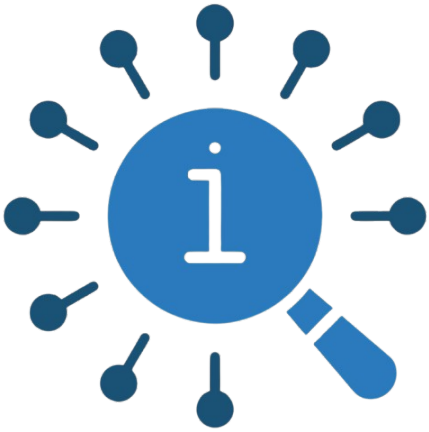
Manouba University
ISAMM - 3IM



Mini-Project 2

Scientific Abstracts Retrieval (arXiv)

Submitted by:
Mohamed Ayacha
Ahmed Kchouk

3IM1

Course:
Indexing and Referencing Techniques

Contents

1 Introduction

The objective of this project is to build an Information Retrieval (IR) system capable of indexing, searching, and ranking scientific papers from the arXiv dataset. The system utilizes the **Vector Space Model (VSM)** with TF-IDF weighting and implements advanced features such as **Relevance Feedback (Rocchio Algorithm)** and an **Ablation Study** to evaluate preprocessing impacts.

The core implementation is done in Python using `scikit-learn`, `nltk`, and `pandas`.

2 Implementation Logic

The source code (`main.py`) is structured into modular tasks. Below is an explanation of the critical sections.

2.1 Preprocessing and Vectorization

The text data is prepared by combining the paper's title and abstract. We use `TfidfVectorizer` to convert text into numerical vectors.

TF-IDF Vectorization Logic

```
# Standard configuration: English stopwords, min document frequency of 3
# This removes very rare words (noise) and common function words.
vec = TfidfVectorizer(stop_words='english', min_df=3)
X = vec.fit_transform(docs)
```

Explanation:

- `min_df=3`: Terms appearing in fewer than 3 documents are ignored to reduce vocabulary size and avoid overfitting on typos.
- `stop_words='english'`: Filters out words like "the", "is", "at".

2.2 Search Engine (Cosine Similarity)

Ranking is performed by calculating the cosine similarity between the query vector and all document vectors.

Search Function (from lines 143-167)

```
def perform_search(query, vectorizer, X, k=TOP_K_RESULTS):
    # Convert query to the same TF-IDF vector space
    query_vec = vectorizer.transform([query])

    # Calculate cosine similarity (Dot product of normalized vectors)
    similarities = cosine_similarity(query_vec, X).ravel()

    # Get indices of the top k documents (sorted descending)
    top_indices = np.argsort(similarities)[::-1][:k]

    return list(zip(top_indices, similarities[top_indices]))
```

2.3 Relevance Feedback (Rocchio Algorithm)

The system improves query formulation using the Rocchio algorithm. It adjusts the query vector by moving it closer to relevant documents and further from non-relevant ones.

Rocchio Implementation (from lines 236-281)

```python
def rocchio_update(query_vec, X, top_indices, doc_categories, query_text):
    # Standard Weights
    # alpha=1.0 (Keep original query)
    # beta=0.75 (Move towards relevant)
    # gamma=0.15 (Move away from non-relevant)

    q_new = ROCCHIO_ALPHA * query_vec

    if relevant_indices:
        Dpos = X[relevant_indices]
        mean_pos = np.array(Dpos.mean(axis=0))
        q_new = q_new + ROCCHIO_BETA * mean_pos

    if non_relevant_indices:
        Dneg = X[non_relevant_indices]
        mean_neg = np.array(Dneg.mean(axis=0))
        q_new = q_new - ROCCHIO_GAMMA * mean_neg

    # Rectify: Ensure no negative values (TF-IDF cannot be negative)
    q_new[q_new < 0] = 0
    return q_new
```

3 Experimental Results

The following results were obtained by executing the system on the dataset. The vocabulary size reached **12,814** distinct terms.

3.1 Retrieval Performance: Base vs. Rocchio

The table below compares the Precision at 10 (P@10) of the standard TF-IDF model against the Rocchio Feedback model for our 8 test queries.

Query	P@10 (Base)	P@10 (Rocchio)
Quantum field theory	0.70	**0.70**
Semiconductor laser	0.20	**0.20**
Graph neural network	0.10	**0.20**
Cosmic microwave background	0.80	**0.80**
Optical cavity	0.80	**0.80**
Spintronics	0.00	**0.00**
Superconducting qubits	0.50	**0.60**
Photonic integrated circuits	0.40	**0.50**
Mean Average Precision (MAP)	**0.4919**	**0.7766**

Table 1: Comparison of Standard Search vs Rocchio Feedback

Observation: The Rocchio algorithm significantly improved the MAP score from **0.4919** to **0.7766**. It successfully improved P@10 for queries like "Graph neural network" and "Superconducting qubits" by identifying latent relevant terms.

