Capstone Project

Machine Learning Engineer Nanodegree (Udacity)

Merlin Rebrović
2018-03-16

I. Definition

Project Overview

The goal of the project is to build a book recommender system from a subset of Goodreads' book data. The system takes reader's book ratings and finds other books the reader did not read yet but would score highly.

Problem Statement

According to Google Books Search[1], there were about 130 million books in the world in 2010, and the number just keeps rising. Avid readers with higher education and higher income will probably read only about 1000 books in their lifetime (15 books per year[2] for 70 years), so having some guidance what to read next is necessary.

Word of mouth and top bestseller lists have been frequent sources for book recommendations in the past. However, those approaches usually don't take reader's preferences and leave a chance of keeping the reader in the bubble of like-minded readers.

With the rise of the Internet and World Wide Web, people have started to share online what they read and how they like it. A book recommender system might take people's book reviews and ratings, analyze them, and find similar books or readers. When a person uses the recommender system by supplying his or her preferences and reading history, the system finds books that the reader did not read yet. If the system works well, the reader will score those books highly.

[1] http://booksearch.blogspot.com/2010/08/books-of-world-stand-up-and-be-counted.html
[2] http://www.pewresearch.org/fact-tank/2015/10/19/slightly-fewer-americans-are-reading-print-books-new-survey-finds/

Metrics

Two metrics are used in the project:

1) **Root Mean Square Error (RMSE)** is a common metric in recommender systems. RMSE shows the difference between the predicted and the true rating of a book. The smaller RMSE is, the closer is the predicted to the true rating. This is a simple metric that's easy to calculate, so it is often used to optimize an algorithm.

2) **Rank through a top-K recommender evaluation.** The recommender system's goal is not to predict the rating of an item, but to recommend a small set of items that a user would like from the vast majority that are available. Showing this small set is a ranking problem, and the ranking evaluation used here is borrowed from Koren, Y. 2010[3].

II. Analysis

Data Exploration and Visualization

"Goodreads is the world's largest site for readers and book recommendations."[4] As such, it is one of the best places to find datasets for building book recommender systems. Zygmunt Zając prepared **goodbooks-10k**[5]: a dataset that contains 6 million ratings for 10,000 most popular books on Goodreads from more than 50,000 users.

The dataset contains 5 CSV files (descriptions by the author):

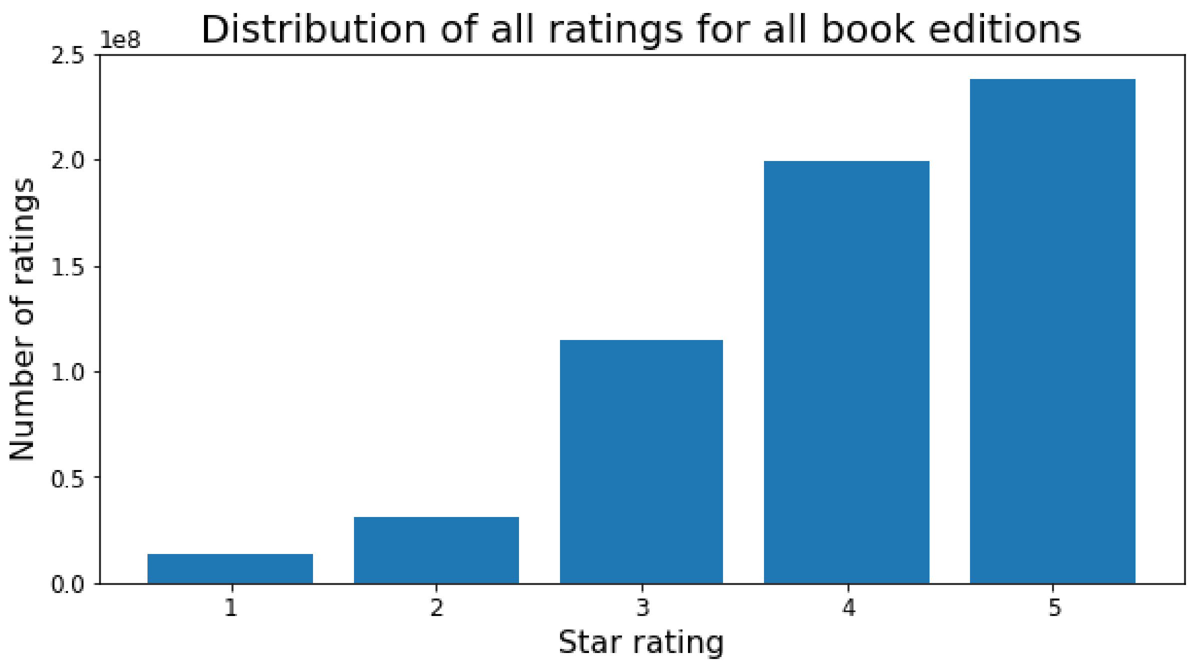
1. **ratings.csv**: contains ratings sorted by time
2. **to_read.csv**: provides IDs of the books marked "to read" by each user, as user_id-book_id pairs, sorted by time
3. **books.csv**: has metadata for each book
4. **book_tags.csv**: contains tags/shelves/genres assigned to books by users
5. **tags.csv**: translates tag IDs to names

