UPF – Information Retrieval and Web Analytics

Final Project Part 2

Jhonatan Barcos Gambaro (u198728)

Daniel Alexander Yearwood Agames (u214976)

Part 2: Indexing and Evaluation

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-2-submission:

https://github.com/laxhoni/IRWA_Final_Project/releases/tag/part-2-submission

2. Introduction

The primary objective of this second part of the project is to build the core components of our search engine. This involves transitioning from the data analysis performed in Part 1 to a functional retrieval system.

This implementation focuses on three main tasks:

- **Indexing**: Building the fundamental data structure (an inverted index) that allows for fast lookups of terms, and pre-calculating the statistical weights (TF, DF, IDF) needed for ranking.

- **Retrieval & Ranking**: Implementing the search logic based on the TF-IDF vector space model with cosine similarity.

- **Evaluation:** Assessing the performance of our TF-IDF model by comparing its results against a known set of relevant documents (*ground truth*) and calculating standard information retrieval metrics.

A critical constraint for this assignment is the implementation of conjunctive queries (AND), meaning a document is only considered a match if it contains all terms from the user's query.

3. INDEXING

3.1. BUILD INVERTED INDEX

To build the inverted index, we adapted the build_terms and create_index functions from the practical session. This function efficiently processed page_contents (combining title and description) to generate all the data structure necessary to build the inverted index.

Below, we explain how the initial functions were adapted to produce:

[1] def build_terms(text):

To adapt this function, we have combined our clean_text function developed in part 1 with the build_terms function from the practical session. Now this function returns a list of tokens and increases the filtering of irrelevant terms.

```python
def clean_text(text):
    # NEW: Remove numbers
    text = re.sub(r'\d+', '', text)
    word_tokens = word_tokenize(text.lower())
    # NEW: Stop words updated
    textos_limpios = ' '.join([word for word in word_tokens if word not in stop_words and word.isalnum()])
    textos_limpios = ' '.join([stemmer.stem(word) for word in word_tokenize(textos_limpios)])
    return textos_limpios
```

```python
def build_terms(text):
    text = re.sub(r'\d+', '', text)
    word_tokens = word_tokenize(text.lower())
    textos_limpios = [word for word in word_tokens if word not in stop_words and word.isalnum()]
    textos_limpios = [stemmer.stem(word) for word in textos_limpios]
    return textos_limpios
```

[2] create_index_tfidf(documents_content, title_index_map, num_documents):

- **Input:** Previously, it only received 'lines', a flat text file that had to be parsed. Now it receives documents_content and title_index_map.
- **Iteration:** Previously, we iterated over the lines and extracted the page_id and title manually. Now we iterate in a cleaner way using enumeration, thus obtaining the page_id and content directly.
- **Processing**: Previously, the build_terms function was used without adaptation. Now the redefined function is used with greater precision.

3.2. PROPOSE TEST QUERIES

For the test queries proposal, we have defined a list of queries based on the previous analysis of term frequency. To do this, we analysed their distribution using Document Frequency dictionary and printed the most relevant terms:

```
Most frequent terms by document frequency:
women : 13434
men : 13106
neck : 12225
solid : 9826
print : 9226
cotton : 8151
fit : 7397
casual : 6901
comfort : 6712
shirt : 5719
```

```
Some rare terms by document frequency:
trendiest : 21
glossi : 21
tone : 21
delic : 21
compress : 21
push : 21
repeat : 21
domin : 21
tast : 21
higher : 21
```

After analysing the different terms according to their frequency in the documents, we have defined the following query proposals:

```
test_queries = {
    # Q1: Compulsory (validation_labels.csv)
    "q1": "women full sleeve sweatshirt cotton",

    # Q2: Compulsory (validation_labels.csv)
    "q2": "men slim jeans blue",

    # Q3: High Frequency Query (Based on Top DF)
    "q3": "neck solid fit",

    # Q4: Small Frequency Query (Based on Low DF)
    "q4": "trendiest glossi",

    # Q5: Combined Query (User Simulation)
    "q5": "trendiest women"
```

3.3. RANK YOUR RESULTS

Once we had defined the list of queries we would use for the ranking, we adapted functions rank_documents(terms, docs, index, idf, tf, title_index) and search(query, index) from the practical session so that they could be used correctly. These adaptations are detailed below:

[2] search_tf_idf(query, index):

The initial search function used an 'OR' approach, whereas in this new redefinition of the function, we have implemented a conjunctive 'AND' search such that all terms in the query must match instead of just one.

