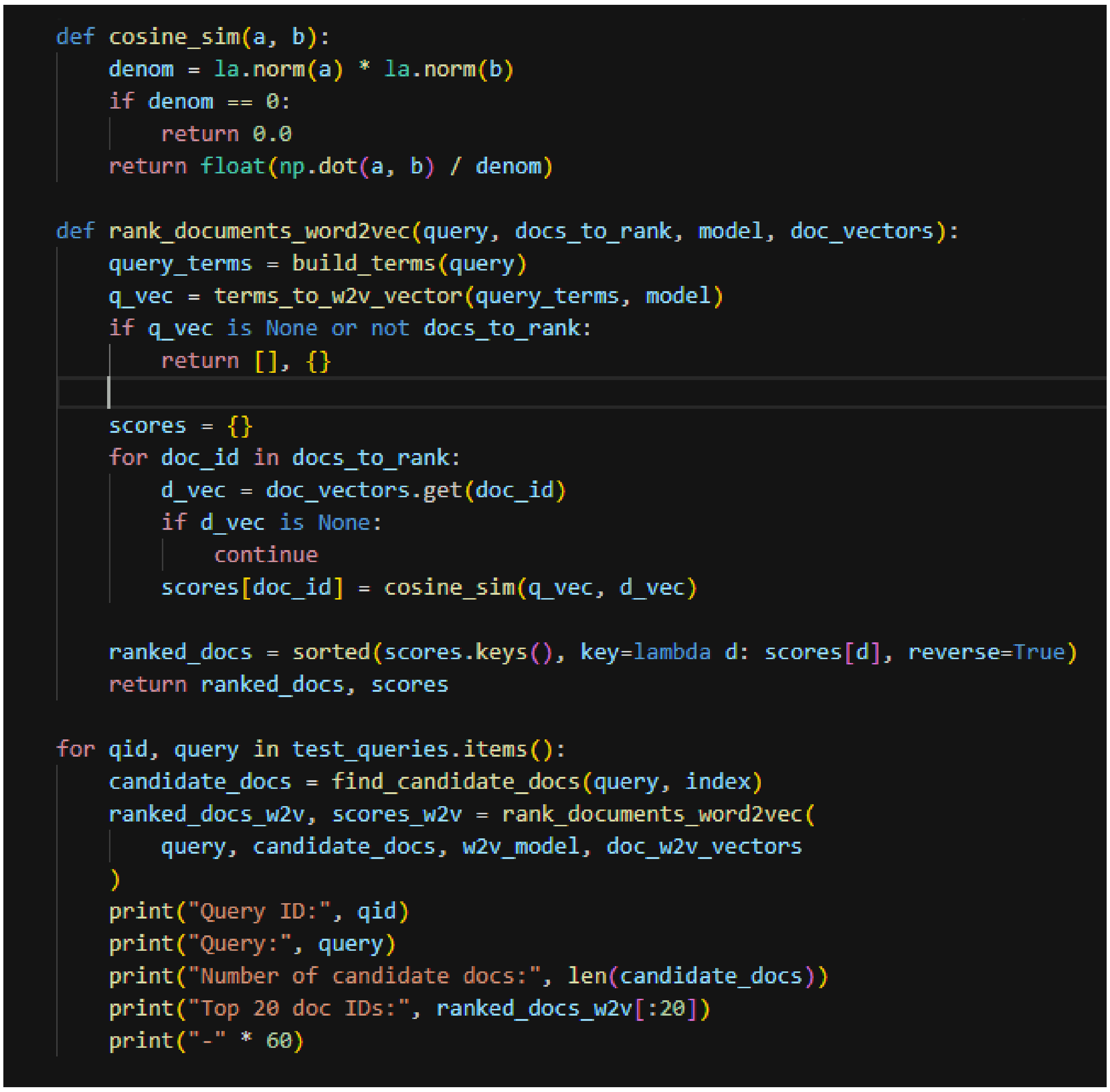


```python
def terms_to_w2v_vector(terms, model):
    vectors = []
    for t in terms:
        if t in model.wv:
            vectors.append(model.wv[t])
    if not vectors:
        return None
    return np.mean(vectors, axis=0)

doc_w2v_vectors = {}

for doc_id in range(N):
    terms = products_cleaned.loc[doc_id, "title"] + products_cleaned.loc[doc_id, "description"
    vec = terms_to_w2v_vector(terms, w2v_model)
    if vec is not None:
        doc_w2v_vectors[doc_id] = vec

print("Docs with Word2Vec vector:", len(doc_w2v_vectors), "of", N)
```