Data: books (books.csv)

There are 10,000 most popular books in the dataset where popularity is measured by how many ratings a book has received. The book data contain book identifiers, titles, authors, ISBN, publication dates, links to images. Additionally, each book has a count of ratings and

[3] Koren, Y. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. ACM Trans. Knowl. Discov. Data. 4, 1, Article 1 (January 2010), 24 pages. DOI = 10.1145/1644873.1644874 http://doi.acm.org/10.1145/1644873.1644874
[4] https://www.goodreads.com/about/us
[5] http://fastml.com/goodbooks-10k-a-new-dataset-for-book-recommendations/

reviews it received, and a count of all ratings all editions of the book received broken down to ratings from 1-star to 5-star.



Numbers of ratings are increasing with the positive rating itself. There might be a few reasons for that:

- The dataset contains the most popular books on Goodreads. The most popular books are, by definition, what people love to read, so they score it highly.
- Many people are drawn towards books they might like, either through word of mouth or by reading various editorial recommendations. Those books are more likely to receive a positive rating then if a person had chosen randomly.
- It might be that people are more eager to leave a good rating than bother with a bad one.

Data: book ratings (ratings.csv)

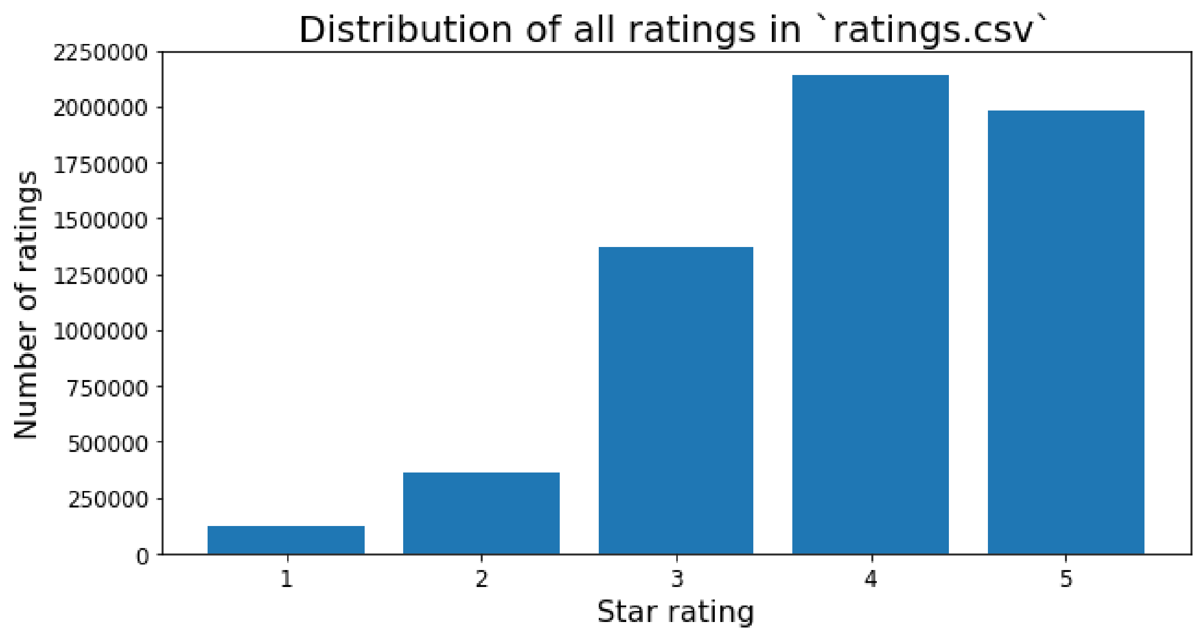
There are 5,976,479 ratings in the dataset. Each rating consists of user ID, book ID, and a rating score from 1 to 5. Numbers of ratings per book range from 8 to 22,806, and ratings by a user range from 19 to 200.

An important finding is that book ratings in `ratings.csv` are not the same as summary book ratings in `books.csv`; they're just a subset. For example:

- The book ID 54 has 1,006,479 ratings in `books.csv`, but only 9,960 ratings in `ratings.csv`.
- The book ID 883 has 111,284 ratings in `books.csv`, but only 1,755 ratings in `ratings.csv`.
- The book ID 7803 has 9,964 ratings in `books.csv`, but only 8 ratings in `ratings.csv`.

We have to use `ratings.csv` dataset because it contains user information. However, we have to keep in mind that the dataset might be lacking because it has orders of magnitude less data than what exists in the Goodreads database.

We can visually compare all ratings and those available to us in `ratings.csv`. We notice the distribution of ratings is different: there are more 4-star than 5-star ratings in our smaller dataset as compared to the distribution chart presented earlier.



