

Latent Semantic Indexing (LSI) in Information Retrieval Systems - Observations and Results

Tim Leute

April 29, 2025

Contents

1	Introduction	2
2	Introduction to Latent Semantic Indexing (LSI)	3
3	Implementation of LSI in the Hiking Trails IR System	4
3.1	Preprocessing	4
3.2	TF-IDF Vectorization	4
3.3	Applying Singular Value Decomposition (SVD)	4
3.4	LSI Search Class	5
3.5	Query Transformation and Search	5
3.6	Summary	6
4	Comparison: IR System with and without LSI	7
4.1	Example Query 1: <i>swimming</i>	7
4.2	Example Query 2: <i>children</i>	9
5	Analysis of Results	13
5.1	Analysis of Query 1: <i>swimming</i>	13
5.2	Analysis of Query 2: <i>children</i>	13
5.3	General Observations	14
6	Purpose and Broader Context	15

1 Introduction

The goal of this project is to enhance an existing information retrieval (IR) system, which uses hiking trail descriptions as its data source, by integrating Latent Semantic Indexing (LSI). The current system already supports ranked retrieval through the vector space model and boolean retrieval. The addition of LSI aims to improve semantic search capabilities and address vocabulary mismatch problems that commonly occur in keyword-based approaches.

2 Introduction to Latent Semantic Indexing (LSI)

Latent Semantic Indexing (LSI) is a mathematical technique used in information retrieval to uncover the underlying relationships between terms and documents. [2]. Unlike simple keyword matching, LSI reduces the dimensionality of the term-document space, revealing hidden semantic structures and capturing synonymy and related concepts. [1].

The process begins by constructing a term-document matrix, where each entry represents the frequency of a term in a document. This matrix is then decomposed using Singular Value Decomposition (SVD) into three matrices:

$$A = U\Sigma V^T$$

where A is the original term-document matrix, U and V are orthogonal matrices, and Σ is a diagonal matrix containing the singular values. [1].

To simplify the model and focus on the most significant patterns, only the top k singular values and corresponding vectors are retained. [2]. This rank- k approximation reduces noise and highlights important semantic relationships.

Documents and queries are then projected into this lower-dimensional semantic space, where similarities are computed, typically using cosine similarity. This allows the retrieval system to find documents that are semantically related to a query, even if they do not share exact terms. [2].

Through this dimensionality reduction and semantic mapping, LSI provides a more robust and intelligent method for document retrieval compared to traditional keyword-based approaches. [1].

3 Implementation of LSI in the Hiking Trails IR System

This section describes how Latent Semantic Indexing (LSI) was implemented within the existing Information Retrieval (IR) system that uses hiking trail descriptions as its data source. The following subsections highlight the important stages of the process.

3.1 Preprocessing

Before applying LSI, all documents undergo a basic preprocessing step:

- Conversion of text to lowercase
- Removal of non-word characters such as punctuation

The following Python function handles preprocessing: [3].

Listing 1: Text Preprocessing

```
import re

def preprocess(text):
    return re.sub(r'\W+', '-', text.lower())
```

3.2 TF-IDF Vectorization

The preprocessed documents are vectorized using TF-IDF (Term Frequency - Inverse Document Frequency), which captures the importance of terms within the document collection. [3].

Listing 2: TF-IDF Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [preprocess(doc) for doc in documents.values()]
vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(corpus)
```

3.3 Applying Singular Value Decomposition (SVD)

Once the TF-IDF matrix is built, dimensionality reduction is performed using Truncated Singular Value Decomposition (SVD). This projects the documents into a lower-dimensional semantic space. [3].

Listing 3: Applying SVD

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=75, random_state=42)
X_lsi = svd.fit_transform(X_tfidf)
```

3.4 LSI Search Class

The entire LSI preparation and search process is encapsulated in the `LSISearch` class, which allows easy integration into the system.

