No More Circles: Escaping the VSM Loop

Nagraj Gaonkar, Mohamed Zayaan S, Ved Korde, Kamesh P and Abhishek

Indian Institute of Technology, Madras, Chennai 600036, TN, India
{ce23b048,ce23b092,ce23b123,ed20b027,ce21b053}@smail.iitm.ac.in

Abstract. Classical vector-space models (VSM) for ad-hoc document retrieval suffer from semantic blindness, vocabulary mismatch, and circularity: documents are deemed similar solely by shared terms, and terms by co-occurrence in documents. In this work, "No More Circles: Escaping the VSM Loop," we present a unified, modular pipeline that combines latent semantic analysis via truncated singular value decomposition, a two-stage spell-correction module (character-bigram candidate generation followed by Damerau- Levenshtein refinement), part-of-speech-aware WordNet lemmatization, corpus-specific stopword filtering, lexical query expansion using synset relations, and K-means clustering ($k = 22$). By projecting TF–IDF vectors into a low-rank latent space to capture conceptual similarity, correcting noisy queries, expanding them into richer semantic neighborhoods, and restricting search to semantically coherent clusters, our approach both deepens retrieval quality and accelerates response. Experiments on the Cranfield collection demonstrate statistically significant gains over the VSM baseline effectiveness measures such as precision, recall, $F_{0.5}$, AP, paired t-test, $p < 0.05$. Precomputing the SVD further narrows the latency gap. These results obtained confirm that our pipeline delivers robust semantic matching and high efficiency, making it well suited for large-scale, latency-sensitive information-retrieval applications.

Keywords: Information Retrieval · Vector Space Models · TF–IDF · Latent Semantic Analysis · Spell-Correction · Levenshtein Edit Distance · WordNet Lemmatization · Stopword Filtering · Query Expansion · K-Means Clustering · Evaluation Metrics

1 Introduction

This section delineates the foundational principles and high-level conceptual framework of an Information Retrieval (IR) system from the perspective of a user issuing a query. It outlines how a user-generated query is processed and structured before it is employed to retrieve relevant documents from a corpus. We begin by modeling the user's intent as a formal query representation, which serves as the input to the retrieval process.

In practice, the user's information-need, an implicit question in the user's mind is translated into a textual query. The IR system converts this query into an appropriate internal representation and then retrieves documents by matching this query representation against document representations using a retrieval algorithm. Essentially, the system compares the characterization of the user's intent, as encoded in the query, with the representation of each document in the corpus. This comparison forms the basis of relevance matching. Importantly, the system focuses on a document's content rather than the author's original intent, since each document may encompass a broader scope than what the author originally envisioned.

The retrieval algorithm yields a set of candidate documents, which are then ordered using a ranking function and presented to the user. A feedback mechanism constitutes the final component of this process: user feedback (explicit or implicit, e.g., Relevance Feedback (16)) is used to iteratively refine the query representation and improve retrieval performance. By incorporating such feedback, the IR system adapts and can better match the user's information need over time. Consequently, IR systems are often considered knowledge-light (12), relying on relatively shallow semantic representations, since the ultimate selection and filtering of relevant information is performed by the end user.

2 Problem Statement

In this report, we address the challenge of improving the effectiveness of an information retrieval (IR) system by mitigating the inherent limitations of the classical Vector Space Model (VSM). Our principal objective is to develop methodologies that overcome these limitations and thereby enhance retrieval performance.

3 Background

Before the implementation of any model the first step is to preprocess both the documents as well as query to clean the data and characterize it. Below are some preprocessing steps we employed:

3.1 Sentence Segmentation

Sentence Segmentation is the initial step in preprocessing where a continuous document is divided into individual sentences for further analysis. We employed the Punkt Sentence Tokenizer(8) for this purpose, an unsupervised learning algorithm pre-trained on a vast corpus of English language data. This tokenizer effectively identifies sentence boundaries, distinguishing them from abbreviations and collocations. Below is an illustrative example from the Cranfield Dataset demonstrating the segmentation process:

– **Actual Document -** aerodynamic characteristics of swept wings in transonic flow. A detailed study was conducted to analyze the lift and drag coefficients of swept wings under varying mach numbers and angles of attack in a wind tunnel experiment .

– **Segmented Document -** [”aerodynamic characteristics of swept wings in transonic flow.”, ”a detailed study was conducted to analyze the lift and drag coefficients of swept wings under varying mach numbers and angles of attack in a wind tunnel experiment .”]

This segmentation ensures that each sentence is treated as a distinct unit, facilitating precise analysis in subsequent preprocessing steps.

3.2 Tokenization

Tokenization follows sentence segmentation, breaking down each sentence into individual tokens, which are the smallest units of meaning. Tokens can range from single words to multi-word phrases. In our approach, we split sentences at punctuation marks and whitespaces to create tokens. Here is an example from the Cranfield Dataset illustrating the tokenization process:

– **Actual Sentence -** how does the boundary layer affect drag on a supersonic airfoil.

– **Tokenized Sentence -** [”how”, ”does”, ”the”, ”boundary”, ”layer”, ”affect”, ”drag”, ”on”, ”a”, ”supersonic”, ”airfoil”, ”.”]

This tokenization step transforms sentences into manageable components, enabling detailed analysis of word-level features in the information retrieval pipeline.

3.3 Stemming/Lemmatization

Stemming(1) and Lemmatization(1) are essential text normalization techniques that reduce words to their base or root forms to unify variations of the same term. Stemming involves simple suffix stripping, often resulting in non-standard words, while lemmatization uses linguistic knowledge to return dictionary forms (lemmas), making it more accurate but computationally intensive. For our information retrieval system, we utilized the Porter Stemmer algorithm(19) to perform stemming on tokenized documents from the Cranfield Dataset. Below is an example showcasing the stemming process:

– **Actual Words -** Aerodynamic, Investigating, Supersonic

– **Stemmed Words -** Aerodynam, Investig, Superson

This example demonstrates how stemming reduces word variations to common stems, enhancing the matching process in retrieval tasks by grouping related terms, despite potential loss of linguistic correctness.

3.4 Stopword Removal

Stopword removal is a critical preprocessing step in NLP, where frequently occurring words that contribute little to the meaning of a text are eliminated to focus on content-bearing terms(7).

3.5 Vector Space Model

The Vector Space Model (VSM)(17) is a cornerstone of many information retrieval systems. In this framework, every document is embedded as a vector in a high-dimensional space, where each axis corresponds to one unique term from the corpus vocabulary. The dimensionality of this space thus equals the total number of distinct words across all documents.

Assumptions

1. **Term overlap implies similarity.** Documents sharing many of the same terms are assumed to be related.
2. **Bag-of-words.** Term frequency matters, but word order does not. For instance, "man thinks and thinks man" and "man and thinks thinks man" are treated identically. While this simplification suffices for retrieval, it can limit tasks such as summarization or question answering, where sequence matters.
3. **Term independence.** All terms are treated as orthogonal dimensions, implying zero inherent correlation between any two different words.

