March 2022

Book Recommendation

CHARLES FRANZNICK

Charles Franznick
cfranznick@gmail.com
https://www.linkedin.com/in/charles-franznick/

Introduction

Why?

Recommendation systems bring many benefits to companies. They can increase the revenue of a company by creating high conversion rates in customers. Recommendation systems also increase user satisfaction and personalization, building new recommendations based on the previous history of a user. These systems also encourage the user to use the product more. These benefits, in turn, benefit the users as well, exposing them to more products and allowing them to get more out of a service.

Problem

Given a user's previous ratings of books, what are the top 10 books this user will most likely enjoy?

Data Source and Evaluating

The data comes from a Kaggle dataset of Goodread book reviews since the Goodreads API is no longer available.

Scoring a recommendation system is difficult. Knowing whether a user actually likes a new product is not testable in my environment; I use previous user data to determine if a user likes or dislikes a product.

My evaluation metric is Root Mean Square Error (RMSE). This is the standard deviation of the prediction errors. I chose this method since I am trying to find a good fit for all users and am trying to avoid large errors - the goal is to recommend books that the user will at least like; I am trying to avoid recommending books the user will definitely dislike in favor of books the user will probably like.

Data

About the data

The data has book metadata, tags from users, and user reviews for 10,000 books. The source data has 53,424 unique users with a total of 5,976,479 unique ratings. Book metadata include book id, language code, ISBN, publication year, number of ratings, average rating, and how many of each of the star ratings it got, from the minimum 1 star to the maximum 5 stars. Source data also provided the user-generated tags for each book, which is Goodread's user-specific organization method.

Cleaning the data

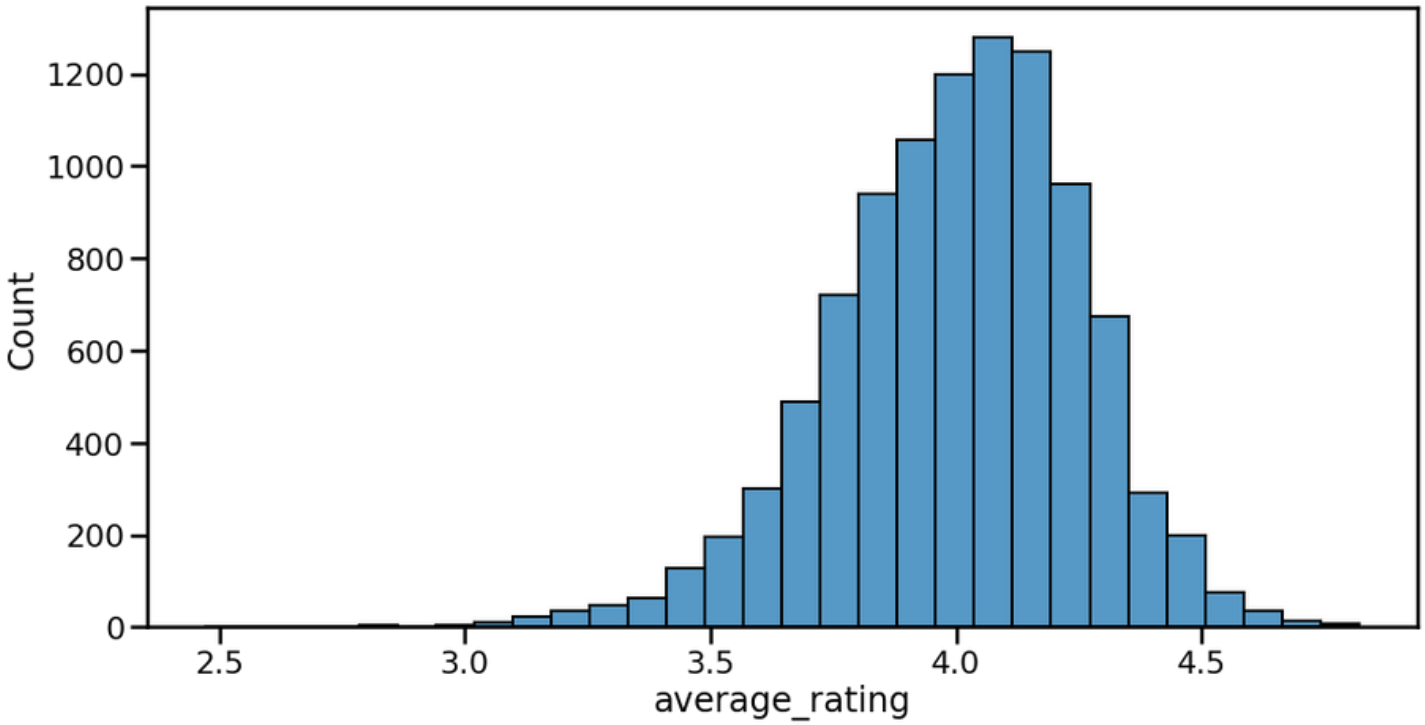
Cleaning the data involved compensating for any null values present, making sure the data types were all appropriate for analysis and ensuring the data was logical and within reasonable bounds. The only missing data were in language codes, original title, and ISBN data - none of which impacted the recommendation system.