[3] rank_documents(terms, docs, index, idf, tf, title_index):

This function remains virtually unchanged; we have simply added a condition to return an empty list when no elements are found, since otherwise, when executing it, we would encounter an infinite loop if no results were found.

Below is the implementation of these functions iterating over the list of proposed queries.

```
print("Insert your query (i.e.: women full sleeve sweatshirt cotton):\n")

for query_id, query in test_queries.items():
    print("Processing test query {}: {}".format(query_id, query))
    ranked_docs, scores = search_tf_idf(query, index)
    top = 10

    print("\n======================\nTop {} results out of {} for the searched query:\n".format(top, len(ranked_docs)))
    for d_id in ranked_docs[:top]:
        print("page_id= {} - page_title: {}".format(d_id, title_index[d_id]))
    print("\n\n")
```

After executing this code, we can see this ranking result for the first query:

```
Insert your query (i.e.: women full sleeve sweatshirt cotton):

Processing test query q1: women full sleeve sweatshirt cotton

======================
Top 10 results out of 215 for the searched query:

page_id= 4290 - page_title: Full Sleeve Solid Women Sweatshirt
page_id= 4288 - page_title: Full Sleeve Solid Women Sweatshirt
page_id= 25149 - page_title: Full Sleeve Self Design Women Sweatshirt
page_id= 25300 - page_title: Full Sleeve Solid Women Sweatshirt
page_id= 14655 - page_title: Full Sleeve Solid Women Sweatshirt
page_id= 25151 - page_title: Full Sleeve Color Block Women Sweatshirt
page_id= 25015 - page_title: Full Sleeve Color Block Women Sweatshirt
page_id= 22995 - page_title: Full Sleeve Color Block Women Sweatshirt
page_id= 25142 - page_title: Full Sleeve Self Design, Color Block Women Sweatshirt
page_id= 24129 - page_title: Full Sleeve Graphic Print Women Sweatshirt
```

4. Evaluation

4.1. Implementation of Evaluation Metrics

In this part we implemented all the metrics needed to evaluate how well the search engine returns relevant products at the top of the ranking.

- **Precision@K (P@K):** percentage of relevant results among the top K items.
- **Recall@K (R@K):** number of relevant items retrieved within the top K compared to all relevant ones.
- **F1@K:** combines precision and recall to get a balanced measure.
- **Average Precision (AP@K):** rewards results that place relevant items earlier in the list.
- **Mean Average Precision (MAP):** average AP across all queries.
- **Mean Reciprocal Rank (MRR):** checks the position of the first relevant result.
- **Normalized Discounted Cumulative Gain (NDCG@K):** measures ranking quality taking into account the position of each document.

```python
# Precision@K
def precision_at_k(ranked_relevances, k):
    ranked_relevances = np.asarray(ranked_relevances)[:k]
    return np.mean(ranked_relevances)

# Recall@K
def recall_at_k(ranked_relevances, total_relevant_docs, k):
    ranked_relevances = np.asarray(ranked_relevances)[:k]
    return np.sum(ranked_relevances) / total_relevant_docs if total_relevant_docs != 0 else 0

# F1-Score@K
def f1_at_k(precision, recall):
    if precision + recall == 0:
        return 0
    return 2 * (precision * recall) / (precision + recall)

# Average Precision@K
def average_precision_at_k(ranked_r (parameter) ranked_relevances: Any
    ranked_relevances = np.asarray(ranked_relevances)[:k]
    precisions = [precision_at_k(ranked_relevances, i + 1) for i in range(len(ranked_relevances)) if ranked_relevances[i]
    return np.mean(precisions) if precisions else 0

# Mean Average Precision (MAP)
def mean_average_precision(all_ranked_relevances, k):
    return np.mean([average_precision_at_k(r, k) for r in all_ranked_relevances])

# Mean Reciprocal Rank (MRR)
def mean_reciprocal_rank(all_ranked_relevances):
    rr_list = []
    for relevances in all_ranked_relevances:
        try:
            first_rel = np.where(np.asarray(relevances) == 1)[0][0]
            rr_list.append(1 / (first_rel + 1))
        except IndexError:
            rr_list.append(0)
    return np.mean(rr_list)

# Normalized Discounted Cumulative Gain (NDCG)
def ndcg_at_k(ranked_relevances, k):
    ranked_relevances = np.asarray(ranked_relevances)[:k]
    dcg = np.sum(ranked_relevances / np.log2(np.arange(2, len(ranked_relevances) + 2)))
    ideal_dcg = np.sum(sorted(ranked_relevances, reverse=True) / np.log2(np.arange(2, len(ranked_relevances) + 2)))
    return dcg / ideal_dcg if ideal_dcg != 0 else 0
```

