Applied Linear Algebra Project

Recommender Systems using Singular Value Decomposition (SVD)

Mohammad Hassan Heydari – Masoud MohammadZade

Introduction

In the vast world of movies, finding the ones that align with our unique tastes can be a challenge. This is where our movie recommender system comes into play. It’s designed to make the process of discovering new movies enjoyable and personalized.

This document presents the design and implementation of a movie recommender system based on Singular Value Decomposition (SVD), a powerful matrix factorization technique. SVD is widely used in recommender systems due to its ability to extract latent features that capture underlying patterns in the user-item interactions. These latent features enable the system to predict a user’s preferences even in the face of sparse data.

Our recommender system is built upon a dataset of movie ratings. It predicts a user’s rating for a movie they haven’t seen based on their past behavior and the behavior of other users. The system then recommends the movies with the highest predicted ratings, providing a personalized list of movie suggestions for each user.

The subsequent sections will delve into the details of data loading, preprocessing, the SVD process, and the recommendation mechanism. The document also includes the Python code for the recommender system and a discussion of the results.

Data Loading and Preprocessing

The first step in building our recommender system is to load and preprocess the data. This involves reading the data from CSV files, creating a user-item matrix, and handling missing values.

- Loading Data

  We start by loading our data from two CSV files: ‘movies.csv’ and ‘ratings.csv’. The ‘movies.csv’ file contains information about the movies, and the ‘ratings.csv’ file contains the ratings given by users to these movies. We use the **pd.read_csv()** function from the pandas library to load these files into dataframes.

```
# Load data from CSV files
movies = pd.read_csv('data/movies.csv')
ratings = pd.read_csv('data/ratings.csv')
```

- Creating User Item Matrix

  Next, we merge the ‘movies’ and ‘ratings’ dataframes on the ‘movieId’ column to create a new dataframe that contains the ‘userId’, ‘movieId’, and ‘rating’. This dataframe represents the interactions between users and movies.

```
  # Create a user-item matrix (Ratings matrix)
user_movie_ratings = pd.merge(ratings, movies, on='movieId')[['userId',
'movieId', 'rating']]
```

  We then transform this dataframe into a user-item matrix using the **pivot_table** function. Each row of this matrix represents a user, each column represents a movie, and each cell represents the rating given by a user to a movie.

```

# Create a pivot table for user-item matrix
user_movie_ratings_pivot =
user_movie_ratings.pivot_table(index='userId', columns='movieId',
values='rating')
```

- Handling Missing Values

  In the user-item matrix, not every user has rated every movie, so there are many missing values. We handle these missing values by filling them with zeros. This is based on the assumption that a missing rating means a rating of zero.

```
# Fill NaN values with zeros (assuming no rating means a rating of
zero)
user_movie_ratings_matrix = user_movie_ratings_pivot.fillna(0).values
```

After these steps, our data is ready to be used in the SVD process.

Singular Value Decomposition (SVD)

Singular Value Decomposition, or SVD, is a matrix factorization technique that decomposes a matrix into three other matrices. Given a matrix **X**, SVD decomposes it into **U**, **Σ**, and **Vt** such that **X = U * Σ * Vt**.

In the context of our recommender system, **X** is the user-item matrix, **U** is the user feature matrix, **Σ** is the diagonal matrix of singular values (representing the strength of each latent feature), and **Vt** is the item feature matrix.

- Initialization
  We start by initializing **U** and **Vt** with random values. **U** is a **m x m** matrix and **Vt** is a **n x n** matrix, where **m** is the number of users and **n** is the number of movies.

```
U = np.random.rand(m, m)   # Random initialization for U
Vt = np.random.rand(n, n)   # Random initialization for Vt
```

- Power Iteration
  Next, we use the power iteration method to update **U** and **Vt**. This involves multiplying **X** with **Vt** and **U** respectively, and then normalizing the result.

```
# Power Iteration for U
U = np.dot(matrix, Vt.T) # U = X * Vt
U = U / np.linalg.norm(U, axis=0) # Normalize U
```

```
# Power Iteration for Vt
Vt = np.dot(U.T, matrix) # Vt = U.T * X
Vt = Vt / np.linalg.norm(Vt, axis=0) # Normalize Vt
```

- Singular Values
  After updating **U** and **Vt**, we calculate the singular values. These are obtained by multiplying **U.T**, **X**, and **Vt**.

```
# Calculate singular values
sigma = np.diag(np.dot(U.T, np.dot(matrix, Vt))) # sigma = U.T * X * Vt
```

  This completes the SVD process. The resulting matrices **U**, **Σ**, and **Vt** capture the latent features in the user-item interactions, which can be used to make accurate predictions of a user's movie ratings.

Making Predictions

Once we have the matrices **U**, **Σ**, and **Vt** from the SVD process, we can use them to make predictions. The goal is to predict the ratings a user would give to movies they haven't seen yet.

- Choosing the number of Latent Features
  First, we choose the number of latent features (**k**) we want to consider. This is a hyperparameter that can be tuned to optimize the performance of the recommender system. In your case, you've chosen **k = 10**.

```
# Choose the number of latent features (k) - it's a hyperparameter
k = 10
```

- Approximating the Original Matrix
  We then approximate the original user-item matrix using the first **k** singular values and vectors. This involves selecting the first **k** columns of **U**, the first **k** singular values from **Σ**, and the first **k** rows of **Vt**.

```
# Use the first k singular values and vectors to approximate the
original matrix
U_k = U[:, :k] # U = [U_1, U_2, ..., U_k]
sigma_k = np.diag(sigma[:k]) # sigma = [sigma_1, sigma_2, ..., sigma_k]
Vt_k = Vt[:k, :] # Vt = [Vt_1, Vt_2, ..., Vt_k]
```

- Predicting Ratings
  We then calculate the predicted ratings by multiplying **U_k**, **sigma_k**, and **Vt_k**.

```
# Make predictions
predicted_ratings = np.dot(np.dot(U_k, sigma_k), Vt_k) # X = U * sigma
* Vt
```

The **predicted_ratings** matrix now contains the predicted ratings for each user-movie pair. These predictions can be used to recommend movies to a user.

Recommendations

After predicting the ratings a user would give to each movie, we can use these predictions to recommend movies to the user.

- Getting User ID
  First, we get the user ID from the user's input.

```
# Get user ID from input
user_id_input = int(input("Enter a user ID: "))
```

- ## Checking ID Validity

  We then check if the entered user ID is valid, i.e., if it exists in our dataset. If it doesn't, we print a message and stop the process.

```
# Check if the entered user ID is valid
if user_id_input not in user_movie_ratings_pivot.index:
    print(f"User ID {user_id_input} not found in the dataset.")
```

- ## Making Recommendations

  If the user ID is valid, we proceed to make recommendations. We start by getting the predicted ratings for the user.

```
# Get the index corresponding to the user ID
user_index = user_movie_ratings_pivot.index.get_loc(user_id_input)

# Get predicted ratings for the user
user_ratings = predicted_ratings[user_index, :]
```

We then find the indices of the movies with the highest predicted ratings. These are the movies that the user is likely to enjoy the most.

```
# Find indices of movies with highest predicted ratings
recommended_movie_indices = np.argsort(user_ratings)[::-1][:10]
```

Finally, we print the titles of the recommended movies.

```
# Print recommended movies
print(f"\nTop 10 Recommended Movies for User {user_id_input} : \n")
for index in recommended_movie_indices:
    movie_id = user_movie_ratings_pivot.columns[index] # Get movie ID
from column index
    movie_title = movies[movies['movieId'] ==
movie_id]['title'].values[0] # Get movie title from movie ID
    print(f"Title: {movie_title}")
```

Implementation

The Whole Implemented code in the python file is as follows :

```
import numpy as np
import pandas as pd

def data_loader():
    # Load data from CSV files
    movies = pd.read_csv('data/movies.csv')
    ratings = pd.read_csv('data/ratings.csv')

    # Create a user-item matrix (Ratings matrix)
    user_movie_ratings = pd.merge(ratings, movies, on='movieId')[['userId',
'movieId', 'rating']]

    # Create a pivot table for user-item matrix
    user_movie_ratings_pivot = user_movie_ratings.pivot_table(index='userId',
columns='movieId', values='rating')

    # Fill NaN values with zeros (assuming no rating means a rating of zero)
    user_movie_ratings_matrix = user_movie_ratings_pivot.fillna(0).values

    return user_movie_ratings_matrix, user_movie_ratings_pivot, movies


def SVD(matrix, num_iterations):
    # Initialize matrices
    m, n = matrix.shape # m = number of users, n = number of movies
    U = np.random.rand(m, m)  # Random initialization for U
    Vt = np.random.rand(n, n)  # Random initialization for Vt

    print('SVD started : \n')
    for i in range(num_iterations):

        print(f'Iteration {i + 1} / {num_iterations}')

        # Power Iteration for U
        U = np.dot(matrix, Vt.T) # U = X * Vt
        U = U / np.linalg.norm(U, axis=0) # Normalize U

        # Power Iteration for Vt
        Vt = np.dot(U.T, matrix) # Vt = U.T * X
        Vt = Vt / np.linalg.norm(Vt, axis=0) # Normalize Vt

    # Calculate singular values
    sigma = np.diag(np.dot(U.T, np.dot(matrix, Vt))) # sigma = U.T * X * Vt

    print('SVD finished \n')

    return U, sigma, Vt
```

```
user_movie_ratings_matrix, user_movie_ratings_pivot, movies = data_loader()

# Perform Singular Value Decomposition
U, sigma, Vt = SVD(user_movie_ratings_matrix, num_iterations = 5)

# Choose the number of latent features (k) - it's a hyperparameter
k = 10

# Use the first k singular values and vectors to approximate the original
matrix
U_k = U[:, :k] # U = [U_1, U_2, ..., U_k]
sigma_k = np.diag(sigma[:k]) # sigma = [sigma_1, sigma_2, ..., sigma_k]
Vt_k = Vt[:k, :] # Vt = [Vt_1, Vt_2, ..., Vt_k]

# Make predictions
predicted_ratings = np.dot(np.dot(U_k, sigma_k), Vt_k) # X = U * sigma * Vt

# Get user ID from input
user_id_input = int(input("Enter a user ID: "))

# Check if the entered user ID is valid
if user_id_input not in user_movie_ratings_pivot.index:
    print(f"User ID {user_id_input} not found in the dataset.")
else:
    # Get the index corresponding to the user ID
    user_index = user_movie_ratings_pivot.index.get_loc(user_id_input)

    # Get predicted ratings for the user
    user_ratings = predicted_ratings[user_index, :]

    # Find indices of movies with highest predicted ratings
    recommended_movie_indices = np.argsort(user_ratings)[::-1][:10]

    # Print recommended movies
    print(f"\nTop 10 Recommended Movies for User {user_id_input} : \n")
    for index in recommended_movie_indices:
        movie_id = user_movie_ratings_pivot.columns[index] # Get movie ID
from column index
        movie_title = movies[movies['movieId'] ==
movie_id]['title'].values[0] # Get movie title from movie ID
        print(f"Title: {movie_title}")
```

References

- GeeksForGeeks
- Introduction to Applied Linear Algebra, Stephen Boyd
- Youtube
- Microsoft Copilot
- Github Copliot