Comprehensive Design and Evaluation of an Information Retrieval System with Vector Space Model, BM25, and Dirichlet Language Model on the Cranfield Collection

Aaqid Masoodi
Dublin City University
Dublin, Ireland
aaqid.masoodi2@mail.dcu.ie

ABSTRACT

This report presents a detailed implementation and evaluation of an Information Retrieval (IR) system developed in Python for the Cranfield collection, a dataset of 1,400 aeronautical abstracts, 225 queries, and 1,837 relevance judgments. The system features an inverted index constructed with selective preprocessing and three ranking models: Vector Space Model (VSM), BM25, and a Dirichlet-smoothed Language Model (LM). Key design choices include domain-aware tokenization, title weighting, and multiprocessing for efficiency. Performance is assessed using `trec_eval`, yielding MAP scores of 0.0804 (VSM), 0.0807 (BM25), and 0.0785 (LM), with P@5 and NDCG metrics indicating modest early precision. The low MAP suggests potential preprocessing or ranking issues, explored herein. The full source code is hosted at https://github.com/aaqidmasoodi/MOSearchEngine.

CCS CONCEPTS

• **Information systems** → **Information retrieval**.

KEYWORDS

Information Retrieval, Inverted Index, Vector Space Model, BM25, Language Model, Cranfield Collection, TREC Evaluation, Python

ACM Reference Format:
Aaqid Masoodi. 2025. Comprehensive Design and Evaluation of an Information Retrieval System with Vector Space Model, BM25, and Dirichlet Language Model on the Cranfield Collection. In *Proceedings of CSC1121 Assignment 1.* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Information Retrieval (IR) systems are essential tools for retrieving relevant documents from large collections based on user queries. This project, developed as part of CSC1121 Assignment 1 at Dublin City University, implements a custom IR system for the Cranfield collection [1], a benchmark dataset comprising 1,400 aeronautical abstracts, 225 queries, and 1,837 relevance judgments. The system is built from scratch in Python, adhering to the assignment's prohibition on external IR libraries like Lucene or Solr, and utilizes NLTK for text processing and multiprocessing for performance optimization.

The architecture consists of three main components: 1. **Indexing**: Parses and indexes documents into an inverted index. 2. **Ranking**: Implements Vector Space Model (VSM), BM25, and a Dirichlet-smoothed Language Model (LM) for document scoring. 3. **Evaluation**: Assesses performance using `trec_eval` with metrics MAP, P@5, and NDCG.

CSC1121 Assignment 1, March 2025, Dublin City University
2025.

This report provides an exhaustive account of the system's design, implementation details, and evaluation results, structured as follows: Section 2 covers indexing, Section 3 details ranking, Section 4 presents evaluation, and Section 5 concludes with findings and future directions.

2 INDEXING

The indexing component transforms the Cranfield collection's XML documents (`cran.all.1400.xml`) into an inverted index, enabling efficient term-based retrieval. Hosted in `indexing.py` and `parser.py`, this module processes raw data into a searchable structure.

2.1 Document Parsing

The `parse_documents` function in `parser.py` uses `xml.etree.ElementTree` to extract `docno`, `title`, `author`, `bib`, and `text` from each document. The `clean_text` function normalizes text by:

- Replacing multiple spaces with a single space (`re.sub(r'\s+', ' ', text)`).
- Retaining alphanumeric characters and basic punctuation (`re.sub(r'[^a-zA-Z0-9\s\.,!?-]', '', text)`).
- Stripping leading/trailing whitespace.

This ensures consistency while preserving domain-specific content. Parsing yields 1,400 documents, verified by logging output.

2.2 Text Preprocessing

Preprocessing, defined in `preprocess.py`, prepares text for indexing:

- **Tokenization**: NLTK's `word_tokenize` splits text into words, converting to lowercase.
- **Stopword Removal**: Excludes standard English stopwords (e.g., "the") but retains aeronautical terms ("flow", "wing", "speed", "jet", "gas") using a custom set: `set(stopwords.words('english')) - {'flow', 'wing', 'speed', 'jet', 'gas'}`.
- **Stemming**: Applies Snowball stemming (`SnowballStemmer('english')`) only to words longer than four characters (`[stemmer.stem(w) if len(w) > 4 else w]`), preserving short technical terms like "jet".

This domain-aware approach balances normalization and specificity, critical for aeronautical abstracts [2].

2.3 Index Construction

The `build_inverted_index` function in `indexing.py` constructs the index:

- **Structure**: A `defaultdict(dict)` maps terms to dictionaries of document IDs and frequencies.
- **Title Weighting**: Title terms are duplicated (`title_terms * 2 + text_terms`) to emphasize their relevance.
- **Term Frequencies**: Computed per document using `defaultdict(int)`, then stored as a regular dictionary.
- **Document Frequencies**: Tracks how many documents contain each term.
- **IDF Calculation**: Uses a smoothed formula: $\log\left(\frac{N-df+0.5}{df+0.5}+1\right)$, where $N = 1400$.
- **VSM Vectors**: TF-IDF weights ($1 + \log(1 + tf) \cdot \text{IDF}$) are normalized by vector magnitude.
- **Statistics**: Includes document lengths, average length, and term counts.

The index is serialized to `index_data.pkl` with `pickle`, enabling reuse. Construction time is logged (e.g., 0.5–1 second), showcasing efficiency [3].

2.4 Design Choices

- **Selective Stemming**: Preserves short terms, enhancing recall for technical queries.
- **Title Weighting**: Reflects title importance, a common IR heuristic [4].
- `defaultdict`: Optimizes memory and speed for sparse term distributions.
- **Caching**: Reduces runtime, justified by iterative development needs.

