THEME : Matrix Factorization Algorithms

I. INTRODUCTION

Matrix factorization is a fundamental linear algebra technique with wide-ranging applications in machine learning, recommender systems, and data analysis. It decomposes a given matrix into a product of two or more matrices, revealing latent structures and reducing dimensionality. This report details the implementation of three core matrix factorization algorithms: Truncated Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), and Randomized SVD from scratch to power an interactive R Shiny dashboard, the "Matrix Factorization Lab." The project's goal is not just to implement the algorithms but to create an educational platform for intuitively understanding and comparing their behavior, strengths, and weaknesses.

II. MOTIVATION

Educational Clarity: Many existing machine learning libraries (e.g., scikit-learning Python, recommender Labin R) offer matrix factorization as black-box functions. While efficient, this obscures the underlying numerical processes, the impact of parameters like rank (k), and the convergence behavior of iterative methods like NMF. By implementing these algorithms from the ground up, we aim to demystify them, making the theory tangible through interactive visualization.

Addressing a Technical Challenge: The project tackles the challenge of building a stable, user-friendly interface for numerical computation. A common issue in such applications is the mismatch between data types (e.g., integers vs. floating-point numbers) that can cause algorithms to fail. A key motivation was to engineer a robust system that seamlessly handles data preprocessing, algorithm execution, and result visualization, providing reliable feedback to the user even in case of errors. The Shiny framework was chosen for its ability to create powerful web applications directly.

III. ALGORITHMS AND IMPLEMENTION

The implementation is structured across several files. The Shiny application (app.R) handles the user interface and logic, while the core algorithms are contained in a separate module (algorithms.R). The following describes the theoretical foundation and our implementation strategy for each algorithm.

1. Truncated Singular Value Decomposition (SVD)
   - Theoretical foundation: The full SVD factorizes a matrix $A(m \times n)$ into $A = U\,\Sigma V^T$, where $U(m \times n)$ and $V(n \times n)$ are orthogonal matrices containing the left and right singular vectors, and $\Sigma(m \times n)$ is a diagonal matrix containing the singular values. Truncated SVD computes only the first k singular values

and corresponding vectors, providing a rank-k approximation $A_k = U_k \Sigma_k V_k^T$. This is optimal in the least-squares sense (Eckart-Young-Mirsky theorem).

- Our implementation (my_trancated_svd):

→ Design Process: We relied on R's built-in svd()function for its numerical stability and efficiency. Our function is a wrapper that performs the full SVD and then truncates the result to the user-specified rank k.

→ Steps:

1. Compute Full SVD: result ← svd(A)
2. Truncate: Extract the first k columns of U, the first k elements of d (singular values), and the first k columns of V.
3. Return: A list containing U_k, D_k(the vector of singular values), and V_k.

2. Non-negative Matrix Factorization (NMF)

- Theoretical Foundation: NMF factorizes a non-negative matrix $A(m \times n)$ into two non-negative matrices $W(m \times k)$ and $H(k \times n)$ such that $A \approx WH$. It is solved as an optimization problem, minimizing the Frobenius norm $\| A - WH \|_F^2$. This is typically done using iterative update rules (Multiplicative Updates) that ensure non-negativity.
- Our Implementation (my_nmf):

→ Design process: We implemented the standard Multiplicative Update algorithm. This was the most complex algorithm to implement due to its iterative nature and the need to monitor convergence.

→ Steps:

1. Initialize: Randomly initialize $W$and $H$with non-negative values.
2. Iterate: For a predefined number of iterations (max_iter), apply the multiplicative update rules:
   - ❖ $H \leftarrow H \times \frac{(W^T A)}{W^T W H + \varepsilon}$(element-wise multiplication and division)
   - ❖ $W \leftarrow W \times \frac{(A H^T)}{W H H^T + \varepsilon}$. A small epsilon (ϵ) is added to prevent division by zero.
3. Track Convergence: After each iteration, calculate the reconstruction error $\| A - WH \|_F$ and store it in a vector.
4. Return: A list containing the matrices W and H, the vector of errors for plotting, and the final iterCount.

3. Randomized SVD

- Theoretical Foundation: For very large matrices, full SVD is computationally expensive. Randomized SVD is a probabilistic algorithm that uses random sampling to approximate the range of A with a smaller matrix Q which captures its essential actions. The SVD is then performed on this much smaller matrix, leading to significant speedups.

