DS-5110-Books Recommendation System

Lasya Manthripragada
manthripragada.l@northeastern.edu

Faiz Khwaja
khwaja.f@northeastern.edu

Abstract:

The general idea of a recommendation system is to predict the preference/liking of a user towards an item. Recommendation systems are very impactful and are the hot topic for any retailers/online system. For example, Goodreads [1] recommends books based on other books that the user read and tries to find similar users. Here, we are trying to implement the same: predict the user's preference towards books by considering user's previous reads, and other users who are similar to the current user. User should be able to login in, and get recommendations given by different algorithms. We used the dataset from IIF [2], it consists of information on users like books they rated, age and location. And book details like publisher, year of publication, ratings given to the book, author. We also scrapped more data from Google API [3]. We implemented algorithms like Item-Item, KNN, User-User, User based autoencoder, popularity based, content based, and correlation based.

Introduction:

Generally, Recommendation systems are used to increase the sales by giving more personalized offers to the users, and it helps the users to access the content that they like the most and thus strengthen the customer experience. The dataset [2] contains three files.

1. Users
   It has three features with User-ID, Age, Location. User-ID is a unique identifier for the user.
2. Books
   It has ISBN, Book-title, publisher, year of publishing, author, and the Image URL. Image URL is going be useful while building the website. ISBN and Book-tile are used to identify a book, they both are unique.
3. Ratings.
   It has the mapping of each user to the books they read and rated.

There are around 10M ratings given to the books by different users. There are around 271K books, and 278K users.

We wanted to implement content-based recommendation systems [4], so we tried to scrape data from Google API [3] so that, if we get description we can perform TF-IDF [5] and find the similarity between those vectors and recommend books based on the similarities. Each algorithm will be discussed in more detail in the coming sections.

Additional data:

a. Genre: It gives the genre of each book.
b. Description: It gives a brief introduction to the book.

We used Django to build a website, so that a user can log in, and see the books that were recommended to them by different algorithms. Which will be discussed further in the Project description section. We have used PowerBI to perform exploratory data analysis [6].

Data Preprocessing:

As mentioned in the above section, we had to over 10M ratings, and around 270K books and users. However, not all the books and users were reliable. To be more precise, a book like "Pride and Prejudice" has an average rating of 4.3, and book has been rated around 400K times on Goodreads, whereas a book like 'Pencils' has an average rating of 4.7 but has been rated only 10 times. We cannot rely on these ratings to declare 'Pencils' a better book than 'Pride and Prejudice' which is not the case. When users only rate 10 books, we cannot predict the user's favorite books. Hence, after many trials, we decided to discard users who rated at least 200 books, and books which have been rated at least 50 times.

After performing EDA on data, we also discarded the NULL entries. We plotted a histogram of user's age. As shown in fig 1, ages above 100 and below 4 were the outliers. So, we discarded those users as well.

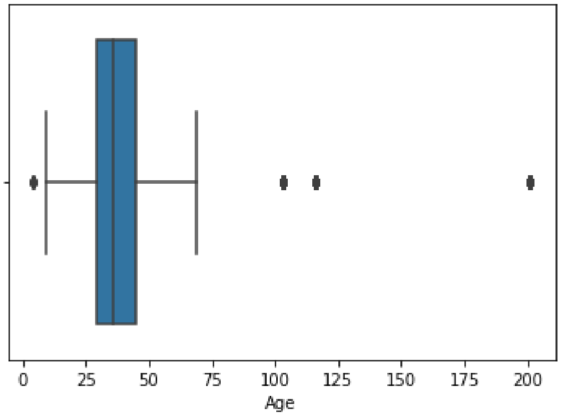
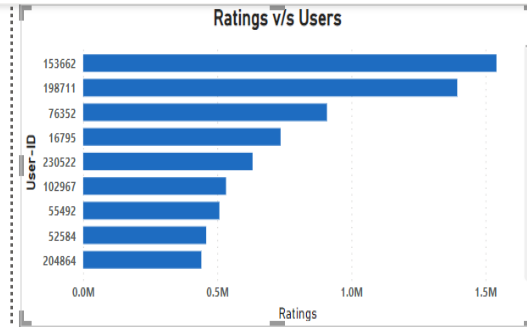


Fig 1 Age Histogram.

