CS 7641 – ML Group 36 Final Report

Michael Anderson, Nathan Brodie, Irene Feijoo, Navya Gautam, Shriya Purohit

Background

This project aims to develop a book recommendation algorithm using three different recommendation ML models based on previous research and literature as described in the Problem Definition section. Inspired by Fayyaz et al.'s work [8], it addresses challenges like data sparsity and varying user preferences. The Amazon Books Review Kaggle dataset, with over 212K book descriptions and 3M reviews of user-generated content, is used to ensure accurate and up-to-date recommendations (a more detailed description of the dataset will be included in the Data Preprocessing section). Please find here a link to the dataset used: (https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews)

Problem Definition

In an era where 23% of American adults haven't read a book in the past year, and social media dominates attention, an effective book recommendation algorithm could help revive reading culture by reducing the “boredom” barrier of investing time in an unappealing book [6]. This project proposes three solutions:

1. A hybrid of two different unsupervised filtering models: collaborative and content-based, which differs from traditional filtering methods by improving relevance and user satisfaction by addressing limitations in standalone methods, including user behavior and content diversity [5].
2. A hybrid KNN filtering recommendation system, which also effectively combines collaborative and content-based methods, enhancing recommendation accuracy by leveraging user-item interactions and item feature similarity [4].
3. A neural network-based recommendation systems leverage deep learning techniques, capturing complex user-item relationships and feature interactions, to provide highly personalized recommendations and improve accuracy over traditional algorithms [2].

Unlike Netflix and Spotify’s deep learning techniques, our approach is tailored to books, accounting for unique parameters and the nature of literature, which demands different recommendation criteria.

Methods

Data Preprocessing

Cleaning the Goodreads review data from Amazon Books was a complex task, primarily due to the dataset's size of over 200K books and 3M reviews. Managing such a large volume of information meant we had to tackle the cleaning process in stages.

We began with the core book attributes, including Title, Description, Authors, Publisher, Published Date, Categories, and Ratings Count. Irrelevant fields, such as links and images, were dropped to streamline the dataset. For text-heavy fields, we ensured consistent formatting, removed extraneous characters, replaced null values with placeholders, and translated non-English entries into English using 'langdetect' and 'googletrans'. Some descriptions were found in other languages, like German, Spanish, and French, so this step was essential for creating a cohesive dataset.

One of the most challenging aspects of this process was clustering similar titles and authors. Many entries had slight variations in titles and author names, often due to differences in punctuation or capitalization (e.g., "Dr Seuss: American Icon" vs. "Dr Seuss - American Icon"). Despite being identical books, these entries were counted separately because of the inconsistencies in data entry. To resolve this, we leveraged OpenRefine's Key Collision clustering with the Fingerprint keying function, which helped identify and merge 2,402 clusters of titles, with the largest cluster containing eight variations of the same title [7]. OpenRefine was leveraged because it is the most efficient and well-trained for finding small variations in title. When trying to cluster in Python, much larger title variations were included in a cluster rendering it invalid. This cleaning process was critical for ensuring accurate and meaningful analysis of the book data.

After the initial cleaning, we merged the dataset with the 3M reviews, which included key attributes like Book ID, Title, User ID, Profile Name, Review Helpfulness, Review Score, Review Time, Review Summary, and Review Text. We standardized text and date fields and converted review helpfulness to a percentage, representing the ratio of upvotes to total votes (e.g., 4/7, 8/8).

To reflect the newly clustered title-author combinations, we recreated Book ID as a numeric identifier, reducing the unique IDs from 212K to 202K. Then, we counted the total reviews per book and per user, aiming to keep only books with at least 15 reviews and reviewers who had reviewed at least 15 books. Balancing these thresholds wasn't straightforward—deleting reviewers below the threshold occasionally dropped book review counts below 15. After adjustments, we ended up with a final dataset of 486K reviews and 7K distinct books.

Finally, we applied word embedding vectorization to key fields such as Review Text, Review Summary, Book Description, and Book Categories. These embeddings will be crucial for figuring out what to recommend to new users moving forwards.

Once the data was cleaned, we ran some initial data visualization analysis to identify trends and validate certain assumptions that were the base of the models we were planning to use. For instance, we wanted to make sure that our data was independent and not correlated. We ran a correlation plot on all the numerical data and label encoding version of the categories and author data. This is what we found:

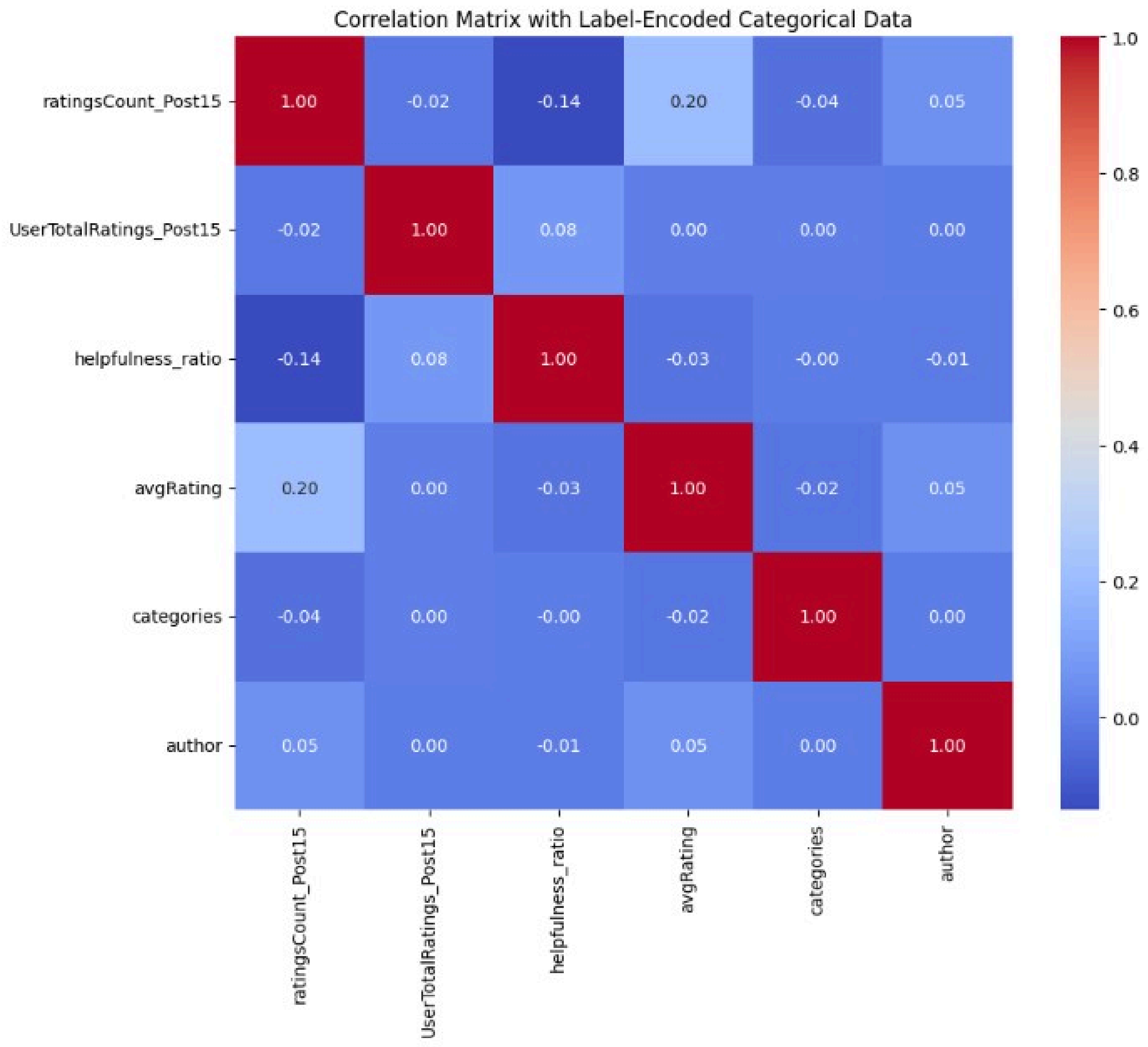
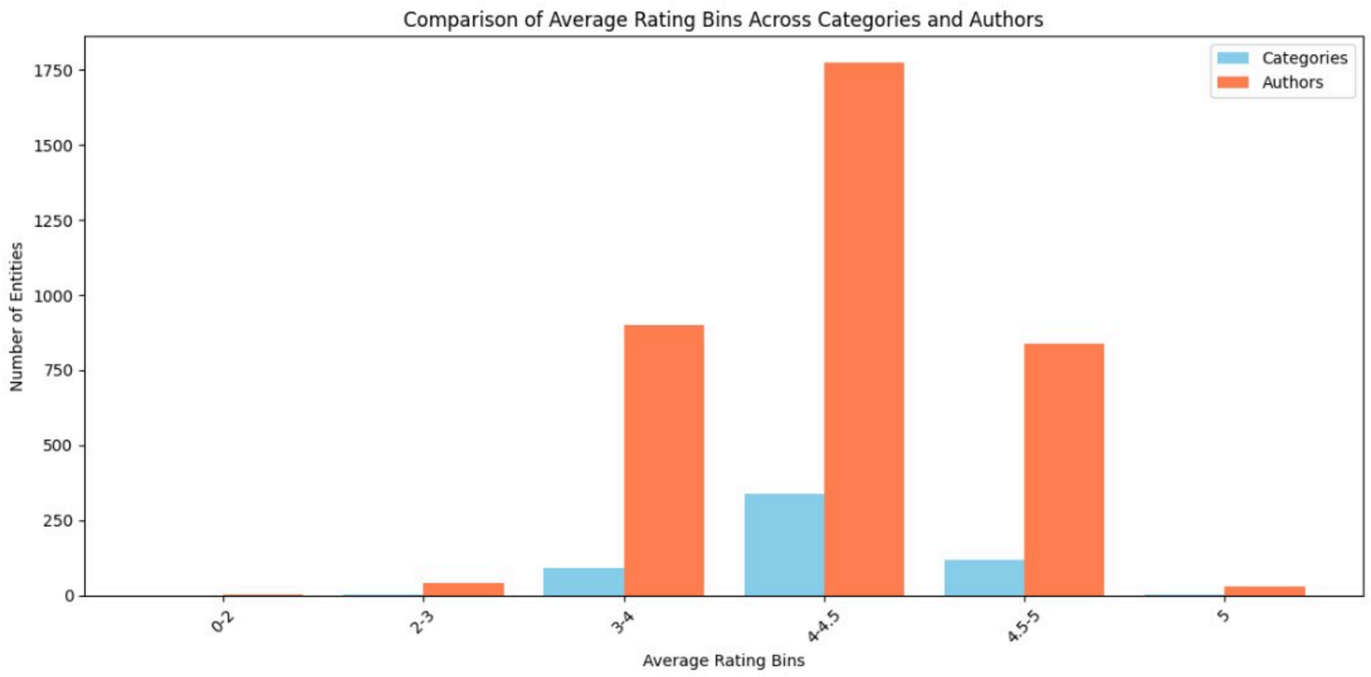
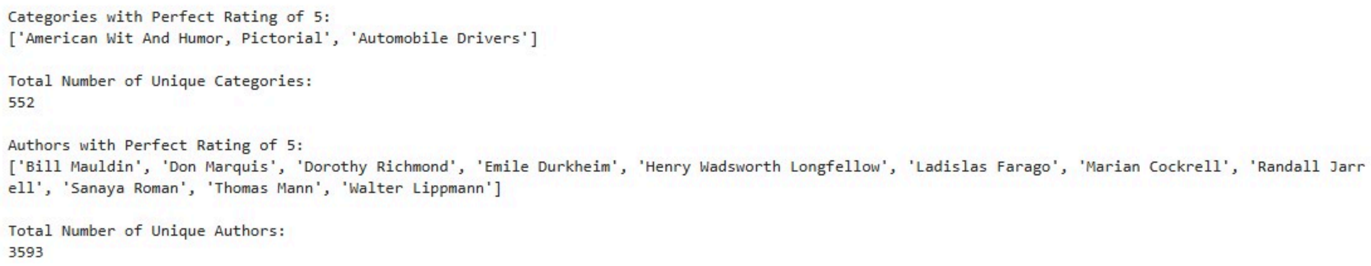


Figure 1: Data Correlation Matrix

As can be seen most variables can show no or very weak correlation, giving us some confidence that these values are independent and good for use in our models. The only one that potentially has some relation is the count of ratings and the average rating. Meaning, the more ratings a book had, the more likely the book was to be good. This would make sense since people tend to read books that are highly rated.

To understand the categorical data that would help content-based filtering, we checked to see if any specific items that ensured a book would have an average rating of 5.0. As can be seen in the graph below, there are 3 categories and 11 authors that always get a 5.0 rating in their books. However, “always” means the 4 books that make up those 3 categories and the 15 books written by those 11 authors. This may mean our dataset is not varied enough – it will also mean that it will be interesting to see whether the content-based model will overfit those categories for testing.

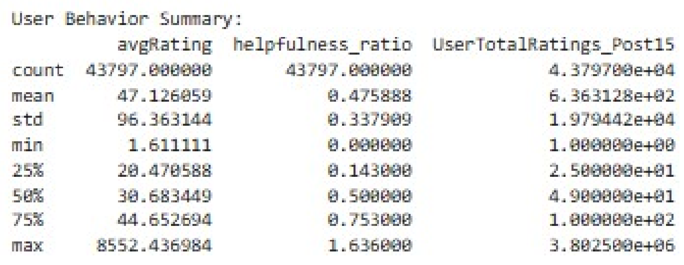




```
Categories with Perfect Rating of 5:
['American Wit And Humor, Pictorial', 'Automobile Drivers']

Total Number of Unique Categories:
552

Authors with Perfect Rating of 5:
['Bill Mauldin', 'Don Marquis', 'Dorothy Richmond', 'Emile Durkheim', 'Henry Wadsworth Longfellow', 'Ladislas Farago', 'Marian Cockrell', 'Randall Jarr
ell', 'Sanaya Roman', 'Thomas Mann', 'Walter Lippmann']

Total Number of Unique Authors:
3593
```

Figure 2: Average Rating Bins Across Categories and Authors

Finally, one of the last things we checked for the user-based filtering model was the average behavior of reviewers and this is what we found:



```
User Behavior Summary:
          avgRating  helpfulness_ratio  UserTotalRatings_Post15
count  43797.000000       43797.000000             4.379700e+04
mean      47.126059           0.475888             6.363128e+02
std       96.363144           0.337909             1.979442e+04
min        1.611111           0.000000             1.000000e+00
25%       20.470588           0.143000             2.500000e+01
50%       30.683449           0.500000             4.900000e+01
75%       44.652694           0.753000             1.000000e+02
max     8552.436984           1.636000             3.802500e+06
```

Figure 3: User Behavior Summary

This table summarizes user behavior metrics, including the average rating (average review score per book id), helpfulness ratio, and the total number of ratings per user. The dataset includes 44k users. The average rating across all users is 47.13, but the high standard deviation (96.36) and maximum value

(8,552.44) suggest that this column may represent a cumulative or scaled score rather than individual ratings. The range of values, from 1.61 to 8,552.44, highlights significant variability among users.

The helpfulness ratio averages 47.6%, meaning that nearly half of the votes users received for their reviews were marked as helpful. However, it ranges from 0 (no helpful votes) to 1.63, with the higher values possibly reflecting inconsistencies or anomalies in the data. For ratings, users submitted an average of 6,363 ratings, but the data is highly skewed. The median is much lower, at only 30.68 ratings, indicating that most users contribute far fewer reviews while a small number of extremely active users dominate the dataset. These outliers may need further investigation to understand their impact on the overall analysis.

Supervised Learning Model

K-Nearest Neighbor Model

For one supervised learning algorithm, we chose a K-Nearest Neighbors (KNN) model due to its effectiveness in similarity-based recommendation tasks and its interpretability. KNN is well-suited for both collaborative filtering (based on user interactions) and content-based filtering (based on book features), making it an ideal choice for a hybrid recommendation system.

In the collaborative filtering model, KNN is applied to a user-item matrix, where it identifies patterns in user behavior based on ratings. By grouping users with similar reading preferences, the model can recommend books that users with similar tastes have enjoyed. This provides recommendations rooted in social proof. However, collaborative filtering alone has limitations, particularly when dealing with less-reviewed books, as it relies heavily on user interaction data.

To address these limitations, we implemented a content-based filtering model using KNN to operate on text-based features like book descriptions, author clusters, and genres. By converting these features into numerical vectors using Term Frequency-Inverse Document Frequency (TF-IDF), the model can recommend books with similar themes, genres, or styles, regardless of user ratings. This content-based approach allows the system to recommend books even if they have few or no ratings, filling the gaps left by collaborative filtering. However, content-based filtering on its own lacks the personalized touch that comes from analyzing individual user preferences.

The hybrid recommendation model balances user preferences with book content by calculating a *Hybrid *Score for each recommended book, using the formula:

$$\text{Hybrid Score} = \alpha \times \text{Collaborative Score} + (1 - \alpha) \times \text{Content Score}$$

In this model:

- Collaborative Score reflects similarity based on user ratings—lower scores mean the book is highly rated by users with similar preferences.
- Content Score reflects similarity based on book features like themes and genres—lower scores indicate higher content similarity.
- α is a weight factor that controls the emphasis between the two types of filtering.

Lower Hybrid Scores indicate higher relevance, with books that better match user preferences and content similarity appearing as top recommendations.

Neural Network Model

Neural Collaborative Filtering (NCF) is a deep learning technique based on user-item interactions. It is similar to our earlier Collaborative Filtering models, except that here we are utilizing a neural architecture. We are going to try this model with different parameters and evaluate model performance on our test set. Each relevant feature has been label encoded. We have chosen a combination of label encoding and embedding layers, so that we can properly process the features that have high-cardinality, without misinterpreting the numerical value of the encoded labels. For each of the eight NCF models, different combinations of features, epochs, and embedding dimensionality were used to train the model, and each model was validated against the same test set. We will examine these results and their meaning in the Results section.

Unsupervised Learning Models

User-Based Collaborative Filtering

One of the models that we are using is a collaborative filtering model. We chose this model because it has demonstrated success in recommender systems [4,7]. Also, our goal is to recommend books to a person. Thus, finding similar users to that person and learning what books they enjoy will enable us to make high quality predictions regarding how much a user will like any certain book. A collaborative filtering model compares data on a user-by-user basis and finds users that have similar preferences to make predictions. In the context of our project, we will use a collaborative filtering model to predict books that a given will enjoy. To implement this, we used the cleaned preprocessed data and then organized the data to show the rating that each user has given to each book. Then we split the data into training, validation, and testing sets using 70% of the data to train, 10% of the data to validate, and 20% of the data to test.

Our initial implementation of the model utilized all features in the dataset. However, as collaborative filtering models can be computationally intensive due to their requirement of iterating through each user in the test set, training the model took a large amount of time. In order to address this, we decided to use sparse matrix representations to increase the efficiency of our model. This dramatically reduced the amount of time needed for the model to train.

Our implementation utilized cosine similarity to compare users. While forming the model, we tested multiple threshold values for the previously calculated similarity values. For the purposes of our model, users in the training data who were above the similarity threshold were used to make predictions for each respective user in the validation set. Next, the average rating for each book for all users in the dataset is calculated and these values are used as the predictions of the ratings that the test user would give these books. Using these ratings, we can recommend the books with the highest ratings to the test user.

Content-Based Filtering

We also looked into an approach that only uses content-based filtering, to see how much we could analyze just using the information in the books. In between the final and midpoint report, we have combined the user-based collaborative filtering approach with this approach. In contrast to user-based collaborative filtering or a hybrid approach, content-based filtering - on its own - only focuses on book characteristics, like title, author, description, and category. We used TD-IDF to vectorize the inputs and then used cosine similarity to identify the n (default choice of 10) nearest books to a given book based on its characteristics. To evaluate this approach, we took a sample of 500 users and identified the precision and recall of the recommendations when comparing the books they have read to other possible similar books they may be interested in reading.

The precision and recall of this only content-based filtering approach on its own was low when tested on various sample sets, so we mainly present the results of our combined filtering model below in the Results and Discussion section of our report.

Combined Filtering Model

After creating a collaborative filtering model and a content-based filtering model, we decided to combine them into a combined-filtering model. This combined model combines the aspects of user-user similarity in the collaborative filtering model and the item-item similarity in the content-based filtering model. Similar to previous models, we split the data into training, validation, and testing sets using splits of 70%, 10%, and 20%. We then trained the two models separately and combined them only in the final prediction phase. Both models utilized cosine similarity to calculate the similarity values within the models. Because the content-based filtering model did not explicitly give predictions for book ratings, we utilized the cosine similarity values and scaled them from 1-5. This enabled us to combine the two models in the final predictions. To find the optimal weighting of each model to be used in our final predictions we calculated the RMSE for different weights. Then using the best weighting for our models we made predictions on the test set.

Results and Discussion

Metric Overview

To evaluate the effectiveness of our models, we are utilizing several metrics, including Cosine Similarity and RMSE. An overview of these metrics, as well as the pros and cons when compared to similar metrics is shown below.

Cosine Similarity measures the similarity between two vectors “by calculating the cosine of the angle between the two vectors.”[11] It is the standard measure of similarity across many fields of data science, and can easily be applied to our dataset.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

calculation of cosine of the angle between A and B

Figure 4: Cosine Similarity [11]

To assess the accuracy of our predicted ratings vs actual user ratings, we will utilize Root Mean Squared Error (RMSE). RMSE is useful because, since the error terms are squared, it penalizes predictions with large differences, and since it is in the root form, it is converted back to the metric form that we are interested in optimizing (prediction rating similarity). This should result in a more stable prediction model when compared to Mean Absolute Error (MAE), as large prediction differences are less significant for MAE, shown in Figure 6.

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} (\hat{r}_{ui} - r_{ui})^2}$$

Figure 5: RMSE Formula [10]

$$\text{MAE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} |\hat{r}_{ui} - r_{ui}|}$$

Figure 6: MAE Formula [10]

K-Nearest Neighbor Results and Analysis

PCA Implementation

For the KNN hybrid recommendation model, we utilized Principal Component Analysis (PCA) to reduce the dimensionality of the input features, CF_Score (collaborative filtering score) and CB_Score (content-based score). PCA transformed these features into a single principal component, retaining 99.97% of the variance. This ensured minimal information loss while simplifying the dataset for more efficient computation. The PCA component revealed that collaborative filtering contributed 95.37% to the variance, highlighting its dominant role in determining the hybrid scores.

RMSE Evaluation Across Alpha Values

The hybrid model's performance varied with the alpha parameter, which controls the balance between collaborative and content-based filtering. The lowest RMSE of 0.2510 was achieved at alpha = 0.2, indicating this weighting provided the best balance for accurate predictions. Alpha values near 0.1 and 0.3 also performed well, with RMSE values of 0.2824 and 0.2541, respectively. However, as alpha increased beyond 0.3, the RMSE steadily worsened, reaching 0.3373 at alpha = 0.5 and 0.5696 at alpha = 0.9. This trend suggests that prioritizing collaborative filtering too heavily can reduce the accuracy of the hybrid recommendations for this model.

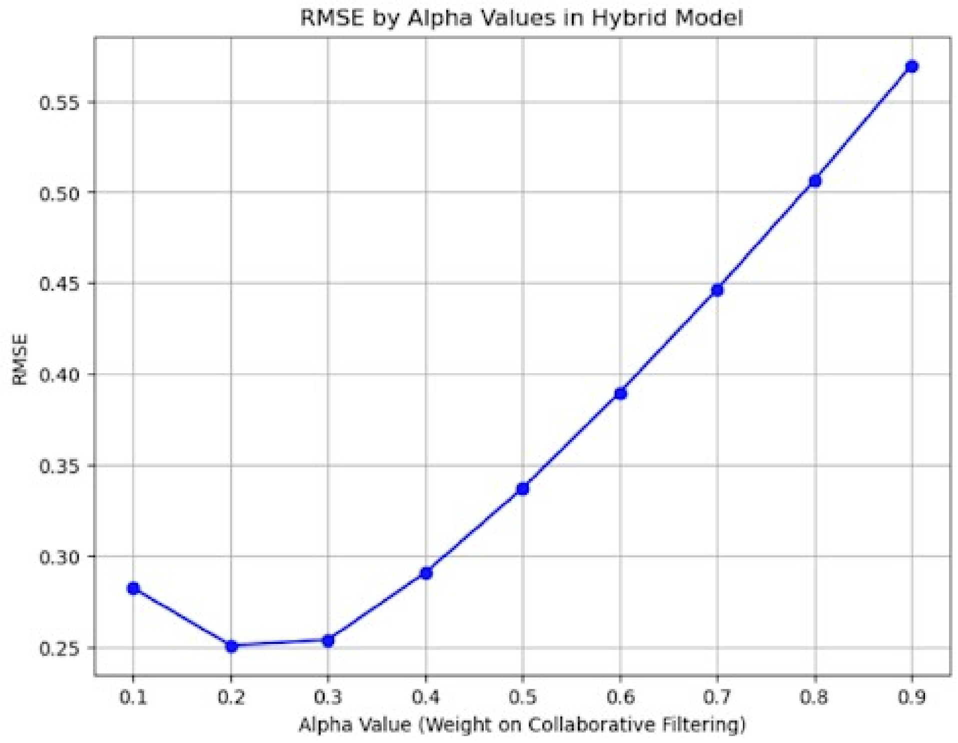


Figure 7: RMSE by Alpha Values in KNN Hybrid Model

This model demonstrated strong performance, particularly at alpha values around 0.2, where it achieved the lowest RMSE of 0.2510. These results emphasize the importance of carefully balancing the weights of collaborative and content-based filtering to optimize the model's accuracy.

Collaborative Filtering Results and Analysis

Similarity Threshold Tuning

To test the quality of our collaborative filtering model at each similarity threshold, we calculated the RMSE score between the predicted ratings for a book by a test user and the actual ratings given by that test user. After using the model with a variety of similarity value thresholds, ranging from 0 to 100 percent similar, we found that the model performed best with thresholds between 20 and 40 percent. To get greater granularity zoomed in on thresholds within this range and found that the model performed best on the validation set when a similarity value threshold of 28% was used. The results for each threshold value can be seen in the figure below. As a result of the similarity threshold hyperparameter tuning above, we chose to use a similarity threshold of 28% on our test set.

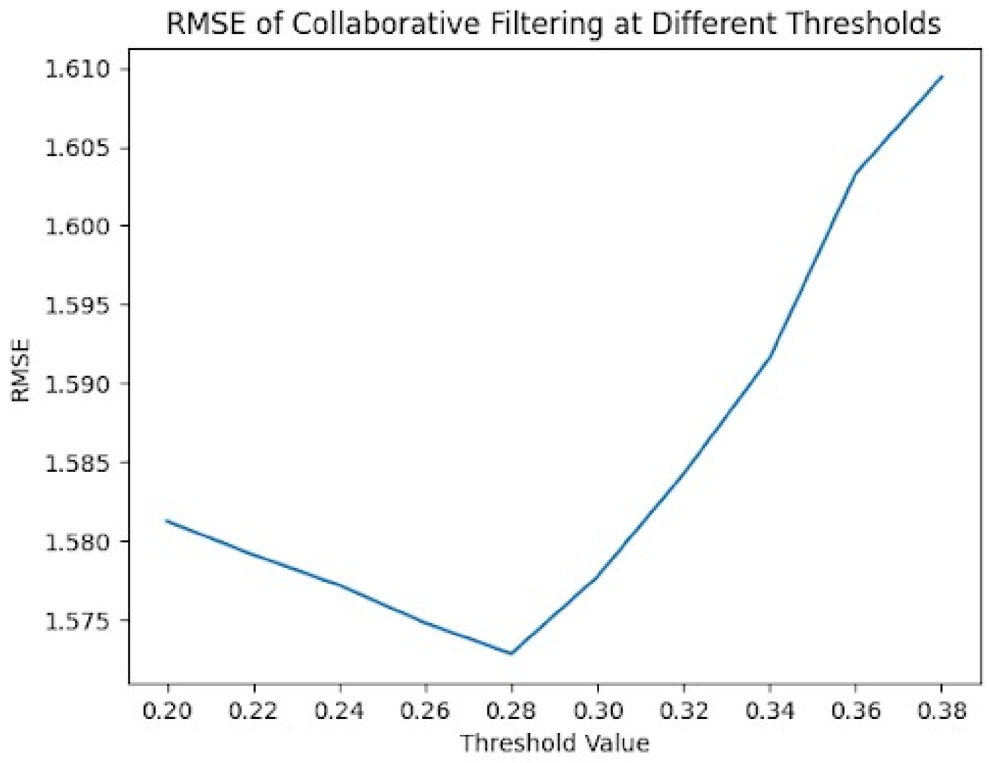


Figure 8: RMSE of Collaborative Filtering at Different Thresholds

After training the model using the PCA and similarity value threshold tuning implementations described above, we then tested our model on the test set. The result of this was an RMSE of approximately 2.0965. This value essentially indicates that when predicting the rating that a user will give a book, the collaborative filtering model will, on average, be off by about 2.0965 points on a 1-5 rating scale.

Combined-Filtering Model Results and Analysis

Weight Tuning

To optimize the combined model's performance, we thought it necessary to give weights to both the collaborative filtering and content-based filtering models. To find the optimal weight we tried a number of different values ranging from 0 to 1. The formula we used was (weight) * (collaborative filtering output) + (1-weight) * (content-based filtering output). This would ensure that both models are being used in the combined model and that they will still have interpretable results even with the weighting. After performing a few iterations, we found that the model performed best with a weight value above 0.8. Then we focused our optimization on this range. Eventually we found that the weight with the lowest RMSE was 0.95.

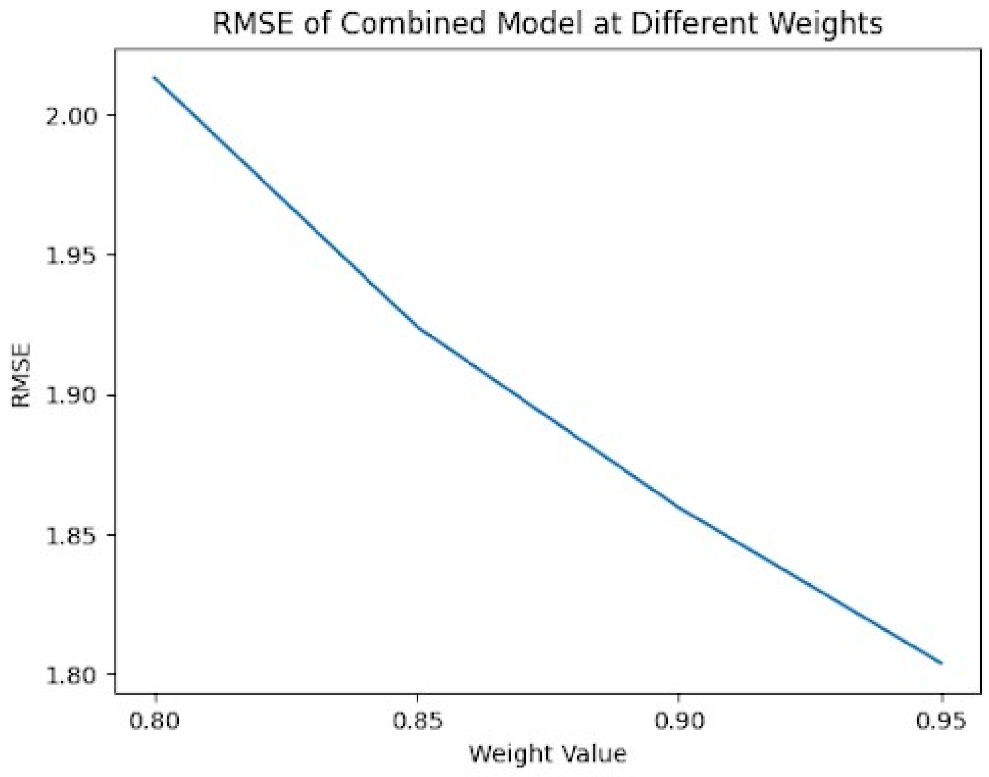
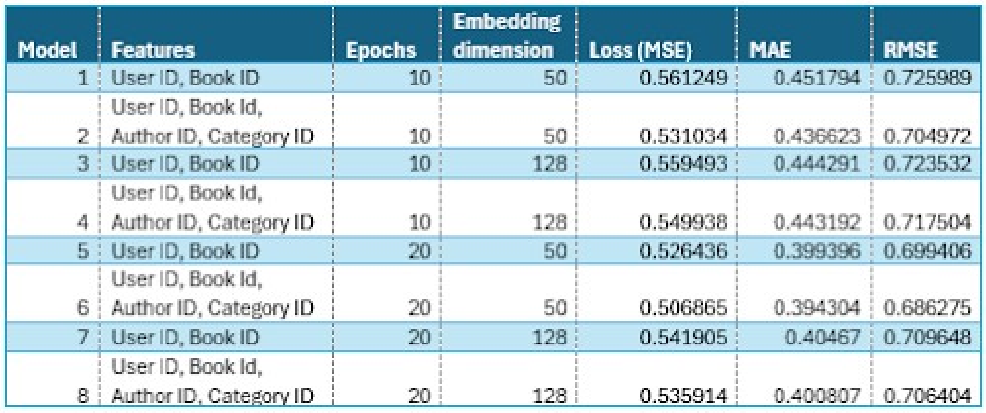


Figure 9: RMSE of Combined Model at Different Weights

After finding the optimal weight value of 0.95, we ran the model using this weight on the test set. The result was an RMSE value of approximately 1.995. This is slightly better than the collaborative filtering model on its own. This, along with the relatively high weight value, indicates that the collaborative filtering model is more predictive overall than the content-based filtering model. However, the content-based filtering model does have some predictive power as indicated by the slight improvement in the combined model's RMSE value on the test set.

Neural Network Model Results and Analysis

Next, we will evaluate each model and explore the patterns that can be seen in the various parameters. Best RMSE was achieved with Model 6, which was 20 epochs, embedding dimensionality of 50, and the features of User ID, Book ID, Author ID, and Category ID. The reason for the success of Model 6 becomes clear as we explore the effects of each parameter change on model performance.



Model	Features	Epochs	Embedding dimension	Loss (MSE)	MAE	RMSE
1	User ID, Book ID	10	50	0.561249	0.451794	0.725989
2	User ID, Book Id, Author ID, Category ID	10	50	0.531034	0.436623	0.704972
3	User ID, Book ID	10	128	0.559493	0.444291	0.723532
4	User ID, Book Id, Author ID, Category ID	10	128	0.549938	0.443192	0.717504
5	User ID, Book ID	20	50	0.526436	0.399396	0.699406
6	User ID, Book Id, Author ID, Category ID	20	50	0.506865	0.394304	0.686275
7	User ID, Book ID	20	128	0.541905	0.40467	0.709648
8	User ID, Book Id, Author ID, Category ID	20	128	0.535914	0.400807	0.706404

Figure 10: Detailed summary of each NCF model's performance

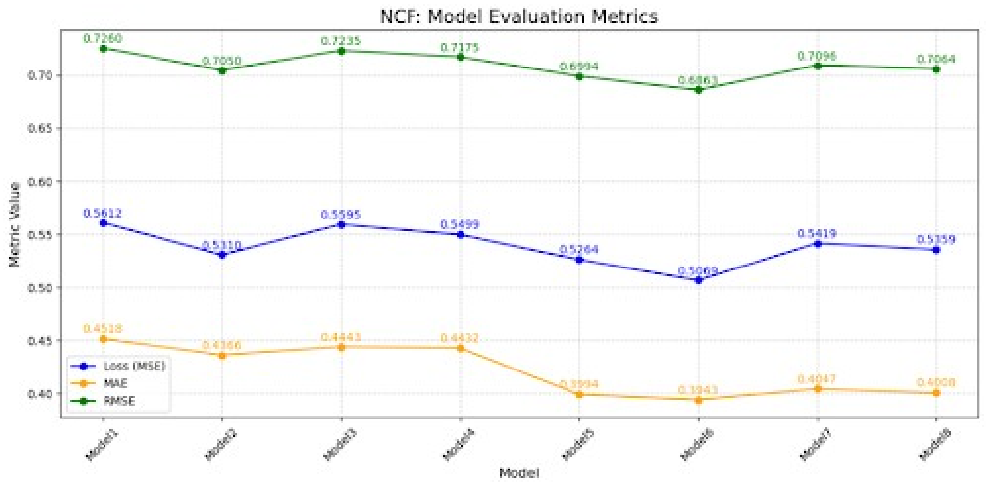


Figure 11: NCF Model Performance

Adding Features

Figure 12 shows the effect of adding the Author and Category features for each model when compared against their otherwise identical counterpart (Model 1 vs2, 3 vs 4, 5 vs 6, and 7 vs 8). For each model comparison we see that the lower RMSE was achieved when adding the features, however it is most pronounced in the comparison between Model 1 and Model 2, which are the two models with lowest number of epochs (10) and embedding dimensionality)

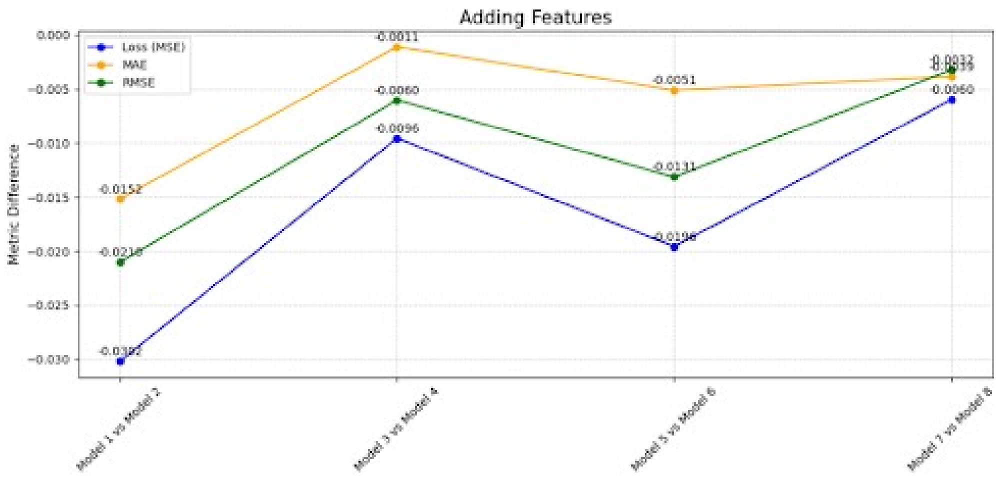


Figure 12: Adding Features

Increasing Embedding Dimensionality

Figure 13 shows the effect of increasing the embedding dimensionality from 50 to 128 when compared against their otherwise identical counterpart (Model 1 vs 3, Model 2 vs 4, Model 5 vs 7, and Model 6 vs 8. The increase of embedding dimensionality increases the amount of space that the model can represent the hidden patterns in the dataset. Embedding dimensionality is important because the size of this space can greatly effect the likelihood of overfitting the model to the training data. This can be seen in Figure D, as the only comparison that showed improvement when increasing dimensionality is Model 1 vs 3. The other model comparisons that had more features and more training epochs, actually became less accurate at predicting user ratings.

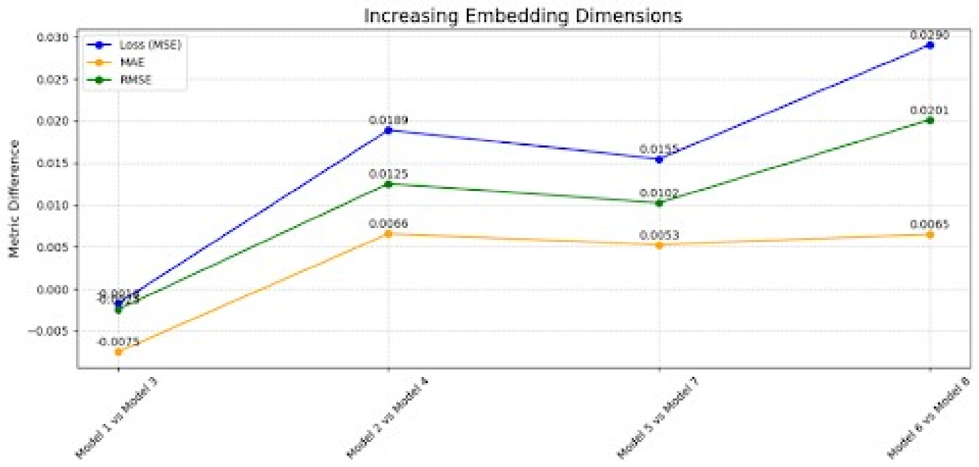


Figure 13: Increasing Embedding Dimensions

Increasing Training Epochs

Figure 14 shows the effect of increasing the number of training epochs from 10 to 20 on each model. Model comparisons are Model 1 vs 5, 2 vs 6, 3 vs 7 and 4 vs 8. Increasing the amount of training epochs showed an improvement in model performance across each comparison, with Model 1 vs 5 showing the most pronounced improvement.

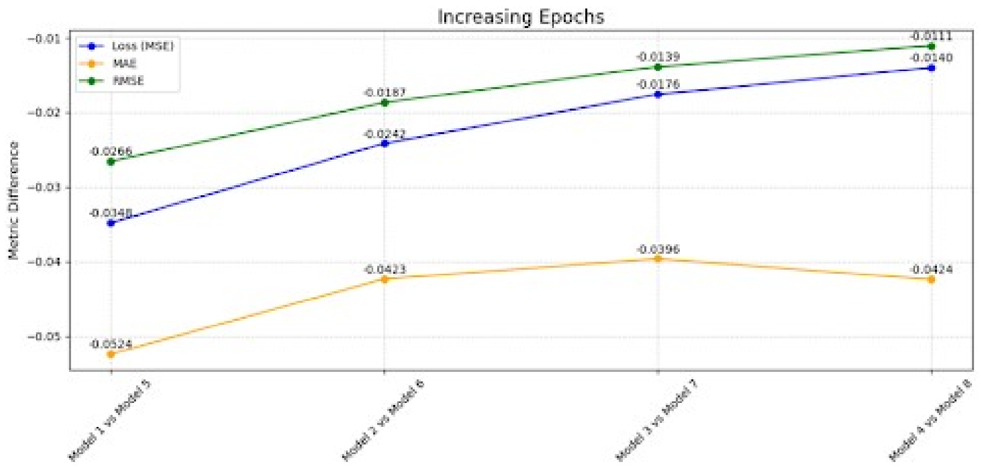


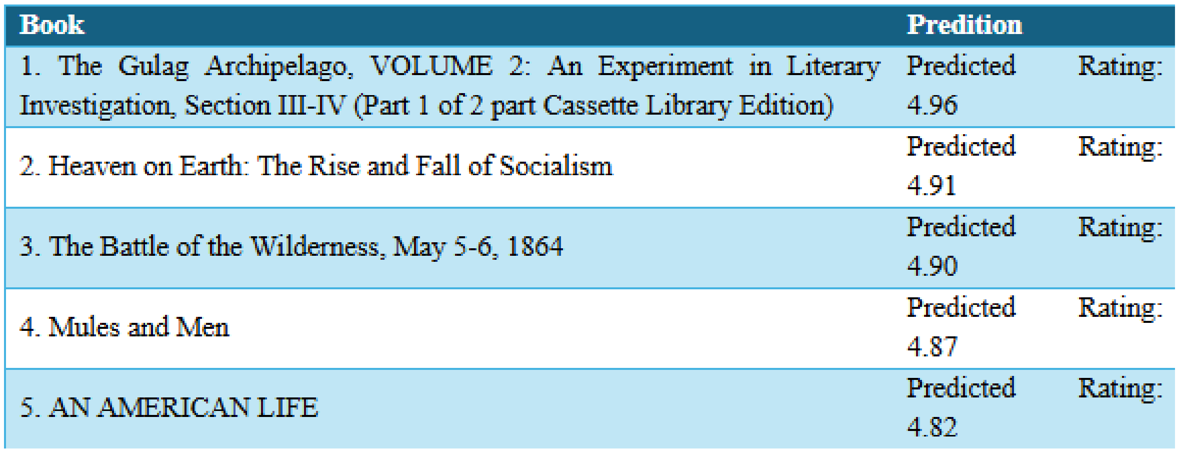
Figure 14: Increasing Training Epochs

NCF Examination of Model Parameters

Data features, training epochs, and embedding dimensionality are all key aspects of properly building a recommendation model. Our highest performing NCF model was Model 6, with 20 epochs, embedding dimensionality of 50, and all 4 features. After examining the effect of each of these, the reasons why Model 6 was most successful become clear. Increasing the training epochs allowed for more accurate model configuration, while limiting the embedding dimensionality allowed us to decrease the likelihood of modeling the random patterns and overfitting the model on training data. And of course, adding features like Author and Category were very beneficial to successfully recommending a book to a user.

Book Recommendation Comparison

For each model, a top 5 recommendation list was calculated for the same random user. Figure 15 shows the top 5 recommendation list for this random user for Model 6, our highest performing deep learning model.

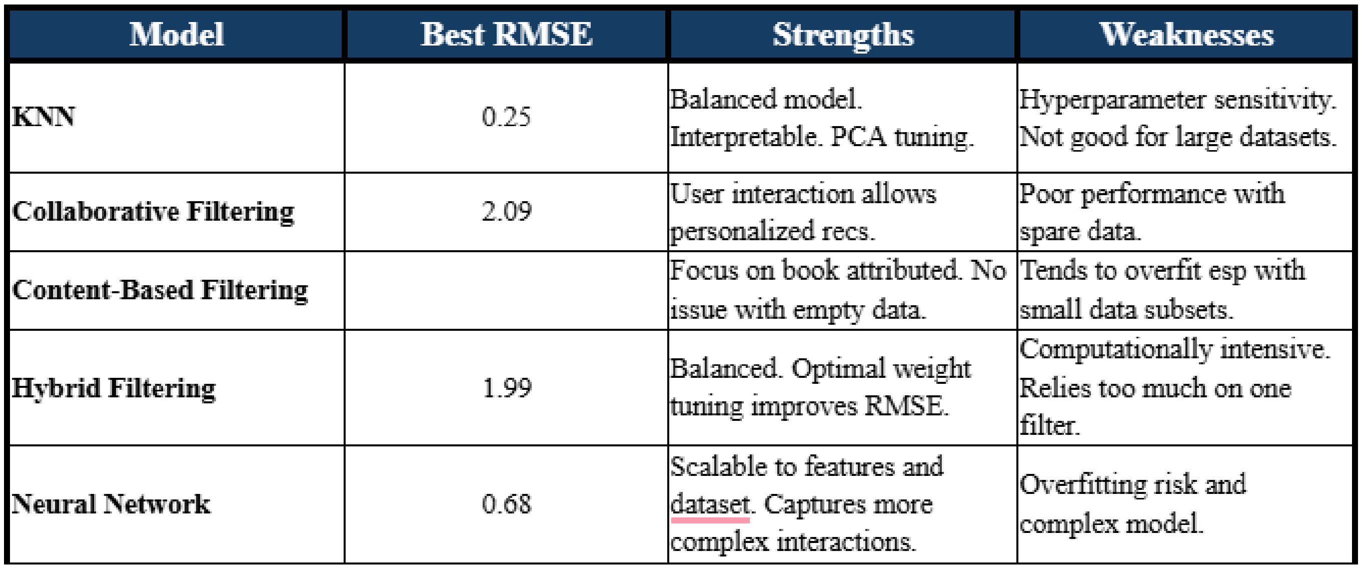


Book	Predition
1. The Gulag Archipelago, VOLUME 2: An Experiment in Literary Investigation, Section III-IV (Part 1 of 2 part Cassette Library Edition)	Predicted Rating: 4.96
2. Heaven on Earth: The Rise and Fall of Socialism	Predicted Rating: 4.91
3. The Battle of the Wilderness, May 5-6, 1864	Predicted Rating: 4.90
4. Mules and Men	Predicted Rating: 4.87
5. AN AMERICAN LIFE	Predicted Rating: 4.82

Figure 15: Model 6 Recommended Books for User 0

Comparison of Models

The overall comparison of all the models we reviewed in this project is as follows.



Model	Best RMSE	Strengths	Weaknesses
KNN	0.25	Balanced model. Interpretable. PCA tuning.	Hyperparameter sensitivity. Not good for large datasets.
Collaborative Filtering	2.09	User interaction allows personalized recs.	Poor performance with spare data.
Content-Based Filtering		Focus on book attributed. No issue with empty data.	Tends to overfit esp with small data subsets.
Hybrid Filtering	1.99	Balanced. Optimal weight tuning improves RMSE.	Computationally intensive. Relies too much on one filter.
Neural Network	0.68	Scalable to features and dataset. Captures more complex interactions.	Overfitting risk and complex model.

Figure 16: Overall Model Comparison

After reviewing this table, we can conclude that the best overall performance was achieved by the KNN hybrid model as it had the lowest RMSE. However, this is considering that the Kaggle dataset is already a small subset and was further reduced to make analysis easier. Considering scalability and RMSE the next better one would be the NCF neural network which would also be able to create more complex potential patterns, so it would be interesting to keep researching that.

Next Steps

Moving forwards, we would like to keep exploring new models and more effective mixed-filtering options. However, at a more high-level, we believe this application would need to start focusing on incorporating real-time feedback into recommendations – such as time spent on a book, likes and dislikes of a user to allow a person's preferences to change over time, and more. Another important idea is ensuring the address of ethical concerns such as promoting diverse content, and ensuring recommendations are appropriate for age-groups. Finally, an interesting future to this project could be incorporating multimodal data such as extending recommendation system to linking books with movies, TV shows, podcasts, audio books, and music. This could make the user experience more interactive and attractive to users who are not necessarily focused on (just) books. These steps focus on the long-term potential and vision of the project.

References

1. Eva Zangerle and Christine Bauer. 2022. Evaluating Recommender Systems: Survey and Framework. ACM Comput. Surv. 55, 8, Article 170 (August 2023), 38 pages. https://doi.org/10.1145/3556536

2. H. Nirwan, O. P. Verma and A. Kanojia, "Personalized hybrid book recommender system using neural network," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2016, pp. 1281-1288.
3. Isinkaye, F.O., et al. “Recommendation systems: Principles, methods and evaluation.” Egyptian Informatics Journal, vol. 16, no. 3, Nov. 2015, pp. 261–273, https://doi.org/10.1016/j.eij.2015.06.005.
4. M. Manwal, D. Rawat, D. Rawat, K. C. Purohit and T. Choudhury, "Movie Recommendation System Using TF-IDF Vectorizer and Bag of Words," 2023 12th International Conference on System Modeling & Advancement in Research Trends (SMART), Moradabad, India, 2023, pp. 163-168, doi: 10.1109/SMART59791.2023.10428182.
5. P. Mathew, B. Kuriakose and V. Hegde, "Book Recommendation System through content based and collaborative filtering method," 2016 International Conference on Data Mining and Advanced Computing (SAPIENCE), Ernakulam, India, 2016, pp. 47-52, doi: 10.1109/SAPIENCE.2016.7684166.
6. R. Gelles-Watnick and A. Perrin, “Non-Book Readers 2021,” Pew Research Center, pp. 1–3, Feb. 2021, Accessed: Sep. 30, 2024. [Online]. Available: https://www.pewresearch.org/wp-content/uploads/2021/09/Non-Book-Readers-2021-Methodology-Topline.pdf
7. Taha, Z. (n.d.). Cell editing: OpenRefine. OpenRefine RSS. https://openrefine.org/docs/manual/cellediting#cluster-and-edit
8. V. Thannimalai and L. Zhang, "A Content Based and Collaborative Filtering Recommender System," 2021 International Conference on Machine Learning and Cybernetics (ICMLC), Adelaide, Australia, 2021, pp. 1-7, doi: 10.1109/ICMLC54886.2021.9737238.
9. Z. Fayyaz, M. Ebrahimian, D. Nawara, A. Ibrahim, and R. Kashef, “Recommendation Systems: Algorithms, Challenges, Metrics, and Business Opportunities,” Applied Sciences, vol. 10, no. 21, p. 7748, Nov. 2020, doi: https://doi.org/10.3390/app10217748.
10. F. Casalegno, "Recommender Systems: A Complete Guide to Machine Learning Models," Towards Data Science, 29-Mar-2022. [Online]. Available: https://towardsdatascience.com/recommender-systems-a-complete-guide-to-machine-learning-models-96d3f94ea748.
11. G. Shani, and A. Gunawardana, “Evaluating Recommendation Systems,” Microsoft Research, Nov. 2009 https://www.researchgate.net/publication/226264572_Evaluating_Recommendation_Systems
12. M. Bhandari, "Cosine Similarity: How Does it Measure the Similarity? Maths Behind and Usage in Python," Towards Data Science, Apr. 6, 2022. [Online]. Available: https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db

Gantt Chart

https://gtvault.sharepoint.com/❌/s/CS7641-MLGroupProject/EZX0jIotyPhIsyPVRWbYlcIBsO1Fwphh2rOKThf2kPWAYg?e=ycgHqH

Contribution Table

Group Member Name	Proposal Contributions
Michael Anderson	Neural Network Algorithm
	Neural Network Testing and Visualization
	Literature Read - References [10], [11]
Nathan Brodie	Collaborative Filtering Algorithm and Testing
	Combined-Filtering Algorithm and Testing
	Literature Read - Reference [9]
Irene Feijoo	Data Preprocessing and Visualizations
	Model Comparisons and Analysis
	Literature Read - References [1], [2], [6]
Navya Gautam	Content-Based Filtering Algorithm and Testing
	Presentation Slides and Video
	Literature Read - References [5], [7]
Shriya Purohit	KNN Algorithm
	KNN Algorithm Testing and Visualization
	Literature Read - Reference [3], [4], [8]