BASE SYSTEM USING LATENT SEMANTIC INDEXING (LSI)

This project focuses on building a document base system that uses semantic indexing techniques like Latent Semantic Indexing (LSI) and Singular Value Decomposition (SVD) to improve document search and retrieval. The system allows users to upload documents, store them in a relational database, and perform similarity queries using SQL or Java-based functions.

HYPOTHESIS

By applying LSI techniques, we can improve document retrieval accuracy compared to traditional keyword-based systems.

We believe that using semantic indexing will allow us to find relevant documents even when they don't share exact keywords.

OBJECTIVES

- Build a system that stores and manages at least 10 documents.
- Use LSI/SVD to represent documents semantically.
- Allow users to submit queries and retrieve relevant documents using similarity/distance metrics.

GOALS

- Implement a relational database schema to store document-term frequency matrices.
- Apply lemmatization to improve semantic understanding.
- Compare results from cosine similarity and Euclidean distance.
- Develop a basic GUI interface for uploading and querying documents.

INTRODUCTION

The project focuses on building a document base system that uses Latent Semantic Indexing (LSI) to represent and query documents based on their semantic content. The collection used in this project consists of 10 text documents related to various aspects of technology, selected to reflect its broad impact across multiple fields. These include topics such as technology and society, technology in sports, technology and social interaction, technology in healthcare, technology and the environment, economic impact of technology, education and technology, public safety, transportation, and democracy. This diverse selection ensures that the system is tested on real-world scenarios where semantic understanding is crucial for accurate retrieval. By applying LSI techniques and SVD (Singular Value Decomposition), the system reduces dimensionality and captures underlying relationships between terms, allowing it to find relevant documents even when they don't share exact keywords. The system also includes a relational database schema

and supports similarity and dissimilarity queries using SQL, enabling users to compare documents or search for information using natural language queries. This approach not only enhances search accuracy but also demonstrates how engineering design can be applied to solve complex information retrieval problems while considering public health, safety, environmental impact, and global accessibility.

METHODOLOGY

The development of this project followed a structured approach that aligns with standard software engineering practices. The methodology was based on the ISO/IEC 12207:2008 international standard for software life cycle processes. This standard provides a comprehensive framework for managing and executing software development, ensuring quality, maintainability, and traceability throughout all stages. The project was developed in four main phases:

Requirements Gathering

The first step involved identifying the functional and non-functional requirements of the system. The goal was to build a document base system capable of storing and retrieving at least 10 technology-related documents using semantic indexing techniques such as Latent Semantic Indexing (LSI) and Singular Value Decomposition (SVD). Requirements included text preprocessing (lemmatization), database design, similarity/dissimilarity queries using SQL, and an interactive user interface.

Planning

The planned workflow for the application is as follows: users will be able to upload documents, which will then be processed by a function that applies a stemmer to lemmatize the content. This function returns the lemmatized version of the document, which is stored in a database along with the frequency count of each word. This enables the construction of a document-term matrix that maps each document to the number of occurrences of its words. Users will then be able to submit queries, which are processed using the same stemming function to remove stop words and lemmatize the terms, producing a query vector. This vector is treated as a new document and compared against the database using Euclidean distance between word occurrence vectors. The system will return a ranked list of the most relevant documents based on similarity, excluding any documents with zero relevance (i.e., no shared terms).

System Design

A relational database schema was designed to store the documents and their term frequencies. Tables were defined to represent documents (TEXT), terms (TERM), and their relationships (HAS). Additional tables were created to support query processing (QUERY, QUERY_TERM). An optional view (COMPLETE) was also implemented to fill in missing values and ensure full matrix representation for similarity calculations.

Implementation

The core logic was implemented in Java, including text cleaning, lemmatization, frequency counting, and database interaction. A simple graphical user interface was built using Swing and AWT to allow users to upload documents and perform queries. The backend used MySQL to manage data persistence and execute SQL-based similarity functions like cosine similarity and Euclidean distance.