Some additional EDA. In fig 2, we can the top 10 users- users who rated the most number of books, and in fig 3, we can we the top 10 books which rated the most.



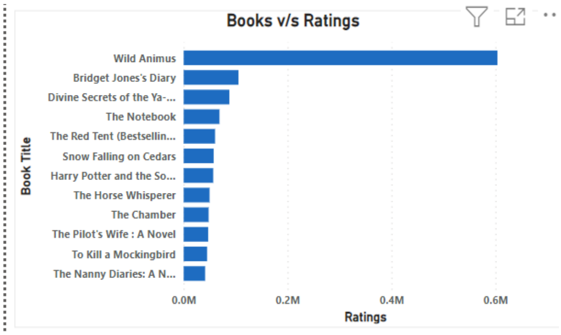
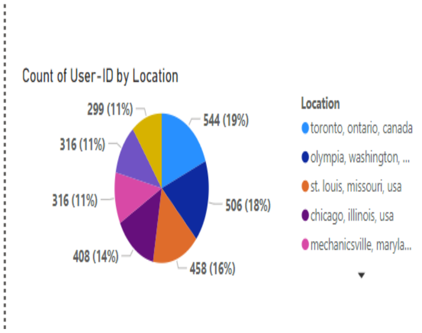


Fig 2 Top 10 Users

Fig 3 Top 10 Books

Below are some additional analyses, like in fig 4 shows the top locations that have the greatest number of users and Fig 5 shows the top 10 ages that users belongs to.



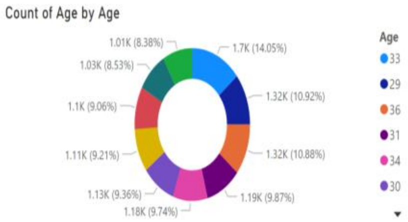


Fig 4 Top 10 locations users belong to.

Fig 5 Top 10 Ages users come from.

Finally, we plotted a barplot to show the number of books released in each year.Fig 6 shows the same. One observation we made was features like age, location did not contribute much to the users ratings towards a books. And the authors, year of publication, and publishers didn't contribute either. So we didn't utilize the these features. However, we needed more data to build a profile so that we can implement content-based algorithm.

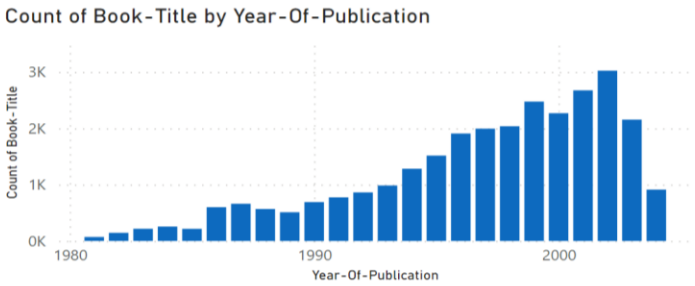


Fig 6 Number of books published in each year.

We also scrapped additional information on the books, such as genre and description. Genre was highly biased towards 'Fiction' , as shown in the fig 7. We decided to not go with the genre as it was futile. We also scrapped description. It was quite useful. Although, we couldn't find descriptions of some books, so we discarded those books as well.

Finally, in our dataset we have 30K ratings, and 637 unique users with 1001 unique books. The scripts regarding the scrapping of the data and the data analysis in detail can be found in our repo [7]. There is more analysis in our PowerBI dashboard.

Implementation of Algorithms:

Given the book ratings of a user, and the ratings given to the book by the user, we will try to predict books that will be potentially liked by the user. For this we implemented six models. One of the first algorithms we implemented was the popularity-based algorithms [8], which recommends the trending books to the user.

i. Popularity-Based Algorithm:

It takes the weighted average of the ratings given to the books. Then the application will give out the top books i.e., the books with the highest ratings will be the trending/popular books.