Reducing size

In order to reduce the processing power needed while retaining accurate and generalizable results, I chose to reduce the total amount of data. I selected only users who had 100 or more ratings, bringing the total data down to 4,615,637 unique ratings, about 70% of the original data size. I then split the data into 80% training data and 20% testing data.

Exploritory Data Analysis

Rating Distribution



The rating distribution was fairly normally distributed around 4.0 stars. Scores less than 3.5 and above 4.5 are rarer.

Tag popularity

Certain tags by users were more popular than others. I wanted to make sure that we were grouping by genre, as tagged by the users, rather than using tags like 'to-read' which is less helpful when comparing books to determine the success of the recommendation system. To do this, I took only the most popular tags and manually removed the ones that were not genres. This resulted in 60 tags.

count	tag_name
167697	to-read
37174	fantasy
34173	favorites
12986	currently-reading
12716	young-adult
9954	fiction
7169	harry-potter
6221	books-i-own
4974	owned

Data Processing

User Tags

I took the top 50 user tags for all the books and added them to a list of genres I wrote. After removing duplicates, I had 62 distinct genres. I wrote two functions: the first added all tags that a book had into a single column. The second added the top three most common tags for a given book to the book data in the form of three new columns. These features allow me to see the similarity of the top-rated books and top recommended books.

Bayesian Average

The Bayesian average adjusts the average rating of products whose rating counts fall below a threshold. I chose that threshold to be the mean count of ratings; for an item with a fewer than average quantity of ratings, the Bayesian average lowers artificially high ratings by weighing it down (slightly). For an item with a lot of ratings (that is, more than the threshold), the Bayesian average doesn't change its rating average by a significant amount.

Listed here are the top 5 rated books according to the bayesian average and the bottom 5 rated books according to the bayesian average.

mean	bayesian_avg	title
4.525941	4.502533	Harry Potter and the Deathly Hallows (Harry Po...
4.443339	4.422748	Harry Potter and the Half-Blood Prince (Harry ...
4.430780	4.411219	Harry Potter and the Goblet of Fire (Harry Pot...
4.446752	4.346652	Wonder
4.358697	4.341526	Harry Potter and the Order of the Phoenix (Har...

Note the lower variance in the top 5 rated books compared to the larger variancein the bottom 5 rated books.

mean	bayesian_avg	title
3.092439	3.150662	Fifty Shades of Grey (Fifty Shades, #1)
3.214341	3.237825	Twilight (Twilight, #1)
3.371173	3.415356	Wicked: The Life and Times of the Wicked Witch...
3.376984	3.422573	Allegiant (Divergent, #3)
2.706422	3.479920	Four Blondes

Modeling

Overview

The process of modeling the data was as follows:

1. Split the data into training and testing data
2. Explore some basic recommendation systems
3. Test and compare more advanced algorithms
4. Build a top 10 recommended book database for each user
5. Display those books and compare the final 2 algorithms with user ratings

Splitting the data

I chose to split the data into 80% training data and 20% testing data. I used surprise's train_test_split function on the users with over 100 ratings.

Basic recommendation systems

Recommendation Systems can be broken down into 3 major categories: collaborative filtering, content-based filtering, and a hybrid of the two. Collaborative filtering focuses on the user, asking "what do users similar to you like?" Content-based filtering asks, "What items are similar to items you like?" Hybrid takes both of these into questions account.

