HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

ARTIFICIAL INTELLIGENT PROJECT REPORT

INTRODUCTION TO ARTIFICIAL INTELLIGENT
(IT3160E)

Book Recommendation System Using Collaborative Filtering

Team members	*Email*
Vu Duc An 20215174	an.vd215174@sis.hust.edu.vn
Le Ha Ngan 20215230	ngan.lh215230@sis.hust.edu.vn
Dang Viet Hoang 20215206	hoang.dv215206@sis.hust.edu.vn
Trinh Diem Quynh 20210737	quynh.td210737@sis.hust.edu.vn

Class
136462 - IT3160E
Lecturer
Nguyen Nhat Quang
quang.nguyennhat@hust.edu.vn
quangnn@soict.hust.edu.vn

January 3, 2023

Time period: 1^{st}Nov - 15^{th}Dec

AI Project Report

Book Recommendation System Using Collaborative Filtering

Class: 136462

Lecturer: Nguyen Nhat Quang

Contents

I Introduction

Virtually everyone has had an online experience where a website makes personalised recommendations based on past activities. Amazon tells you "Customers Who Bought This Item Also Bought", Udemy tells you "Students Who Viewed This Course Also Viewed". And Netflix awarded a $1 million prize to a developer team in 2009, for an algorithm that increased the accuracy of the company's recommendation system by 10 percent.

Recommendation systems have become an essential tool for businesses and organizations looking to provide a personalized and relevant experience for their users. By analyzing the preferences and behaviors of individual users, recommendation systems are able to make tailored suggestions for products, services, or content that align with their interests. This not only enhances the user experience, but it also has the potential to drive sales and increase customer loyalty.

With the vast amount of information and options available online, recommendation systems play a crucial role in helping users sift through the noise and find what they are looking for. As such, the implementation of a recommendation system has become a key strategy for many companies looking to stay competitive in today's digital landscape. Building such system today may requires specialised expertise in analytics, machine learning and software engineering. In this report, we are going to explain how to build a Recommendation System using Collaborative Filtering technique.

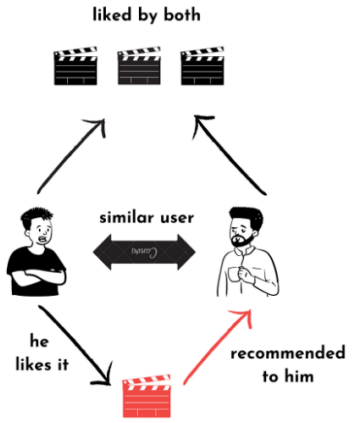
This project aims to build a recommendation system using collaborative filtering and memory-based algorithms. Collaborative filtering is a method that suggests items to users based on the idea that individuals with similar tastes and preferences will have similar ratings for items. Our system used ratings provided by users to identify other users with similar ratings, and then predict the ratings that a chosen user would give to items they had not yet rated. We evaluated the performance of our system using various metrics and found that it was able to provide reliable recommendations for users.

II Method

II.1 Collaborative-based Filtering

Collaborative based filtering recommender systems are based solely on past interactions of users and target items. The motivation for collaborative filtering comes from the idea that people often get the best recommendations from someone with tastes similar to theirs.

In simple words here, we try to search for the look-alike customers and offer products based on what his or her lookalike has chosen. Let us understand with an example. X and Y are two similar users: user X has watched movie A, B, and C; user Y has watched movie B, C, and D. Then we will recommend movie A to user Y and movie D to user X.



The system does not take into consideration an item's features like author, publisher, genre etc nor a user's features like age, gender, location etc. This type of recommendation system can take either a memory based approach or a model based approach.

- **Memory based approach:** Utilises entire user-item rating information to calculate similarity scores between items or users for making recommendations. The two approaches are mathematically quite similar, but there is a conceptual difference between the two:
  - **User based:** Two users are considered similar, if they rate items in a similar manner. An item is recommended to a user, if another user i.e., similar to the user in question has liked the item.
  - **Item based:** Two items are considered similar, if users rate them in a similar manner. An item is recommended to a user, that is similar to the items the user has rated in the past.
- **Model based approach:** Utilises user-item rating information to build a model and the model (not the entire dataset) is thereafter used for making recommendations. This approach is preferred in instances where time and scalability are a concern.

In this project, we implemented a recommendation system using Memory-based Collaborative Filtering.

II.2 User-user Collaborative Filtering

II.2.1 Normalizations

One of the problems faced with using ratings as a means of representing taste, is that each user has personal interpretation of the scale. While one rater might tend to give high marks to films he likes, another rater might keep the highest grades for exceptional movies. The question then is how similar both users are, and how to user their ratings appropriately.

There are two widely used systems to compensate for variations in the rating approach by different users, one is called mean-centering, and the other is called Z-score.

The mean-centering algorithm re-maps a user's rating by subtracting the mean value of all his ratings, effectively signalling if the particular rating is positive or negative when compared to the mean. Positive values represent above-average ratings, negative results represent below-average ratings, and zero represents an average rating. The formulation is given by

$$h(r_{ui}) = r_{ui} - \bar{r}_u \tag{1}$$

The Z-score approach is very similar to the mean-centering, but it is normalized by the standard deviation σ_u or σ_i depending on whether we apply the User or the Item based approach.

$$h(r_{ui}) = \frac{r_{ui} - \bar{r}_u}{\sigma_u} \tag{2}$$

Normalization of ratings does come with a word of caution. If a user only gives high ratings to all items reviewed, the mean-centered approach will yield them average, and any rating below the highest would be a negative one, even if it is fairly high in the scale. Likewise, if a user has rated all items with the same exact grade, the standard deviation will be zero, and the Z-score won't be computable.

II.2.2 Similarity metric

At the heart of the Collaborative Filtering algorithm lies the strong assumption that "similar users like similar items". While this assumption appears to hold true to a great extent, the key question becomes: how do we define and identify similar users in order to find similar candidate items we can recommend? Any such formulation needs to consider the data we have available, a sparse matrix of User-Item Preferences, where many users have likely only rated a small proportion of the many items in stock.

The only data we have is the Utility matrix Y so this similarity must be chosen based on the columns corresponding to the two users in the matrix. Consider the below example of an utility matrix based on the user ratings for an item.

The goal of a recommendation system is to predict the blanks in the utility matrix.

Obviously, user u_0 is more similar to user u_1 than users u_2, u_3, u_4, u_5, u_6, which means we can predict that u_0 will be interested in item i_2 because user u_1 rated this item 4. However, they rated the same item i_4 with different scores. So how can we measure their similarity?

