

INFORMATION RETRIVAL

PROJECT REPORT

NAME : SADVIK KONDADI

STUDENT ID : 11785837

1. INTRODUCTION:

In this project, I developed the query processing and retrieval part of my search engine using the Vector Space Model with TF-IDF weighting and Cosine Similarity. The work is a direct extension of the index structures developed in Project 2, namely dictionary, inverted index, forward index, and stopword list. The objective was to:

- Handle multi-field queries sourced from the TREC topics.txt file
- Dynamically compute TF-IDF weights for both queries and documents
- Rank documents by cosine similarity
- Produce output in the TREC format: - Evaluate the system using precision and recall according to main.qrels.

2. SYSTEM ARCHITECHTURE:

2.1 Files and Data Structures Used

My system relies on the following files from Project 2:

File	Role
dictionary.txt	Maps terms → termIDs
inverted_index.txt	Stores termID → list(docID, tf)
forward_index.txt	Stores docID → terms and tf values (for normalization)
stopwords.txt	Used to filter out stopwords
topics.txt	Contains queries (title, description, narrative)
main.qrels	Relevance judgments for evaluation

3. Query Processing Pipeline:

3.1 Query Parsing:

Each query in “topics.txt” contains three sections:

- **Title** – short main query
- **Description** – extended definition
- **Narrative** – relevance explanation

I implemented three models:

Mode	Query Text Used
title	Only the title
Titledesc	Title + Description
titlenarr	Title + Narration

I apply the same preprocessing used in phase-2:

- Convert to lower keys
- Tokenize
- Remove stopwords
- Apply Porter Stemming

3.2 TF-IDF Weighting:

I compute TF–IDF weights at runtime.

Document Weight Formula:

$$wt,d = (1 + \log tf_{t,d}) \times \log N / df_t$$

Query Weight Formula:

$$wt,q = tf_{t,q} \times \log N / df_t$$

Where:

- tf = term frequency
- df = document frequency from inverted index
- N = total number of documents

I compute document vector length using data from the forward index.

3.3 Cosine Similarity:

- After computing the query vector and retrieving matching posting lists, I calculate:
- $\text{score}(\mathbf{q}, \mathbf{d}) = \frac{\sum \mathbf{wt}_q \cdot \mathbf{wt}_d}{\|\mathbf{q}\| \|\mathbf{d}\|}$ $\text{score}(\mathbf{q}, \mathbf{d}) = \frac{\sum \mathbf{wt}_q \cdot \mathbf{wt}_d}{\|\mathbf{q}\| \|\mathbf{d}\|}$
- Then I will accumulate contributions of all terms, normalize by document length
- Then I will sort results to generate the top-ranked documents.

4. Output Format:

My system outputs results using the required format:

<query ID> <DOCNO> <rank> <score>

Example:

380 3189 1 0.664901

5. Evaluation Method:

To evaluate my system, I used the provided file **main.qrels**.

For each query, I compare:

- The documents retrieved by my system
- The ground-truth relevant documents

I compute:

Precision

$P = \frac{\text{relevant retrieved}}{\text{retrieved}}$

Recall

$R = \frac{\text{relevant retrieved}}{\text{total relevant}}$

I evaluate precision and recall separately for:

- Title
- Title + Description
- Title + Narrative

6. Experimental Results and Analysis:

After running my system with all three query modes using:

- vsm_output_title.txt
- vsm_output_titledesc.txt
- vsm_output_titenarr.txt

I analyzed precision and recall using the relevance judgments.

Observations:

Title Only

- Queries are short and lack context
- Precision is decent for the top few results
- Recall is noticeably lower

Title + Description

- This mode performed the best
- Adding the description provides meaningful context
- Both precision and recall increased compared to title-only mode

Title + Narrative

- The narrative contains many irrelevant words (e.g., “documents not relevant I include...”)
- Recall increases due to more terms
- Precision drops slightly because of noisy terms

7. Design Choices:

7.1 Indexing

- I reused the index created in Project 2 without modifications:
- Inverted index for retrieving documents quickly
- Forward index for computing document vector lengths
- Dictionary for resolving termIDs

7.2 Normalization

I used:

- Logarithmic TF for documents
- Raw TF for queries
- L2 normalization for both document and query vectors

8. Conclusion:

In this project, I successfully implemented:

- TF-IDF based vector space retrieval
- Cosine similarity scoring
- Multi-field query processing
- Support for FT-style document identifiers
- Performance evaluation using precision and recall
- Based on my experiments, **Title + Description** queries produced the best overall performance, achieving a good balance between relevance and coverage.
- Title-only queries missed important context, while narrative-heavy queries introduced noise.
- Overall, my system performs effectively as a functional vector-space search engine built entirely from scratch using my Project 2 index.

9. Files Used in my system:

1. dictionary.txt
2. inverted_index.txt
3. forward_index.txt
4. stopwordslist.txt
5. topics.txt
6. main.qrels
7. vsm_output_title.txt
8. vsm_output_titledesc.txt
9. vsm_output_titlenarr.txt