While I could spend much time building these systems myself, my results would be lacking in comparison to advanced algorithms and would not be significantly faster.

My data does not include demographic data of the users, therefore the best option will be a content-based filtering system or a hybrid system that focuses nearly exclusively on the content side of things.

Modeling

Models

Model	Test RMSE	Fit Time, Test Time
SVD++	0.847	482.36, 18.35
KNN Baseline	0.853	9.61, 54.60
KNN with Z Score	0.857	8.00, 51.53
KNN with Means	0.858	7.49, 49.25
SVD	0.860	19.16, 1.54

Tested on users with 100+ ratings. Lower score is better. Time is in seconds.

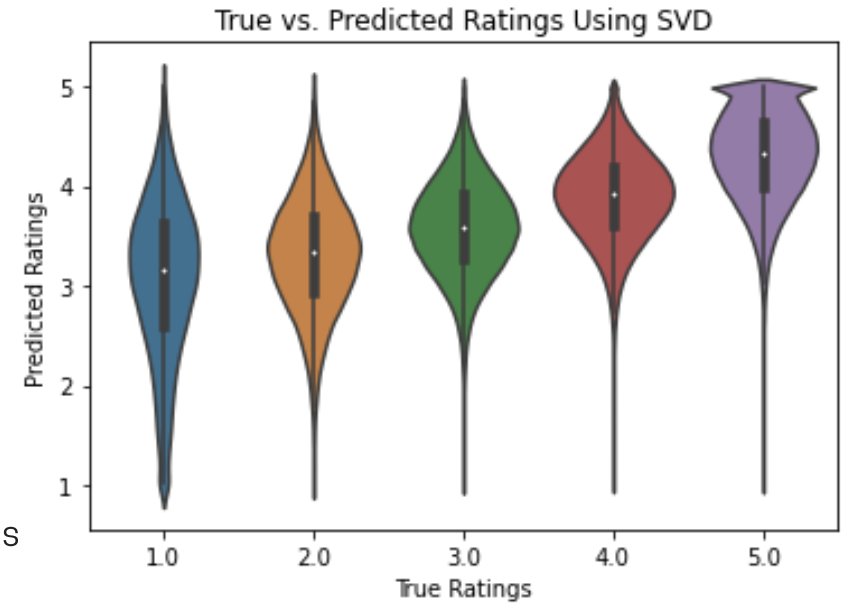
I tested 11 algorithms using the surprise scikit library. I used 3 fold cross-validation on all rating data from users who had 150 or more ratings. Above are the top 5 scoring algorithms. I chose to proceed with the SVD++ and KNN Baseline models since they had the two best RMSE scores and would allow me to test both a form of Matrix Factorization algorithm and a form of k-NN algorithm, which are the only two types of model in the top 5.

SVD++ is an extension of the SVD algorithm which takes into account implicit ratings. KNN Baseline is a basic collaborative filtering algorithm taking into account a baseline rating.

Comparing Models

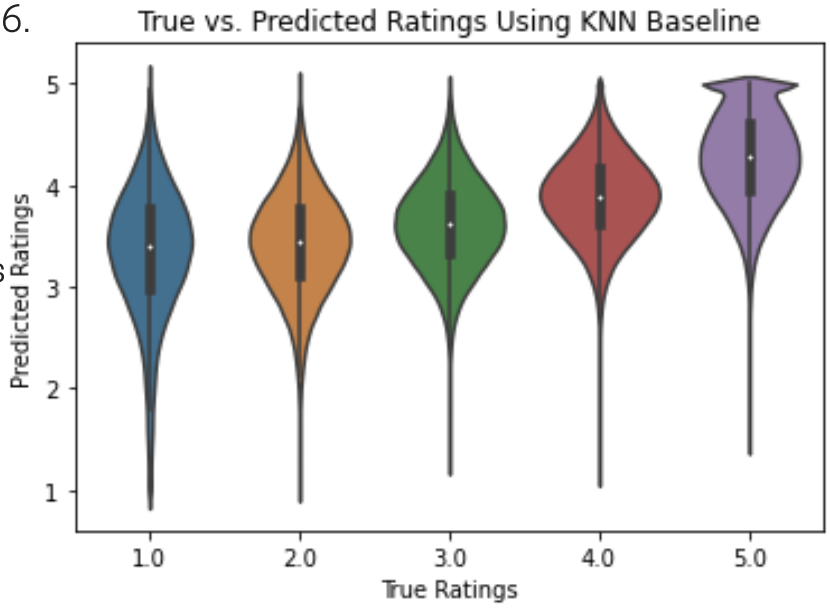
Outputs

The ratings were more accurate as the true ratings were higher. This is fine since we care most about the highest ratings.