Testing

Multiple test cases were executed to validate the correctness and performance of the system. Sample documents were uploaded and queried to verify that:

- Text was correctly preprocessed
- Term frequencies were accurately stored
- Similarity and dissimilarity metrics returned relevant results
- The database schema supported efficient querying and scalability

This systematic approach ensured that the project was well-organized, met its objectives, and could be extended or maintained in the future.

TECHNICAL SPECIFICATIONS

The system is built around the concept of semantic document retrieval, where documents are represented as vectors of term frequencies. These vectors are then compared using mathematical similarity measures to determine relevance. The key technical components include:

Text Processing

Documents are cleaned by removing special characters and stopwords. Words are lemmatized using the Snowball stemmer, reducing them to their root form for more accurate matching. We use the first diagram as a reference to develop our own based on our SQL tables in the database.

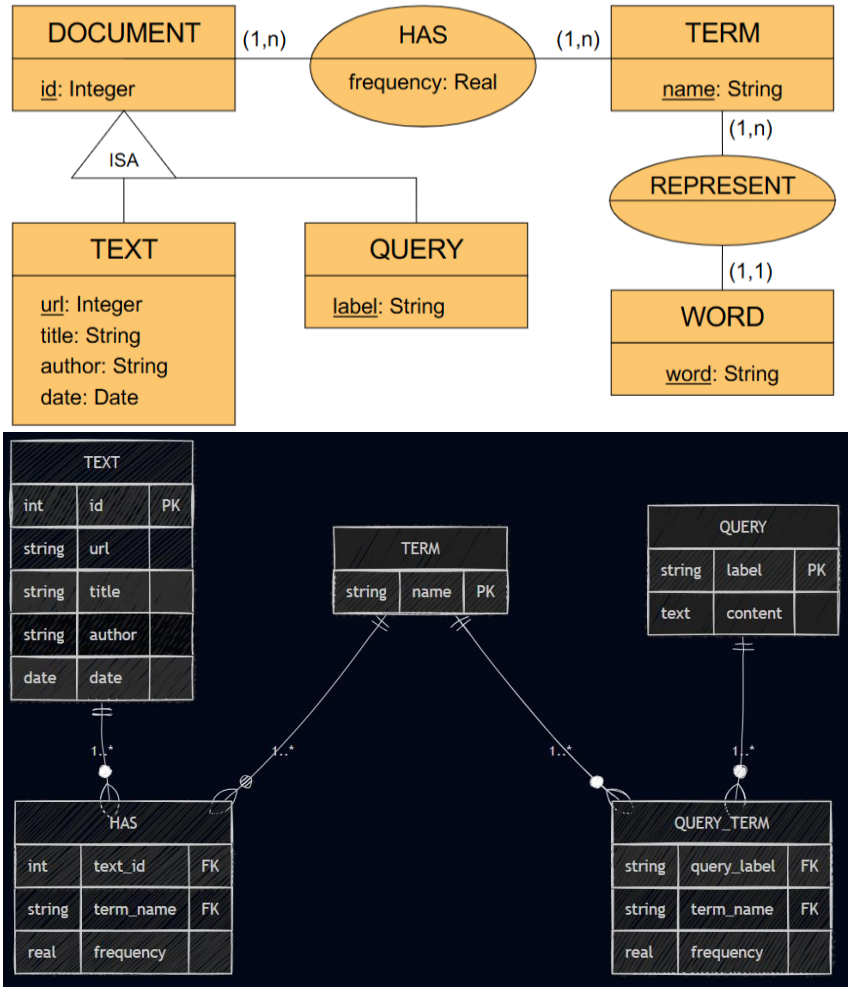


Image 1. Database Schema

The relational schema consists of five main tables:

TEXT: Stores metadata about each document.

TERM: Holds all unique terms found across the documents.

HAS: Represents the term-document frequency matrix.

QUERY: Stores user-defined search queries.

QUERY_TERM: Stores term frequencies for each query.

