Chapter 5: Recommender System using SVD

5.1 What is a Recommender System [4]?

Recommender systems[10] are algorithms aimed at suggesting relevant items to users, such as movies, products, books, or any other content. Among the various techniques, **matrix factorization** techniques, especially **Singular Value Decomposition (SVD)**, have gained popularity for building collaborative filtering-based recommendation systems. In collaborative filtering, recommendations are based on past user-item interactions, without needing detailed information about the items or users. SVD helps in reducing the dimensionality of these interactions, capturing the underlying preferences and associations.

5.2 Objective

To design a recommendation system that analyses past user interactions with items and identifies non-interacted items that a user is highly likely to engage with and provide personalized suggestions, enhancing user experience and engagement.

5.3 Mathematical Principle

We are reconstructing the original rating matrix with reduced dimension preserving the most important latent information present in the original matrix which also helps in reducing the sparseness of the matrix. This new reconstructed matrix will have some insights on the user-item interactions which were not present before. We know that each entry in reconstructed matrix $\hat{R}$ is expressed as:

$$\hat{R}_{ij} = \sum_{p=1}^{k} \sigma_p u_{ip} v_{pj}$$

Here:

- u_i and v_j are generally non-zero.
- $As\ a\ result\ \hat{R}_{ij}$ tends to become non-zero even if it was zero in the original matrix R.

5.4 Procedure of building a Recommender System using SVD

Step 1: Preparing the data

- **Collect User-Item Interaction Data:** Gather data on how users have interacted with items. This could be explicit feedback (like ratings) or implicit feedback (e.g. clicks, purchases).
- **Create a User-Item Matrix:** Organize the data into a matrix R, where rows represent users, columns represent items, and the entries represent the interaction (e.g. rating).

$$R = \begin{bmatrix} 5 & 3 & 0 & 1 \\ 4 & 0 & 0 & 1 \\ 1 & 1 & 0 & 5 \\ 0 & 0 & 5 & 4 \\ 1 & 0 & 4 & 0 \end{bmatrix}$$

In this matrix:

- Rows represents users.
- Columns represent items.
- A '0' indicates that the user has not rated the item.

Step 2: Applying SVD

Using SVD, the matrix R is decomposed into three matrices:

$$R = U\Sigma V^T$$

- The columns of U represent users in the latent factor space.
- The rows of V^T represents items in the latent factor space.
- Σ contains the singular values, which represent the importance of each latent factor.

Step 3: Truncating the Matrices

To improve computational efficiency and avoid overfitting, we select only the top k singular values in Σ and their corresponding vectors in U and V^T. This reduces the dimensionality while capturing the most significant patterns.

Step 4: Reconstructing the Approximate User-Item Matrix [6]

After truncating, we can reconstruct an approximation of the original matrix R by multiplying the truncated matrices:

$$\hat{R} = U_k \Sigma_k V_k^T$$

The entries in $\hat{R}$ represent the predicted interactions or ratings between users and items.

Step 5: Making Predictions

Using the reconstructed matrix $\hat{R}$, we can:

- **Predict missing ratings:** If user u has not rated item i, $\hat{R}_{ui}$ provides a predicted rating.
- **Recommend items:** For each user, recommend items with the highest predicted ratings or interactions.

5.5 Example of SVD-based Recommendation

Suppose we have the following user-item rating matrix R:

$$R = \begin{bmatrix} 5 & 3 & 0 & 1 \\ 4 & 0 & 0 & 1 \\ 1 & 1 & 0 & 5 \\ 0 & 0 & 5 & 4 \\ 1 & 0 & 4 & 0 \end{bmatrix}$$

The full SVD decomposition of R factors it into three matrices: U, Σ and V^T.

