COL764 – Assignment 2

Priyanshu Agrawal (Entry No: 2022CS11641)
Vishakha Agarwal (Entry No: 2022CS11097)

September 11, 2025

1 Introduction

This assignment implements and evaluates boolean and ranked-retrieval methods over the provided CORD-19 dataset in `python`. We implemented (i) boolean phrase search over a positional inverted index (Task 2), (ii) the Vector Space Model (VSM) with cosine similarity (Task 3), and (iii) Okapi BM25 (Task 4). Effectiveness is reported as Precision, Recall, and F1. For ranked retrieval, we evaluate at cut-offs $k \in \{20, 200\}$, where k denotes the number of top documents to be returned.

2 Dataset

The dataset is `cord19-trec-covid-docs-a2`, provided in JSONL format. Each line is a JSON object with the fields: `doc_id`, `title`, `DOI`, `date`, and `abstract`. The total number of documents is 191,175. Queries (about 50) were provided separately, and relevance judgments (qrels) are used for evaluation.

3 Task 0 — Tokenization (spaCy)

Method. We apply spaCy's *default rule-based English tokenizer* (`spacy.blank("en")`) on the concatenation of all fields of each document (title, DOI, date, abstract). No stemming or lemmatization is performed. Tokens are used exactly as emitted by spaCy.

Output summary.

- `vocab.txt`: fixed vocabulary (one token per line).

4 Task 1 — Inverted Index Creation (positions, tf, df)

Format. We build a *positional inverted index* from the corpus. For each term t (restricted to the fixed vocabulary in `vocab.txt`), the index stores its *document frequency* $df(t)$ and postings list. For each document d containing t, the posting entry records:

$$\{\texttt{tf}, \texttt{pos[]}\}$$

- `tf`: the raw term frequency in d.
- `pos[]`: a strictly increasing list of 0-based token positions of t within the document text.

Document positions are computed by concatenating the fields `[title, doi, date, abstract]` in this order. Both terms and document IDs are serialized in *lexicographic order* for determinism.

Additional statistics. To support ranked retrieval models, the indexer also produces other index files:

- `bm25.json`: contains corpus-wide statistics required for BM25 scoring:
  - `N`: total number of documents.
  - `avgdl`: average document length.

  - `doc_len`: per-document tokenized length.
  - `hyperparams`: default BM25 parameters ($k_1 = 1.5, b = 0.75, k_3 = 0, idf\_clamp\_zero = true$).

- `vsm.json`: contains vector space model data for cosine similarity:
  - `idf[t]`: inverse document frequency for each term, $\log(N/df)$.
  - `postings[t][d]`: raw tf values for each term–document pair.
  - `doc_norms[d]`: precomputed ℓ_2-norm of the weighted term vector for each document.

Output summary.

- `vocab.txt`: fixed vocabulary (one token per line).
- `index.json`: full positional inverted index with df, tf, and positions.
- `bm25.json`: BM25 corpus statistics.
- `vsm.json`: VSM weights and document norms.

5 Task 2 — Boolean Phrase Search

Description.
This task implements exact phrase search using the positional inverted index from Task 1. Each query is tokenized with `spaCy`'s rule-based English tokenizer (`spacy.blank("en")`), with no stopword removal or further normalization. For multi-term queries we verify exact phrases by performing positional intersection across adjacent term position lists; only documents that contain the consecutive token sequence are returned.

Scoring and Ranking.
Being a boolean phrase retrieval model, all matching documents receive a constant score of 1. Matches are ordered by lexicographic `doc_id`, and ranks are assigned sequentially ($1..N$). Output follows TREC format: `qid docid rank score`.

Table 1: Phrase Search effectiveness

Method	Precision	Recall	F1
Phrase Search	0.1348	0.0012	0.0023

6 Task 3 — Vector Space Model (VSM)

Description.
This task implements a ranked retrieval model using the Vector Space Model. Each query is tokenized with `spaCy`'s rule-based English tokenizer (without stopword removal or stemming). Queries are title-only by default, as specified in the assignment. For each query term, log-normalized TF–IDF weights are computed, and documents are scored using cosine similarity between the query and document vectors stored in the precomputed VSM index.

Scoring.
Term weights are calculated as

$$w_{t,q} = (1 + \log tf_{t,q}) \cdot \mathrm{idf}(t), \quad w_{t,d} = (1 + \log tf_{t,d}) \cdot \mathrm{idf}(t),$$

where $tf_{t,d}$ is the raw term frequency in document d and df_t is the document frequency of term t. The similarity score is

$$\mathrm{score}(q, d) = \frac{\sum_{t \in q \cap d} w_{t,q}\, w_{t,d}}{\|q\| \cdot \|d\|}.$$

Output.
For each query, the top-k ranked documents are returned in descending similarity score. Results are written in TREC format with fields `qid docid rank score`.

