Books recommendation project

Team:

- *Nazarii Drushchak*
- *Uliana Zbezhkhovska*
- *Marianna Kovalova*
- *Oleksandr Vashchuk*

Repo Link: https://github.com/naz2001r/recsys_project_MAUN

Introduction

There are a lot of books in the world, and people often need help finding the right one. It can be difficult and time-consuming to decide among millions of books. Some people try to help by giving unbiased ratings, but they sometimes exaggerate and mislead others. This makes book recommendations less accurate. Our project aims to solve this problem by using better recommendation systems. We tried different algorithms, like baseline, collaborative filtering, content-based, matrix factorization, sentence transformers, neural networks, and combinations (hybrid models). We also created a data pipeline for data/models versioning and experimenting, an inference application to try algorithms in action. The goal is to get hands-on experience with the most common algorithms for recommendation systems and try to experiment with data and model architecture.

Exploratory Data Analysis

Our data consist of three datasets - users, books, and ratings. Below we show the most interesting insights from each dataset.

Insights from users' dataset:

The distribution of users' age (Fig.1) shows that the distribution is right-skewed, and the most active users are aged 20 to 40.

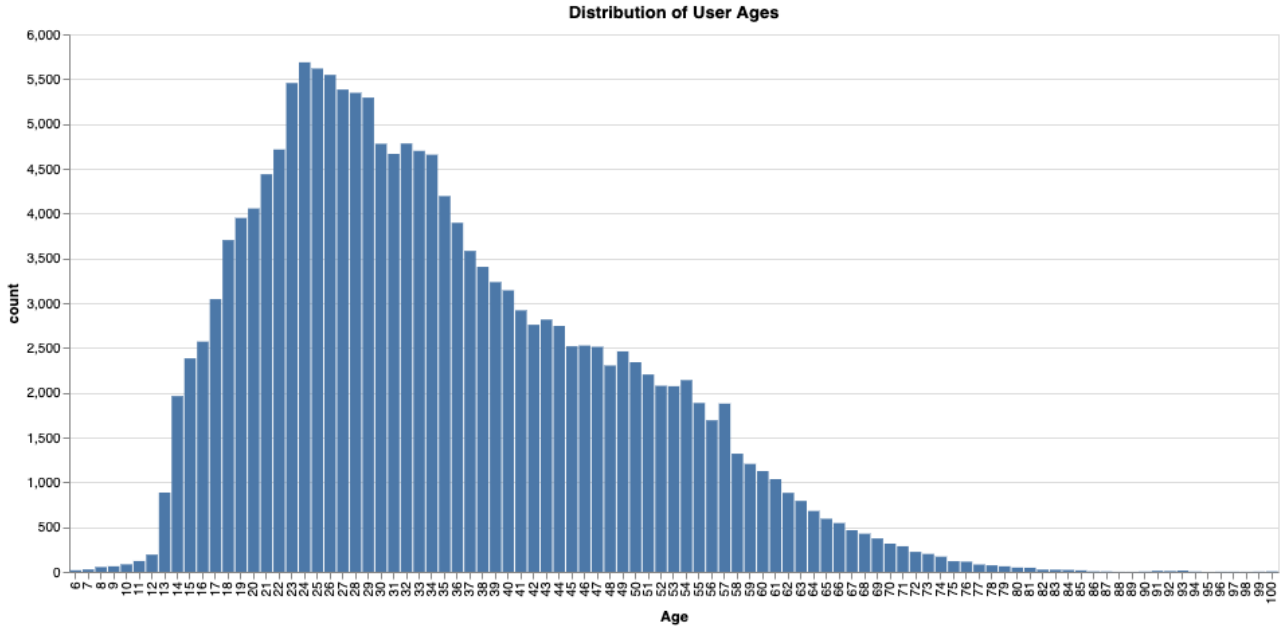

Fig. 1 Distribution of users' age

Also, from Figure 2, it can be seen that most active users are USA-based.