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	?	?
i_1	3	?	?	0	?	?	?
i_2	?	4	1	?	?	1	2
i_3	2	2	3	4	4	?	4
i_4	2	0	4	?	?	?	5

Figure 1: Utility Matrix

- Cosine Distance

  Cosine similarity means the similarity between two vectors of inner product space. It is measured by the cosine of the angle between two vectors.

  We can treat blanks as a 0 value. This choice is questionable, since it has the effect of treating the lack of a rating as more similar to disliking the movie than liking it. If we normalise ratings by subtracting from each rating the average rating of that user, we turn low ratings into negative numbers and high ratings into positive numbers. If we then take the cosine distance, we find that users with opposite views of the items they have in common will have vectors in almost opposite directions, and can be considered as far apart as possible. However, users with similar opinions about the items rated in common will have a relatively small angle between them

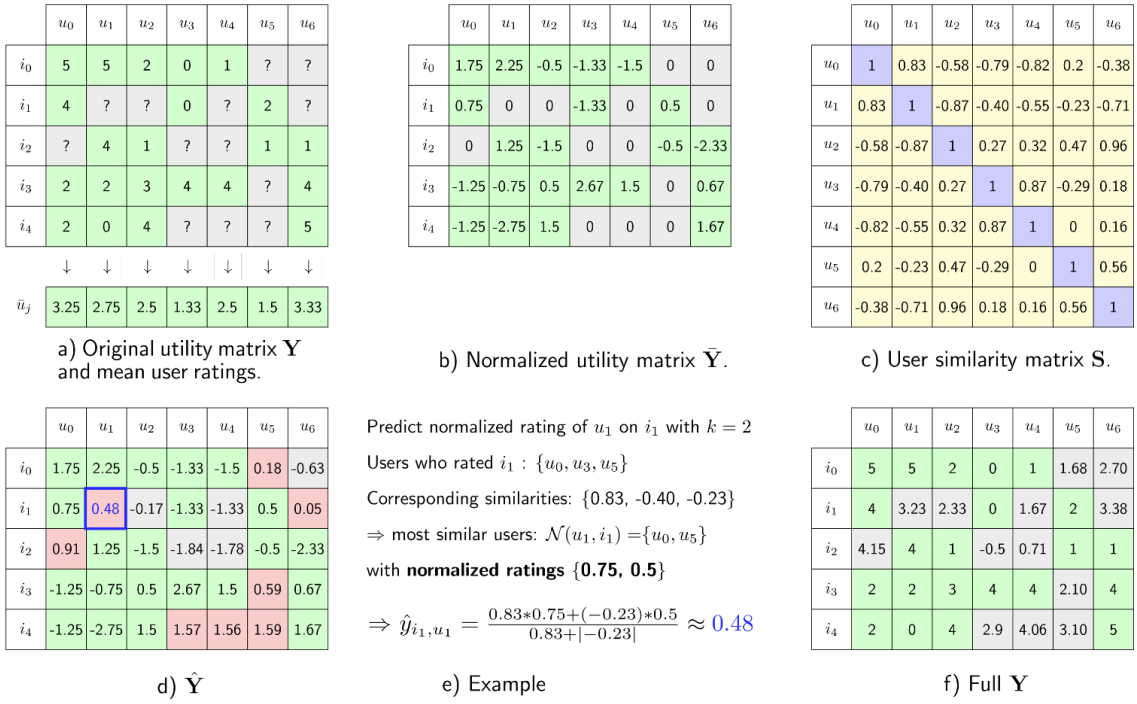


Figure 2

From the utility matrix, we find that u_0 is closer to u_1 and u_5 than the rest. This observation makes intuitive sense, given that u_0 and u_1 have a penchant for items i_0, i_1, i_2, while u_0 and u_5 give all the items positive scores.

When using the Cosine Distance, the similarity is represented by a scale of -1 to +1 where a positive high value suggests a high correlation, a negative high value suggests inversely high correlation (when one says True the other says False), and lastly a zero correlation indicates uncorrelated samples.

In vector form, the Cosine Distance is defined as:

$$\cos(\vec{a}, \vec{b}) = \frac{\vec{a}.\vec{b}}{||\vec{a}||^2 * ||\vec{b}||^2} \tag{3}$$

The User-Based Cosine similarity equation, defined as a summation, becomes:

$$cosine\_sim(u, v) = \frac{\sum_{i \in I_{uv}} r_{ui}.r_{vi}}{\sqrt{\sum_{i \in I_{uv}} r_{ui}^2}.\sqrt{\sum_{i \in I_{uv}} r_{vi}^2}} \tag{4}$$

- Mean Squared Difference

  The Mean Squared Distance similarity is defined as:

$$msd(u, v) = \frac{1}{|I_{uv}|}. \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2 \tag{5}$$

- Pearson Correlation

  When using the Pearson Correlation, the similarity is represented by a scale of -1 to +1 where a positive high value suggests a high correlation, a negative high value suggests inversely high correlation (when one says True the other says False), and lastly a zero correlation indicates uncorrelated samples.

  The User-Based Pearson Correlation similarity equation is defined as:

$$pearson\_sim(u, v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u).(r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2}.\sqrt{\sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}} \tag{6}$$

  Where $pearson\_sim(u, v)$ represents the similarity of users u and v, I_{uv} represents the set of items rated by both users u and v, r_{ui} and r_{vi} represent the rating of user u or v for item i, and $\bar{r}_u$ and $\bar{r}_v$ represent the mean rating of user u or v, across all items rated.

  This similarity algorithm approaches by mean-centering, and computing cosine distance (angle) between 2 vector.

- Pearson correlation using baseline

  Compute the Pearson correlation coefficient between all pairs of users (or items) using baselines for centering instead of means.

$$pearson\_baseline\_sim(u, v) = \hat{}\rho_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - b_{ui}).(r_{vi} - b_{vi})}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - b_{ui})^2}.\sqrt{\sum_{i \in I_{uv}} (r_{vi} - b_{vi})^2}} \tag{7}$$

  Typical Collaborative Filtering data exhibit large user and item effects—systematic tendencies for some users to give higher ratings than others—and for some items to receive higher ratings than others. It is customary to adjust the data by accounting for these effects, which we encapsulate within the baseline estimates. Denote by μ the overall average rating. A baseline estimate for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i \tag{8}$$

The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. For example, suppose that we want a baseline estimate for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies, μ, is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline estimate for Titanic's rating by Joe would be 3.9 stars by calculating 3.7 - 0.3 + 0.5.

- Euclidean Distance

  The Euclidean Distance similarity is defined as:

$$s(u,v) = \sqrt{\frac{\sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2}{|I_{uv}|}} \tag{9}$$

II.2.3 Rating prediction

Determining a user's interest in an item based on these nearest users is very similar to what we see in K-nearest neighbours (KNN). When working with large-scale problems, we will further see that the method KNN is used a lot because of its simplicity. Of course, we cannot directly use KNN, but we need to do many more intermediate steps.