Pipeline of the information

The implemented system follows a structured processing flow that begins with loading documents in .txt format. Each document undergoes processing through a function that cleans the text, removes stopwords, and performs lemmatization using the Snowball Stemmer library, effectively reducing words to their root forms (lemmas). This process

generates a frequency vector that is stored in the database as a document-term matrix, using the TEXT, TERM, and HAS tables. When the user inputs a query, it undergoes the same preprocessing process and is compared with stored documents using Euclidean distance, resulting in the most relevant documents sorted by semantic proximity. Documents that have no words in common with the query (relevance equals zero) are excluded from the results, ensuring efficient and precise searches.

Similarity Functions

Two types of queries were implemented:

- Similarity Queries: Used cosine similarity and Jaccard index to compare documents or queries.

```java
private double computeCosine(Map<String, Integer> query, Map<String, Integer> doc) {
    double dotProduct = 0.0;
    double queryNorm = 0.0;
    double docNorm = 0.0;

    // Dot product
    for (String word : query.keySet()) {
        int queryCount = query.get(word);
        int docCount = doc.getOrDefault(word, 0);

        dotProduct += queryCount * docCount;
    }

    // Normas
    for (int count : query.values()) {
        queryNorm += count * count;
    }
    for (int count : doc.values()) {
        docNorm += count * count;
    }

    if (queryNorm == 0 || docNorm == 0) return 0.0;

    return dotProduct / (Math.sqrt(queryNorm) * Math.sqrt(docNorm));
}
```

```sql
-- Similarity Queries
-- Cosine Similarity Between Two Documents
SELECT d.text_id,
       SUM(d.frequency * q.frequency) / (
           SQRT(SUM(d.frequency * d.frequency)) *
           SQRT(SUM(q.frequency * q.frequency))
       ) AS cosine_similarity
FROM COMPLETE d
JOIN COMPLETE q ON d.term_name = q.term_name AND q.text_id = 1
GROUP BY d.text_id
```

```
ORDER BY cosine_similarity DESC;
```

- Dissimilarity Queries: Used Euclidean distance to measure how far apart two documents are.

```
private double computeEuclidean(Map<String, Integer> query, Map<String, Integer> doc) {
    double sum = 0.0;

    // Unir todas las palabras de query y doc
    Set<String> allWords = new HashSet<>();
    allWords.addAll(query.keySet());
    allWords.addAll(doc.keySet());

    for (String word : allWords) {
        int queryCount = query.getOrDefault(word, 0);
        int docCount = doc.getOrDefault(word, 0);

        sum += Math.pow(queryCount - docCount, 2);
    }

    return Math.sqrt(sum);
}
```

```
-- Dissimilarity Queries
-- Euclidean Distance Between Query and All Documents

SELECT td.text_id,
       SQRT(SUM(POWER(td.frequency - tq.frequency, 2))) AS
euclidean_distance
FROM COMPLETE tq
JOIN COMPLETE td ON tq.term_name = td.term_name
WHERE tq.text_id = 1
GROUP BY td.text_id
ORDER BY euclidean_distance ASC;
```

The database was designed to support efficient storage and retrieval of document-term relationships. Each document is broken down into individual words (after preprocessing) and stored with its corresponding frequency. This allows the system to build a term-document frequency matrix, which serves as the basis for LSI/SVD operations.

The following tables make up the core structure:

```
-- Table: TEXT
CREATE TABLE TEXT (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    url VARCHAR(255) NOT NULL,
    title VARCHAR(255)
```

```
);

-- Table: TERM
CREATE TABLE TERM (
    name VARCHAR(100) PRIMARY KEY
);

-- Table: HAS
CREATE TABLE HAS (
    text_id INTEGER,
    term_name VARCHAR(100),
    frequency REAL NOT NULL,
    PRIMARY KEY (text_id, term_name),
    FOREIGN KEY (text_id) REFERENCES TEXT(id) ON DELETE CASCADE,
    FOREIGN KEY (term_name) REFERENCES TERM(name) ON DELETE CASCADE
);

-- Table: QUERY
CREATE TABLE QUERY (
    label VARCHAR(100) PRIMARY KEY,
    content TEXT NOT NULL
);

-- Table: QUERY_TERM
CREATE TABLE QUERY_TERM (
    query_label VARCHAR(100),
    term_name VARCHAR(100),
    frequency REAL NOT NULL,
    PRIMARY KEY (query_label, term_name),
    FOREIGN KEY (query_label) REFERENCES QUERY(label) ON DELETE CASCADE,
    FOREIGN KEY (term_name) REFERENCES TERM(name) ON DELETE CASCADE
);
```

The HAS table is central to the system, capturing how often each term appears in each document. Foreign keys ensure referential integrity between tables. Additionally, the COMPLETE view was created to handle comparisons where some terms may be absent in certain documents, ensuring consistent vector representations for similarity calculations.