Weighted_Rating = [n]*[R_avg]/(n+m)+[m]*[Avg]/(n+m)

Where, n is the number of times the book has been rated
m is the minimum number of times a book should be rated to be reliable.
R_avg is the average of rating of the book
Avg is the average rating all the books
Here, m is 50, as we are only taking the books that have been rated at least 50 times. The formula is based on the IMDb rating [9]. The formula is based on 'a priori noncommittal' which means we cannot rely on books that have rated very few number of times, so we take a book that is firm and reliable. Please refer [9] for more information in this.

The results after implementing made sense to us, we can see that it has Lord of the rings, Life of pie etc., which are few of the classics.

ii. Correlation-Based Algorithm:

Books that highly correlated to the books that the user read will be recommend. The implementation of this algorithm was straight forward. To find the correlation between books we used Pandas correlation method [10] which uses Pearson correlation coefficient. Below is the formula for the same [11].

$$\mathbf{r} = \frac{\mathbf{n(\sum xy) - (\sum x)(\sum y)}}{\sqrt{\mathbf{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}}$$

Where,

- r = Pearson Coefficient
- n= number of the pairs of the books
- $\sum$xy = sum of products of the paired books
- $\sum$x = sum of the x scores
- $\sum$y= sum of the y scores
- $\sum x^2$ = sum of the squared x scores
- $\sum y^2$ = sum of the squared y scores

It can be interpreted as the measure of linear correlation between two sets of data. When we implemented the algorithm, the results were straight forward, if the user liked one of the “Harry Potter” books, the algorithm recommended the other ‘Harry Potter’ books. It is a good algorithm, but it doesn’t recommend outside of the user’s likings.

iii. Content-based Algorithm:

To implement this algorithm, we need to build a user/book profile, and thus find the similarities between them. To do this we didn’t have enough data, so we had to scrape data like description. This was discussed in more detail in the data preprocessing section. We performed TF-IDF on the descriptions of each book, and performed cosine similarity on the same,

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF - IDF(t,d) = TF(t,d) * IDF(t)$$

The above formula shows how we calculate TF-IDF, however, we used the Sklearn TF_IDF package. [12] After we performed TF-IDF, we calculated the cosine similarity to find similar books. We took top 10 similar books, and the books that the user rated to predict the ratings of the books that user haven’t rated yet. Below, are the formulae for cosine similarity, and predicted rating.

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

$$p_{u,i} = \frac{\sum_{j\in N} s(i,j) r_{u,j}}{\sum_{j\in N} |s(i,j)|}$$

Books that were rated by the users were considered while taking the most similar books. The RMSE was 1.35 which was decent. The algorithm was always recommended books that were in their domain, it couldn’t predict new books i.e., outside of their content profile.

iv. Item-Item Collaborative filtering:

In Item-Item, we find the books that the user read, and recommend the books like that book. First step was to find the nearest neighbors to the items. We used KNN to find the nearest neighbors. We took top 5 nearest neighbors for each book and took all the books which were rated by the user in that neighborhood. And we calculated the predicted rating of the user by the below formula.

$$p_{u,i} = \frac{\sum_{j \in N} s(i,j) r_{u,j}}{\sum_{j \in N} |s(i,j)|}$$

Where, N number of nearest/similar books to the book that we are predicting for. And this N books must be rated by the user as well. The RMSE score was 1.85. One of the main drawbacks was that it couldn’t recommend books to a user who hasn’t read any of the books. It only recommends the most popular books. We had lesser number of users than items so we can say that users are more stable than items and user based would give better recommendations. The detailed implementation of this algorithm is our repo, please refer.

v. KNN-based Algorithm:

To implement this algorithm, we took two approaches. We first needed to find K-nearest neighbors, as mentioned above we used 5 as the neighborhood score. We used cosine similarity to find the nearest neighbors. We implemented this using the Sklearn library [13]. We built a pivot table to calculate the similar books, with books vs user matrix. We calculated the cosine similarity matrix, and for each user we recommended the books to the user: which are the closest books in similarity with the books that the user already read. This we calculated the item similarity books.