Similar to KNN, in Collaborative Filtering, missing rating is also determined based on information about k neighbour users . Of course, we are only interested in users who have rated the item in question. Predicted ratings are usually defined as the weighted average of normalised ratings. There is a point to note: in KNN, the weights are determined based on the distance between 2 points, and these distances are non-negative numbers. Whereas in Collaborative Filtering, weights are determined based on similarity between two users, these weights can be less than 0 as shown in Figure 2.c.

The common formula used to predict the rating of user u for item i is:

$$\hat{y}_{i,u} = \frac{\sum_{u_j \in N_(u,i)} \bar{y}_{i,u_j} sim(u,u_j)}{\sum_{u_j \in N(u,i)} |sim(u,u_j)|} \tag{10}$$

After predicting the missing ratings, we get the normalised ratings matrix as shown in Figure 2.f and can now recommend items for a user.

II.2.4 k-NN inspired algorithms in Surprise

These are algorithms that are directly derived from a basic nearest neighbours approach.

- KNNBasic

  A basic collaborative filtering algorithm. The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u,v).r_{vi}}{\sum_{v \in N_i^k(u)} sim(u,v)} \tag{11}$$

- KNNWithMeans

  A basic collaborative filtering algorithm, taking into account the mean ratings of each user.

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} sim(u,v).(r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} sim(u,v)} \tag{12}$$

- KNNWithZScore

  A basic collaborative filtering algorithm, taking into account the z-score normalization of each user:

$$\hat{r}_{ui} = \mu_u + \sigma_u \frac{\sum_{v \in N_i^k(u)} sim(u,v).(r_{vi} - \mu_v)/\sigma_v}{\sum_{v \in N_i^k(u)} sim(u,v)} \tag{13}$$

- KNNBaseline

  A basic collaborative filtering algorithm taking into account a baseline rating.

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_i^k(u)} sim(u,v).(r_{vi} - b_{vi})}{\sum_{v \in N_i^k(u)} sim(u,v)} \tag{14}$$

II.3 Item-item Collaborative Filtering

	u_0	u_1	u_2	u_3	u_4	u_5	u_6	
i_0	5	5	2	0	1	?	?	→ 2.6
i_1	4	?	?	0	?	2	?	→ 2
i_2	?	4	1	?	?	1	1	→ 1.75
i_3	2	2	3	4	4	?	4	→ 3.17
i_4	2	0	4	?	?	?	5	→ 2.75

a) Original utility matrix $\mathbf{Y}$ and mean item ratings.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-.6	-2.6	-1.6	0	0
i_1	2	0	0	-2	0	0	0
i_2	0	2.25	-0.75	0	0	-0.75	-0.75
i_3	-1.17	-1.17	-0.17	0.83	0.83	0	0.83
i_4	-0.75	-2.75	1.25	0	0	0	2.25

b) Normalized utility matrix $\bar{\mathbf{Y}}$.

	i_0	i_1	i_2	i_3	i_4
i_0	1	0.77	0.49	-0.89	-0.52
i_1	0.77	1	0	-0.64	-0.14
i_2	0.49	0	1	-0.55	-0.88
i_3	-0.89	-0.64	-0.55	1	0.68
i_4	-0.52	-0.14	-0.88	0.68	1

c) Item similarity matrix $\mathbf{S}$.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-.6	-2.6	-1.6	-0.29	-1.52
i_1	2	2.4	-0.6	-2	-1.25	0	-2.25
i_2	2.4	2.25	-0.75	-2.6	-1.20	-0.75	-0.75
i_3	-1.17	-1.17	-0.17	0.83	0.83	0.34	0.83
i_4	-0.75	-2.75	1.25	1.03	1.16	0.65	2.25

d) Normalized utility matrix $\bar{\mathbf{Y}}$.

Figure 3

The results of choosing which items to recommend for each user are shown by the red boxes in Figure 3.d. This result is slightly different from the result found by User-user Collaborative Filtering in the last 2 columns corresponding to u_5, u_6. This result seems to

make more sense because from the Utility Matrix, there are two groups of users who like two different groups of items.

Computationally, the Item-item Collaborative Filtering can be obtained from the User-user Collaborative Filtering by transposing the utility matrix, and treating the items as rating users. After calculating the final result, we transpose again to get the result.

II.4 The Duality of Similarity

There is a difference in the typical behaviour of users and items, as it pertains to similarity. Intuitively, items tend to be classifiable in simple terms. For example, music tends to belong to a single genre. It is impossible, e.g., for a piece of music to be both 60's rock and 1700's baroque. On the other hand, there are individuals who like both 60's rock and 1700's baroque, and who buy examples of both types of music. The consequence is that it is easier to discover items that are similar because they belong to the same genre, than it is to detect that two users are similar because they prefer one genre in common, while each also likes some genres that the other doesn't care for.

As we suggested in II.2.2 above, one way of predicting the value of the utility matrix entry for user U and item I is to find the n users (for some predetermined n) most similar to U and average their ratings for item I, counting only those among the n similar users who have rated I. It is generally better to normalise the matrix first. That is, for each of the n users subtract their average rating for items from their rating for i. Average the difference for those users who have rated I, and then add this average to the average rating that U gives for all items. This normalisation adjusts the estimate in the case that U tends to give very high or very low ratings, or a large fraction of the similar users who rated I (of which there may be only a few) are users who tend to rate very high or very low.

Dually, we can use item similarity to estimate the entry for user U and item I. Find the m items most similar to I, for some m, and take the average rating, among the m items, of the ratings that U has given. As for user-user similarity, we consider only those items among the m that U has rated, and it is probably wise to normalise item ratings first.

Note that whichever approach to estimating entries in the utility matrix we use, it is not sufficient to find only one entry. In order to recommend items to a user U, we need to estimate every entry in the row of the utility matrix for U, or at least find all or most of the entries in that row that are blank but have a high estimated value. There is a tradeoff regarding whether we should work from similar users or similar items.

- If we find similar users, then we only have to do the process once for user U. From the set of similar users we can estimate all the blanks in the utility matrix for U. If we work from similar items, we have to compute similar items for almost all items, before we can estimate the row for U.
- On the other hand, item-item similarity often provides more reliable information, because of the phenomenon observed above, namely that it is easier to find items of the same genre than it is to find users that like only items of a single genre.

Whichever method we choose, we should precompute preferred items for each user, rather than waiting until we need to make a decision. Since the utility matrix evolves slowly, it is

generally sufficient to compute it infrequently and assume that it remains fixed between recomputations.

