Book Recommendation System

Sara Adra, Anika Das, Mirah Gordon, Genny Jawor
Northeastern University, Boston, MA, USA
GitHub Repository: https://github.com/mirahg/Book-Recommendation-Engine

Abstract

Our project focuses on using book reviews and user input to produce less mainstream recommendations. We approached this project by sourcing books and reviews from Goodreads, storing data in MongoDB, cleaning and organizing data with pymongo and pandas, as well as creating a simple user interface using Python (via a Jupyter Notebook), creating visualizations on review word frequency, lastly providing book recommendations using Neo4j. We created two subsets of the data, one with 100 and another with 10,000 randomly selected books and used frequency analysis of reviews to make recommendations. Our system was able to successfully make unique recommendations based on our favorite books as well as those of our peers.

Introduction

The purpose of this project is to create a recommendation system that went beyond the normal scope of a recommendation engine and would yield novel, less conventional recommendations. Instead of connecting books based on author or genre, we wanted to utilize user reviews, specifically focusing on the richness and depth already present in Goodreads data. By using review data, we aimed to create an exclusive recommendation for each user based on books we know they have enjoyed. The idea for this project came from the fact that our group consists of bibliophiles who wanted to find new and exciting books that would fit everyone's individual taste. We hypothesized that we could make a unique recommendation system using book reviews by performing frequency analysis on them and employing upvotes on reviews from the Goodreads dataset. While review systems are already vastly popular, creating something more personalized was an interesting challenge and it was fun to explore such a large dataset.

Methods

There were several different sources we could have used for project data, but we decided to utilize datasets from Goodreads compiled by UCSD [1]. Goodreads is a social platform that allows users to document books they read and provide a rating and review of the book. Other users can interact with these reviews by upvoting them to indicate that they like that review or found it helpful. Using both books with many reviews and also reviews with a significant number of upvotes allowed us to make more meaningful connections within our data, and ensure the data we used would ultimately be helpful towards recommending a new book. The original dataset sizes were huge; they contained about 2.36 million books from the Books dataset and over 15 million reviews from the Book Reviews dataset. The magnitude of this data was too

much to use for our project, so it was recommended for us to start with 100 books with 5 highly rated reviews per book and scale up to 10,000 books, also with 5 reviews. Preliminary cleaning included dropping some of the fields of the book document that we would not use in our exploration: work_id, isbn13, kindle_asin, and others.

We initially started by loading all the data into MongoDB and filtered down the datasets to only include books that would be worthwhile to sample from before we moved the data into Neo4j. This meant keeping only highly rated books, so any entries with an average rating less than 4.0 were dropped from the collection. Entries with less than 5,000 text reviews and 5,000 ratings count were subsequently dropped as well. This left us with a total of about 45,000 books, which was far more manageable for cleaning and fit our parameters for what makes a good book to recommend. We then created lists based on a random sampling of 100 and 10,000 books. A sample of the final version of the book data can be seen in Figure 1.

```
In [276]: books_100.head()
```

Out[276]:

title	text_reviews_count	average_rating	description	authors	publisher	num_pages	publication_year	book_id	ratings_count	author_name	book_size
Tandia	8	4.01	Half Indian, half African and beautiful, Tandi...	63	Mandarin	901.0	1992.0	1016896	53	Bryce Courtenay	large
Evil Twins (Case File 13, #3)	8	4.48	With thrills, chills, and laughs on every page...	1369219	HarperCollins	272.0	2014.0	18635043	53	J. Scott Savage	small
All the Lovely Bad Ones	775	4.02	Travis and his sister, Corey, can t resist a g...	10006	Clarion Books	192.0	2008.0	2294984	7588	Mary Downing Hahn	small
Hikaru no Go, Vol. 19: One Step Forward! (Hikaru no Go, #19)	8	4.25	An ancient ghost possesses Hikaru and unleashe...	8477	VIZ Media LLC	193.0	2010.0	6797429	530	Yumi Hotta	small
Areas of Fog	8	4.54	Poetry. One needs only to watch and listen in ...	721234	Shearsman Books	116.0	2009.0	6335957	72	Joseph Massey	small

Figure 1: Screenshot of dataframe head from the cleaned book dataframe

Next, we pulled positive reviews using the book_id from the random sample and looked at the most frequent words from each review compared with all the top reviews for a book [2]. Many of these words were stop words, or words with little connotation such as ‘I’ or ‘the’ which we eliminated from our data by using the Natural Language Toolkit (NLTK) stop words extension. It took a long time to retrieve the desired reviews from MongoDB as it cycles through the 15 million reviews for each book and pulls several reviews for each book. After running for a long time which yielded few results (125 reviews total), we decided to find a new way to sort reviews for faster matching by creating an index [3] to speed up the efficiency of the query. Now the query looks just at the index instead of the collection as a whole. The indexing produced much faster results so then we were able to move the data over to Neo4j to start querying. This indexing technique was further applied to other collections to improve queries across the board.