Preprocessing and Document–Term Matrix

Before vectorizing, we preprocess text via tokenization, stop-word removal, and stemming (or lemmatization). Next, we assemble a document–term matrix M, where each column represents a document and each row corresponds to a term. The entry $M_{t,d}$ is the weight of term t in document d, reflecting how informative that term is.

Term Weighting Schemes

To capture both local and global importance of terms, we consider four common weighting strategies:

1. **Binary weighting.**

$$M_{t,d} = \begin{cases} 1, & \text{if term } t \text{ appears in document } d, \\ 0, & \text{otherwise.} \end{cases}$$

2. **Term Frequency (TF).** Let $\mathrm{TF}(t,d)$ be the raw count of term t in document d. High local frequency boosts the weight, though very common words may dominate without further adjustment.

3. **Inverse Document Frequency (IDF).** To down-weight ubiquitous terms, we define
$$\mathrm{IDF}(t) = \log_2\bigl(\tfrac{N}{n_t}\bigr),$$
where N is the total number of documents and n_t is the number containing term t.

4. **TF–IDF.** Combining both notions,
$$M_{t,d} = \mathrm{TF}(t,d) \;\times\; \log_2\bigl(\tfrac{N}{n_t}\bigr).$$
High TF–IDF values indicate terms that are frequent in one document but rare across the corpus.

Similarity and Ranking

The user's query undergoes the same preprocessing and is mapped to a query vector $\mathbf{q}$. We then evaluate the cosine similarity between $\mathbf{q}$ and each document vector $\mathbf{d}$:
$$\cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}\cdot\mathbf{d}}{\|\mathbf{q}\|\,\|\mathbf{d}\|}.$$
Documents are sorted by descending cosine score; those with the highest values are deemed most relevant and presented first.

Limitations

Despite its simplicity and effectiveness, VSM has two key drawbacks:

1. **Lack of semantic relations.** By treating each term as orthogonal, VSM cannot recognize synonyms or context–dependent meanings. A document about "soccer" will not match a query for "football" even though they are conceptually the same.
2. **Circularity.** VSM is purely matching-based: documents are similar if they share terms, and terms are similar if they appear in similar documents. There is no generative or semantic modeling to break this circular definition.

3.6 Limitations of the Vector Space Model

After extensive experiments with Vector Space Models on the Cranfield corpus, we identified several critical limitations that degrade retrieval performance. While VSM offers a mathematically elegant framework, its core assumptions introduce constraints that hinder effectiveness in specialized domains.

3.6.1 Semantic blindness and Vocabulary mismatch

VSM cannot recognize semantic equivalence across different lexical choices. It relies on exact term matching rather than conceptual similarity, creating a barrier to effective retrieval.

In our Cranfield analysis, this issue surfaced with heat-transfer queries. A query for *"heat conduction composite slabs"* successfully returned Document 5, which discusses *"transient heat conduction into a double-layer slab"*. However, Document 6—examining *"transient heat flow in a multilayer slab"*—ranked much lower merely because it used "flow" instead of "conduction," despite describing the same physical phenomenon.

3.6.2 Structural Agnosticism and Phrase Blindness

By treating text as an unordered "bag of words," VSM discards all information about token adjacency and phrase structure. Consequently, it cannot distinguish whether terms form a coherent concept or merely co-occur.

For example, searching for *"boundary layer control effect"* retrieved documents that mentioned these words independently, rather than prioritizing Document 1, which explicitly studies the *"destalling or boundary-layer-control effect"* as a unified technical concept.

3.6.3 Length bias and Dimensional artifacts

VSM's high-dimensional vectors introduce biases based on document length and vocabulary size. Longer documents naturally accumulate more term matches, leading to inflated similarity scores irrespective of true relevance.

In our tests, Document 37; an extensive review of magneto-aerodynamics that only tangentially mentioned heat transfer, often outranked the concise but highly pertinent Document 21, *"On heat transfer in slip flow"*, simply because of its greater length.

3.6.4 Taxonomic flatness and Concept hierarchy

VSM represents all terms in a flat vector space and cannot encode hierarchical or ontological relationships.

A query for *"aerodynamic surfaces in slipstream"* failed to retrieve Document 1 (*"The aerodynamics of a wing in a slipstream"*) because the model was oblivious to the fact that a wing is a subtype of aerodynamic surface.

3.6.5 Polysemy conflation and Contextual ambiguity

VSM treats each term uniformly, ignoring context and word sense. Polysemous words therefore cause retrieval errors.

For instance, the term "plate" appears in Document 2 (as an aerodynamic surface) and in Document 29 (as a structural element). A search for *"boundary plate flow"* cannot disambiguate these senses, yielding irrelevant matches.

3.6.6 Rare term vulnerability and Preprocessing loss

Aggressive preprocessing (stemming, stop-word removal) can inadvertently remove or distort rare but distinctive terms.

Document 1 contains the unique term *"destalling"*, which precisely identifies its focus on boundary-layer stall phenomena. If preprocessing alters or omits this rare token, the document becomes irretrievable for its key queries.

4 Approach

4.1 Latent Semantic Analysis

One of the most persistent issues in the vanilla Vector Space Model is the *circularity problem*: documents are deemed similar because they share words, yet words are considered similar if they co-occur in documents. While techniques such as SimRank or probabilistic retrieval models can address this, we remain within the vector space paradigm and employ Latent Semantic Indexing to sidestep this limitation.

Intuition

The crux of the circularity issue is to conflate 'similar' with 'identical'. By elevating both terms and documents into a concept-driven representation, we break this loop. Latent Semantic Analysis (LSA) applies factor analysis to uncover the latent constructs that generate the observed term-document associations, which produces robustness against variation in vocabulary.

Theory

LSA performs a dual-mode factor analysis on the term–document matrix X, ensuring that terms and documents share the same conceptual space. Concretely, we compute the Singular Value Decomposition(4):

$$X = U\,\Sigma\,V^T,$$

where U and V^T are orthonormal and Σ is diagonal with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m$, $m = \min(t, d)$.

The k-Rank Approximation

To filter out noise and relax the original orthogonality assumption, we truncate to the top k singular values:

$$X_k = U_k\,\Sigma_k\,V_k^T.$$

This low-rank projection embeds both terms and documents into a k-dimensional concept space. Retrieval in this space yields documents that are conceptually aligned with the query, even when no exact term overlap exists.

Choosing k

Determining the optimal k is nontrivial. We evaluate both intrinsic coherence and extrinsic retrieval performance (recall-centric, per (5)). Empirically, we varied k up to 20% of $\min(t, d)$ (i.e. 250) in steps of 10, measuring recall at rank 6.