II.5 Metrics for evaluating Recommender Systems

Using accuracy to measure the performance of the algorithm. It quantifies how the estimations produced by the strategy deviate from their known values. The smaller the error, the better the Recommender System works. Two types of Accuracy measures are in widespread use in the literature; they are the Mean Absolute Error (MAE), and the Root Mean Squared Error (RMSE).

II.5.1 Mean absolute error (MAE)

$$MAE = \left(\frac{1}{n}\right) \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{15}$$

Mean absolute error (MAE) computes the deviation between predicted ratings and actual ratings.

This is the most straightforward metric of evaluation known as Mean absolute error. The above is a fancy equation for evaluating it. It is literally the difference between what a user might rate an item to what our system predicts.

II.5.2 Root mean square error (RMSE)

$$RMSE = \sqrt{\left(\frac{1}{n}\right) \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{16}$$

Root mean square error (RMSE) is similar to MAE, but places more emphasis on larger deviation.

This is another common and perhaps most popular metric of evaluation. One reason is that it penalises you way less when you are close to actual prediction and way more when far from actual prediction compared to MAE.

Example:

Nr.	UserID	MovieID	Rating (r_i)	Prediction (p_i)	$\lvert p_i - r_i \rvert$	$(p_i - r_i)^2$
1	1	134	5	4.5	0.5	0.25
2	1	238	4	5	1	1
3	1	312	5	5	0	0
4	2	134	3	5	2	4
5	2	767	5	4.5	0.5	0.25
6	3	68	4	4.1	0.1	0.01
7	3	212	4	3.9	0.1	0.01
8	3	238	3	3	0	0
9	4	68	4	4.2	0.2	0.04
10	4	112	5	4.8	0.2	0.04
					4.6	5.6

- MAE = 0.46
- RMSE = 0.75

III Solution and Dataset

The Book-Crossing Dataset is a book ratings dataset compiled by Cai-Nicolas Ziegler[3]. It contains 1.1 million ratings of 271,360 books by 278,858 users. The ratings are on a scale from 1 to 10.

The Book-Crossing dataset comprises 3 tables.

- BX-Users: Contains the users. Note that user IDs ('User-ID') have been anonymized and mapped to integers. Demographic data is provided ('Location', 'Age') if available. Otherwise, these fields contain NULL-values.
- BX-Books: 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 flavours ('Image-URL-S', 'Image-URL-M', 'Image-URL-L'), i.e., small, medium, large. These URLs point to the Amazon web site.
- BX-Book-Ratings: 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.

III.1 Data Preparation

Import the necessary libraries and load the datasets.

III.1.1 Data Description

In [7]:
```
print("shape of Users :",user.shape)
print("shape of books :",book.shape)
print("shape of ratings :",rating.shape)
```

Shape of Users : (278868, 3)
Shape of Books : (271360, 8)
Shape of Ratings : (1149780, 3)

In [10]:
```
user
```

Out[10]:

	User-ID	Location	Age
0	1	nyc, new york, usa	NaN
1	2	stockton, california, usa	18.0
2	3	moscow, yukon territory, russia	NaN
3	4	porto, v.n.gaia, portugal	17.0
4	5	farnborough, hants, united kingdom	NaN
...	...	...	...
278853	278854	portland, oregon, usa	NaN
278854	278855	tacoma, washington, united kingdom	50.0
278855	278856	brampton, ontario, canada	NaN
278856	278857	knoxville, tennessee, usa	NaN
278857	278858	dublin, n/a, ireland	NaN

278858 rows × 3 columns

The user dataset provides the user demographic information. It includes 278,858 records and 3 fields: user id, location and age.

The books data set provides book details. It includes 271,360 records and 8 fields: ISBN, Book-Title, Book-Author, Year-Of-Publication, Publisher and Image-URL-S, Image-URL-M, Image-URL-L.

```
In [9]: book
```

Out[9]:

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Imag
0	0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	http://images.amazon.com/images/P/0195153448.0...	http://images.amazon.com/images/P/01951
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/images/P/0002005018.0...	http://images.amazon.com/images/P/00020
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial	http://images.amazon.com/images/P/0060973129.0...	http://images.amazon.com/images/P/00609
3	0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux	http://images.amazon.com/images/P/0374157065.0...	http://images.amazon.com/images/P/03741
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton & Company	http://images.amazon.com/images/P/0393045218.0...	http://images.amazon.com/images/P/03930
...	...	...	...	...	...	...	
271355	0440400988	There's a Bat in Bunk Five	Paula Danziger	1988	Random House Childrens Pub (Mm)	http://images.amazon.com/images/P/0440400988.0...	http://images.amazon.com/images/P/04404
271356	0525447644	From One to One Hundred	Teri Sloat	1991	Dutton Books	http://images.amazon.com/images/P/0525447644.0...	http://images.amazon.com/images/P/05254
271357	006008667X	Lily Dale : The True Story of the Town that Ta...	Christine Wicker	2004	HarperSanFrancisco	http://images.amazon.com/images/P/006008667X.0...	http://images.amazon.com/images/P/00600
271358	0192126040	Republic (World's Classics)	Plato	1996	Oxford University Press	http://images.amazon.com/images/P/0192126040.0...	http://images.amazon.com/images/P/01921
271359	0767409752	A Guided Tour of Rene Descartes' Meditations o...	Christopher Biffle	2000	McGraw-Hill Humanities/Social Sciences/Languages	http://images.amazon.com/images/P/0767409752.0...	http://images.amazon.com/images/P/07674

271360 rows × 8 columns

The ratings data set provides a list of ratings that users have given to books. It includes 1,149,780 records and 3 fields: userID, ISBN, and bookRating.

```
In [11]: rating
```

Out[11]:

	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
...	...	...	...
1149775	276704	1563526298	9
1149776	276706	0679447156	0
1149777	276709	0515107662	10
1149778	276721	0590442449	10
1149779	276723	05162443314	8

