LSI vector model

Andrii Kozlov

ČVUT–FIT

kozloan3@fit.cvut.cz

20.03.2025

1 Introduction

The goal of this project was to develop a web-based document search application using **Latent Semantic Indexing (LSI).** The application allows users to perform efficient semantic search over a collection of documents stored in a MongoDB database.

- **Inputs:** A collection of text documents containing various topics.
- **Outputs:** A ranked list of documents relevant to the user's query, based on **semantic similarity**.

2 Method of Solution

Preprocessing:

- Lowercasing
- Removing stopwords, special characters
- Tokenization (splitting text into words)
- Lemmatization (ex. go, went, gone -> go).

TF-IDF Vectorization:

- Convert words into numerical values using tf–idf (Term Frequency-Inverse Document Frequency).
- This gives each word a **weight** based on how important it is in the document relative to the whole dataset.

Weight is calculated as:

$$\textbf{TF-IDF}(t,d) = \textbf{TF}(t,d) \times \textbf{IDF}(t)$$

- TF (Term Frequency) = how often term **t** appears in document **d**
- IDF (Inverse Document Frequency) = how unique the term is across all documents

Dimensionality Reduction (SVD):

- Application of **Singular Value Decomposition (SVD)** to extract meaningful concepts from documents.

Query Processing:

- The user's query undergoes the same **preprocessing and transformation** as the documents.

Cosine Similarity Calculation:

- Computing the **cosine similarity** between the transformed query vector and document vectors.

Ranking & Retrieval:

- Documents are ranked and displayed based on similarity scores.

3 Implementation

Programming Languages

- **Backend:** Python (Flask for API development)
- **Frontend:** React.js, html, css
- **Database:** MongoDB

Libraries Used

- **Backend:** nltk, pandas, numpy, scikit-learn, pickle
- **Frontend:** react-scripts
- **Database:** pymongo

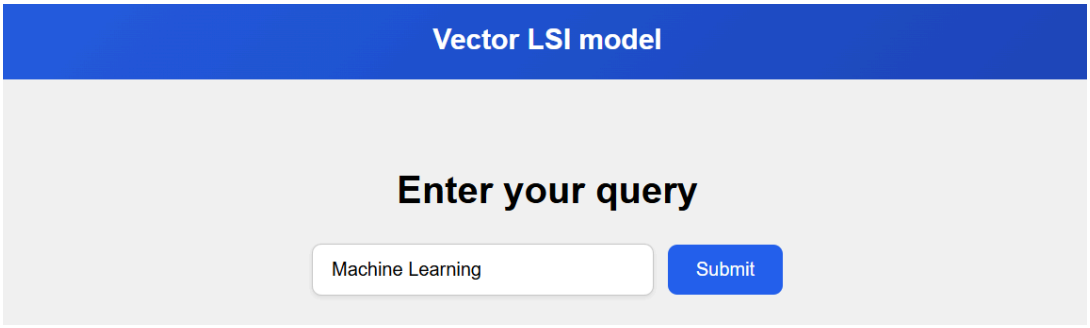
Application Architecture

- **Backend API** (Flask): Handles data retrieval and LSI computation.
- **Frontend** (React.js): Provides an interface for users to input queries and view results.
- **Database** (MongoDB): Stores documents and precomputed LSI components.

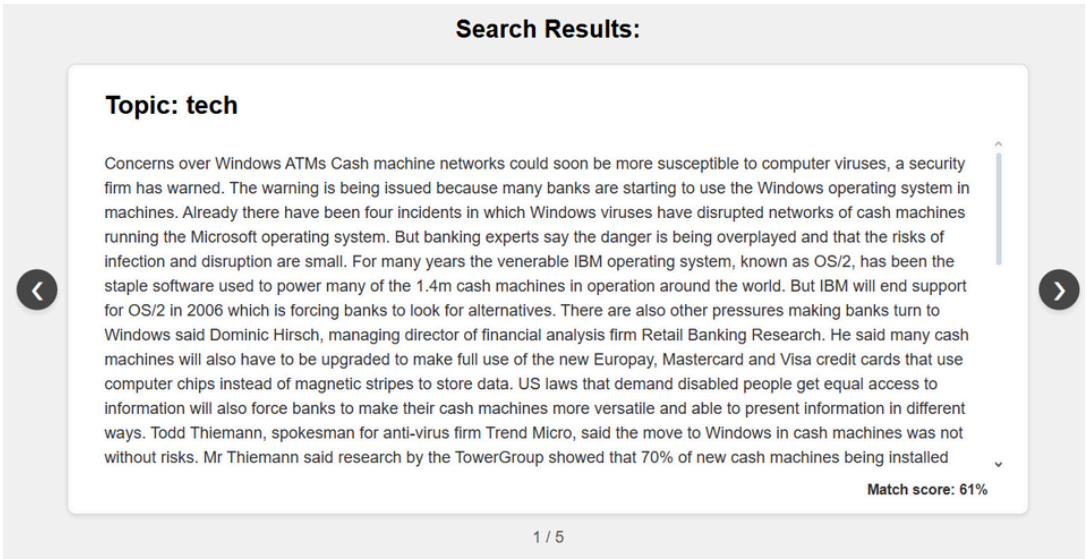
Requirements to Run the Application

- Python libraries installed via *pip install -r requirements.txt*
- Python 3.10+
- Node.js & npm

4 Examples of Input and Output



Example Query:
"Machine Learning"



Here we can see search results presented as a carousel sliders. Documents are displayed based on similarityscores.

5 Discussion

- **Scalability Issues:** As the dataset grows, **SVD computation** becomes more expensive.
- **Lack of Real-Time Learning:** The system does not update **semantic structures** dynamically.
- **Handling of Multi-word Expressions:** Some complex terms might not be accurately captured.

6 Conclusion

The project successfully implemented a **Latent Semantic Indexing (LSI)-based** document search engine, enabling semantic search over a collection of documents. By leveraging **TF-IDF, Singular Value Decomposition (SVD), and cosine similarity**, the system retrieves relevant documents based on conceptual meaning rather than exact keyword matching.

Overall, the project successfully applied **theoretical knowledge of information retrieval** to a practical problem, demonstrating the potential of **semantic search** in real-world applications.

7 References

- **Python | Lemmatization with NLTK - GeeksforGeeks**
- **Removing stop words with NLTK in Python - GeeksforGeeks**
- **TfidfVectorizer**
- **Singular value decomposition**
- Cosine similarity
- W3Schools.com