Machine Learning Report

Mansi Upadhyay
College of Engineering
Northeastern University
Boston, MA
Upaddhyay.ma@northeastern.edu

1. Introduction

In an era dominated by information overload, the challenge of selecting the perfect book can be overwhelming. Recognizing the need for a more personalized approach to reading recommendations, our Book Recommendation Project was initiated. This report provides an insightful overview of the project's development, implementation, and the seamless fusion of technology and literature to enhance the reader's experience.

As we navigate through the subsequent sections, we delve into the methodologies employed in creating our Recommendation System. From understanding user preferences to implementing cutting-edge algorithms, this report sheds light on the journey undertaken to make literary exploration a more tailored and enjoyable endeavor. Join us in exploring the transformative potential of our Book Recommendation Project as we strive to make the vast literary landscape more accessible and delightful for every reader.

2. Problem Definition

Creating a book recommendation system using collaborative filtering is prompted by the challenge of enhancing the reader's experience in the face of a vast and diverse literary landscape. The need arises from the limitations of existing generic recommendation approaches, which fail to deliver tailored suggestions, leaving readers to navigate an overwhelming sea of choices. This problem statement emphasizes the demand for a refined system that leverages collaborative filtering to provide readers with personalized and engaging book recommendations, minimizing the frustration of choice overload and decision fatigue.

3. Dataset Description

In this project, I have use Goodreads dataset. The dataset was originally scraped from the Goodreads API in September 2017 by Zygmunt Zając and updated by Olivier Simard-Hanley.

This dataset consist of 5 files which are:

ratings.csv: contains user ratings for books they read
books_enriched.csv: contains metadata for each book (book ID, title, authors, year published, etc)
to-read.csv: contains books marked "to read" by users
book_tag.csv: contains tags/shelves/genres assigned by users to books
tag.csv: contains the tag names corresponding to the tag ids in book_tag.csv

I will only use two files which are user ratings and metadata of the books.

```
[ ] pd.options.display.float_format = '{:.2f}'.format
    r = pd.read_csv('data/ratings.csv')
    b = pd.read_csv('data/books_enriched.csv')
```

4. Data Understanding

In **dataset r**, there are three columns, which are:

- user_id: user identification number
- book_id: book identification number
- rating: rating given by user_id

```
r.head()
```

	user_id	book_id	rating
0	1	258	5
1	2	4081	4
2	2	260	5
3	2	9296	5
4	2	2318	3

In **dataset r**, there are three columns, which are:

1. `user_id`: user identification number
2. `book_id`: book identification number
3. `rating`: rating given by `user_id`

```
[ ] r.shape

(5976479, 3)
```

```
[ ] for col in r.columns:
        print(f"Number of {col} is {r[col].nunique()}")

Number of user_id is 53424
Number of book_id is 10000
Number of rating is 5
```

There are 5,976,479 ratings given by 53,424 people on 10,000 books.

In this dataset, there are 30 columns. However, after investigating them, there are several columns that are repeated. For example too many book_id columns, 2 title columns, 2 authors column, details of the number of reviews per rating for each book, etc. In order to make cleaner metadata, I will do data preprocessing.

Columns in dataset b:

- Identification number related (`book_id`, `goodreads_book_id`, `best_book_id`, `work_id`, `isbn`, `isbn13`)
- Title related (`original_title`, `title`)
- Authors related (`authors`, `authors_2`)
- Publication year related (`original_publication_year`, `publishDate`: the publication date)
- Rating related (`average_rating`, `ratings_count`: number of review, `work_ratings_count`, `work_text_reviews_count`, `ratings_1`, `ratings_2`, `ratings_3`, `ratings_4`, `ratings_5`)
- Image Url (`image_url`, `small_image_url`)
- `books_count`: number of edition available
- `language_code`: abbreviated language tags for all books
- `genres`: the genre tags taken from the top shelves users have assigned to a book. Only the main Goodreads genres have been retained
- `pages`: the total page count
- `description`: a free text summarizing the book's content
- Others (`Unnamed: 0`, `index`)

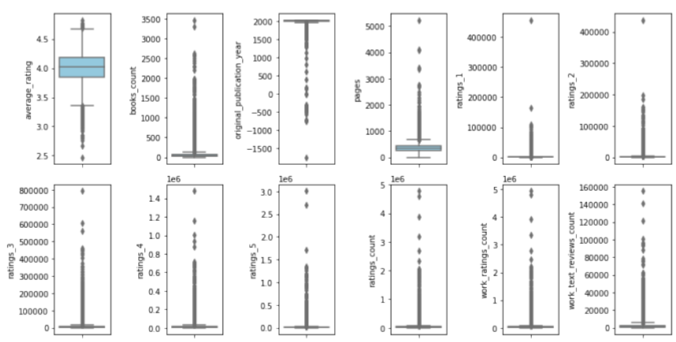
```
[ ] b.shape
(10000, 30)
```

It has 10000 rows and 30 columns. We want to display values in all columns, therefore we will transpose the table and only show first three rows.

```
b.head(3).T
```

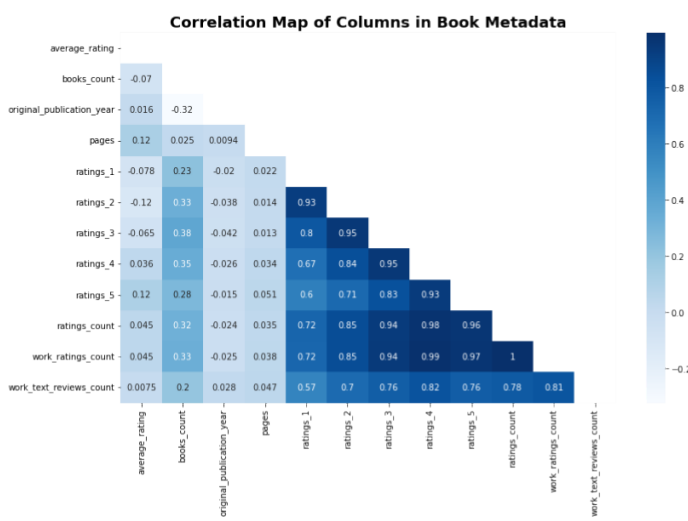
	0	1	2
Unnamed: 0	0	1	2
index	0	1	2
authors	['Suzanne Collins']	['J.K. Rowling', 'Mary GrandPré']	['Stephenie Meyer']
average_rating	4.34	4.44	3.57
best_book_id	2767052	3	41865
book_id	1	2	3
books_count	272	491	226
description	WINNING MEANS FAME AND FORTUNE.LOSING MEANS CE...	Harry Potter's life is miserable. His parents ...	About three things I was absolutely positive.\...
genres	['young-adult', 'fiction', 'fantasy', 'science...	['fantasy', 'fiction', 'young-adult', 'classics']	['young-adult', 'fantasy', 'romance', 'fiction...
goodreads_book_id	2767052	3	41865

Univariate Analysis



The data varies greatly, so there is no need to clean the outliers.

Bivariate Analysis



- There is multicollinearity between ratings related columns. We can also see that the higher ratings show higher correlation to work_ratings_count and work_text_reviews_count. People tend to leave reviews for books they like. I will only use ratings_count.
- There is a high correlation between original_publication_year and books_count. It makes sense since the older book probably has been printed more and translated to many languages.

5. Data Pre-processing

Missing Value and Duplicated Rows

```
#check missing value
dataset = [r, b]
for data in dataset:
    print(data.isnull().values.any())
```

```
False
True
```

```
#check duplicated rows
for data in dataset:
    print(data.duplicated().values.any())
```

```
False
False
```

There is no duplicate row in both dataset, but we have missing value in book metadata.

	percent_missing
isbn	7.00
original_title	5.85
isbn13	5.85
pages	0.73
description	0.57
original_publication_year	0.21
publishDate	0.08

- Impute `original_publication_year` by using `publishDate`, then drop `publishDate`. I chose `original_publication_year` because it has the same format meanwhile `publishDate` has various data styles.
- Impute `pages` with median.
- Impute `description` with book's title
- Drop `isbn`, `original_title`, `isbn13`, there is no need to impute these columns.

3881	NaN	September 29th 2009
4252	NaN	November 10th 2010
4303	NaN	June 23rd 2009
4392	NaN	April 8th 2013
5001	NaN	November 9th 2004
5208	NaN	December 6th 2010
5648	NaN	October 11th 2006
7163	NaN	November 25th 2004
7850	NaN	October 15th 2007
8119	NaN	2009
8227	NaN	('6', '1', '1998')
8532	NaN	('11', '5', '2003')
8568	NaN	('11', '5', '2003')
9140	NaN	('10', '2', '2010')
9151	NaN	('2', '2', '2016')
9198	NaN	('9', '1', '2001')
9254	NaN	(None, None, '2000')
9728	NaN	('9', '25', '2012')
9842	NaN	('6', '16', '2009')

All invalid rows in 'original_publication_year' have the year in 'publishDate'. As a result, we can extract the year from 'publishDate' and assign it to 'original_publication_year'.

The preprocessing is done and now the data has no missing values.