1149780 rows × 3 columns

III.1.2 Data Cleaning

Checking and cleaning book data

```
In [16]: # There are missing values in these two columns so we need to fill them
book.loc[book['Book-Author'].isnull(), 'Book-Author'] = "No author"
book.loc[book['Publisher'].isnull(), 'Publisher'] = "Other"

In [17]: book['Year-Of-Publication'].unique()

Out[17]: array([2002, 2001, 1991, 1999, 2000, 1993, 1996, 1988, 2004, 1998, 1994,
       2003, 1997, 1983, 1979, 1995, 1982, 1985, 1992, 1986, 1978, 1980,
       1952, 1987, 1990, 1981, 1989, 1984,    0, 1968, 1961, 1958, 1974,
       1976, 1971, 1977, 1975, 1965, 1941, 1970, 1962, 1973, 1972, 1960,
       1966, 1920, 1956, 1959, 1953, 1951, 1942, 1963, 1964, 1969, 1954,
       1950, 1967, 2005, 1957, 1940, 1937, 1955, 1946, 1936, 1930, 2011,
       1925, 1948, 1943, 1947, 1945, 1923, 2020, 1939, 1926, 1938, 2030,
       1911, 1904, 1949, 1932, 1928, 1929, 1927, 1931, 1914, 2050, 1934,
       1910, 1933, 1902, 1924, 1921, 1900, 2038, 2026, 1944, 1917, 1901,
       2010, 1908, 1906, 1935, 1806, 2021, 2012, 2006, 1909, 2008, 1378,
       1919, 1922, 1897, 2024, 1376, 2037], dtype=int64)

In [18]: # Since year data has the year 0 and >2022 which is invalid, we shall convert it into null data
book.loc[(book['Year-Of-Publication'] > 2022) | (book['Year-Of-Publication'] == 0), 'Year-Of-Publication'] = np.NAN

# Replacing null data with median
book['Year-Of-Publication'].fillna(book['Year-Of-Publication'].median() , inplace = True)
print(book['Year-Of-Publication'].isna().sum())

# Convert it back to int
book['Year-Of-Publication'] = book['Year-Of-Publication'].astype(np.int64)

0

In [19]: book['Year-Of-Publication'].unique()

Out[19]: array([2002, 2001, 1991, 1999, 2000, 1993, 1996, 1988, 2004, 1998, 1994,
       2003, 1997, 1983, 1979, 1995, 1982, 1985, 1992, 1986, 1978, 1980,
       1952, 1987, 1990, 1981, 1989, 1984, 1968, 1961, 1958, 1974, 1976,
       1971, 1977, 1975, 1965, 1941, 1970, 1962, 1973, 1972, 1960, 1966,
       1920, 1956, 1959, 1953, 1951, 1942, 1963, 1964, 1969, 1954, 1950,
       1967, 2005, 1957, 1940, 1937, 1955, 1946, 1936, 1930, 2011, 1925,
       1948, 1943, 1947, 1945, 1923, 2020, 1939, 1926, 1938, 1911, 1904,
       1949, 1932, 1928, 1929, 1927, 1931, 1914, 1934, 1910, 1933, 1902,
       1924, 1921, 1900, 1944, 1917, 1901, 2010, 1908, 1906, 1935, 1806,
       2021, 2012, 2006, 1909, 2008, 1378, 1919, 1922, 1897, 1376],
      dtype=int64)
```

Some years have been entered as 0 and larger than 2022. We'll correct those entries

with the average of all publication year.

Checking and cleaning user data

Similarly, we also need to make corrections for the user dataset.

In [20]:
```
print(sorted(user['Age'].unique()))
```

```
[nan, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.
0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.
0, 47.0, 48.0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.
0, 70.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 80.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 90.0, 91.0, 92.
0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0, 100.0, 101.0, 102.0, 103.0, 104.0, 105.0, 106.0, 107.0, 108.0, 109.0, 110.0, 111.0, 113.0, 11
4.0, 115.0, 116.0, 118.0, 119.0, 123.0, 124.0, 127.0, 128.0, 132.0, 133.0, 136.0, 137.0, 138.0, 140.0, 141.0, 143.0, 146.0, 147.0, 148.0,
151.0, 152.0, 156.0, 157.0, 159.0, 162.0, 168.0, 172.0, 175.0, 183.0, 186.0, 189.0, 199.0, 200.0, 201.0, 204.0, 207.0, 208.0, 209.0, 210.
0, 212.0, 219.0, 220.0, 223.0, 226.0, 228.0, 229.0, 230.0, 231.0, 237.0, 239.0, 244.0]
```

A lot of values from 'Age' column is missing and some others are too low or too high for an average person (200 years old).

In [21]:
```
# Removing age above 120 and below 4
user.loc[(user['Age'] > 120) | (user['Age'] < 4) , 'Age' ] = np.NAN
user['Age'].isna().sum()
```

Out[21]:
```
111694
```

In [22]:
```
# Filling the null values with mean
user['Age'].fillna(user['Age'].mean(), inplace = True)
user['Age'] = user['Age'].astype(np.int64)
```

In [23]:
```
print(sorted(user['Age'].unique()))
```

```
[4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 7
4, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
107, 108, 109, 110, 111, 113, 114, 115, 116, 118, 119]
```

Data Processing

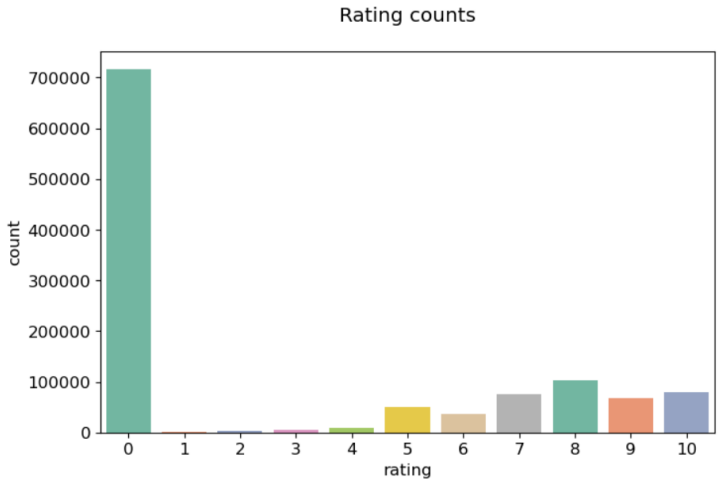
We need to remove redundant or irrelevant data such as publishing date, format, cover image, number of pages, etc. and avoid repetitions by creating an ID for each book and clearing duplicate ratings.