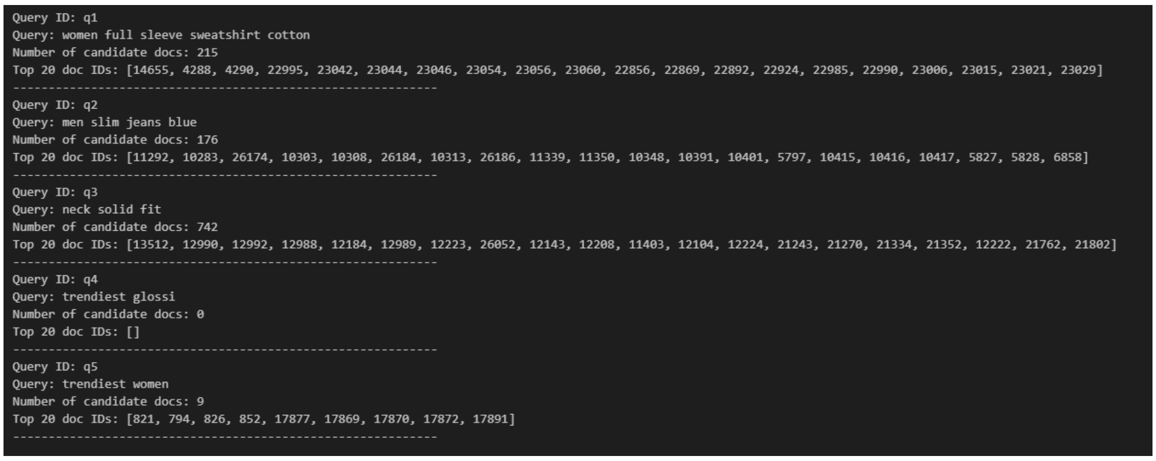
✓ 1.2s

```
Docs with Word2Vec vector: 28080 of 28080
```

At query time we reuse the same conjunctive search as before (AND semantics) to obtain a candidate set of documents. We represent the query as the average of the Word2Vec vectors of its terms and we compute the cosine similarity between the query vector and each candidate document vector. Finally, we sort candidates by cosine similarity and keep the top-20 results. We apply this procedure to the same 5 test queries defined in Part 2.



```python
def cosine_sim(a, b):
    denom = la.norm(a) * la.norm(b)
    if denom == 0:
        return 0.0
    return float(np.dot(a, b) / denom)

def rank_documents_word2vec(query, docs_to_rank, model, doc_vectors):
    query_terms = build_terms(query)
    q_vec = terms_to_w2v_vector(query_terms, model)
    if q_vec is None or not docs_to_rank:
        return [], {}

    scores = {}
    for doc_id in docs_to_rank:
        d_vec = doc_vectors.get(doc_id)
        if d_vec is None:
            continue
        scores[doc_id] = cosine_sim(q_vec, d_vec)

    ranked_docs = sorted(scores.keys(), key=lambda d: scores[d], reverse=True)
    return ranked_docs, scores

for qid, query in test_queries.items():
    candidate_docs = find_candidate_docs(query, index)
    ranked_docs_w2v, scores_w2v = rank_documents_word2vec(
        query, candidate_docs, w2v_model, doc_w2v_vectors
    )
    print("Query ID:", qid)
    print("Query:", query)
    print("Number of candidate docs:", len(candidate_docs))
    print("Top 20 doc IDs:", ranked_docs_w2v[:20])
    print("-" * 60)
```



```
Query ID: q1
Query: women full sleeve sweatshirt cotton
Number of candidate docs: 215
Top 20 doc IDs: [14655, 4288, 4290, 22995, 23042, 23044, 23046, 23054, 23056, 23060, 22856, 22869, 22892, 22924, 22985, 22990, 23006, 23015, 23021, 23029]
------------------------------------------------------------
Query ID: q2
Query: men slim jeans blue
Number of candidate docs: 176
Top 20 doc IDs: [11292, 10283, 26174, 10303, 10308, 26184, 10313, 26186, 11339, 11350, 10348, 10391, 10401, 5797, 10415, 10416, 10417, 5827, 5828, 6858]
------------------------------------------------------------
Query ID: q3
Query: neck solid fit
Number of candidate docs: 742
Top 20 doc IDs: [13512, 12990, 12992, 12988, 12184, 12989, 12223, 26052, 12143, 12208, 11403, 12104, 12224, 21243, 21270, 21334, 21352, 12222, 21762, 21802]
------------------------------------------------------------
Query ID: q4
Query: trendiest glossi
Number of candidate docs: 0
Top 20 doc IDs: []
------------------------------------------------------------
Query ID: q5
Query: trendiest women
Number of candidate docs: 9
Top 20 doc IDs: [821, 794, 826, 852, 17877, 17869, 17870, 17872, 17891]
------------------------------------------------------------
```