Table 2: VSM effectiveness at different number of required documents

Method	k	Precision@k	Recall@k	F1@k
VSM	20	0.3290	0.0165	0.0310
VSM	200	0.3061	0.1345	0.1783

7 Task 4 — BM25 Retrieval

Description.
We implement BM25 retrieval to rank documents for given queries. BM25 is a probabilistic retrieval model that scores documents based on term frequency, document length, and inverse document frequency. Queries are restricted to the title field, tokenized using spaCy's rule-based English tokenizer. Stopwords are accepted but ignored per assignment instructions. For each query, we retrieve the top-k documents in TREC format (qid, docid, rank, score).

Formula.
Standard Okapi BM25 with parameters k_1 and b:

$$\text{score}(q,d) = \sum_{t \in q} \text{idf}(t) \cdot \frac{tf_{t,d} \cdot (k_1+1)}{tf_{t,d} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)},$$

$$\text{idf}(t) = \max\left\{0,\ \log \frac{N - df_t + 0.5}{df_t + 0.5}\right\}.$$

Here N is the number of documents, df_t is the document frequency of term t, $|d|$ is the document length, and avgdl is the average document length. Negative idf values are clamped to zero, consistent with the implementation. A small safeguard is applied to the denominator to avoid division by zero. Throughout this report, k denotes the retrieval cut-off (20 or 200), not a BM25 parameter.

Hyperparameter Tuning.
We sweep (k_1, b) and visualize macro-F1 with heatmaps for $k = 20$ and $k = 200$. The best parameter settings observed were: $k = 20$: best $k_1 : 0.8$, $b : 0.08$; $k = 200$: best $k_1 : 1.618$, $b : 0.498$.
Scoring and Ranking.
Each document receives a BM25 score for the query; higher scores indicate higher relevance. Documents are sorted in descending score order; ties are broken by lexicographic document ID. Only the top-k documents are listed for each query.

Table 3: BM25 effectiveness at different cut-offs

Method	k	k_1	b	Precision	Recall	F1
BM25	20	0.8	0.08	0.6350	0.0290	0.0547
BM25	200	1.618	0.498	0.3920	0.1641	0.2200

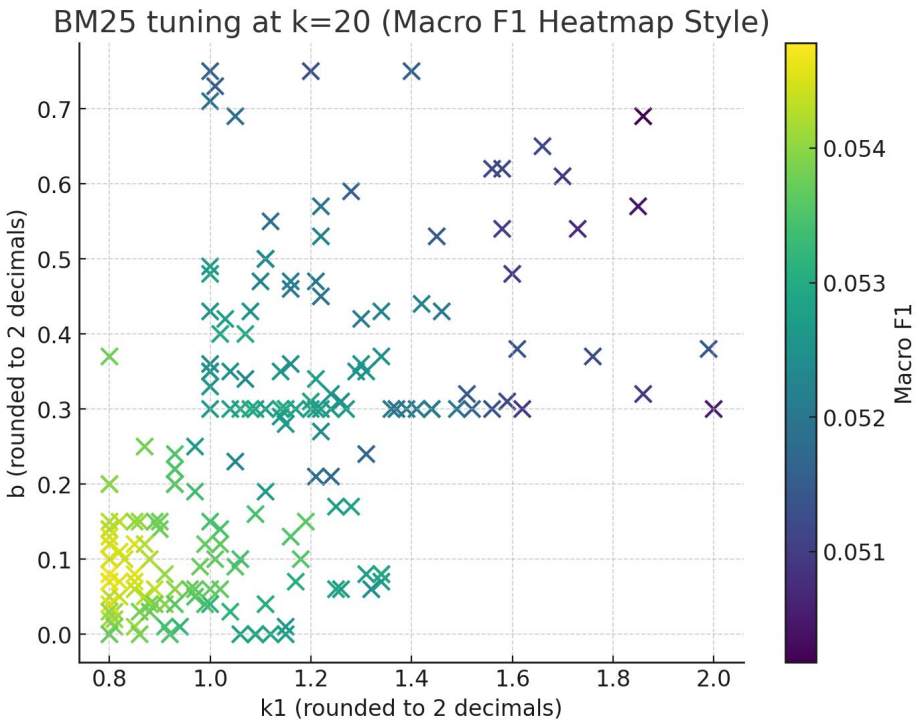

Figure 1: BM25 hyperparameter sweep at $k = 20$ (k_1 vs b).

8 Task 5 — Pseudo-Relevance Feedback (PRF)

Description. We implement one-round pseudo-relevance feedback (PRF) over the VSM retrieval model. The workflow is as follows: (1) retrieve top-k documents using VSM; (2) treat the top-R retrieved documents as pseudo-relevant and compute a feedback vector; (3) update the original query using Rocchio-style combination with the feedback vector; (4) re-retrieve documents once using the expanded query. Stopwords are ignored, and queries are restricted to the title field.