- Our Implementation (my_randomized_svd):
  → Design process: We implemented a basic version of the randomized SVD algorithm.
  → Steps:
    1. Random Projection: Create a random test matrix $\Omega(n \times (k+p))$, where p is an oversampling parameter (n_over samples).
    2. Form Projected Matrix: Compute $Y = A\Omega$ to form a matrix that approximates the column space of A.
    3. Orthonormal Basis: Compute the QR decomposition $Y = QR$ to get an orthonormal basis Q for the range of Y.
    4. Project A Down: Form the smaller matrix $B = Q^T A$.
    5. Compute SVD of B: Compute the SVD of this small matrix, $B = \overline{U}\,\Sigma V^T$.
    6. Project Back: Obtain the approximate left singular vectors of A as $U = Q\overline{U}$.
    7. Return: A list containing U, D (singular values), and V.

IV. RESULTS

The Shiny application successfully integrates these implementations, providing clear visual and quantitative results. The data_utils.R module generates synthetic datasets for testing.

Figure 1: Matrix Heatmaps: The plot_matrix_heatmap function effectively visualizes both the original and reconstructed matrices. For the "Movie Ratings" dataset, the sparse nature is clear, and the reconstruction fills in the missing entries based on the latent factors.

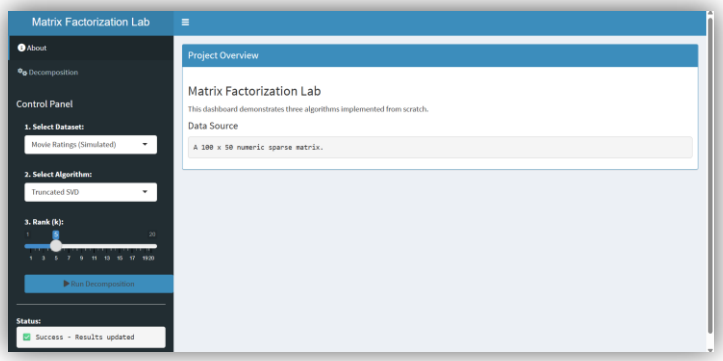
Figure 2 : Algorithm-Specific Plots :

- For NMF, the plot_convergence function displays the characteristic error curve, showing a rapid decrease that plateaus, confirming the algorithm's convergence.
- For SVD/RSVD, the plot shows the relative magnitude of the top-k singular values, illustrating the energy captured by the decomposition.

Performance Metrics: The value boxes in the app provide immediate feedback:

- Time: Randomized SVD consistently runs faster than Truncated SVD on larger matrices, demonstrating its efficiency.
- Error: The relative reconstruction error decreases as the rank k increases, as expected. NMF typically has a higher error than SVD for the same rank, as it is a more constrained problem.
- Iterations: This metric is active only for NMF, showing the number of iterations performed before the stopping criteria were met.

Below is the screenshot of our Shiny dashboard interface.



V. DISCUSSION

Efficiency and Stability:

- Truncated SVD is the most stable, leveraging R's optimized LAPACK routines. However, it is the slowest for large matrices.
- NMF is less stable; its performance and convergence can be sensitive to random initialization and the choice of the max_iter parameter. Our implementation is functional but not optimized for speed.
- Randomized SVD provides an excellent trade-off, offering near-optimal accuracy with significantly better computational efficiency on larger datasets, validating its purpose.

Comparison with Existing Implementations:

Our implementations are pedagogically focused and cannot match the speed, stability, and feature-completeness of professional libraries like **irlba**(for randomized SVD) or NMF (for NMF). For example, professional NMF implementations include multiple algorithms, sophisticated initialization methods, and automatic convergence detection. The value of our work lies in its transparency and integration into an interactive learning tool.

Potential Improvements:

- **Algorithmic Enhancements**: Implement more advanced techniques (e.g., SVD with implicit restarts for NMF, a wider range of randomized SVD algorithms).
- **Robustness**: Add more sophisticated convergence checks for NMF (e.g., based on error change between iterations).

- **Features**: Allow users to upload their own datasets and add more performance metrics like Mean Absolute Error (MAE).

VI. CONCLUSION

This project successfully achieved its goal of creating an interactive Matrix Factorization Lab. We implemented three core algorithms from scratch and integrated them into a robust Shiny application that effectively visualizes their behavior and performance. The platform serves as a powerful educational resource, bridging the gap between theoretical mathematics and practical application. While the implementations are not intended to compete with industrial-grade libraries, they provide clear, accessible, and interactive insights into the world of matrix factorization, fulfilling the core motivational aim of demystifying these essential algorithms. The modular codebase also provides a solid foundation for future extensions and improvements.