Listing 4: LSI Search Class

```
class LSISearch:
    def __init__(self, documents, n_components=75):
        self.documents = documents
        self.titles = list(documents.keys())
        self.n_components = n_components
        self._prepare_lsi()

    def _prepare_lsi(self):
        corpus = [preprocess(doc) for doc in self.documents.values()]
        self.vectorizer = TfidfVectorizer()
        X_tfidf = self.vectorizer.fit_transform(corpus)
        self.svd = TruncatedSVD(n_components=self.n_components, random_state=None)
        self.X_lsi = self.svd.fit_transform(X_tfidf)

    def search(self, query):
        query = preprocess(query)
        query_vec = self.vectorizer.transform([query])
        query_lsi = self.svd.transform(query_vec)
        similarities = np.dot(self.X_lsi, query_lsi.T).flatten()
        results = [(self.titles[idx], similarities[idx])
                    for idx in range(len(similarities)) if similarities[idx] > 0]
        results = sorted(results, key=lambda x: x[1], reverse=True)
        return results
```

Note on `n_components`: The parameter `n_components` defines the number of dimensions retained in the reduced semantic space after applying Singular Value Decomposition (SVD). It determines how many latent topics or concepts are captured from the original TF-IDF matrix. A higher value preserves more detail but may also retain noise, while a lower value emphasizes the strongest semantic relationships but may oversimplify. In this project, the number of components will be set to different values during the experiment queries to show the difficulty of finding a balanced trade-off between performance and semantic generalization. The effects of this parameter will be further discussed in the query-specific examples. [3].

3.5 Query Transformation and Search

When a user submits a search query, it undergoes the same preprocessing and vectorization steps before being projected into the reduced semantic space. Documents are then ranked based on their cosine similarity to the query vector.

- Preprocess query text
- Vectorize query with TF-IDF vectorizer

- Transform query into LSI space using SVD
- Compute cosine similarities with documents
- Return ranked document list

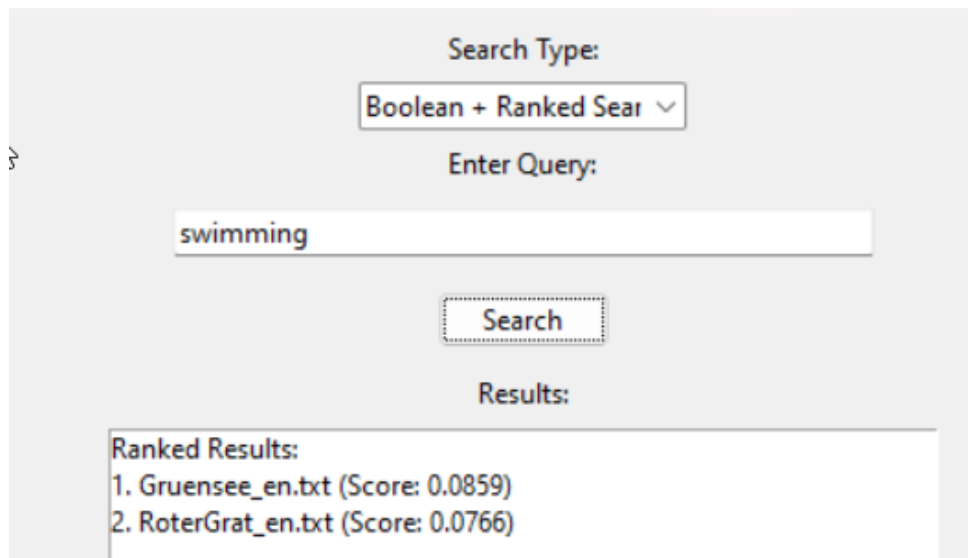
3.6 Summary

By implementing LSI as described, the hiking trails IR system is now able to retrieve documents based on their latent semantic relationships rather than relying solely on exact keyword matches. This enhances the system's ability to recognize conceptually related hiking trails even when different wording is used.

4 Comparison: IR System with and without LSI

In this section, we present the retrieval results for two different queries. For each query, the results without LSI (Boolean/Ranked Retrieval) and with LSI at different levels of dimensionality reduction are shown. A detailed analysis and interpretation of the differences will follow in the next chapter.