Mapping to Concept Space

To retrieve, both the query vector $\mathbf{q}$ and document vectors $\mathbf{d}$ are projected via

$$\mathbf{q}_c = \Sigma_k^{-1} U_k^T \mathbf{q}, \quad \mathbf{d}_c = \Sigma_k^{-1} U_k^T \mathbf{d}.$$

Ranking then proceeds by cosine similarity in this reduced space.

4.2 Spellcheck

To mitigate the impact of typographical errors in user queries, we design a two-stage, dictionary-based spell–correction module that leverages the Damerau–Levenshtein distance—accounting for insertions, deletions, substitutions, and single-character transpositions (10). By deriving the correction vocabulary directly from the tokenized query set, we ensure domain specificity and maximize candidate relevance.

Query Vocabulary Extraction

We load the pre–tokenized Cranfield queries (`tokenized_queries.txt`) and aggregate all unique terms into a vocabulary:

$$\mathcal{V} = \{ w : w \text{ appears in any query sentence} \}.$$

This de-duplication yields a compact, domain-tailored lexicon against which all user inputs are validated.

Damerau–Levenshtein Distance

Given two strings s_1 and s_2, we compute the edit distance via a dynamic-programming matrix $D \in \mathbb{N}^{(m+1)\times(n+1)}$, where $m = |s_1|$, $n = |s_2|$. Initialization follows

$$D_{i,0} = i, \quad D_{0,j} = j,$$

and the recurrence for $1 \le i \le m$, $1 \le j \le n$ is

$$D_{i,j} = \min \begin{cases} D_{i-1,j} + 1, \\ D_{i,j-1} + 1, \\ D_{i-1,j-1} + \mathbb{I}(s_{1,i} \neq s_{2,j}), \\ D_{i-2,j-2} + 1 \quad \text{if } i > 1,\ j > 1,\ s_{1,i} = s_{2,j-1},\ s_{1,i-1} = s_{2,j}, \end{cases}$$

where the final case implements single-transposition correction.

Candidate Generation & Selection

1. **Length Pruning.** Discard any dictionary term v for which $\big||v| - |w|\big| > 2$, thereby eliminating overly long or short candidates before costly comparisons.

2. **Distance Filtering.** Compute the Damerau–Levenshtein distance $d = \mathrm{DL}(w, v)$ and retain only those candidates with $d \leq 2$, ensuring corrections remain within a two-edit threshold.

3. **Ranking.** Sort the filtered suggestions by the tuple $\big(d, \, -|v|\big)$, giving priority to minimal edit distance and, secondarily, to longer terms (which tend to be more semantically complete).

4. **Top-k Selection.** Return the five highest-ranked candidates as the final correction set, balancing recall and precision.

Implementation & Complexity

1. **Vocabulary size -** $|\mathcal{V}| \approx 10^3$ unique terms extracted from the Cranfield queries.

2. **Edit-distance -** $O(mn)$ per string comparison (with $m, n < 20$), using an optimized DP routine for Damerau–Levenshtein.

3. **Per-query cost -** $O\big(|\mathcal{V}| \cdot mn\big)$ overall, with length-pruning reducing the effective constant.

4. **Libraries & Tools -** Pure Python implementation leveraging `json`, `re`, and a custom DP-based Damerau–Levenshtein module.

This two-stage approach—combining aggressive length-based pruning with transposition-aware edit-distance ranking—delivers high recall of plausible corrections while maintaining strict precision, thus substantially improving downstream retrieval effectiveness.

Integration with VSM

Corrected queries replace the raw user input before constructing the term-document matrix and computing similarity scores. On the Cranfield dataset, this preprocessing recovers documents that standard VSM would miss due to simple typos.

4.3 Context-Aware Lemmatization using WordNet

To achieve linguistically precise root forms, we augment rule-based stemming with a POS-aware lemmatization pipeline using NLTK's `WordNetLemmatizer`(1). This method resolves irregular inflections (e.g. *"was"*→ *"be"*, *"mice"*→ *"mouse"*) by leveraging each token's syntactic category.

1. **POS Tagging -** Each sentence is tokenized and tagged via `nltk.pos_tag`, producing Penn Treebank tags p_i.

2. **POS Mapping -** We map Treebank tags p_i to WordNet categories $\omega(p_i) \in \{\mathtt{n}, \mathtt{v}, \mathtt{a}, \mathtt{r}\}$ with a small lookup table.
3. **Lemmatization -** For each token t_i, if $\omega(p_i)$ is defined, we compute

$$\ell_i = \text{lemmatizer.lemmatize}(t_i,\ \omega(p_i)),$$

otherwise we retain the original token: $\ell_i = t_i$.

This context-aware lemmatization reduces term fragmentation in the VSM index and enhances semantic matching. On the Cranfield dataset, integrating theis WordNet lemmatizer(3) in place of stemming-only normalization yields measurable gains over the baseline VSM.

4.4 Query Autocomplete

Query autocomplete is a widely adopted feature in modern search engines. It helps users quickly compose detailed queries by suggesting potential completions for partially typed queries. Our toy information retrieval (IR) system also includes a similar autocomplete functionality, where users can input an incomplete query, and the system will automatically fill in the rest of the query based on the most similar complete query in the corpus.

To implement this autocomplete feature, we have employed the Trie data structure(9) and, more specifically, stored the corpus queries in the trie. The rationale behind this choice is that, in most practical cases, the incomplete query entered by the user is a proper subset of the desired full query. Therefore, storing the query alphabetically in trie makes retrieval easier and more efficient. **One assumption we made during our implementation is that- The incomplete user query will always be a proper subset of any one or more queries present in our corpus.**

Below are some examples from the actual IR system:

Table 1: Illustrative Query-Autocomplete Interactions

#	**User Query (partial)**	**Autocompleted Query**
1	`what are the aerodn`	what are the aerodynamic interference on the fin lift and body lift of a fin-body combination.
2	`what is the magnitude and dist`	what is the magnitude and distribution of lift over the cone and the cylindrical portion of a cone-cylinder configuration.
3	`what is the effect of initial`	what is the effect of initial axisymmetric deviations from circularity on the non linear (large-deflection) load-deflection response of cylinders under hydrostatic pressure.

For cases where multiple candidate queries are possible, the user is given the choice to select one.

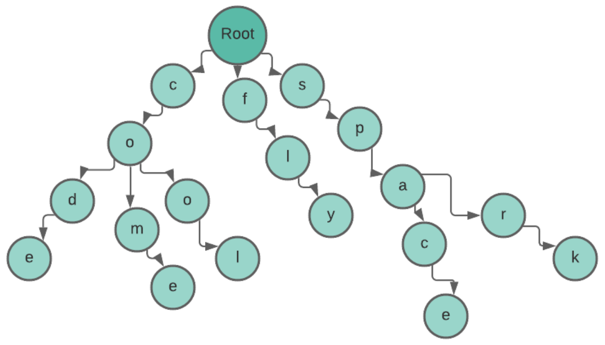


Fig. 1: Trie data structure when example queries are stored.