This design supports both basic keyword-based queries and advanced semantic searches using LSI techniques.

The platform used includes:

- Java for text processing and GUI development
- MySQL for persistent data storage

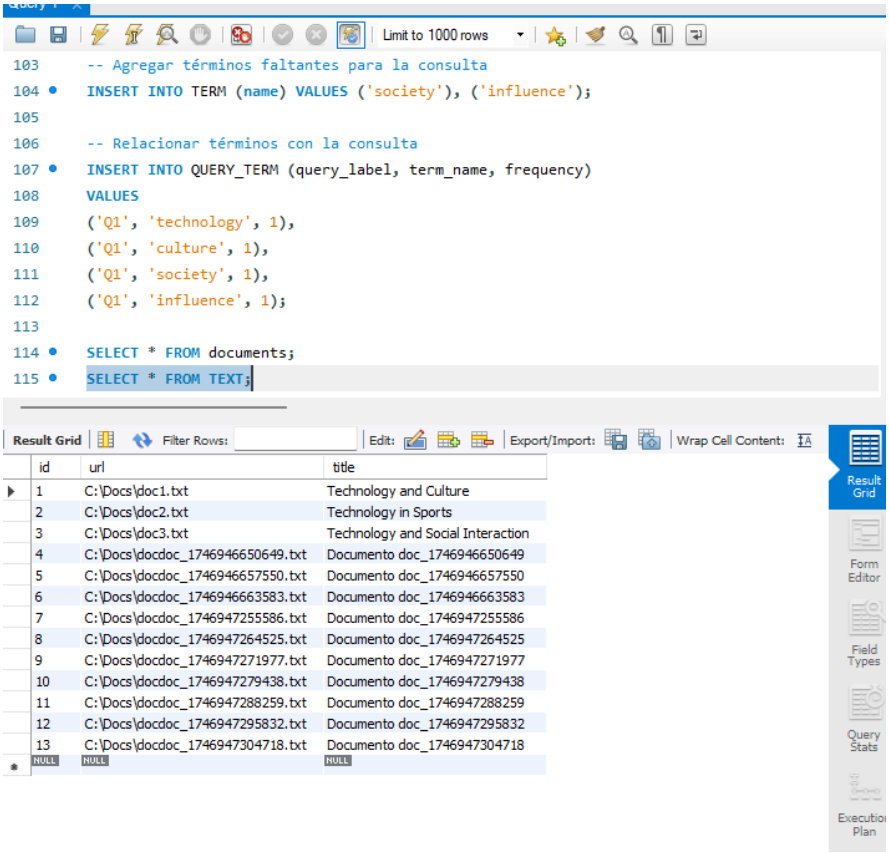
- Snowball Stemmer for lemmatization
- Swing for building the graphical user interface

This combination allowed for a modular and extensible system that can be adapted for larger-scale applications.

IMPLEMENTATION

The system performs the following key functions:

Document Upload: Users can input or upload .txt files, which are then processed and stored in the database.



```
103    -- Agregar términos faltantes para la consulta
104    INSERT INTO TERM (name) VALUES ('society'), ('influence');
105
106    -- Relacionar términos con la consulta
107    INSERT INTO QUERY_TERM (query_label, term_name, frequency)
108    VALUES
109    ('Q1', 'technology', 1),
110    ('Q1', 'culture', 1),
111    ('Q1', 'society', 1),
112    ('Q1', 'influence', 1);
113
114    SELECT * FROM documents;
115    SELECT * FROM TEXT;
```