In [24]:
```
# Preprocessing Data
book = book[['ISBN', 'Book-Title', 'Book-Author', 'Year-Of-Publication', 'Publisher']]
book.rename(columns = {'Book-Title':'title', 'Book-Author':'author', 'Year-Of-Publication':'year', 'Publisher':'publisher'}, inplace=True)
user.rename(columns = {'User-ID':'user_id', 'Location':'location', 'Age':'age'}, inplace=True)
rating.rename(columns = {'User-ID':'user_id', 'Book-Rating':'rating'}, inplace=True)
```

III.2 Data Analysis

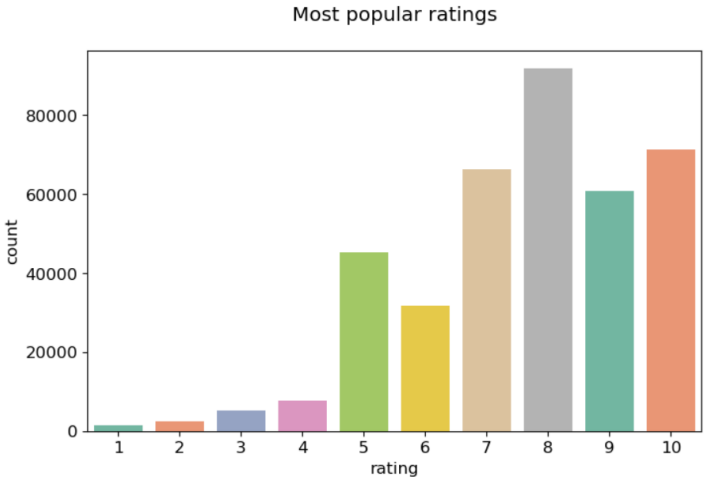
III.2.1 Rating Distribution

The ratings are very unevenly distributed, and the vast majority of ratings are 0.



This countplot shows users have rated 0 the most, which means they haven't rated books at all. Still we can see patterns to recognize in ratings from 1-10.

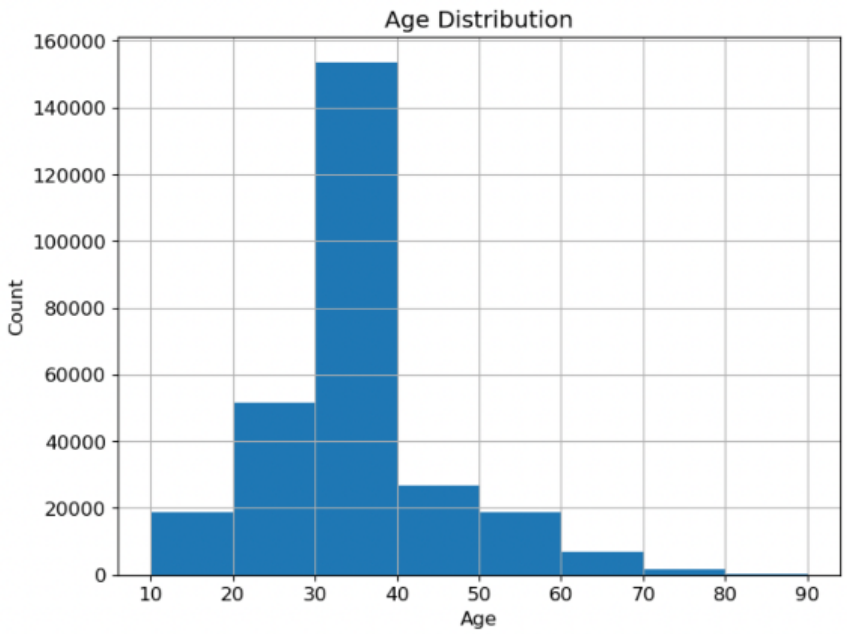
Mostly the users have rated 8 ratings out of 10 as per books. It might happen that the feedback is positive but not extremely positive as 10 ratings (i.e best books ever).



Now this countplot of bookRating indicates that higher ratings are more common amongst users and rating 8 has been rated highest number of times. There can be many assumptions based on ratings of users:

- Let's take the rating groups from 1-4. This can have a negative impact for books being published if they have ratings from 1 to 4. It can be issues related to:
  - Language
  - Offend by any chapter's incident/paragraph/author.
  - They have read the worst book ever.
- If we think analytically about rating 5, it might happen for the same reason as the above key points mentioned.
- For 5 ratings the users might not be sure about book ratings whether it's positive or negative impact.
- Let's take the rating groups from 6-10. This is positive feedback:
  1. A lot of the users have decided to rate 8. This could possibly be because the users enjoyed the book but it is rare that a book can be exactly perfect to one's liking so they gave it an 8.
  2. The 6 rating is very low among other ratings.
  3. As we can see, the number of books rated as 7 or 8 is average and these books generally get more ratings from users.
  4. The 9 and 10 ratings are top best ratings based on Author's, Publisher's and Books being published.

III.2.2 Age Distribution:



- The most active users are among those in their 30–40s. The reason could be that the users are more interested in what the authors are publishing in the market. There can be a lot of different explanations as to why there are such differences among the age groups.

IV Difficulties and Solutions

The recommendation system implementation, which in theory does not look so difficult, became quite complex to implement due to the problems that occurred when we processed the data. It is hard to detect similarity among either items or users, because we have little information about user-item pairs in the sparse utility matrix. In the perspective of section II.1, even if two items belong to the same genre, there are likely to be very few users who rated both. Likewise, even if two users both like a genre or genres, they may not have read any books in common.

IV.1 Some limitations of User-user CF:

- In reality, the number of users is always so much bigger than the number of items. Thus the Similarity matrix is very large with the number of elements to be kept being more than half of the square of the number of users (note that this matrix is symmetric). This, as mentioned above, makes it impractical to store this matrix in many cases.
- Utility Matrix is usually very sparse. With a huge number of users compared to the number of items, many columns of this matrix will be very sparse, i.e. only a few non-zero elements. The reason is that users are often lazy to rate. Because of this, once the user changes the rating or rates more items, the average of the ratings as well as the normalised vector corresponding to this user would change a lot. Accordingly, the calculation of the Similarity matrix, which takes a lot of memory and time, also needs to be done again.

On the contrary, if we calculate similarity between items and recommend items that are close to a user's favourite item, there will be the following benefits:

- Since the number of items is usually smaller than the number of users, the Similarity matrix in this case is also much smaller, which is convenient for storage and computation in later steps.
- Since the number of known elements in the Utility matrix is the same but the number of rows (items) is less than the number of columns (users), on average, each row of this matrix will have more known elements than known elements in each column. This is understandable since each item can be rated by multiple users . As a result, the average value of each row is less likely to change when a few more ratings are added. As such, updating the Similarity Matrix can be done less frequently.

This second approach is called Item-item Collaborative Filtering. This approach is more commonly used in practice.

The procedure for predicting missing ratings is the same as in User-user Collaborative Filtering. Figure 3 depicts this process with the example given above.

IV.2 Limit Dataset for matrix

To ensure that we can effectively compute a matrix and make reliable recommendations, it is important to limit the size of our dataset. Otherwise the computer running the algorithms won't have enough memory to store the matrix. We will do this by only including users who have made at least 100 ratings and books that have received at least 50 ratings in our explicit dataset. For the dataset containing 0 ratings, we will only include users who have made at least 250 ratings and books that have received at least 50 ratings. This will help us avoid rating bias and ensure the quality of our recommendations.