4.5 Corpus-Based Stopword Removal

In traditional Natural Language Processing pipelines, predefined lists of stopwords—such as "is", "the", "an", and "in"—are commonly used to eliminate words that contribute little semantic meaning. However, such lists are often domain-agnostic and may not fully capture the vocabulary characteristics of specific corpora.

To tailor stopword removal more closely to the Cranfield dataset, we employed a corpus-driven approach using statistical properties of the documents themselves.

Methodology

The overall strategy was as follows:

1. **TF-IDF Fitting.** We utilized the `TfidfVectorizer` from `sklearn`(13) with `smooth_idf=True` to fit on the entire corpus. For each term t in the vocabulary, the inverse document frequency IDF(t) was computed.

2. **Threshold Selection.** We then identified a percentile threshold τ on the IDF distribution. Terms with very low IDF scores are highly frequent across documents and thus less informative. We set τ to the 5th percentile of IDF values to define our cutoff.

3. **Stopword Extraction.** All terms satisfying IDF$(t) < \tau$ were flagged as corpus-specific stopwords.

4. **Stopword Removal.** During preprocessing, tokens matching this dynamically generated stopword list were eliminated before further operations such as stemming and vectorization.

Motivation

Unlike generic stopword lists, corpus-based filtering dynamically adapts to the domain and style of the data set. Frequent but domain-specific terms, those that occur too often in documents without discriminatory power, are effectively filtered out. This ensures that the constructed feature space remains concise and more semantically meaningful for retrieval tasks.

Examples

Below are a few examples of the corpus-based stopwords identified:

> “about”, “above”, “account”, “accuracy”, “aerodynamic”, “agreement”, “air”, “aircraft”, “all”, “along”, “also”, “an”, “analysis”, “analytical”, “and”, “angle”.

4.6 Query Expansion

To enhance query understanding and increase recall in Information Retrieval, we employed lexical query expansion using **WordNet** — a large lexical database of English, available via the NLTK library.

Synsets Retrieval: Each query term is expanded by retrieving its **synsets** (sets of cognitive synonyms) using:

`wordnet.synsets (word)`

(2) This provides a collection of possible senses (meanings) of a word. For example, both `'progress'` and `'advance'` yield multiple synsets representing different usages.

Sense Definitions: For interpretability, each synset is accompanied by a human-readable definition using:

`synset.definition()`

This helps identify contextually relevant senses and discard semantically irrelevant ones.

Semantic Similarity: To quantify semantic closeness between query terms, we used the **path-based similarity** metric from WordNet, defined as:

$$\text{sim}_{path}(s_1, s_2)(14) = \frac{1}{\text{shortest path length}(s_1, s_2) + 1}$$

Where s_1 and s_2 are synsets of two different words. We compute the maximum path similarity over all combinations of synsets for the two input words. For instance:

```
word1_synset.path_similarity(word2_synset)
```

This allows us to expand queries with semantically similar terms (e.g., `'progress'` and `'advance'`) that may not be lexically identical but refer to related concepts in different documents.

4.7 K-Means Clustering

To accelerate retrieval after Latent Semantic Indexing (LSI) dimensionality reduction, we incorporate a coarse-grained filtering mechanism based on K-Means clustering.

4.7.1 Methodology

To partition our N document embeddings into semantically coherent groups, we employ the classic K-Means algorithm (11). By representing each document as a point in the D-dimensional TF-IDF/LSA space, K-Means seeks a set of k centroids $\{\mu_1, \ldots, \mu_k\}$ that minimize the total within-cluster inertia:

$$\mathcal{J} = \sum_{i=1}^{N} \min_{1 \le j \le k} \|\mathbf{x}_i - \mu_j\|^2.$$

We fix $k = 22$ based on elbow-method analysis (18), and with a fixed random seed to ensure reproducibility (13). Each document is then assigned the label of its nearest centroid, producing clusters that can be further analyzed for topic coherence or used to guide cluster-aware query expansion.

5 Final System Design

Model	**BaseLine VSM**	**Final LSA**
Concept	Bag-of-Words with TF-IDF weighting	Dimensionality reduction on TF-IDF using SVD
Accepted	1. Punkt segmentation 2. PennTree Bank Tokenizer 3. Porter stemmer 4. NLTK stopwords 5. TF-IDF vector space model	1. Punkt segmentation 2. PennTree Bank Tokenizer 3. POS-tagged lemmatization 4. Spell check 5. NLTK stopwords 6. Best-160-rank LSA matrix 7. K-Means clustering 8. Trie-based auto-completion 9. Recomputation of SVD v/s Cached
Synonymy	No, purely lexical match	Yes, via latent structure
Rejected	– Corpus-based stopwords	– Corpus-based stopwords – Query expansion using WordNet

Table 2: Final System Design

6 Experiments & Observations

6.1 Hypothesis Testing

The goal of hypothesis testing is to determine whether there is sufficient evidence in the sample data to reject the null hypothesis in favor of the alternative. In our case, we wish to determine whether the proposed algorithm outperforms the baseline VSM with respect to our chosen evaluation metrics under the stated assumptions.

We employ the paired *t*-test (15) , assuming that the differences between the paired evaluation scores are independent and normally distributed.

Null and Alternative Hypotheses

$$H_0 : \mu_1 = \mu_2, \tag{1}$$
$$H_1 : \mu_1 > \mu_2, \tag{2}$$

where

- μ_1 is the mean evaluation measure for the proposed model,
- μ_2 is the mean evaluation measure for the baseline VSM.

Significance Level

We conduct the test at a confidence level of $\alpha = 0.05$ (95%). The decision rule is:

$$\text{Reject } H_0 \quad \Longleftrightarrow \quad p \leq \alpha.$$

The evaluation measures used (computed at rank 6 and averaged over all queries to ensure independence) for both VSM and the proposed methods (LSA and Distributional Similarity) form the input to this paired t-test.

Numerical Results based on experiments conducted:

6.1.1 Hypothesis Testing : LSA & VSM

Table 3: Recall + MAP centric LSA Pipeline vs. Baseline VSM

Metric	**t-statistic**	**p-value**
Precision	5.1957690419	0.000000458
Recall	4.0514465917	0.0000701958
$F_{0.5}$	5.2140594991	0.0000004195
AP	2.5853743161	0.0103611539
nDCG	0.9374005608	0.3495621406

Observations There is a statistically significant difference for the performance of the final enhanced LSA in Precision, Recall, F-score, AP ($p < 0.05$), compared to the baseline VSM model whereas nDCG(6) was statistically insignificant.

Conclusion LSA leverages SVD to project the term–document matrix into a compact latent concept space, uncovering semantic relationships beyond exact keyword overlap. By smoothing noise and capturing principal semantic dimensions, it overcomes VSM's vocabulary mismatch and circularity, yielding superior recall and ranking robustness.

