

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).

Dimensionality Reduction (SVD):

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

TF-IDF Vectorization:

- Convert words into numerical values using \oplus 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:

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

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

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

Vector LSI model

Enter your query

Example Query:
"Machine Learning"

Search Results:

Topic: tech

Concerns over Windows ATMs Cash machine networks could soon be more susceptible to computer viruses, a security firm has warned. The warning is being issued because many banks are starting to use the Windows operating system in machines. Already there have been four incidents in which Windows viruses have disrupted networks of cash machines running the Microsoft operating system. But banking experts say the danger is being overplayed and that the risks of infection and disruption are small. For many years the venerable IBM operating system, known as OS/2, has been the staple software used to power many of the 1.4m cash machines in operation around the world. But IBM will end support for OS/2 in 2006 which is forcing banks to look for alternatives. There are also other pressures making banks turn to Windows said Dominic Hirsch, managing director of financial analysis firm Retail Banking Research. He said many cash machines will also have to be upgraded to make full use of the new Europay, Mastercard and Visa credit cards that use computer chips instead of magnetic stripes to store data. US laws that demand disabled people get equal access to information will also force banks to make their cash machines more versatile and able to present information in different ways. Todd Thiemann, spokesman for anti-virus firm Trend Micro, said the move to Windows in cash machines was not without risks. Mr Thiemann said research by the TowerGroup showed that 70% of new cash machines being installed

Match score: 61%

1 / 5

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

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](#)