INDIAN INSTITUTE OF TECHNOLOGY JODHPUR

NH 62 Nagaur Road, Karwar, Jodhpur, Rajasthan

MTech in Data Engineering – Semester 1

Machine Learning with Big Data

Project Report
On
Book Recommendation System

Submitted By:

Aneerban Chowdhury, M24DE3011

Pavani Racham, M24DE3059

Naresh Krishna Vemuri, M24DE3052

Book Recommendation System

Abstract

The Book Recommendation System aims to enhance user experience by providing personalized book recommendations based on user preferences and reading history. This system will leverage Google Cloud services such as **Google Cloud Storage (GCS), and Machine Learning models** to process and analyze large datasets efficiently. The recommendation engine will employ **collaborative filtering** and to generate accurate suggestions. The solution will also feature a user-friendly frontend for interaction and a backend for data processing and model inference.

List of Key Technology Challenges:

- **Data Collection & Storage:** Handling large volumes of book metadata, user interactions, and preferences efficiently in GCS.
- **Data Processing & Pipelines:** Creating scalable ETL pipelines for cleaning, transforming, and processing data.
- **Real-Time Processing:** Ensuring low-latency recommendations through optimized model execution and inference.
- **Scalability & Performance:** Managing high user traffic and large datasets without performance bottlenecks.
- **Frontend Integration:** Delivering seamless interaction between users and the recommendation engine.

Technology Stack:

- **Code Environment:** Python, Jupyter Notebook
- **UI**: Flask
- **Storage**: Google Cloud Storage
- **Machine Learning**: Scikit-Learn, Google AI Platform

List of Deliverables:

- **Data Ingestion & Processing Pipelines**: GCS for ingesting book data and user interactions.
- **Recommendation Engine**: Collaborative filtering for book recommendations.
- **User Interface**: Interactive frontend for users to browse and receive recommendations.
- **Performance Optimization**: Efficient model execution and inference for real-time suggestions.
- **Deployment & Scaling**: Fully deployed system on Google Cloud ensuring scalability and reliability.
- **Technical Documentation**: Comprehensive documentation for system architecture and ML models.

Data Sources:

- Kaggle Dataset - Collaborative Filtering Books Recommendation
- Size of Dataset & Attributes
    - Books 271360 rows X 8 columns – ISBN, Title, Author, Publication, Year of Publishing
    - Users 278858 rows X 3 columns – User ID, Location, Age
    - Ratings 1149780 rows X 3 columns – User ID, ISBN, Rating

Introduction

The Book Recommendation System is designed to deliver personalized book suggestions by analyzing user preferences and reading history. Leveraging Google Cloud technologies such as Google Cloud Storage, and the Google AI Platform, the system processes large-scale datasets efficiently. It combines collaborative filtering to enhance recommendation accuracy. The solution includes scalable ETL pipelines, a Flask-based interactive user interface, and a backend for real-time inference. Using a Kaggle dataset of books, users, and ratings, the system ensures high performance, scalability, and a seamless user experience, making it an effective tool for discovering books tailored to individual interests.

Problem Statement

1. **Data Management**:
   Efficiently managing and processing large-scale datasets comprising book metadata, user information, and rating histories is crucial. The system must handle both structured and semi-structured data formats while ensuring data consistency, accuracy, and security. Challenges include the ingestion of millions of records, reliable storage in Google Cloud Storage and fast retrieval for downstream analytics and recommendations.
2. **Analytics**:
   Extracting meaningful insights from user interaction data to understand reading habits, popular authors, and genre preferences is essential for improving engagement. The system will analyze trends in user ratings and behavior to support decision-making around content curation, targeted marketing, and user retention strategies.
3. **Prediction**:
   The system will implement machine learning models using collaborative filtering techniques to accurately predict book preferences for individual users. Leveraging historical data, the models aim to forecast future user interests, providing personalized suggestions and enhancing the discovery experience.
4. **Accessibility**:
   A user-friendly frontend built with Flask will offer seamless interaction between users and the recommendation engine. The interface will present recommendations clearly and intuitively, making it accessible for both technical and non-technical users. Interactive features will enable users to explore suggestions, provide feedback, and refine their preferences.
5. **Scalability**:
   The system architecture will be designed to scale effortlessly with growing datasets and user base. It will support high concurrency and performance by leveraging cloud-native solutions, ensuring real-time recommendation delivery without latency issues, even under increased load conditions.