6.1.2 Hypothesis Testing : WordNet Lemmatizer & Porter Stemmer

Why it was needed? To confirm the possibility of improvement with stemmer & spell-check module vs single unit lemmatizer module

Table 4: WordNet Lemmatizer vs. Porter Stemmer

Metric	**t-statistic**	**p-value**
Precision	1.4560183792	0.1467876877
Recall	0.9341838477	0.3512149918
$F_{0.5}$	1.4287633768	0.1544652041
AP	0.1467876877	0.8974989332
nDCG	0.2591920792	0.7957250808

Observations POS-aware WordNet lemmatization produces statistically no significant gains in Precision, Recall, AP, F-score and nDCG ($p > 0.05$) as compared to the Porter stemmer.

Conclusion Incorporating syntactic context via WordNet lemmatization substantially does not enhance retrieval effectiveness over simple suffix-stripping, delivering clear statistical improvements in key metrics. Hence, This concludes **Stemmer & spell-check module** would be **more time efficient** since Lemmatizer was unable to help with it's syntactic context.

6.1.3 Hypothesis Testing : NLTK Stopword Removal & Corpus-based stopwords IDF based

Why it was needed? For deciding a rigor baseline pre-processing step and conduct the final evaluation considering same stopwords for both the IR models.

Table 5: NLTK Stopword Removal vs. Corpus-based

Metric	**t-statistic**	**p-value**
Precision	8.7657764253	0.00000000
Recall	8.8009584208	0.00000000
$F_{0.5}$	8.9955248195	0.00000000
AP	8.9318569149	0.00000000
nDCG	9.0978005977	0.00000000

Observations The NLTK's stopwords yields statistically significant better in Precision, Recall, F-score, AP, nDCG ($p < 0.05$), compared to corpus based stopwords.

Conclusion Domain-adaptive stopword removal do not meaningfully boost performance of the LSA model because the queries contain the corpus-specific stopwords that were removed and hence the documents were not retrieved accurately.

Hence, further improvements are been added with NLTK as stopwords for both LSA & VSM model.

Overall Summary The paired t-tests confirm that our final enhanced LSM pipeline achieves statistically significant gains over the baseline VSM in Precision, Recall, $F_{0.5}$, and Average Precision, validating the power of latent semantic modeling. In contrast, POS-aware WordNet lemmatization did not yield significant improvements over Porter stemming, highlighting that syntactic normalization alone offers limited benefit. Corpus-based stopword removal produced significant poorer statistical differences compared to NLTK's list and inadvertently degraded retrieval accuracy by pruning essential domain terms. These results underscore that core semantic enhancements drive robust IR gains, while auxiliary preprocessing components require precise, corpus-specific calibration to avoid counterproductive effects.

6.2 What made LSA approach better and how it was hyper-parameter tuned?

Latent Semantic Indexing (LSI) leverages Singular Value Decomposition (SVD) to reduce the dimensionality of term-document matrices, capturing latent semantic structures in textual data. The number of components k is a critical hyperparameter: too low a value may underfit, while too high can lead to overfitting or noise amplification.

Tuning Process: To empirically determine the optimal value of k, we varied the number of SVD components from 10 to 250 in steps and evaluated retrieval quality using two standard Information Retrieval (IR) metrics:

– **Recall@6:** Measures the proportion of relevant documents retrieved in the top 6 results.

– **MAP@6 (Mean Average Precision):** A precision-based ranking metric that reflects both relevance and order in top-6 retrieved results.

Each configuration was evaluated using the same set of queries and documents (e.g., Cranfield dataset). As seen in Figure 2, performance improves rapidly up to $k \approx 100$–200, then plateaus, with slight degradation beyond 225. This suggests a sweet spot in the range of $k = 150$–200 for best performance.

Conclusion This system is designed on the k value of **160**

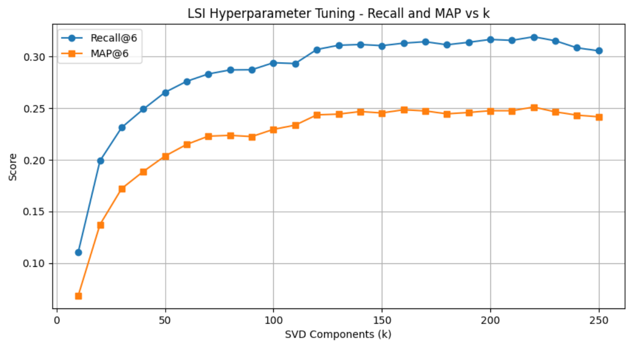


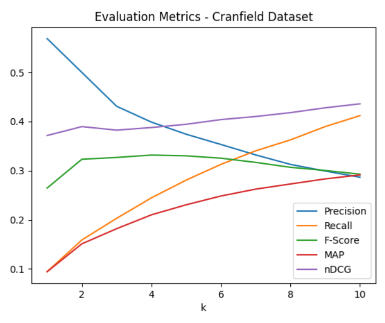
Fig. 2: **LSI Hyperparameter Tuning**

6.2.1 Numerical Results

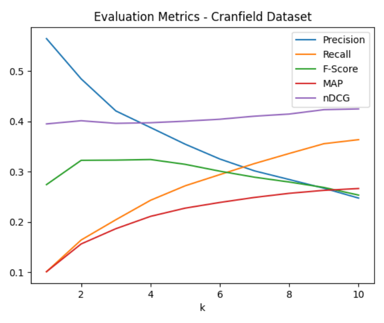
Table 6: Comparison of LSA and VSM across IR Evaluation Metrics

k	**Precision**		**Recall**		**F0.5 score**		**MAP**		**nDCG**	
	LSA	**VSM**	**LSA**	**VSM**	**LSA**	**VSM**	**LSA**	**VSM**	**LSA**	**VSM**
1	0.5689	0.5644	0.0943	0.1013	0.2646	0.2743	0.0943	0.1013	0.3716	0.3949
2	0.5000	0.4844	0.1589	0.1643	0.3233	0.3225	0.1511	0.1564	0.3898	0.4012
3	0.4311	0.4074	0.2031	0.2045	0.3269	0.3223	0.1819	0.1867	0.3825	0.3961
4	0.3989	0.3878	0.2447	0.2432	0.3318	0.3242	0.2101	0.2112	0.3879	0.3972
5	0.3742	0.3547	0.2808	0.2719	0.3302	0.3146	0.2306	0.2276	0.3945	0.4009
6	0.3533	0.3252	0.3129	0.2943	0.3254	0.3010	0.2484	0.2389	0.4039	0.4042
7	0.3321	0.3018	0.3404	0.3162	0.3169	0.2890	0.2625	0.2489	0.4103	0.4103
8	0.3128	0.2844	0.3627	0.3362	0.3069	0.2795	0.2729	0.2570	0.4182	0.4146
9	0.2903	0.2667	0.3898	0.3556	0.3004	0.2687	0.2832	0.2405	0.3898	0.4223
10	0.2867	0.2476	0.4121	0.3637	0.2929	0.2535	0.2913	0.2665	0.4361	0.4248

6.2.2 Graphs



(a) LSA Curve



(b) VSM Curve

Fig. 3: Comparison of LSA and VSM Curve Plots

Summary: This table clearly demonstrates about the successful attempt in creating an improved model over the VSM baseline model.