After implementation, we saw that if the user read books like ‘Harry Potter’ and other fictional books, the model recommends the same books i.e., books in the same genre like the other ‘Harry Potter’ books and fantasy books.

We also implemented KNN baseline using the Surprise python library [14]. The RMSE was 3.2 which was very less when compared to ‘Item-Item’ and ‘Content-Based’. It performed almost the same as the Baseline algorithm, which we also implemented using the Surprise algorithm.

vi. User Based Collaborative filtering using TensorFlow:

To implement this algorithm, we heavily referred from [25], they implemented Autorec model by using the partially built users vectors – where each value represents the rating of the book given to the book by the user. However, we call it partially built because each user would not have rated all the books.

r (u) = (Ru1, . . . Run)

Where each Rui, represents the rating given to the book by the user.
Our model takes each user’s vector (the partially built vector) as an input, project into a lower dimension space and the model tries to reconstruct the vector. Fig 14 is a representation of user-based CF. We can see that an item vector is sent and the weights are updated accordingly. In our case we take Only the values i.e., the ratings given by the user to update the weights and bias terms. The picture is taken from [26].

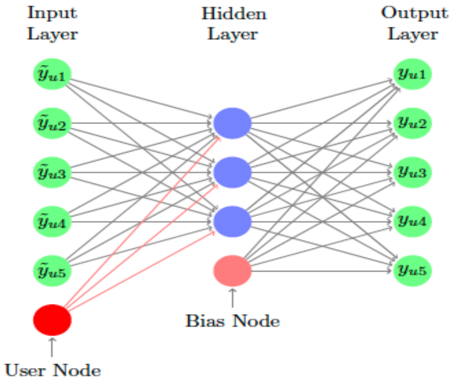


Fig 14 User based AutoRec.

Initially, the weights and bias terms are randomly initialized from gaussian distributions. The activation function we uses is the sigmoid function.

$$\sigma(x) = 1/(1+\exp(-x))$$

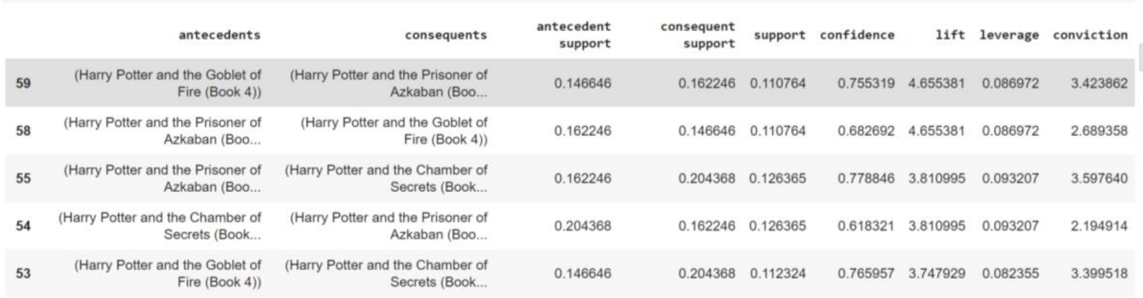
Model has 10 hidden layers with 10,5,5,10 neurons respectively, and all the layers are densely connected. The model is trained on user vectors. We divided the users into batches of size 35 with 100 epochs [27]. We defined the loss function to be mean squared error, and the RMSPropOptimizer to be the optimizer. The RMSE score was 0.6, and it performed very well when compared to other algorithms. The user who read ‘Harry Potter’ was recommended books like ‘Killing a mocking bird’ and other fictional books like ‘Bridget Jone’s diary’. It was recommended better books, and also outside of the user’s content profile. Considering all these factors, it is the best model for our book recommendations system.

vii. Apriori Algorithm:

We implemented Apriori to see the association between the books. We took all the users and list of all the books that they rated. We used the mlxtend python library [15] to transform the data into a transaction table. Association algorithms used two parameters – support and confidence.