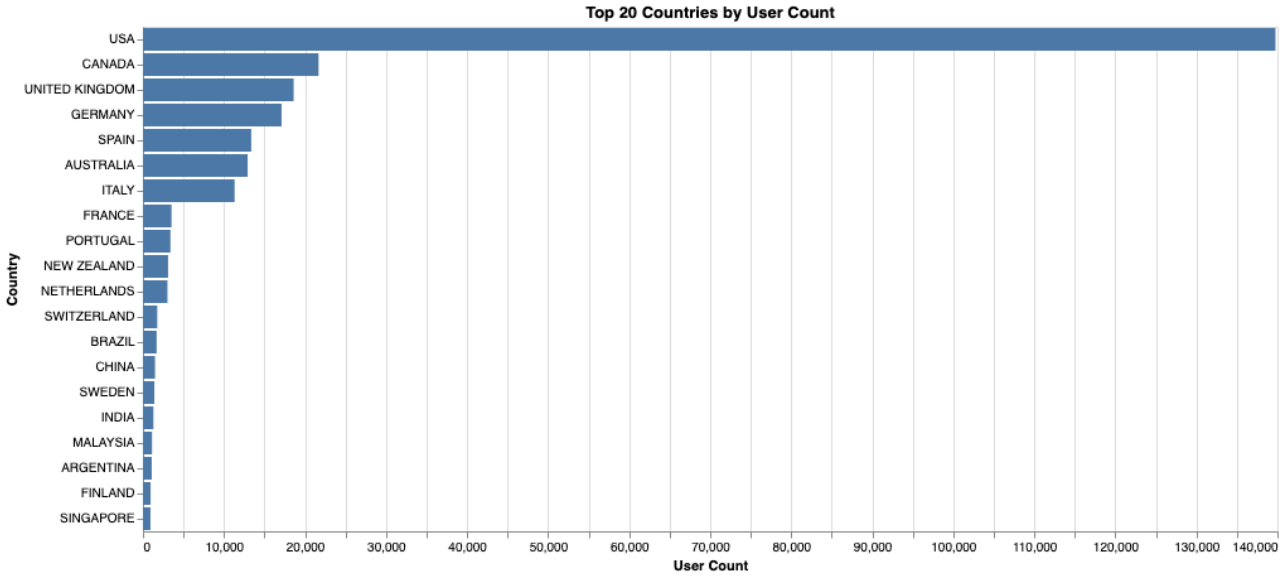

Fig. 2 Users' location

Insights from the rating dataset:

We have many 0 ratings in this dataset, so first, we delete them. Then the book-rating counts are shown in Figure 3. Rating 8 has been rated the highest number of times.

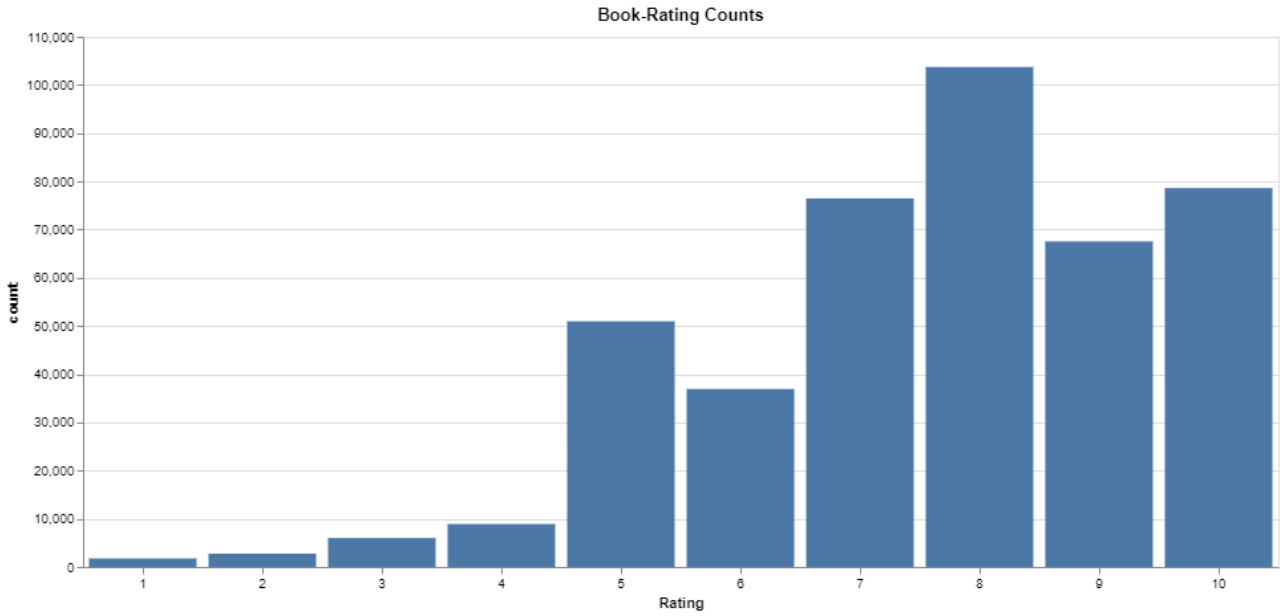

Fig.3 Book-rating counts

Insights from the books dataset:

Figure 4 shows the distribution of the year of the book's publication. The distribution is left-skewed, and a significant number of books were published in the years from 1990 to 2003.