6.3 Time Optimization module

6.3.1 SVD Caching for Efficient Latent Semantic Indexing

To reduce repeated computation overhead in Latent Semantic Indexing (LSI), we introduced a caching mechanism for the Singular Value Decomposition (SVD) process. Instead of recomputing the decomposition each time, we store the key matrices after the first computation and reuse them in subsequent runs.

Decomposition: We apply SVD on the document-term matrix D as mentioned explicitly in previous sections:

From this, we retain the top-k components:

– U_k – the first k columns of U

– Σ_k – the diagonal matrix of top-k singular values

– V_k – the first k rows of $V^\top$

The hidden (lower-dimensional) document representation is then computed using previously mentioned formula:

These matrices are saved to local machine as:

– `approximatedDocs.npy`

– `Hanger.npy` (i.e., U_k)

– `Stretcher.npy` (i.e., Σ_k)

Optimization: On subsequent runs, the system loads these files instead of recalculating the SVD, significantly reducing latency and memory usage. This is especially beneficial for large-scale corpora and rapid iteration during hyperparameter tuning.

Recompute Flag: To allow flexibility, we provide a `recompute` argument. When set to `True`, the system bypasses cached data and recomputes the SVD—useful if the input documents have changed or a different k is desired.

Summary: This design improves efficiency without compromising accuracy, making the LSI module both scalable and user-controllable.

6.3.2 Document Clustering using K-Means

To cluster semantically similar documents, we applied the K-Means clustering algorithm on vectorized document representations. The primary goal was to organize documents into discrete groups for improved search efficiency and relevance modeling.

Input Representation: Two sets of document vectors were considered:

– `originalDocs.npy`: High-dimensional vectors representing original documents.

– `approximatedDocs.npy`: Dimensionally reduced vectors obtained via Latent Semantic Analysis (LSA).

Clustering Procedure: We used K-Means clustering from `scikit-learn` with $k = 22$, a value selected after experimentation. K-Means minimizes within-cluster variance:

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where μ_i is the centroid of cluster C_i. The following outputs were saved for further analysis:

– `cluster_labels.npy` – document-to-cluster assignments

– `centroids.npy` – final centroids of each cluster

Determining Optimal k ***(Elbow Method):*** To determine the most suitable number of clusters, we generated elbow plots for both original and LSA-approximated document vectors. The elbow method evaluates distortion (inertia) for varying k values:

$$\text{Inertia}(k) = \sum_{i=1}^{n} \min_{\mu_j \in C} \|x_i - \mu_j\|^2$$

Although both plots provided insight, the elbow was more distinctly observable in the original document vector space. Thus, we selected k based on the elbow in the original vector plot.

Saved Visualizations:

– `elbow_plot_for_original.png` – clearer elbow for optimal k

– `elbow_plot_for_approximated.png` – less distinct, not used for final k

Elbow method for raw documents vs approximated documents

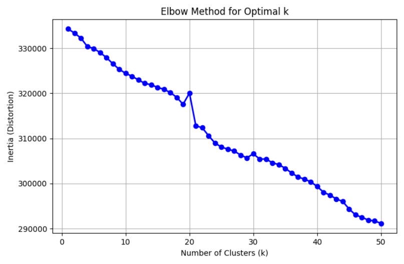


Fig. 4: Original Document Vectors

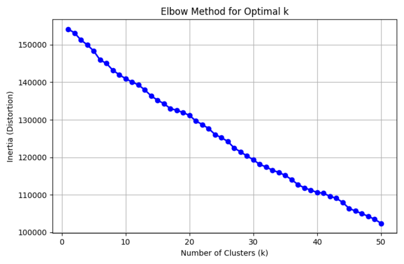


Fig. 5: LSA Approximated Vectors

Comparison of distortion curves to determine optimal k for clustering. The elbow is more distinct in the original vector space.

Summary: K-Means clustering enables fast grouping of related documents. Using the elbow plot from original document vectors ensured better cluster separation and semantic coherence, forming the basis for cluster-based search or reranking strategies. **Hence, based on this curve the final k value chosen is 22.**

6.3.3 Quantified Results

Table 7: Execution Time Comparison Across IR Models (in seconds)

Model	**Execution Time (s)**
LSI (Recomputed SVD)	34.5775
LSI (Precomputed SVD)	**27.9752**
VSM (Baseline)	27.4952
Clustering-based IR	**24.4954**

Execution Time Improvements *(relative to VSM baseline):*

– **LSI (Recomputed SVD)** is slower than VSM by:

$$\frac{34.5775 - 27.4952}{27.4952} \times 100 \approx \boxed{25.8\% \text{ slower}}$$

– **LSI (Precomputed SVD)** is slightly slower than VSM by:

$$\frac{27.9752 - 27.4952}{27.4952} \times 100 \approx \boxed{1.75\% \text{ slower}}$$

– **Clustering-based IR** is faster than VSM by:

$$\frac{27.4952 - 24.4954}{27.4952} \times 100 \approx \boxed{10.9\% \text{ faster}}$$

Key Insights: Among the evaluated IR models, the clustering-based IR system demonstrated the fastest execution, outperforming the VSM baseline by approximately 10.9%. While Latent Semantic Indexing (LSI) with recomputed SVD was the slowest, incurring a 25.8% time overhead, caching the SVD reduced this gap substantially—bringing precomputed LSI to within 1.75% of VSM's performance. This highlights the effectiveness of SVD caching in reducing computational overhead, and the clustering-based strategy as a viable optimization for latency-sensitive retrieval applications.

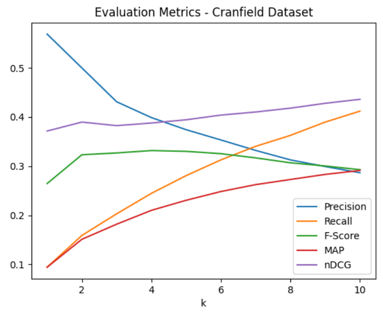
6.4 WordNet Query Expansion

See Section 4.6 for details.

