Implementing Latent Semantic Indexing (LSI) Using Singular Value Decomposition (SVD)

Ömer Faruk Zeybek
Student Id: 150220743
Email: zeybeko22@itu.edu.tr
Major : Artificial Intelligence and Data Engineering

***Abstract*—The aim of this project is to create a type of SLI using Latent Semantic Indexing (LSI) and apply it on the dataset.It creates a term-document matrix using the SVD (Singular Value Decomposition) method and processes it with the LSI algorithm.As a result of the project, semantic relationships between specific words were discovered and these relationships could be effectively utilised across documents in the dataset**

I. Introduction

Latent Semantic Indexing (LSI) is a powerful technique used for indexing and retrieving information from large datasets. It employs Singular Value Decomposition (SVD), a mathematical method, to create a ”semantic space” that represents both terms and documents within a collection. This project focuses on implementing a form of LSI using the SVD method.

May 12, 2024

II. Data and Methodology

A. Data Set

The dataset for this project consists of consumer complaints about Comcast Corporation. The dataset includes two CSV files, and we will utilize the ”comcast consumeraffairs complaints.csv” file. It contains four columns: ”author,” ”posted on,” ”rating,” and ”text.” The ”text” column contains details of the complaints.

B. Methodology

1) Creating Term-by-Document Matrix: To implement LSI, we first need to create a term-by-document matrix. Each row in this matrix represents a different document (complaint), and the columns represent terms extracted from the complaint texts.

- Determining Terms: This involves several steps such as tokenization, stopword removal, stemming, and lemmatization to obtain a list of unique terms.
- Generating Term-by-Document Matrix: We construct a matrix where each element represents the frequency of a term in a document.
- Normalization: The matrix is then normalized using the numpy.linalg.norm function in Python.

2) Implementing SVD: The SVD decomposition is performed on the normalized term-by-document matrix. This involves decomposing the matrix into three sub-matrices: U, Σ, and V.

- Calculating Lower-Rank Approximation: We approximate the matrix A by using a lower-rank approximation to reduce its size while retaining essential structure. The optimum rank (k) is determined based on Mean Squared Error (MSE) and Frobenius Norm (FN) metrics.
- Query-Document Cosine Similarity: We calculate the cosine similarity between given queries and existing documents using reduced vectors.

III. Questions/Required Outputs

A. Q1

Determine the optimum rank (k) for the lower-rank approximation based on MSE and FN metrics.
In SVD, to assess the quality of the approximate matrix, we can calculate the reconstruction error. In this project I used two metrics to calculate the reconstruction error: 1) Mean Squared Error (MSE), 2) Frobenius Norm (FN). These two evaluation metrics are used to quantify the difference between two matrices.

1) Code:

```
# Words to a list
all_words = list(word_counts.keys())

# BoW matris
Bow_matrix = np.zeros((len(data), len(all_words)))

for i, row in enumerate(data["stemmed"]):
    for word in row.split():
        if word in all_words:
            j = all_words.index(word)
            Bow_matrix[i, j] += 1

# Applying SVD
U, Sigma, V_T = np.linalg.svd(Bow_matrix)

SE
def calculate_MSE(A, A_hat):
    return np.mean((A - A_hat) ** 2)
```

```
def calculate_FN(A, A_hat):
    return np.linalg.norm(A - A_hat)



t, d = Bow_matrix.shape
k_range = range(10, min(t, d) // 10 + 1, 20)

best_k_MSE = None
best_MSE = float('inf')

best_k_FN = None
best_FN = float('inf')

for k in k_range:
    U_k = U[:, :k]
    Sigma_k = np.diag(Sigma[:k])
    V_T_k = V_T[:k, :]

    A_hat = np.dot(np.dot(U_k, Sigma_k),
    V_T_k)

    MSE = calculate_MSE(Bow_matrix, A_hat)
    FN = calculate_FN(Bow_matrix, A_hat)

    if MSE < best_MSE:
        best_k_MSE = k
        best_MSE = MSE

    if FN < best_FN:
        best_k_FN = k
        best_FN = FN

print("Optimum k for MSE:", best_k_MSE)
print("MSE:", best_MSE)
print("Optimum k for FN:", best_k_FN)
print("FN:", best_FN)
```

2) Output:

```
Optimum k for MSE: 510
MSE: 0.0012205662585910754
Optimum k for FN: 510
FN: 296.32384280448014
```

B. Q2

Find the most relevant document for each given query using query-document cosine similarity.

$$\mathrm{E}quation9 : \mathrm{sim}(q, d) = \mathrm{sim}(q^{-1}_{\mathrm{TU}k}, d^{-1}_{\mathrm{TU}k}) \qquad (1)$$

$$\mathrm{E}quation10 : \mathrm{sim}(q, d) = \vec{q} \cdot \vec{d}\frac{||q|| \cdot ||d||}{||q|| \cdot ||d||} \qquad (2)$$

For this question I used Equation 10.

1) Code:

```
def calculate_cosine_similarity(query_vector,
document_matrix):
    similarities = []

for doc_vector in document_matrix:
    dot_product = np.dot(query_vector, doc_vector)
    similarity = dot_product /
    (np.linalg.norm(query_vector) *
    np.linalg.norm(doc_vector))
    similarities.append(similarity)

return similarities
```

2) Output:

```
Most relevant document for query1: Document 9,
Similarity: 0.27
Most relevant document for query2: Document 280,
Similarity: 0.30
Most relevant document for query3: Document 208,
Similarity: 0.30
Most relevant document for query4: Document 282,
Similarity: 0.35
```

IV. Conclusion and Thoughts

This project focuses on information extraction from large datasets using Latent Semantic Indexing (LSI). Here are some important points I discovered:

- Latent Semantic Indexing (LSI) is a great tool for extracting meaningful information from large datasets. In particular, it is really powerful for discovering hidden links between documents.
- Mathematical techniques such as Singular Value Decomposition (SVD) have helped me better understand how LSI works. This allows me to represent data in a more meaningful way.
- Experiments on the dataset used in the project have shown that LSI can be successfully applied on real-world data.

Since I was dealing with a large data set at the beginning, I was very tired of the long time it took to calculate the matrix. Later, after finding a simple formula, this situation also disappeared. While doing this project, I got a better understanding of how to work with LSI and SVD. I hope I can make better use of this experience in my future projects.