SVD++

The mean predicted rating for books with a true value of 5 stars was4.295 in the SVD++ model with a standard deviation of 0.466.



KNN Baseline

In the KNN Baseline model, the mean predicted rating for books with a true value of 5 stars was 4.260 with a standard deviation of 0.456.

Picking a model

In order to choose between these models, I would likely use the model with the faster run time. In this case, the model with the faster run time on my computer was the SVD++ model, coming in around 1 hour. The KNN Baseline model came in at about 1.5 hours. Between the two, the SVD++ model looks like the better choice.

Recommendations Data

Method

I built a database using each of my models using the surprise build_anti_testset function. I saved the top 100 recommendations per user into a CSV file.

To make the predictions, I decided to take the top 10 books with at least 2 tags that matched the top user's top 5 tags. This decision was driven by the top 10 books being skewed by more popular books being over-recommended.

I also split the users by users in the database and users not in the database. For users not in the database, I recommended the best-rated books in order of their bayesian average.

Sample predictions - user not in the database

1. Harry Potter and the Deathly Hallows (Harry Potter, #7)
2. Harry Potter Boxset (Harry Potter, #1-7)
3. Words of Radiance (The Stormlight Archive, #2)
4. Calvin and Hobbes
5. Harry Potter and the Half-Blood Prince (Harry Potter, #6)
6. The Name of the Wind (The Kingkiller Chronicle, #1)
7. Harry Potter and the Goblet of Fire (Harry Potter, #4)
8. The Essential Calvin and Hobbes: A Calvin and Hobbes Treasury
9. Harry Potter and the Prisoner of Azkaban (Harry Potter, #3)
10. A Court of Mist and Fury (A Court of Thorns and Roses, #2)

Final Predictions

Sample predictions - user in the database

For a user in the data, getting the top 10 book recommendations was as simple as getting the top 10 books with at least 2 tags that matched the top user's top 5 tags. This user's top genres are fiction, science-fiction, sci-fi, fantasy, and historical fiction. Here are the top 10 books for the user with the id "1234".

SVD++

1. Words of Radiance (The Stormlight Archive, #2)
2. Lonesome Dove
3. Homegoing
4. The Poisonwood Bible
5. The Complete Maus (Maus, #1-2)
6. The Harry Potter Collection 1-4 (Harry Potter, #1-4)
7. The Way of Kings, Part 1 (The Stormlight Archive #1.1)
8. Roots: The Saga of an American Family
9. Fool's Quest (The Fitz and The Fool, #2)
10. Musashi

KNN Baseline

1. The Way of Kings, Part 1 (The Stormlight Archive #1.1)
2. A Court of Mist and Fury (A Court of Thorns and Roses, #2)
3. Words of Radiance (The Stormlight Archive, #2)
4. Harry Potter Boxset (Harry Potter, #1-7)
5. A Song of Ice and Fire (A Song of Ice and Fire, #1-4)
6. Mark of the Lion Trilogy
7. Collected Fictions
8. The Way of Kings (The Stormlight Archive, #1)
9. Harry Potter Collection (Harry Potter, #1-6)
10. BookRags Summary: A Storm of Swords

Takeaways

Choosing a model

Both of the models I built worked well and gave an average of about 25 book recommendations, with a standard deviation of around 13. If I had to choose one model, I would choose the KNNBaseline model since it is faster for production and the recommendations for both models are subjectively very good.

Improvements

- I could improve both models by modifying the hyperparameters.
- Using all users instead of only the users with more than 100 ratings might also improve results, though it could also skew the results further toward popular books like the Harry Potter series.
- Due to RAM constraints, I had to train a 65% sample of the original rating dataset. Without resource limitations, I would love to train on the full dataset.