Latent Semantic Analysis with Randomized SVD

Final Report

Authors:
Lucio Baiocchi (ID: 360244)
Leonardo Passafiume (ID: 358616)

Computational Linear Algebra for Large Scale Problems
Politecnico di Torino
Academic Year 2025/2026

Contents

Chapter 1

Introduction to Latent Semantic Analysis

1.1 The Information Retrieval Problem

In the context of Natural Language Processing (NLP), one of the main challenges is extracting meaning from large corpora of unstructured text. Classical methods based on simple keyword matching suffer from two fundamental problems:

- **Synonymy**: Different words can have the same meaning (e.g., "car" and "automobile").
- **Polysemy**: A single word can have different meanings depending on the context.

Latent Semantic Analysis (LSA) addresses these issues by assuming the existence of a "latent" structure in the associations between words and documents. The goal is to project documents into a reduced-dimensional vector space that captures semantic concepts rather than mere lexical presence.

1.2 Vector Representation and Preprocessing

Before applying linear algebra algorithms, text must be transformed into a numerical matrix. In our project, this process is handled by the `stemming.py` module and the `SemanticEngine` class.

1.2.1 Preprocessing

As implemented in the `stemmed_tokenizer` function, the text undergoes:

1. **Tokenization**: Splitting the text into elementary units (tokens).
2. **Filtering**: Removing punctuation and non-alphabetic characters.
3. **Stemming**: Reducing words to their root form (e.g., "computing" $\rightarrow$ "comput") using the *SnowballStemmer*.

1.2.2 Term-Document Matrix and TF-IDF

The input matrix A for our analysis is constructed using the **TF-IDF** (Term Frequency - Inverse Document Frequency) technique. Given a collection of m documents and a vocabulary of n terms, the matrix is formally defined as $A \in \mathbb{R}^{m \times n}$.

In the `semantic_engine.py` module, we utilize `TfidfVectorizer` with specific preprocessing steps. We limit the feature space to the top $n = 15,000$ terms and include **n-grams** (specifically bigrams) to capture compound concepts such as "social_network" or "operating_system".

```
self.vectorizer = TfidfVectorizer(
    max_features=15000,
    stop_words=stemmed_stops,
    tokenizer=stemmed_tokenizer,
    token_pattern=None,
    ngram_range=(1, 2), # Captures local relations
    min_df=5
)
A = self.vectorizer.fit_transform(data)
```

Listing 1.1: Matrix construction in semantic_engine.py

Structure and Sparsity of Matrix A

The resulting matrix A exhibits several key properties due to the preprocessing configuration:

- **Dimensions:** The matrix has size $m \approx 18,846$ (rows, corresponding to documents) by $n = 15,000$ (columns, corresponding to the vocabulary).
- **Stemmed Features:** The columns do not represent raw words but *stemmed tokens*. For instance, "computer" and "computing" are collapsed into the single feature column `comput`.
- **N-Grams:** Due to `ngram_range=(1, 2)`, distinct columns exist for compound terms (e.g., `oper_system`) alongside single terms (e.g., `system`). This preserves local context.

A conceptual visualization of a sub-section of matrix A is shown below, where values $a_{i,j}$ represent the TF-IDF weight of term j in document i:

$A =$

	Vocabulary Features (Stemmed)					
	`analysi`	`comput`	`crash`	`oper_system`	`system`	...
Doc_1	0	0.34	0	**0.52**	0.28	...
Doc_2	0	0.31	**0.65**	0	0.25	...
Doc_3	**0.45**	0	0	0	0	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Finally, LSA involves finding a rank-k approximation of this matrix, A_k, which reduces noise and brings out latent semantic relationships between these features.

Chapter 2

Randomized SVD: Theory and Implementation

2.1 Singular Value Decomposition (SVD)

SVD is the mathematical foundation of LSA. Any real matrix $A \in \mathbb{R}^{m \times n}$ can be factorized as:

$$A = U\Sigma V^T \tag{2.1}$$

where U and V are orthogonal matrices containing the left and right singular vectors, and Σ is a diagonal matrix containing the singular values σ_i in descending order. The optimal rank-k approximation is obtained by truncating the sum to the first k elements.

However, calculating the exact SVD for very large matrices is computationally expensive, typically with a complexity of $O(\min(m^2n, mn^2))$.

