PROJECT REPORT

Book Recommender System

Course Code: CS4053

Course Title: Recommender Systems

Group Members:

Muhammad Ayyan Tahir	20K-0278
Syed Abdullah	20k-0275
Muhammad Anas Sheikh	20K-0179
Muhammad Warzan	20K-1649

National University of Computer and Emerging Sciences
Karachi, Pakistan

Introduction:

In today's digital age, where the abundance of choices can sometimes overwhelm users, recommender systems play a pivotal role in aiding decision-making processes. This project focuses on developing a sophisticated book recommender system which employs a hybrid approach by integrating two distinct recommendation techniques: content-based filtering and deep neural networks.The aim is to provide users with personalized recommendations based on their preferences and historical interactions with books.By combining these approaches, the recommender system aims to provide more accurate and diverse recommendations, enhancing the overall user experience, engagement, and retention, thereby benefiting both readers and book platforms alike.

Dataset:

The dataset used in this project was sourced from Kaggle and is titled "Goodreads Best Books." This dataset provides comprehensive information about a wide range of books, including their titles, authors, ratings, genres, and descriptions. The dataset consists of the following key attributes:

1. book_id: Unique identifier for each book.
2. book_title: Title of the book.
3. book_authors: Author(s) of the book.
4. book_desc: Description of the book, providing insights into its content and theme.
5. genres: Genres associated with the book, categorizing it into different literary categories.
6. book_format: Format of the book (e.g., paperback, hardcover, ebook).
7. book_rating: Average rating of the book on Goodreads.
8. book_rating_count: Number of ratings the book has received on Goodreads.
9. book_review_count: Number of reviews the book has received on Goodreads.

By utilizing this dataset, this project can analyze and suggest books based on factors like content similarity and popularity. This enables the system to offer personalized book recommendations aligned with users' preferences and interests, thereby enhancing their experience and engagement with the platform.

Data Preprocessing:

Data preprocessing was a crucial step in building a robust recommender system. It involved cleaning the dataset and preparing it for analysis. Various preprocessing steps were employed;

Cleaning the Dataset:
The dataset underwent cleaning to handle inconsistencies, missing values, or irrelevant information that could adversely affect the recommendation process. This involved: Handling Missing Values: Missing values, if any, were handled using the *dropna()* function, which removed rows containing missing values.

Textual Data Processing:
Textual attributes, such as book descriptions, genres, and authors, underwent processing to extract meaningful insights.

Text processing techniques included:

- Tokenization: Breaking down text into smaller units such as words or tokens.
- Removing Punctuation and Special Characters: Non-alphanumeric characters were removed to enhance text readability.
- Converting Text to Lowercase: Text was standardized by converting all characters to lowercase.

Model Explanation:

The project utilizes a Content-based filtering approach and DNN(Deep Neural Network) approach,both widely-used techniques in recommendation systems.

The content based approach suggests items similar to those the user has previously shown interest in. In the context of book recommendations, the system identifies books that are similar to those the user has read or interacted with before. This is achieved by analyzing textual attributes such as descriptions, genres, and authors.

Content Based:

Feature Extraction:

Feature extraction transformed raw textual data into numerical representations suitable for machine learning algorithms. Various techniques were employed, including Count Vectorization.

Textual Features Used:

1. Book Descriptions: The descriptions provided insights into the content and theme of each book, capturing its essence in textual form.

2. Genres: Genre information categorized books into different literary categories, allowing users to explore books based on their preferences.
3. Authors: Author information identified the creators of each book, providing insights into the writing style and body of work associated with each author.

Count Vectorization:

The CountVectorizer was configured to consider the top 5000 most frequent words and exclude common English stop words. Each book's textual attributes were represented as a numerical vector, where each element of the vector corresponded to the frequency of a particular word in the combined textual attributes (descriptions, genres, authors).

```
cv = CountVectorizer(max_features=5000, stop_words='english')
```

```
vector = cv.fit_transform(new_books['tags']).toarray()
```

```
vector
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

These numerical representations captured important features of the textual attributes, allowing for further analysis and comparison between books. By extracting and encoding these features, the system prepared the dataset for similarity calculation and recommendation generation, ultimately enhancing the accuracy and relevance of the recommendations provided to users.

Similarity Calculations:

Cosine Similarity Calculation

Once the textual features were represented as numerical vectors, cosine similarity was computed between every pair of books. Cosine similarity measured the cosine of the angle between two vectors and ranged from -1 to 1. A cosine similarity of 1 indicated that the vectors were identical, while a cosine similarity of -1 indicated complete dissimilarity.

Interpretation of Similarity Scores

The resulting similarity matrix contained similarity scores for every pair of books in the dataset. Higher similarity scores indicated that two books were more similar in content, while lower scores suggested dissimilarity. These similarity scores served as the basis for generating recommendations, as books with higher similarity scores were considered more relevant for recommendation to users who had shown interest in a particular book.

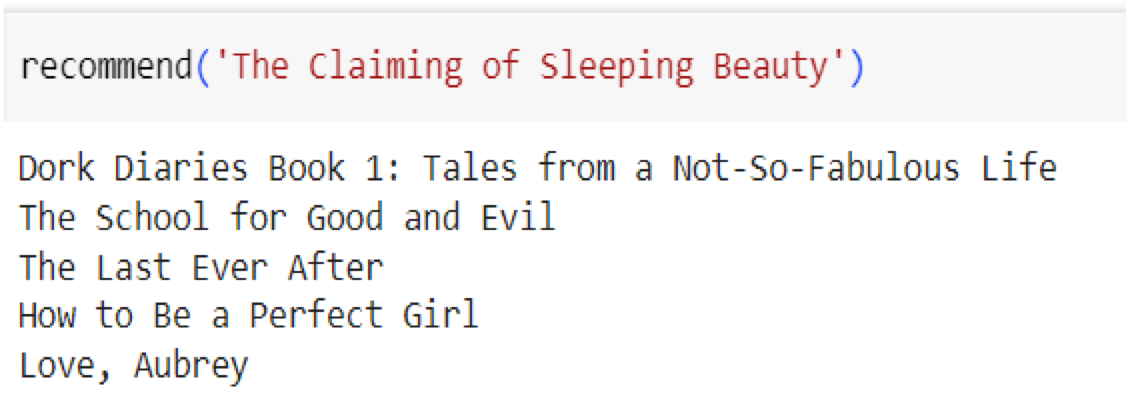
By computing cosine similarity between textual feature vectors, the system effectively identified books that shared similar content, enabling the generation of accurate and personalized recommendations for users based on their preferences and historical interactions with books.

<u>Recommendation Generation:</u>

Given an input book, the system retrieved the most similar books based on the computed similarities. These recommended books were then presented to the user, tailored to their specific interests and preferences. A function named recommend() was defined to generate recommendations based on a given input book:

This function retrieved the index of the input book, calculated the similarity scores between the input book and all other books, sorted them in descending order of similarity, and finally presented the top recommended books to the user.

By following this comprehensive pipeline, the recommender system effectively analyzed textual attributes, computed similarities, and generated personalized book recommendations tailored to the user's preferences.



```
recommend('The Claiming of Sleeping Beauty')
```

```
Dork Diaries Book 1: Tales from a Not-So-Fabulous Life
The School for Good and Evil
The Last Ever After
How to Be a Perfect Girl
Love, Aubrey
```

<u>DNNS:</u>

The model is trained using the provided dataset, with 90% of the data used for training and 10% for validation. The training process involves minimizing the mean squared error (MSE) loss between predicted and actual ratings. The model is trained for 20 epochs using the Adam optimizer.

Embedding Layers:

Embedding layers are pivotal in transforming categorical features such as user and book IDs into dense vectors in a latent space. These embeddings capture the latent characteristics of users and books, facilitating similarity calculations and enabling the model to capture intricate patterns in user preferences and book characteristics.

In the provided code snippet, the user and book IDs are passed through embedding layers, which convert them into dense vectors of size output_dim, representing their latent features in the model.

Dot Product:

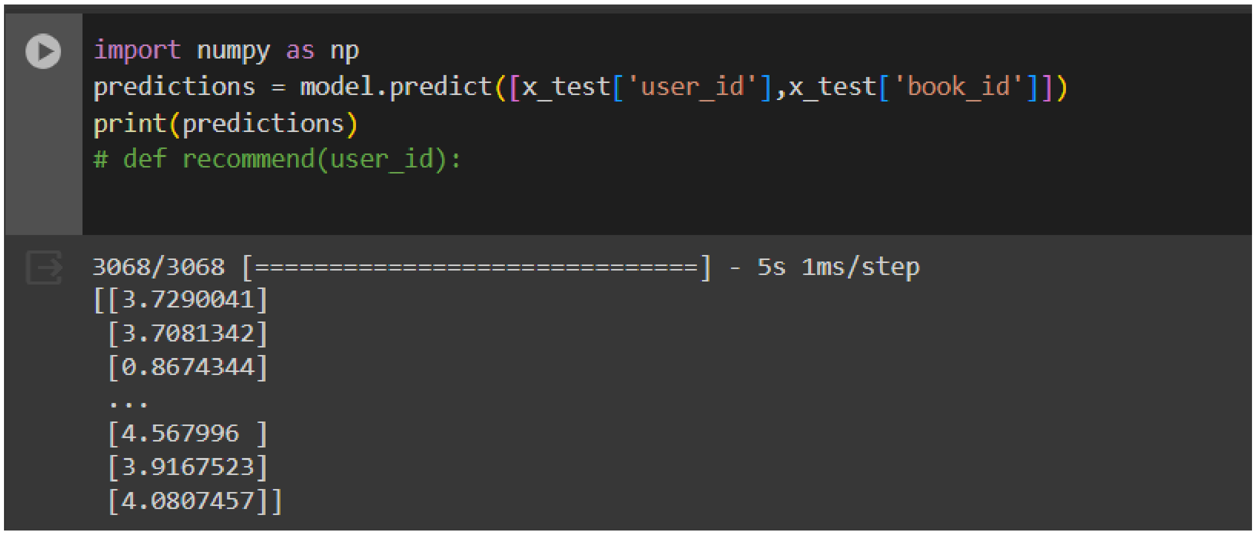
The dot product operation is fundamental in collaborative filtering models for predicting ratings or interactions between users and books. By computing the dot product between user and book embeddings, the model measures the similarity between them, enabling personalized recommendations based on similar user-book interactions.

In the code snippet above, the dot product is computed between the flattened user and book embeddings, resulting in a single scalar value representing the predicted rating or interaction between the user and the book.

By leveraging embedding layers and the dot product operation, the model effectively learns latent representations of users and books, enabling it to make personalized recommendations tailored to individual user preferences. These components contribute to the model's ability to capture complex user-book interactions and provide accurate recommendations in book recommendation systems.

<u>Rating Generation:</u>

The function takes the user_id and book_id as arguments and predicts the rating that the user will provide to the book.



