Springboard School of Data Science

Book Recommendations with Singular Value Decomposition

An Exploration of Personalized Recommendations in a Cold Start Scenario

Logan Schmitt

November 29, 2023

I. Background

The intent of this project was to build a model accepting a list of "liked" books as an input and returning a list of book recommendations as an output. The strategy I aimed to use was user-based collaborative filtering, where a model finds similar users (based on similar ratings of the same items) and recommends other items that those users have rated highly. There are a couple challenges one must address when attempting to create this type of model, as demonstrated by the Netflix Prize competition. The first challenge revolves around data: acquisition, storage, and access all proved to be roadblocks. The next challenge revolves around the recommendation libraries and the cold-start problem. Recommendations are really tough to personalize for users with little of their own data.

II. Problem Identification

The problem for this project is easily generalized to any product category, but I focused on books due to (perceived) data availability. The question was: Knowing just a few of a user's favorite books, how can I quickly provide them with personalized recommendations that they're likely to enjoy?

III. Data Collection

I initially approached this problem as a data collection problem. Understanding that a typical recommender system relies on a sparse matrix of user-item interactions, I sought to build a sizable database of book reviews covering a good number of users and books. This was my first misstep.

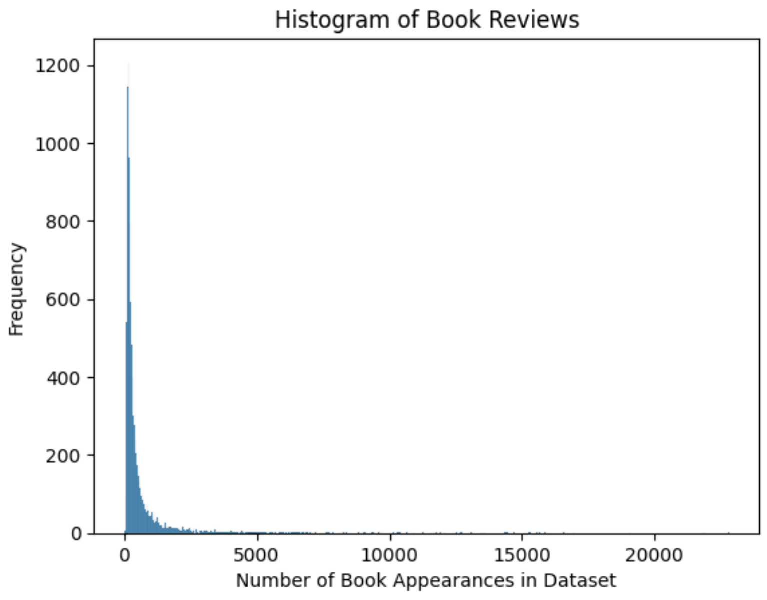
I found the 300 most popular users on Goodreads and collected ratings for the top 100 most popular books in their libraries. This approach ensured that almost every row featured some degree of similarity, as the same popular fantasy series and modern classics appeared throughout the dataset. As a result, the model consistently returned the same recommendations for all inputs.

My next approach was to gather the 100 highest-rated books from each of the 300 users. This created a more sparse matrix, as many of the books with the highest average rating only had a couple reviews and thus only appeared once or twice in the dataset. Unfortunately, most of the same popular books from the first set dominated the second set as well.

I tried combining the datasets, which increased the number of books to almost 30,000. Ultimately, the issue remained that only 300 users is too few for this type of problem. I found the Goodbooks dataset online, which was formatted identically to mine, but consisted of nearly 6 million records and over 50,000 users. Fascinatingly, this new dataset only accounted for 10,000 unique books. This meant that I was far more likely to find similar users as long as they had read a few of the books within the dataset.

