INDENG 243 Project: GoodReads: Book Analytics & Recommendation System

Group 18: Yanbo Wang[1], Yinuo Hu[1], Jiaming Xiong[1], Shichen Wu[1], Xinlin Huang[1] and Wenxuan Yang[1]
[1]IEOR Department, UC Berkeley
{yanbo.wang, evahuyn, jiamingx, shichenwu, xinlin_huang, ywenxuan616}@berkeley.edu
May 06, 2023

1 Introduction

GoodReads.com[1], an American social cataloging website and a subsidiary of Amazon, has emerged as the world's largest platform for book enthusiasts that offers a comprehensive database of books, annotations, quotes and reviews. While the website provides an extensive collection of books, users may still struggle to find books that match their personal reading preferences due to the overwhelming volume of information available. The searching process can be time-consuming and exhausting, leading to a demand for a more efficient and personalized recommendation system.

In this report, we aim to address this issue by building an intelligent book recommendation system based on book analytics. We first deployed a back-end web crawler to collect up-to-date book data and then explore various recommendation models to provide insightful advice based on users' past reading history. We also leverage Natural Language Processing (NLP) models for tag filtering and extracting summarization from reader reviews which enable users to quickly have a snapshot of book content. As a final outcome, we built an interactive online interface that enables users to play with our recommendation engine and customize their next to-read book lists.

2 Data collection & cleansing

2.1 Data Collection

We scraped data directly from *GoodReads* website to ensure the data quality is under control and get enough volume for future analysis. But this time the web scraping is a little bit challenging, so we followed the following train of thoughts to retrieve data in a more structured way.

We started from scraping book genres (40 genres in total) and scraped around 100~300 books under each genre with these books' basic information (e.g. title, author, rating, etc.) and corresponding 15 usable reviews left by readers (e.g. ratings, review texts, etc.). Apart from this information, *GoodReads* also provides some statistics regarding the book dating back to 6 months ago, e.g. number of times added to shelves. We also scraped these meaningful data to help analysis.

Therefore, we finally have scraped in total 3 datasets: ***Book information***, ***Book reviews*** and ***Book statistics***. For more detailed information, please refer to Appendix.

2.2 Potential Limitations of Datasets:

- Compared to the original website, our data volume might not be large enough to make the best recommendations. But we are trying to balance the precision of decisions and the time effort needed for data collection and model training;
- Data comes from a website that book lovers will leave their opinions, which might not be that representative for the general public's reading tastes;
- Incomplete book statistics history, we can only retrieve the past 6 months data, but this might be fine if we simply make up-to-date recommendations.

2.3 Data Cleansing

For numerical features like number of followers and rating distributions, we cleaned the useless suffixes and cast them into integer or float numbers.

For text data, such as book genres, the most direct approach is to use regular expressions to match the content. For easier classification, we choose the top 40 genres as the main genre for books and the others are regarded as sub genres since it is in nature for a book's content to be the cross of multiple genres.

However, for attributes like '*Award*', the tricky situation is that the source from html is a string of list-like objects, but the result we want is a list, the elements in which are combined string tags. There are too many variations from html sources, making regular expressions an unrealistic way. So our final approach is to first remove punctuation but keep the comma at this step. Then we turn them into lower case to remove duplicates and finally split the string on the comma kept from before and replace the empty space by underline for better format.

Finally we get the cleaned datasets with **7776 unique books** and corresponding **114,063 reader reviews** with 6 months of book statistics data.

[1] GoodReads website: https://www.goodreads.com/

3 Recommendation models

3.1 Model settings

The goal of a recommendation model is to accurately recommend unread books to users through predicting users' ratings for books they have not read before.

<u>**Model data preparation:**</u>
For recommendation models use, we need to examine the unique reviewers in our datasets. Because *GoodReads* allow users to use the same username to register their accounts, we would like to use the combination of *reviewer name* and *number of reviews* they have written to help identify a unique user. In this situation, we can find 41271 users for 7776 books, but we only have 114,063 review ratings, that means **99.97%** data are missing, producing a super sparse rating matrix.

