Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Engineering

Name of the Student: Shashwat Shah **Roll Number: O201**

SAP ID: 60004220126 **Class: C2** **Division: C2** **Batch: C22**

Subject: Web Intelligence

DATE OF PERFORMANCE: 13/02/2025 **DATE OF SUBMISION: 13/02/2025**

EXPERIMENT NO: 2

AIM: Implementation of Latent Semantic Analysis/Latent Semantic Indexing (CO1)

SOFTWARE/IDE USED: Google Colab/Jupyter Notebook

THEORY:

1. What is Latent Semantic Analysis (LSA)?
   Latent semantic analysis (**LSA**) is a technique in natural language processing, in particular distributional semantics, of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. LSA assumes that words that are close in meaning will occur in similar pieces of text (the distributional hypothesis). A matrix containing word counts per document (rows represent unique words and columns represent each document) is constructed from a large piece of text and a mathematical technique called singular value decomposition (SVD) is used to reduce the number of rows while preserving the similarity structure among columns. Documents are then compared by cosine similarity between any two columns. Values close to
   1 represent very similar documents while values close to 0 represent very dissimilar documents.

2. What is the main objective of LSA in text processing?
   The main objective of **Latent Semantic Analysis (LSA)** in text processing is to discover the hidden, or **latent**, relationships between words and documents in a corpus. LSA reduces the dimensionality of text data, identifying patterns in word usage and co-occurrence that are not immediately apparent by using methods like **Singular Value Decomposition (SVD)**. This process allows LSA to capture the semantic meaning of words in context, even when different words are used to describe similar concepts.

3. How does LSA help in identifying relationships between words and documents?
   LSA identifies relationships between words and documents by applying **Singular Value Decomposition (SVD)** to the term-document matrix (TDM). By reducing the matrix's dimensionality, LSA groups terms and documents into a smaller set of latent (hidden) topics that capture the underlying structure of the data. Words that are often used together or share similar contextual usage patterns will be grouped into the same latent topic, revealing

relationships between words (semantic similarity) and between documents (content similarity). This allows LSA to discover associations and relationships that may not be apparent through surface-level word matching alone.

4. What are the key advantages of using LSA in information retrieval?
   The key advantages of using LSA in information retrieval are:

- **Dimensionality Reduction**: LSA reduces the complexity of the term-document matrix, eliminating noise and irrelevant information. It can capture the most important semantic patterns with fewer dimensions, improving the retrieval process.
- **Improved Query Matching**: LSA can match documents to queries even when the query terms do not exactly match words in the documents, as it captures semantic similarity.
- **Noise Reduction**: LSA can de-emphasize words that are rare or irrelevant (e.g., common stopwords or uncommon terms) that do not contribute much to the semantic meaning.
- **Handling Synonymy**: LSA helps with synonymy (different words with similar meanings) by grouping related words together, allowing for better matching between documents and queries.

5. What are some real-world applications of LSA?
   Some real-world applications of **LSA** include:

- **Information Retrieval**: Used in search engines to improve the relevance of search results by capturing the underlying meaning of the query and the document content.
- **Document Clustering**: Grouping similar documents together by analyzing their latent semantic structure.
- **Topic Modeling**: Identifying topics in large text corpora, useful in fields like social media analysis, academic research, and news categorization.
- **Recommender Systems**: Suggesting similar documents or items based on latent semantic relationships, such as recommending articles, products, or movies.
- **Text Summarization**: Reducing a large document to a summary by capturing its most important concepts or latent topics.
- **Sentiment Analysis**: Understanding the sentiment or opinion expressed in text by identifying topics related to positive or negative sentiments.

6. What is Singular Value Decomposition (SVD), and how does it relate to LSA? What is term document matrix (TDM)?
   Singular Value Decomposition (SVD) is a matrix factorization technique that decomposes a matrix into three smaller matrices: one representing the term-space, one representing the document-space, and one diagonal matrix of singular values. In the context of LSA, SVD is used to reduce the dimensions of the **Term-Document Matrix (TDM)** by identifying the most important singular values and the corresponding word and document vectors. This allows LSA to capture the most relevant semantic relationships while ignoring the noise or less important components in the data.

   Term-Document Matrix (TDM) is a matrix where:

   - Rows represent terms (words).
   - Columns represent documents.
   - Each entry in the matrix represents the frequency or weight (e.g., TF-IDF) of a term in a document.

By applying **SVD** to the TDM, LSA reduces the matrix's dimensionality, focusing on the most significant singular values and latent structures that best represent the semantic content.

7. How does LSA differ from TF-IDF (Term Frequency - Inverse Document Frequency) in document ranking? How does the cosine similarity measure help in finding document similarity in LSA?

   L**SA vs TF-IDF in Document Ranking**:

   - **TF-IDF** is a statistical method used to rank documents based on the term frequency within a document and its inverse frequency across the entire corpus. It measures how important a word is to a document based on its occurrence in that document versus across all documents. However, TF-IDF is a **shallow representation** of the text that doesn't capture deeper, latent semantic relationships between terms.
   - **LSA**, on the other hand, uses **SVD** to reduce dimensionality and capture latent semantic structures in the text. It focuses on **concepts and topics** rather than individual terms, so LSA can rank documents based on their semantic similarity, even if they do not share common terms. LSA is **more robust** in dealing with synonymy and polysemy (words with multiple meanings).

   Cosine Similarity in LSA: **Cosine similarity** is a measure of similarity between two vectors based on the cosine of the angle between them. In the context of LSA, cosine similarity is often used to measure the similarity between document vectors in the **reduced latent space** (after SVD). The cosine similarity gives a value between -1 and 1, where 1 indicates identical direction (maximum similarity), 0 indicates no similarity, and -1 indicates opposite directions (completely dissimilar).