IV.3 Explicit and Implicit

For the dataset we have, the rating 0 is much more prominent than others rating.

```
# ratings with 0 rating and without 0 rating ( in rating data includes books given in the dataset )
ratings_explicit=ratings_new[ratings_new['rating']!=0]
ratings_implicit=ratings_new[ratings_new['rating']==0]
print(ratings_new.shape)
print(ratings_explicit.shape)
print(ratings_implicit.shape)
```

```
(1031136, 3)
(383842, 3)
(647294, 3)
```

We can use both the implicit dataset and explicit dataset to evaluate the performance of our system.

V Result and Analysis

Result when using Surprise to build system

- Dataset with '0' ratings:

	MAE	RMSE	fit_time	test_time
knns.KNNWithMeans	2.704184	3.388098	0.413353	2.334252
knns.KNNBaseline	2.715056	3.392585	0.735178	4.491177
knns.KNNWithZScore	2.688870	3.411428	0.621295	3.125594
knns.KNNBasic	2.819927	3.568940	0.382429	2.028248

- Dataset without '0' ratings:

	MAE	RMSE	fit_time	test_time
knns.KNNWithMeans	1.275442	1.717533	0.124490	0.232515
knns.KNNWithZScore	1.274988	1.728506	0.320817	0.322542
knns.KNNBaseline	1.308050	1.731568	0.091116	0.176107
knns.KNNBasic	1.435886	1.884052	0.062674	0.146085

When using system for dataset with non-zeros ratings, the MAE and RMSE are smaller than dataset with 0 ratings. For the reasons, the data become smaller so the time it would take is less and the accuracy is increased (because the rating is quite clear).

- Machine Learning – Hyperparameter tuning with GridSearchCV

  Using GridSearchCV to know the parameter for the dataset. If you want to know which parameter combination yields the best results, the GridSearchCV class comes to the rescue.

```
# Hyperparameter tuning - KNNWithMeans with data_nonzero

param_grid = { 'sim_options' : {'name': ['msd','cosine','pearson','pearson_baseline'], \
                                'min_support': [1,5], \
                                'user_based': [False, True]}
             }

gridsearchKNNWithMeans = GridSearchCV(KNNWithMeans, param_grid, measures=['mae', 'rmse'], \
                                      cv=5, n_jobs=-1)

gridsearchKNNWithMeans.fit(data_nonzero)

print(f'MAE Best Parameters:  {gridsearchKNNWithMeans.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchKNNWithMeans.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchKNNWithMeans.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchKNNWithMeans.best_score["rmse"]}\n')
```

```
MAE Best Parameters:  {'sim_options': {'name': 'pearson_baseline', 'min_support': 5, 'user_based': True}}
MAE Best Score:       1.2477344142760778

RMSE Best Parameters: {'sim_options': {'name': 'pearson_baseline', 'min_support': 5, 'user_based': True}}
RMSE Best Score:      1.6757927514357989
```

KNNWithMeans: best parameters are

- Similarity function: pearson_baseline
- Min users: 5
- item_based

```
# Hyperparameter tuning - KNNBasic

param_grid = { 'sim_options' : {'name': ['msd','cosine','pearson','pearson_baseline'], \
                                'min_support': [1,5], \
                                'user_based': [False, True]}
             }

gridsearchKNNBasic = GridSearchCV(KNNBasic, param_grid, measures=['mae', 'rmse'], \
                                  cv=5, n_jobs=-1)

gridsearchKNNBasic.fit(data_nonzero)

print(f'MAE Best Parameters:  {gridsearchKNNBasic.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchKNNBasic.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchKNNBasic.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchKNNBasic.best_score["rmse"]}\n')
```

```
MAE Best Parameters:  {'sim_options': {'name': 'cosine', 'min_support': 1, 'user_based': False}}
MAE Best Score:       1.2455148592739367

RMSE Best Parameters: {'sim_options': {'name': 'cosine', 'min_support': 1, 'user_based': False}}
RMSE Best Score:      1.694018748657123
```

KNNBasic: best parameters are

- Similarity function: cosine
- Min users: 1
- item_based

```
# Hyperparameter tuning - KNNWithZScore

param_grid = { 'sim_options' : {'name': ['msd','cosine','pearson','pearson_baseline'], \
                                'min_support': [1,5], \
                                'user_based': [False, True]}
             }

gridsearchKNN = GridSearchCV(KNNWithZScore, param_grid, measures=['mae', 'rmse'], \
                                  cv=5, n_jobs=-1)

gridsearchKNN.fit(data_nonzero)

print(f'MAE Best Parameters:  {gridsearchKNN.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchKNN.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchKNN.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchKNN.best_score["rmse"]}\n')
```

```
MAE Best Parameters:  {'sim_options': {'name': 'pearson_baseline', 'min_support': 5, 'user_based': True}}
MAE Best Score:       1.234916795291311

RMSE Best Parameters: {'sim_options': {'name': 'pearson_baseline', 'min_support': 5, 'user_based': True}}
RMSE Best Score:      1.6600244966827442
```

KNNWithZScore: best parameters are

- Similarity function: pearson_baseline
- Min users: 5
- item_based

```
# Hyperparameter tuning - KNNBaseLine

param_grid = { 'sim_options' : {'name': ['msd','cosine','pearson','pearson_baseline'], \
                                'min_support': [1,5], \
                                'user_based': [False, True]}
             }

gridsearchKNN = GridSearchCV(KNNBaseline, param_grid, measures=['mae', 'rmse'], \
                                    cv=5, n_jobs=-1)

gridsearchKNN.fit(data_nonzero)

print(f'MAE Best Parameters:  {gridsearchKNN.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchKNN.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchKNN.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchKNN.best_score["rmse"]}\n')
```

```
MAE Best Parameters:  {'sim_options': {'name': 'pearson_baseline', 'min_support': 5, 'user_based': False}}
MAE Best Score:       1.1832349103678226

RMSE Best Parameters: {'sim_options': {'name': 'pearson', 'min_support': 5, 'user_based': False}}
RMSE Best Score:      1.5521472879073785
```

KNNBaseline: best parameters are

- Similarity function: pearson
- Min users: 5
- item_based

When we use system to train, fit and test the data: KNNWithMeans brings good performance.

```
# Model fit & prediction - KNNWithMeans

sim_options = {'name':'cosine','min_support':1,'user_based':False}
final_model = KNNWithMeans(sim_options=sim_options)

# Fitting the model on trainset & predicting on testset, printing test accuracy
pred = final_model.fit(trainset).test(testset)

print(f'\nUnbiased Testing Performance:')
print(f'MAE: {accuracy.mae(pred)}, RMSE: {accuracy.rmse(pred)}')
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.

Unbiased Testing Performance:
MAE:  1.2205
RMSE: 1.6299
MAE: 1.220476817580134, RMSE: 1.6299293946102147
```