To relieve this situation, we noticed that some users with the same name but different review numbers are actually the same user! This problem arises from the fact that *GoodReads* did not update user information cascaded on some old book review pages. Therefore, we make an ***assumption* that the user with the same name and close number of reviews are the same user**, so we combine these review records under the same user.

As a result of this approach, the unique users greatly reduced from 41,271 to 27,581. To further mitigate the sparsity of the recommendation matrix, we only picked users with **40+** review records as our final experiment sets, increasing the known data ration from **0.03%** to **1.14%**.

<u>**Metrics:**</u> Recall@5 and Recall@10

Here we used the so-called **Top-N accuracy** metrics to evaluate our recommendation model performance. We chose ***Recall@5*** and ***Recall@10*** as our metrics and the following are definitions for them:

Recall@5: For one user, if we have 100 randomly selected books, the percentage of the interacted books in the test set will be ranked among the top 5 books by the model. (similar for ***Recall@10***)

3.2 Popularity Model (Baseline)

The baseline model for our recommendation system is the Popularity Model. This model is not actually personalized - it simply recommends to a user ***the most popular items that the user has not previously consumed***. As the popularity accounts for the "wisdom of the crowds", it usually provides good recommendations, generally interesting for most people.

Popularity Model (Baseline) Performance:

- **Recall@5 ≈ 12.46%**
- **Recall@10 ≈ 22.62%**

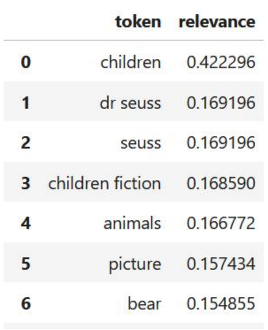
Nonetheless, the main objective of a recommender system is to leverage the ***long-tail items*** to the users with very specific interests, which goes far beyond this simple technique. Hence, we will delve deeper into the following two more powerful and theory-based models.

3.3 Content-Based Filtering Model

To construct more personalized recommendations, we employed the *Content-Based Filtering* approach. This method utilized a user's past interactions to create a profile of their preferences, making it robust to avoid the cold-start problem. Here we used a well-known technique in information retrieval named TF-IDF[2] to convert unstructured text into a vector representation, where each word is represented by a position in the vector, and the value measures ***how relevant a given word is for a book***; By representing all items in the same Vector Space Model[3], we can calculate the cosine similarity between books and consequently identify and recommend the most similar items to a user's previous selections.

We incorporated the *title*, *genres*, and *descriptions* of each book as its textual information for constructing the words' vector space. Then we take all the book files the user has interacted with and average them weighted by user's review rating to construct user profiles.

We find that the results of the Content-based model are easy to interpret and featured user profiles generated can help facilitate personalized recommendations. For example, when we examine the user profile of 'Abigail', it suggests that Abigail is interested in reading books about children, with frequently occurring keywords such as 'children' (relevance = 0.4223), 'bear' (relevance = 0.1549) , and 'cat' (relevance = 0.1146). Additionally, the presence of 'dr seuss' (relevance = 0.1692) in the profile underscores the user's affinity for this renowned writer of children's literature.



	token	relevance
0	children	0.422296
1	dr seuss	0.169196
2	seuss	0.169196
3	children fiction	0.168590
4	animals	0.166772
5	picture	0.157434
6	bear	0.154855

Figure 3.1: Snapshot of Abigail's user profile

[2] TD-IDF: https://en.wikipedia.org/wiki/Tf%E2%80%93idf
[3] Vector Space: https://en.wikipedia.org/wiki/Vector_space_model

Content-based Model Performance:

- **Recall@5 ≈ 36.39%**
- **Recall@10 ≈ 48.69%**

It indicates that around 36% of the items that the user interacted within the test set were included in the top-5 recommended items generated by the model from a list of 100 random books. This represents a significant enhancement over the baseline and serves as proof of the effectiveness of this content-based technique.

3.4 Collaborative Filtering Model

Collaborative filtering is a commonly used approach in recommendation systems that aims to predict users' preferences by leveraging the preferences of similar users. The core idea is to learn user-item interaction patterns directly from historical data. The underlying assumption is that users who have shown similar preferences in the past are likely to have similar preferences in the future.