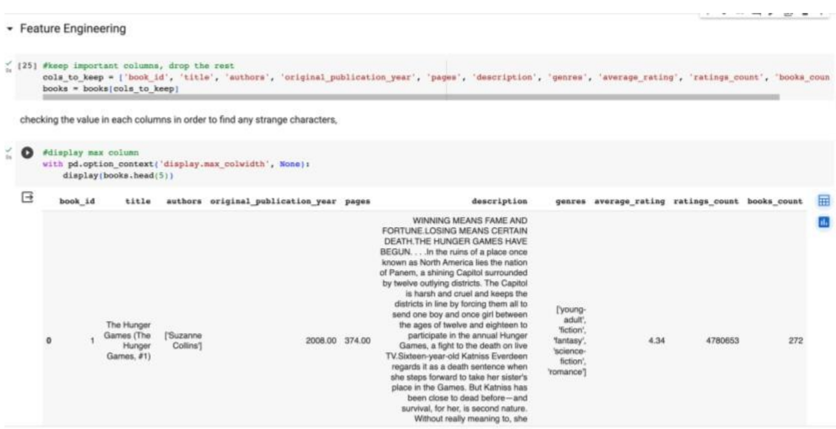
6. Feature Scaling

During the analysis of the dataset, I identified the presence of unnecessary characters in the description column of the top 5 entries. In an effort to enhance the quality of the dataset for finding similar books, I undertook the task of character cleansing and optimization. This process involved expanding the column width to its maximum to meticulously inspect and identify any irregular characters. Although this task is time-consuming, I aimed to comprehensively capture and rectify the presence of undesirable characters.

After inspecting the top 5 entries, I successfully identified and removed the unnecessary characters in the description column, ensuring that only relevant and meaningful information remains. To improve the consistency and effectiveness of the analysis, I further standardized the data by converting all text to lowercase.

To validate the efficacy of my cleaning process, I randomly sampled 10 entries from the dataset. My thorough review indicates that the character removal and standardization have been successful, with no discernible unnecessary characters present in the description column. This process is crucial for ensuring that the dataset is free from artifacts that could potentially interfere with the accurate identification of similar books.

As a result, I am confident that the cleaned dataset is well-prepared for subsequent analysis, providing a solid foundation for finding similar books based on their descriptions. The removal of unnecessary characters and the standardization of text contribute to a more robust and reliable dataset, enhancing the overall quality of the recommendation system.



In the top 5 entry, I found unnecessary characters that are not supposed to be in `description` column. We need to get rid of those extraneous characters if we want to find related novels faster. It takes time to go through it one by one, but we did our best to collect all unneeded characters and guess the rest. Then we remove the unneeded characters and lowercase all fonts.

```
#delete unnecessary characters from authors, genres and description column
col_trans = ['authors', 'genres', 'description']
for col in col_trans:
    books[col].replace(r"[\[\]\-\\\/\"-'()|:]", "", regex=True, inplace=True)
books['description'].replace({"\n" : " ", "isbn13" : "", "isbn" : "", r"[0-9]{8,}" : "", r"[.,]" : ""}, regex=True, inplace=True)
books['description'] = books['description'].str.lower()
```

```
[28] #rename column and change year, pages, and book_count to integer
books.rename(columns = {'original_publication_year':'year'}, inplace = True)
books[['year', 'pages', 'books_count']] = books[['year', 'pages', 'books_count']].astype(int)
```

Getting10 samples to check there are no unnecessary characters anymore

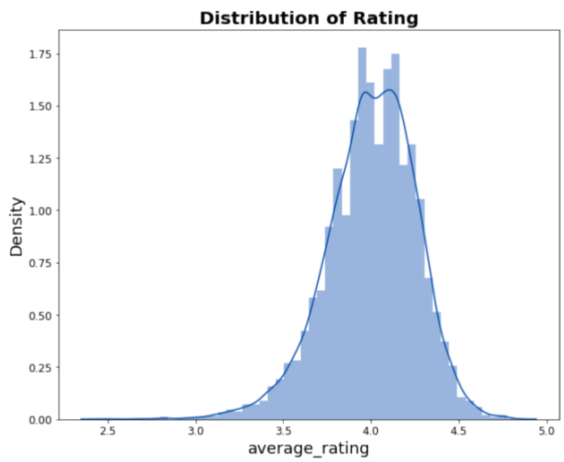
```
[29] with pd.option_context('display.max_colwidth', None):
    display(books.sample(10))
```

	book_id	title	authors	year	pages	description	genres	average_rating	ratings_count	books_count
5498	6240	Conversations With God: An Uncommon Dialogue, Book 3	Neale Donald Walsch	1996	392	the sequel to the bestselling books one and two of conversations with god this powerful and inspirational dialogue expands as readers enter into an adventurous exploration of life gain a better understanding of the universe and fully develop a personal relationship with god	spirituality, nonfiction, religion, philosophy, selfhelp	4.18	12301	38
						the years of lyndon johnson is the political biography of our time no presidentno era of american politicshas been				

Executing (17m 49s) <cell line: 14> > fit() > __call__() > _get_outputs() > _retrieve()

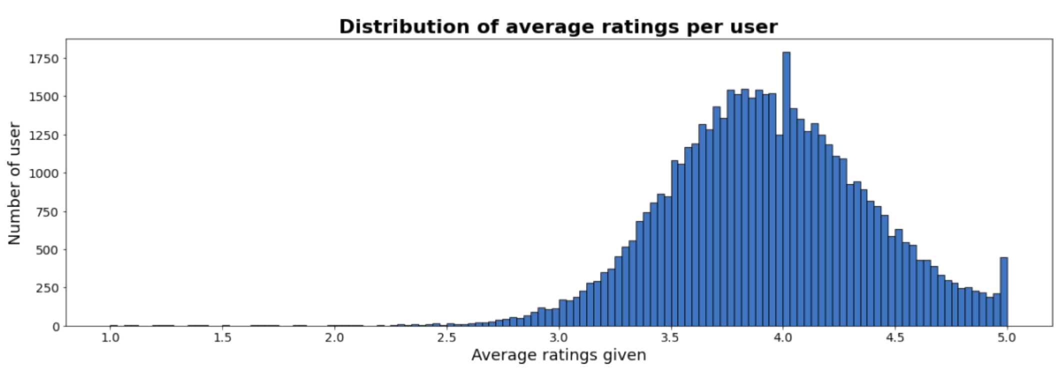
7. Exploratory Data Analysis

1. How is the rating for all books distributed?



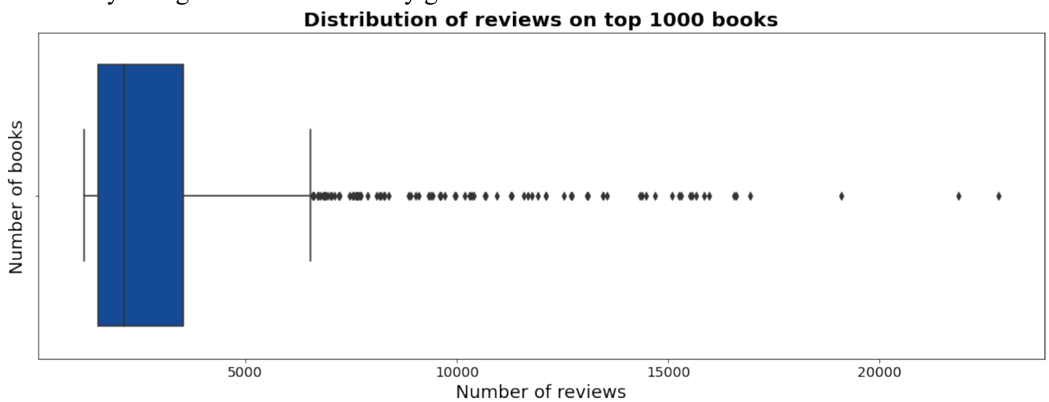
Since this is a list of 10,000 popular books, we admit the fact that the majority of the books are good books with an average value of 4.

2. How is the average rating per user distributed?



There are few low ratings, and the peak seems to be at 4.

3. How many ratings does a book usually get ?



The distribution of reviews on books is positively skewed. There are more books that has less ratings. Let's check the number of rating distribution.

```
ratings_per_book['rating'].describe()
```

```
count    10000.00
mean       597.65
std       1267.29
min          8.00
25%        155.00
50%        248.00
75%        503.00
max      22806.00
Name: rating, dtype: float64
```

+ Code + Text

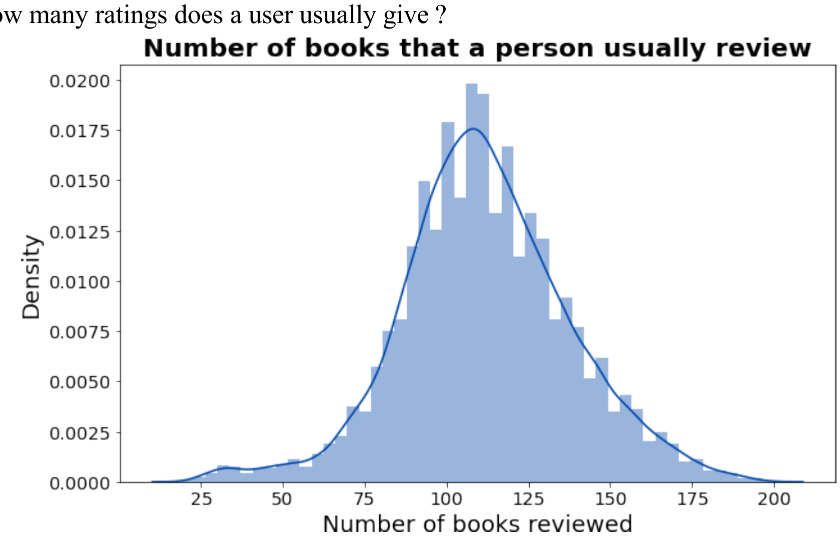
Books in this database have at least 8 reviews, while popular books have 22,806 reviews. The gap is too large. Now, let's check the outliers.

```
[ ] q3, q1 = np.percentile(ratings_per_book, [75 ,25])
    iqr = q3 - q1
    maximum = q3 + (1.5 * iqr)
    outlier_books = len(ratings_per_book[ratings_per_book['rating'] > maximum])
    print(f'We have {outlier_books} books that are considered outlier, because they have more than {int(maximum)} number of ratings')
```

```
We have 26 books that are considered outlier, because they have more than 12515 number of ratings
```

I will not delete the outliers, to keep the data variety.