IV. Exploratory Data Analysis

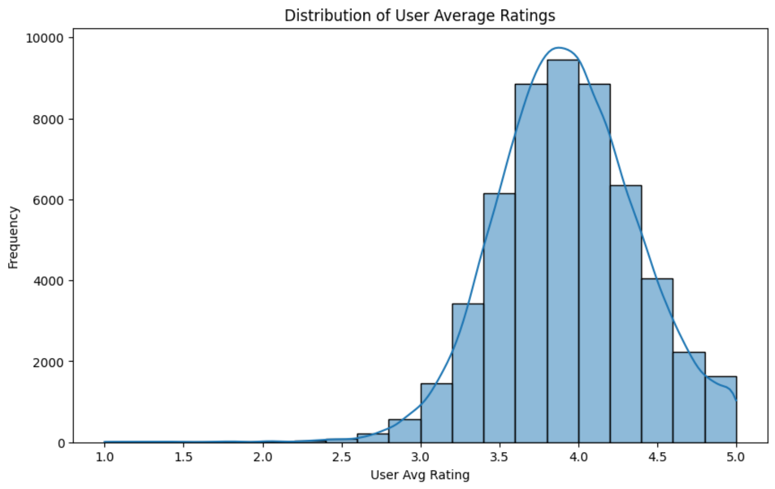
The EDA process was fairly straightforward for these datasets. I found that regardless of dataset source, it was by far most common for a book to appear fewer than 10 times in the dataset. In the case of the Goodbooks dataset, less than 2,000 of the 10,000 books appeared more than 10 times in the dataset. In my original dataset with 25,548 unique books, only 116 of them appeared more than 10 times.



It was quite encouraging that the larger dataset had so many more books with repeat appearances. In my dataset, just a few books dominated the recommendations because they were the only common books. In the new dataset, nearly a fifth of the books had more than 10 appearances, meaning I'd be more likely to generate useful recommendations.

I also took a look at user rating distributions. In both datasets, I found that user ratings skew left and are centered around 4 stars. With a cap at 5 stars, it makes sense that there would be a long left tail toward 0 stars. In my original dataset, I found a small increase in reviews around 0 stars, likely consisting of users who added a book but didn't rate it. This phenomenon wasn't present in the Goodbooks dataset, meaning they likely excluded those reviews.

The review distribution was pretty normal, all things considered. I initially investigated



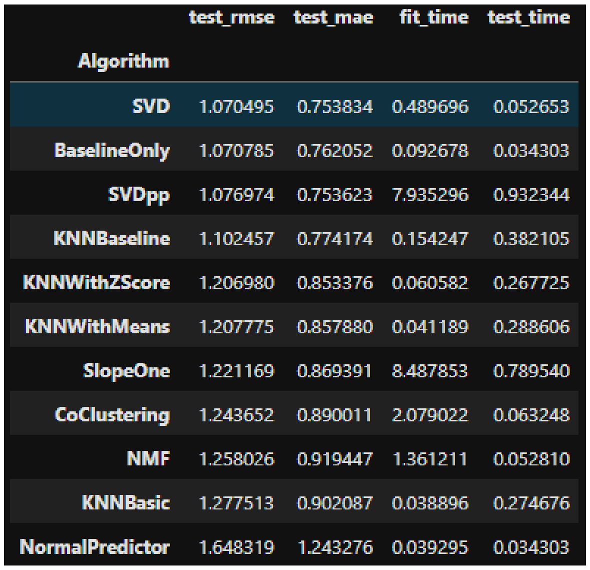
reviewers who averaged a rating below 3, but this type of rater made up a much smaller portion of the Goodbooks dataset compared to my own.

V. Preprocessing

The preprocessing necessary for this modeling was thankfully minimal. Having book ID, user ID, and rating as integers made the process fairly simple. The main question involved a need to normalize ratings per user. This task was handled during the model selection phase, as I will discuss later. For pre-processing, I simply created a SQLite database with a ratings table and a books table, so I could look books up by their ID and return title, author, cover image, and URL information.

VI. Model Selection

The model selection process was a fairly straightforward algorithm benchmarking process. The scikit-surprise library comes with several algorithms with which to fill the sparse matrix with predictions. I simply wrote a function to iterate over each algorithm on a training subset of data and calculate the RMSE and MAE on the test set. I expected some of the many K Nearest Neighbors (KNN) algorithms to perform well and was particularly curious about the KNN variant that used Z-score normalizations for each reviewer. This algorithm performed well among KNN variants but was outperformed by both SVD variants and the BaselineOnly algorithm.



	test_rmse	test_mae	fit_time	test_time
Algorithm				
SVD	1.070495	0.753834	0.489696	0.052653
BaselineOnly	1.070785	0.762052	0.092678	0.034303
SVDpp	1.076974	0.753623	7.935296	0.932344
KNNBaseline	1.102457	0.774174	0.154247	0.382105
KNNWithZScore	1.206980	0.853376	0.060582	0.267725
KNNWithMeans	1.207775	0.857880	0.041189	0.288606
SlopeOne	1.221169	0.869391	8.487853	0.789540
CoClustering	1.243652	0.890011	2.079022	0.063248
NMF	1.258026	0.919447	1.361211	0.052810
KNNBasic	1.277513	0.902087	0.038896	0.274676
NormalPredictor	1.648319	1.243276	0.039295	0.034303

The SVD algorithms are matrix factorization algorithms and therefore don't use a similarity measure like cosine or pearson similarity. They were popularized during the Netflix Prize competition. I performed hyperparameter tuning on a standard SVD algorithm and an SVDpp algorithm. The time difference was staggering (5 minutes versus almost 3 hours), so I proceeded with the standard SVD.

