PROJECT TITLE: A BOOK RECOMMENDER SYSTEM

Marwa Omar, Syed Zakriya Ibrahim
University of Engineering and Technology, Lahore (New Campus)
Department of Electrical Engineering and Technology

ABSTRACT

A book recommender system is a tool that helps users discover new books to read based on their personal preferences and reading history. It uses machine learning algorithms to analyze the characteristics of a book and the user's reading habits to make recommendations. The goal of a book recommender system is to provide personalized recommendations to users that are relevant, in order to help them find their next great read. There are several different approaches to building a book recommender system, including collaborative filtering, content-based filtering, and hybrid approaches that combine both techniques. Regardless of the approach used, the ultimate goal is to create a system that can accurately predict which books a user will enjoy, and provide recommendations accordingly.

1 INTRODUCTION

During the last few decades, with the rise of Youtube, Amazon, Netflix and many other such web services, recommender systems have taken more and more place in our lives. From e-commerce (suggest to buyers' articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys. In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users (items being movies to watch, text to read, products to buy).

By applying this simple dataset and related tasks and notebook, we will evolutionarily go through different paradigms of recommender algorithms. For each of them, we will present how they work, describe their theoretical basis and discuss their strengths and weaknesses.

The types of recommender system on which books can be recommended are:

- Popularity Based
- Content Based
- Collaborative Filtering based
- Hybrid Recommender System

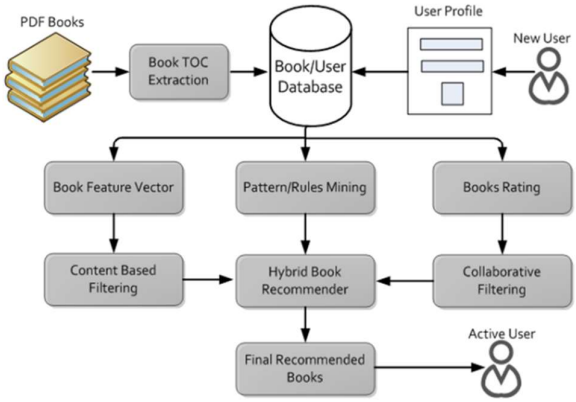

Figure 1: Basic Diagram of how Book Recommender System works

Popularity Based:
A popularity-based recommender system is a type of recommendation system that recommends items that are popular, or highly rated by other users. These systems do not take into account the preferences or characteristics of the individual user, but rather recommend items that have been well received by a large number of users. This can be effective for recommending items in a new product launch or for recommending items that have a broad appeal, but may not be as effective for recommending items that are more personalized to the individual user.

Content Based:
A content-based recommender system recommends items based on the characteristics of the items themselves. These systems analyze the characteristics of the items and use them to identify other items that are similar. For example, a content-based recommendation system for movies might recommend movies to a user based on the genres of movies that the user has previously watched.

Collaborative Filtering based:
A collaborative filtering-based book recommendation system is a type of recommendation system that uses the preferences and behaviors of a group of users to recommend items to an individual user. These systems analyze the ratings or other interactions that users have had with a particular item (such as reading or purchasing a book) and use this information to identify other users who have had similar interactions. The system can then use these similar users to make recommendations to the individual user.

Hybrid Recommender System:
A hybrid recommender system is a recommendation system that combines the strengths of multiple different types of recommendation systems in order to make more accurate and effective recommendations. Hybrid recommender systems can use a combination of techniques, such as collaborative filtering, content-based recommendation, and popularity-based recommendation, in order to make recommendations.
Here we are using **Popularity** and **Collaborative Filtering Based Book Recommender System** using two approaches:

- KNN (K-Nearest Neighbors)
- SVD (Singular Value Decomposition)

3 RELATED Work

Data Exploration:
The data set for our project is collected by the famous website named **Kaggle**. From here, we downloaded three csv files whose description is given below:

1. Users.csv File:
   Contains the users. Note that user IDs (`*User-ID*`) have been anonymized and map to integers. Demographic data is provided (`Location`, `Age`) if available. Otherwise, these fields contain NULL-values.
2. Books.csv File:
   Books are identified by their respective ISBN. Invalid ISBNs have already been removed from the dataset. Moreover, some content-based information is given (`Book-Title`, `Book-Author`, `Year-Of-Publication`, `Publisher`) obtained from Amazon Web Services. Note that in case of several authors, only the first is provided. URLs linking to cover images are also given, appearing in three different flavors (Image-URL-S, Image-URL-M, Image-URL-L`), i.e., small, medium, large. These URLs point to the Amazon web site.
3. Ratings.csv File:
   Contains the book rating information. Ratings (Book-Rating) are either explicit, expressed on a scale from 1-10 (higher values denoting higher appreciation), or implicit, expressed by 0.

Data Cleaning:

- We load the dataset into our program this can usually be done using a function or library specifically designed for reading in data such as:

```
In [1]: import numpy as np
        import pandas as pd

In [2]: books=pd.read_csv('Books.csv')
        users=pd.read_csv('Users.csv')
        ratings=pd.read_csv('Ratings.csv')
```

Figure 2: Importing Libraries and Loading datasets

- After that, we clean the data by removing any missing or incomplete values this can be done by dropping rows or columns with missing values or by imputing the missing values with estimates (e.g., the mean or median of the remaining values).

```
In [8]: users.isnull().sum()

Out[8]: User-ID          0
        Location         0
        Age         110762
        dtype: int64

In [9]: ratings.isnull().sum()

Out[9]: User-ID        0
        ISBN           0
        Book-Rating    0
        dtype: int64
```

Figure 3: Checking for null values

- Then we convert the categorical data into numerical form this can be done through techniques such as one-hot encoding or label encoding.
- We extract user-id column from Users and ISBN column from Books and then make a new excel file with name Books-Ratings.

```
In [5]: ratings.head()
```

Out[5]:

	User-ID	ISBN	Book-Rating
0	276725	034545104X	0
1	276726	0155061224	5
2	276727	0446520802	0
3	276729	052165615X	3
4	276729	0521795028	6

Figure 4: Checking Ratings Data Set

- Then we merge rating table with book table using 'ISBN' to include book titles instead of 'ISBN' for interpretability.
- Then we normalize or standardize the data to ensure that all features are on the same scale this can be done by subtracting the mean and dividing by the standard deviation for each feature.
- Then we do split the dataset into training and test sets. This is typically done by randomly selecting a portion of the data for the training set and using the remaining data for the test set.
- Check for any imbalanced classes and handle them as needed (e.g., through under sampling or oversampling).

```
In [6]: print(books.shape)
        print(users.shape)
        print(ratings.shape)

        (271360, 8)
        (278858, 3)
        (1149780, 3)
```

Figure 5: Checking Shape of all Data Set

Popularity Based Filtering Book Recommender System:

In popularity-based Book RS, the books are sorted on the basis of their popularity among users. We designed an algorithm in which the books having rating >250 are sorted and then we show the top 50 out of them as:

```
In [18]: popular_df = popular_df[popular_df['num_ratings']>=250].sort_values('avg_rating',ascending=False).head(50)
```

Figure 6: Algorithm for Popularity based Book RS

On printing the data frame, containing the top 50 book we get:

```
In [21]: popular_df
```

Out[21]:

	Book-Title	Book-Author	Image-URL-M	num_ratings	avg_rating
0	Harry Potter and the Prisoner of Azkaban (Book 3)	J. K. Rowling	http://images.amazon.com/images/P/0439136350.0...	428	5.852804
3	Harry Potter and the Goblet of Fire (Book 4)	J. K. Rowling	http://images.amazon.com/images/P/0439139597.0...	387	5.824289
5	Harry Potter and the Sorcerer's Stone (Book 1)	J. K. Rowling	http://images.amazon.com/images/P/0590353403.0...	278	5.737410
9	Harry Potter and the Order of the Phoenix (Boo...	J. K. Rowling	http://images.amazon.com/images/P/043935806X.0...	347	5.501441
13	Harry Potter and the Chamber of Secrets (Book 2)	J. K. Rowling	http://images.amazon.com/images/P/0439064872.0...	556	5.183453
16	The Hobbit : The Enchanting Prelude to The Lor...	J.R.R. TOLKIEN	http://images.amazon.com/images/P/0345339681.0...	281	5.007117