The collaborative filtering algorithm typically employs matrix factorization techniques. Here we applied a popular latent factor model named ***singular value decomposition (SVD)***[4] to learn low-dimensional representations of users and items. This representation captures the latent factors that drive user preferences and item attributes. By multiplying these low-dimensional representations, the algorithm generates a predicted rating for each user-item pair.

The full Singular Value Decomposition is in the form:

$$M_{m\times n} = U_{m\times m} * \Sigma_{m\times n} * V^T_{n\times n} \ \ (Full\ SVD)$$

However, we can choose the number of latent factors k to factor the user-item matrix and approximate original matrix using the following schema:

$$M_{m\times n} = U_{m\times k} * \Sigma_{k\times k} * V^T_{k\times n} \ \ (Thin\ SVD)$$

The higher the number of factors, the more precise is the factorization in the original matrix reconstructions. But if the model is allowed to memorize too much details of the original matrix, it may not generalize well for data it was not trained on. Reducing the number of factors increases the model generalization. We loop over a potential space and find the best number of factors $k_{optimal} = 23$.

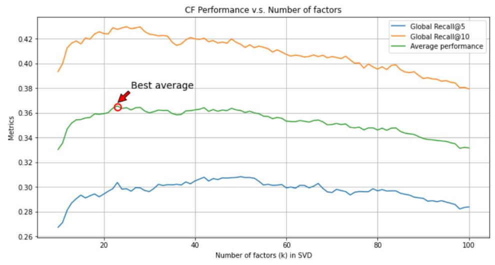


Figure 3.2: CF performance v.s. number of factors (k)

[4] SVD:https://en.wikipedia.org/wiki/Singular_value_decomposition

Collaborative Filtering Model Performance:

- **Recall@5 ≈ 30.37%**
- **Recall@5 ≈ 42.76%**

Also a great improvement from the baseline, but a little bit inferior to our content-based filtering model, possibly due to the great sparsity nature of our dataset. Methods like SVD, which are based on some mathematical theories, might not be able to take account of too many 'exceptions' in review ratings in the real world.

3.5 Hybrid Model

We anticipate ensembling models will produce a better result than single models, so we combine previous two models: Content-based filtering & Collaborative filtering together as a hybrid model.

Hybrid Model Performance:

- **Recall@5 ≈ 51.54%**
- **Recall@10 ≈ 62.10%**

Generally we can observe the highest performance in our final hybrid model:

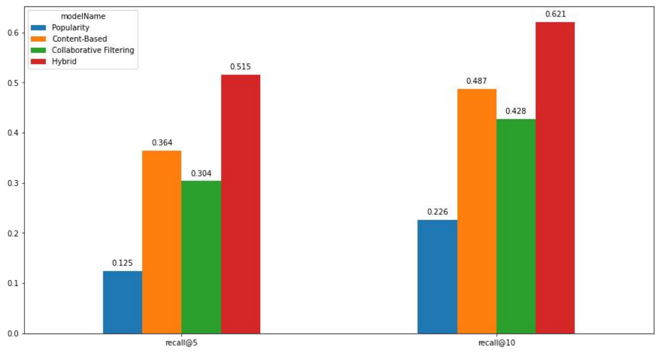


Figure 3.4 Model performance comparisons

Though our models demonstrate some impressive results, but we should pay attention to some potential limitations:

- Here we use ***Recall@N*** to measure the performance of models, but imagine for some 'extreme' user who is almost always being critical to books (i.e. always posts negative reviews), then our model only learns from the books relevant to him and it will be quite likely to recommend books that might not be the real favor of him;

- This can be extended to the cold-start problem for new users, if you only know one rating from this new user and that rating happens to be negative, the recommendation models might not work well;

- Therefore, currently we use users with 40+ ratings, but we will try to test our trained models on users with less ratings (eliminated before) to simulate this situation and see how our models work and discuss potential ways to advance performance.

4 Natural Language Processing (NLP) models

In this part, we consider some NLP models to prepare for the interaction tools that will be presented in the next module. The main functions are as follows:

- **Tag filtering system**: Extract tags for each book to enable users to filter books & comments by tags
  - Topic Modeling: LDA method
  - Keywords Extraction: TextRank
  - Key phrases Extraction: Yake!, keyBERT, etc.