Analysis

The main outcome of our project is a Book Recommendation engine created using cypher queries on our Goodreads book and review data. This is presented through a Jupyter Notebook, where the user inputs four of their favorite books (the title and author of each), and through a connection to our recommendation engine in Neo4j, we print out a summary for the user ('based on your favorite books: A, B, C, D, our engine recommends the following books: X, Y, Z'). We also took into account the lengths of the books provided by the user in order to recommend books of similar length.

The user interface firstly connects with MongoDB to find the entries for the specific books inputted and uses the returned document to further query for the top reviews of each book (following the same protocol as we used in our code to train the engine). The frequency of words are analyzed on the reviews and once a list is compiled, a cypher query connects with Neo4j– ultimately producing book recommendations based on the user's input.

Once the data was loaded into Neo4j, the graph represented two nodes– a book and a common word– and an association between them. An example of this for one book can be seen in Figure 2.

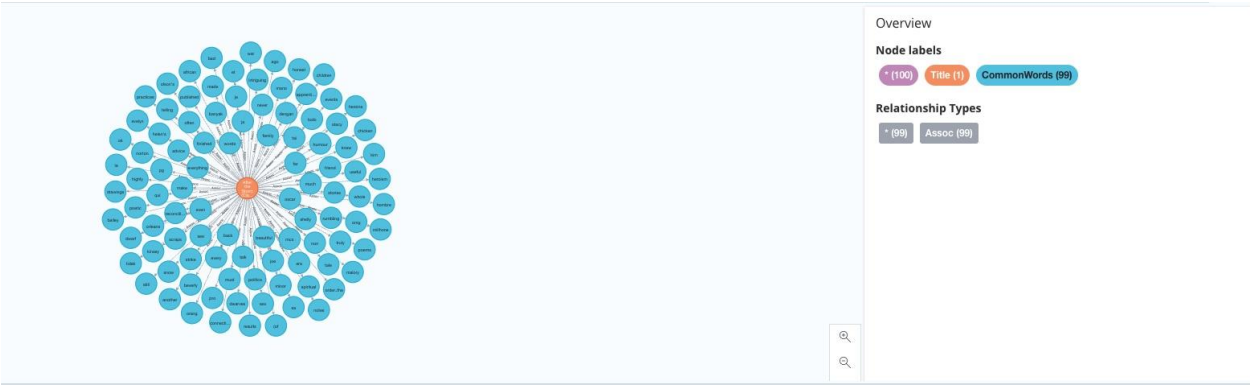

Figure 2: Screenshot of title node and 100 common word nodes associated with book title

The user will be prompted to enter four books and will then be shown an output of their recommended books, the interface for both segments for the user will look like those in Figures 3 and 4.

```
Please provide your 4 favorite books below:

Book 1:

     Title: 1984
     Author: George Orwell

Book 2:

     Title: The Martian
     Author: Andy Weir

Book 3:

     Title: Wool
     Author: Hugh Howey

Book 4:

     Title: Crime and Punishment
     Author: Fyodor Dostoevsky
```

Figure 3: User input of books

```
Based on your favorite books:

1984 by: George Orwell
The Martian by: Andy Weir
Wool by: Hugh Howey
Crime and Punishment by: Fyodor Dostoevsky

Our engine recommends the following books:

Charlotte by: David Foenkinos
M Is For Malice by: Sue Grafton
A Working Man (Men of Manhattan, #4) by: Sandrine Gasq-Dion
Holy Hustler by: P.L. Wilson
```

Figure 4: User output of recommended books

Conclusions

As testing the accuracy and validity of a recommendation engine is difficult, we used anecdotal testing to determine how well our engine worked. We each ran the engine on ourselves, using our top four books, and tried to evaluate the recommended books that the engine returned in an unbiased manner. For further testing– and to cast a wider net to include different types of books and reduce bias– we also asked friends and classmates for their help by providing us with their favorite books and evaluating the engine's responses to those as well.