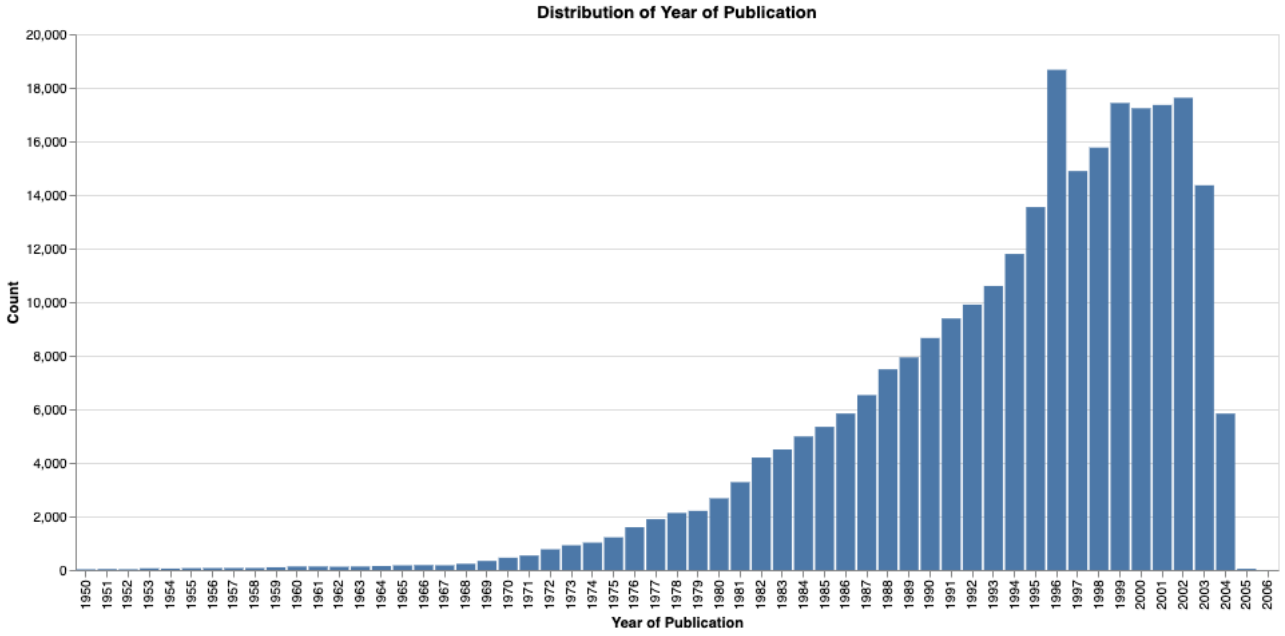

Fig.4 Distribution of the year of the book's publication

From Figure 5, the publisher Harlequin has more published books.