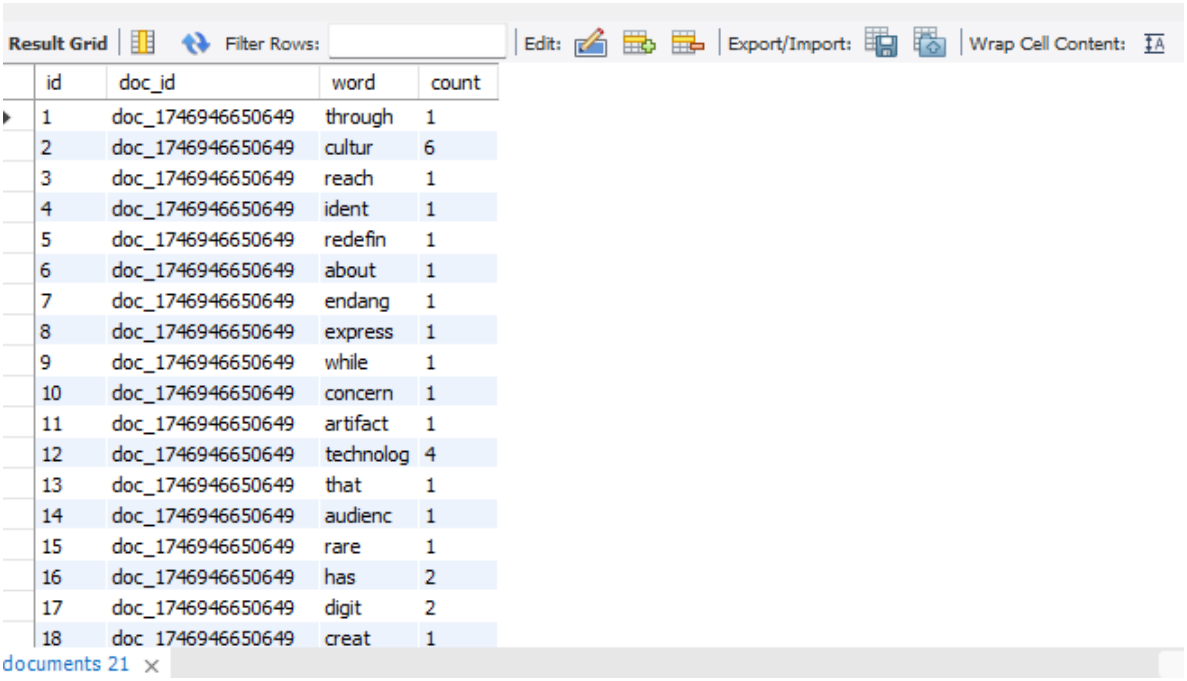
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	url	title
1	C:\Docs\doc1.txt	Technology and Culture
2	C:\Docs\doc2.txt	Technology in Sports
3	C:\Docs\doc3.txt	Technology and Social Interaction
4	C:\Docs\docdoc_1746946650649.txt	Documento doc_1746946650649
5	C:\Docs\docdoc_1746946657550.txt	Documento doc_1746946657550
6	C:\Docs\docdoc_1746946663583.txt	Documento doc_1746946663583
7	C:\Docs\docdoc_1746947255586.txt	Documento doc_1746947255586
8	C:\Docs\docdoc_1746947264525.txt	Documento doc_1746947264525
9	C:\Docs\docdoc_1746947271977.txt	Documento doc_1746947271977
10	C:\Docs\docdoc_1746947279438.txt	Documento doc_1746947279438
11	C:\Docs\docdoc_1746947288259.txt	Documento doc_1746947288259
12	C:\Docs\docdoc_1746947295832.txt	Documento doc_1746947295832
13	C:\Docs\docdoc_1746947304718.txt	Documento doc_1746947304718
NULL	NULL	NULL

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

Image 2. Document stored in database.

Text Preprocessing: The system cleans the text, removes stopwords, and applies lemmatization using Snowball stemmer to improve semantic matching. Using the SVD (Single Value Decomposition).