Methodology

System Architecture

The project is built using a modular architecture that integrates the following components:

1. **Data Storage:** Google Cloud Storage for structured and unstructured data.
2. **Data Processing:** Python scripts for data cleaning, transformation, and loading (ETL).
3. **Machine Learning:** Scikit-learn for predictive analytics.
4. **Frontend:** Flask for web interface development.

Implementation Steps

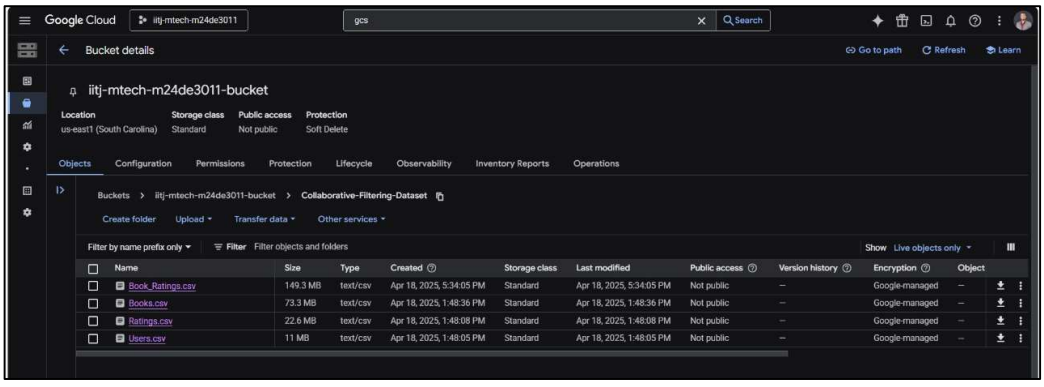
Step 1: Environment Setup

- Installed Python and essential libraries like Scikit Learn, Surprise, Numpy, Pandas, Pickle
- Set up GCP with appropriate permissions and billing.
- Configured Jupyter Notebook for development and debugging

Step 2: Data Design

Uploaded three main files to GCS. Data sourced from Kaggle

- Books 271360 rows X 8 columns – ISBN, Title, Author, Publication, Year of Publishing
- Users 278858 rows X 3 columns – User ID, Location, Age
- Ratings 1149780 rows X 3 columns – User ID, ISBN, Rating



Step 3: EDA and Save the Processed Dataset on GCS

- Did EDA to clean the data
- Created Books_Rating dataset by joining two datasets i.e. Books and Ratings.
- Save it to GCS to use it further.

```
In [19]: complete_df['Location'] = complete_df['Location'].str.split(',').str[-1].str.strip()

In [20]: complete_df.head()
```

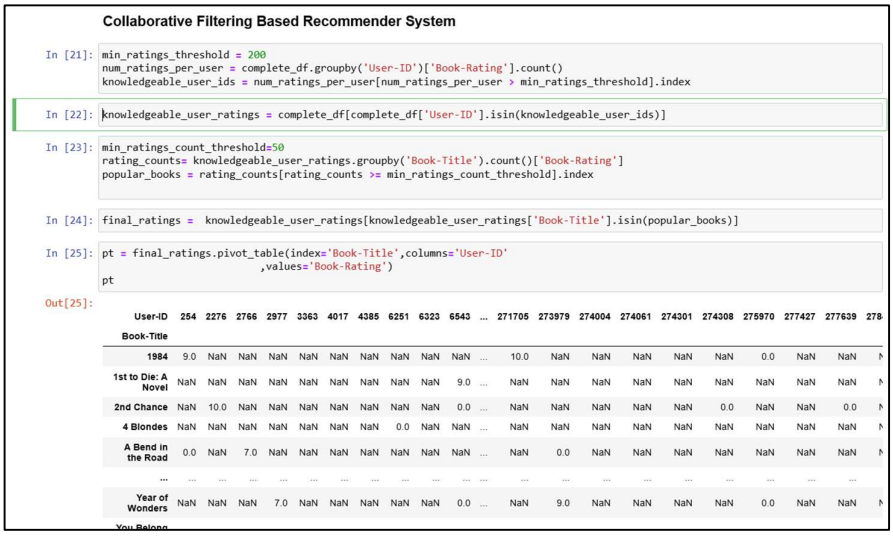
Out[20]:

	User-ID	Book-Rating	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-L	Location
0	276725	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	http://images.amazon.com/images/P/034545104X.0...	usa
1	276726	5	Rites of Passage	Judith Rae	2001	Heinle	http://images.amazon.com/images/P/0155061224.0...	usa
2	276727	0	The Notebook	Nicholas Sparks	1996	Warner Books	http://images.amazon.com/images/P/0446520802.0...	australia
3	276729	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	http://images.amazon.com/images/P/052165615X.0...	croatia
4	276729	6	The Amsterdam Connection : Level 4 (Cambridge ...	Sue Leather	2001	Cambridge University Press	http://images.amazon.com/images/P/0521795028.0...	croatia

```
In [45]: def upload_to_gcs(bucket_name, dataframe, destination_blob_name):
             client = storage.Client()
             bucket = client.bucket(bucket_name)
             blob = bucket.blob(destination_blob_name)

             if blob.exists():
                 print(f"File {destination_blob_name} already exists in bucket {bucket_name}. Deleting it.")
                 blob.delete()
                 print(f"File {destination_blob_name} deleted.")

             buffer = io.BytesIO()
             dataframe.to_csv(buffer, index=False)
             buffer.seek(0)

             blob.upload_from_file(buffer, content_type='text/csv')
             print(f"DataFrame uploaded to {bucket_name}/{destination_blob_name}.")

In [47]: upload_to_gcs(bucket_name, dataframe= complete_df, destination_blob_name="Collaborative-Filtering-Dataset/Book_Ratings.csv" )
         DataFrame uploaded to iitj-mtech-m24de3011-bucket/Collaborative-Filtering-Dataset/Book_Ratings.csv.
```

Step 4: Collaborative Filtering and SVD

- **Create User-Item Matrix**:
  Construct a matrix where rows represent users, columns represent books (ISBN), and cells contain the user's rating for that book.
- **Compute User Similarity**:
  Use cosine similarity or Pearson correlation to calculate similarity between users based on their rating vectors.
- **Predict Ratings**:
  Estimate ratings for unrated books by taking a weighted average of ratings from similar users.
- **Generate Recommendations**:
  Recommend the top-N highest predicted rated books that the user hasn't rated yet



Collaborative Filtering Based Recommender System

In [21]:
```
min_ratings_threshold = 200
num_ratings_per_user = complete_df.groupby('User-ID')['Book-Rating'].count()
knowledgeable_user_ids = num_ratings_per_user[num_ratings_per_user > min_ratings_threshold].index
```

In [22]:
```
knowledgeable_user_ratings = complete_df[complete_df['User-ID'].isin(knowledgeable_user_ids)]
```