Data: books to read (to_read.csv)

There are 912,705 book-user pairs in this dataset. Books to read could be a strong additional signal to ratings because 99.86% of all books are in someone's to read list and 91.48% of all users have books in their “to read” list.

Data: tags (tags.csv and book_tags.csv)

There are 34,252 unique user-assigned tags in this dataset. Even though there are tens of thousands tags, many are similar. For example, 681 tags have “to read” in their names of which a few randomly selected ones are “want-to-read,” “to-read-maybe-someday,” “to-read-memoir,” and “to-read-cookbook.” In addition to that, tags are also in different languages. Tags with the same meaning but different spelling should be considered the same. Some examples of German, French, and Arabic are “noch-zu-lesen,” “101-à-lire-dans-sa-vie,” and “أجمل-ما-قرأت” respectively.

The 10 most popular tags are (starting from the most popular): to-read, favorites, owned, books-i-own, currently-reading, library, owned-books, fiction, to-buy, kindle.

All 10,000 books have tags attached to it. This dataset contains only 100 tags for each book. It is not clear nor indicated anywhere in the dataset description how those 100 were selected: random sampling, most common, or something else.

Algorithms and Techniques

Two commonly-used recommendation approaches are content-based filtering and collaborative filtering.

Content-based filtering

The approach is based on finding a match between features of an item and the user's preferences. If a reader expresses interest in a specific author, genre, period, or story type, the system can find books that match that criteria.

Pros:

- No need for data of other users.
- Easy to explain why an item was recommended.
- Can recommend new and unpopular items.

Cons:

- Feature selection is hard.
- Never recommends items outside user's preferences.

The goodreads-10k dataset does not provide enough book metadata for us to extract necessary features. The only part of the subset of the dataset that could be considered are tags, but they are user-generated content and are not structured.

The content-based approach was not used for this project.

Collaborative filtering

The approach uses past user behavior for predicting future preferences. For our project, past book ratings could indicate future ratings. In collaborative filtering, two categories of methods are widely used: the neighborhood-based methods and the model-based method.

The neighborhood-based methods are about finding similar items or users. If a user positively reviews books A, B, C, and D, and another user positively reviews books A, B, and C, it is more likely that the second user will score book D highly too.

Pros of the item-to-item neighborhood method:

- Less complexity and computation when there are less items than users.
- The model does not need to be retrained when a new user arrives.

- Items are simpler than humans so item similarity is easier to find.
- If a person loves a book of a particular genre, like thrillers, the recommender system will probably find something the person will be happy with.

Cons of the item-to-item neighborhood method:

- Recommendations can feel boring because they all look very similar.

Pros of the user-to-user neighborhood method:

- Recommendations can be from very different genres and book types which adds to the surprise factor. Facilitates exploration.

Cons of the user-to-user neighborhood method:

- Harder to compute since there are many more users than items.
- Users have a lot of different preferences and tastes. Since users usually don't score all items, there many gaps to cross, and it might be hard to connect users.

One of the model-based methods we explored in this project is modeling latent features. Instead of building direct user or item connections and neighborhoods, which is especially hard in sparse datasets, this method tries to reduce big datasets to latent (implicit) features. In books and reading, latent features could be writing style, genres, preference for certain types of protagonists, and similar.

Pros of modelling latent features:

- A more global approach that might recommend new and unexplored books, but still stays within preferences.

Cons of modelling latent features:

- Requires a richer dataset that can uncover latent features.
- Depending on the technique used, it might be costly to recalculate with each new review since the model needs to be updated globally.
- Hard to explain results.

Benchmark

We compared collaborative filtering approaches for building recommender systems to a few models.

1. **Random model.** The model predicts a random rating.
2. **Baseline model.** The model predicts the mean rating of an item adjusted for the bias of a user. This is basically recommending best sellers because those usually have top ratings.
3. **Existing models.** Entities like Amazon and YouTube do not publicize their metrics and results, so it is hard to get to those numbers today. However, there are results of

the best models in the Netflix prize competition that can be used to determine if our model is in a similar range.

III. Methodology

Data Preprocessing

No data preprocessing was needed. The library we used for collaborative filtering uses raw data in the particular format in which the dataset was already available.

The only correction in the dataset was that there were six user-assigned tags with a negative count. We removed them because it is not indicated in the dataset what the negative count means or how the labels were counted in the first place. This action had no impact on the recommender system because we never used book tags as input.

Implementation

To facilitate and speed up implementation, we used Surprise[6] library—a Python scikit for building and analyzing recommender systems. Surprise supports a few well-known algorithms with many options to tweak and alleviates the pain of dataset handling. It also allows evaluation, analysis, and comparison of algorithm performance, and makes it easy to implement new algorithms.

Metrics

We used two metrics mentioned at the beginning. Surprise supports RMSE. However, we had to implement the top-K rank metric.