Figure 7: Showing the Data frame

Collaborative Based Filtering Book Recommender System:

In this type of filtering, we first make a sparse matrix. This matrix has number of user-id in the columns position and number of book-title in the index position by using a command of **pivot_table** as shown:

```
final_ratings = filtered_rating[filtered_rating['Book-Title'].isin(famous_books)]
pt = final_ratings.pivot_table(index='Book-Title',columns='User-ID',values='Book-Rating')
pt.fillna(0,inplace=True)
pt
```

Out[27]:

User-ID	254	2276	2766	2977	3363	4017	4385	6251	6323	6543	...	271705	273979	274004	274061	274301	274308	275970	277427	277639	278418
Book-Title																					
1984	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1st to Die: A Novel	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2nd Chance	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 8: Showing the PivotTable

KNN-Method: In KNN-Method we designed an algorithm such as shown. Here we make a function named **recommend_books** in which book name is entered by user and its nearest or you can say the closet books are popped up.

```
In [32]: model = NearestNeighbors(n_neighbors=5, algorithm='brute')

         # Fit the model to the data
         model.fit(book_sparse)

         # Function to recommend books
         def recommend_books(book_name):
             # Find the index of the book in the data
             idx = np.where(pt.index == book_name)[0][0]

             # Find the distances and indices of the nearest neighbors
             distances, suggestions = model.kneighbors(pt.iloc[idx, :].values.reshape(1, -1))

             # Recommend the most similar books
             for i in range(len(suggestions)):
                 if i==0:
                     print("The suggestion for", book_name, "are :")
                 if not i:
                     print(pt.index[suggestions[i]])
```

Figure 9: Algorithm for KNN-Method

On testing upon the upon the above function, the output we get is as follows. As we set the **n_neighbors** value to be **5**. The function will show us the closet and relatable 5 books related to user specification.

```
In [33]: recommend_books("Animal Farm")

The suggestion for Animal Farm are :
Index(['Animal Farm', 'Exclusive', 'Hearts in Atlantis', 'Jacob Have I Loved',
       'Second Nature'],
      dtype='object', name='Book-Title')
```

Figure 10: Output of KNN-Method

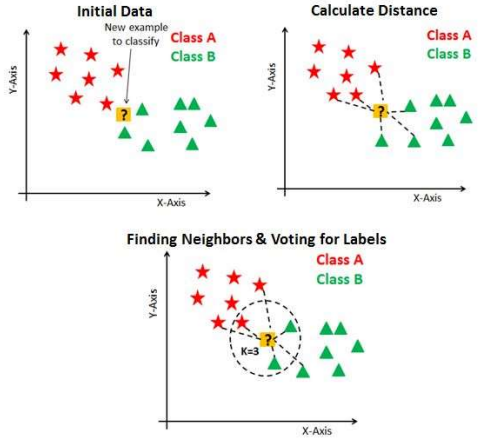

Figure 11: Working of KNN-Method

SVD-Method: In SVD Method we designed an algorithm such as shown. Here we make a function named **recommend** in which user-id and book name is entered by user and its nearest or you can say the closet books are popped up.

```
In [75]: from scipy.sparse.linalg import svds


         # Perform SVD decomposition and make recommendations
         def recommend(user_id,book_name):
             # Perform SVD decomposition
             U, sigma, Vt = svds(pt, k=5)

             # Construct the diagonal matrix of singular values
             sigma = np.diag(sigma)

             # Make predictions for all books
             all_predictions = np.dot(np.dot(U, sigma), Vt)

             # Select the predictions for the user we are interested in
             user_predictions = all_predictions[user_id, :]

             # Sort the predictions in descending order
             sorted_predictions = np.flip(np.argsort(user_predictions))

             # Select the top-k recommendations
             top_k = sorted_predictions[:num_recommendations]
```