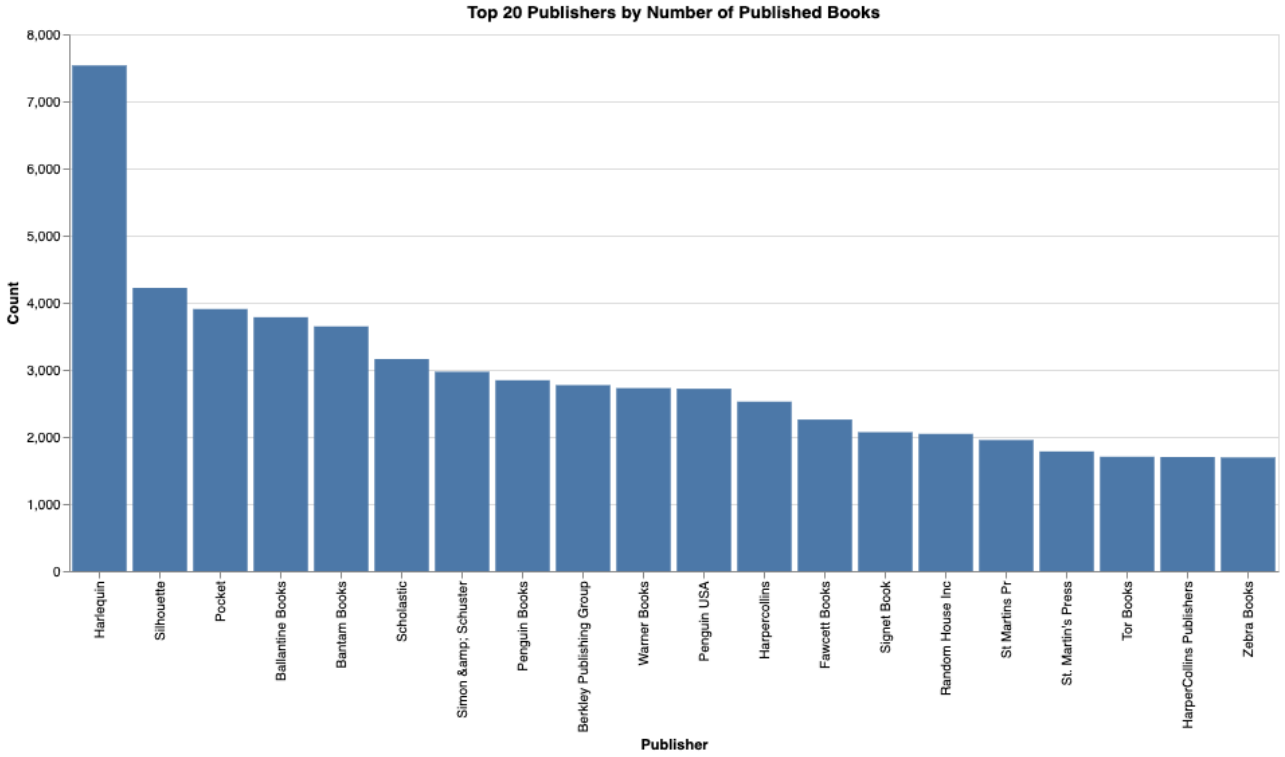

Fig.5 Top 20 publishers by number of published books

Agatha Christie, William Shakespeare, and Steven King are topping the list as the three authors by the number of books (fig. 6).