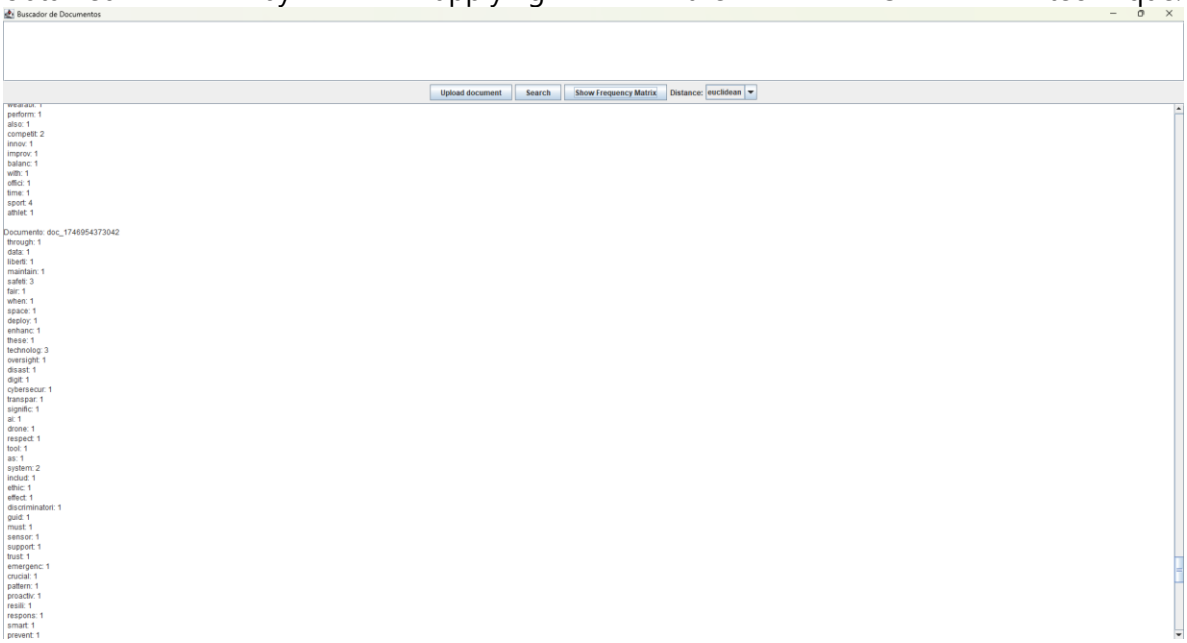
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	doc_id	word	count
1	doc_1746946650649	through	1
2	doc_1746946650649	cultur	6
3	doc_1746946650649	reach	1
4	doc_1746946650649	ident	1
5	doc_1746946650649	redefin	1
6	doc_1746946650649	about	1
7	doc_1746946650649	endang	1
8	doc_1746946650649	express	1
9	doc_1746946650649	while	1
10	doc_1746946650649	concern	1
11	doc_1746946650649	artifact	1
12	doc_1746946650649	technolog	4
13	doc_1746946650649	that	1
14	doc_1746946650649	audienc	1
15	doc_1746946650649	rare	1
16	doc_1746946650649	has	2
17	doc_1746946650649	digit	2
18	doc_1746946650649	creat	1

documents 21

Image 3. Indexing terms with Snowball stemmer to create lemmatization

Frequency Matrix Generation: Term frequencies are calculated and stored in the HAS table. It's created to make comparisons of efficiency and effectiveness with the representation obtained by applying the LSI technique.



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

doc_id	title	term_name	frequency
1	Technology and Culture	culture	9
1	Technology and Culture	diversity	2
1	Technology and Culture	expression	3
1	Technology and Culture	heritage	2
1	Technology and Culture	identity	2
1	Technology and Culture	media	3
1	Technology and Culture	platform	4
1	Technology and Culture	preservation	2
1	Technology and Culture	social	4
1	Technology and Culture	technology	10
1	Technology and Culture	virtual	2
2	Technology in Sports	biometrics	2
2	Technology in Sports	broadcast	2
2	Technology in Sports	e-sports	2
2	Technology in Sports	fairness	2
2	Technology in Sports	performance	4
2	Technology in Sports	privacy	2
2	Technology in Sports	sports	9

Images 4, 5. Frequency Matrix in table.