We found that while the engine was able to return new book recommendations, they were not always the best fit to the given books or to the individual user. Reasons for this include our training data being a significantly small portion of all books (discussed further below) and common words appearing across many books, even if they are not incredibly related to one another. However, for the purposes and scope of this class, we would call our project successful: we were able to load and clean large datasets using MongoDB and load a minimalistic version of this data in Neo4j and ultimately could query on our graph and return novel results. If we had more time or resources, it would be incredibly interesting to explore how we could fine-tune our project to achieve greater results, but we are content with the current outcomes.

In terms of constraints, we were limited by our massive dataset and had to use small subsets of the books and reviews. We chose books randomly and had 10,000 books that we used in our evaluation, but it would have been more exciting to use 100,000 or 1 million books to find more accurate recommendations. Additionally, this dataset was last updated in 2019 before the pandemic, which means both datasets would be much larger today as new books and many reviews have been published. Another limitation of our work revolves around the simplicity of our current front-end / user interface. As mentioned earlier, the current design receives inputs from the user through a Jupyter Notebook, but ideally, this interface would be on an app or web application for increased accessibility and ease of use. Further, the current design is limited in that the user can enter anything into the title and author input boxes for their favorite books without any input validation. While we do clean the data to eliminate differences in capitalization

(between users as well as between user input and existing entries in the database), by standardizing the input to follow capitalization rules, there is still a lot of room for user input error. This ranges from misspellings or incorrect author-to-title submissions, as well as variations in inputs for the same books/authors between different users (ex. some may include an author's middle name or initial while others may not). In response to this limitation, we would like to include a drop down / autocomplete feature for user inputs in the future that would help eliminate this user error and input variation, while making the input process easier and more convenient for the user.

Author Contributions

Each of our four group members contributed to the successful completion of this project. After choosing a topic, we initially worked together to create a design for our book recommendation engine. After meeting with and soliciting input from professor Rachlin, we agreed that we wanted our book recommendations to go beyond basic recommendations and to provide our users with somewhat surprising results. From there we worked together to find the data that we wanted to use for our project and ultimately found publicly available datasets that contained a wealth of book and review data. We worked together to find convenient and agreed upon meeting times, and often screenshared to see the progress that was being made.

Mirah specifically contributed to the coding: namely loading data into MongoDB and filtering data through the local shell, continuing to clean and query data via pymongo in a jupyter notebook, and writing commands via pymongo again for the user facing script. She also worked with connecting Neo4j to the user interface script and building code between the mongo queries and the cypher query to return a book recommendation.

Genny contributed to a combination of the coding as well as the outlining and writing a bulk of the final report. In regards to coding, she aided in writing helper functions for cleaning the data as well as writing mongo commands to filter data inside the various collections. For the report, she wrote the introduction as well as methods sections, and contributed to editing the report overall.

Anika contributed to the project and final report by choosing and developing a topic through discussion with group members and working to find data relevant to our chosen topic. She also proactively and routinely reached out to group members regarding availability in order to schedule meeting times and worked to keep the project on track. In terms of deliverables, she and Sara created slides used for the second session of in class presentations, and documented portions of the final project report.

Sara also contributed to the code: namely creating the front-end user interface with input validation, calculating and storing the most frequent words found in each of the reviews, and debugging any issues that came up regarding data cleaning and cypher queries. She also helped write the Cypher queries for the Neo4J aspect of the engine. Sara also strongly contributed to the Analysis and Conclusion sections of the report.

References

[1] Wan, Menting. “UCSD Book Graph.” *UCSD Book Graph*, 2019, sites.google.com/eng.ucsd.edu/ucsdbookgraph/home.

https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home

[2] “Find the K Most Frequent Words from Data Set in Python.” *GeeksforGeeks*, GeeksforGeeks, 10 Dec. 2017, https://www.geeksforgeeks.org/find-k-frequent-words-data-set-python/.

https://www.geeksforgeeks.org/find-k-frequent-words-data-set-python/

[3] MongoDB Manual. (2021). *Create Indexes to Support Your Queries — MongoDB Manual*. MongoDB. https://www.mongodb.com/docs/manual/tutorial/create-indexes-to-support-queries/

https://www.mongodb.com/docs/manual/tutorial/create-indexes-to-support-queries/