2.2 The Randomized SVD (rSVD) Algorithm

To overcome computational costs, we use a probabilistic approach. The key idea is to project the matrix A into a subspace of much lower dimension that approximates its range, and then compute the SVD on this reduced matrix.

The algorithm implemented in the `functions.py` file follows these steps:

2.2.1 Phase 1: Probabilistic Range Finder

We want to find a matrix Q with orthonormal columns such that $A \approx QQ^T A$.

1. Generate a random Gaussian matrix $\Omega \in \mathbb{R}^{n \times (k+p)}$, where p is an *oversampling* parameter to improve stability.
2. Compute the "sketch" of the matrix: $Y = A\Omega$.
3. To improve the decay of the singular value spectrum, we apply **Power Iterations**. This step is crucial in NLP where the spectrum decays slowly:

$$Y = (AA^T)^q A\Omega$$

4. Orthonormalize Y to obtain Q (using QR decomposition).

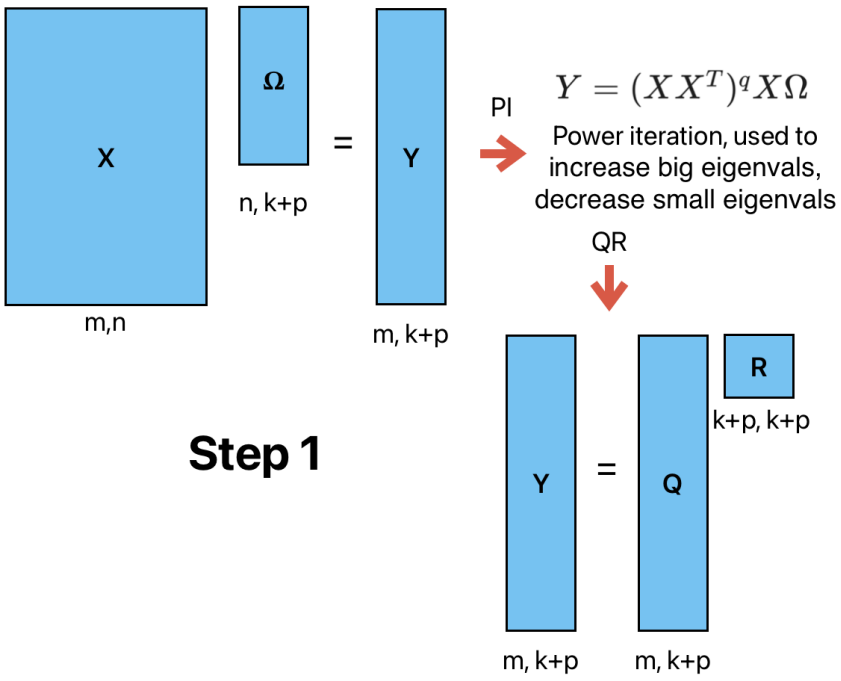

Figure 2.1: Phase 1 of the rSVD algorithm

2.2.2 Phase 2: Decomposition in the Reduced Subspace

Once the orthonormal basis Q captures the relevant range of A, we can perform the expensive SVD operation on a much smaller matrix. This phase consists of three logical steps:

1. **Projection to Low-Dimensional Space**
   We project the original matrix A onto the subspace defined by Q. This creates a new matrix B:
   $$B = Q^T A$$
   Why efficiently? While A is large $(m \times n)$, B is very "short" with dimensions $(k+p) \times n$. It contains all the dominant information of A but compressed into fewer rows.

2. **SVD on the Small Matrix**
   We compute the standard (deterministic) SVD on the small matrix B:
   $$B = \hat{U}\Sigma V^T$$
   Since B has only $(k+p)$ rows, this operation is extremely fast.

   - Σ and V^T obtained here are already the correct approximate singular values and right singular vectors for the original matrix A.
   - $\hat{U}$ represents the left singular vectors, but they are currently expressed in the compressed subspace coordinates.