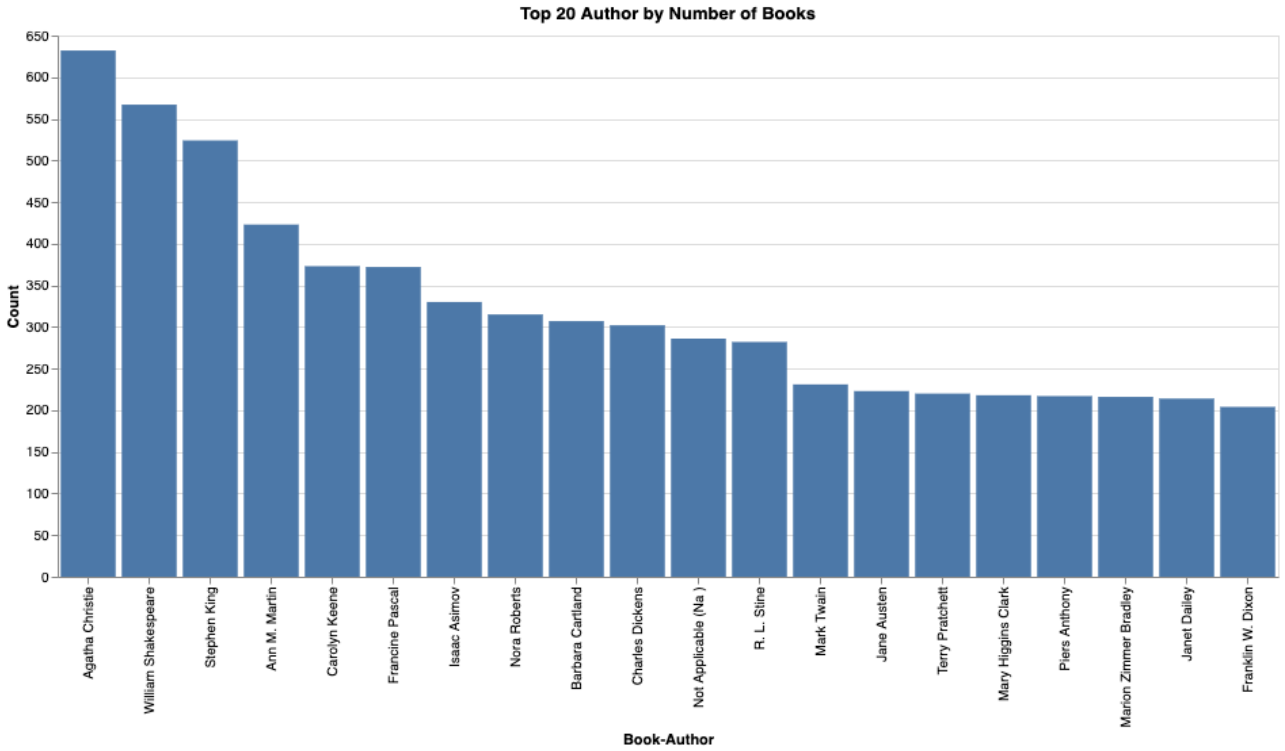

Fig. 6 Top 20 authors by number of published books

Insights from the joined dataset:

But if to speak about authors in terms of most reading (by count their names occurring in the dataset) topping three are Stephen King (still), Nora Roberts and John Grisham (Fig. 7). So, Stephen King not just wrote a lot books but they are also reading a lot.

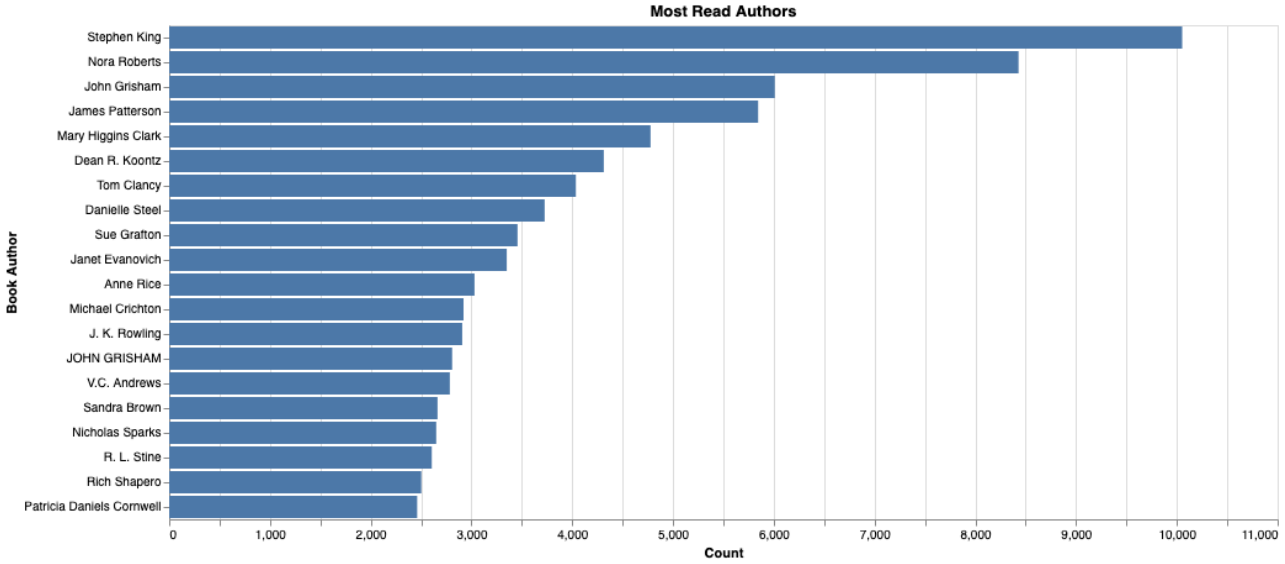

Fig.7 Top 20 most reading authors

If to speak about most read books (by the number of them occurring in the dataset) they are Wild Animus, The Lovely Bones: A Novel, and The Da Vinci Code (Fig. 8).

