IR - PROJECT

Information Retrieval Search Engine

Presented by:

- Ido Badash (badasi@post.bgu.ac.il)
- Shani Zilberberg (shanizil@post.bgu.ac.il)

Search Engine Architecture Overview

Our Flask-based search system processes queries through a multi-faceted architecture, combining various indexing and ranking techniques to deliver highly relevant results.

Title Indexing

Matches query terms against document titles for direct and high-relevance initial hits.

Body Search (TF-IDF)

Utilizes TF-IDF (Term Frequency-Inverse Document Frequency) to evaluate term importance within the main document content.

Anchor Text Analysis

Incorporates text from incoming links to assess a page's relevance and topic, often capturing external context.

Ranking Signals

Applies PageRank and PageViews to prioritize results based on authority, popularity, and estimated user engagement.

This modular design allows for comprehensive query processing, from raw input to a refined list of ranked results.

Main Search Ranking Formula

Ranking Algorithm

The main search functionality (`/search`) uses a composite scoring formula to rank relevance. The score for a document d given a query q is calculated as:

$$Score(d, q) = \sum_{term \in q} (W_{title} \times \mathbb{1}_{title}) + (W_{PR} \times \log(PR_d + 1)) + (W_{PV} \times \log(PV_d + 1))$$

Ranking Weights:

- **Title Match (W_{title}):** `5.0` (Heaviest weight for direct title relevance).
- **PageRank (W_{PR}):** `2.0` (Log-smoothed importance).
- **PageViews (W_{PV}):** `0.001` (Log-smoothed traffic signal).

Additional Logic:

- **Title Boost:** If the exact query appears in the title, the score is multiplied by 3.
- **Body Search:** Uses Cosine Similarity on TF-IDF vectors.

Specialized Search Methods

Three main search endpoints:

1

search_title()

- Returns ALL documents where query terms appear in title
- Ranking: By number of DISTINCT query words matched
- Formula: score = count of distinct matched terms
- Use case: Exact topic matching, high precision

2

search_anchor()

- Returns ALL documents ranked by anchor text matches
- Ranking: (distinct_terms_count × 10000) + total_frequency
- Prioritizes documents with more distinct query terms in anchor text
- Use case: Authority and external reference validation

3

search_body()

- Uses TF-IDF and cosine similarity (as shown in previous slide)
- Returns top 100 results
- Most computationally intensive
- Use case: Deep content relevance matching

Key Implementation Details:

- All methods use the same tokenizer (no stemming, stopword removal)
- Posting lists read from GCS binary files for efficiency

GCS Integration & Performance

Our search engine leverages Google Cloud Storage for robust data management and implements key optimizations to ensure high performance and scalability.

Cloud Storage Architecture

- All indices stored in Google Cloud Storage (GCS) bucket: badashbucket1
- Binary posting lists for efficient memory usage
- Pickle files for index metadata and document mappings

Data Loading

- Indices loaded at startup from GCS
- Body Index, Title Index, Anchor Index
- Document titles mapping, PageRank scores, PageViews data

Performance Optimizations

- Posting lists read directly from GCS binary files with offset seeking
- Memory limits: 10,000 postings per term to prevent overflow
- Pre-computed title term-to-docs mapping for fast title searches
- Flask app runs on port 5000 with multiprocessing for concurrent requests

Results

- Average retrieval time: ~3-5 seconds per query
- Harmonic mean of precision@5 and F1@30: 0.6415 (final version)
- Scalable architecture supporting Wikipedia-scale document collections