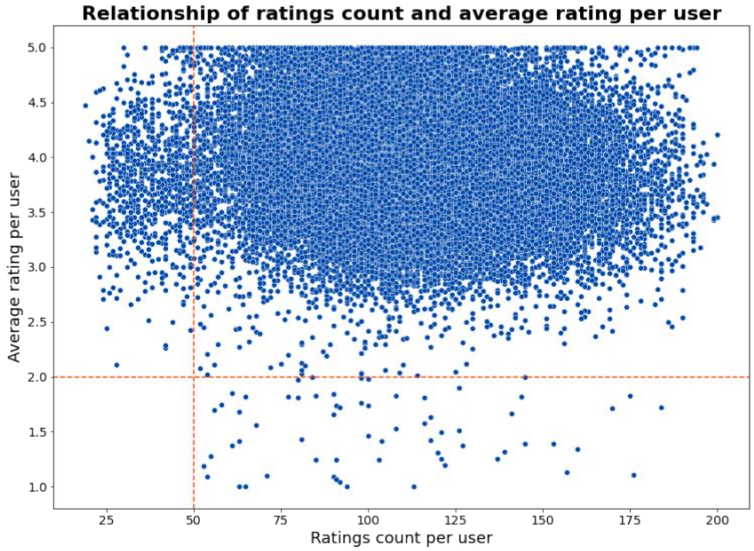
4. How many ratings does a user usually give ?



At most people review 200 books and at least 19 books. The average person gives a review of 111 books. From 10,000 books in our dataset, even the **useí with the highest numbeí of íeviews** managed to give a rating to **only 2% of all of the books**. Data in user is very sparse, so it will be better to use item-based collaborative filtering.

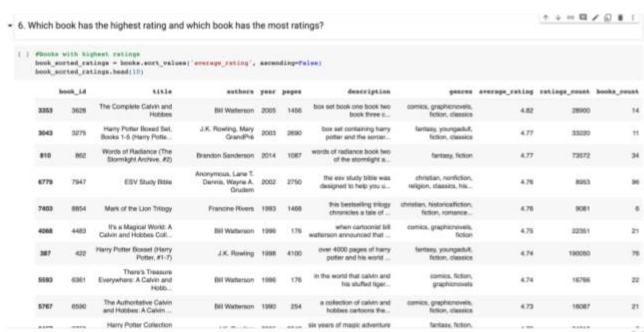
I also checked which user has the most reviews. I will use their `user_id` to test the recommendations.

5. Does the ratings count affect average rating ?



- People who rate < 50 books tend to give higher ratings.
- People start to give lower rating if they read more books.
- This could be a result of an inappropriate book recommendation system, so that people end up reading books they don't like.

6. Which book has the highest rating and which book has the most ratings?



6. Which book has the highest rating and which book has the most ratings?

```
#Books with highest ratings
book_sorted_ratings = books.sort_values('average_rating', ascending=False)
book_sorted_ratings.head(10)
```

	book_id	title	authors	year	pages	description	genres	average_rating	ratings_count	books_count
3353	3628	The Complete Calvin and Hobbes	Bill Watterson	2005	1456	box set book one book two book three c...	comics, graphicnovels, fiction, classics	4.82	28900	14
3043	3275	Harry Potter Boxed Set, Books 1-5 (Harry Potte...	J.K. Rowling, Mary GrandPré	2003	2690	box set containing harry potter and the sorcer...	fantasy, youngadult, fiction, classics	4.77	33220	11
810	862	Words of Radiance (The Stormlight Archive, #2)	Brandon Sanderson	2014	1087	words of radiance book two of the stormlight a...	fantasy, fiction	4.77	73572	34
6779	7947	ESV Study Bible	Anonymous, Lane T. Dennis, Wayne A. Grudem	2002	2750	the esv study bible was designed to help you u...	christian, nonfiction, religion, classics, his...	4.76	8953	96
7403	8854	Mark of the Lion Trilogy	Francine Rivers	1993	1468	this bestselling trilogy chronicles a tale of ...	christian, historicalfiction, fiction, romance...	4.76	9081	6
4068	4483	It's a Magical World: A Calvin and Hobbes Coll...	Bill Watterson	1996	176	when cartoonist bill watterson announced that ...	comics, graphicnovels, fiction	4.75	22351	21
387	422	Harry Potter Boxset (Harry Potter, #1-7)	J.K. Rowling	1998	4100	over 4000 pages of harry potter and his world ...	fantasy, youngadult, fiction, classics	4.74	190050	76
5593	6361	There's Treasure Everywhere: A Calvin and Hobb...	Bill Watterson	1996	176	in the world that calvin and his stuffed tiger...	comics, fiction, graphicnovels	4.74	16766	22
5767	6590	The Authoritative Calvin and Hobbes: A Calvin ...	Bill Watterson	1990	254	a collection of calvin and hobbes cartoons the...	comics, graphicnovels, fiction, classics	4.73	16087	21
[illegible]	[illegible]	Harry Potter Collection	[illegible]	[illegible]	[illegible]	six years of magic adventure	fantasy, fiction,	[illegible]	[illegible]	[illegible]

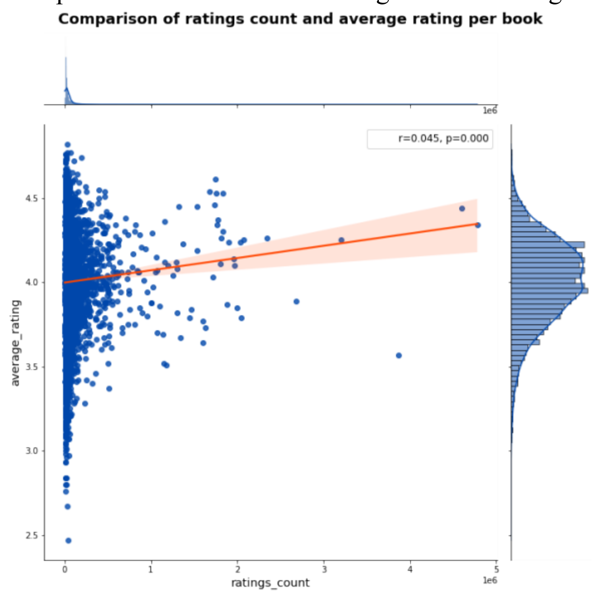
```
#(Books with most reviews)
book_sorted_ratings_count = books.sort_values('ratings_count', ascending=False)
book_sorted_ratings_count.head(10)
```

	book_id	title	authors	year	pages	description	genres	average_rating	ratings_count	books_count
0	1	The Hunger Games (The Hunger Games, #1)	Suzanne Collins	2008	374	winning means fame and fortunelosing means cer...	youngadult, fiction, fantasy, sciencefiction, ...	4.34	4780653	272
1	2	Harry Potter and the Sorcerer's Stone (Harry P...	J.K. Rowling, Mary GrandPré	1997	309	harry potters life is miserable his parents ar...	fantasy, fiction, youngadult, classics	4.44	4602479	491
2	3	Twilight (Twilight, #1)	Stephenie Meyer	2005	501	about three things i was absolutely positive ...	youngadult, fantasy, romance, fiction, paranormal	3.57	3866839	226
3	4	To Kill a Mockingbird	Harper Lee	1960	324	the unforgettable novel of a childhood in a sl...	classics, fiction, historicalfiction, youngadult	4.25	3198671	487
4	5	The Great Gatsby	F. Scott Fitzgerald	1925	200	alternate cover edition isbn isbn13 the great...	classics, fiction, historicalfiction, romance	3.89	2683664	1356
5	6	The Fault in Our Stars	John Green	2012	313	despite the tumorshrinking medical miracle tha...	youngadult, romance, fiction, contemporary	4.26	2346404	226
6	7	The Hobbit	J.R.R. Tolkien	1937	366	in a hole in the ground there lived a hobbit n...	fantasy, classics, fiction, youngadult	4.25	2071616	969
7	8	The Catcher in the Rye	J.D. Salinger	1951	277	the heronarrator of the catcher in the rye is ...	classics, fiction, youngadult	3.79	2044241	360
9	10	Pride and Prejudice	Jane Austen	1813	279	alternate cover edition of isbn since its imme...	classics, fiction, romance, historicalfiction	4.24	2035490	3455
8	9	Angels & Demons (Robert Langdon, #1)	Dan Brown	2000	736	worldrenowned harvard symbologist robert langd...	fiction, mystery, thriller, suspense, crime, h...	3.85	2001311	311

- When we sort book based on ratings_count, we found several books that have an average_rating lower than the mean (less than 4.002)

- When we sort book based on average_rating, we found several books lower number of reviews (ratings_count).
- Therefore we should make a new score calculation that also takes into account the average_rating and ratings_count.

