First project: Programming for data science course

Dr. Ahmed El Kerim*

Project Overview

In this project, we will develop a movie recommendation system using Singular Value Decomposition (SVD) and sparse matrices. The system will analyze a dataset of user ratings for movies and leverage SVD to perform dimensionality reduction and extract latent features. We will implement the recommendation algorithms using SciPy for numerical computations, lambda functions for data manipulation, and Object-Oriented Programming (OOP) principles for modular design.

- Gain practical experience in building recommendation systems using matrix factorization techniques.
- Understand the theory and applications of Singular Value Decomposition (SVD) for collaborative filtering.
- Learn to handle large, sparse datasets efficiently using sparse matrices.
- Apply lambda functions for concise and efficient data manipulation.
- Develop modular and reusable code using Object-Oriented Programming (OOP) principles.

1 First Part: Matrix Operations with Sparse and Dense Matrices

1.1 Matrix Class Hierarchy

- Define a base `Matrix` class with common methods for matrix operations.
- Implement two child classes: `DenseMatrix` and `SparseMatrix`, to handle dense and sparse matrices respectively.
- Ensure that both child classes can handle square and non-square matrices.

*Laboratory of Mathematics in Interaction with Computer Science, CentraleSupélec, Paris-Saclay University, Gif-sur-Yvette, France.

1.2 Implement Basic Matrix Operations (Libraries not allowed)

- Define methods for basic matrix operations, including matrix-vector multiplication, matrix-matrix multiplication, addition, and subtraction.
- Optimize these operations for dense and sparse matrices to ensure efficiency.
- Implement efficient algorithms specifically tailored for handling sparse matrices (Implement your storage).
- Implement Methods allowing to compute the norms: L_1, L_2, L_∞ for a given matrix (Check the formulas in the internet).

1.3 Eigenvalue and SVD Computations

- Implement methods to compute eigenvalues for square matrices.
- Perform Singular Value Decomposition (SVD) for non-square matrices.
- Utilize numerical libraries like SciPy to ensure accurate and efficient computations.

1.4 Linear System Solving

- Implement methods to solve linear systems of equations for both dense and sparse matrices.
- Optimize algorithms for handling sparse matrices by utilizing storage functions optimized for sparsity.

1.5 Testing and Timing Comparison

- Create test cases with matrices of size 500×500 to ensure the correctness of implemented methods.
- Profile each implemented method to identify performance bottlenecks and optimize code accordingly.
- Measure the time operations take on sparse and dense matrices and compare their performance.
- Increase the size of the matrices incrementally, doubling the size each time until reaching the limits of your computational resources.
- Plot the performance comparison results to visualize the efficiency of operations on sparse and dense matrices as the matrix size increases.

2 Second Part: Recommendation algorithm based on the Singular Value Decomposition

This section will explore the workings of the Netflix recommendation system, which relies on collaborative filtering. When Netflix suggests a movie, it considers users' similar tastes and preferences. For example, if you enjoy action movies, Netflix might recommend thrillers or superhero films. This principle applies to movies and other media platforms like Spotify, which suggests music based on your preferences. We'll delve into the mechanics of these recommendation systems using a straightforward dataset and Python libraries such as NumPy and SciPy. Additionally, we'll leverage mathematical techniques like Singular Value Decomposition (SVD). SVD offers a powerful tool for extracting valuable data insights while reducing its dimensionality. This process, known as dimensionality reduction, is a fundamental application of SVD in various fields.

2.1 Lambda Function Application: Matrix Transformation

Imagine you have a matrix filled with numerical values and wish to convert it into a binary matrix where each element is either 0 or 1 based on specific conditions.

- Utilize lambda functions to achieve this transformation from a full matrix to a binary representation. You can consider the technique where each element becomes one if its counterpart in the original matrix exceeds a threshold, such as 5 and 0 otherwise. You are welcome to explore alternative approaches that suit your objectives.

- Create test cases with square and non square matrices to ensure the correctness of implemented methods.

2.2 Data Processing

- Utilize the matrix class to generate random data, specifically non square matrix with rows >> columns.