Support is the indication of how frequently the books appear in the database. It helps give the most rated books and/or the books which we might be reliable on. After many trails of using different support, we decided the support should be 0.1.

Confidence is used to find the rules of the dataset, which means that, we will find books that will lead the user to read the other books. It basically gives the probability of an association occurring. It is the likeliness of the occurrence of a book when there already exist books that the user read. Amazon used this kind of algorithm, which is also called the “Market Basket Analysis’. [16]. After many trials after finding the minimum support of 0.1, we decided the minimum confidence to be 0.4.
Below fig 7 shows some of the top rules that we mined. We can see that books like ‘Harry Potter’ will lead to books of the same genre. We can also see that most of the book, doesn’t matter which genre, they lead to some one of the most famous books like ‘Wild Animus’.



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
59	(Harry Potter and the Goblet of Fire (Book 4))	(Harry Potter and the Prisoner of Azkaban (Boo...	0.146646	0.162246	0.110764	0.755319	4.655381	0.086972	3.423862
58	(Harry Potter and the Prisoner of Azkaban (Boo...	(Harry Potter and the Goblet of Fire (Book 4))	0.162246	0.146646	0.110764	0.682692	4.655381	0.086972	2.689358
55	(Harry Potter and the Prisoner of Azkaban (Boo...	(Harry Potter and the Chamber of Secrets (Book...	0.162246	0.204368	0.126365	0.778846	3.810995	0.093207	3.597640
54	(Harry Potter and the Chamber of Secrets (Book...	(Harry Potter and the Prisoner of Azkaban (Boo...	0.204368	0.162246	0.126365	0.618321	3.810995	0.093207	2.194914
53	(Harry Potter and the Goblet of Fire (Book 4))	(Harry Potter and the Chamber of Secrets (Book...	0.146646	0.204368	0.112324	0.765957	3.747929	0.082355	3.399518

Fig 7 Top association rules.

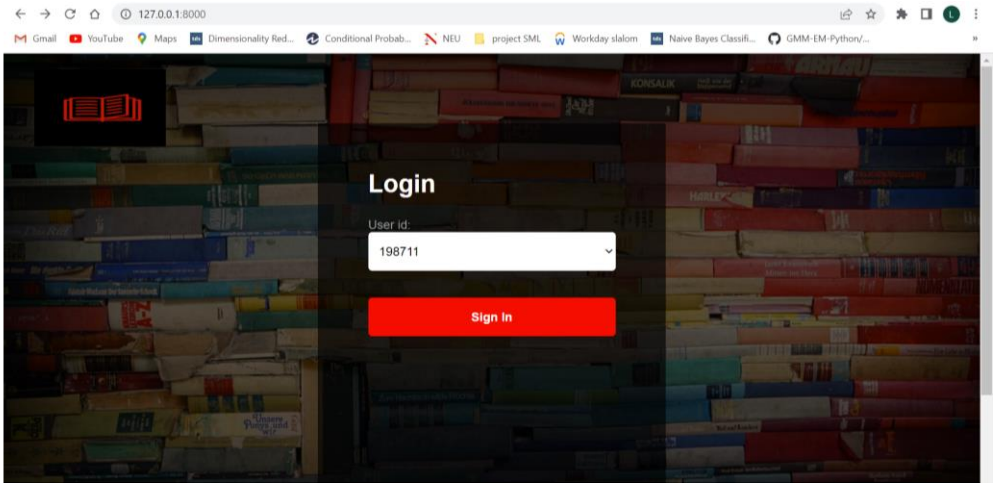
We also implemented Baseline algorithm, and SVD using the Surprise algorithm. They didn’t do exceptionally well especially when we compare them to algorithms like Content-based, User-based, and item-item Collaborative filtering. We are going to discuss the implementation of the web application, and the integration of these models with the same in detail in the next section.