To evaluate the rank of a book, we take a rating r_{ij} with five stars from a test set. The rating contains: a user *u*, a rating score (1-5), and a book *i*. 100 additional books[7] are selected randomly, and the recommender system predicts ratings by *u* for *i* and the 100 books. All predictions are then sorted from the highest to the lowest. Some of the randomly selected books may be of interest to user *u*, but most of them are probably not. As a consequence, the expected results is that *i*, for which *u* gave the rating of five, will be positioned before all other books. That position is the rank of the book. If *i* is the first and has no other books in front of it, the rank is 0; if it is the last, the rank is 100.

[6] http://surpriselib.com/

[7] We can randomly select any number of books because we'll always normalize the rank to be from 0% to 100%.

Algorithms

We built the recommender system with six different algorithms from three groups: basic algorithms used as benchmarks[8], k-NN inspired algorithms[9], and matrix factorization-based algorithms[10].

1) Random estimate

From the Surprise documentation:

> *The algorithm is predicting a random rating based on the distribution of the training set, which is assumed to be normal. The prediction $\hat{r}_{ui}$ is generated from a normal distribution $N(\hat{\mu},\hat{\sigma}^2)$ where $\hat{\mu}$ and $\hat{\sigma}$ are estimated from the training data using Maximum Likelihood Estimation.*

2) Baseline estimate

Ratings tend to be affected by some users who always give higher or lower ratings than others and by some items receiving higher or lower ratings than others. These effects can be adjusted to baseline estimates.

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i$$

In this method, we estimated the rating of a book $\hat{r}_{ui}$ as its baseline rating b_{ui} which consists of mean ratings of all books μ, the offset factor b_u of a user u, and the offset factor b_i of item i.

3) Item-based k-NN

Item-based (also known as item-to-item) collaborative filtering tries to find similar items by looking at how they were rated by the same users. In the example on the next page, users 1, 2, and 4 gave exact or very similar ratings to items 1 and 2 (marked in blue cells). From that, we infer that those two items are similar.

[8] http://surprise.readthedocs.io/en/stable/basic_algorithms.html
[9] http://surprise.readthedocs.io/en/stable/knn_inspired.html *"These are algorithms that are directly derived from a basic nearest neighbors approach."*
[10] http://surprise.readthedocs.io/en/stable/matrix_factorization.html

	Item 1	Item 2	Item 3	Item 4
User 1	**3**	2	**3**	2
User 2	**5**	-	**4**	-
User 3	1	3		-
User 4	**5**	-	**5**	3

To predict the rating of a book a user has not yet read, the recommender system takes the mean of ratings of similar books but weighed by how similar they are.

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i;u)} sim(i,j) \cdot r_{uj}}{\sum_{j \in N(i;u)} sim(i,j)}$$

where *sim(i, j)* is Pearson[11] similarity between items *i* and *j*, r_{uj} is rating of item *i* by user *u*, and *N(i;u)* is a set of items rated by user *u* similar to item *i*.

4) Baseline item-based k-NN

Same as regular item-based collaborative filtering but with baseline adjustments for users and items.

5) Baseline user-based k-NN

User-based (also known as user-to-user) collaborative filtering tries to find similar users by looking at how they were rated the same items. In the example below, users 1 and 3 gave exact or very similar ratings to items 1, 2, and 4 (marked in blue cells). From that, we infer that those two users are similar.

	Item 1	Item 2	Item 3	Item 4
User 1	3	-	1	-
User 2	**5**	**2**	3	**4**
User 3	**5**	**3**	-	**4**
User 4	-	5	-	3

[11] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

To predict the rating of a book a user has not yet read, the recommender system takes the mean of ratings of that book by similar users but weighed by how similar they are.

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u;i)} sim(u, v) \cdot r_{vi}}{\sum_{v \in N(u;i)} sim(u, v)}$$

where *sim(i, j)* is Pearson similarity between users *u* and *v*, r_{vi} is rating of item *i* by user *v*, and *N(u;i)* is a set of similar user who rated item *i*.

6) Singular Value Decomposition (SVD)

This version of SVD uses baseline adjustments so the prediction for a book is made by

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

where q_i are item latent factors and p_u are user latent factors.

Refinement

Note: All outcomes of refinements are discussed later in the *Results* section.

Baseline adjustments

The implementation proceeded in the order described in the previous section. By the third step (item-based collaborative filtering) it was clear that baseline adjustments are a significant factor in model's performance. In other words, merely adjusting for bias has produced better results than the item-to-item model, so we decided to use baseline adjustments for all future models.

Reducing inputs for user-based collaborative filtering

The memory on the machine used for this project did not have enough memory to build the entire user-to-user neighborhood. We decided to build a few models with subsets of data, and compare the results. We sorted all 53.424 users by how much ratings they have given from the most (200 ratings) to the least (19 ratings). We created three subsets in this way:

1. Top users: the first 13.000 users (~25%) on the sorted list.
2. Bottom users: the last 13.000 users (~25%) on the sorted list.
3. Sampled users: every fourth user (25%) on the sorted list.