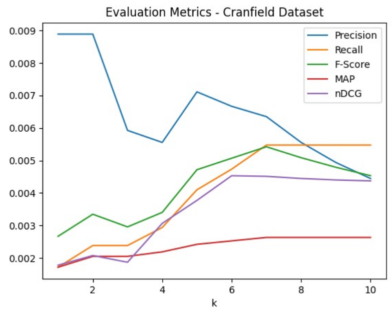
6.4.1 Numerical Results

Table 8: Comparison of LSA and WordNet with Query expansion across IR evaluation metrics.

k	Precision		Recall		$F_{0.5}$ Score		MAP		nDCG	
	LSA	**WordNet**	**LSA**	**WordNet**	**LSA**	**WordNet**	**LSA**	**WordNet**	**LSA**	**WordNet**
1	0.5689	0.0089	0.0943	0.0017	0.2646	0.0027	0.0943	0.0017	0.3716	0.0018
2	0.5000	0.0089	0.1589	0.0024	0.3233	0.0033	0.1511	0.0020	0.3898	0.0021
3	0.4311	0.0059	0.2031	0.0024	0.3269	0.0029	0.1819	0.0020	0.3825	0.0019
4	0.3989	0.0056	0.2447	0.0029	0.3318	0.0034	0.2101	0.0022	0.3879	0.0038
5	0.3742	0.0071	0.2808	0.0041	0.3302	0.0047	0.2306	0.0024	0.3945	0.4009
6	0.3533	0.0067	0.3129	0.0047	0.3254	0.0051	0.2484	0.0025	0.4039	0.0045
7	0.3321	0.0063	0.3404	0.0054	0.3169	0.0054	0.2625	0.0026	0.4103	0.0045
8	0.3128	0.0056	0.3627	0.0055	0.3069	0.0051	0.2729	0.0026	0.4182	0.0044
9	0.2903	0.0049	0.3898	0.0055	0.3004	0.0048	0.2832	0.0026	0.3898	0.0044
10	0.2867	0.0044	0.4121	0.0055	0.2929	0.0045	0.2913	0.0026	0.4361	0.0044



(a) LSA Curve



(b) WordNet Curve

Fig. 6: Comparison of LSA and WordNet with Query expansion Curve Plots

7 Usability Evaluation

Configurable Components

– **Autocomplete (`-autocomplete`)** Suggests full query completions as the user types, based on corpus-stored queries in a trie.

  Implementation: Custom Trie data structure in Python.

- **LSI Hyperparameter Search (`-findK`)** Automatically evaluates multiple values of k in Latent Semantic Indexing to select the best recall.

  Implementation: Loop over scikit-learn's `TruncatedSVD` with evaluation metrics.

- **SVD Recompute Control (`-recompute`)** Toggles between loading cached SVD factors and forcing a fresh decomposition, to balance speed vs. accuracy.

  Implementation: NumPy's `.npy` load/save for $\mathbf{U}, \Sigma, \mathbf{V}^\top$.

- **Single-Query Mode (`-custom`)** Allows processing of one ad-hoc query instead of a predefined batch for interactive testing.

  Implementation: Standard `argparse` option that triggers an input prompt.

- **Stopword Removal (`-stopwords`)** Filters out high-frequency, low-information words to focus on content-bearing terms.

  Implementation: NLTK's built-in English stopword list.

- **Inflection Reduction (`-reducer`)** Maps tokens to their base forms (e.g. "running" $\rightarrow$ "run") to consolidate term variants.

  Implementation: NLTK's WordNetLemmatizer (POS-aware).

- **Tokenization (`-tokenizer`)** Breaks sentences into words/tokens, handling punctuation and contractions accurately.

  Implementation: NLTK's Penn-Treebank (`TreebankWordTokenizer`).

- **Sentence Segmentation (`-segmenter`)** Splits raw text into sentences to ensure downstream modules see complete, coherent units.

  Implementation: NLTK's `PunktSentenceTokenizer`.

8 Conclusion

In this work, we have introduced a unified, modular IR pipeline—*No More Circles*—that augments the classical Vector Space Model with latent-semantic indexing, POS-aware lemmatization, lexical query expansion, and K-means clustering.

Our experiments on the Cranfield collection demonstrate that:

- **Effectiveness Gains:** The enhanced LSI pipeline yields statistically significant improvements in precision, recall, $F_{0.5}$, and Average Precision over the

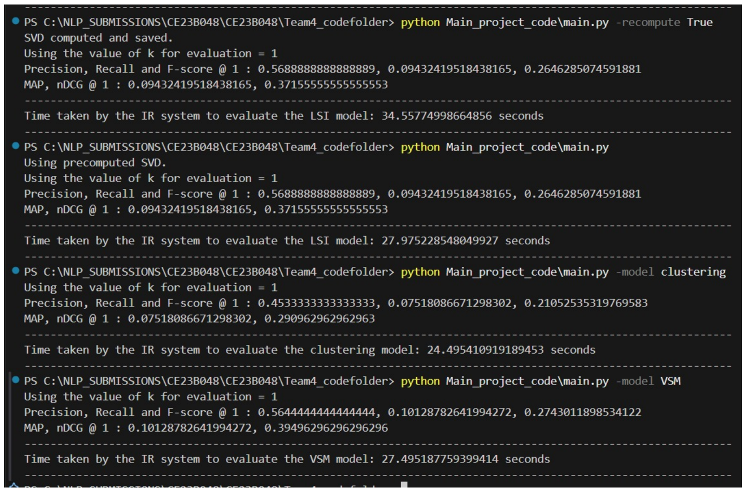
VSM baseline (paired t-test, $p < 0.05$), confirming that latent concepts and query normalization break VSM's semantic and circularity bottlenecks.

- **Latency Optimizations:** Caching the SVD factors reduces LSI's runtime overhead to within 1.8% of VSM, while cluster-aware retrieval accelerates queries by approximately 10.9%, making the system viable for latency-sensitive applications.

- **Usability & Flexibility:** The configurable components—autocomplete, custom query mode, stopword strategy, lemmatizer vs. stemmer, and dynamic hyperparameter search—allow practitioners to tailor the pipeline to diverse corpora and interactive scenarios.

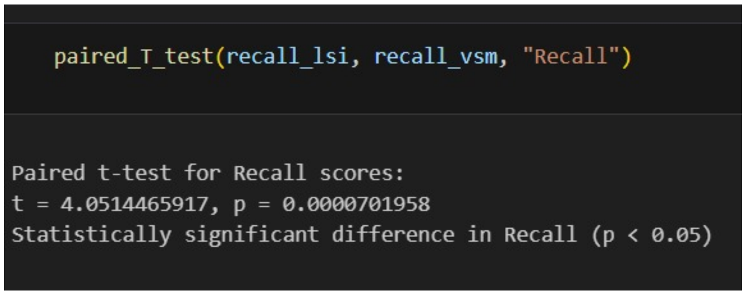
Looking forward, we plan to explore deeper semantic representations (e.g. word-embeddings or transformer-based indexing), richer relevance feedback loops, and domain-specific ontologies to further reduce vocabulary mismatch and polysemy errors. Overall, our results validate that carefully combining latent-semantic modeling, robust preprocessing, and efficient filtering yields both high retrieval quality and practical performance for large-scale IR tasks.

9 Archives



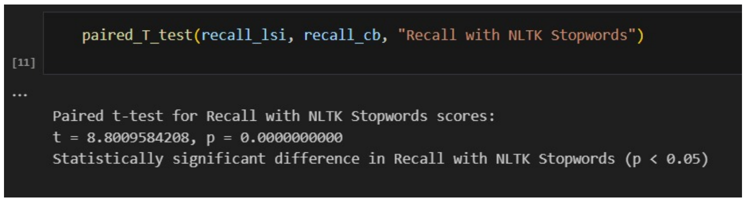
```
--------------------------------------------------------------------------------------------------------------
PS C:\NLP_SUBMISSIONS\CE23B048\CE23B048\Team4_codefolder> python Main_project_code\main.py -recompute True
SVD computed and saved.
Using the value of k for evaluation = 1
Precision, Recall and F-score @ 1 : 0.5688888888888889, 0.09432419518438165, 0.2646285074591881
MAP, nDCG @ 1 : 0.09432419518438165, 0.3715555555555553
--------------------------------------------------------------------------------------------------------------
Time taken by the IR system to evaluate the LSI model: 34.55774998664856 seconds
--------------------------------------------------------------------------------------------------------------
PS C:\NLP_SUBMISSIONS\CE23B048\CE23B048\Team4_codefolder> python Main_project_code\main.py
Using precomputed SVD.
Using the value of k for evaluation = 1
Precision, Recall and F-score @ 1 : 0.5688888888888889, 0.09432419518438165, 0.2646285074591881
MAP, nDCG @ 1 : 0.09432419518438165, 0.3715555555555553
--------------------------------------------------------------------------------------------------------------
Time taken by the IR system to evaluate the LSI model: 27.975228548049927 seconds
--------------------------------------------------------------------------------------------------------------
PS C:\NLP_SUBMISSIONS\CE23B048\CE23B048\Team4_codefolder> python Main_project_code\main.py -model clustering
Using the value of k for evaluation = 1
Precision, Recall and F-score @ 1 : 0.4533333333333333, 0.07518086671298302, 0.21052535319769583
MAP, nDCG @ 1 : 0.07518086671298302, 0.290962962962963
--------------------------------------------------------------------------------------------------------------
Time taken by the IR system to evaluate the clustering model: 24.495410919189453 seconds
--------------------------------------------------------------------------------------------------------------
PS C:\NLP_SUBMISSIONS\CE23B048\CE23B048\Team4_codefolder> python Main_project_code\main.py -model VSM
Using the value of k for evaluation = 1
Precision, Recall and F-score @ 1 : 0.5644444444444444, 0.10128782641994272, 0.2743011898534122
MAP, nDCG @ 1 : 0.10128782641994272, 0.39496296296296296
--------------------------------------------------------------------------------------------------------------
Time taken by the IR system to evaluate the VSM model: 27.495187759399414 seconds
--------------------------------------------------------------------------------------------------------------
```

Fig. 7: Time optimization



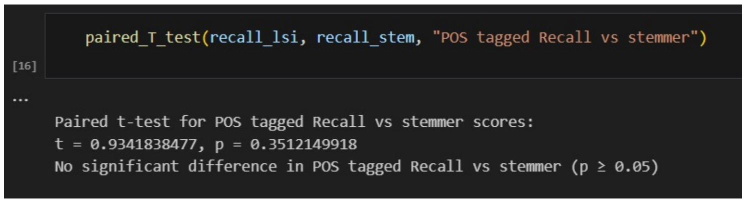
```
paired_T_test(recall_lsi, recall_vsm, "Recall")

Paired t-test for Recall scores:
t = 4.0514465917, p = 0.0000701958
Statistically significant difference in Recall (p < 0.05)
```

Fig. 8: LSA vs VSM



```
[11]
     paired_T_test(recall_lsi, recall_cb, "Recall with NLTK Stopwords")
...
     Paired t-test for Recall with NLTK Stopwords scores:
     t = 8.8009584208, p = 0.0000000000
     Statistically significant difference in Recall with NLTK Stopwords (p < 0.05)
```

Fig. 9: NLTK vs Corpus Based



```
[16]
     paired_T_test(recall_lsi, recall_stem, "POS tagged Recall vs stemmer")
...
     Paired t-test for POS tagged Recall vs stemmer scores:
     t = 0.9341838477, p = 0.3512149918
     No significant difference in POS tagged Recall vs stemmer (p ≥ 0.05)
```

Fig. 10: Lemmatizer vs Stemmer

Bibliography

[1] Balakrishnan, V., Lloyd-Yemoh, E.: Stemming and lemmatization: A comparison of retrieval performances. Information Retrieval Journal (2014)

[2] Church, K., Hanks, P.: Word association norms, mutual information, and lexicography. Computational Linguistics 16(1), 22–29 (1990)

[3] Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press, Cambridge, MA (1998)

[4] Golub, G., Reinsch, C.: Singular value decomposition and least squares solutions. Numerische Mathematik 14(5), 403–420 (1970)

[5] Hollmann, N., Eickhoff, C.: Ranking and feedback-based stopping for recall-centric document retrieval. In: CEUR Workshop Proceedings. s.n., S.l. (2017), working Notes of CLEF 2017 - Conference and Labs of the Evaluation Forum; Conference Location: Dublin, Ireland; Conference Date: September 11-14, 2017

[6] Järvelin, K., Kekäläinen, J.: Cumulated gain–based evaluation of ir techniques. ACM Transactions on Information Systems 20(4), 422–446 (2002)

[7] Kaur, J., Buttar, P.: A systematic review on stopword removal algorithms. International Journal on Future Revolution in Computer Science & Communication Engineering 4(4), 207–210 (2018)

[8] Kiss, T., Strunk, J.: Unsupervised multilingual sentence boundary detection. Computational Linguistics 32(4), 485–525 (2006)

[9] Knuth, D.: The Art of Computer Programming, Volume3: Sorting and Searching, The Art of Computer Programming, vol. 3. Addison–Wesley, Reading, MA, 1 edn. (1973)

[10] Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)

[11] MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. vol. 1, pp. 281–297. University of California Press (1967)

[12] Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)

[13] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, : Scikit-

learn: Machine learning in python. Journal of Machine Learning Research 12, 2825–2830 (2011)

[14] Pierrehumbert, J.: Probabilistic phonology: Discrimination and robustness. In: Bod, R., Hay, J., Jannedy, S. (eds.) Probability Theory in Linguistics, pp. 177–228. MIT Press, Cambridge,MA (2003)

[15] Publishing, S.: T test - statpearls - ncbi bookshelf (2022), `https://www.ncbi.nlm.nih.gov/books/NBK553048/`, accessed: 2025-05-02

[16] Salton, G., Buckley, C.: Improving retrieval performance by relevance feedback. In: Journal of the American Society for Information Science. vol. 41, pp. 288–297 (1990)

[17] Salton, G., McGill, M.: Introduction to Modern Information Retrieval. McGraw-Hill, NewYork (1983)

[18] Sculley, D.: Web-scale k-means clustering. In: Proceedings of the 19th International Conference on World Wide Web. pp. 1177–1178. ACM (2010)

[19] Willett, P.: The porter stemming algorithm: Then and now. Program 40(3), 219–223 (2006)