Figure 12: Algorithm for SVD-Method

On testing upon the upon the above function, the output we get is as follows. The function will show us the closet and relatable 5 books related to user specification.

```
: recommend_books( 254, "Animal Farm")

The suggestion for Animal Farm are :
Index(['Animal Farm', 'Exclusive', 'Hearts in Atlantis', 'Jacob Have I Loved',
       'Second Nature'],
      dtype='object', name='Book-Title')
```

Figure 13: Output of SVD-Method

4 FIGURES AND DIAGRAMS

As we are using two machine learning (ML) algorithms to implement this book recommender system and they are:

- KNN (K-Nearest Neighbors)
- SVD (Singular Value Decomposition)

Popularity based book recommender system:

We assigned each book title a number ranging from 1-51 as we are showing only top 50 books in order to check through the plot that which one is the top most popular book among all.

```
In [24]: popular_df
```

Out[24]:

	number_assinged_to_book	Book-Title	Book-Author	Image-URL-M	num_ratings	avg_rating
0	1	Harry Potter and the Prisoner of Azkaban (Book 3)	J. K. Rowling	http://images.amazon.com/images/P/0439136350.0...	428	5.852804
3	2	Harry Potter and the Goblet of Fire (Book 4)	J. K. Rowling	http://images.amazon.com/images/P/0439139597.0...	387	5.824289

Figure 14: Showing Data Frame

Bar plot: The bar plot between the book title (number assigned to books) and their ratings is as follows:

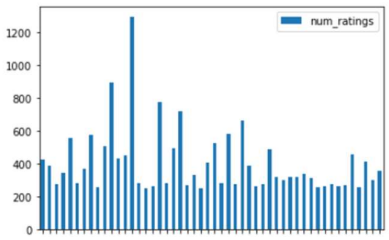

Figure 15: Bar Plot of Popular Book RS

Line plot: The line plot between the book title (number assigned to books) and their ratings is as follows:

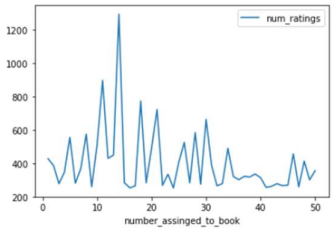

Figure 16: Line Plot of Popular Book RS

KNN (K-Nearest Neighbors): To make recommendations using the KNN method, the model first needs to determine the similarity between users. This is typically done using a distance measure such as Euclidean distance or Pearson correlation coefficient. The model then selects the K users who are most similar to the target user, based on the chosen distance measure. Finally, the model recommends books to the target user that have been rated highly by these K nearest neighbors.

```
In [32]: model = NearestNeighbors(n_neighbors=5, algorithm='brute')

         # Fit the model to the data
         model.fit(book_sparse)

         # Function to recommend books
         def recommend_books(book_name):
             # Find the index of the book in the data
             idx = np.where(pt.index == book_name)[0][0]

             # Find the distances and indices of the nearest neighbors
             distances, suggestions = model.kneighbors(pt.iloc[idx, :].values.reshape(1, -1))

             # Recommend the most similar books
             for i in range(len(suggestions)):
                 if i==0:
                     print("The suggestion for", book_name, "are :")
                 if not i:
                     print(pt.index[suggestions[i]])
```

Figure 17: Algo for KNN-Method

Output:

```
In [33]: recommend_books("Animal Farm")

The suggestion for Animal Farm are :
Index(['Animal Farm', 'Exclusive', 'Hearts in Atlantis', 'Jacob Have I Loved',
       'Second Nature'],
      dtype='object', name='Book-Title')
```

Figure 16: Output

SVD (Singular Value Decomposition): In a recommender system, singular value decomposition (SVD) is a technique that can be used to factorize a matrix of user-item ratings into a product of matrices. This can be useful for a variety of tasks in a recommender system, such as data compression, dimensionality reduction, and making predictions about user ratings for items.

```
In [75]: from scipy.sparse.linalg import svds


         # Perform SVD decomposition and make recommendations
         def recommend(user_id,book_name):
             # Perform SVD decomposition
             U, sigma, Vt = svds(pt, k=5)

             # Construct the diagonal matrix of singular values
             sigma = np.diag(sigma)

             # Make predictions for all books
             all_predictions = np.dot(np.dot(U, sigma), Vt)

             # Select the predictions for the user we are interested in
             user_predictions = all_predictions[user_id, :]

             # Sort the predictions in descending order
             sorted_predictions = np.flip(np.argsort(user_predictions))

             # Select the top-k recommendations
             top_k = sorted_predictions[:num_recommendations]
```

