**Book Recommendation System - Project Report**

Project Owner: Suyog Desai

Affiliation: Arizona State University

**Project Overview:**

This project leverages data mining and collaborative filtering techniques to analyze a large-scale dataset of book ratings and build a personalized book recommendation system. Using proximity-based collaborative filtering, the project suggests books to users based on their similarity to other users.

**Dataset Information:**

Source:Book Crossing Dataset on Kaggle (<https://www.kaggle.com/datasets/somnambwl/bookcrossing-dataset>)

**Files Used:**

* Books.csv: Metadata about books including title, author, and ISBN
* Ratings.csv: User-provided book ratings on a scale of 0 to 10

**Dataset Stats:**

* Over 270,000 books
* More than 1 million ratings
* Wide and sparse user-book rating matrix

**Objective:**

The main goal is to develop a book recommendation engine that

* Uses collaborative filtering to compute user similarity
* Recommends top 5 books to each user that they haven't yet rated
* Efficiently handles large, sparse user-book rating data

**Tools & Libraries:**

**1. Python for Programming**

The primary programming language utilized in this project is Python. It was picked because of its ease of use, readability, and wide range of libraries that are appropriate for scientific computing, machine learning, and data analysis. Python facilitates rapid development and prototyping, enabling the effective application of algorithms like data processing, matrix manipulations, and collaborative filtering.

**2. Pandas: Data Manipulation**

Pandas is a robust Python data analysis and manipulation library. The Ratings.csv file, which includes user ratings for books, was loaded and cleaned using it. Pandas was used to efficiently factorize user IDs and ISBNs into numerical values and load the data into a DataFrame for convenient access and manipulation. Pandas also made it simple to handle and transform data before incorporating it into a recommendation system.

**3. NumPy: Numerical Computation**

NumPy is a fundamental Python package for scientific computing that supports large numerical calculations, arrays, and matrices. NumPy was utilized in this project to handle and work with numerical data, especially when determining user similarity scores. When working with large datasets and carrying out operations like weighted averages or similarity calculations, its array operations made vectorized computations quick and efficient.

**4. SciPy: Sparse Matrix Handling**

Built on top of NumPy, SciPy is a library that offers more features for scientific computing. SciPy's sparse matrix features were essential to this project's effective handling of the sizable, sparse user-book rating matrix. Because they only store non-zero values, sparse matrices save memory and lower the dataset's computational and memory footprint. The sparse matrix representation of user-book ratings was created and manipulated using SciPy's coo\_matrix and tocsr() functions, allowing computations on large datasets without encountering memory constraints.

**5. scikit-learn: Cosine Similarity Calculations**

scikit-learn is one of the most widely used libraries for machine learning in Python. In this project, it was utilized to determine the cosine similarity between users. The cosine of the angle between two vectors, which in this case represent the users' book rating patterns, is measured by a metric called cosine similarity. The project effectively computed user similarity scores using scikit-learn's cosine\_similarity() function, which were then utilized to produce tailored book recommendations according to user preferences.

**Data Preparation Workflow**

Step 1: Load Data

The Ratings.csv file is loaded using the correct delimiter ';' to separate user ID, ISBN, and rating.

Step 2: Create User-Book Matrix

* User-ID and ISBN are factorized to generate unique numeric IDs
* A sparse matrix is built using SciPy's coo\_matrix, where:
* Rows represent users
* Columns represent books
* Values are ratings

Step 3: Save Matrix in LIBSVM Format

* Matrix is exported in .libsvm format for efficiency
* LIBSVM requires a target value (y), so a zero-vector is used as a placeholder

Note: This placeholder y value is unused in analysis and is only needed for saving.

**Recommender System Methodology**

Task: Collaborative Filtering with Cosine Similarity

1. Similarity Calculation:

Cosine similarity is used to measure proximity between users based on their rating patterns.

2. Find Top K Similar Users (K=10):

For each user, retrieve 10 most similar users using similarity scores.

3. Aggregate Ratings from Similar Users:

* Gather books rated by similar users that the target user hasn't read
* Calculate a weighted average rating for each book:

r\_hat(u,b) = sum(sim(u,v) \* r(v,b)) / sum(sim(u,v))

4. Top-N Recommendations (N=5)

* Recommend top 5 books with the highest estimated ratings
* Skip users with no valid interactions or similarity scores

**Output Format**

The recommendation results are stored in Book-recommendations.csv with the following format:

User\_ID, Book\_ID, Title, Score. Each row represents a book recommended to a specific user along with the calculated recommendation score.

**Explanation of Output Results**

* For each user, the recommender identifies books read by similar users
* It excludes books already rated by the user
* Recommends top 5 books using similarity-weighted ratings
* Even users with no prior activity receive suggestions based on similar users (when possible)

**Metadata Mapping:**

Books.csv is used to map book IDs to their ISBNs and titles

**Dictionaries:**

* id\_to\_isbn: Maps internal Book\_IDs to actual ISBNs
* isbn\_to\_title: Maps ISBNs to book titles

**Key Notes:**

* Ratings range from 0 to 10
* Matrix is extremely sparse; efficient storage and retrieval are crucial
* LIBSVM format used to reduce memory footprint
* Placeholder target values (y=0) used only for saving LIBSVM, not for analysis

**Conclusion:**

This project demonstrates how collaborative filtering and sparse matrix modeling can effectively provide book recommendations on a scale. By leveraging similarities between users, the system can generate personalized suggestions even in the absence of explicit user profiles.

**Possible Improvements:**

* Include content-based filtering for hybrid recommendations
* Explore matrix factorization (e.g., SVD, ALS)
* Add user demographic or book genre metadata