Information Retrieval System : Saarthi

Nimish Bansal
Vritant Chopra
BITS PILANI

Abstract— **In the era of rapidly expanding digital content, efficient and accurate Information Retrieval (IR) systems have become essential for extracting relevant information from large document corpora. This paper presents the design and implementation of a modular IR system based on the classical Vector Space Model (VSM), incorporating Term Frequency-Inverse Document Frequency (TF-IDF) weighting and cosine similarity for ranking. The system preprocesses a set of plain text documents through tokenization, customized stopword removal, punctuation filtering, and normalization. It constructs TF-IDF vectors for each document, enabling precise comparison with user queries, which are similarly vectorized. Cosine similarity is used to compute relevance scores, and the top-k documents are retrieved accordingly.**

The system also integrates seamlessly with a web-based frontend, allowing users to input queries via a graphical interface. A PostgreSQL backend is used to store metadata and fetch document links for display. The project is implemented in Python using NLTK, NumPy, and Pandas, and is designed for extensibility, allowing future support for semantic search, phrase queries, and additional document formats such as PDF or HTML. This paper details the system architecture, methodology, and results, highlighting its usability and potential as a foundational IR framework for educational or research purposes.

I. INTRODUCTION

With the proliferation of digital content across the internet and private repositories, the ability to efficiently retrieve relevant textual information has become more critical than ever. From academic search engines and corporate document management systems to question-answering bots and personal knowledge assistants, Information Retrieval (IR) is foundational to a wide range of real-world applications. The core objective of any IR system is to identify documents from a corpus that best match a user's information need, typically expressed in the form of a textual query.

Despite the emergence of advanced neural models in recent years, traditional statistical approaches such as the Vector Space Model (VSM) using Term Frequency-Inverse Document Frequency (TF-IDF) and cosine similarity remain widely used due to their simplicity, interpretability, and computational efficiency. These models provide a strong baseline for ranking documents based on the importance of query terms and their relative distribution across documents.

This project presents a modular, Python-based IR system that utilizes TF-IDF to convert both documents and queries into numerical vectors, and then ranks the documents using cosine similarity with respect to the query vector. The system includes a preprocessing pipeline that handles tokenization, custom stopword removal, punctuation filtering, and normalization to ensure high-quality feature extraction from raw text.

Beyond core retrieval functionality, the system is integrated with a frontend web interface that enables real-time query submission and response viewing. A PostgreSQL database backend maintains document metadata and URLs for retrieval and linking. This integration ensures an interactive and user-friendly experience, making the IR system suitable for deployment in educational platforms, lightweight enterprise tools, or proof-of-concept academic prototypes.

The remainder of this paper outlines the architecture, methodology, and implementation details of the IR system. We also evaluate its functionality and propose potential enhancements such as phrase query support and semantic similarity models for improved relevance detection.

II. EXPERIMENTS

In this section, we describe the datasets, evaluation metrics, and procedures used to analyze the effectiveness of our Information Retrieval (IR) system.

A. Defining Relevance Ranking

There are various metrics and interpretations for assessing the relevance of documents in IR systems. In this work, we adopt a standard **top-k cosine similarity** based ranking, where $k = 10$ by default. A document is considered relevant to a query if it appears in the top-k list ranked by cosine similarity between the query vector and document TF-IDF vectors.

B. System Configuration

We experiment with a Python-based IR system that uses a traditional vector space model with TF-IDF weighting. Cosine similarity is employed to score documents relative to the user query. All preprocessing is performed using the NLTK library, and matrix operations use pandas and numpy. Document metadata and links are stored and fetched using a PostgreSQL database. The entire pipeline is wrapped with a web interface for real-time querying.

C. Dataset

Our primary dataset consists of plain-text '.txt' documents collected from publicly available web and academic resources. For evaluation, we randomly select 3% of the full corpus to form a representative sample of the document distribution. Each document is truncated to the first 64 words

for uniformity in vector length and consistency across query processing.

D. Unmatched Results and Noise Filtering

To estimate the uninformative portion of the dataset, we define an *unmatched* set by subtracting the set of top-k retrieved documents from the original 3% sampled corpus. We perform frequency analysis and TF-IDF score comparison to confirm that high-ranking documents generally have better signal-to-noise ratios in terms of token composition and semantic closeness to the query.

This methodology enables us to evaluate and interpret the discriminative capacity of the IR system, and helps inform future improvements such as incorporating semantic embeddings or reranking strategies.