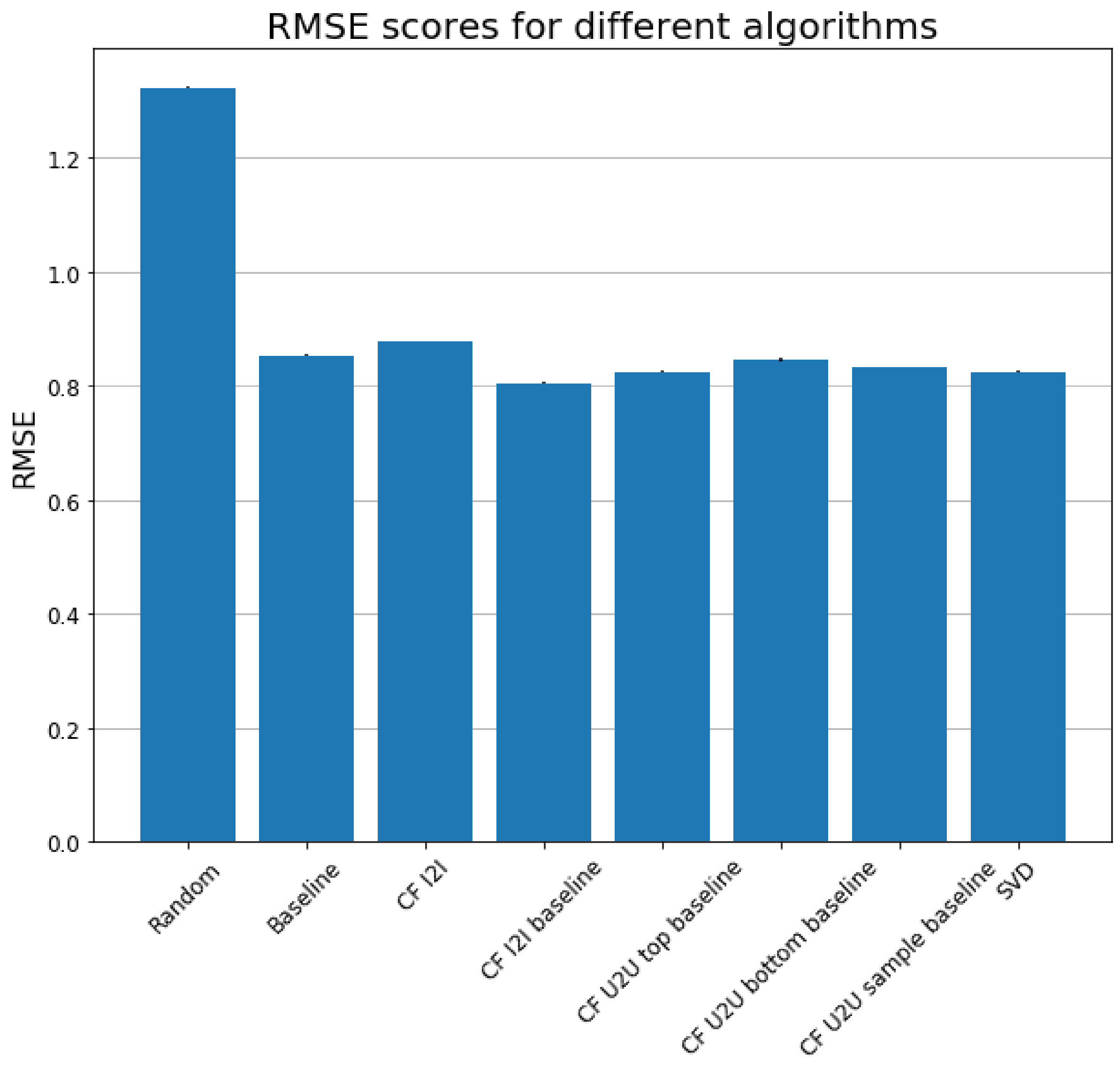
Grid search

For each model that accepts tuning parameters, we ran grid search over a 3-fold cross-validation procedure. We set ranges of parameters around recommended values in the Surprise library, and we optimized for RMSE.