Tools and Technologies:

To build the web application we have used Django [17]. It is an MVC web framework. It is a python-based framework. You can learn more about the web application tool by refereeing [17]. It free and open source. We have used TensorFlow v1 [18] and Surprise python libraries to implement the machine learning part of this project. We used HTML, CSS, JS to build the website in Django.

We have used some other helping libraries like NumPy [19], Pandas, Seaborn, and Matplotlib. Everything data manipulation we have to Pandas [20] to do it. For data visualization we have used Seaborn [21] and Matplotlib [22].

Below is how our website looks like. The user can log in and see the recommendation given to them using different algorithms we implemented. We did not include algorithms like SVD, Baseline, and Apriori as they were not significant. Fig 8 shows the login page of our web application.



127.0.0.1:8000

User id: 198711

Fig 8 Login page.

Fig 9 shows the book that the user already read, here there is only one book because we discarded books which have not been rated at least 50 times, and during scrapping of the data we discarded few of them too.

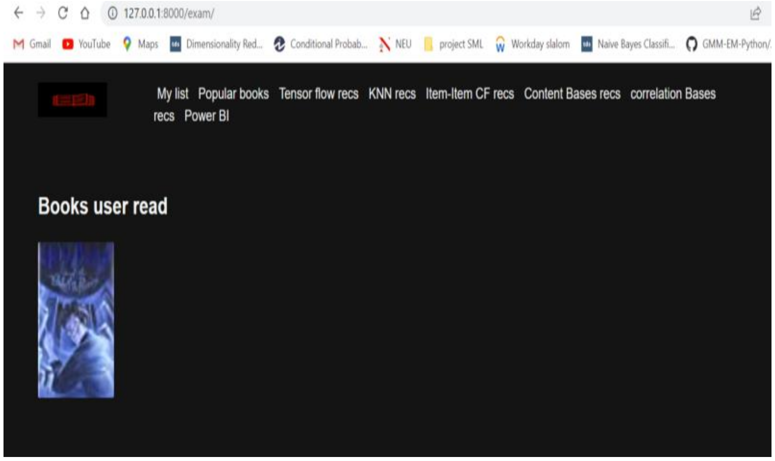


Fig 9 Books user read.

Here when the user selects the ID, the website will go the next page where all the algorithms' recommendations will be displayed. Fig 10 shows the popularity-based recommendations which will be same for all the users. You can see that 'Lord of the Rings' and 'Life of pi' were two of the most popular books.

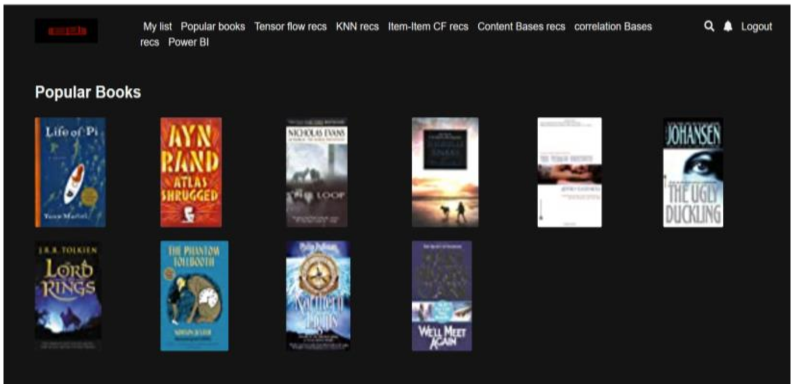


Fig 10 Popular/Trending Books.

Fig 11 shows the recommendations user got by the content-based and correlation-based algorithms. Both algorithms couldn't recommend books outside of the user's profile.