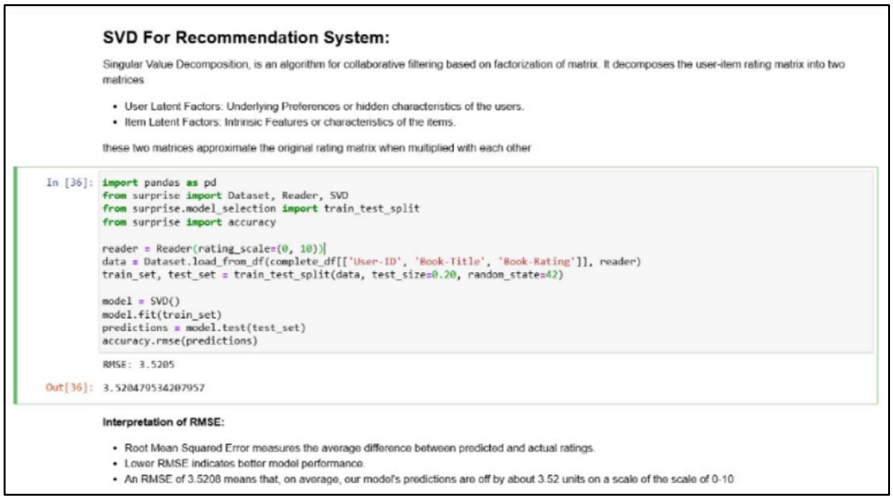
In [23]:
```
min_ratings_count_threshold=50
rating_counts= knowledgeable_user_ratings.groupby('Book-Title').count()['Book-Rating']
popular_books = rating_counts[rating_counts >= min_ratings_count_threshold].index
```

In [24]:
```
final_ratings =  knowledgeable_user_ratings[knowledgeable_user_ratings['Book-Title'].isin(popular_books)]
```

In [25]:
```
pt = final_ratings.pivot_table(index='Book-Title',columns='User-ID'
                               ,values='Book-Rating')
pt
```

Out[25]:

User-ID	254	2276	2766	2977	3363	4017	4385	6251	6323	6543	...	271705	273979	274004	274061	274301	274308	275970	277427	277639	278
Book-Title																					
1984	9.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	10.0	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	N
1st to Die: A Novel	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2nd Chance	NaN	10.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	...	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	0.0	N
4 Blondes	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
A Bend in the Road	0.0	NaN	7.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Year of Wonders	NaN	NaN	NaN	7.0	NaN	NaN	NaN	NaN	NaN	0.0	...	NaN	9.0	NaN	NaN	NaN	NaN	0.0	NaN	NaN	N
You Belong																					



SVD For Recommendation System:

Singular Value Decomposition, is an algorithm for collaborative filtering based on factorization of matrix. It decomposes the user-item rating matrix into two matrices

- User Latent Factors: Underlying Preferences or hidden characteristics of the users.
- Item Latent Factors: Intrinsic Features or characteristics of the items.

these two matrices approximate the original rating matrix when multiplied with each other

In [36]:
```
import pandas as pd
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from surprise import accuracy

reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(complete_df[['User-ID', 'Book-Title', 'Book-Rating']], reader)
train_set, test_set = train_test_split(data, test_size=0.20, random_state=42)

model = SVD()
model.fit(train_set)
predictions = model.test(test_set)
accuracy.rmse(predictions)
```

```
RMSE: 3.5205
```

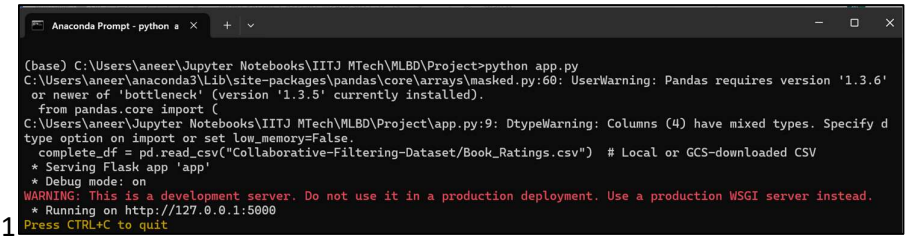
Out[36]: 3.520479534207957

Interpretation of RMSE:

- Root Mean Squared Error measures the average difference between predicted and actual ratings.
- Lower RMSE indicates better model performance.
- An RMSE of 3.5208 means that, on average, our model's predictions are off by about 3.52 units on a scale of the scale of 0-10