6. How is the relationship between the number of ratings and the average rating ?



A book that is popular (has lots of ratings) is more likely to get a good rating. However, if we look at our data, the correlation between `average_rating` and `ratings_count` is not too big, which means that many popular books have low ratings.

7. Who is the author with most books ?

8. Who is the author with most books?

```
#First try with the simple count
author_book_count = books.groupby('authors')['title'].count().sort_values(ascending = False)
author_book_count.head()
```

```
authors
Stephen King        60
Nora Roberts        59
Dean Koontz         46
Terry Pratchett     42
Agatha Christie     39
Name: title, dtype: int64
```

Based in simple count, author with most books is Stephen King. However, in fact there are books written by more than one author, right?. So let's check them.

```
[ ] #take 'Stephen King' as an example
auth = []
books.authors.apply(lambda x: auth.append(x) if 'Stephen King' in x else [])
auth[:5]
```

```
['Stephen King',
 'Stephen King, Bernie Wrightson',
 'Stephen King',
 'Stephen King',
 'Stephen King']
```

Actually, there are several books written by more than one authors. These books are not included in the total books written by the author! I decided to pick only the first author to simplify our process.

```
3                    Harper Lee
4            F. Scott Fitzgerald
Name: authors, dtype: object
```

```python
splitted_authors = authors_list.apply(lambda x: pd.Series(x)).stack().reset_index(level=1, drop=True)
splitted_authors.name = 'authors'
splitted_authors.head()
```

```
0    Suzanne Collins
1       J.K. Rowling
1      Mary GrandPré
2    Stephenie Meyer
3         Harper Lee
Name: authors, dtype: object
```

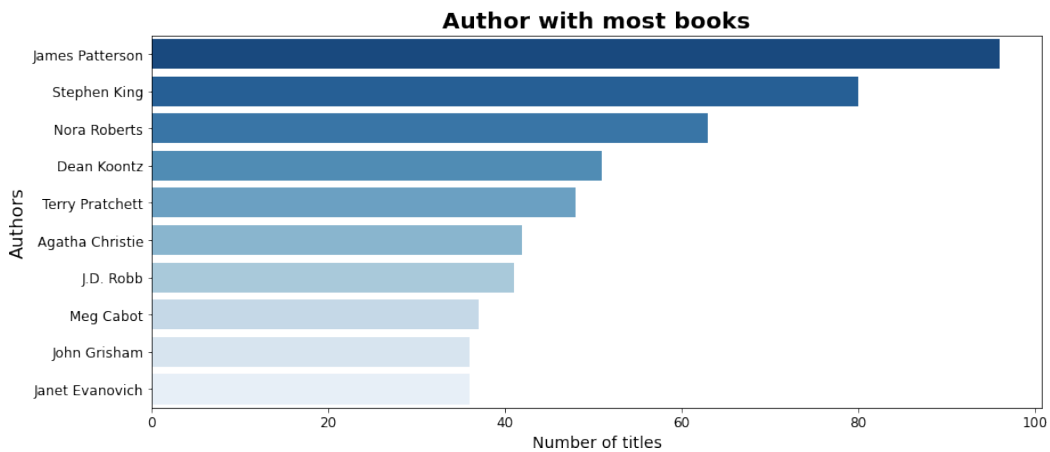
```python
#remove previous author column and join the new splitted authors
books = books.drop('authors', axis=1).join(splitted_authors)
```

```python
# Take the first author for each book and remove the rest
books = books.drop_duplicates(subset='book_id', keep='first')
```

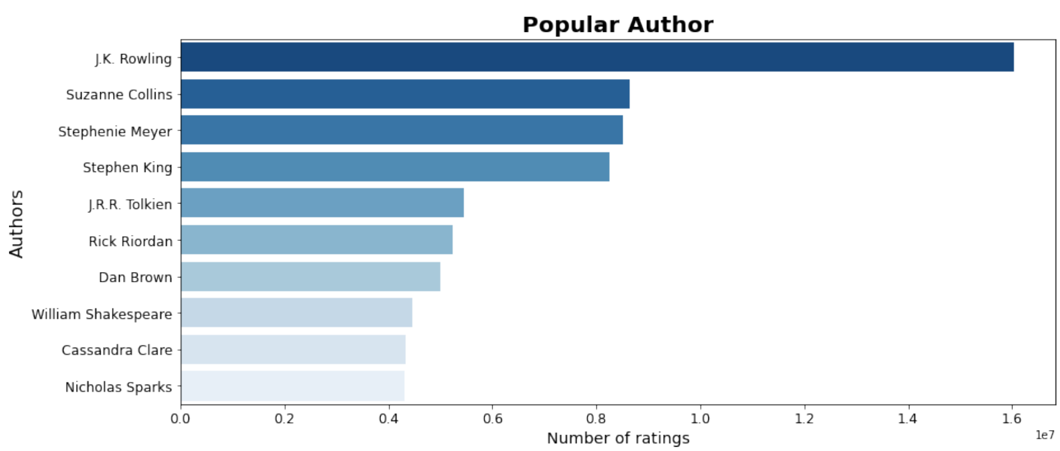
```python
#author with most books
book_count_real = books['authors'].value_counts()
book_count_real.head()
```

```
James Patterson    96
Stephen King       80
Nora Roberts       63
Dean Koontz        51
Terry Pratchett    48
Name: authors, dtype: int64
```

Now, instead of Stephen King, we found James Patterson is actually the author with the most books!

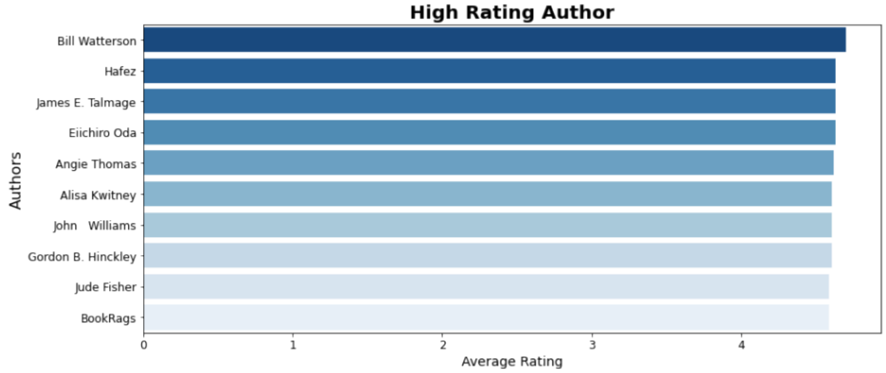


8. Who is the most popular author ?



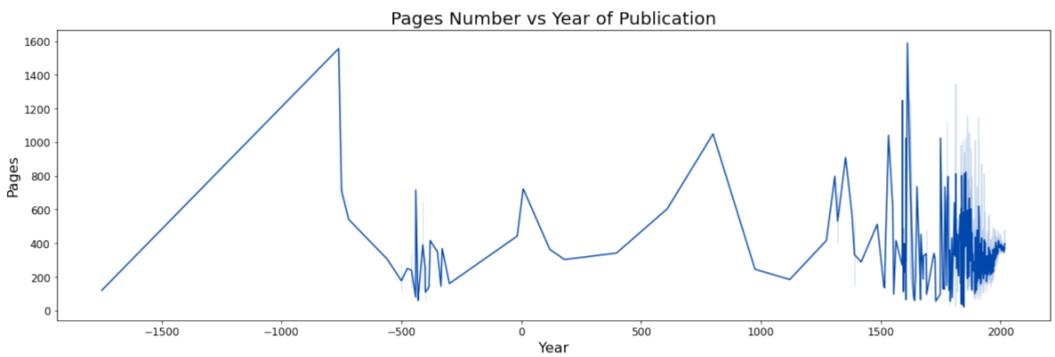
Even though James Patterson is the author with the most books, he is not the one who has the most ratings. We have J.K Rowling as the most ratings author with her 20 books in this dataset.

9. Who is the author that has good ratings book ?

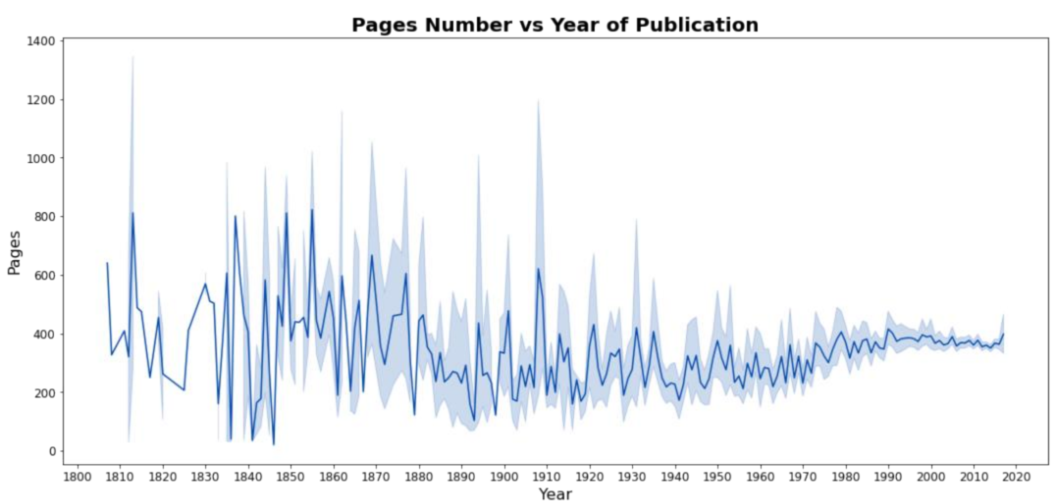


Bill Watterson is the one with the highest rating. But as we can see, there isn't much of a difference between the authors average rating.