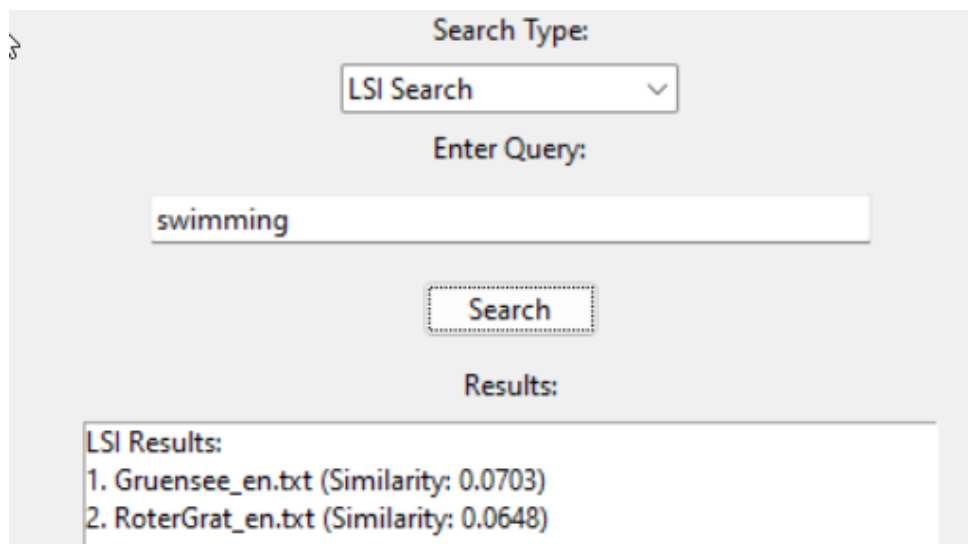
4.1 Example Query 1: *swimming*



The screenshot shows a web-based search interface. At the top, there is a 'Search Type:' dropdown menu set to 'Boolean + Ranked Sear'. Below it is an 'Enter Query:' text box containing the word 'swimming'. A 'Search' button is positioned below the text box. Underneath the button, the 'Results:' section displays 'Ranked Results:' followed by a list of two items: '1. Gruensee_en.txt (Score: 0.0859)' and '2. RoterGrat_en.txt (Score: 0.0766)'.

Rank	Document	Score
1.	Gruensee_en.txt	0.0859
2.	RoterGrat_en.txt	0.0766

Figure 1: Retrieval results without LSI for Query 1



The screenshot shows the same search interface as Figure 1, but with the 'Search Type:' dropdown menu set to 'LSI Search'. The 'Enter Query:' text box still contains 'swimming', and the 'Search' button is present. The 'Results:' section displays 'LSI Results:' followed by a list of two items: '1. Gruensee_en.txt (Similarity: 0.0703)' and '2. RoterGrat_en.txt (Similarity: 0.0648)'.

Rank	Document	Similarity
1.	Gruensee_en.txt	0.0703
2.	RoterGrat_en.txt	0.0648

Figure 2: Retrieval results with LSI (N=100) for Query 1

LSI Search

Enter Query:

swimming

Results:

LSI Results:

1. Gruensee_en.txt (Similarity: 0.0703)
2. RoterGrat_en.txt (Similarity: 0.0648)
3. Mottakopf_en.txt (Similarity: 0.0003)

Figure 3: Retrieval results with LSI (N=84) for Query 1

Search Type:

LSI Search

Enter Query:

swimming

Results:

LSI Results:

1. RoterGrat_en.txt (Similarity: 0.0415)
2. Gruensee_en.txt (Similarity: 0.0331)
3. Grindjisee_en.txt (Similarity: 0.0302)
4. Rinnenspitze_en.txt (Similarity: 0.0152)
5. HoheTauern_en.txt (Similarity: 0.0102)
6. EdelweissTrail_en.txt (Similarity: 0.0085)
7. Lodron_en.txt (Similarity: 0.0073)
8. Puehringerhuette_en.txt (Similarity: 0.0054)
9. Traunsee_en.txt (Similarity: 0.0054)
10. Kaunergrat_en.txt (Similarity: 0.0052)
11. Pizol_en.txt (Similarity: 0.0051)
12. Oeschinensee_en.txt (Similarity: 0.0045)
13. Braunarlspitze_en.txt (Similarity: 0.0042)
14. Madrisella_en.txt (Similarity: 0.0038)
15. Saoseo_en.txt (Similarity: 0.0036)
16. Gratlspitze_en.txt (Similarity: 0.0036)
17. RubihornTrail_en.txt (Similarity: 0.0035)

