Fayaz Shaik (11795213)
Fayazshaik@my.unt.edu
fs0388

CSCE 5200 Section 404 - Information Retrieval and Web Search

1. Introduction

In this phase of the project, I implemented a Query Processor for the Information Retrieval (IR) system built in Phase 2. The system uses the Vector Space Model (VSM) with TF-IDF weighting and cosine similarity to rank documents for given queries. The goal was to process queries from topics.txt, retrieve relevant documents, and evaluate performance using precision and recall metrics under three query settings:

1. Title-only (short query)
2. Title + Description (expanded query)
3. Title + Narrative (detailed query)

The system was tested on the ft911 dataset, and results were validated against relevance judgments in main.qrels.

2. System Design

2.1 Term Weighting and Normalization

The system calculates term weights using the TF-IDF scheme for both documents and queries. The following steps were implemented:

1. Term Frequency (TF):
   - For documents, term frequencies were precomputed during indexing (Project Phase 2) and stored in the forward and inverted indexes.
   - For queries, term frequencies were calculated at runtime by tokenizing the query text.
2. Inverse Document Frequency (IDF):
   - IDF values were computed dynamically during query processing using the formula:

$$\mathrm{IDF}(\mathrm{t}) = \log\left(\frac{\text{Total Documents}}{\text{Documents Containing Term t}}\right)$$

3. Normalization:
   - Both query and document vectors were normalized to unit length to ensure consistent comparison using Cosine Similarity.
4. Cosine Similarity:
   - The similarity between a query vector and a document vector was calculated as:

$$\text{Cosine Similarity} = \frac{\text{Query Vector} \cdot \text{Document Vector}}{|\text{Query Vector}| \times |\text{Document Vector}|}$$

2.2 Data Structures and Classes

The system leverages several key data structures and classes:

Fayaz Shaik (11795213)
Fayazshaik@my.unt.edu
fs0388

1. Used Data Structures:
   - HashMaps: Store word/document IDs (wordDictionary, fileDictionary).
   - Nested HashMaps: Represent forward (docID → {termID: count}) and inverted (termID → {docID: count}) indexes.
   - Priority Queues: Rank documents by cosine scores during query processing.
2. Indexer Class:
   - Maintains the forward index, inverted index, word-to-ID mappings, and document-to-ID mappings.
   - Provides methods for querying terms, calculating IDF, and generating document vectors.
3. QueryProcessor Class:
   - Handles query parsing, tokenization, stopword removal, stemming, and relevance scoring.
   - Supports three query settings: title-only, title + description, and title + narrative.
4. TextParser Class:
   - Responsible for preprocessing the dataset, including tokenization, stopword removal, stemming, and dictionary creation.
5. Porter Stemmer:
   - Used to reduce words to their base stems for consistent indexing and querying.
6. Relevance Judgments:
   - The main.qrels file was used to evaluate the system's performance by comparing retrieved documents against manually judged relevance scores.

2.3 Key Components

1. Query Processing Pipeline:
   - Tokenization: Split queries into terms using regex ([^a-zA-Z]+).
   - Stopword Removal: Filter out common words using stopwordlist.txt.
   - Stemming: Apply Porter Stemmer to normalize terms.
2. Vector Space Model:
   - Document Vectors: Precomputed TF-IDF weights for all documents (using Indexer from Phase 2).
   - Query Vectors: Dynamically computed TF-IDF weights for queries.
   - Cosine Similarity: Calculated between query and document vectors for ranking.
3. Performance Metrics:
   - Precision: (Relevant Retrieved) / (Total Retrieved).
   - Recall: (Relevant Retrieved) / (Total Relevant).

Fayaz Shaik (11795213)
Fayazshaik@my.unt.edu
fs0388

3. Implementation Details

3.1 Document Parsing & Indexing

- TextParser reads TREC files, extracts <DOCNO> and <TEXT>, and tokenizes/stems terms.
- Indexer constructs indexes and validates consistency (e.g., ensuring forward/inverted index alignment).

3.2 Query Processing Workflow

The query processing pipeline consists of the following steps:

1. Query Parsing:
    - Queries were parsed from the topics.txt file, extracting the title, description, and narrative fields.
2. Preprocessing:
    - Each query was tokenized, stopwords were removed, and tokens were stemmed using the Porter Stemmer.
3. Vector Representation:
    - A query vector was constructed using TF-IDF weights for each term.
4. Cosine Similarity Calculation:
    - For each document in the collection, the cosine similarity between the query vector and the document vector was computed.
5. Ranking and Output:
    - Documents were ranked based on their cosine similarity scores and written to the vsm_output.txt file in the specified format.

3.3 Performance Evaluation

Performance was evaluated using Precision and Recall metrics for each query setting. The results were summarized in the performance_report.txt file.

4. Results and Analysis

4.1 Output Format

- The output file vsm_output.txt was generated in the following format:
  <TOPIC> <DOCUMENT> <UNIQUE#> <COSINE_VALUE>
- Each line represents a document retrieved for a specific query, along with its rank and cosine similarity score. For example:

```
352    FT911-3246     1    0.206355

352    FT911-4701     2    0.176523

352    FT911-437     3    0.153762
```

4.2 Performance Metrics

The performance_report.txt file provides a detailed comparison of Precision and Recall for each query under different settings. Key observations include:

- Topic 352 (British Chunnel Impact):
    - Title-only achieved a Precision of 0.0018 and Recall of 0.0385.
    - Adding the narrative did not improve performance significantly.
- Topic 353 (Antarctica Exploration):
    - Combining the title and description improved Recall to 0.4545 but reduced Precision to 0.0029.
- Topic 354 (Journalist Risks):
    - Title + description achieved the highest Recall (0.3200) but had the lowest Precision (0.0032).
- Topic 359 (Mutual Fund Predictors):
    - Performance remained consistent across all settings, with low Recall (0.1667).

4.3 Results Summary

Topic	Setting	Precision	Recall
352	Title	0.0018	0.0385
	Title+Description	0.0005	0.0385
	Title+Narrative	0.0018	0.0385
353	Title	0.0560	0.3182
	Title+Description	0.0029	0.4545
	Title+Narrative	0.0560	0.3182
354	Title	0.0135	0.2400
	Title+Description	0.0032	0.3200
	Title+Narrative	0.0135	0.2400
359	Title	0.0011	0.1667
	Title+Description	0.0004	0.1667
	Title+Narrative	0.0011	0.1667

4.4 Observations

- Precision vs. Recall Trade-off:
    - Expanding queries with additional fields (description or narrative) generally increased Recall but decreased Precision.
- Impact of Query Expansion:
    - While query expansion helped retrieve more relevant documents, it also introduced noise, reducing overall Precision.
- System Limitations:
    - The system relies heavily on the quality of the relevance judgments in main.qrels. Missing judgments for some documents may affect evaluation accuracy.

5. Conclusion

This project successfully implemented a Query Processor capable of retrieving and ranking documents using the Vector Space Model and Cosine Similarity. The system demonstrated the trade-offs between Precision and Recall when using different query settings. Future improvements could include:

1. Advanced Query Expansion Techniques:

- Incorporating synonym expansion or semantic analysis to improve query representation.

2. Improved Relevance Scoring:

- Exploring weighting schemes like BM25 for better term importance modeling.

3. Handling Large-Scale Data:

- Optimizing the system for scalability and efficiency with larger datasets.

Overall, the project provided valuable insights into the complexities of information retrieval and the challenges of balancing precision and recall in real-world applications.