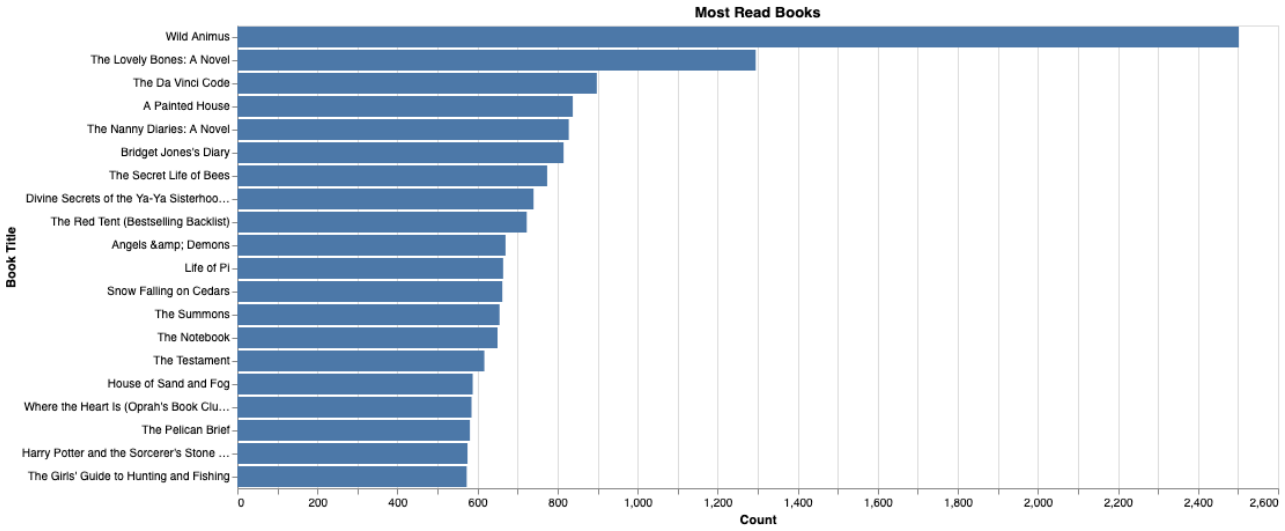

Fig. 8. The most-read books

Development & Experiment Journal

- It was decided to start development from creating the ML pipeline for data storage and experimenting. For this purpose, we selected DVC. With this pipeline, we could share data across team, store metrics, store models, track progress, launch experiments, etc. Working on this project, we constantly improved our pipeline by adding more granularity and flexibility. And now we will not concentrate on it. Please use the DVC extension in VS Code to fully explore its capabilities and use readme file with hints in case of any problems.
  Please pull the dvc pipeline before working with the project.
- Developed base model class with service functions, baseline model, and basic metrics (MAP@K)
- Added collaborative filtering and base version of preprocess step.
- Developed matrix factorization algorithm. New evaluation metrics were researched and then coded with tests coverage by our most experienced data scientist in a team.
- Experimented with stratify approach for the dataset. Books with one review were filtered out, so stratify for books worked.
- Developed content-based sentence transformer algorithm.
- Developed hybrid neural network model + sentence transformer.
- Splitted and improved pipeline steps to give more flexibility
- Developed filter for already read books per each user per each algorithms.
- Early stopping for NN model developed.
- Cuda support for NN model added.
- Title preprocessing
- Poetry setup, for easier project bootstrap
- Computational improvements for NN
- Developed content-based algorithm
- Added books sorting logic for NN
- Created inference pipeline
- Experimented with rating score filtering. Added that to main
- Experimented with bigger sentence transformer

- Experimented with books deduplication, but it didn't improve the models significantly, so it wasn't merged.
- Developed hybrid matrix factorization + sentence transformer
- Dropped MAP@100 metric as it was very computationally expensive
- Experimented with user filtering to filter out users with 1 review

Comparison of models and approaches

Table 1 shows the comparison of scores depending on different algorithms and metrics.

Table 1

	Baseline	Collaborative filering	Matrix factorization	Content based	Content-based sentence transformer	Hybrid NN	Hybrid factorization + sentence transformer
map@5	0.000140	0.000244	0.001573	0.000242	0.000932	0.000267	0.000529
precision@5	0.000158	0.000222	0.000983	0.000158	0.000476	0.000190	0.000285
recall@5	0.000375	0.000391	0.003081	0.000481	0.001870	0.000513	0.000883
ndcg@5	0.000237	0.000346	0.002171	0.000327	0.001212	0.000396	0.000703
map@10	0.000226	0.000344	0.001951	0.000357	0.001090	0.000361	0.000604
precision@10	0.000238	0.000285	0.000983	0.000253	0.000491	0.000269	0.000269
recall@10	0.000975	0.001165	0.005800	0.001314	0.003126	0.001225	0.001518
ndcg@10	0.000491	0.000639	0.003186	0.000652	0.001683	0.000677	0.000938
map@20	0.000305	0.000452	0.002242	0.000423	0.001335	0.000510	0.000749

precision @20	0.000238	0.000396	0.000896	0.000238	0.000547	0.000317	0.000301
recall@20	0.002348	0.002892	0.010332	0.002445	0.006514	0.003613	0.003759
ndcg@20	0.000857	0.001190	0.004453	0.000962	0.002645	0.001313	0.001540
coverage	0.031547	0.031547	0.031547	0.031547	0.031547	0.031547	0.031547

We can conclude that the best algorithm for the book's recommendation system is matrix factorization, which shows higher scores in each metric.