- Top 10 books recommended to user

  Algorithm: Using surprise - KNNWithMeans

	ISBN	Book-Title
0	0312966970	Four To Score (A Stephanie Plum Novel)
1	0440226430	Summer Sisters
2	0842329129	Left Behind: A Novel of the Earth's Last Days ...
3	0451172817	Needful Things
4	0440224764	The Partner
5	0312990456	One for the Money (A Stephanie Plum Novel)
6	0671003755	She's Come Undone (Oprah's Book Club (Paperback))
7	0515131229	Dance upon the Air (Three Sisters Island Trilogy)
8	043935806X	Harry Potter and the Order of the Phoenix (Boo...
9	0553569910	The Ugly Duckling
10	0385511612	Bleachers

Algorithm: Using Surprise - KNNBasis

	ISBN	Book-Title
0	0312966970	Four To Score (A Stephanie Plum Novel)
1	0440226430	Summer Sisters
2	0842329129	Left Behind: A Novel of the Earth's Last Days ...
3	0451172817	Needful Things
4	0440224764	The Partner
5	0312990456	One for the Money (A Stephanie Plum Novel)
6	0671003755	She's Come Undone (Oprah's Book Club (Paperback))
7	0515131229	Dance upon the Air (Three Sisters Island Trilogy)
8	043935806X	Harry Potter and the Order of the Phoenix (Boo...
9	0553569910	The Ugly Duckling
10	0385511612	Bleachers

Algorithm: Using Surprise - KNNBaseline

	ISBN	Book-Title
0	044021145X	The Firm
1	0312195516	The Red Tent (Bestselling Backlist)
2	0439139600	Harry Potter and the Goblet of Fire (Book 4)
3	0385335881	Shopaholic Takes Manhattan (Summer Display Opp...
4	0440213525	The Client
5	0515131229	Dance upon the Air (Three Sisters Island Trilogy)
6	0515128554	Heart of the Sea (Irish Trilogy)
7	0142001740	The Secret Life of Bees
8	0440226430	Summer Sisters
9	080410526X	All I Really Need to Know
10	0425163407	Unnatural Exposure

KNNWithMeans and KNNBasic with same similarity function: 'cosine' have similar results, and not same as KNNBaseline, because the similarity function is different.

VI Typical problems and proposal for improvement of the system in future

VI.1 Sparsity

One of the hardest problems to deal with at the time of making recommendations is that of sparsity of the information. A user will tend to only rate a few items, and generally the system will hold on file a large number of users with little item ratings recorded for each. In other words, the input matrix of users and items is inherently very sparse.

This has profound effects for the Recommender System. There might be lots of truly similar candidate users to a target user, but all similarity algorithms work by matching the

ratings that were shared by those users, and without ratings, similar users will not be identified.

A common strategy to deal with the sparsity problem is unknown rating values are replaced by averages of the known ratings entered by a user. But this strategy assumes that the missing ratings are efficiently modelled by an average function, when in fact they might well lay low below or high above the known ratings mean.

VI.2 Cold Start

At the heart of the Collaborative Filtering algorithm lies the user rating. Without ratings, no recommendation can take place for any user of the system. This problem is termed Cold Start, and describes the inability of a Recommender System to start offering recommendations.

One way of breaking this pattern is to force new users to rate a number of items before allowing them to use the system. This strategy ensures there are always new ratings coming into the system, while solving also a second problem that affects every newcomer: a new user that has yet to enter his own preferences cannot be recommended anything at all by the system.

VI.3 Other improvements

Giving our project with a GUI (Graphical User Interface) would greatly enhance the user experience and make it more user-friendly for those who may not be as familiar with Jupyter Notebook or command-line interfaces.

VII Conclusion

As a book lover, finding your next great read can be both exhilarating and overwhelming. With so many options to choose from, it's hard to know where to start. That's where a book recommendation system comes in handy. By using collaborative filtering to suggest titles based on your reading preferences, you can easily discover new and exciting books that are tailored just for you. Imagine being able to narrow down your choices and find the perfect read in just a few seconds, instead of scrolling through an multiple libraries of books. A book recommendation system using collaborative filtering is the ultimate tool for any book lover looking to expand their reading horizons.

In this project, we have implemented a recommendation system using memory-based collaborative filtering. Collaborative filtering is a method of recommendation that is based on the idea that people who have similar tastes and preferences will have similar ratings for items. In our project, we have used two approaches: user-based and item-based. For the user-based Collaborative Filtering, the concept is to use the ratings given by users for a set of items to find other users who have similar ratings and use those ratings to predict the ratings that the active user would give to items they have not yet rated. We have evaluated the performance of our system using some evaluation metrics. Overall, our system is able to provide recommendations for users and show good performance on the evaluation metrics.

VIII Reference

References

[1] Raghav Agrawal. *Book Recommendation System — Build A Book Recommendation System*. en. June 2021. URL: https://www.analyticsvidhya.com/blog/2021/06/build-book-recommendation-system-unsupervised-learning-project/.

[2] Amit. *Book Recommendation System*. en. Oct. 2022. URL: https://medium.com/@amitdlmlai/book-recommendation-system-61bf9284f659.

[3] *Book-Crossing Dataset*. URL: http://www2.informatik.uni-freiburg.de/~cziegler/BX/.

[4] Parth Chokhra. *Evaluating Recommender Systems*. en. Apr. 2021. URL: https://medium.com/nerd-for-tech/evaluating-recommender-systems-590a7b87afa5.

[5] *Data Science Blogathon - 9*. en. URL: https://datahack.analyticsvidhya.com/contest/data-science-blogathon-9/.

[6] Munia Humaira. *Book-Crossing Data Review*. en. Mar. 2020. URL: https://github.com/muniah/Book-Recommendation-System.

[7] Real Python. *Build a Recommendation Engine With Collaborative Filtering – Real Python*. en. URL: https://realpython.com/build-recommendation-engine-collaborative-filtering/ (visited on 01/03/2023).

[8] Rochita Sundar. *Book Recommendation System using Surprise*. en. Mar. 2022. URL: https://github.com/rochitasundar/Collaborative-Filtering-Book-Recommendation-System.

[9] Chhavi Saluja. *My Journey to building Book Recommendation System. . .* en. Mar. 2018. URL: https://towardsdatascience.com/my-journey-to-building-book-recommendation-system-5ec959c41847.

[10] Tiep Vu. *Bài 24: Neighborhood-Based Collaborative Filtering*. May 2017. URL: https://machinelearningcoban.com/2017/05/24/collaborativefiltering/.

[11] *Welcome to Surprise' documentation! — Surprise 1 documentation*. URL: https://surprise.readthedocs.io/en/stable/index.html.