8. Can you imagine a better representation than word2vec?

Word2Vec is a good starting point to represent text, but it is still quite simple: it learns one fixed vector for each word, and we represent a document by just averaging all its word vectors. In this process we lose information about document structure and some nuances of meaning.

One possible improvement is Doc2Vec, which directly learns a vector for each document (or paragraph) instead of only for individual words. This allows the model to capture more global information about the document (such as topics) in a single embedding. The drawback is that Doc2Vec is usually harder to train and more sensitive to hyperparameters and data size than the simple "average of Word2Vec vectors", so it is more difficult to tune and debug in practice.

9. Statement of use of generative Artificial Intelligence

9.1. For which aspects of the practice we have used AI

Definition and understanding of the objectives of our project: First of all, we have used generative AI to guarantee a correct and complete understanding of the objectives of the practice at a technical and mathematical level. Likewise, we have been able to establish the statement through practical examples to be able to develop the practice with total guarantees at the level of understanding.

Resolving doubts about the problems that arose during the development of the practice: As we were solving the proposed statement, different problems arose at code level as execution errors, which we easily solved through VSCode debugging with the help of generative AI to detect where in our code was the error in order to solve it.

Complex code writing guided by the provided notebooks and prompt engineering techniques to adapt our base code with new modalities in order to simplify our initial implementation and improve the results based on the different evaluation methods used.

Code review and validation of our model: Once our code was finalized, we performed an exhaustive analysis of it to verify that it was correct.

9.2. To what extent has it facilitated the realization of the practice

In general, access to these new tools in a responsible and coherent way has been a great advantage when carrying out the practice, especially in terms of time, since it has facilitated different aspects such as:

Quick understanding of complex concepts, for which, in the past, it would take us much more time to find the indicated information on the internet, interpret it and adapt it to our needs.

Resolution of errors, for which, as in the previous point, in the past we could take a long time to detect and solve correctly.

Obtaining explanations about python library methods that we do not know in depth in a much faster and more efficient way than consulting the library guide.

Verification of the content of the memory to verify that we did not leave anything relevant for the total understanding of our work developed in this practice.

9.3. How did you verify and adapt the information obtained

As mentioned in the previous section, we consider the use of these new tools to be a great advantage over the past, as long as they are used responsibly and conscientiously. Therefore, we have used a cross-validation method to ensure that the information provided by the generative AI was correct and optimal. For this purpose, the following methodology was followed:

Checking on the validity of the answers through the Internet and resources provided in the referenced notebooks.

Contrasting the theoretical explanations with the Internet and referenced notebooks, especially in the mathematical part since the generative AIs are not perfected in this area.

Use of AI as a support tool and not as a substitute for critical thinking to solve the different problems posed.