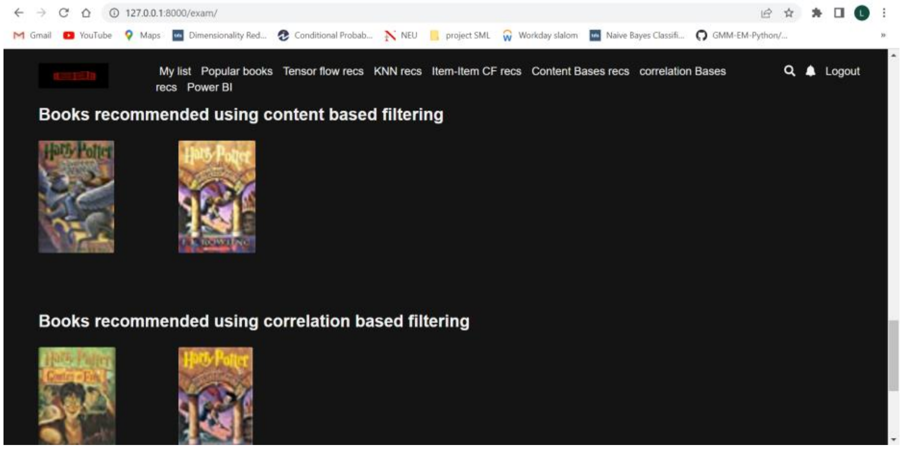


Fig 11 Content and Correlation based algorithms.

Fig 12 shows the books recommended to the user with Item-Item CF. You can see that that the algorithm recommended the rest of the ‘Harry potter’ books and other fictional books as the user read ‘Harry Potter’.

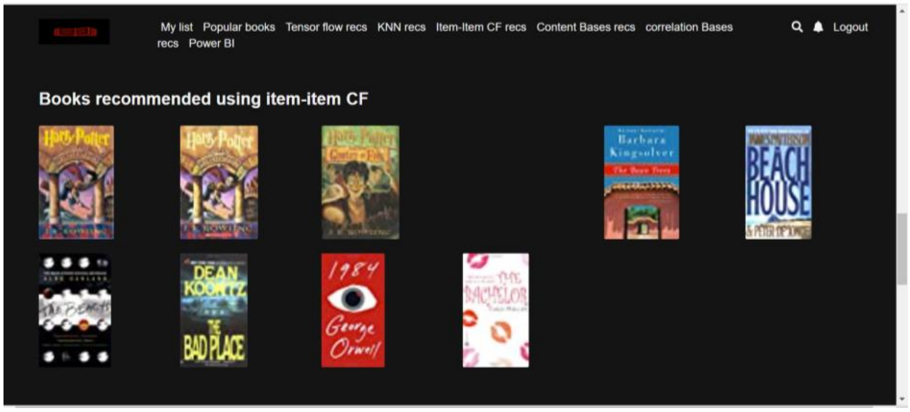


Fig 12 Item-Item CF.

Finally, we used autoencoders to perform user-based CF, and as shown in Fig 13 we can see the books that the algorithm recommended.

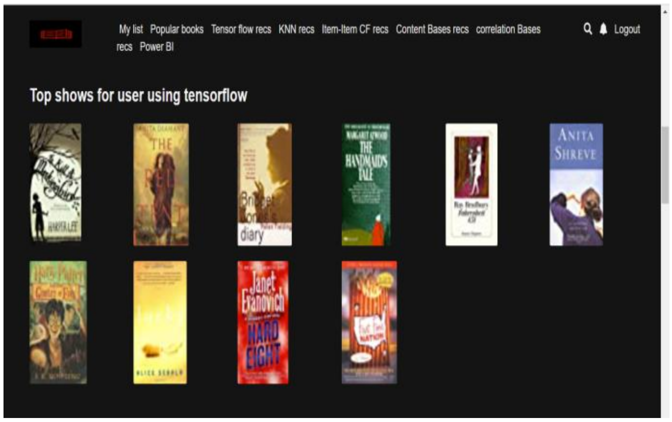


Fig 13 User-Based autoencoder.

