Goodreads Recommender

Aarav Gambhir, Ava Knight, Elliot Jerng, Thaddeus Kopczynski
Professor John Rachlin
DS3500: Advanced Programming with Data
December 6, 2023

Abstract:

Using Goodreads book data, we created a book recommender that uses sentiment analysis and KMeans clustering to find similar books over a wide range of variables. Using this, we calculated a similarity score between a user-given book and titles similar to it using the distance of centroids within clusters. This algorithm was implemented into a dashboard to display recommended titles to users in an easy-to-read way.

Introduction:

Whenever I finish a really good book, I immediately go to find something just like it for my next read. I usually end up going weeks before finding a book that evokes the same excitement as the one that had left such an impression on me, but this is after I had already read through several mediocre books. I have always hoped that Goodreads or the like would implement a book recommender where I could put in the title of my latest read, and could get ideas for my next book choice with similar storylines, writing styles, and ratings. Similar to how Netflix calculates its x% match for movies you've seen and liked, we wrote a program that does the same thing for books based on Goodreads data.

Data Collection and Methodology:

Data was taken from the Goodreads top books of 2021 website and organized into data frames by genre. We used the BeautifulSoup library to scrape the data from the top books of 2021 website, as well as each individual book's web page to get more detailed information. We collected data we felt might influence similarity between books to curate the recommender including each book's title, author name, rating out of 5, number of pages, list of genres the book falls under, main genre the book is categorized under on the top books page, and description. The genres and main genre were cleaned using one-hot encoding for easier comparison and the book's descriptions were given a sentiment analysis score using NLTK's sentiment analysis package.

Books were then compared using KMeans clustering based on all of the above data, excluding title and author name. Our original program used the recommended 3 clusters given by the elbow graph shown in our program. This, however, provided mixed results with limited success, so we decided to focus on what we believed to be the most influential factor in book similarity: genre and subgenre. Given that there were about 10 main genres over the collection of books, we used a k value of 10 for our later calculation.

From the resulting clustering program and graph, we used the distance between centroids to calculate book similarity. Given a book title, the recommender calculated the distance between the points in that cluster, due to the other given x factors, and calculated a percentage similarity

score. The recommender then displays the number of books with the highest similarity scores to the user (decided by the slider) , those being the most similar books to the given title.

The algorithm is displayed in a simple dashboard format for ease of use. A user can type a book title from the dataset into the search bar and select a number of recommendations from using the slider. This will then output the selected number of books with the highest similarity score to the given title and displays a bar chart beside with a breakdown of the subgenres of each recommended title. With this, the user can decide if they want to try a similar book from another genre or stick to the same genre. The books are also visible with their book covers; we utilized web-scraping from the OpenLibrary website, a free-to-public source that contains all the books. Using the name of the book and the author, we took the first search result that popped up and obtained the cover from that. If a cover didn't have an image, we inserted a placeholder image instead.

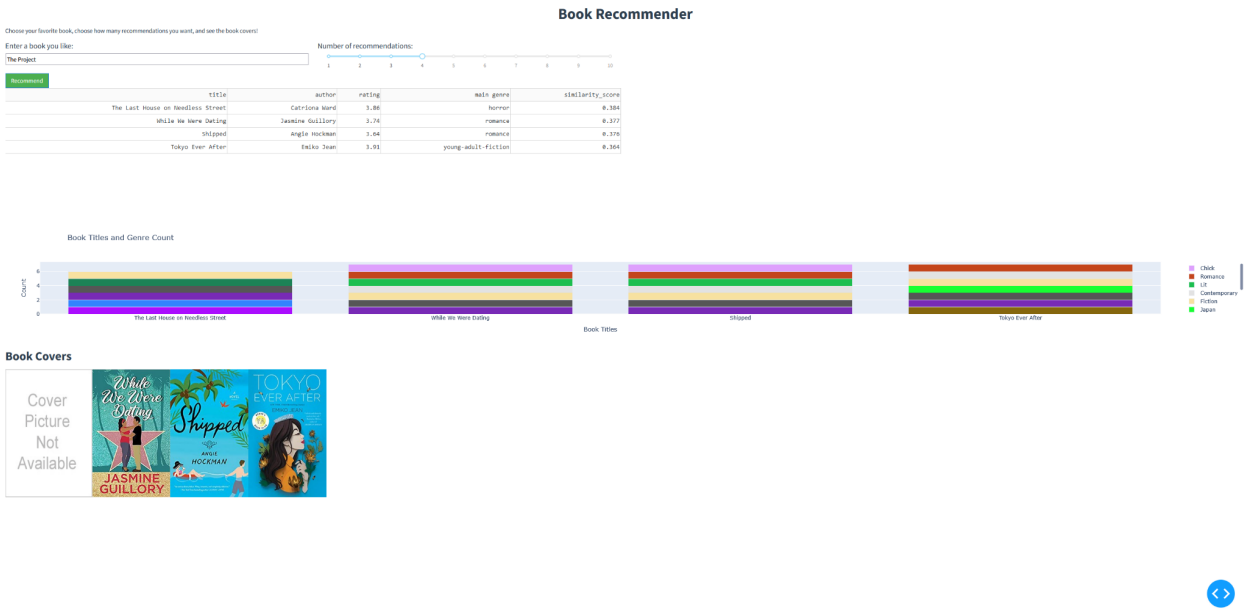

Figure 1: Dashboard Sample Implementation

Analysis and Future Improvements:

Many of the highest calculated similarity scores hovered around 0.4, or 40% similarity by centroid distance, which isn't very high for our program's intended purpose. We believe this is largely due to the small dataset of about 200 books which explains why there might not be many books that are extremely similar over such a small set of titles. However, as shown in Figure 2, this may not be a bad thing. The clusters shown in the PCA graph in Figure 2 do not depend solely on genre, indicating that the similarity score is not calculated solely based on similar genre, meaning the recommender offers a variety of titles to a user that are similar in more ways than just falling under the same genre. This may also explain why the similarity score isn't

extremely high: because books in different genres will likely differ in more ways than they will be similar.

Figure 2. Clustering of main genres.

To improve our program in the future, we would like to have a much larger sample of titles, ideally taking from the whole Goodreads website, to find many more similar books and multiple books by the same author or in the same series. We were limited by our novice web-scraping abilities and the layout of the Goodreads homepage not displaying a broad range of titles.

To further improve our recommender, we would like to include a more nuanced sentiment analysis of the book descriptions and even titles as opposed to simply negative and positive scores overall. This could result in more tailored recommendations due to degrees of mystery, romance, humor, etc in books under multiple subgenres. In addition to this, a calculation of feature importance could improve our recommender algorithm. We acted under the assumption that genre was the most influential factor in what makes books similar, but perhaps a reader might be a strict paperback-user and content does not matter as much to them. More user input aside from book title could tailor the recommender to each individual offering not only similar books but new ones to try that might be in their interest even if it is not something they had read before, learning based off of more than one book title. Additionally, the way we web-scraped the photos was highly inefficient; we were initially trying to find an API to obtain the book covers from, but we weren't able to find one that worked for it. We had to rely on individual web-scraping, which slowed the processes down.