4.2. Evaluation on the Validation Set

After implementing the metrics, we applied them to the two official queries included in the validation dataset (validation_labels.csv).

```
# Load validation labels
validation = pd.read_csv("../../data/validation_labels.csv")

# Check structure
print(validation.head())
```

✓ 0.0s

```
                                    title                pid  query_id  \
0        Full Sleeve Printed Women Sweatshirt  SWSFFVKBCQG5FHPF         1
1        Full Sleeve Striped Women Sweatshirt  SWSFJY5ZFHQ7HXKW         1
2        Full Sleeve Printed Women Sweatshirt  SWSFUY89NHMZHZPX         1
3  Full Sleeve Graphic Print Women Sweatshirt  SWSFXQ5YX6RZKHP4         1
4          Full Sleeve Solid Women Sweatshirt  JCKFTZBC3DMCVYXH         1

   labels
0       1
1       0
2       1
3       1
4       0
```

These are the queries provided in the project statement:

- q1: “women full sleeve sweatshirt cotton”
- q2: “men slim jeans blue”

For each query, the system retrieved the ranked list of products using our TF-IDF model with cosine similarity and AND search.

We then compared the retrieved results with the ground-truth relevance labels in the validation file

```
Evaluating query q1: women full sleeve sweatshirt cotton

Evaluating query q2: men slim jeans blue

Evaluation results for official queries:
  Query  P@10  R@10  F1@10  AP@10  NDCG@10    MAP    MRR
0    q1   0.0   0.0    0.0  0.000    0.000  0.056  0.087
1    q2   0.1   0.1    0.1  0.111    0.301  0.056  0.087
```

As we can see, the values are low for both queries.

This happens because the search is very strict — if one term from the query doesn't appear in the document (for example cotton), that product is ignored.

So, even if the item is clearly relevant, it's not counted, which reduces recall and precision at the same time.

4.3. Manual Evaluation and Analysis

Besides the automatic evaluation, we also did a manual relevance check to see how our search behaves with queries we created ourselves.

We used the five queries defined in Part 1 (q1–q5) and manually checked the top 10 retrieved products for each one.

```
==================================
Query q1: women full sleeve sweatshirt cotton
```

	doc_id	pid	title
0	4290	SWSFWTND3EPMFCQD	Full Sleeve Solid Women Sweatshirt
1	4288	SWSFWTNDJFCF72WU	Full Sleeve Solid Women Sweatshirt
2	25149	SWSFBCZAJEFJZGFV	Full Sleeve Self Design Women Sweatshirt
3	25300	SWSFN2XZYZMGHD9A	Full Sleeve Solid Women Sweatshirt
4	14655	SWSFMJF98EY2FXBH	Full Sleeve Solid Women Sweatshirt
5	25151	SWSFATS4Y9ZKZKRY	Full Sleeve Color Block Women Sweatshirt
6	25015	SWSFBCZBYMGFCKHA	Full Sleeve Color Block Women Sweatshirt
7	22995	SWSFW6KB8GZHEC72	Full Sleeve Color Block Women Sweatshirt
8	25142	SWSFATS4WGGZHSBT	Full Sleeve Self Design, Color Block Women Swe...
9	24129	SWSF5R7F2ZYCZYKH	Full Sleeve Graphic Print Women Sweatshirt

```
==================================
Query q2: men slim jeans blue
```

	doc_id	pid	title
0	26186	JEAFZ5MTE7GYFKBG	Slim Men Blue Jeans
1	26184	JEAFZ5MR57ZWHPDM	Slim Men Blue Jeans
2	26174	JEAFWZXC9EZG9DKC	Slim Men Blue Jeans
3	24435	JEAF6FFZ4VQNHKFX	Slim Men Blue Jeans
4	24430	JEAF6FFZBFFYPZKM	Slim Men Blue Jeans
5	17157	JEAFVPFUA97ZETDB	Slim Men Blue Jeans
6	16194	JEAFS57EYDGY4EJM	Slim Men Blue Jeans
7	16173	JEAFS2KDU3UMSA7U	Slim Men Blue Jeans
8	14232	JEAFTGSGTYKZGAEZ	Slim Men Blue Jeans
9	14167	JEAFSGSYEAFCYHEE	Slim Men Blue Jeans

```
==================================
Query q3: neck solid fit
```