Query modification. Let the original query vector be $\vec{q}_0$, with term weights

$$w_{t,q_0} = (1 + \log tf_{t,q}) \cdot \mathrm{idf}(t).$$

The query expansion process proceeds as follows:

1. **Initial retrieval:** Compute scores of all documents using cosine similarity between $\vec{q}_0$ and document vectors $\vec{d}$:
$$\mathrm{score}(q_0, d) = \frac{\sum_{t \in q \cap d} w_{t,q_0}\, w_{t,d}}{\|\vec{q}_0\| \cdot \|\vec{d}\|}.$$
2. **Select top-R feedback documents:** Take the R highest-scoring documents as pseudo-relevant.
3. **Compute feedback centroid:** Average term weights across the top-R documents:
$$\vec{c}_{\text{feedback}}[t] = \frac{1}{R} \sum_{d \in \text{top-}R} w_{t,d}.$$
4. **Select top-m feedback terms:** Retain the m terms with highest weights from the centroid.

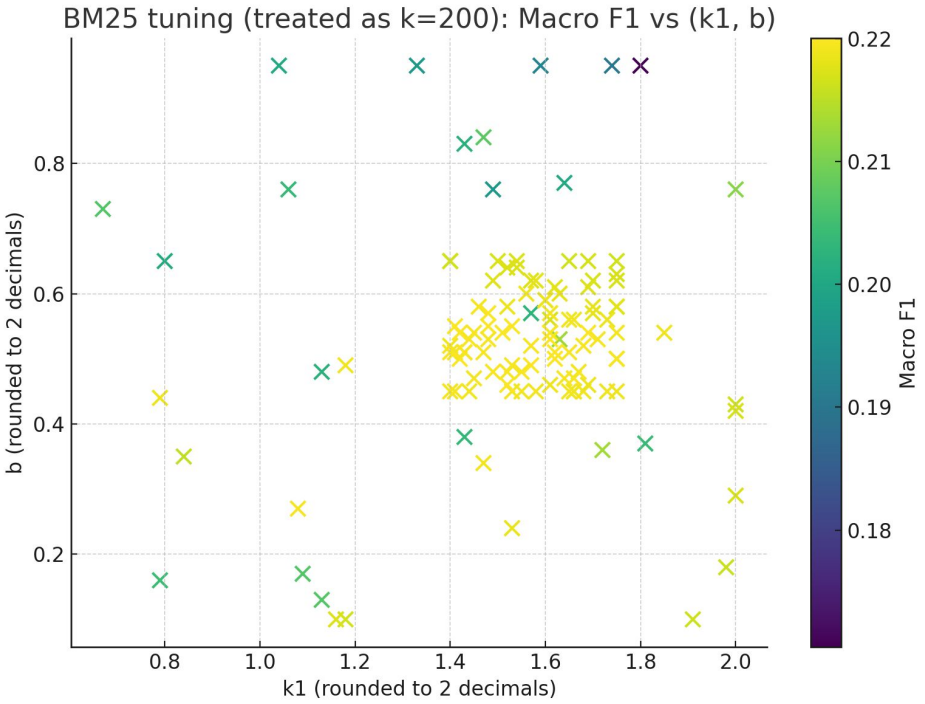

Figure 2: BM25 hyperparameter sweep at $k = 200$ (k_1 vs b).

5. **Rocchio update:** Combine the original query vector and feedback vector:

$$\vec{q}_{\text{new}}[t] = \alpha \cdot \vec{q}_0[t] + \beta \cdot \vec{c}_{\text{feedback}}[t],$$

where α and β control the contribution of the original and feedback components, respectively.

6. **Second retrieval:** Score all documents against $\vec{q}_{\text{new}}$ using cosine similarity to obtain the final ranking.

Hyperparameters used in our experiments: $R = 55$ feedback documents, $\alpha = 1.0$, $\beta = 0.8$, top-$m = 45$ feedback terms.

Scoring. Term weights for documents are log-normalized TF–IDF:

$$w_{t,d} = (1 + \log tf_{t,d}) \cdot \text{idf}(t),$$

where $tf_{t,d}$ is term frequency in document d and $\text{idf}(t)$ is the precomputed inverse document frequency. Cosine similarity between the query and document vectors determines document ranking.

Output. The top-k ranked documents after query expansion are written in TREC-eval format: `qid Q0 docid rank score PRF`.

Table 4: PRF (VSM) effectiveness at different cut-offs

Method	k	Precision	Recall	F1	Notes
PRF (VSM)	20	0.3920	0.0196	0.0369	one round
PRF (VSM)	100	0.3774	0.0876	0.1369	one round
PRF (VSM)	200	0.3498	0.1544	0.2041	one round

9 Discussion

- Phrase search returns exact matches only; effectiveness depends on strict adjacency constraints.
- VSM benefits from term weighting and can retrieve semantically related documents beyond exact phrases.
- BM25 typically improves early precision via length normalization (parameter b) and saturation (parameter k_1).
- Increasing k from 20 to 200 generally raises recall and may reduce precision; resulting F1 depends on the trade-off.

10 Implementation Notes & Reproducibility

Python 3.12 on Ubuntu. Only standard library and spaCy tokenizer are used. Evaluation (Precision/Recall/F1) is computed by a separate script after generating TREC-style run files. Shell scripts orchestrate each task; the stopwords argument is accepted but ignored end-to-end.