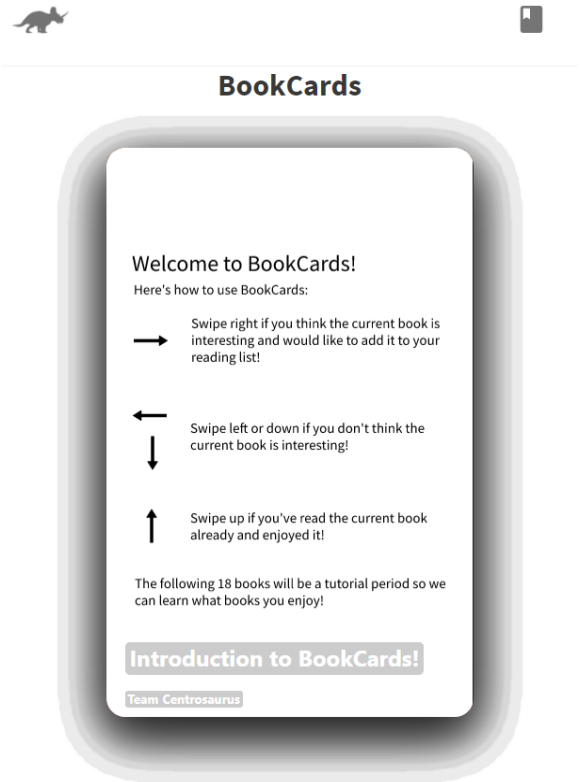
Team Centrosaurus Final Project Report

Abstract

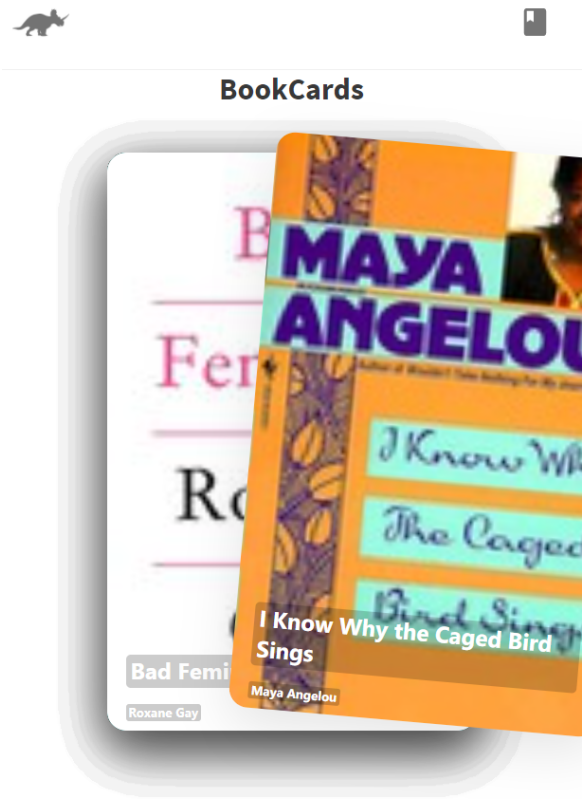
Our project makes book recommendations powered by machine learning. Our web app uses a user's reactions towards 18 representative books to match them with a reader in our existing dataset. The user is then presented with books that the machine learning model recommends, using data from the paired user, then prompting the user as to whether they like this book or not. This is done through an interface evocative of modern dating apps like Tinder or Bumble. As the user swipes, they begin to cultivate a list of books they've liked, and the underlying machine learning model adapts based on their swipe activity. Thus, as the app is used, the model learns more about the user's book tastes, improving its future recommendations.

Web App Walkthrough

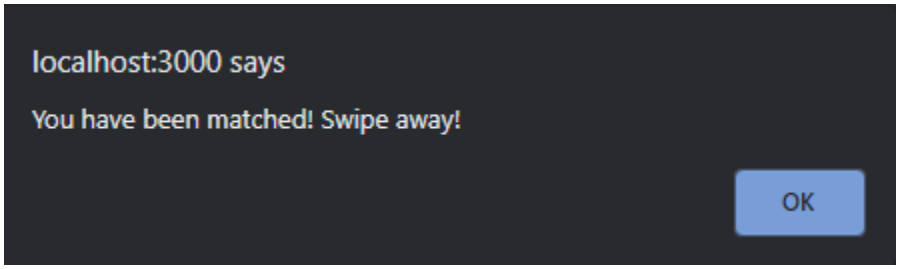
When you open the BookCards app you are greeted by a tutorial card explaining the function of each swipe direction.