Query Support: Users can enter natural language queries, which are processed the same way as documents.

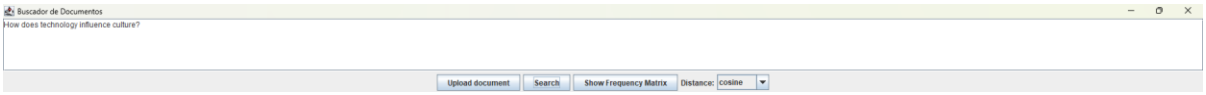


Image 6. Interface query example

Similarity/Dissimilarity Search: The system allows to calculate cosine similarity, dice coefficient and Euclidean distance to return the most relevant documents for any given query.

The system allows us to execute two types of queries; these queries are successfully executed using these two functions.

- Given two documents D1 and D2, it evaluates their degree of similarity.
- Given a query Q, it returns the n most relevant documents to answer Q.

```java
// Calcular distancia/similaridad
for (Map.Entry<String, Map<String, Integer>> entry : dbDocs.entrySet()) {
    String docId = entry.getKey();
    Map<String, Integer> docCounts = entry.getValue();

    double score;
    switch (metric.toLowerCase()) {
        case "cosine":
            score = 1.0 - computeCosine(queryCounts, docCounts); // distancia = 1 - similitud
            break;
        case "dice":
            score = 1.0 - computeDice(queryCounts, docCounts);
            break;
        case "euclidean":
        default:
            score = computeEuclidean(queryCounts, docCounts);
            break;
    }

    // Solo incluir si hay palabras en común (distancia > 0 o similitud > 0)
    if (score >= 0) {
        results.put(docId, score);
    }
}
```

Image 7. Example code to calculate distance with the different techniques.

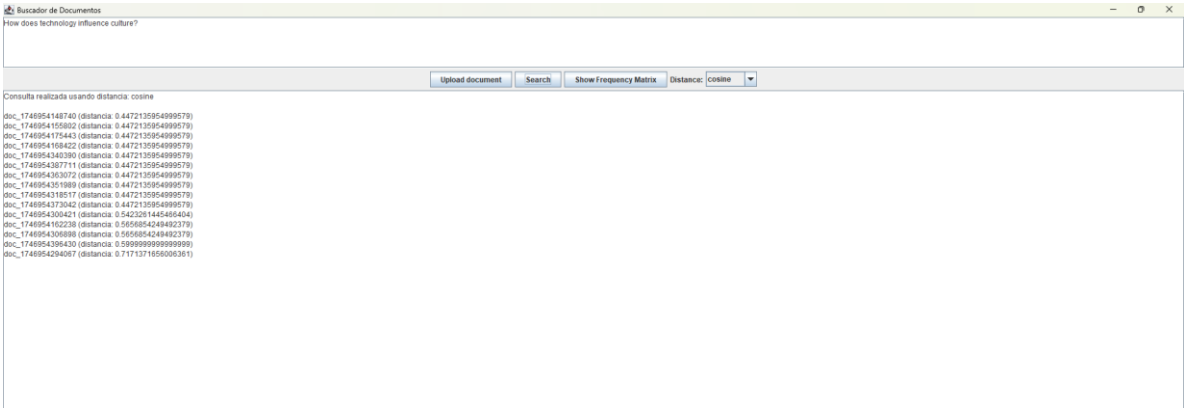


Image 8. Query using cosine similarity function technique to compare the documents.