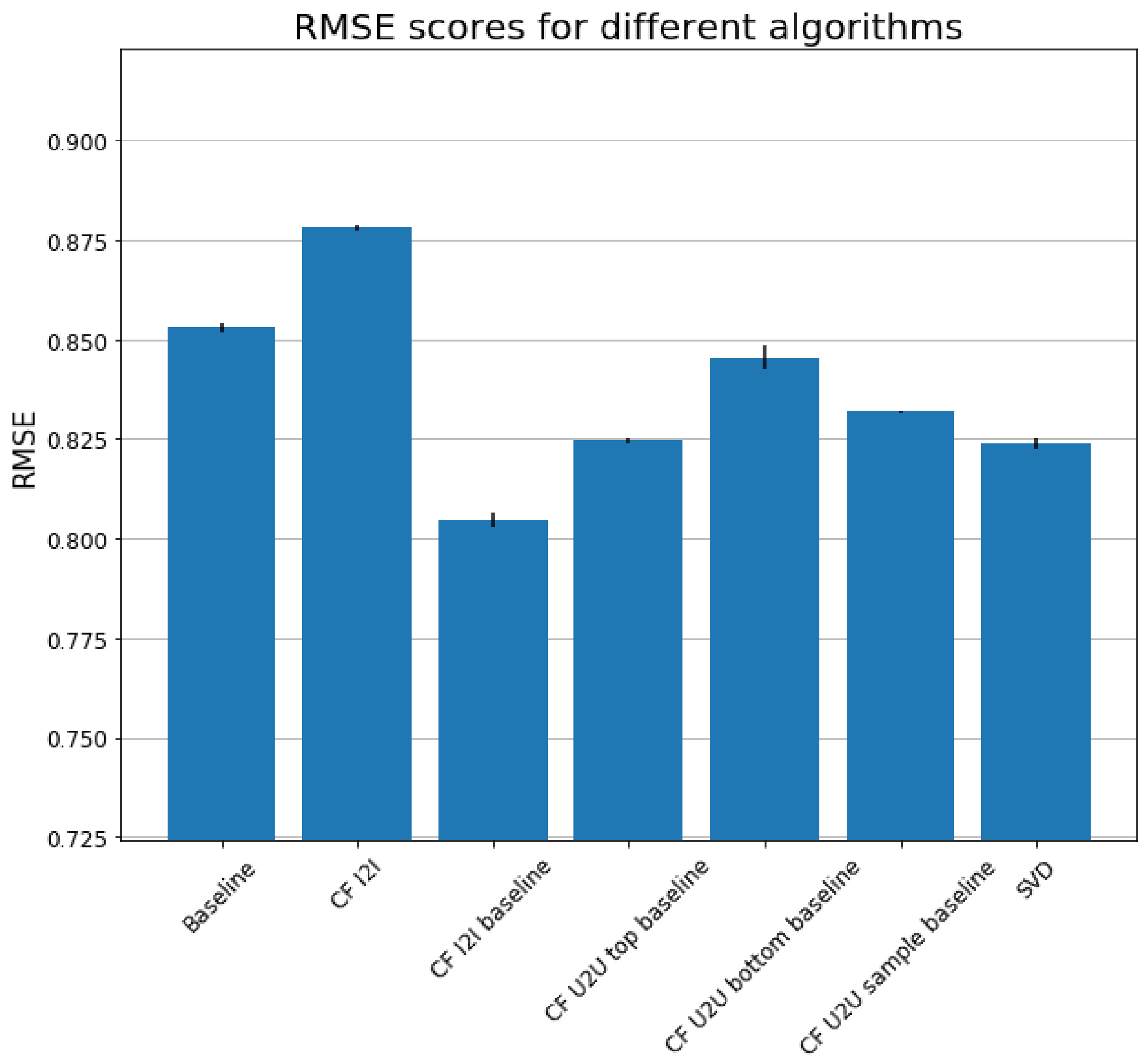
IV. Results

Model Evaluation and Validation

In the figure below, RMSE scores for the best model of each algorithm are presented. It is immediately visible that all algorithms are significantly better than random.



In the next figure, let us remove random estimate to focus more on differences between other algorithms.