Figure 18: Algo for SVD-Method

Output:

```
: recommend_books( 254, "Animal Farm")

The suggestion for Animal Farm are :
Index(['Animal Farm', 'Exclusive', 'Hearts in Atlantis', 'Jacob Have I Loved',
       'Second Nature'],
      dtype='object', name='Book-Title')
```

Figure 19: Output

Exploratory data analysis:

Ratings are of two types, an implicit rating & explicit rating. An implicit rating is based on tracking user interaction with an item such as a user clicking on an item '0'. An explicit rating is when a user explicitly rates an item, i.e., b/w '1-10' • Majority of ratings are implicit i.e., rating '0' • Rating of '8' has the highest rating count among explicit ratings '1-10'.

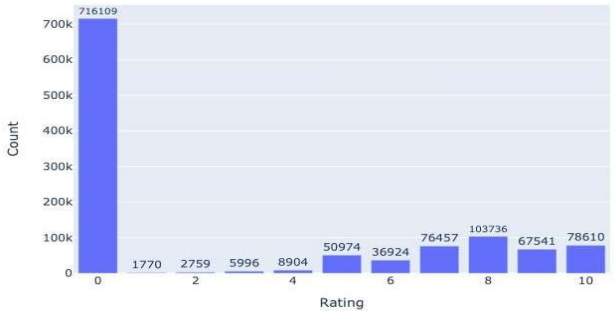

Figure 20: Count Ratings plot

Rating by User (>250): There is inherent bias in the dataset. There are few users who rate a lot & several users that provide very few ratings. One user has provided 13K+ ratings, only ~700 users (out of 100K+ users) have provided over 250 ratings.

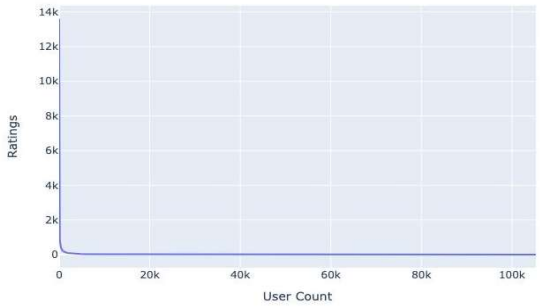

Figure 21: User Count plot

Rating received per book (>50): A similar bias is observed. There are a few books that have received many ratings and several books that have received very few ratings. One book has received over 2500 ratings, only ~2100 books (out of 300K+ books) have received more than 50 ratings.

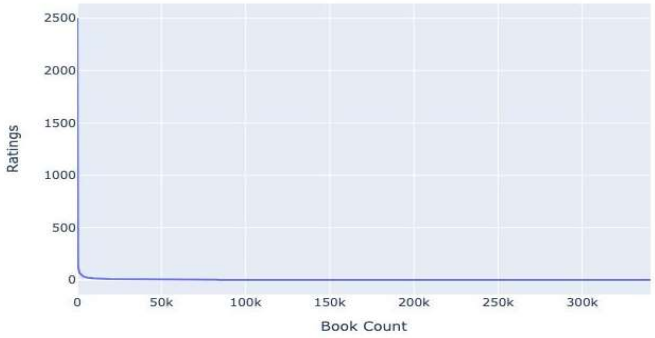

Figure 22: Book Counts plot

5 COMPARISON AND DEMONSTRATION:

With KNN-method: By using KNN-method the score accuracy calculated is as:

Unbiased Testing Performance:	
MAE	2.3052
RSME	3.2579

Table 1: Results of KNN-Algo

Output:

```
In [33]: recommend_books("Animal Farm")

         The suggestion for Animal Farm are :
         Index(['Animal Farm', 'Exclusive', 'Hearts in Atlantis', 'Jacob Have I Loved',
                'Second Nature'],
               dtype='object', name='Book-Title')
```