   By using **cosine similarity** between the vector representations of documents in the reduced LSA space, we can rank documents by how closely they relate to a query or to each other. This helps in identifying documents that are semantically similar, even when they do not share the same words.

8. What are the limitations of LSA compared to Latent Dirichlet Allocation (LDA)?

- Latent Structure Assumptions:
  - LSA assumes that co-occurrence patterns between words are the primary structure, using Singular Value Decomposition (SVD). It doesn't model topics explicitly.
  - LDA assumes documents are mixtures of latent topics, with each topic being a distribution over words. It uses a probabilistic approach.
- Interpretability:
  - LSA produces continuous latent vectors, making topics hard to interpret and label.
  - LDA produces discrete topics, each with a distribution over words, making it easier to understand and label topics.
- Probabilistic vs. Linear:
  - LSA is a linear model that captures correlations between terms in a reduced space.
  - LDA is probabilistic, modeling the generative process behind words and topics.

- Handling Synonymy and Polysemy:
  - LSA handles synonymy well but struggles with polysemy (same word, different meanings).
  - LDA handles both synonymy and polysemy better due to its probabilistic modeling of words across topics.
- Scalability and Flexibility:
  - LSA is faster but less flexible and harder to scale with large datasets.
    LDA is more computationally intensive but can better scale and allows explicit topic specification.

IMPLEMENTATION:
Perform implementation for the same using Python programming.

Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity

# Sample documents
documents = [
    "The cat is on the mat.",
    "The dog is in the house.",
    "The cat and the dog are friends.",
    "I love programming with Python.",
    "Machine learning is fun and exciting."
]

# Step 1: Vectorize the documents using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(documents)

# Step 2: Apply LSA using Truncated SVD (reduce to 2 latent dimensions)
lsa = TruncatedSVD(n_components=2, random_state=42)
lsa_matrix = lsa.fit_transform(tfidf_matrix)

# Step 3: Display vocabulary and documents
print("Vocabulary:", vectorizer.get_feature_names_out())
print("\nDocuments:")
for i, doc in enumerate(documents):
    print(f"Document {i + 1}: {doc}")

# Step 4: Query
query = "I enjoy learning about Python programming."
```

```
# Step 5: Transform the query using the same TF-IDF vectorizer
query_vector = vectorizer.transform([query])
query_lsa_vector = lsa.transform(query_vector)

# Step 6: Calculate cosine similarity between the query and all documents
similarities = cosine_similarity(query_lsa_vector, lsa_matrix)
most_similar_doc_index = similarities.argmax()

# Step 7: Output the most similar document
print(f"\nQuery: '{query}'")
print(f"Most similar document: Document {most_similar_doc_index + 1}:
'{documents[most_similar_doc_index]}'")
```

Output:

```
Vocabulary: ['cat' 'dog' 'exciting' 'friends' 'fun' 'house' 'learning' 'love'
 'machine' 'mat' 'programming' 'python']

Documents:
Document 1: The cat is on the mat.
Document 2: The dog is in the house.
Document 3: The cat and the dog are friends.
Document 4: I love programming with Python.
Document 5: Machine learning is fun and exciting.

Query: 'I enjoy learning about Python programming.'
Most similar document: Document 4: 'I love programming with Python.'
```

CONCLUSION: LSA effectively reduces dimensionality and captures semantic relationships like synonyms and polysemy, making it valuable for text processing. However, it faces computational challenges with large datasets due to SVD's complexity. Preprocessing steps like stop-word removal and stemming further enhance its performance.

POST-EXPERIMENTAL EXERCISE:

1. What are the computational challenges of applying LSA to very large datasets?

- High Memory Usage: The term-document matrix can become very large, requiring significant memory.
- Singular Value Decomposition (SVD): LSA relies on SVD, which is computationally expensive for large matrices ($O(mn^2)$ complexity).
- Scalability: Incremental updates are difficult since SVD is typically computed in one large batch.
- Noise Sensitivity: Large datasets can introduce noise that may distort the decomposition.

2. Given a new query document, how can you use LSA to find the most relevant document in the dataset?

- Convert the Query: Transform the query into the same term space as the original dataset using the same term-document matrix.
- Projection: Project the query into the lower-dimensional LSA space using the SVD matrices (U, Σ, V).
- Similarity Calculation: Compute the cosine similarity between the query's vector and all document vectors in the reduced space.
- Rank Documents: Rank the documents by similarity to find the most relevant one.

3. How does LSA reduce dimensionality, and why is it beneficial for text processing?

- Dimensionality Reduction: LSA uses SVD to decompose the term-document matrix into three matrices (U, Σ, V), retaining only the top k singular values and corresponding vectors. This reduces the feature space to k dimensions.
- Benefits:
  - Noise Reduction: By discarding smaller singular values, noise and irrelevant features are filtered out.
  - Improved Generalization: Captures underlying patterns in the data rather than surface-level word co-occurrences.
  - Efficiency: Reduces computational complexity and storage requirements.

4. How does LSA handle synonyms and polysemy in a text corpus?

- Synonyms: LSA maps words with similar meanings to similar positions in the reduced vector space by identifying co-occurrence patterns across multiple documents.
- Polysemy: LSA captures multiple contexts of the same word by distributing its meaning across different latent dimensions, mitigating the effect of ambiguity.

5. Explain the effect of removing stop words and stemming before applying LSA.

- Stop Words Removal: Eliminating common words (e.g., "the," "is") prevents them from dominating the term-document matrix, which improves the quality of the latent semantic space.
- Stemming: Reducing words to their root form ensures that variations (e.g., "running" and "run") are treated as the same term, improving the clustering of related terms and reducing dimensionality. Both steps enhance LSA's ability to capture meaningful patterns.