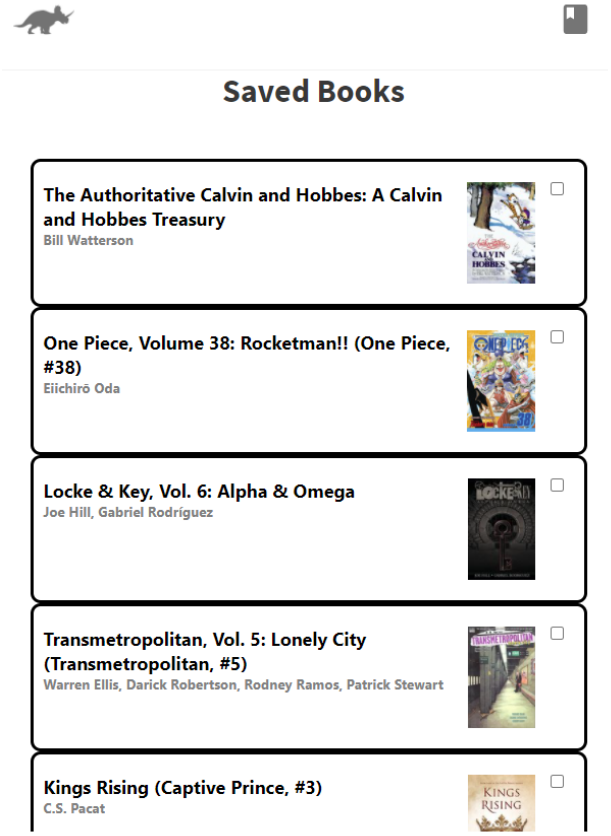
Swiping past this, you are then presented with 18 books, chosen to represent a range of genres, which are used to determine which user from the dataset you most resemble. As you swipe your responses are communicated to the backend, which interprets a positive swipe as 5 stars and negative swipe as 0 stars. The set of 18 book-rating pairs are then used to find the user with whom you are most similar to.



This process takes time, so when you swipe on the last book in the representative set, you are presented with a waiting message. The message informs you to wait until the app alerts you that it has completed the pairing process. When the back end has selected a similar user, your browser will alert you. You can now swipe through the books highest ranked for this user.