3. **Recovering the Original Left Singular Vectors**
   To find the actual left singular vectors U relative to the original space (m-dimensional), we map $\hat{U}$ back using our basis Q:
   $$U = Q\hat{U}$$
   Finally, we discard the auxiliary oversampling components (p), keeping only the top k singular values and vectors:
   $$U_k = U_{(:,1:k)}, \quad \Sigma_k = \Sigma_{(1:k)}, \quad V_k^T = V_{(1:k,:)}^T$$

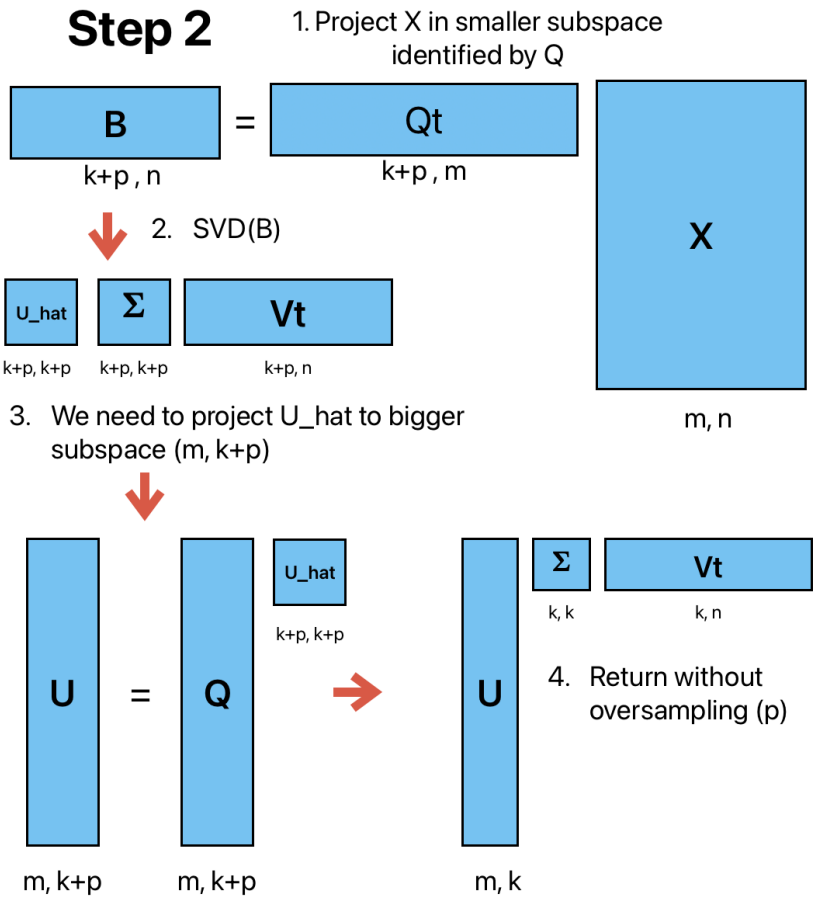

Figure 2.2: Phase 2 of the rSVD algorithm

2.3 Implementation Details

The `rSVD` function in `functions.py` implements this exact workflow:

```
def rSVD(X, k, p=10, q=2):
    m, n = X.shape
    # 1. Random Projection
    Omega = np.random.normal(size=(n, k + p))
    Y = X @ Omega

    # 2. Power Iterations for denoising
    for _ in range(q):
        Y = X @ (X.T @ Y)

    # 3. Orthonormal Basis
    Q, _ = np.linalg.qr(Y)

    # 4. Projection onto B
    B = Q.T @ X

    # 5. SVD of the reduced matrix
    U_hat, Sigma, Vt = np.linalg.svd(B, full_matrices=False)

    # 6. Reconstruction
    U = Q @ U_hat
    return U[:, :k], Sigma[:k], Vt[:k, :]
```

Listing 2.1: rSVD Implementation in functions.py

Note the use of $q = 2$ power iterations, a standard value that offers an excellent trade-off between accuracy and speed for matrices derived from textual data.

Chapter 3

Latent Semantic Application

3.1 Application in the Semantic Engine

In the `semantic_engine.py` file, the matrices U and V^T obtained via rSVD are used for:

1. **Document Clustering**: Documents are projected into the latent space defined by the columns of U. The **K-Means** algorithm is applied to this dense, low-dimensional representation. This allows grouping text that is semantically similar even if they do not share exactly the same words.