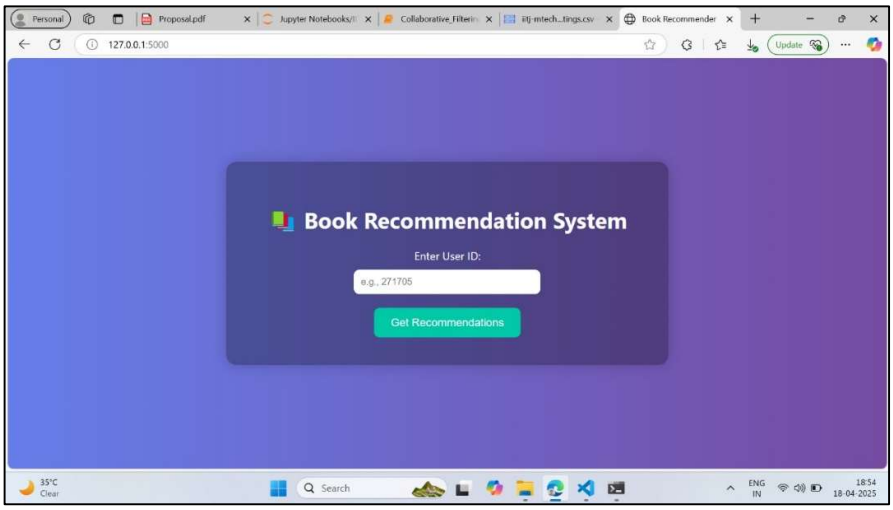
Step 5: Frontend using Flask

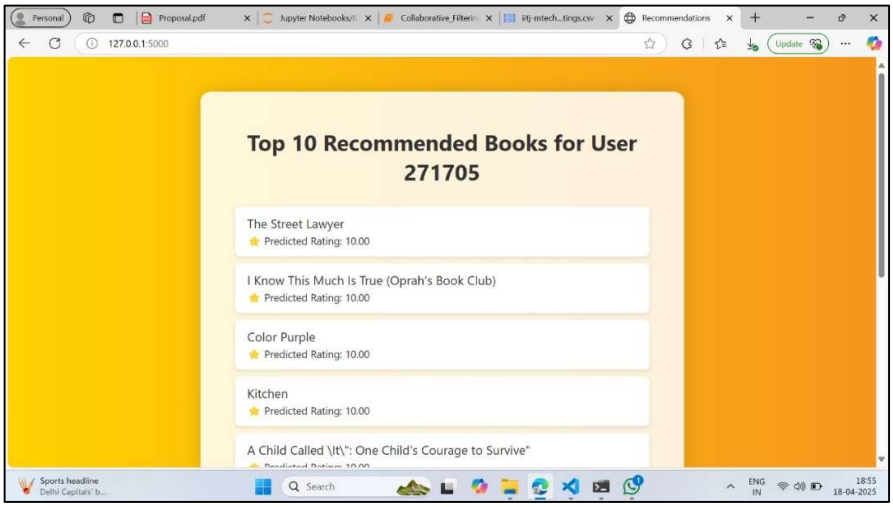
- Designed a basic yet functional HTML interface integrated with Flask templates.
- Displayed order summaries and visualizations dynamically.