11. How is the relationship between the number of pages and the year the book was published ?

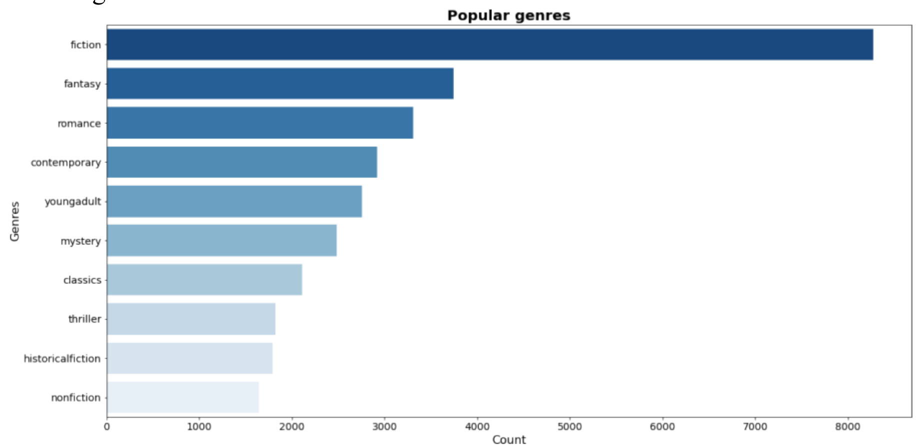


The range of year of publication is too large, therefore we need to check when the entry is more dense.



- Before 1900, the number of pages was randomly distributed. There are books that have more than 1000 pages but also books that have less than 100 pages. We can see that after 1900 the distribution started to stable, but it is less than 500 pages.
- After 1980, the trend also show slight increase. Most of recent books have around 400 pages.

12. What genre dominates the dataset ?



We can see that the majority (around 80%) of the books here are included in fiction books. The second rank goes to fantasy followed by romance. The difference between the first and second ranks is more than half. If you use a filtering method based on genre similarities, it is very unlikely that the engine will recommend non-fiction books.

8. Model Performance Evaluation

Collaborative Filtering

The notebook details the development and evaluation of six distinct models using Surprise:

- Normal Predictor: A baseline model providing predictions without considering user-item interactions.
- KNN (Memory-based): A memory-based model using collaborative filtering to find similar users or items.
- SVD (Model-based): A model-based approach employing Singular Value Decomposition for recommendation.
- SVD++ (Model-based): An extension of SVD, incorporating implicit feedback for enhanced recommendations.
- Baseline: Utilizing baseline estimates for predictions.
- NMF (Non-Negative Matrix Factorization): A collaborative filtering model based on matrix factorization.

a. Normal Predictor

The implemented recommendation model utilizes the Normal Predictor algorithm, which is designed to provide baseline predictions by estimating ratings based on the mean rating of items. In the context of collaborative filtering, Normal Predictor serves as a benchmark model, offering a simple yet effective approach for making predictions in the absence of intricate user-item interactions. The evaluation of the model is conducted on a dataset divided into 5 folds for cross-validation, and the root mean square error (RMSE) is employed as the performance metric.

The RMSE results for each fold, along with the mean and standard deviation, showcase the model's consistency in predicting ratings. The average RMSE across the folds is 1.3230, with a relatively low standard deviation of 0.0008, indicating the stability of the predictions. The fit time, representing the duration taken for the model to train on the dataset, ranges from 3.14 to 4.78 seconds across the folds, with an average of 3.80 seconds. Similarly, the test time, denoting the time taken to make predictions on the test set, shows consistency, averaging 10.83 seconds.

The train and test RMSE values, reported separately, provide insights into the model's performance on both the training and testing data. The small difference between the two RMSE values (1.3233 for training and 1.3236 for testing) suggests that the model generalizes well to unseen data, demonstrating its effectiveness in making accurate predictions beyond the training set.

Strengths:

1. Simplicity: The Normal Predictor model offers a simple and intuitive approach to recommendation by relying on mean ratings. This simplicity makes it easy to implement and understand, making it an effective baseline model.

2. Efficiency: With relatively low computational times for both training and testing phases, the Normal Predictor is efficient and scalable, making it suitable for large datasets.

3. Consistency: The model demonstrates consistent performance across the 5-fold cross-validation, as indicated by the narrow range of RMSE values and low standard deviation. This reliability suggests that the model can provide stable predictions.

Weaknesses:

1. Limited Complexity: The reliance on mean ratings limits the model's ability to capture intricate user preferences or item nuances. It may struggle to provide accurate recommendations in scenarios where more sophisticated algorithms are necessary.

2. Sparse Data Challenges: The Normal Predictor may face challenges when dealing with sparse or incomplete datasets. Its effectiveness is contingent on the availability of mean ratings, and it may struggle to make accurate predictions in situations where data sparsity is pronounced.

3. Lack of Personalization: Due to its simplistic nature, the Normal Predictor does not incorporate personalized user-item interactions. It treats all users and items alike, potentially leading to suboptimal recommendations for individual users with distinct preferences.

Book Recommendation System - Part 2 (Modeling).ipynb

File Edit View Insert Runtime Tools Help Last saved at 00:45

a. Normal Predictor

Algorithm predicting a random rating based on the distribution of the training set, which is assumed to be normal.

```
# prepare train data
data.raw_ratings = train_ratings

#select algorithm
npred = NormalPredictor()
```

```
%%time

#cross validation for train data
np_result = cross_validate(npred, data, measures=['RMSE'], cv=5, verbose=True, n_jobs=2)
```

```
Evaluating RMSE of algorithm NormalPredictor on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3232  1.3233  1.3220  1.3223  1.3243  1.3230  0.0008
Fit time          3.17    4.75    3.14    4.78    3.14    3.80    0.79
Test time         12.99   10.18   12.31   10.26   8.39    10.83   1.65
CPU times: user 1min 32s, sys: 3.55 s, total: 1min 35s
Wall time: 1min 56s
```

In order to make prediction on test data, we have to retrain whole train set first.

```
%%time
#retrain whole train test
trainset = data.build_full_trainset()
npred.fit(trainset)
```

In order to make prediction on test data, we have to retrain whole train set first.

```
%%time
#retrain whole train test
trainset = data.build_full_trainset()
npred.fit(trainset)

# Compute RMSE on trainset (without fold)
np_train_pred = npred.test(trainset.build_testset())
print('Train RMSE:')
train_rmse = accuracy.rmse(np_train_pred)

#compute RMSE on testset
testset = data.construct_testset(test_ratings)
np_test_pred = npred.test(testset)
print('Test RMSE:')
test_rmse = accuracy.rmse(np_test_pred)
```

```
Train RMSE:
RMSE: 1.3233
Test RMSE:
RMSE: 1.3236
CPU times: user 1min 37s, sys: 1.14 s, total: 1min 38s
Wall time: 1min 38s
```

```
# uncomment this code to dump the calculation result for future use
# dump.dump('./dump_np', np_test_pred, npred)
```

Now we can get recommendation for user 12874

```
get_recommendation_npred(12874, 5)
```

	book_id	title	authors	average_rating	ratings_count
0	1	The Hunger Games (The Hunger Games, #1)	Suzanne Collins	4.34	4780653
5	6	The Fault in Our Stars	John Green	4.26	2346404
24	28	Lord of the Flies	William Golding	3.64	1605019
39	45	Life of Pi	Yann Martel	3.88	1003228
40	46	Water for Elephants	Sara Gruen	4.07	1068146

b- KNN Basic

The KNN Basic recommendation model utilizes the k-nearest neighbors algorithm, employing cosine similarity matrices to gauge the similarity between users or items. The iterative process of computing the cosine similarity matrix is a key aspect of the model's training. The subsequent evaluation involves a 5-fold cross-validation on the dataset, with the root mean square error (RMSE) serving as the performance metric.

The RMSE results across the folds reveal the model's effectiveness, consistently yielding low values. The average RMSE is 0.8876, with a standard deviation of 0.0009, indicating precision and stability in predictions. Fit time, representing the duration of model training, varies from 21.83 to 23.65 seconds across folds, with an average of 23.16 seconds. Test time, denoting the duration to make predictions, is relatively higher, averaging 110.24 seconds.

Train and test RMSE values highlight the model's generalization capability, with a small difference between training (0.8001) and testing (0.8851) RMSE values. This suggests that KNN Basic effectively captures underlying patterns without overfitting to the training set.

Strengths:

1. High Precision: KNN Basic demonstrates high precision with consistently low RMSE values across cross-validation folds, indicating accurate predictions and reliable recommendations.

2. Effective Generalization: The model generalizes effectively to unseen data, evident in the small difference between training and testing RMSE values, suggesting robust performance beyond the training set.