3 RANKING

The ranking module, in `ranking.py`, processes queries from `cran.qry.xml` using three models, each with distinct scoring mechanisms.

3.1 Vector Space Model (VSM)

VSM, implemented in the `VectorSpaceModel` class, represents documents and queries as TF-IDF vectors:

- **Initialization**: Uses precomputed `doc_vectors` and `term_idf` from the index.
- **Query Vector**: Computes sublinear TF ($1 + \log(1 + tf)$) and IDF, normalized by magnitude.
- **Scoring**: Cosine similarity ($\sum(q_t \cdot d_t)$) ranks documents.

Its simplicity assumes term independence, effective for term-matching tasks [4].

3.2 BM25

The `BM25` class enhances VSM with term saturation and length normalization:

- **Parameters**: $k1 = 1.2$ (term frequency scaling), $b = 0.5$ (length normalization).
- **IDF**: $\log\left(\frac{N-df+0.5}{df+0.5}+1\right)$.
- **Scoring**:

$$\text{score} = \sum_{t\in Q} \text{IDF}_t \cdot \frac{\text{TF}_t \cdot (k1+1)}{\text{TF}_t + k1 \cdot (1 - b + b \cdot \frac{\text{doc\_length}}{\text{avg\_doc\_length}})} \cdot w_t$$

  where w_t is query term frequency.
- **Optimization**: Precomputes term frequencies and lengths via `compute_doc_term_freqs`.

Chosen parameters align with IR standards [5].

3.3 Language Model (LM)

The `LanguageModel` class uses Dirichlet smoothing ($\mu = 500$):

- **Initialization**: Computes collection-wide term frequencies and lengths.
- **Scoring**:

$$\text{score} = \sum_{t\in Q} \log\left(\frac{\text{TF}_t + \mu \cdot P(t|C)}{\text{doc\_length} + \mu}\right)$$

  where $P(t|C) = \frac{\text{collection\_TF}_t}{\text{collection\_length}}$.
- **Smoothing**: $\mu = 500$ balances document and collection probabilities, tuned for small collections [6].

3.4 Implementation Details

- **Query Processing**: Matches document preprocessing for consistency.
- **Output**: Top 1000 results per query in TREC format (`query_id Q0 doc_id rank score model_name`).
- **Multiprocessing**: `main.py`'s `evaluate_single_query` uses `Pool` to parallelize across CPU cores, reducing evaluation time (e.g., 10–20 seconds for 225 queries).
- **Debugging**: Logs top 10 results per model for each query, aiding verification.

4 EVALUATION

The system is evaluated on the Cranfield collection, with results analyzed using `trec_eval`.

4.1 Methodology

- **Data**: Parses 1,400 documents, 225 queries, and 1,837 qrels from `cran.all.1400.xml`, `cran.qry.xml` and `cranqrel.trec.txt`.
- **Pipeline**:
  (1) Loads or builds index.
  (2) Initializes VSM, BM25, and LM.
  (3) Ranks all queries, saving results to `results/{model}_results.txt`.
  (4) Runs `trec_eval -m map -m P.5 -m ndcg`.
- **Metrics**: MAP (overall precision), P@5 (top-5 precision), NDCG (ranking quality).

4.2 Results

4.3 Discussion

The MAP scores (0.0785–0.0807) are lower than typical Cranfield benchmarks (0.25–0.35) [5], suggesting issues:

Table 1: Evaluation Results on Cranfield Collection

Model	MAP	P@5	NDCG
VSM	0.0804	0.2000	0.3328
BM25	0.0807	0.2400	0.3261
LM	0.0785	0.2400	0.3241

- **Preprocessing**: Selective stemming might miss variants (e.g., "aeroelastic" to "aeroelast"), reducing recall. Logging tokenized terms for query 1 ("what similarity laws...") shows terms like "similar", "law", "obey", matching qrels (e.g., doc 184), but variants may be lost.
- **Ranking**: VSM's term independence struggles with short queries. BM25's length normalization helps top ranks (P@5 = 0.2400), but MAP indicates deeper ranking issues. LM's $\mu = 500$ may over-smooth, diluting relevance.
- **Data Alignment**: Qrels match document IDs, but sparse relevance (e.g., 29 relevant for query 1) amplifies ranking errors.
- **Debugging Insight**: Top-10 logs show relevant docs (e.g., 184) ranked low, suggesting score scaling or normalization flaws.

P@5 and NDCG are more promising, indicating some early precision and ranking consistency.

5 CONCLUSIONS

This IR system, hosted at https://github.com/aaqidmasoodi/MOSearchEngine, fulfills CSC1121 requirements with an efficient inverted index and three ranking models. BM25's slight MAP advantage (0.0807) highlights its robustness, while low scores suggest preprocessing or tuning opportunities. Strengths include multiprocessing and caching; weaknesses lie in ranking depth. Future enhancements could include lemmatization, parameter tuning (e.g., $\mu = 1000$), or query expansion. This work provides a solid IR foundation, ripe for refinement.

REFERENCES

[1] O. Ben Khiroun, "Cranfield TREC Dataset," GitHub, 2023, https://github.com/oussbenk/cranfield-tredataset.

[2] M. F. Porter, "An Algorithm for Suffix Stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[3] T. H. Cormen et al., *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.

[4] G. Salton et al., "A Vector Space Model for Automatic Indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[5] S. E. Robertson et al., "Okapi at TREC-3," *NIST Special Publication*, 1994.

[6] C. Zhai and J. Lafferty, "A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval," *SIGIR*, 2004.