$$R = U\Sigma V^T$$

For given R, we get,

$$U = \begin{bmatrix} 0.460393 & -0.650863 & -0.208074 & -0.543744 & -0.159571 \\ 0.321684 & -0.404323 & -0.147866 & 0.838722 & -0.0878797 \\ -0.47747 & -0.00895023 & 0.736696 & -0.00740583 & 0.478713 \\ 0.607137 & 0.584716 & -0.0378937 & -0.0138841 & -0.536528 \\ 0.296589 & 0.266321 & -0.625047 & -0.025293 & 0.670661 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 8.4647 & 0 & 0 & 0 \\ 0 & 6.5257 & 0 & 0 \\ 0 & 0 & 4.43223 & 0 \\ 0 & 0 & 0 & 1.76618 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0.515406 & -0.707087 & -0.342985 & 0.341682 \\ 0.219576 & -0.300587 & 0.0253764 & -0.927788 \\ 0.498782 & 0.611254 & -0.606841 & -0.0965885 \\ 0.661332 & 0.189853 & 0.716563 & 0.114605 \end{bmatrix}$$

Since we are reconstructing an approximation, we choose the top k components. Let's set $k = 2$, so we will keep only the first two largest singular values and their corresponding vectors.

The truncated matrices U_k, Σ_k and V_k^T are as follows:

$$U_k = \begin{bmatrix} 0.460393 & -0.650863 \\ 0.321684 & -0.404323 \\ -0.47747 & -0.00895023 \\ 0.607137 & 0.584716 \\ 0.296589 & 0.266321 \end{bmatrix}$$

$$\Sigma_k = \begin{bmatrix} 8.4647 & 0 \\ 0 & 6.5257 \end{bmatrix}$$

$$V_k^T = \begin{bmatrix} 0.515406 & -0.707087 & -0.342985 & 0.341682 \\ 0.219576 & -0.300587 & 0.0253764 & -0.927788 \end{bmatrix}$$

To approximate R, we compute:

$$\hat{R} = U_k \Sigma_k V_k^T$$

For $k = 2$, $\hat{R}$ is:

$$\hat{R} = \begin{bmatrix} 5.0118224 & 2.13240365 & -0.65240615 & 1.77089735 \\ 3.2690716 & 1.39099312 & -0.25462924 & 1.29984959 \\ 2.12438423 & 0.90500531 & 1.98019575 & 2.66177573 \\ -0.04922684 & -0.01848888 & 4.89571048 & 4.12315947 \\ 0.06507691 & 0.02885542 & 2.31452544 & 1.99024697 \end{bmatrix}$$

The reconstructed matrix $\hat{R}$ is an approximation of R, but with reduced dimensions. This approximation retains the most important patterns in the data, which allows us to make recommendations even for entries that were originally missing. Now for each user, recommend **top *t*** items with the highest predicted ratings among those items which were not rated before. For given system, we can see that user 2 has not rated movies 2 and 3 so by looking at the predicted ratings we can recommend movie 2 to user 2 as it has higher predicted ratings among the unrated movies.