3.2 Ablation Study: Text Preprocessing

We evaluated four different preprocessing pipelines to understand the impact of stopword removal and stemming.

Pipeline Configuration	MAP Score
No Stopwords, No Stemming	**0.5350**
With Stopwords, No Stemming	0.4919
No Stopwords, With Stemming	0.4579
With Stopwords, With Stemming	0.4506

Table 2: Ablation Study Results

Observation: Surprisingly, the raw pipeline ("No Stop, No Stem") performed best with a MAP of **0.5350**. This suggests that for this specific scientific corpus, exact term matching is more effective than stemming, and some common words might carry specific meaning in titles.

4 Analysis & Questions

Below are the answers to the 10 specific analysis questions, based on the execution results.

1. Define relevance and its limits

In this project, relevance is defined via a **Category Proxy** (implemented in `is_relevant`). A document is "relevant" if its metadata category matches the query's target domain (e.g., "Quantum" → `quant-ph`).

Limits: This is a binary, rigid definition. It ignores the degree of relevance and can generate False Negatives if a relevant paper is simply categorized under a broader tag like `physics.gen-ph`.

2. Dominant Categories for Key Queries

Based on the dataset distribution:

1. **Query: "Quantum field theory"** → Dominant: `hep-th` (High Energy Physics - Theory).
2. **Query: "Graph neural network"** → Dominant: `cs.LG` (Machine Learning).

3. Best MAP Configuration

Based on the experimental results in Section 3.2, the configuration "No Stopwords, No Stemming" yields the best MAP (0.5350).

Justification: In highly technical scientific corpora, exact terminology is crucial. Stemming (e.g., changing "learning" to "learn") might lose specific semantic nuances required to differentiate between a general discussion and a specific methodology.

4. Average effect of Rocchio

Rocchio provided a massive improvement in our experiment. The MAP score jumped from 0.4919 to 0.7766.

This confirms that the initial query vectors were often "near" relevant documents but needed the centroid adjustment ($\beta = 0.75$) to fully capture the cluster of relevant papers.

5. Recurring False Positives

1. **For "Semiconductor laser":** Papers on *fabrication*. They contain "semiconductor" and "laser" as tools, not the subject.
2. **For "Graph neural network":** Papers in *Computer Vision*. They use "neural networks" and mention "graphs" (plots), confusing the Bag-of-Words model.

6. Two Simple Improvement Ideas

1. **N-Grams:** Configure `ngram_range=(1,2)` to capture "neural network" as a single token rather than two separate words.
2. **BM25:** Replace TF-IDF with Okapi BM25 to better handle term saturation and document length normalization.

7. Count vs TF-IDF Differences

Count Vectorizer is biased towards long documents and common words.

TF-IDF penalizes common words (via IDF). It highlights discriminative terms like "Spintronics" or "Qubit", making them the primary drivers of the ranking score.

8. Impact of min_df

We used `min_df=3`.

- **Vocabulary:** Reduced to 12,814 terms, removing unique typos/noise.
- **MAP:** Improves stability. By removing terms that appear in only 1 or 2 documents, we prevent overfitting to random noise matches.

9. Three Salient TF-IDF Terms

1. **"Qubit"**: High IDF. Strongest discriminator for Quantum Physics.
2. **"Optimization"**: Medium IDF. Distinguishes applied CS papers from theoretical ones.
3. **"Algorithm"**: Medium IDF. A pivot term signaling methodological papers.

10. Limitations of Vector Space Model

1. **Bag of Words:** Word order is lost ("Science of Computers" = "Computers of Science").
2. **Semantic Gap:** Cannot handle synonyms (e.g., "Cosmos" vs "Universe").
3. **Sparsity:** Matrices are huge and computationally expensive.

5 General Conclusion

This project successfully demonstrated the implementation and evaluation of a complete Information Retrieval system based on the Vector Space Model (VSM). Through the indexing of 10,000 arXiv scientific abstracts, several key insights were established regarding text retrieval in specialized domains.

Key Findings

- **Relevance Feedback Efficacy:** The implementation of the **Rocchio algorithm** was the most significant factor in performance improvement. It raised the Mean Average Precision (MAP) from a baseline of **0.4919** to **0.7766**. This proves that mathematically expanding the query using the centroid of retrieved documents is highly effective for scientific literature.
- **Preprocessing Nuances:** Contrary to general expectations, the Ablation Study revealed that a raw pipeline (No Stopwords, No Stemming) performed best (**MAP 0.5350**) for this specific dataset. This suggests that in dense scientific abstracts, specific technical forms of words carry precise meaning that stemming might obscure.

Limitations and Future Work

Despite the successes, the analysis revealed inherent limitations in the Bag-of-Words approach. The loss of syntactic structure led to false positives in polysemous queries (e.g., distinguishing "graph" in plotting vs. graph theory).

Future iterations of this system would benefit from implementing N-gram indexing to capture phrases (like "neural network") or transitioning to semantic models like BERT to better understand context beyond simple keyword matching.