As we can see, all the books were recommended by different algorithms were displayed. The results were desirable as all the books that were recommended made sense with the books that the user already read. Additionally, analyzed and visualized our data in PowerBI- which is a data visualization tool developed by Microsoft. It was used to analyze and draw insights from the dataset. You can view the dashboard by referring [23]. The template is taken from codepen [28].

Results:

To evaluate the algorithms implemented we used RMSE score as the metric. It is a measure of the difference between the actual values and the predicted values. It is calculated by using the below formula [24].

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

It can basically give the standard deviation- by how much is our algorithm off while predicting the user's rating, i.e., that noise is the variables that our model couldn't be able to capture. So, the lower the value the better, since will have very little noise.

Table 1 shows the RMSE scores of the different algorithms using which we predicted the user's ratings of a book.

Algorithm	**RMSE score**
Baseline	3.2
KNN	3.3
SVD	3.3
Item-Item CF	1.85
Content- Based	1.3
User-Based CF	0.6

Table 1 RMSE scores

As we can see in the above table, User-based autoencoder model has done exceptionally well, with 0.6 RMSE score. Although algorithms like Item-Item and Content-Based has done moderately well when compared to the Baseline model. In conclusion, the results were desirable, and our best recommendations model was User-Based autoencoder model.

Conclusion and Future Improvements:

In conclusion, we created a web application using Django where we displayed all the books recommended to the user by different algorithms like item-item, content-based and user-based etc., Our best algorithm was user-based autoencoder model, which also gave the least RMSE. Overall, content-based and item-item has also done moderately well.

We can perform neural networks for content-based item based. We decided to go with user-based since we had a smaller number of users than items. We can scrape more information on books like page count, maturity ratings, sale information etc., We can compute the predicted rating for all users so it will take lesser time. We can implement SVD by deciding on the number of factors. However, this was a little tricky as the genre or description was not given. Genre was heavily biased towards fiction. Which leaves us description, we can leverage this feature to decide on the factors.

The advice we want to give to the future DS5230 students is to be hands-on during the lectures, there are concepts like PCA and SVD which are needed to be understood from scratch. Also, start the project as soon as possible, it can get tricky.

References:

1. https://www.goodreads.com/
2. http://www2.informatik.uni-freiburg.de/~cziegler/BX/
3. https://www.googleapis.com/books/v1/volumes?q=isbn:
4. https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421
5. https://kinder-chen.medium.com/introduction-to-natural-language-processing-tf-idf-1507e907c19
6. https://powerbi.microsoft.com/en-us/
7. https://github.com/faiz2399/Book-Recommendation-System
8. https://medium.com/the-owl/recommender-systems-f62ad843f70c#:~:text=As%20the%20name%20suggests%20Popularity,user%20who%20just%20signed%20up.
9. https://math.stackexchange.com/questions/169032/understanding-the-imdb-weighted-rating-function-for-usage-on-my-own-website
10. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html
11. https://www.wallstreetmojo.com/pearson-correlation-coefficient/
12. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
13. https://scikit-learn.org/stable/modules/neighbors.html
14. https://surpriselib.com/
15. https://pypi.org/project/mlxtend/
16. https://www.amalytix.com/en/knowledge/controlling/aba-market-basket-analysis/
17. https://www.djangoproject.com/
18. https://www.tensorflow.org/api_docs/python/tf/compat/v1
19. https://numpy.org/
20. https://pandas.pydata.org/
21. https://seaborn.pydata.org/
22. https://matplotlib.org/
23. https://app.powerbi.com/links/vJWOomAZIF?ctid=a8eec281-aaa3-4dae-ac9b-9a398b9215e7&pbi_source=linkShare
24. https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e
25. https://users.cecs.anu.edu.au/~akmenon/papers/autorec/autorec-paper.pdf
26. https://towardsdatascience.com/recommendation-system-series-part-6-the-6-variants-of-autoencoders-for-collaborative-filtering-bd7b9eae2ec7
27. https://towardsdatascience.com/building-a-collaborative-filtering-recommender-system-with-tensorflow-82e63d27b420
28. https://codepen.io/cb2307/pen/XYxyeY