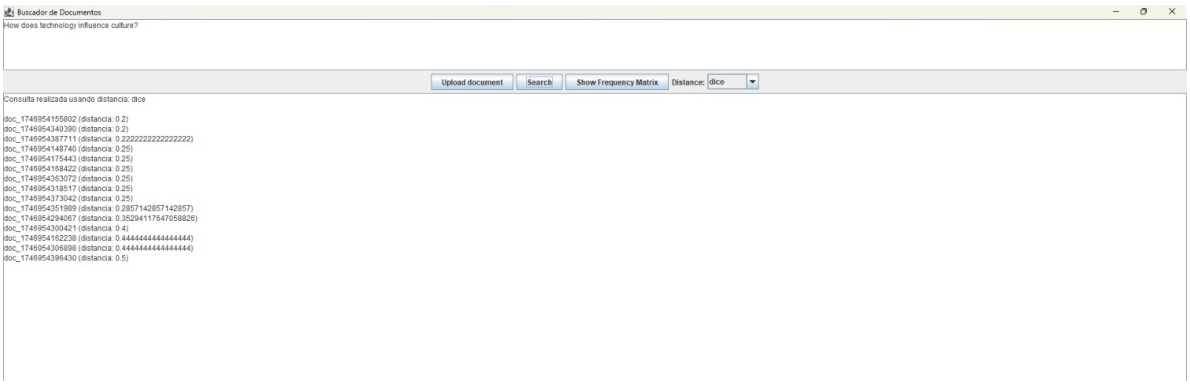


Image 9. Query using dice similarity function technique to compare the documents.

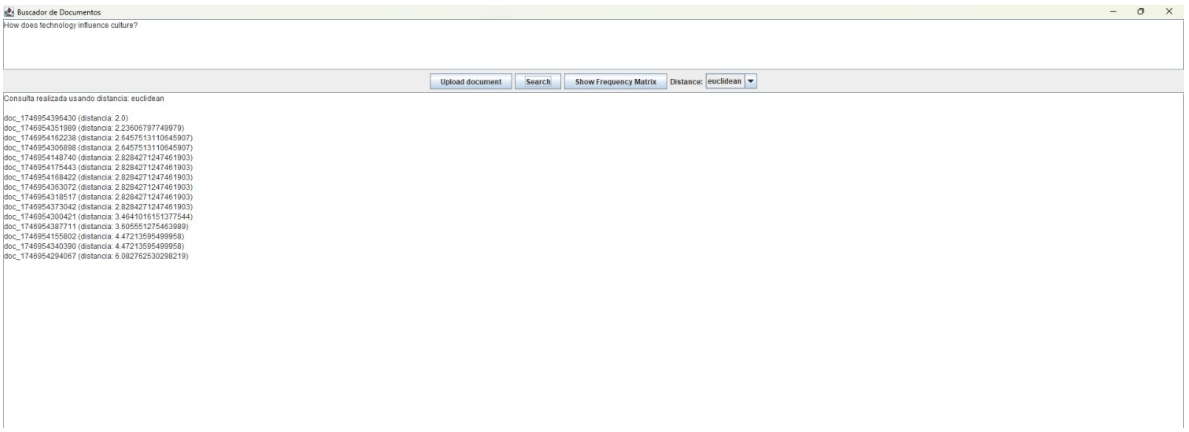


Image 10. Query using Euclidean distance dissimilarity function technique to compare the documents.

GitHub codes: https://github.com/KD08GG/Base-System-LSI-

CONCLUSION

This project demonstrates the application of engineering design to develop a document base system that uses Latent Semantic Indexing (LSI) to improve document retrieval. By implementing semantic indexing techniques such as Singular Value Decomposition (SVD), similarity and dissimilarity functions, the system goes beyond traditional keyword-based searches, enabling users to find relevant documents based on meaning rather than exact word matches. The system successfully processes and stores at least 10 technology-related documents, covering diverse areas such as health, education, environment, public safety, economy, and democracy, showing how engineering solutions can address real-world information retrieval challenges. It also incorporates non-technical considerations, including public health, environmental impact, and global accessibility, aligning with broader societal needs. The relational database schema, text preprocessing pipeline, and SQL-based querying functions provide a solid foundation for future enhancements, such as integrating TF-IDF weighting, expanding the user interface, or supporting larger document

collections. Overall, this project highlights how combining software development standards, information retrieval techniques, and engineering design principles leads to effective, scalable, and socially aware solutions.

REFERENCES

Snowball Stemming Algorithms. Retrieved from http://snowballstem.org/

Manning, C.D., Raghavan, P., Schütze, H. Introduction to Information Retrieval. Cambridge University Press, 2008.

ISO/IEC 12207:2008 — Systems and software engineering — Software life cycle processes.