Research Online Methodology

For online evaluation, we plan to use A/B testing. The objectives we set when training the book recommendation model and the offline metrics we measure may not necessarily reflect our desired outcomes. For instance, even if the model is highly accurate in predicting books that users would click on, it does not guarantee that it will effectively reduce user dropout or increase book purchases. Despite the model recommending books that align with a user's preferences, they might still lose interest and not engage further with the system.

Furthermore, training a model specifically to recommend books with the intention of reducing user dropout or increasing purchases is a more intricate task compared to simply recommending books based on click behavior.

Some common online metrics for evaluating the performance of a book recommendation system include:

1. User retention
2. Engagement (likes, bookmarks, follows, etc.)
3. Clicks
4. Purchase
5. Revenue
6. Time-spent
7. Diversity in recommendations.

The definitive evaluation for any model occurs during a live A/B test, where a new variation is compared against the existing book recommendation model. For instance, let's say the hypothesis suggests that incorporating a collaborative filtering approach will improve recommendation accuracy. To test this, we deploy the new model to a randomly selected group of users on the platform. Both the current model and the new variation operate on the same user population, ensuring that any variations in online

metrics can be attributed solely to the change in recommendation approach. By analyzing the net change in metrics, one can determine whether the new variation performs better or worse than the current model.

On our dataset, we haven't information about users' clicks, but if we will use a website, we can collect this information online and based on book rates from the database propose new books to the users online more effectively. In our online book recommendation system, each user will be assigned a recommender based on their ID. Throughout the online evaluation, the system will track the recommended books presented to users, their click behavior on these recommendations, the books they visited, and their rates. The website relies on cookies to track individual users and does not require registration or sign-in for browsing book selections and making book ratings.

The evaluation of the system's performance utilized two metrics: click-through rate (CTR) and visit-after-recommend rate (VRR). CTR is calculated as the ratio of clicked books to the recommended books, serving as an indicator of the relevance and effectiveness of the recommendations in capturing users' attention. CTR has been widely used in web and mobile advertising, as well as online marketing campaigns since the early stages of the internet. It is also a key metric in the field of recommender systems, as it allows researchers to examine how many recommended items users consume. The underlying assumption is that when a user clicks, interacts with, or consumes a recommended item, they perceive the recommendation as useful. From a business perspective, CTR indicates the effectiveness of the recommender system in accurately predicting items that are valuable to the user. The calculation for this metric can be found in Equation 1.

$$CTR = \frac{Number\ of\ clicks\ on\ recommended\ items}{Number\ of\ times\ recommendations\ were\ shown} \quad (1)$$

VRR, on the other hand, is a less stringent criterion that measures situations where a recommended book eventually gets visited by the user. In VRR, users may not have seen the recommendations, they may not have been suitable for their current context, or the presentation might not have been compelling. Nonetheless, the recommended books themselves are likely relevant. This metric is calculated according to equation 2.

$$VRR = \frac{Number\ of\ users\ who\ visited\ a\ recommended\ item}{Number\ of\ users\ who\ received\ recommendations} \quad (2)$$

While VRR is generally considered weaker than CTR, it will be employed for two primary reasons. Firstly, the volume of collected feedback for VRR can be significantly higher. Secondly, due to the placement of recommended books outside the initially visible area, CTR results might underestimate the actual utility of the recommendations. It should be noted that a higher VRR indicates that the recommendations are successful in capturing user interest and motivating them to explore the recommended books. When the higher CTR suggests that the recommendations are relevant and engaging, leading users to click on the recommended books.

By comparing the VRR and CTR metrics over time or between different variations of the book recommendation system, we will be able to assess the system's effectiveness. An increase in both VRR and CTR indicates that the recommendations are successfully driving user engagement and interest in the recommended books. On the other hand, a decrease in these metrics may indicate areas for improvement in the recommendation algorithms or strategies. It's important to monitor these metrics continuously and consider other factors such as user feedback and book purchasing behavior to get a comprehensive understanding of the system's effectiveness.