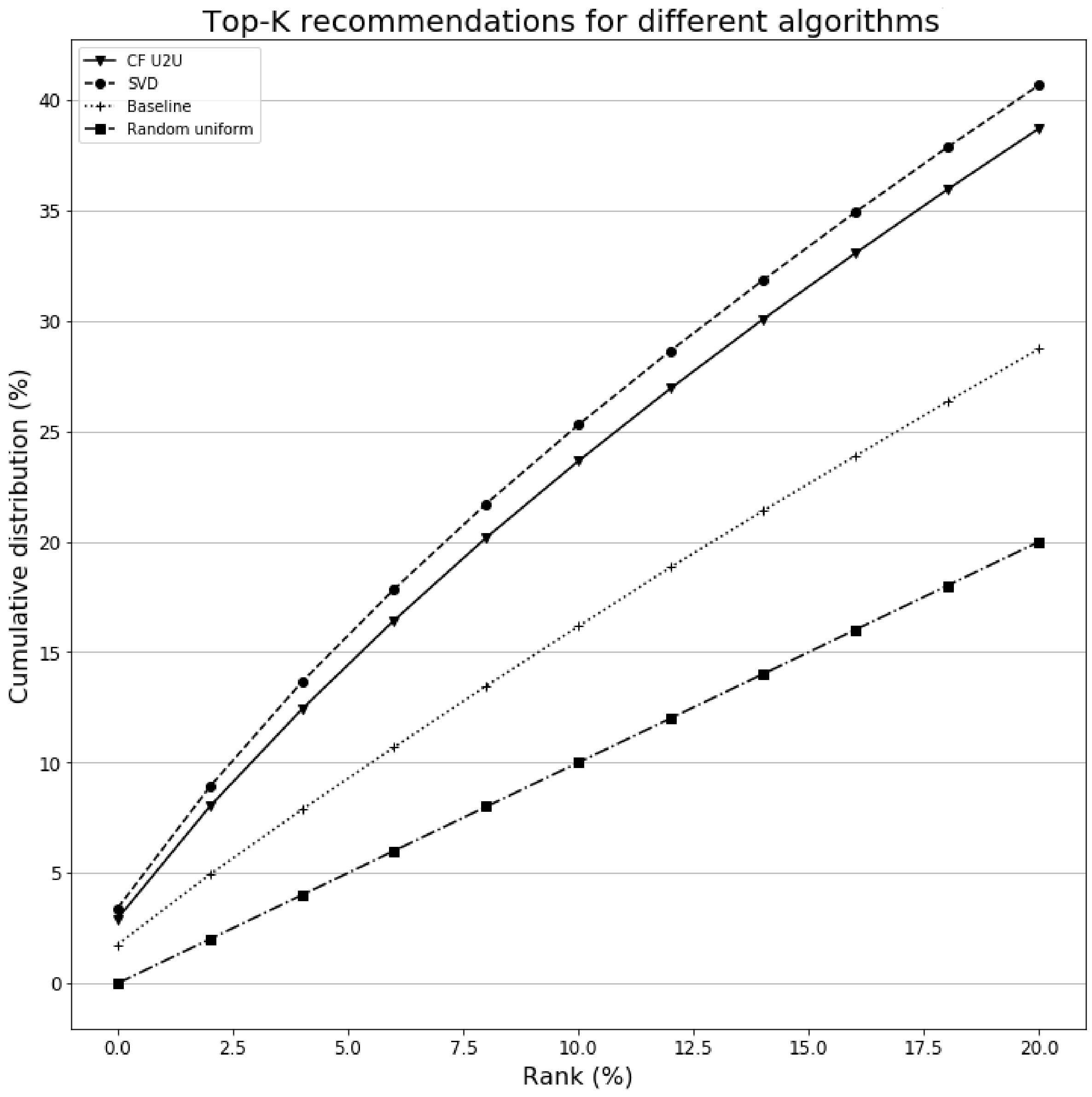


As noted in the Refinement section, baseline estimates are a dominant factor for estimating book ratings. Baseline estimator outperformed item-based (CF I2I) estimator with RMSE of 0.8529 compared to 0.8781. After including baseline adjustment to the item-based algorithm, the resulting item-based baseline (CF I2I baseline) estimator reached the best RMSE of 0.8046.

User-based estimators performed as expected. The estimator that used users with most ratings to build its neighborhood model had the lowest RMSE score out of three; the estimator with users with least ratings had the highest RMSE score.

It is not clear from the results if the full user-based model with all ratings—the one that couldn't fit into memory—would outperform the full item-based. Results from research and industry indicate that item-to-item collaborative filtering works better in practice than user-to-user collaborative filtering. Main reasons are that items are simpler to compare than users, there are often fewer items than users, and that items do not change as often, so there's less to recompute. In our search for model parameters, the best scores for item-to-item was when the maximum number of neighbors was limited to 30, and for user-to-user that limit was 50 (resulting in bigger models).

As part of the model evaluation, we also used top-K rank metric. RMSE does not tell the whole story: since the error is always positive, we do not know if the model is under- or over-rating a book. We have taken the best performing neighborhood model, the SVD model, and random and baseline models as benchmarks, and calculated the rank for all of them.



Two important things are visible in the figure:

1. SVD model slightly but consistently overperforms item-based collaborative filtering even though SVD had higher (worse) RMSE scores.
2. Both SVD and item-based models are better than benchmark models.

Justification

All baseline adjusted collaborative filtering models performed better in all metrics than two internal benchmark models we set at the beginning: random estimator and baseline estimator.

RMSE for collaborative filtering models is in the range from 0.8046 to 0.8453, while random and baseline are 1.3237 and 0.8529 respectively (higher is worse). We cannot easily compare external models to ours regarding RMSE. However, we can use external scores to check if we are in the right range. The Netflix prize started with the benchmark RMSE of 0.9525, and the winning model had RMSE of 0.8567. Datasets are entirely different, but the scores are in the close range. If our models had RMSE > 1, we could conclude they were performing almost randomly. If RMSE was 0.2, this would be a potential sign of overfitting.

All our models outperformed benchmark models regarding ranking. For example, there is 10% chance that a 5-star book finds itself in the top 10% of items for a random model. However, there is 25% of that happening if SVD or item-based models make recommendations. When comparing rank to models described in Koren, Y. 2010, our results are not as impressive. The best models there, built from Netflix datasets, are one order of magnitude better. For example, there is 30% chance for the best model that a 5-star movie finds itself in the top 1% of items.