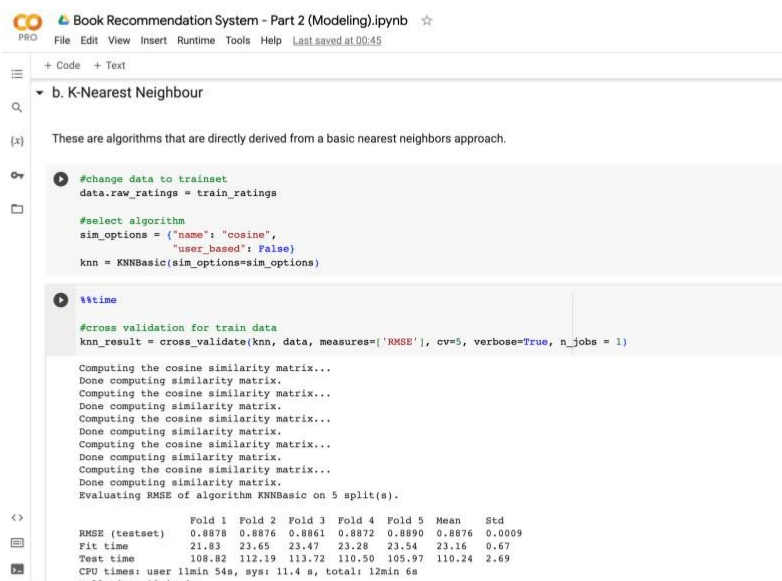
3. Robust Similarity Measurement: The use of cosine similarity enhances the model's ability to capture complex relationships in the data, especially in scenarios where user preferences exhibit non-linear patterns.

Weaknesses:

1. Computational Intensity: KNN Basic exhibits higher computational times, particularly during the test phase, potentially limiting its scalability for large datasets or real-time applications.

2. Sensitivity to Noise: The model can be sensitive to noisy or irrelevant features in the data, leading to suboptimal recommendations in the presence of outliers.

3. Cold Start Problem: KNN Basic may face challenges in scenarios with new users or items (cold start problem), relying on existing user-item interactions for accurate predictions.



```
] get_recommendation_knn(12874, 5)
```

	book_id	title	authors	average_rating	ratings_count
27	31	The Help	Kathryn Stockett	4.45	1531753
89	98	The Girl Who Played with Fire (Millennium, #2)	Stieg Larsson	4.22	563994
103	114	Tuesdays with Morrie	Mitch Albom	4.06	556518
118	132	The Five People You Meet in Heaven	Mitch Albom	3.90	449501
126	140	The Girl Who Kicked the Hornet's Nest (Millenn...	Stieg Larsson	4.20	443951

c- Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) recommendation model is evaluated using a 5-fold cross-validation approach. The evaluation metrics, specifically the root mean square error (RMSE), provide insights into the model's performance across different subsets of the dataset.

The RMSE results across the folds indicate the model's high precision, with consistently low values. The average RMSE is 0.8500, and the standard deviation is 0.0009, highlighting the stability and accuracy of the predictions. Fit time, representing the duration of model training, varies from 32.70 to 46.56 seconds across folds, with an average of 42.53 seconds. Test time, denoting the duration to make predictions, ranges from 9.49 to 12.47 seconds, with an average of 10.30 seconds.

Train and test RMSE values further demonstrate the model's generalization capability, with a small difference between training (0.6441) and testing (0.8386) RMSE values. This indicates effective learning from the training set and accurate predictions on unseen data.

Strengths:

1. High Precision: SVD exhibits high precision with consistently low RMSE values across cross-validation folds, signifying accurate predictions and reliable recommendations.

2. Efficient Training: The model demonstrates efficient training times, averaging 42.53 seconds, making it suitable for datasets of moderate size.

3. Generalization Capability: SVD effectively generalizes to unseen data, as evidenced by the small difference between training and testing RMSE values, indicating robust performance beyond the training set.

Weaknesses:

1. Longer Training Time: Although training times are reasonable, they are longer compared to some other models, such as KNN Basic. This could be a limitation for real-time or large-scaleapplications.

2. Test Time Variability: Test times exhibit variability across folds, with a range of 9.49 to 12.47 seconds. This variability may impact the model's suitability for applications with strict latency requirements.

3. Sensitivity to Hyperparameters: SVD's performance can be sensitive to hyperparameters, and fine-tuning may be necessary to optimize its effectiveness for specific datasets.

c. Singular Value Decomposition (SVD)

The famous SVD algorithm, as popularized by Simon Funk during the Netflix Prize. SVD finds the latent factors associated with some matrix. SVD will decompose user-rating matrix into matrices that represents latent user-user features and item-item features.

```
#change data to trainset
data.raw_ratings = train_ratings

#select algorithm
svd = SVD(random_state=0)
```

```
%%time

#cross validation for train data
svd_result = cross_validate(svd, data, measures=["RMSE"], cv=5, verbose=True, n_jobs = 2)
```

```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8503  0.8500  0.8484  0.8503  0.8510  0.8500  0.0009
Fit time          46.56   45.32   46.15   41.94   32.70   42.53   5.18
Test time         9.49    12.47   9.49    10.48   9.61    10.30   1.14
CPU times: user 1min 32s, sys: 3.3 s, total: 1min 35s
Wall time: 3min 14s
```

```
%%time

#retrain whole train test
trainset = data.build_full_trainset()
svd.fit(trainset)

# Compute RMSE on trainset (without fold)
svd_train_pred = svd.test(trainset.build_testset())
print('Train RMSE:')
train_rmse = accuracy.rmse(svd_train_pred)

#compute RMSE on testset
testset = data.construct_testset(test_ratings)
svd_test_pred = svd.test(testset)
print('Test RMSE:')
test_rmse = accuracy.rmse(svd_test_pred)
```

```
Train RMSE:
RMSE: 0.6441
Test RMSE:
RMSE: 0.8386
CPU times: user 2min 48s, sys: 1.35 s, total: 2min 49s
Wall time: 2min 49s
```

```
get_recommendation_svd(12874, 5)
```

	book_id	title	authors	average_rating	ratings_count
27	31	The Help	Kathryn Stockett	4.45	1531753
194	215	Ready Player One	Ernest Cline	4.31	349423
226	250	Wonder	R.J. Palacio	4.43	228538
390	425	Go Ask Alice	Beatrice Sparks	3.77	196677
1698	1808	Morning Star (Red Rising, #3)	Pierce Brown	4.50	47736

d. Singular Value Decomposition with Implicit Feedback (SVDpp)

The Singular Value Decomposition with Implicit Feedback (SVDpp) recommendation model is assessed using a 5-fold cross-validation method. The evaluation metrics, particularly the root mean square error (RMSE), provide a comprehensive understanding of the model's performance across various data subsets.

The RMSE results across the folds indicate the model's high precision, consistently yielding low values. The average RMSE is 0.8315, and the standard deviation is 0.0010, highlighting the stability and accuracy of the predictions. Fit time, representing the duration of model training, varies from 438.23 to 452.81 seconds across folds, with an average of 447.40 seconds. Test time, denoting the duration to make predictions, shows minimal variability, ranging from 66.31 to 66.61 seconds, with an average of 66.48 seconds.

Train and test RMSE values further underscore the model's generalization capability, with a small difference between training (0.7085) and testing (0.8238) RMSE values. This suggests effective learning from the training set and accurate predictions on unseen data.

Strengths:

1. High Precision: SVDpp demonstrates high precision with consistently low RMSE values across cross-validation folds, indicating accurate predictions and reliable recommendations.

2. Improved Generalization: The model shows enhanced generalization to unseen data, as evidenced by the small difference between training and testing RMSE values, indicating robust performance beyond the training set.

3. Implicit Feedback Handling: SVDpp incorporates implicit feedback in its training process, allowing it to capture more nuanced user preferences and potentially improving recommendations in scenarios where explicit feedback is sparse.

Weaknesses:

1. Longer Training Time: SVDpp exhibits longer training times compared to other models, with fit times averaging 447.40 seconds. This may be a limitation for real-time or large-scale applications.

2. Resource Intensive: The model requires substantial computational resources, as indicated by the long fit times, making it less suitable for applications with constraints on computational resources.

3. Potential Sensitivity to Hyperparameters: SVDpp's performance may be sensitive to hyperparameters, necessitating careful tuning to optimize its effectiveness for specific datasets.

d. SVD++

The singular value decomposition (SVD)++ algorithm is employed as an optimized SVD algorithm to enhance the accuracy of prediction by generating implicit feedback.

```
#change data to trainset
data.raw_ratings = train_ratings

#select algorithm
svdpp = SVDpp(random_state=0)
```

```
%%time

svdpp_result = cross_validate(svdpp, data, measures=["RMSE"], cv=5, verbose=True, n_jobs = 2)
```

```
Evaluating RMSE of algorithm SVDpp on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8325  0.8315  0.8298  0.8313  0.8326  0.8315  0.0010
Fit time          452.81  450.51  448.53  446.92  438.23  447.40  4.99
Test time         66.38   66.49   66.61   66.60   66.31   66.48   0.12
CPU times: user 1min 33s, sys: 3.76 s, total: 1min 36s
Wall time: 26min 19s
```

```
%%time

#retrain whole train test
trainset = data.build_full_trainset()
svdpp.fit(trainset)

# Compute RMSE on trainset (without fold)
svdpp_train_pred = svdpp.test(trainset.build_testset())
print('Train RMSE:')
train_rmse = accuracy.rmse(svdpp_train_pred)

#compute RMSE on testset
testset = data.construct_testset(test_ratings)
svdpp_test_pred = svdpp.test(testset)
print('Test RMSE:')
test_rmse = accuracy.rmse(svdpp_test_pred)
```

```
Train RMSE:
RMSE: 0.7085
Test RMSE:
RMSE: 0.8238
CPU times: user 22min 28s, sys: 4.05 s, total: 22min 32s
Wall time: 22min 29s
```

```
[ ] get_recommendation_svdpp(12874, 5)
```

	book_id	title	authors	average_rating	ratings_count
27	31	The Help	Kathryn Stockett	4.45	1531753
597	642	World Without End (The Kingsbridge Series, #2)	Ken Follett	4.23	128715
1243	1328	I Am Pilgrim (Pilgrim, #1)	Terry Hayes	4.23	49740
4310	4778	The Holy Bible: English Standard Version	Anonymous	4.66	17863
5302	5990	الطنطورية	Radwa Ashour	4.32	13931

e. Non-negative Matrix Factorization (NMF)

The Non-negative Matrix Factorization (NMF) recommendation model is evaluated using a 5-fold cross-validation approach, providing insights into its performance across different subsets of the dataset. The evaluation metrics, specifically the root mean square error (RMSE), offer a measure of the model's predictive accuracy.