Figure 4: Retrieval results with LSI (N=50) for Query 1

4.2 Example Query 2: *children*

Search Type:
Boolean + Ranked Sear ▾

Enter Query:
children

Search

Results:

Ranked Results:

1. Gratspitze_en.txt (Score: 0.0732)
2. Grossarl_en.txt (Score: 0.0623)
3. Traunsee_en.txt (Score: 0.0605)
4. Braunarlspitze_en.txt (Score: 0.0497)
5. Madrisella_en.txt (Score: 0.0469)
6. Olpererhuetten_en.txt (Score: 0.0432)
7. Nigardsbreen_en.txt (Score: 0.0374)
8. Saentis_en.txt (Score: 0.0339)
9. CreuxDuVan_en.txt (Score: 0.0252)

Figure 5: Retrieval results without LSI for Query 2

Search Type:
LSI Search

Enter Query:
children

Search

Results:

LSI Results:

1. Gratlspitze_en.txt (Similarity: 0.0699)
2. Grossarl_en.txt (Similarity: 0.0581)
3. Traunsee_en.txt (Similarity: 0.0540)
4. Braunarlspitze_en.txt (Similarity: 0.0441)
5. Madrisella_en.txt (Similarity: 0.0407)
6. Olpererhuetten_en.txt (Similarity: 0.0361)
7. Nigardsbreen_en.txt (Similarity: 0.0346)
8. Saentis_en.txt (Similarity: 0.0293)
9. CreuxDuVan_en.txt (Similarity: 0.0242)

Figure 6: Retrieval results with LSI (N=100) for Query 2

Search Type:
LSI Search

Enter Query:
children

Search

Results:

LSI Results:

1. Gratlspitze_en.txt (Similarity: 0.0700)
2. Grossarl_en.txt (Similarity: 0.0581)
3. Traunsee_en.txt (Similarity: 0.0540)
4. Braunarls Spitze_en.txt (Similarity: 0.0441)
5. Madrisella_en.txt (Similarity: 0.0407)
6. Olpererhuetten_en.txt (Similarity: 0.0361)
7. Nigardsbreen_en.txt (Similarity: 0.0346)
8. Saentis_en.txt (Similarity: 0.0293)
9. CreuxDuVan_en.txt (Similarity: 0.0242)
10. Grindjisee_en.txt (Similarity: 0.0007)
11. ZugspitzeTrail2_en.txt (Similarity: 0.0005)
12. Gatterl_en.txt (Similarity: 0.0002)
13. Luenersee_en.txt (Similarity: 0.0002)

Figure 7: Retrieval results with LSI (N=82) for Query 2

Search Type:

LSI Search

Enter Query:

children

Search

Results:

20. Avdallsfossen_en.txt (Similarity: 0.0066)
21. Itonskopf_en.txt (Similarity: 0.0063)
22. Sonntagshorn_en.txt (Similarity: 0.0063)
23. Geiranger_en.txt (Similarity: 0.0061)
24. PazolaStock_en.txt (Similarity: 0.0058)
25. Rinnenspitze_en.txt (Similarity: 0.0057)
26. Gruensee_en.txt (Similarity: 0.0056)
27. HoheTauern_en.txt (Similarity: 0.0055)
28. RoterGrat_en.txt (Similarity: 0.0053)
29. Pfannspitze_en.txt (Similarity: 0.0053)
30. Upsspitze_en.txt (Similarity: 0.0053)
31. FueelaSchwarzhorn_en.txt (Similarity: 0.0052)
32. Alesund_en.txt (Similarity: 0.0052)
33. Grindjisee_en.txt (Similarity: 0.0049)
34. Pizol_en.txt (Similarity: 0.0048)
35. Griessenkareck_en.txt (Similarity: 0.0045)

Figure 8: Retrieval results with LSI (N=50) for Query 2

5 Analysis of Results

In this chapter, we analyze and interpret the retrieval results presented in the previous section, focusing on the influence of Latent Semantic Indexing (LSI) on the relevance and quality of the search outcomes.