- Apply the previously implemented function (lambda functions) to transform this matrix into a binary matrix.

2.3 SVD of the rating matrix

Now, we have a binary matrix representing a dataset corresponding to a rating matrix of movies. In this matrix, rows represent users, columns represent movies, and entries indicate ratings. Specifically, $A_{i,j}$ denotes the rating given by the i^{th} user for the j^{th} movie, where one indicates that the user liked the film and 0 suggests that they disliked it.

Analyzing this matrix allows us to uncover similarities between users with identical movie preferences. This similarity could stem from factors such as shared genre preferences or a tendency to enjoy movies featuring specific actors or actresses. We anticipate that singular value decomposition (SVD) will capture these patterns in some capacity.

- Perform singular value decomposition on the rating matrix from scipy.linalg and numpy.linalg.
- Compare the computation time and keep the one giving the best performance in your case.
- Plot the singular values.

SVD remark

Singular Value Decomposition (SVD) is a technique where we decompose the user-item matrix into three separate matrices. These matrices play a crucial role in making recommendations. But why do we need to do this? By performing SVD, we find a new representation for our data. We can then truncate this representation and retain only the most significant parts of the "new matrices." Let's visualize these new matrices to understand why this approach is practical.

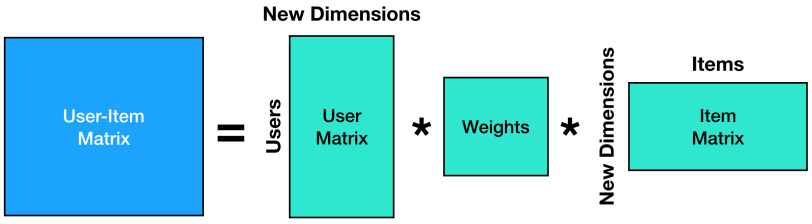

Figure 1: The output of SVD. These new matrices are usually labeled as U, Σ, and V_transpose (VT), respectively. Source: `http://zwmiller.com/projects/simple_recommender.html`

2.4 Dimensionality Reduction with SVD

for dimensionality reduction, as we have seen earlier. More technically speaking, it corresponds to the number of "concepts" or dimensions we will extract from the matrix.

- Select an appropriate number of singular values to retain based on your chosen criterion.
- Print the values in U and V^t, It's evident that the last few columns of matrix U contains values so small that their contribution to the data will be minimal.

2.5 Recommendation Algorithm

The intuition behind the recommendation algorithm is a distance calculation. Now that we understand what SVD does, we must code a recommender function that uses distance calculation to output movie recommendations. A dot product is used as a means of determining distance. An advantage of using the dot product is that it is computationally not expensive and easy to code. The recommend function recommends an output number of movies given the user liked the movie. For example, some users liked Movie A and are looking for two more movies similar to Movie A. Then, we can call the function above by passing appropriate arguments. Then, the function tells us the closest Movies to movie A (The more significant the dot product, the closer the movie).

Algorithm 1: Recommendation Algorithm

Input: liked_movie_index, VT, Selected_movies_num
Output: final_recomed_list
Function `recommend`(*liked_movie_index, VT, Selected_movies_num*)**:**
    recomended ← [];
    for *i* ∈ *range(len(VT.rows))* **do**
        if *i* ≠ *liked_movie_index* **then**
            rec.append([i, dotProduct(VT[i], VT[liked_movie_index])]);
        end
    end
    final_rec = sorted(rec) ;
    return *final_rec[:Selected_movies_num]*;

This pseudocode outlines the recommend function, which takes the liked movie index, the V matrix (VT) transpose, and a parameter Selected_movies_num, which gives the number of movies we want. It iterates through each item in the VT matrix, calculates the dot product between the liked movie and the current item, sorts the results in descending order based on the dot product values, and returns the top-recommended movies.

Deliverables

- Python code implementing both sections using SVD, sparse matrices, SciPy, lambda functions, and OOP principles.
- Documentation describing How to execute the code think about a Readme.
- Presentation slides summarizing the project goals, methodology, results, and potential extensions.
- Optional: Report with detailed results.