```
import numpy as np
predictions = model.predict([x_test['user_id'],x_test['book_id']])
print(predictions)
# def recommend(user_id):
```

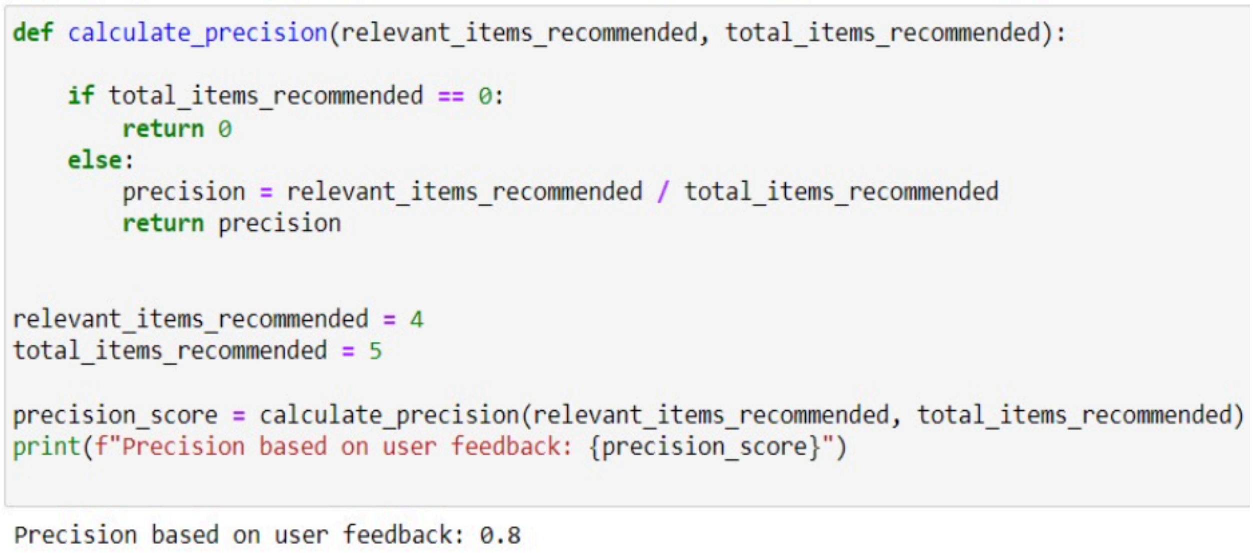
```
3068/3068 [==============================] - 5s 1ms/step
[[3.7290041]
 [3.7081342]
 [0.8674344]
 ...
 [4.567996 ]
 [3.9167523]
 [4.0807457]]
```

Results:

CNNS:

The calculate_precision function computes the precision score, which measures the proportion of relevant items recommended by the system among all items recommended. If no items are recommended, it returns 0 to avoid division by zero. Otherwise, it calculates precision by dividing the number of relevant items recommended by the total number of items recommended.

An example demonstrates its usage, where 4 out of 5 recommended items are deemed relevant by the user. This results in a precision score of 0.8.

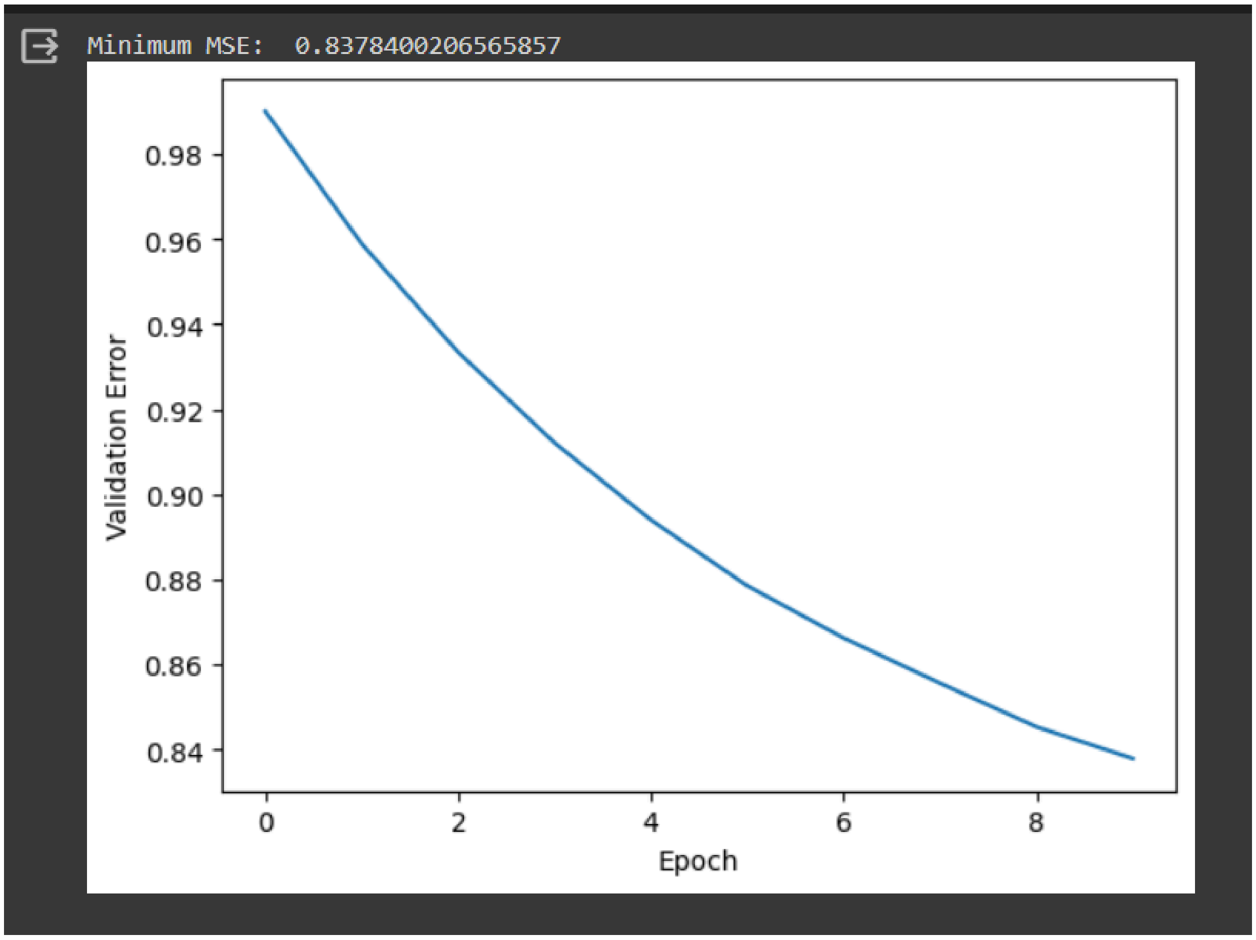


```
def calculate_precision(relevant_items_recommended, total_items_recommended):

    if total_items_recommended == 0:
        return 0
    else:
        precision = relevant_items_recommended / total_items_recommended
        return precision


relevant_items_recommended = 4
total_items_recommended = 5

precision_score = calculate_precision(relevant_items_recommended, total_items_recommended)
print(f"Precision based on user feedback: {precision_score}")
```

```
Precision based on user feedback: 0.8
```

DNN:

The plot shows how the validation error changes with each training epoch. As the model learns from more data, the validation error decreases, indicating improved performance in predicting user preferences for books. The minimum validation error achieved is 0.8378, suggesting that the deep neural network (DNN) model is effectively learning to recommend books to users.



```
Minimum MSE:  0.8378400206565857
```