- **Extractive Summarization**: Extract top review sentences to serve as a summary of all reviews
  - Baseline approach
  - Balanced summarization

4.1 Texts Pre-processing

Before we start the NLP models, we first conduct some pre-processing on our review content. In order to clean the review data, we pre-processed the textual data by removing punctuation, removing digits, converting letters to lowercase, removing stop words and frequent but less meaningful words. For the last step, we define a *drop probability* (idea from inverse frequency used in TD-IDF) to detect high-frequency words as follows:

$$Pdrop = 1 - \frac{t}{frequency}$$

where t is a very small number and we will drop the word has $Pdrop >= threshold$ to reduce model noise and accelerate training.

4.2 Tag filtering System

4.2.1 Topic Modeling: Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is an unsupervised learning method that helps us to assign documents to a particular set of topics and each topic will be characterized by a particular set of words. Since we have to set the number of topics to train the LDA model, we experimented with different options from 2 to 10.

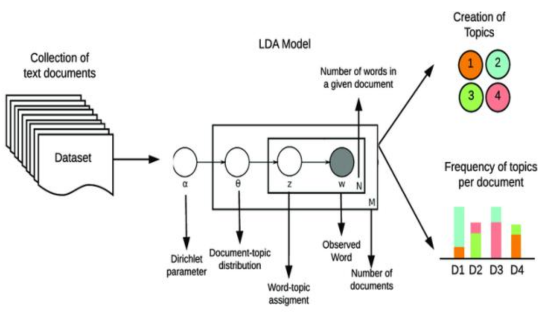


Figure 4.1 Latent Dirichlet Allocation (LDA)

To build an LDA model, we need to extract a corpus from our review data. We first map each normalized words with an integer ids as a dictionary. Then we convert the document (a list of words) into the bag-of-words format. After comparing different topic numbers, we recognized that the optimal topic number is 7.

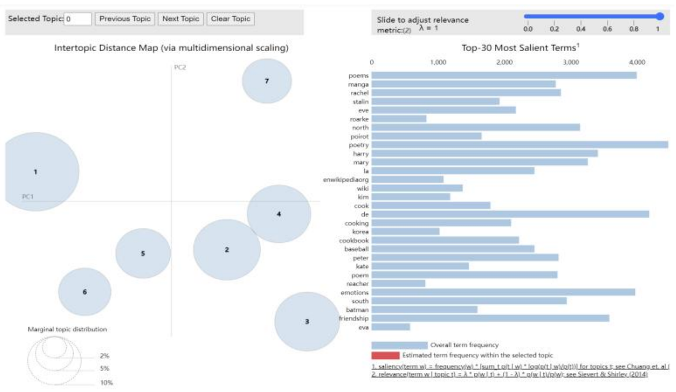


Figure 4.2 Screenshot of the trained LDA model

However, the problem is that the topic of each cluster is ambiguous, thus we cannot decide a specific topic name for each cluster. So we changed our mind to extract the keywords for each book instead of all the books together.

4.2.2 Keyword Extraction: TextRank

TextRank is a graph-based ranking model for text processing which can be used to find the most relevant sentences or keywords in text. Its idea is similar to *PageRank*, which is a powerful search engine algorithm. We employed *TextRank* to extract keywords of each book and classify each review containing the keyword to that tag since we hope the user could filter the reviews by each keyword.

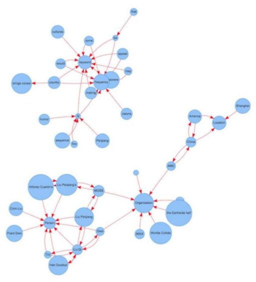


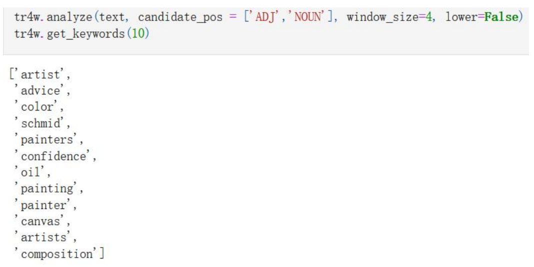
Figure 4.3 Illustration of TextRank

Since most of the words in a sentence are not useful to determine the importance, we only consider the noun and adjective words. Each word is a 'node' in *TextRank* and if we set the window size as k, the window will work like following:

- $Word\ vector$: $[\omega_1, \omega_2 \ldots \omega_n]$
- $[\omega_1, \omega_2 \ldots \omega_k]$, $[\omega_2, \omega_3 \ldots \omega_{k+1}]\ldots$

  are corresponding windows with size k

One example for book *Alla Prima: Everything I Know about Painting* is as following: you can witness that it generally catches important keywords like: ***artist, painting, canvas, color***, etc.



```
tr4w.analyze(text, candidate_pos = ['ADJ','NOUN'], window_size=4, lower=False)
tr4w.get_keywords(10)
```

```
['artist',
 'advice',
 'color',
 'schmid',
 'painters',
 'confidence',
 'oil',
 'painting',
 'painter',
 'canvas',
 'artists',
 'composition']
```

Figure 4.4 Example of TextRank key word extraction

As compared with the results of topic modeling, the results of keyword extraction are more typical and clearer, which could help better on the function of filtering reviews.

4.2.3 Other Trials: Key Phrases

We have also considered using ***key phrases*** instead of keywords as our tags and we have tried many methods like *Yake!*, *KeyBERT*, *Rake* etc. to extract key phrases, but the phrases we got are not very meaningful as a tag.

So we guess for these methods to get a better result, they might need more effort for tuning model parameters and improving the quality of original textual reviews, both could be time-consuming and have no promise for final result. Therefore, we conclude these methods are not very suitable and finally reject them.

4.3 Extractive Summarization

We explored two approaches of extractive summarization including a baseline model and an improved balanced summarization algorithm to extract top sentences from all reviews of a given book. With the extracted top review sentences, book readers can easily get an overview of the major opinions of other readers who have read the book and use them as references for their choices. Our two implementations and results of the two approaches can be summarized below:

4.3.1 Baseline: LexRank + Sentence-BERT embedding

The main idea of the baseline model is to calculate the cosine similarity of the sentence embeddings of all reviews and extract the top 5 sentences as the resulting summary, which can be further breakdown into the following steps:

1) Using Sentence-BERT to acquire sentence embeddings of each review sentence;

2) Using LexRank to get the graph-based "centrality" score of each sentence which is based on the cosine similarity score between sentences;

3) Extracting the top 5 sentences with the highest centrality score. The centrality score represents the degree of prominence the sentence compared to the whole review. If one sentence is similar to many other sentences in the review, we can claim that it is more central and important to the "center" of the review.

However, we noticed a few issues with the baseline model:

- First, with the sentiment scores calculated using TextBlob for each review sentence (ranges from -1 to 1 for negative to positive), we observed that the mean sentiment score of the extracted summary and the mean sentiment of the review texts of one book can be different and even far apart;

- Second, there may exist some redundant information among sentences in the extracted summary. Two or more sentences in the extracted summary may convey similar meanings. For example, "*Good book with clear structure.*" and "*Great book with organized structure!*" express similar meanings.

To handle the above issues, we propose a more **balanced** summarization algorithm below.

4.3.2 Balanced Summarization Algorithm

We use a greedy algorithm to extract a more balanced summary with three objectives

- **Maximize centrality score** which represents the importance of the sentence
- **Minimize** the difference between the summary sentiment and the overall sentiment of the original review texts
- **Minimize** the redundant information between extracted sentences

The learning objective of the model can be written as:

$$O(s, S, A) = Centrality(s) - CosSimilar(s, S) - SentDiff(s \cup S, A)$$

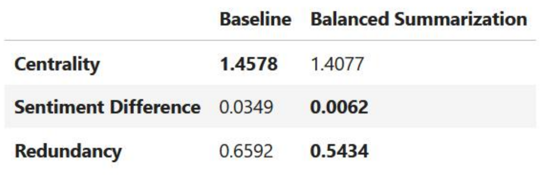
Where
- s represent the current review sentence
- S is the target extracted summary
- A represents all the review texts of specific book

Essentially, our goal is to extract a summary with high centrality, low redundancy, and a balanced sentiment.