Also, we plan to do the system like in paper *Do Offline Metrics Predict Online Performance in Recommender Systems?*. Such evaluation comprises two fundamental elements: Environments and Recommenders. An environment is composed of a collection of users and items. The recommender and environment engage in iterative interactions. During each time step, the environment identifies a group of online users who require recommendations for an item. Leveraging the historical data of user-item interactions, the recommender either suggests a single item (top-1 recommendation) or a set of items (slate-based recommendation) to each online user. Subsequently, the environment assigns ratings to some or all the recommended items.

The diagram below (Fig. 9) illustrates the interplay between the environment and the recommender:

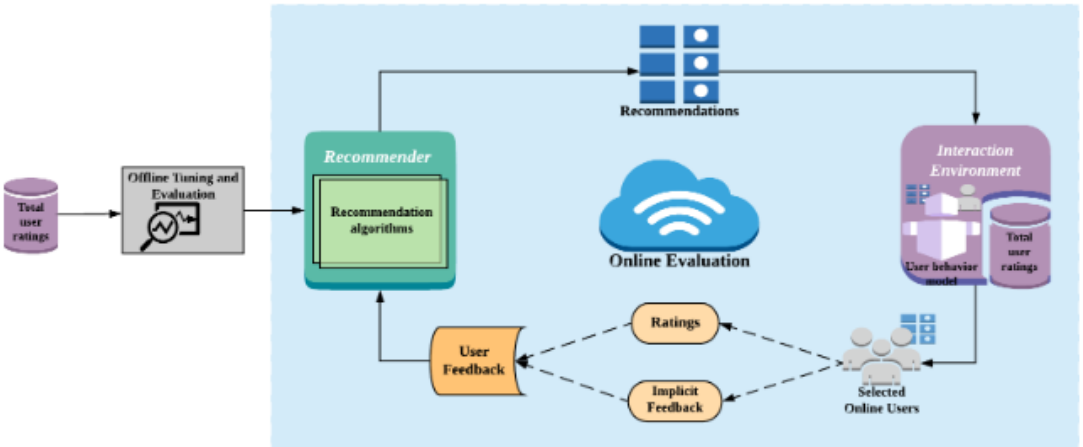

Fig. 9 A visualization of the interaction between the environment and recommender.

This system is available online on the GitHub [https://github.com/berkeley-reclab/RecLab], so we plan to try to make it suitable for our book recommendation system.

Summary & Conclusions

This project helped us to understand how different algorithms work for recommendation systems. It was the first project during a master's program that was so heavily developed to be production ready. We first time used here DVC to manage data and build a pipeline. And in our opinion, it saved us a lot of time and made it easier to exchange data, version it, and experiment with algorithms. At the end of the project, it was satisfying just to run one command to test the hypothesis.

We researched and developed different recommended systems with a hybrid approach that combines our knowledge in other domains. Even though these models are worse than classic matrix factorization, it was very interesting to create something new.

We experimented with different ideas. Not all of them were successful, and those ideas leave in separate branches of the project.

We optimized our NN model to work on small GPU with CUDA and made it 10x faster for prediction. The content-based model was struggling with memory efficiency and was successfully optimized as well.

Main challenges that we faced:

- Learn how to use and leverage DVC functionality
- Lack of computational resources (overall pipeline is running almost ~14 hours)
- Algorithms and metrics research
- Managing all branches with experiments and running them in parallel
- Different time zones in the team

Possible improvements:

All these improvements were not made due to a time constraint. The pipeline grows big, we have a lot of models, and experimenting is time-consuming due to a lack of computational resources.

- Try bigger transformer and optimize hyperparameters in NN
- Try to build more complex NN

- Feature engineering with books description (to get all books' descriptions will take 50 hours in multithreading mode), but this might improve Sentence Transformer's performance.
- Split should be different for content-based and collaborative algorithms, we wanted to try customer cohorts stratify.
- Finetune hybrid matrix-factorization + sentence transformer model
- Build even more complex solution with MF + Collaborative filtering + Sentence Transformer

This course has been an enriching experience, leading to a very complex project with many new things from an engineering standpoint and introducing us to new algorithms. The amount of effort invested in this project was substantial, but now it is not a mere research exercise in notebooks but a complex production-ready solution.