	doc_id	pid	title
0	12152	TSHFV5VY5CVGYXNM	Solid Women Round Neck Green T-Shirt
1	24946	TSHFGQREVW9HHAU3	Solid Women Round Neck Orange T-Shirt
2	24945	TSHFGQREYHXENFDM	Solid Men Round Neck Orange T-Shirt
3	24944	TSHFDVE67DZXVDGJ	Solid Women Round Neck Blue T-Shirt
4	24942	TSHFGQREFGZGXFPY	Solid Men Round Neck Orange T-Shirt
5	24936	TSHFGRPU3FZJFPWZ	Solid Women Round Neck Green T-Shirt
6	24929	TSHFGDHMEVFWNNBZ	Solid Women Round Neck Yellow T-Shirt
7	24927	TSHFGQREHPCMP6VJ	Solid Women Round Neck Orange T-Shirt
8	24925	TSHFGRPUFSQH3CSQ	Solid Men Round Neck Green T-Shirt
9	24923	TSHFDVE75ACHFCNP	Solid Men Round Neck Grey T-Shirt

```
No results found, try another query.

==================================
Query q4: trendiest glossi
```

```
==================================
Query q5: trendiest women
```

	doc_id	pid	title
0	821	ETHFUQZ3DP7BDA6R	Women Kurta and Pyjama Set Pure Cotton
1	794	ETHFUQZ3ZH9X68XA	Women Kurta and Pyjama Set Pure Cotton
2	826	ETHFYBRYTJMNQMCY	Women Kurta and Pyjama Set Jacquard
3	852	ETHFYE2FY64ZFKTA	Women Kurta and Pyjama Set Tussar Silk
4	17877	JEAFYAFZTH8CNT5H	Skinny Women Light Blue Jeans
5	17891	JEAFS7R4QBUQGRF8	Skinny Women Light Blue Jeans
6	17872	JEAFS7R4DZUFNPDH	Skinny Women Dark Blue Jeans
7	17870	JEAFS7R4ZRY8FSRC	Skinny Women Dark Blue Jeans
8	17869	SRTFYUY3ZH9HY6YJ	Graphic Print Women Denim Black Denim Shorts

Each result was labelled as relevant (1) or not relevant (0) based on the product title and description.

```python
manual_labels = {}

# q1 (10 docs)
manual_labels["q1"] = {
    "doc_ids": manual_judgments["q1"],
    "relevances": [1,1,1,1,1,1,1,1,1,1]
}

# q2 (10 docs)
manual_labels["q2"] = {
    "doc_ids": manual_judgments["q2"],
    "relevances": [1,1,1,1,1,1,1,1,1,1]
}

# q3 (10 docs)
manual_labels["q3"] = {
    "doc_ids": manual_judgments["q3"],
    "relevances": [1,1,1,1,1,1,1,1,1,1]
}

# q4 (no results)
manual_labels["q4"] = {
    "doc_ids": manual_judgments.get("q4", []),
    "relevances": []
}

# q5 (9 docs)
manual_labels["q5"] = {
    "doc_ids": manual_judgments["q5"],
    "relevances": [1,1,1,1,1,1,1,1,1]
}
```

Once the manual labels were defined, we applied the same metrics again to measure precision, recall and overall ranking quality.

The table below shows the results:

```
Manual evaluation of our queries:
  Query  P@K  R@K  F1@K  AP@K  NDCG@K
0    q1  1.0  1.0   1.0   1.0     1.0
1    q2  1.0  1.0   1.0   1.0     1.0
2    q3  1.0  1.0   1.0   1.0     1.0
3    q4  NaN  NaN   NaN   NaN     NaN
4    q5  1.0  1.0   1.0   1.0     1.0
```

All queries except q4 obtained perfect scores, meaning that the retrieved products were all relevant for the given queries.

Query q4 (“trendiest glossi”) returned no results, which shows that our system cannot handle very uncommon or unseen terms.

4.4. Limitations and Possible Improvements

Even if our manual results were all 1.0, the system still has some clear limits.

Right now, the search is very strict and simple, so some improvements could make it work much better:

1. Strict AND search: if one word from the query isn’t in the product text, that item doesn’t appear at all.

   -We could make it softer, for example using OR search or BM25, so it finds more possible matches.

2. All fields have the same weight: title and description count the same, but the title is usually more important.

   - We could give more weight to the title or category.

3. No metadata used: right now we ignore fields like brand, gender or sub_category.

   - Adding them could help to rank or filter better.

4. No synonyms: words like hoodie and sweatshirt are treated as different.

   -Adding some synonym list or lemmatization could fix that.

Overall, the TF-IDF model works well when the query words exist in the dataset, but when the words are rare or different, the results drop fast.

These small changes would already make the search more flexible and realistic.

5. Statement of use of generative Artificial Intelligence

5.1. 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.

5.2. 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.

5.3. 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.