4.3.3 Metrics and Results

Since our dataset does not contain gold summaries, we define three metrics to evaluate two methods:
- **Centrality**: average centrality of summary sentences (higher the better);
- **Sentiment Difference**: mean sentiment score of original text minus the mean sentiment score of the extracted summary (lower the better);
- **Redundancy**: average cosine similarity among summary sentences (lower the better).

The table below demonstrates the metrics comparisons:



	Baseline	Balanced Summarization
Centrality	**1.4578**	1.4077
Sentiment Difference	0.0349	**0.0062**
Redundancy	0.6592	**0.5434**

Figure 4.5: Metrics comparison between two summarizations

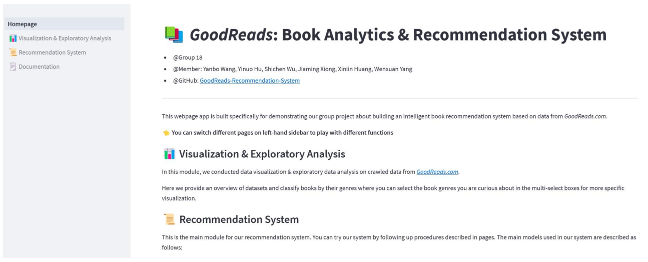
The result shows that our improved method achieved a much lower sentiment difference and redundancy score than the baseline model, though we sacrificed some centrality score. In general, the improved methods can provide users with a more balanced summary and more diverse information efficiently.

5 Interaction User Interface

In order to provide better user experience with our Book Analytics and Recommendation system, our group decided to develop an interactive web-based user interface and deploy it online such that every user can access our system and enjoy the functions to help build their next-to-read book lists.

You can directly access our system online through the following link:

GoodReads: Book Analytics & Recommendation System User Interaction Website



GoodReads: Book Analytics & Recommendation System

- @Group 18
- @Member: Yanbo Wang, Yinuo Hu, Shichen Wu, Jiaming Xiong, Xinlin Huang, Wenxuan Yang
- @GitHub: GoodReads-Recommendation-System

This webpage app is built specifically for demonstrating our group project about building an intelligent book recommendation system based on data from *GoodReads.com*.

You can switch different pages on left-hand sidebar to play with different functions

Visualization & Exploratory Analysis

In this module, we conducted data visualization & exploratory data analysis on crawled data from *GoodReads.com*.

Here we provide an overview of datasets and classify books by their genres where you can select the book genres you are curious about in the multi-select boxes for more specific visualization.

Recommendation System

This is the main module for our recommendation system. You can try our system by following up procedures described in pages. The main models used in our system are described as follows:

Figure 5.1 Screenshot of online interface

Target Audience & Impacts:

- **Non-technical book lovers** who simply wish to build their next to-read book lists. For this group, they can use our "Recommendation System" page and follow the prompts for several questions about their past reading history, and then our system will output a list of recommended books for them, which is easier to use and highly accurate to cater for different users' reading tastes.

- **Researchers and professionals** with technical background and expertise who wish to explore more reading patterns and statistical trends of book lovers for research use. For this group, they can investigate our "Visualization & Exploratory Analysis" page to view clear visualizations of the datasets and insightful analytics from various perspectives to comprehensively grasp readers' preferences.

6 Summary & Future plans

Based on the book analytics and Natural Language Processing (NLP) techniques, our team developed an intelligent book recommendation system for book enthusiasts who want to build their next to-read book lists. We explored various recommendation models and extracted textual information from book reviews using NLP techniques for more precise preference matching. For better user experiences, we developed an online interactive user interface to generate personalized book lists for users, saving them time in exhaustive searching and efficiently providing more personalized book recommendations. Overall, our project successfully achieved its goal of building an intelligent book recommendation system to enhance users' reading experiences.

As future work, we plan to broaden our book database, improve recommendation accuracy with more sophisticated models, and enhance the appearance and functions of our website to build even better user experiences.