The next challenge was how to use the SVD for predictions. The SVD can only predict on empty cells in the sparse matrix, it doesn't fit a line or plane to the data like other models, so it can't predict on users that aren't in the matrix. Instead, the new user must be added to the matrix before the model is trained. My first attempts to incorporate new users into the Goodbooks dataset resulted in memory errors as the dataset was far too large for the algorithm to perform the necessary operations.

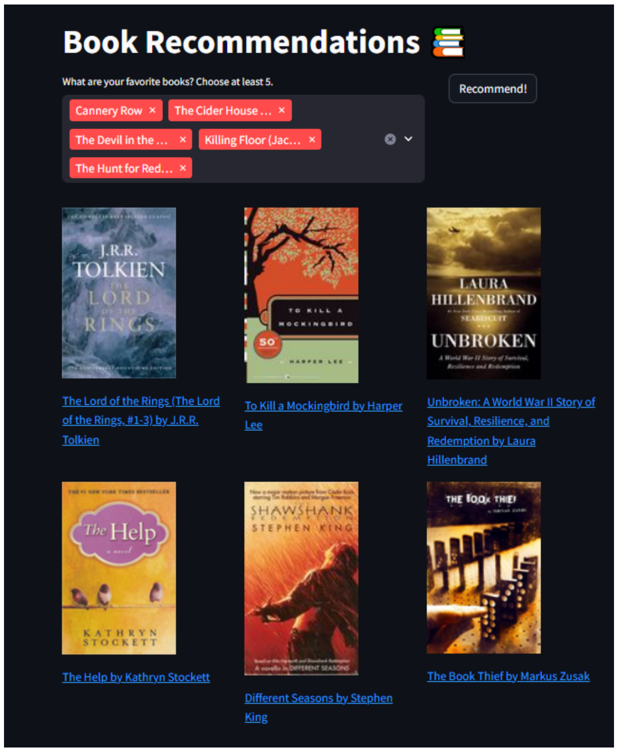
I realized I didn't need to be using users without any commonality with my target user to train the model. As such, I didn't have to keep any records that had no common books with my target user. In fact, I could improve the accuracy of my recommendations by only keeping records from users with at least 3 books in common with my target user. This cut way down on compute time and improved recommendation accuracy tremendously.

VII. Model Deployment

Once the entire model process was working to my satisfaction, I deployed it on Streamlit. This tool allows me to host the model as an interactive website for free on their Community Cloud server. The link is https://loganbookrecommender.streamlit.app/ and the model has successfully generated recommendations for friends and family members. The only downside is that I had to upgrade my GitHub Large File Storage plan because I quickly exceeded the free gigabyte of streaming data with all my test runs of the app.

VIII. Recommendations

The recommender makes tuned recommendations according to the user input in a timely manner. Integrated into an online database like Goodreads or a retailer, this type of model can help provide users with personalized, useful recommendations without years of personal historical data. There are a couple improvements I intend to make going forward. The first improvement is to the user interface. I'm new to Streamlit so I'm trying to learn about how to create an attractive and useful interface. This interface works for my purposes, but it isn't visually balanced and the red book title blocks aren't easy on the eyes.



The Lord of the Rings (The Lord of the Rings, #1-3) by J.R.R. Tolkien

To Kill a Mockingbird by Harper Lee

Unbroken: A World War II Story of Survival, Resilience, and Redemption by Laura Hillenbrand

The Help by Kathryn Stockett

Different Seasons by Stephen King

The Book Thief by Markus Zusak

The next improvement is to add enhanced error handling. Currently, a user could input a list of books such that there are no users with at least 3 of the books in common. They would have to be fairly obscure titles but it is possible nonetheless. I want to update the code so that the recommendation function checks the matrix size and prompts the user to add more books in that case.

The next optimization is to manually handle book series. It is pretty common for the model to include a book from a series if the user has included a book from the same series in their input list. I'd like to manually handle these instances for many popular series. For instance, if a person likes Fellowship of the Ring, they don't need my model to tell them to read Return of the King. They'd be better served by a recommendation from a different series.