Our solution is on the right track but not good enough for production environments with many items. Here are some of the reasons for that:

- **The dataset is limited in size.** As noted in the data analysis section, the dataset has orders of magnitude less data than what exists in the Goodreads database. In addition to that, researchers around the Netflix prize realized that time (when a movie was published and when it was rated) is a significant factor in predictions; the time component does not exist in this book dataset.
- **Limited compute power.** More powerful machines could have explored more tuning parameters and built the full neighborhood models.
- **We did not use the whole dataset for input.** For example, there are additional signals in the tags and to-read sections that we did not use.

V. Conclusion

Free-Form Visualization

We used the SVD model to recommend 20 books to a user. We compared that set to the books that the user rated with five stars and sorted by mean book rating. 18 of 20 top books rated by the user are in fantasy, comic, and sci-fi categories (marked light gray in the table).

Books recommended by the model are different. Only 5 of 20 are in fantasy, comic, and sci-fi categories. Instead, 6 of 20 books are fiction (marked blue in the table), and the rest are classics, history, and children or parenting books. This situation outlines two things:

1. **Ranking problem.** Even if a recommender system can estimate the ranking of a book well, the system is not good if it is not able to rank good books highly and show them to a user.
2. **Explainability.** The system picked up that a user likes fiction as a latent feature. However, it is hard to understand or explain why. Sometimes it might be better to go with a marginally worse performing model that is easier to explain or maintain.

Rated 5-stars by user		Recommended by model
The Complete Calvin and Hobbes Bill Watterson		دیوان [Dīvān] Hafez
The Complete Works William Shakespeare		ESV Study Bible Lane T. Dennis, Wayne A. Grudem
The Complete Maus Art Spiegelman		Blood Meridian: Or the Evening Redness… Cormac McCarthy
The Hobbit and The Lord of the Rings J.R.R. Tolkien		Preacher, Volume 9: Alamo Garth Ennis, Steve Dillon
Percy Jackson and the Olympians Boxed Set Rick Riordan		This is Not My Hat Jon Klassen
The Return of the King J.R.R. Tolkien		The Revenge of the Baby-Sat: A Calvin … Bill Watterson
Night Watch Terry Pratchett		The Authoritative Calvin and Hobbes Bill Watterson
The Lord of the Rings J.R.R. Tolkien		Lonesome Dove Larry McMurtry
The Hunger Games Box Set Suzanne Collins		Transmetropolitan, Vol. 5: Lonely City Warren Ellis, Darick Robertson, Rodney ...

Maus: A Survivor's Tale : My Father Bleed... Art Spiegelman		The Book with No Pictures B.J. Novak
Harry Potter and the Philosopher's Stone J.K. Rowling, Mary GrandPré		The Lions of Al-Rassan Guy Gavriel Kay
The Two Towers J.R.R. Tolkien		Far from the Tree: Parents, Children, and ... Andrew Solomon
To Kill a Mockingbird Harper Lee		The Amazing Adventures of Kavalier & Clay Michael Chabon
Going Postal Terry Pratchett		Tiny Beautiful Things: Advice on Love... Cheryl Strayed
The Last Olympian Rick Riordan		The Poetry of Pablo Neruda Pablo Neruda, Ilan Stavans
The Ultimate Hitchhiker's Guide Douglas Adams		Cuentos completos Jorge Luis Borges, Andrew Hurley
The Foundation Trilogy Isaac Asimov		A Supposedly Fun Thing I'll Never Do Again: Es...David Foster Wallace
Watchmen Alan Moore, Dave Gibbons, John Higgins		The Making of the Atomic Bomb Richard Rhodes
The Fellowship of the Ring J.R.R. Tolkien		Lincoln in the Bardo George Saunders
The Stand Stephen King, Bernie Wrightson		The Universe Versus Alex Woods Gavin Extence

Reflection

We have built a book recommender system from a subset of Goodreads' book data. The dataset mainly consists of ratings and tags users have assigned to books. This kind of data lends itself well to collaborative filtering, the method that takes past user behavior and predicts future preferences.

The most surprising part of the project is seeing how big of an impact just adjusting for bias has. The winners of the Netflix prize shared the same sentiment[12]:

> *Out of the numerous new algorithmic contributions, I would like to highlight one – those humble baseline predictors (or biases), which capture main effects in the data. While the literature mostly concentrates on the more sophisticated algorithmic aspects, we have learned that an accurate treatment of main effects is probably at least as significant as coming up with modeling breakthroughs.*

[12] https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

Improvement

The first improvement that could be done is to use the whole dataset that is available; we have not used tags and to-read information in collaborative filtering models. Tags and to-read contain additional signals from a user for their own preferences: genres, important books to read, preferred authors, or style. If we were to build a model taking tags and to-read into account, that model would fall into content-based filtering methods. It would be interesting to see how content-based and collaborative filtering models would compare, and how their combination would compare to the collaborative filtering model built in this project.