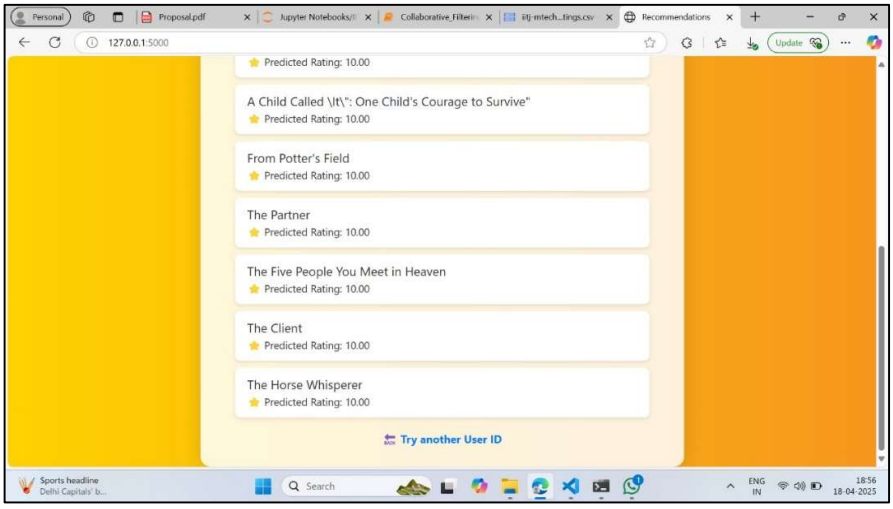


```
Anaconda Prompt - python a

(base) C:\Users\aneer\Jupyter Notebooks\IITJ MTech\MLBD\Project>python app.py
C:\Users\aneer\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires version '1.3.6'
 or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
C:\Users\aneer\Jupyter Notebooks\IITJ MTech\MLBD\Project\app.py:9: DtypeWarning: Columns (4) have mixed types. Specify d
type option on import or set low_memory=False.
  complete_df = pd.read_csv("Collaborative-Filtering-Dataset/Book_Ratings.csv")  # Local or GCS-downloaded CSV
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

1


Book Recommendation System
Enter User ID:
e.g., 271705
Get Recommendations


Top 10 Recommended Books for User 271705
The Street Lawyer
Predicted Rating: 10.00
I Know This Much Is True (Oprah's Book Club)
Predicted Rating: 10.00
Color Purple
Predicted Rating: 10.00
Kitchen
Predicted Rating: 10.00
A Child Called \It\": One Child's Courage to Survive"


Predicted Rating: 10.00
A Child Called \It\": One Child's Courage to Survive"
Predicted Rating: 10.00
From Potter's Field
Predicted Rating: 10.00
The Partner
Predicted Rating: 10.00
The Five People You Meet in Heaven
Predicted Rating: 10.00
The Client
Predicted Rating: 10.00
The Horse Whisperer
Predicted Rating: 10.00
Try another User ID

Results

Achievements

1. **Scalability:** Successfully stored and processed datasets on GCS.
2. **Efficiency:** Reduced data processing time using GCP tools.
3. **Predictive Insights:** Achieved acceptable accuracy in predicting order trends.
4. **Accessibility:** Enabled stakeholders to interact with the system through frontend.

Challenges

1. **Data Cleaning:** Ensuring data consistency during EDA.
2. **Visualization:** Balancing clarity and complexity in visual outputs using Flask.

Future Scope

1. **Hybrid Recommendation Models**: Combine collaborative filtering, content-based filtering, and deep learning-based approaches to improve personalization and accuracy.
2. **Real-Time Personalization**: Integrate real-time user interaction tracking to update recommendations dynamically using stream processing tools like Apache Kafka or Pub/Sub.
3. **User Profile Enrichment**: Include demographic data, reading goals, and sentiment analysis to build richer user profiles for more targeted suggestions.
4. **Mobile App Integration**: Extend the system to a mobile app using Flutter or React Native, enabling users to access personalized recommendations on the go.
5. **Cloud Optimization**: Optimize cost and performance by experimenting with serverless architectures, auto-scaling, and managed AI services on GCP.

Conclusion

The Book Recommendation System successfully demonstrates how user preferences and historical reading behavior can be leveraged to deliver personalized book suggestions using collaborative filtering techniques. By integrating Google Cloud services for scalable data storage, processing, and model deployment, the system ensures efficient handling of large datasets and real-time recommendations. A user-friendly interface enables seamless interaction, while machine learning models enhance the accuracy and relevance of suggestions.

Project Link: **https://github.com/aneerban10/mlbd-group-project-Aneerban-Pavani-Naresh**

References

1. Google Cloud Platform Documentation
2. Scikit-learn Reference
3. Flask Documentation