5.1 Analysis of Query 1: *swimming*

When examining the results for the query "swimming," several patterns emerge:

- **Without LSI (Boolean/Ranked Retrieval):** Only documents explicitly mentioning "swimming" are retrieved. Synonyms or related concepts like "lake access" or "water activities" are not captured.
- **LSI with high N (e.g., $N=100$):** The results are very similar to the Boolean and ranked retrieval outputs. The system remains close to keyword matches, retrieving mainly documents that directly discuss swimming.
- **LSI with medium N (e.g., $N=84$):** A more balanced retrieval is achieved. Relevant documents that imply swimming-related activities (without explicitly using the word) are retrieved, improving semantic coverage while maintaining precision.
- **LSI with low N (e.g., $N=50$):** The results become too broad. Many documents are retrieved that are only weakly connected to the swimming concept, such as general references to nature or lakes without direct relevance.

Conclusion: At high dimensions, LSI behaves similarly to keyword-based retrieval; at medium dimensions, it enhances semantic matching without significant noise; at low dimensions, precision drops as unrelated results appear.

5.2 Analysis of Query 2: *children*

For the query "children," a comparable trend is observed:

- **Without LSI (Boolean/Ranked Retrieval):** Documents mentioning "children" or "kids" directly are retrieved. Broader concepts like "family-friendly" or "easy trail" are overlooked.
- **LSI with high N (e.g., $N=100$):** The results are still very close to the Boolean approach, focusing mainly on direct mentions of children.
- **LSI with medium N (e.g., $N=82$):** A well-balanced retrieval emerges, capturing not only direct matches but also trails suitable for families, based on descriptions like "easy," "short," or "safe," enhancing the semantic understanding of the query.
- **LSI with low N (e.g., $N=50$):** The search becomes less precise, with many retrieved documents only vaguely related to the idea of children or even completely unrelated topics like general tourism.

Conclusion: High N values lead to retrievals close to pure keyword search, medium N values offer a good balance between semantic flexibility and accuracy, and low N values result in loss of focus and retrieval of irrelevant documents.

5.3 General Observations

- **High N (e.g., 100):** Results are very similar to traditional keyword matching. Only direct matches are retrieved, limiting semantic improvements.
- **Medium N (e.g., 82-84):** A balanced trade-off between recall and precision is achieved. Related concepts are effectively captured without introducing much noise.
- **Low N (e.g., 50):** Retrieval becomes too generalized, and many documents are included that are only marginally or not at all relevant to the query.

Final Note: Latent Semantic Indexing enhances the semantic capabilities of the hiking trails IR system. However, careful tuning of the number of dimensions N is critical. Choosing an appropriate N allows the system to recognize conceptually relevant documents while avoiding a flood of irrelevant results, leading to a significantly improved search experience compared to traditional retrieval methods.

6 Purpose and Broader Context

This project demonstrates how a semantic Information Retrieval (IR) system can serve as the foundation for a larger, user-centered application in the outdoor and tourism sector. By enabling natural-language search through hiking trail descriptions, the system opens the door to a smart trail discovery platform where users can find routes based on concepts like "easy for kids," "has waterfalls," or "good for swimming."

Such a system could be developed into a mobile app that helps hikers plan their trips more intuitively, offering personalized suggestions based on trail features, user preferences, or accessibility needs. Beyond hiking, the same concept could be extended to other domains like biking routes or nature tourism, where semantic understanding greatly improves user experience.

References

- [1] David A. Grossman and Ophir Frieder. *Information Retrieval: Algorithms and Heuristics*. Springer, 2nd edition, 2004.
- [2] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [3] Tim Leute. *Source Code*. GitHub-Repository, 2025. <https://github.com/Tim10022023/InformationRetrievalSystem>