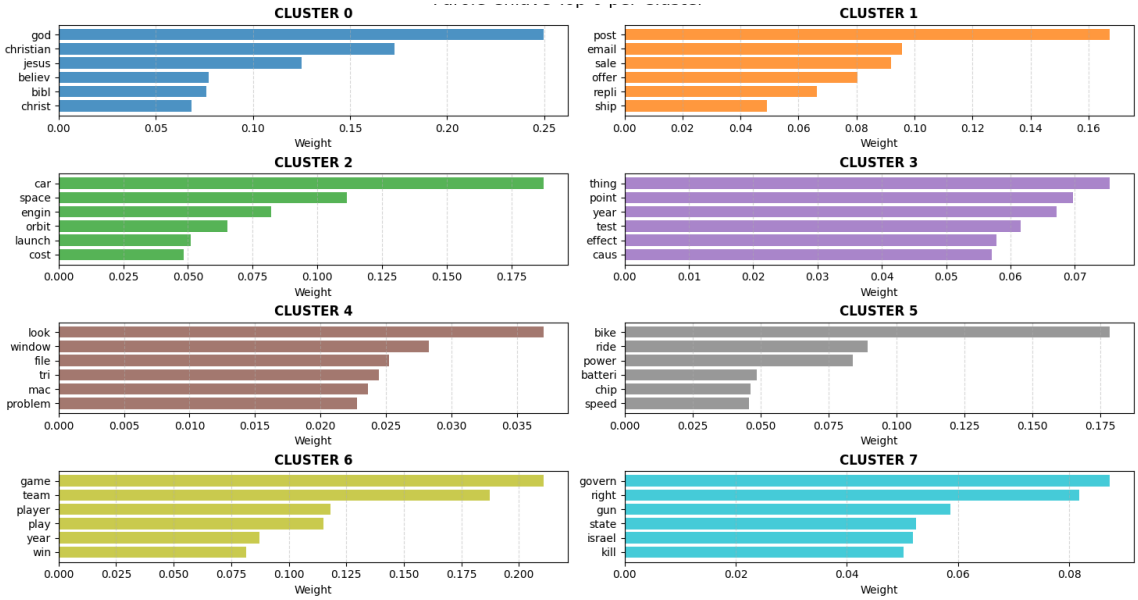

Figure 3.1: Top 6 keywords for each identified cluster. The bar length represents the weight of the term within the cluster centroid. The terms are displayed in their stemmed form, highlighting distinct semantic topics such as religion (Cluster 0), sports (Cluster 6), and politics (Cluster 7).

2. **Search Engine**: As seen in the `search` method, a user query is projected into the same latent space:

$$q_{topic} = q_{vec} \cdot V^T$$

Cosine similarity is then calculated between the projected query and the documents (U).

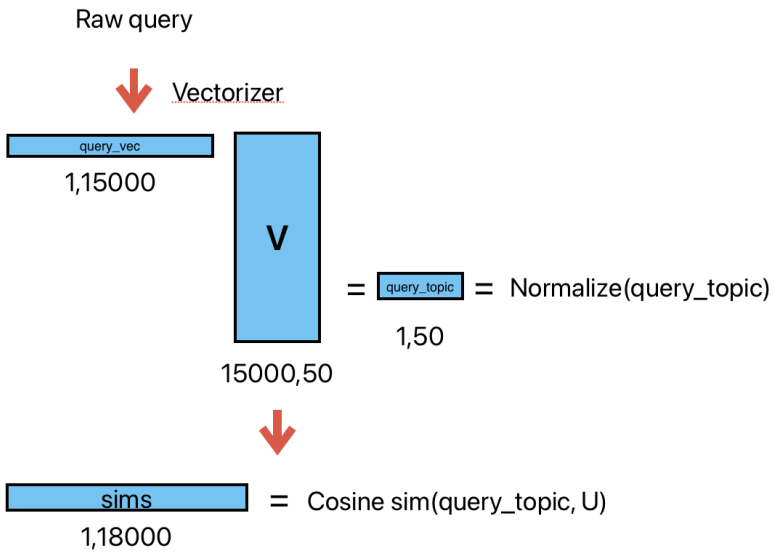

Figure 3.2: How does the search engine work: from the raw query to the cosine similarity

3.2 Conclusions

The implementation demonstrates that rSVD allows Latent Semantic Analysis to be performed in reduced time without sacrificing the quality of semantic results, enabling advanced features such as thematic clustering and efficient semantic search.