III. Potential Factors in Retrieval Performance

We examine several potential factors that influence whether a document is ranked highly by the IR system in response to a query. These factors stem from statistical characteristics of the document corpus, intrinsic properties of the query-document pair, and similarity scores derived from the TF-IDF vectors. Features may be computed over the query tokens, document tokens, or both in combination.

Some of the key features considered include:

- **Token Overlap:** The number of tokens shared between the query and the document.
- **Document Length:** Total number of tokens in the document after preprocessing.
- **TF-IDF Score Distribution:** The concentration of high TF-IDF values in the document vector.
- **Cosine Similarity Score:** Final score used to rank the document.
- **Query Term Density:** Ratio of query terms present in the document to its total number of tokens.

Many of these properties exhibit distinct distributions between retrieved and non-retrieved documents. For instance, Figure **??** shows that cosine similarity scores for top-ranked documents tend to be more tightly clustered and higher in magnitude compared to documents not included in the top-k set.

Other attributes—such as document length and query term density—have visibly different medians between retrieved and unretrieved sets. These distinctions suggest that the properties in question meaningfully influence retrieval performance. In particular, cosine similarity and token overlap appear to be highly predictive of whether a document will be included in the top-ranked results.

We leverage these observations in later sections to interpret retrieval behavior and discuss future enhancements, such as incorporating more semantic-aware embeddings or term proximity weighting.

IV. Factors Influencing Retrieval Performance

In this section, we explore multiple categories of factors that may influence whether a document is successfully retrieved by the IR system. These factors span corpus-level statistics, document-intrinsic features, and final model output behavior, including cosine similarity.

A. Corpus-Level Statistics

Several factors relate a given document to the overall properties of the corpus. These statistics help explain how common, duplicated, or generic content may affect retrieval ranking.

1) Token Frequency: We compute summary statistics for each document based on corpus-wide token frequency: mean, median, maximum, minimum, and interquartile range (25th and 75th percentiles). Documents with higher frequencies of common tokens tend to be penalized in TF-IDF weighting, which may reduce their retrieval rank.

2) Duplicates: To measure content duplication across the corpus, we compute an approximate match frequency for each document using a sliding window of 32 tokens and count how often this window reoccurs across other documents. High duplication is typically associated with lower TF-IDF scores due to higher document frequency values.

B. Document-Intrinsic Features

Some documents are inherently more likely to match queries due to their structure, content type, or repetition. We evaluate these intrinsic factors using handcrafted heuristics.

C. Cosine Similarity and Scoring Behavior

The primary model behavior we assess is the final cosine similarity score between a query and a document. This score reflects both the content overlap and TF-IDF weighting across the two vectors.

D. Summary

The above features—ranging from token frequency to cosine score variance—offer insights into how and why certain documents are retrieved or overlooked. This analysis lays the groundwork for future improvements, including adaptive weighting, reranking strategies, and hybrid scoring models that account for both surface-level and semantic features.

V. Methodology Used for Information Retrieval

A. TF-IDF Based Vector Space Model

The core of the Information Retrieval system relies on the classical vector space model, wherein both documents and user queries are represented as vectors in a high-dimensional term space. The Term Frequency-Inverse Document Frequency (TF-IDF) method is used to assign appropriate weights to terms, ensuring that commonly occurring terms are down-weighted, and rare, more informative terms are emphasized. Cosine similarity is then employed to calculate the angle between the query vector and each document vector, yielding a ranked list of results.

Let $D = \{d_1, d_2, ..., d_n\}$ represent a set of documents and Q be the user query. Each document d_i and the query Q are converted to vector representations $\vec{d_i}$ and $\vec{q}$ respectively, using the following equations:

$$TF(t,d) = 1 + \log_{10}(f_{t,d}), \quad if f_{t,d} > 0 \tag{1}$$

$$IDF(t) = \log_{10}\left(\frac{N}{df_t}\right) \tag{2}$$

$$TFIDF(t,d) = TF(t,d) \cdot IDF(t) \tag{3}$$

$$CosineSimilarity : \cos(\theta) = \frac{\vec{q} \cdot \vec{d}}{||\vec{q}|| \cdot ||\vec{d}||} \tag{4}$$

Where $f_{t,d}$ is the raw frequency of term t in document d, df_t is the number of documents containing term t, and N is the total number of documents.