The RMSE results across the folds indicate the model's precision, consistently yielding low values. The average RMSE is 0.8708, and the standard deviation is 0.0004, emphasizing the stability and accuracy of the predictions. Fit time, representing the duration of model training, ranges from 42.57 to 44.84 seconds across folds, with an average of 43.47 seconds. Test time, denoting the duration to make predictions, varies from 6.00 to 6.78 seconds, with an average of 6.36 seconds.

Strengths:

1. Predictive Accuracy: NMF demonstrates high precision with consistently low RMSE values across cross-validation folds, indicating accurate predictions and reliable recommendations.

2. Computational Efficiency: The model exhibits reasonable fit and test times, averaging 43.47 seconds and 6.36 seconds, respectively, making it suitable for datasets of moderate size and real-time applications.

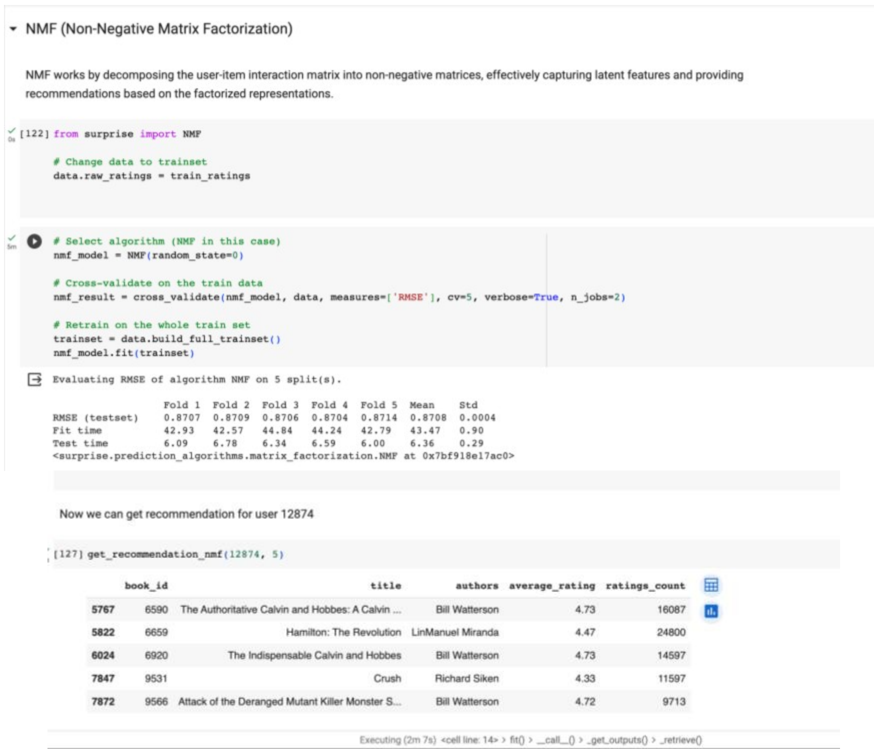
3. Interpretability: NMF generates factorized matrices that can be interpreted as latent features, providing insights into the underlying structure of the user-item interactions.

Weaknesses:

1. Cold Start Problem: Similar to other matrix factorization models, NMF may face challenges in scenarios with new users or items (cold start problem), as it relies on existing user-item interactions for accurate predictions.

2. Limited to Non-negative Values: NMF assumes non-negativity in the input data and factorized matrices, which might not be suitable for datasets with negative feedback or interactions.

3. Sensitivity to Initialization: NMF's performance may be sensitive to the choice of initial values, requiring careful initialization for optimal results.



NMF (Non-Negative Matrix Factorization)

NMF works by decomposing the user-item interaction matrix into non-negative matrices, effectively capturing latent features and providing recommendations based on the factorized representations.

```
[122] from surprise import NMF

# Change data to trainset
data.raw_ratings = train_ratings
```

```
# Select algorithm (NMF in this case)
nmf_model = NMF(random_state=0)

# Cross-validate on the train data
nmf_result = cross_validate(nmf_model, data, measures=['RMSE'], cv=5, verbose=True, n_jobs=2)

# Retrain on the whole train set
trainset = data.build_full_trainset()
nmf_model.fit(trainset)
```

```
Evaluating RMSE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8707  0.8709  0.8706  0.8704  0.8714  0.8708  0.0004
Fit time          42.93   42.57   44.84   44.24   42.79   43.47   0.90
Test time         6.09    6.78    6.34    6.59    6.00    6.36    0.29
<surprise.prediction_algorithms.matrix_factorization.NMF at 0x7bf918e17ac0>
```

Now we can get recommendation for user 12874

```
[127] get_recommendation_nmf(12874, 5)
```

	book_id	title	authors	average_rating	ratings_count
5767	6590	The Authoritative Calvin and Hobbes: A Calvin ...	Bill Watterson	4.73	16087
5822	6659	Hamilton: The Revolution	LinManuel Miranda	4.47	24800
6024	6920	The Indispensable Calvin and Hobbes	Bill Watterson	4.73	14597
7847	9531	Crush	Richard Siken	4.33	11597
7872	9566	Attack of the Deranged Mutant Killer Monster S...	Bill Watterson	4.72	9713

Executing (2m 7s) <cell line: 14> › fit() › __call__() › _get_outputs() › _retrieve()

Model Comparison:

The comparison of different recommendation models based on RMSE, duration, and memory usage provides valuable insights into their performance and practical considerations. Here's an overview of the findings:

1. **Normal Predictor (NP):**
    - RMSE: 1.32
    - Duration: 3 minutes and 40 seconds
    - Memory Use: 8.70

NP serves as a baseline model with relatively high RMSE, indicating a simplistic approach. The model's quick training time and moderate memory use make it suitable for initial explorations but may not deliver highly accurate recommendations.

2. **K-Nearest Neighbors (KNN):**
    - RMSE: 0.89
    - Duration: 24 minutes and 45 seconds
    - Memory Use: 16.00

KNN exhibits improved predictive accuracy compared to NP but at the cost of increased training time and higher memory usage. This suggests that KNN may be suitable for scenarios where accuracy is a priority, and resource constraints are not stringent.

3. **Singular Value Decomposition (SVD):**
    - RMSE: 0.84
    - Duration: 7 minutes and 5 seconds
    - Memory Use: 8.80

SVD showcases a good balance between accuracy and efficiency. Its lower RMSE, moderate training time, and reasonable memory use make it a practical choice for recommendation systems, especially for medium-sized datasets.

4. **SVD++:**
    - RMSE: 0.82
    - Duration: 1 hour, 9 minutes, and 50 seconds
    - Memory Use: 8.40

SVD++ delivers enhanced accuracy but at the expense of significantly longer training times. The model's memory use remains comparable to SVD, suggesting that the increased complexity contributes to improved recommendations.

5. **Non-negative Matrix Factorization (NMF):**
    - RMSE: 0.87
    - Duration: 43.47 seconds
    - Memory Use: 6.36 seconds
    -

NMF provides a trade-off between accuracy and efficiency. With a slightly higher RMSE than SVD and SVD++, its quick training time and low memory use make it suitable for real-time applications and datasets where computational resources are limited.

	Model	RMSE	Duration	Memory Use
0	NP	1.32	3m 40s	8.70
1	KNN	0.89	24m 45s	16.00
2	SVD	0.84	7m 5s	8.80
3	SVD++	0.82	1h 9m 50s	8.40
4	NMF	0.87	43.47s	6.36s

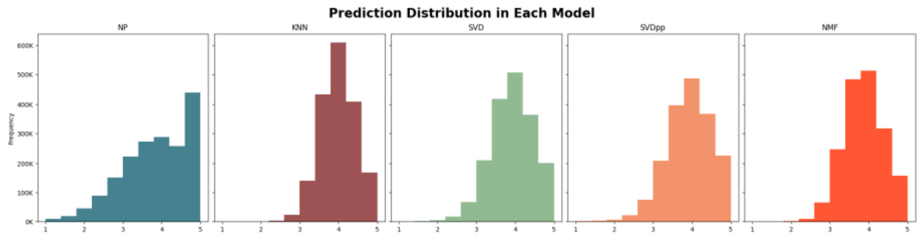
By comparing the metric evaluation of four model, we get the lowest RMSE in SVD++. However, although SVD++ shows lower RMSE results, it takes a very long time to do the calculations. If we look at the rating predictions, the distribution of ratings on SVD and SVD++ is not much

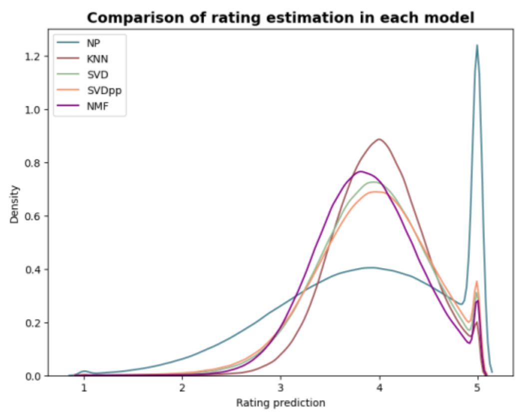
In summary, while SVD++ achieves the lowest RMSE, its extended training time and computational intensity may limit its practicality in real-time or resource-constrained environments. The rating predictions between SVD and SVD++ show minimal differences, questioning the necessity of SVD++'s increased complexity.