5.6 Code

Here is the source code for Recommender System [9].

```
'''Hi! Pluto'''

import numpy as np
import pandas as pd
import math


movie_list = []


# Compute the eigenvalues and eigenvectors
def svd(A, k = -1):
    At_A = A.T@A
    eigenvalues, eigenvectors = np.linalg.eig(At_A)
    singularvalues = abs(eigenvalues) ** 0.5

    # Indices of sorted eigenvalues in descending order
    idx = np.argsort(singularvalues)[::-1]

    sorted_singularvalues = singularvalues[idx]
    sorted_eigenvectors = eigenvectors[:, idx]

    kx = len(A)
    ky = len(A[0])

    if k > min(kx, ky):
        k = min(kx, ky)

    if k != -1:
```

```
        kx = k
        ky = k

    Sigma = np.diag(sorted_singularvalues)[:kx,:ky]

    V = sorted_eigenvectors.T
    if k != -1 :
        V = V[:k,:]

    U = []
    for i in range(min(kx, ky)):
        vi = np.array([V[i]]).T
        ui = ((((A @ vi) / sorted_singularvalues[i]).T)[0]) if
sorted_singularvalues[i] != 0 else (np.zeros(kx))

        U.append(ui)

    U = np.array(U).T

    return U, Sigma, V


# Function to load data for recommender system

def load_data():
    ratings = pd.read_csv('Data/ratings.csv')
    movies = pd.read_csv('Data/movies.csv')

    # Create user-item matrix
    user_item_matrix = ratings.pivot(index='userId', columns='movieId',
values='rating').fillna(0)

    # Populate movie_list with the given information
    global movie_list
    movie_list = [
        {'movie_id': row['movieId'], 'movie_name': row['title'], 'genre':
row['genres']}
        for _, row in movies.iterrows()
    ]

    return np.array(user_item_matrix)



# Normalize the data (Standard scaling)
def normalize(X, max_rating):
    r_min = np.min(X)
    r_max = np.max(X)

    X = ((X - r_min) / (r_max - r_min)) * max_rating

    return X


def estimate_r(X, K = 5):
    u_k, sigma_k, v_t_k = svd(X, K)

    X = u_k @ sigma_k @ v_t_k

    X = normalize(X, 5)

    return X
```

```
def recommended_movies(R, R_hat, user_row, max_movies = 10):
    # Extract the user's predicted ratings and their original ratings
    user_predictions = R_hat[user_row]
    user_original_ratings = R[user_row]

    # Create a list of tuples (index, predicted rating)
    predictions_with_indices = [(index, rating) for index, rating in
enumerate(user_predictions)]

    # Sort the predictions in descending order of predicted rating
    predictions_with_indices.sort(key=lambda x: x[1], reverse=True)

    # Select top 10 movies that were not rated by the user (original rating is 0)
    recommendations = []
    for movie_index, predicted_rating in predictions_with_indices:
        if user_original_ratings[movie_index] == 0 and len(recommendations) <
max_movies:
            recommendations.append(movie_list[movie_index])

    return recommendations



def watched_movies_indices(R, user_row):
    # Find the indices where the rating is not zero
    user_ratings = R[user_row]  # Get the user's ratings row
    watched = []

    for movie_index, rating in enumerate(user_ratings):
        if rating != 0:  # Check if the movie is watched (rating is non-zero)
            watched.append(movie_list[movie_index])

    return watched



def human_readable_format(X):
    # Determine the width for the movie name column dynamically
    max_movie_name_length = max(len(movie['movie_name']) for movie in X)
    column_width = max_movie_name_length + 5  # Add some padding

    # Display recommendations
    print(f"{'Movie Name'.ljust(column_width)} Genre")
    print("-" * (column_width + 50))  # Separator line for clarity
    for movie in X:
        genres = ", ".join(movie['genre'].split('|'))  # Split and join genres
        print(f"{movie['movie_name'].ljust(column_width)} {genres}")



np.set_printoptions(suppress = True)

# Load dataset
R = load_data()

# R = R[:100, :250]

R_hat = estimate_r(R)
```

```
# input the user and display the recommended movies
user_row = 0  # Row number of the user (0-indexed)
watched = watched_movies_indices(R, user_row)
recommendations = recommended_movies(R, R_hat, user_row)


print("Already Watched Movies:")
human_readable_format(watched)


print("Recommended Movies:")
human_readable_format(recommendations)
```

5.7 Advantages and limitations

Advantages

- **Dimensionality Reduction:** Reduces the size of data, extracting the most important features and improving efficiency and scalability.
- **Mitigates Sparsity:** Can still provide recommendations even with sparse matrices which is a common problem in recommender system.
- **Theoretical Foundation:** As recommender system's result are very subjective, SVD which is a concept of linear algebra, with well-studied properties and numerical stability, ensures consistent and predictable performance.

Limitations

- **Computationally Intensive:** SVD computation can be heavy for very large matrices.
- **Cold Start Problem:** SVD relies on historical data, so new users or items with no interactions are challenging to recommend.
- **Static Approach:** SVD models are not inherently dynamic and need retraining as new data becomes available.