B. Preprocessing Pipeline

The preprocessing block ensures that only informative, meaningful terms contribute to the TF-IDF vectors. The steps involved are:

1) **Tokenization:** Raw text is split into individual words using NLTK's `word_tokenize()`.
2) **Stopword Removal:** Standard English stopwords are removed using the NLTK corpus, with custom exclusions for terms like *to*, *in*, and *where*, which hold semantic importance in user queries.
3) **Punctuation and Symbol Filtering:** Non-alphabetic tokens, punctuation marks, and numerals are discarded.
4) **Length Filtering:** Tokens shorter than three characters (e.g., 'p', 'i', 'u') are removed as noise.
5) **Normalization:** All remaining tokens are lowercased.
6) **Stemming:** Porter stemming is applied for conflating word variants.

C. TF-IDF Matrix Construction

After preprocessing, a term-document matrix is constructed. Each document is represented as a vector of TF-IDF scores. The matrix is stored in a Pandas DataFrame, where rows correspond to terms and columns to document IDs.

The system normalizes each vector using its L2 norm. This facilitates cosine similarity computation with the query vector, ensuring scale-invariant comparison.

D. Query Handling and Ranking

When a query is received, it undergoes the same preprocessing steps as the documents. The resulting query vector is then compared with each document vector using cosine similarity. Documents are ranked in descending order of similarity.

The system retrieves the top-k documents, typically $k = 10$, and their associated links and metadata are fetched from a PostgreSQL database. These results are returned to the frontend.

E. Frontend and Backend Integration

The IR system is equipped with a web-based frontend where users can:

- Login or sign up using an authentication portal
- Submit search queries
- View top-ranked document results with clickable links

The backend is connected to a PostgreSQL database that stores:

- Document IDs and titles
- TF-IDF preprocessed vectors
- Associated document links and metadata

APIs or internal function calls retrieve the relevant document links based on the query results and render them in the frontend dynamically.

F. System Structure and Flow

The flow of the system can be summarized as:

1) User submits a query via the frontend interface.
2) The query is sent to the backend and preprocessed.
3) Cosine similarity is calculated between the query and all document vectors.
4) Top-k documents are ranked and selected.
5) Corresponding links are fetched from the database.
6) Results are displayed on the frontend.

G. Functional Blocks

1) **preprocessing(file_path)**: Reads and cleans the text file, returning a list of useful tokens.
2) **build_tfidf(documents)**: Computes and normalizes the TF-IDF matrix.
3) **handle_query(query)**: Converts the user query into a vector and ranks documents based on cosine similarity.
4) **retrieve_links(doc_ids)**: Pulls document links from the PostgreSQL backend.
5) **frontend_interface**: Accepts user query input and renders the results.

VI. CONCLUSIONS

We developed and implemented a modular Information Retrieval (IR) system based on classical TF-IDF weighting and cosine similarity. This approach represents both documents and queries in a shared vector space and computes their similarity to retrieve relevant matches. The system supports preprocessing steps such as tokenization, stopword removal, punctuation filtering, and document vectorization.

The IR system was tested on a corpus of plain-text documents and successfully retrieved top-k documents for a wide range of query types. Integration with a PostgreSQL backend and a web-based frontend enables seamless user interaction and real-time result delivery.

While the system is effective in ranking documents based on surface-level lexical overlap, there remain opportunities to enhance retrieval quality:

1) To reduce false positives in ranked results, the system can be extended to incorporate semantic filtering or reranking using contextual embeddings (e.g., BERT or SBERT).
2) Instead of relying solely on a single TF-IDF vector per document, we can store multiple vector representations—e.g., by splitting documents into sections or segments—which may improve granularity and retrieval accuracy for long documents.

3) Incorporating query expansion techniques can improve recall by including synonyms or related concepts in the search process.
4) The current top-k ranking can be augmented with user feedback loops or adaptive scoring strategies to better personalize or refine results.

Overall, the presented IR system serves as a strong baseline for retrieval tasks and offers a robust foundation for further research and development into semantic IR, multilingual support, and real-world deployment use cases.

REFERENCES

[1] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, 2008.

[2] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, O'Reilly Media, 2009.

[3] L. Gao et al., "The Pile: An 800GB Dataset of Diverse Text for Language Modeling," in *arXiv preprint arXiv:2101.00027*, 2020.

[4] K. Sparck Jones, "A Statistical Interpretation of Term Specificity and Its Application in Retrieval," *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[5] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *EMNLP*, 2019.

[6] G. Salton, A. Wong, and C. S. Yang, "A Vector Space Model for Automatic Indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[7] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," in *ACL System Demonstrations*, 2014.

[8] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, 2010.

[9] PostgreSQL Global Development Group, "PostgreSQL: The World's Most Advanced Open Source Relational Database," https://www.postgresql.org/, accessed May 2025.

[10] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.