Figure 23: Output

With SVD-method: By using SVD-method the score accuracy calculated is as:

Unbiased Testing Performance:	
MAE	2.3800
RSME	3.1341

Table 2: Results of SVD-Algo

Output:

```
: recommend_books( 254, "Animal Farm")

  The suggestion for Animal Farm are :
  Index(['Animal Farm', 'Exclusive', 'Hearts in Atlantis', 'Jacob Have I Loved',
         'Second Nature'],
        dtype='object', name='Book-Title')
```

Figure 24: Output

Comparison Table:

	MAE	RMSE	fit_time	test_time
knns.KNNWithMeans	2.351416	3.289229	0.131446	1.228990
knns.KNNBaseline	2.356247	3.299194	0.148220	1.486081
matrix_factorization.SVD	2.392815	3.313362	5.206008	0.194471

Figure 25: Comparison Table

The outputs (above & below) shows the top 10 recommendations for the user 13552 using KNNWithMeans & SVD Both algorithms recommended instances of Harry Potter novels for user 13552. Additionally, the recommended books seem to be a similar genre lending confidence in interpretability of recommendations

Demonstration:

Popularity based:

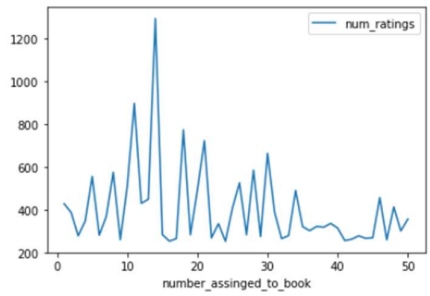

Figure 26: Demonstration of Popularity based book recommender system

KNN method: It has the advantage of being simple to implement and easy to understand. However, it can be computationally expensive, as the model needs to compute the similarity between the target user and all other users in the system in order to make recommendations. Additionally, the quality of the recommendations produced by the KNN method can be sensitive to the choice of the value of K.

```
In [33]: recommend_books("Animal Farm")

         The suggestion for Animal Farm are :
         Index(['Animal Farm', 'Exclusive', 'Hearts in Atlantis', 'Jacob Have I Loved',
                'Second Nature'],
               dtype='object', name='Book-Title')
```

Figure 27: Demonstration of KNN-Method

SVD: This works by decomposing the user-item matrix into three matrices: a user matrix, a diagonal matrix, and an item matrix. These matrices can be used to reconstruct the original matrix, but with fewer dimensions. This can be useful for reducing the complexity of the data and improving the efficiency of the recommender system. SVD is a powerful tool that has been widely used in recommendation systems, but it can be computationally expensive and may not work well with very large datasets. There are also other techniques that can be used for recommendation systems, such as collaborative filtering and matrix factorization.

```
: recommend_books( 254, "Animal Farm")

  The suggestion for Animal Farm are :
  Index(['Animal Farm', 'Exclusive', 'Hearts in Atlantis', 'Jacob Have I Loved',
         'Second Nature'],
        dtype='object', name='Book-Title')
```

Figure 28: Demonstration of SVD-Method

REFERENCES

[1]. https://www.analyticsvidhya.com/blog/2021/06/build-book-recommendation-system-unsupervised-learning-project/
[2]. https://thecleverprogrammer.com/2021/01/17/book-recommendation-system/
[3]. https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset
[4]. https://nevonprojects.com/online-book-recommendation-using-collaborative-filtering/
[5]. https://ieeexplore.ieee.org/document/7416895
[6]. https://projectsgeek.com/2018/07/online-book-recommendation-system-project.html
[7]. https://t4tutorials.com/book-recommendation-system-using-collaborative-filtering-project-in-php-or-asp-net/
[8]. https://pub.towardsai.net/step-by-step-approach-to-building-a-recommendation-system-a65be5a54045?gi=0a0b3c1561c7
[9]. https://data-flair.training/blogs/book-recommendation-system-machine-learning-project/
[10]. https://www.geeksforgeeks.org/singular-value-decomposition-svd/
[11]. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning