Improving Book Recommendation Systems using Ensemble Models and Natural Language Processing

Ang Kai En [*1] **Meritxell Camp Garcia** [*1] **Sidharth Pahuja** [*1] **Sim Jun You** [*1] **Sim Yew Chong** [*1]

Abstract

This report presents an enhanced book recommendation system to improve recommendation precision. By integrating unstructured text data and diverse data sources, the proposed system offers more robust recommendations tailored to individual users, ultimately improving user retention rates.

Utilizing a Goodreads dataset from 2017, comprising book information, user-tagged genres, and user reviews, we built and trained six distinct models based on different data types. These models were then combined into an ensemble logistic regression model, outperforming individual models in precision and exhibiting higher F1 scores in binary classification for book recommendations.

While the ensemble model requires more computational resources than the user similarity model, it effectively mitigates popularity bias, a common issue in naive recommendation systems. Finally, the system's user interface, developed with Flask, offers transparent recommendations with explainability graphs, enhancing user trust and experience.

Overall, the enhanced book recommendation system shows promising results and has the potential to outperform the naive user similarity model with further data refinement and model training.

Codes and sample data can be found at the following link here: https://github.com/lyncsghrk/BT5153-Books

1. Description of Problem

1.1. Problem Statement

In response to the overwhelming number of book choices online, which often leads to decision paralysis and wasted time, we propose the implementation of a Natural Language Processing (NLP) powered recommendation system to address this challenge.

Current recommendation systems in prominent book platforms, such as Amazon or Goodreads, primarily rely on criteria like ratings and user similarity, often overlooking the valuable insights embedded within detailed book reviews. While Goodreads (2011) suggests rating more books to get better recommendations from their system to form a better profile of each user, they focus on the numerical ratings and do not fully utilize the text information (Goodreads, 2011; Garai, 2022). Thus, users frequently express dissatisfaction with the inadequacy or lack of personalized recommendations, as seen from Goodreads (2013; 2020). Furthermore, such recommendation models commonly suffer from popularity bias, since popular books are more likely to be read and reviewed by more users, and hence recommended even to niche book readers. On the other hand, less popular items are recommended rarely or not at all (Naghiaei et al., 2022; Pragathi., 2023), which can hurt lesser-known but potentially good recommendations.

This is a problem found not only in book recommendation systems but in e-commerce and social platforms alike. For example, Audible's (Amazon's audiobook and podcast company) recommendation system, has also faced significant criticism from its 27.3 million monthly users due to subpar user experience (Allen, 2023).

With over 140 million users in 2022, Goodreads relies on effective book recommendations to keep users engaged (Reedsy, 2024). Several market research firms estimate the global recommendation engine market to reach a total value between US$28 billion and US$54 billion by 2030, reflecting a compound annual growth rate (CAGR) exceeding 30% (Straits, n.d.; Allied, n.d.; Mordor, n.d.), indicating a great market opportunity here. Offering relevant suggestions to users is extremely crucial to users, enhancing their overall experience, fostering long-term loyalty, and ultimately increasing user retention rates, and it is therefore crucial for business growth by improving their recommendation systems.

1.2. Project Objective

We seek to create an improved recommendation system that harnesses the power of text-based book reviews alongside book-related information, to create a more robust and

fair recommendation system that more accurately mirrors and predicts books that align with the preferences of actual readers. This strategy mirrors Spotify's (n.d.) groundbreaking introduction of the "taste profile" feature in the music industry .

To accomplish this, we developed different underlying recommender models capturing varying aspects of books and user profiles. Some important book information include genres, book titles and book descriptions, while examples of important user information are users' reviews and liked books. We then combine the models into one ensemble, assigning different weights to various models according to model training against each user's actual ratings. Our ensemble model will then be compared against the naive model using user similarity only, which relies solely on simple vector similarity searches to find other users with similar reading behavior and patterns as the target user, and subsequently recommending new books from these similar users to the target user. Model precision is the most important metric, to ensure the books recommended to the users are books they will actually be interested in.

Finally, we built a user interface (UI) which accepts user profiles, and curates a list of recommended books for them. Our UI focuses on model explainability, giving a breakdown of why each book was recommended to the user, promoting transparency between our model and the user, thereby increasing user retention and trust.

2. Dataset

2.1. Dataset Description

The main data source is a Goodreads dataset (Wan, 2017), consisting of information scraped from users' public shelves in late 2017. It contains 2 million books and 15 million reviews from 465,000 users. We chose this dataset due to its extensive supplementary book data, which include user-tagged genres, book descriptions, and author details. The user-assigned star rating (from 1 to 5) is also provided for each text review, making it easy to categorize the information as well.

There are four main tables in this dataset:

1. **Books** – Book IDs and associated details of each book, such as title, number of pages, average rating, number of ratings, etc.
2. **Genres** – Book IDs and a column of JSON strings mapping associated genres (tagged by users, as is Goodreads' system) to tag frequency.
3. **Reviews** – User ID of the reviewer, Book ID of the book getting reviewed, and associated details of the review such as review text, rating, date submitted, etc.
4. **Users** – User IDs and number of reviews made.

2.2. Data Preprocessing

To ensure that the models are effectively run within computational constraints with our available RAM and vCPUs on our cloud compute instance, we utilized a subset from the available dataset. We took a subsection of users with 400-500 reviews, which provides us with a balanced representation of user engagement while keeping the dataset within a manageable level of our computing resources. This still gives us a diverse dataset, with a total of 457,320 reviews, 1,857 unique users, and 134,172 unique books to work with.

However, we do acknowledge the current filtering criteria limit the generalization of our recommender system since it will work best with users who fulfill this kind of profile. An improved way of approaching this would have been to randomly sample users with varying numbers of reviews to provide a more holistic representation of the entire user population while keeping the size of the overall data to acceptable limits. Nevertheless, the current dataset provieds a sufficiently adequate subset to build the model on without compromising on quality.

2.3. Data Splitting

To prevent data leakage, it is crucial to ensure the independence of the training and testing sets while ensuring they are representative of the original dataset. Since our recommender relies on user reviews for personalization, we first randomly segregate the data by user, ensuring that a user's reviews are exclusively allocated to either the training or testing set, eliminating any potential overlap. However, models that do not require training such as clustering-based models will use the entire dataset from the start.

To achieve this, after cleaning the data, we split it into two sets, sub, and ensemble (or unseen dataset), with a ratio of 70-30. The sub dataset (70% of data), containing 1,299 users, 320,152 reviews, and 109,296 books, is used to train our underlying recommendation models, while the 30% unseen dataset amounting to 558 users, 137,168 reviews, and 83,075 books is put aside to test the effectiveness of our ensemble book suggestion model with unseen users. The ensemble dataset is further split into train and test sets with a ratio of 80-20 for training and testing purposes. When performing these splits, the data associated with the users in the ensemble set is removed from the subset, leading to some loss of books that our underlying recommender models do not take into consideration. While we acknowledge this limitation, in theory, it can simply be mitigated with a periodic data refresh in a real-world scenario.

Based on the star ratings, we classify each of the user's

book reviews into two categories: ratings of 4 and 5 stars represent good reviews (indicating that the user liked the book), and ratings 1, 2, and 3 stars represent mixed reviews (indicating that the user may not have liked the book). We use this binary classification at a later stage to train the ensemble model.

3. Methodology

3.1. Machine Learning Models

Our book recommendation algorithm ensembles six unsupervised models to capture various aspects of user preferences and book characteristics, and these recommendations will then be combined into an ensemble model. The individual models are summarized below:

1. *Jaccard similarity*-based user similarity model to find other users with similar favored books and recommend books based on the tastes of these users;
2. *Latent Dirichlet Allocation (LDA)* to discover topics from positive user reviews and then match books with users' preferred topics;
3. *Sentence2Vec* to generate vector representations of book descriptions and recommend books that are semantically similar to the user's liked books;
4. *Word2Vec* to generate vector representations of book genres and recommend books that are similar in overall themes to the user's liked books;
5. Huggingface BAAI general embedding *(bge)-small-en-v1.5* to generate vector representations of book titles and recommend books that are semantically and contextually similar to the user's liked books;
6. *Euclidean distance-based similarity* on general book statistics (such as number of pages, book format, etc.) and recommend similar books as the user's liked books.

Section 3.2 explores the individual models in detail. Each model recommends 200 books per target user, which provides a good trade-off between sufficient level of detail, as any book recommended past the 200th one is likely to receive a score nearing zeros, diminishing their relevance, while still allowing for relatively quick computations with lesser storage space needed to store all the data.

Following which, Section 3.3 details the training of the ensemble model weights. This is done via binary classification for the recommended output of each model against the actual star ratings given by users, and the results are discussed in Section 4. Section 5 then explains our recommender system's user interface.

3.2. Individual Models

3.2.1. Jaccard Similarity (Similar-user Clustering)

The user similarity model (also used as our naive baseline model) aims to suggest books a user is likely to enjoy based on the positive book reviews that other similar users have provided to books via computing Jaccard similarity. This can be calculated as such:

$$J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|} = \frac{\|A \cap B\|}{\|A\| + \|B\| - \|A \cap B\|}$$

Based on the books that the target user has left a positive book review with four or five stars for, we identify similar users by finding the Jaccard similarity score between the target user and other users via pairwise comparison, indicating they have similar tastes in preferred books, and therefore could enjoy each other's preferred books. We then identify the top 20 users with the highest Jaccard similarity score with the target user, and all their reviewed books with a positive rating are compiled by adding the score equivalent to the book's user's similarity score. We then select the top 200 books with the highest scores as recommendations.

3.2.2. Latent Dirichlet Allocation (User Reviews)

This model aims to calculate book similarities based on user reviews. Typically, book similarities are estimated with book statistics clusters, but reviews are seldom leveraged. We believe that the language readers use to discuss their favourite books contain valuable insights. With this model, we process these reviews to create a list of topics, and then assign an array of these topics to each book.

We chose Gensim's Latent Dirichlet Allocation (LDA) model to process the user reviews due to its efficient handling of large amounts of text data with fewer memory resources (Huang, 2023) and built-in coherence score functionality (Greer, 2018). To ensure the model learns the correct keywords, expansive preprocessing is necessary. For instance, we remove reviews that only contain numbers and/or whitespace characters, have 'http' indicating links to external reviews, are less than 100 characters or are not in the English language. To further ensure better and more coherent definitions of each topic, we remove stop words , review-specific generic words like 'book', 'read', 'review', etc. (about 30 such words) identified during the model training stage, and words with less than three characters. Next, nltk's part-of-speech tagging was employed to identify and remove proper nouns. All reviews are then appended to one entry per book.

We proceed to tokenize the processed reviews to create a

mapping of word tokens to unique integer IDs, which is then transformed into a Bag-of-Words (BoW) representation. Ultimately, we model 15 topics (and their respective word distributions) as it gives the highest coherence score. A higher value of coherence score, which evaluates how semantically similar the highest-probability words within a given topic are (Bismi, 2023), results in more interpretable topics (Roepke, 2022). The results of the modeling can be found in Figure 1 and Table 1. Hence, we create a fixed vector of size 15 for each book containing the topic probability distributions to generate recommendations for a target book. This is done by calculating the cosine similarity (using cosine angle) between the target book and all the other books in the sub-dataset, which is a measure of their perceived similarity in content and reader preferences.

$$\text{Cosine Similarity} = S_C(A, B) = \frac{A \cdot B}{\|A\|\|B\|}$$

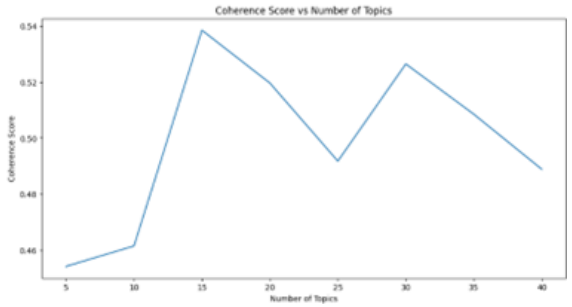

Figure 1. Breakdown of individual model composition

3.2.3. Sentence2Vec (Book Descriptions)

The objective of employing the sentence-to-vector (Sentence2Vec) model is to transform the descriptions of books into distinctive vector representations. By doing so, we can leverage the cosine similarity between vectors to identify books whose descriptions are similar to those preferred by the user. The Sentence2Vec model is able to capture the meaning of entire sentences, making it more efficient in capturing the semantic content of varied length. This makes Sentence2Vec suitable for application on book descriptions, which are typically a few sentences long.

The `all-MiniLM-L6-v2` model was chosen as it offers good performance, with a decently fast speed (Reimers, 2024). This model leverages an efficient MiniLM architecture, which is still able to retain most of the understanding capabilities of the larger transformer-based models like raw BERT (Reimers, 2024; Grebennikov, 2023).

The model produces normalized output embeddings of 384 dimensions (Reimers, 2024). Consequently, calculating the cosine similarity between all vectors is computationally in-

Table 1. Topic Words with Weights

Topic	Words
0	science, human, question, society, fiction, body, future, social, ideas, space
1	murder, case, crime, kill, killer, police, thriller, solve, dead, detective
2	father, mother, husband, marry, historical, women, daughter, wife, sister, marriage
3	annoy, guess, hate, bore, okay, half, wrong, suppose, instead, either
4	town, small, house, water, animals, dream, island, travel, night, city
5	fantasy, magic, adventure, power, fight, game, battle, trilogy, kill, ship
6	normal, vampire, ghost, fairy, human, vampires, witch, evil, magic, power
7	american, women, white, political, culture, country, black, slave, society, state
8	perhaps, narrative, language, narrator, fiction, prose, graphic, theme, often, original
9	sweet, sexy, scenes, chemistry, heroine, exchange, hero, chance, romantic, absolutely
10	emotional, tear, pain, emotions, struggle, death, truly, dark, cry, journey
11	funny, laugh, humor, hilarious, loud, entertain, listen, cute, light, music
12	include, information, memoir, personal, research, essay, provide, women, christian, cover
13	intrigue, receive, slow, question, exchange, develop, excite, future, fast, unique
14	school, parent, children, kid, high, girls, mother, adult, child, father

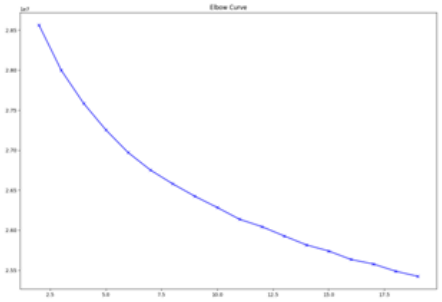

Figure 2. Elbow curve for Sentence2Vec model

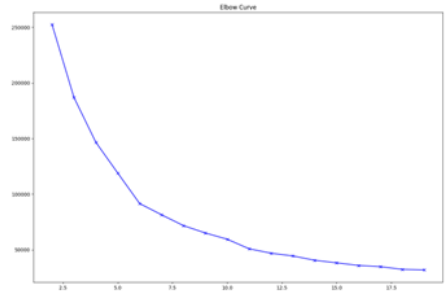

Figure 3. Elbow curve for Word2Vec model

tensive. To solve this, before calculating similarities, we apply principal component analysis (PCA) that reduces dimensionality from 384 to 108 dimensions while retaining 75% of the total variance explained by the original embeddings.

To further optimize the computations, we use K-Means clustering to group books with similar embeddings together. Determined via the elbow method as plotted in Figure 2, which involves plotting the variance explained by different number of clusters and identifying the 'elbow' point (Saji, 2024), we clustered the embeddings into six clusters. This tactic reduces runtime since doing pairwise comparisons in a $\frac{n}{6} \times \frac{n}{6}$ dimensional space is 36 times faster than iterating within a $n \times n$ space. This method improves the performance while not compromising on model accuracy, because similar book descriptions will be close in terms of cosine similarity, and consequently, it is highly likely that two similar books are in the same cluster.

Finally, we generate the recommendations for each book ID, iterating within its cluster only. Like in the LDA model, we generate recommendations for each of the user's positively reviewed books in the sub-dataset via cosine similarity of the embeddings, and then take the top 200 books with the highest similarity scores as this model's recommendations for a user.

3.2.4. Word2Vec (Book Genres)

In this dataset, each book has a varying number of genres tagged by users, with a total of ten different genres to choose from. The goal of this model is to discover similarities in books depending on their genres. We convert the genre information for each book into a list of genre keys using a Word2Vec model with a vector size of 100 to learn vector representations for these genres. The objective of this is to capture the semantic relationships between genres in a high-dimensional space.This method was preferred over one-hot encoding as Word2Vec is better able to capture the relationship between genres. For instance, the genre 'fiction' is more likely to be related to 'fantasy, paranormal' than it is to 'non-fiction'.

We then weigh the embeddings of each array by the percentage of times the user has tagged that genre, to scale the values accordingly, and add the arrays of all genres together to get the weighted genre vector. An optimisation process similar to that of the Sentence2Vec model was done here as well to speed up computation time: With a KMeans clustering method, we group books with similar genre vector representation. The 'elbow' point was determined to be six groups as well as shown in Figure 3. This allows us to only search the most similar books within the near neighborhood of a book, which is more efficient than comparing a vector to all others.

3.2.5. Embeddings (Book Titles)

For book titles, the hypothesis is that users will want to read books with similar titles to them. This is because book titles can often contain niche information, often including the subject of the book or named characters ("*Harry Potter and the Prisoner of Azkaban*", "*The Lord of The Rings: Return of The King*"), or specific anecdotes ("*Think and Grow Rich*", "*Rich Dad Poor Dad*"), or clearly defining what the book talks about ("*Atomic Habits*", "*The Subtle Art of Not Giving A F*ck*"). This information can be captured inside a vector, and similarity comparison can be done to find the most similar books.

Based on the Massive Text Embedding Benchmark (MTEB) leaderboard (Huggingface, n.d.), `bge-small-en-v1.5` embeddings were chosen as a text-to-vector embedder. This is because it is ranked 37th, and while other embeddings potentially outperform it, it has a decent performance without sacrificing memory usage and model size. Moreover, since all books in our dataset are English, directly using the model with English language is sufficient for our use case.

To process book titles for similarity comparisons, we first convert them into a vector space using `bge-small-en-v1.5`. This is done by creating a list of document objects, where each document contains the book title and associated metadata, including the book id. These

documents are then embedded into a high-dimensional space. To manage and retrieve these embeddings efficiently, we employ Facebook AI Similarity Search (FAISS), an advanced library designed for fast similarity searching of dense vectors compared to other databases such as Chroma and Pinecone.

The titles are concatenated into a single string in random order, separated by a new line "\n". This is used to do the similarity search. Then, our algorithm is able to perform a similarity search for all of the books in the database to recommend 200 books per user that are the most similar to their current list.

3.2.6. Euclidean Distance (Book Statistics)

This model aims to cluster books by general statistics available by book:

- Average Rating reflects the overall reader approval and enjoyment level of a book;
- Number of Ratings indicates the popularity or the amount of reader engagement a book has received;
- Number of Text Reviews offers insight into how popular a book is;
- Number of Pages helps in matching books of similar length or depth;
- Publication Year allows us to recommend books from similar periods or contexts; and
- Genres provides a basic thematic linking between books.

While we originally considered book formats as well, books tend to be produced in more than one format (hardback, digital, audiobooks). Therefore, it should not be a limitation. For the preprocessing of genres, the value was calculated as a percentage of user-assigned tags for that genre over all the user-assigned tags for each book. The Euclidean distance between two books was calculated as a metric of similarity.

$$Euclidean(u, v) = \sum_i (u_i - v_i)^2$$

To ensure standardization with the other models in which a larger number represents higher similarity, a homographic function $\frac{1}{1+x}$, where x is the Euclidean distance, is applied, scaling the similarity score between $[0, 1]$. Similar to the genre, description, and reviews models, our algorithm then does book-wise comparisons and outputs the top 30 most similar books.

3.2.7. Output Standardization

With all our underlying models in place, we then proceed to compile their recommendations. Out of the six models, two (user similarity and title embedding) take in target users as inputs, while the other four (genre embedding, reviews embedding, description embedding and clustering of general book statistics) take in target books as inputs.

Therefore, to convert the four models to take in target users as inputs as well, for each target user, we run the algorithm with all the positively rated books (denoted by star rating of 4 or 5). The algorithm then calculates the total similarity score for each book based on the user's previously reviewed books, normalizes these scores by the number of books reviewed, and then selects the top 200 books with the highest normalized scores for each user, compiling these into a consolidated DataFrame.

The output of each of our six underlying models is then fed into a supervised ensemble model, which is then used for determining the appropriate model weights.

3.3. Ensemble Model

After we generate the similarity score and recommended books for all users for each of the six models, we merge them together to form a final recommendation output for each user. The main objective is to determine the weightage of each individual model for the best results, as well as to compare the performance of individual models against the ensemble model.

For model training, we used the sub-dataset used to train the additional models with unseen training data, summing to 94% of the entire dataset (1,745 users). The remaining testing dataset (6% of the entire dataset, or 112 users), is used to validate and choose the best model.

In reality, the metric to be used for model training and validation is click-through rates of whether users like the recommended books or not. However, since we are unable to do so, we take the user's reviews as an indication of whether they would have been inclined to click on and accept the recommended book. Hence, the ratings of 4 and 5 are mapped to positive sentiment of the book ($binary = 1$), and ratings of 1, 2 and 3 are mapped to negative sentiment of the book ($binary = 0$) by each respective user. Therefore, during the model training stage, the six models intentionally do not exclude recommending books that have already been read by each user, but they will be excluded in our final product. This reduced the training and testing set by 86.63% and 87.47% respectively. Nevertheless, there is still a decent dataset of 229,786 and 13,296 rows in the training and testing set respectively.

However, each model stores 200 recommended books per

user to reduce the computation time and space required, but there are 109,296 books in the subset. Hence, it leads to a very sparse matrix, as shown in Figure 4.

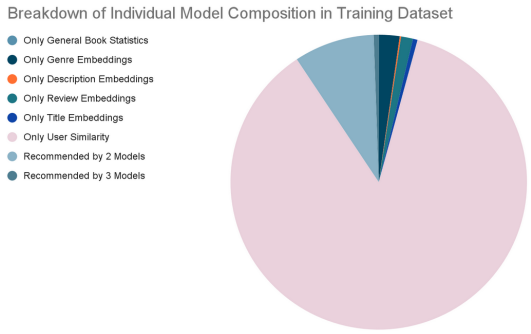

Figure 4. Breakdown of individual model composition

90.03% of recommendations are only recommended by one model, 8.77% of recommendations recommended by two models, and only 0.55% are recommended by three models. The user similarity model heavily dominates the dataset as well, with 84.51% of the training sets respectively consisting only of books recommended by the user similarity model. This is likely because Goodreads, where the dataset came from, recommends books based on user similarity modeling, possibly in an algorithm similar to ours. Hence, it is logical that user similarity will account for the bulk of books reviewed.

Furthermore, all *NaN* values caused by not all models having suggestions for all rows were replaced with 0.

In our case, it is important to ensure the books that are recommended are books that users will enjoy and leave good reviews for. Hence, precision, which measures the accuracy of positive predictions, is the most important metric.

We used Python's Scikit-Learn library. We first applied StandardScaler to standardize the training data. Then, several classification models were tested, including Linear Support Vector Machine Classifier (Linear SVC), Logistic Regression (with balanced classes), Ridge Classifier and Stochastic Gradient Descent Classifier. The metrics of each model is available in Table 2.

Out of these, LinearSVC and Logistic Regression performed the best, with the highest precisions of 0.856 and 0.855 respectively. Hence, we decided to use Logistic Regression as our final model, as it performs better on other metrics, with a higher accuracy, indicating the model's performance is closer to the true value. Logistic Regression is also highly explainable, which is one of the objectives of our recommendation system as well.

Table 2. Metrics for each candidate model

Model	**Accuracy**	**Precision**
SGDClassifier	0.734582	0.735869
RidgeClassifier	0.734883	0.741674
LogisticRegression	0.670126	0.854682
LinearSVC	0.667043	0.855965

Model	**Recall**	**F1**
SGDClassifier	0.997035	0.846772
RidgeClassifier	0.981391	0.844857
LogisticRegression	0.664519	0.747699
LinearSVC	0.658078	0.744089

4. Discussion of Result

4.1. Comparison with Logistic Regression on Individual Models

Table 3. Coefficients for each candidate model

Model	*naive*	*rev*	*desc*
SGDClassifier	0.009377	0.014328	-0.119328
RidgeClassifier	0.273136	0.017124	0.003939
LogisticRegression	0.868651	0.073475	0.018187
LinearSVC	0.377111	0.031524	0.008723

Model	*genre*	*title*	*book*
SGDClassifier	-0.000777	0.006737	0.040987
RidgeClassifier	0.080223	0.110881	0.005648
LogisticRegression	0.295762	0.335316	0.017975
LinearSVC	0.120624	0.145180	0.007777

For Table 3, we map the column names to these models:

- *naive* – User similarity model
- *rev* – Review LDA model
- *desc* – Description Sentence2Vec model
- *genre* – Genre Word2Vec model
- *title* – Title embeddings model
- *book* – Book statistics similarity model

As seen from Table 3, for logistic regression, user similarity (column `naive`) has the highest coefficient of 0.8687, indicating it could be the most important model of the six. On the other hand, the model using general book statistics (column `book`) to calculate Euclidean distance has the smallest coefficient of 0.01798, hence it is the least important model.

However, the high weightage of the user similarity model could be due to 92.80% of the training rows having data for the user similarity model, while for the other columns, the matrix is decently sparse. For example, for Euclidean

distance between general book statistics, only 0.36% of training rows have data for this model. Therefore, the logistic regression model could have regarded this model as less important, while overstating the importance of user similarity score.

Due to the good performance and boons of Logistic Regression as described in Table 2 and the following paragraph, we have decided to go ahead with using **Logistic Regression** as the main classification model for our ensemble model.

4.2. Coefficients of Candidate Models

Table 4. Metrics of each individual model (Logistic Regression)

Model	Accuracy	Precision
User similarity	0.687876	0.847445
Review LDA	0.304302	0.806713
Description Sentence2Vec	0.734582	0.735869
Genre Word2Vec	0.298135	0.790909
Title embedding	0.726233	0.737543
Book similarity	0.735560	0.735701

Model	Recall	F1
User similarity	0.702045	0.767923
Review LDA	0.071268	0.130966
Description Sentence2Vec	0.997035	0.846772
Genre Word2Vec	0.062270	0.115450
Title embedding	0.974642	0.839676
Book similarity	0.999591	0.847581

Doing an analysis of the performance of individual models, it can be seen that in terms of precision, our ensemble outperforms all individual models. In particular, comparing it against our naive model of user similarity-based recommendation, our ensemble model achieves a higher precision of 0.8547 compared to the naive model with a precision of 0.8474. While it should also be noted that using other metrics such as accuracy, the naive model does perform slightly better (accuracy scores of 0.6701 for ensemble model versus 0.6879 for naive model), these differences can be attributed to the overwhelming data size of the user similarity model, which may lead to minor fluctuations in performance metrics.

Looking at recall, which indicates how good the model is at correctly identifying positive instances, it is interesting to note that review embedding and genre embedding have very low recall of 0.0713 and 0.0623 respectively. The low recall rates for the review embedding and genre embedding models suggest that these models are missing a significant number of positive instances, which can be problematic if correctly identifying positive instances is a priority.

F1 score is the harmonic mean between precision and recall, measures the reliability of a machine learning model. It is commonly used to measure performance of binary classification. Description embeddings, title embeddings and general book data have significantly better F1 scores (0.8468, 0.8397 and 0.8476 respectively) than the naive user similarity model's F1 score of 0.7679. Therefore, they are important in ensuring that while the recommendation system does not return books that users will not enjoy, they should not skip past too many books that users may enjoy as well, thereby increasing model robustness.

4.3. Ease of Computing

Comparing the computational resources required for each model, the ensemble model is computationally intensive due to several factors. Firstly, it requires training and storing embeddings for the four models that utilize embeddings. This process is resource-intensive and time-consuming. Additionally, four of the models in the ensemble require accessing recommended books for each target book, which further increases the computational load and training time. In contrast, the naive user similarity model is much more lightweight. It can be computed rapidly in real-time for each user request, making it suitable for dynamic updates and ensuring that it is always up-to-date. Overall, while the ensemble model offers superior performance, it comes at the cost of increased computational resources and time. However, it should be noted that once trained and stored, the ensemble model does not require many computation resources, and an occasional refresh of the meta-deta is sufficient to overcome this limitation.

4.4. Popularity Bias and Other Drawbacks

As mentioned in Section 1, one big limitation of the naive user similarity model is its tendency for popularity bias. We tested this hypothesis using recommendations in our test set. Using the count of text reviews per book, we generated the top 20 most popular books. Similarly, we generated the top 20 most recommended books by the ensemble model and user similarity model respectively. Of the top 20 most recommended books by the ensemble model, 9 were among the 20 most popular books, while 10 of the top 20 most recommended books by the naive user similarity model were also among the 20 most popular books. Hence, we can conclude that the user similarity model is more likely to recommend books that are popular and appear more on user's shelves, and is hence prone to popularity bias. However, our ensemble method addresses this challenge without distorting the recommendations. This is because it incorporates recommendations based on various information sources beyond user similarities, such as book descriptions, titles, genres, text reviews, and book statistics. This broader range of data helps to mitigate the popularity bias and makes our model robust.

Beyond the naive user similarity model, other individual models have their individual drawbacks as well. For example, the title embedding model may recommend irrelevant books based on superficial keyword matches (e.g., "*When Breath Becomes Air*" vs. "*Avatar: The Last Airbender*"). Likewise, a model that heavily relies on review embeddings could struggle with sarcasm, potentially recommending works entirely different from the user's preference.

All in all, the ensemble method improves robustness by mitigating individual model biases and fostering a broader, more user-centric approach to book recommendations. This is also why explainability in our models are extremely crucial, to allow users to understand why each book is recommended, and for us to ensure the model does not have inherent biases.

5. Recommender System User Interface

To provide the user with an intuitive way to check the recommendations, we have created a user interface in Flask:

- **Home Page** – As shown in Figure 5, recommendations are initially generated using a test dataset. Once loaded, a new page displays users and a few of the books they have read, as shown in Figure 6. This allows the app user to better understand the readers taste profile.
- **Viewing Individual Users** – Clicking on a specific user takes you to a new page as shown in Figure 7, listing the model's recommendations for that user, presented as a book list, ordered by importance according to our ensemble model.
- **Understanding Recommendations** – Clicking the "Explain" button provides insights into why particular books are recommended. Since our ensemble model uses linear regression, we leverage the model coefficients for this task. A waterfall chart, as shown in Figure 8, will then be displayed, highlighting the importance of each variable used in the recommendation process.

This interface serves as a proof of concept for both business users and readers. Business users gain a comprehensive understanding of why specific books are recommended, providing valuable insights into user preferences. We also believe that click-through rates are likely to increase, since readers will be more compelled to explore suggestions if they come with a valid reason or explanation. This explanation makes the user interface more transparent and fosters trust in the recommendation system.

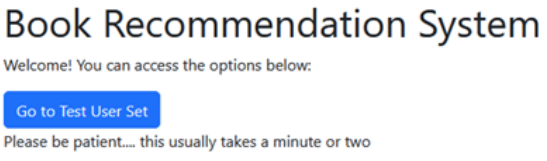

Figure 5. Recommender System UI Homepage

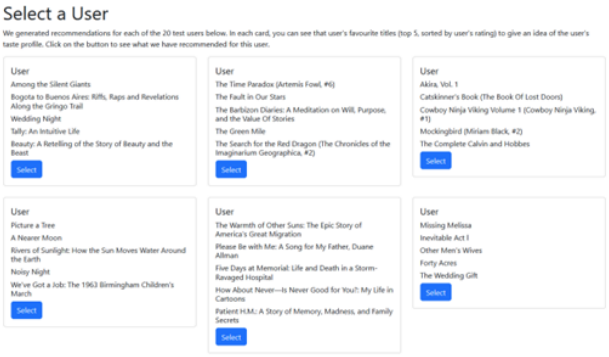

Figure 6. Test users and 5 of their favorite books

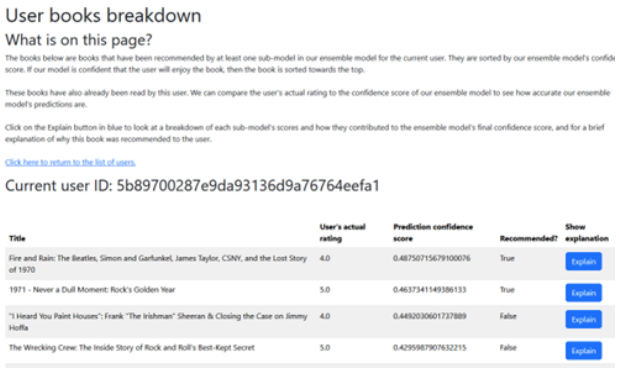

Figure 7. Viewing individual users

5.1. Explainability

The logistic function is as follows:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots)}}$$

In our ensemble model's final classifier, the vector of all sub-model's confidence scores (that a given book is a good recommendation) is x. The vector of corresponding coefficients is then β, where the intercept is β_0.

The values in the waterfall chart are each sub-model's scores multiplied by their coefficient in the Logistic Regression model, $\beta_i x_i$. Due to the linearity of the Logistic Regression model, these values conveniently represent the impact of each sub-model on the ensemble model's final score.

Figure 8. Understanding recommendations

6. Limitations and Future Work

Our book recommendation model, while effective in many respects, has certain limitations that should be acknowledged to better understand the outcomes and discover potential areas for future improvement.

As stated in Section 2.3, using a limited subset of data to avoid data leakage means that about 30% books in the dataset were excluded, and cannot be recommended. This exclusion potentially limits the diversity and range of book recommendations. A possible solution, since the weightage of each model has been determined, is to redo this process with all the books in the dataset, and adjust the weightages via other methods like click-through rates by actual users.

Currently, we use a single type of data to train our model. For example, we have one model for titles, and another for description. This is mainly because of computation resource limitations, and the selected model is the most suitable for that specific type of data. However, integrating multiple subsets could potentially improve the robustness and accuracy of our recommendations. Combining models trained on different subsets of data could capture a broader range of user preferences and book characteristics, and improve the overall performance of our recommendation system.

The model currently struggles with the cold start problem for new users without a review history, for which we will be unable to give accurate recommendations. Ideally, as a future workline, we want to integrate demographic and psychographic data into the initial user profile setup, which could offer a preliminary basis for personalization. This approach would improve our model's ability to serve new users effectively from their first interaction. However, we currently lack access to this type of information and new data sources should be searched in the future.

Another important limitation is that our model does not use authors' data. Given that many readers have preferences not only for specific genres but also for the writing styles of certain authors, this could be a critical problem. In the future, including information about authors in the model could improve its personalization capabilities since it would allow for recommendations that consider the user's preference for books by particular authors or within specific literary schools. Additionally, we could improve our book information database by scraping data from external sources. Potential additions might include awards won, appearances in thematic blogs or Subreddits, or the number of mentions on Twitter.

7. Conclusion

In this report, we explore how Natural Language Processing (NLP) can transform book recommendations for platforms like Goodreads. We aim to overcome limitations of current systems by analyzing text details from unstructured text data like book reviews, titles or descriptions, allowing us to analyze user data and book features from different angles, enriching the recommendation process. Through careful data preparation and various machine learning models, we build an ensemble model, which we evaluate using holistics factors such as precision, computational resources required and model robustness.

Overall, while our ensemble model performs better than the naive user similarity model that Goodreads currently uses in terms of precision, it performs worse on other metrics such as f1 score and accuracy, and requires more computation resources. However, while we acknowledge the user similarity model gives decent results, our ensemble increases model diversity and robustness, reducing the fundamental popularity bias in user similarity model, which is extremely important in an environment where users are looking for increasingly niche books.

Furthermore, we believe that with further training, especially via users rating the books that we have recommended to them, we are able to improve the model training, where the target data is currently extremely imbalanced and primarily consists of user similarity data. Future work could entail expanding the data, including more user and book metrics, and using higher volumes of data. Ideally, in the future, we will also deploy various ensemble models in production and gather real user feedback.

The potential to integrate this system into real-world applications is exciting. Once deployed, continuous updates based on user feedback will be crucial for ongoing improvement. This project lays a strong foundation for further research and development in personalized book recommendations, leading the way for a more enjoyable book discovery experience.

References

Allen, T. Spotify's Play for Audiobooks: Challenging Amazon's Audible Dominance — The AI Journal — aijourn.com. https://aijourn.com/spotifys-play-for-audiobooks-challenging-amazons-audible-dominance/#:~:text=As%20of%20May%202023%2C%20the,of%20Amazon%27s%20book%20publishing%20arm, 2023. [Accessed 05-05-2024].

Allied. Recommendation Engine Market Size — Forecast - 2031 — alliedmarketresearch.com. https://www.alliedmarketresearch.com/recommendation-engine-market-A14635#:~:text=The%20global%20recommendation%20engine%20market,32.1%25%20from%202022%20to%202031., n.d. [Accessed 05-05-2024].

Bismi, I. Topic Modelling using LDA — iqra.bismi. https://medium.com/@iqra.bismi/topic-modelling-using-lda-fe81a2a806e0, 2023. [Accessed 05-05-2024].

Garai, S. Maths Behind Collaborative Recommendation System: A review of Goodreads Book dataset — swetaprabha. https://medium.com/@swetaprabha/introduction-cbc02aedc03d, 2022. [Accessed 05-05-2024].

Goodreads. Announcing Goodreads Personalized Recommendations — goodreads.com. https://www.goodreads.com/blog/show/303-announcing-goodreads-personalized-recommendations, 2011. [Accessed 05-05-2024].

Goodreads. Audiobooks - Archives: Audible Recommendations Showing 1-32 of 32 — goodreads.com. https://www.goodreads.com/topic/show/1546732-audible-recommendations, 2013. [Accessed 05-05-2024].

Goodreads. Goodreads Help — help.goodreads.com. https://help.goodreads.com/s/question/0D51H00005af4TnSAI/why-are-the-book-recommendations-so-bad, 2020. [Accessed 05-05-2024].

Grebennikov, R. From zero to semantic search embedding model — blog.metarank.ai. https://blog.metarank.ai/from-zero-to-semantic-search-embedding-model-592e16d94b61, 2023. [Accessed 05-05-2024].

Greer, B. Sklearn LDA vs. GenSim LDA — benzgreer. https://medium.com/@benzgreer/sklearn-lda-vs-gensim-lda-691a9f2e9ab7, 2018. [Accessed 05-05-2024].

Huang, K. H. Gensim — Topic Modelling in Python — joloiuy. https://medium.com/@joloiuy/gensim-topic-modelling-in-python-1492a3e9a873, 2023. [Accessed 05-05-2024].

Huggingface. MTEB Leaderboard - a Hugging Face Space by mteb — huggingface.co. https://huggingface.co/spaces/mteb/leaderboard, n.d. [Accessed 05-05-2024].

Mordor. Product Recommendation Engine Market - Size, Companies & Share — mordorintelligence.com. https://www.mordorintelligence.com/industry-reports/recommendation-engine-market, n.d. [Accessed 05-05-2024].

Naghiaei, M., Rahmani, H. A., and Dehghan, M. The unfairness of popularity bias in book recommendation, 2022.

Pragathi., Sangam, A. S. V. A. A. G. A. Hidden bias in book recommendation. https://www.jetir.org/papers/JETIR2305176.pdf, 2023. [Accessed 05-05-2024].

PyPI. langdetect — pypi.org. https://pypi.org/project/langdetect/, 2021. [Accessed 05-05-2024].

Reedsy. Goodreads vs. StoryGraph: Which is Better in 2024? — reedsy.com. https://reedsy.com/discovery/blog/goodreads-vs-storygraph, 2024. [Accessed 05-05-2024].

Reimers, N. Pretrained Models Sentence-Transformers documentation — sbert.net. https://www.sbert.net/docs/pretrained_models.html, 2024. [Accessed 05-05-2024].

Roepke, B. How to Build NLP Topic Models to Truly Understand What Customers Want — dataknowsall.com. https://dataknowsall.com/blog/topicmodels.html, 2022. [Accessed 05-05-2024].

Saji, B. Elbow Method for Finding the Optimal Number of Clusters in K-Means — analyticsvidhya.com. https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/, 2024. [Accessed 05-05-2024].

Spotify. Your Taste Profile - Spotify — support.spotify.com. https://support.spotify.

com/us/article/your-taste-profile/, n.d. [Accessed 05-05-2024].

Straits. Recommendation Engines Market, Size, Trends, Share, Forecast to 2030 — straitsresearch.com. https://straitsresearch.com/report/recommendation-engines-market, n.d. [Accessed 05-05-2024].

Wan, M. Goodreads Datasets — mengtingwan.github.io. https://mengtingwan.github.io/data/goodreads.html, 2017. [Accessed 05-05-2024].

Y., M. 9+ Audible Statistics: Subscribers, Market Share & .. (New) — onlinedasher.com. https://www.onlinedasher.com/audible-statistics/, 2024. [Accessed 05-05-2024].