K-nearest neighbors (KNN) exhibits longer runtimes due to its inherent pairwise similarity computations, which become computationally intensive as the dataset size increases. The memory demands of KNN are substantial, requiring storage for a large dataset and scanning it for nearest neighbors during recommendations.

In practical terms, the choice of a recommendation model should carefully weigh predictive accuracy, computational efficiency, and memory requirements. While SVD and KNN offer reasonable compromises, with SVD striking a balance and KNN providing accurate predictions but with longer runtimes and high memory use, the selection depends on specific application needs and constraints.

Comparing Prediction Distribution





As we can see, Normal Predictor model predicts higher ratings more often, meanwhile KNN predictions are concentrated around the mean. For SVD, SVD++ and NMF, the ratings are morefairly distributed.

2. Content Based Recommendation System

a. *Recommendation based on Cosine Similarity*

To personalise our recommendations, we will measure the cosine similarity between books. The steps are:

1. Make new column which consist of authors, title, genres and description of each book.
2. Use TFIDFVectorizer to convert our data to vector
3. Calculate the cosine similarity score for all books
4. User will input their favorite book, we will sort book that more similar to the input
5. Recommend a user books

Since we use TFIDFVectorizer, the dot product will directly give us the cosine similarity score. Therefore, we will use sklearn's linear_kernel instead of cosine_similarities since it is much faster.

```
                          ).str.join(' '))

    tf_content = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')
    tfidf_matrix = tf_content.fit_transform(books['content'])
    cosine = linear_kernel(tfidf_matrix, tfidf_matrix)
    index = pd.Series(books.index, index=books['title'])

    return cosine, index
```

```
def content_recommendation(books, title, n=5):
    cosine_sim, indices = content(books)
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:n + 1]
    book_indices = [i[0] for i in sim_scores]
    return books[['book_id', 'title', 'authors', 'average_rating', 'ratings_count']].iloc[book_indices]
```

```
content_recommendation(b, '1984')
```

	book_id	title	authors	average_rating	ratings_count
795	846	Animal Farm / 1984	George Orwell	4.26	116197
2048	2187	We	Yevgeny Zamyatin	3.95	40020
3670	4004	Homage to Catalonia	George Orwell	4.14	22227
6857	8056	1Q84 #1-2 (1Q84, #1-2)	Haruki Murakami	4.07	8342
4915	5510	The Far Side Gallery	Gary Larson	4.42	20022

Notice that the system recommends a book with average_rating (3.95) lower than average and book with low `ratings_count` (8342). We will try to improve our recommendation by adding popularity-rating filter.

Strengths:

1. User Independence: Content-based systems using cosine similarity work well for individual users, requiring minimal reliance on collaborative data.
2. Transparency: Cosine similarity provides transparent and understandable recommendations, enhancing user trust.
3. Cold Start Problem Mitigation: Cosine similarity-based systems effectively recommend new items with sparse user interactions.
4. Feature Flexibility: Cosine similarity can handle diverse feature types, offering flexibility in item representation.

Weaknesses:

1. Limited Serendipity: Content-based systems may struggle to introduce users to unexpected items, lacking serendipity.
2. Dependency on Feature Quality: The effectiveness of cosine similarity relies on the quality and relevance of selected features.
3. Over-Specialization: There's a risk of over-specialization, potentially reducing diversity in recommendations.
4. Scalability Challenges: Calculating cosine similarity becomes computationally expensive in large-scale systems.

2. *Content Based + Popularity-Rating Filter*

The mechanism to remove books with low ratings has been added on top of the content based filtering. This system will return books that are similar to your input, are popular and have high ratings. However, in this filter, our cutoff will be the quantile 75. In order for a book to appear in the recommendation, it must be ranked in top 25 similar and receive at least 75% weight score of the other books on the list (around 800 ratings).

```
def improved_recommendation(books, title, n=5):
    cosine_sim, indices = content(books)
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    book_indices = [i[0] for i in sim_scores]
    books2 = books.iloc[book_indices][['book_id', 'title', 'authors', 'average_rating', 'ratings_count']]

    v = books2['ratings_count']
    m = books2['ratings_count'].quantile(0.75) #here the minimum rating is quantile 75
    R = books2['average_rating']
    C = books2['average_rating'].median()
    books2['new_score'] = (v/(v+m) * R) + (m/(m+v) * C)

    high_rating = books2[books2['ratings_count'] >= m]
    high_rating = high_rating.sort_values('new_score', ascending=False)

    return high_rating[['book_id', 'title', 'authors', 'average_rating', 'ratings_count','new_score']].head(n)
```

```
improved_recommendation(b, '1984')
```

	book_id	title	authors	average_rating	ratings_count	new_score
795	846	Animal Farm / 1984	George Orwell	4.26	116197	4.20
759	809	Brave New World / Brave New World Revisited	Aldous Huxley	4.16	108124	4.13
1044	1120	Aesop's Fables	Aesop	4.05	88508	4.05
8316	2375	Tinker, Tailor, Soldier, Spy	John le Carré	4.04	40871	4.04
604	649	1Q84	Haruki Murakami	3.89	125195	3.93

Connected to Python 3 Google Compute Engine backend

Strengths:

1. User Engagement: Content-based systems with a popularity-rating filter consider both item features and user preferences, enhancing user engagement.
2. Simplicity: The inclusion of a popularity-rating filter adds simplicity to the recommendation process, prioritizing items with broad appeal.
3. Cold Start Problem Mitigation: The combined approach addresses the cold start problem by leveraging item features and overall popularity.
4. Customization: Users receive recommendations based on both content relevance and the popularity of items, providing a balanced and customized experience.

Weaknesses:

1. Homogeneity: The popularity-rating filter may lead to recommendations that favor mainstream or popular items, potentially limiting diversity.
2. Limited Personalization: The system may struggle to provide highly personalized recommendations, as popularity influences the suggestions.
3. Sensitivity to Ratings: The effectiveness depends on the quality and distribution of user ratings, and noisy or biased ratings can impact results.
4. Scalability Challenges: Large-scale implementations may face challenges in handling the computational load associated with popularity-based filtering.

This hybrid approach combines content-based and popularity-rating filtering to offer a recommendation system that considers both item features and overall popularity, striking a balance between personalization and broad appeal. Conclusion:

9. Conclusion

1- <u>Collaboration Filtering</u>

I built a recommender with 5 algorithms: Normal Predictor, KNN, SVD, and SVD++ and NMF.

1. **RMSE (Root Mean Square Error):**
    - o Lower RMSE values indicate better predictive performance.
    - o SVD++ has the lowest RMSE (0.82), suggesting it performs the best in terms of accuracy.
2. **Duration:**
    - o Lower duration values are generally preferable as they indicate faster model training.
    - o NMF has the lowest training duration (43.47s), indicating it is the fastest to train.
3. **Memory Use:**
    - o Lower memory usage is usually desirable, as it allows for more efficient resource utilization.
    - o NMF has the lowest memory usage (6.36), suggesting it is the most memory-efficient.

Based on specific priorities:

- If accuracy is the top priority, SVD++ with the lowest RMSE might be the best choice.
- If training speed is crucial, NMF with the lowest duration might be preferable.
- If memory efficiency is a key consideration, NMF with the lowest memory use would be a good option.

It's important to strike a balance between these factors depending on your specific requirements and constraints. Consider the trade-offs and choose the model that aligns best with your goals for the book recommendation system.

To strike a balance between accuracy, training speed, and memory efficiency, we might consider SVD (Singular Value Decomposition) as the best model.

1. Reasonable RMSE: While SVD++ has the lowest RMSE, SVD also has a good RMSE of 0.84. It strikes a balance between accuracy and computational efficiency.
2. Moderate Training Duration: SVD has a reasonable training duration of 7 minutes and 5 seconds. It's faster than SVD++, making it a good compromise in terms of training speed.
3. Moderate Memory Use: With a memory use of 8.80, SVD's memory consumption is reasonable, indicating a balanced use of resources.

Considering these factors, SVD seems to offer a good compromise between accuracy, training speed, and memory efficiency.

2- Content based filtering

Recommendations based on title, authors, description, and genre using cosine similarity have been made. To provide a balance of book recommendations, an additional popularity-rating filter has been added. This method is suitable for people who are looking for books that are similar to their favorite books, but this system cannot capture tastes and provide recommendations across genres. By applying a content based model, instead of having to rate 30 books to start the recommendation engine, users can just pick one book they liked for Goodreads to provide good recommendations for new users, making the process easier.

References

1. https://www.kaggle.com/datasets/zygmunt/goodbooks-10k
2. https://www.cs.cornell.edu/people/tj/publications/joachims_02c.pdf
3. https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models- performance-230449055f15
4. https://electronicsmarket.org/2017/12/28/an-intuitive-guide-to-nmf/