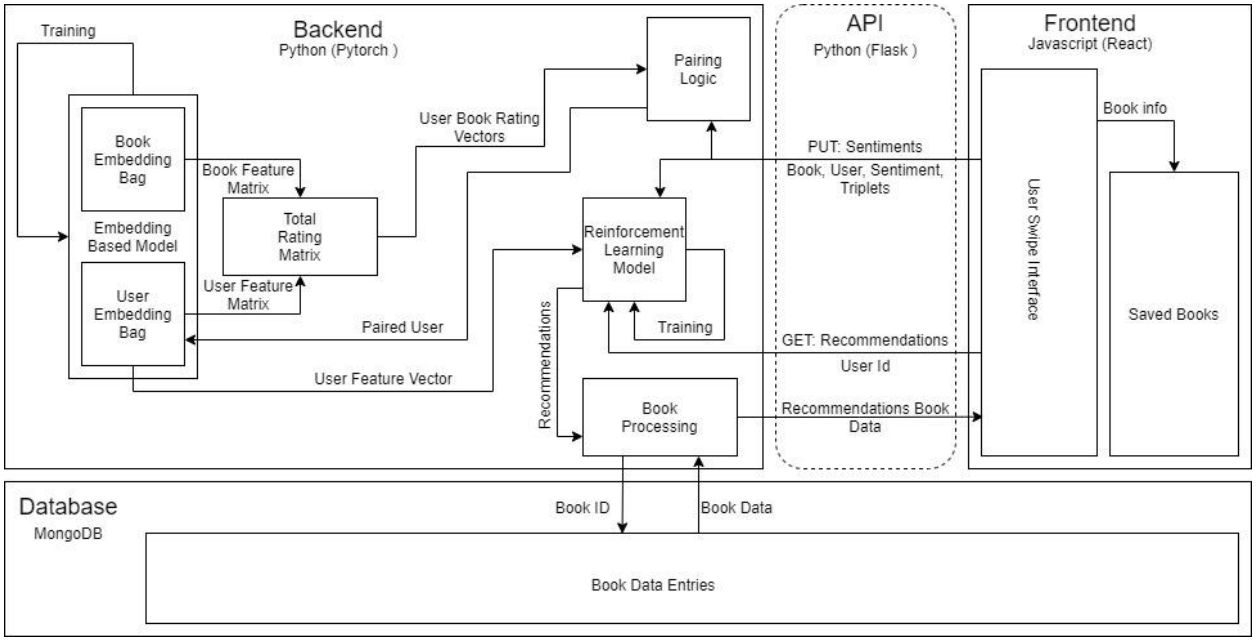


As you swipe, the reinforcement learning model trains itself to improve the books that it will recommend to you. The books you swipe right on get added to a saved books list available through a button at the top right corner. By clicking the check box on the top right of each card and switching back to the home page, by clicking the centrosaurus icon on the top left, the user is able to delete that book from the saved list.



That's it, using BookCards is a simple and easy way to get personalized book recommendations, where your next read may only be a swipe away!

Component Diagram



Component Descriptions

Frontend

The frontend component of the project is evocative of Tinder where each card consists of a book's title, author, and the front cover. Each card can be swiped in a certain direction to record the user's preference on that entry. The user can swipe in four directions:

- Right: The recommended book seems interesting and the user would likely want to read it. This sends a positive response to the back end and adds the book to the user's reading list.
- Left/Down: The user doesn't find the recommended book interesting and would not see themself reading it. This sends a negative response to the back end.
- Up: The user has read the current book and enjoyed it. This sends a positive response to the back end, but doesn't add the book to the user's reading list.

The front end was developed with React, and consists of 3 main components: App, BookCards, and SavedList. BookCards and SavedList are passed props from App to hold the states of the current list of books to recommend, the list of saved books, and the functions to update the states. (For more detailed explanations, please refer to the comments in the GitHub repository)

App

App.js contains code that determines which subcomponent to render, and, as aforementioned, the states of book lists. To determine the current page to render, there is a separate state that contains a boolean for whether the user wants to view their saved books. To toggle between the main page with the book cards and the saved list, a header is inserted at the top of the page. Upon clicking either the centrosaurus icon or the saved book icon, a function will be called to update the boolean state accordingly. The header is a simple flex box with a row flex direction and the SavedList icon is from the material-ui React component. A ternary statement persistently checks the state of the boolean, and therefore renders the correct component.

BookCards

BookCards.js is the main page where users are shown books and where they can input their sentiments. The cards and swiping motion are from a React component called react-tinder-card (https://www.npmjs.com/package/react-tinder-card). Upon swiping on a card, a switch statement handles the direction of the swipe and sends information to the back end through the API (the format will be explained in the API section). Additionally, the book lists are both updated upon swiping the cards by using their respective set functions (a part of the React component) and removing or adding the current book. When the card stack is emptied, a call is made to fetch the next 5 books to recommend. For the style of the book cards, a large influence comes from Tinder. The shape of each entry is modified to better fit the common aspect ratio of books. The book's title and author are aligned a distance away from the bottom of the entry and have a shadow to ensure the white font can be seen on bright covers.

SavedList

When toggled, SavedList renders the user's current saved books in a vertical list. SavedList is passed the savedBooks prop from App, and displays each entry using the map() function. It displays the book title, author, cover, and a checkbox for each entry in the list. Upon clicking on the checkbox, the selected entry is removed from the saved list by calling a helper function that utilizes the saved list's set function. The styling of SavedList is a container in the centre of the screen, and each entry is a flexbox. Additional code is added to display a neater scrollbar rather than the default one.

API

The API is a Flask server with two different request options: get and put. The 'put' method is used by the front end to send the data to the back end that is necessary for updating the model for each swipe. This includes the user id, the id of the book being swiped on, the sentiment the user chose for that swipe (1 for positive and 0 for negative), and a boolean flag called 'init_flag' which specifies if the swipe is one of the first 20 swipes and therefore the backend is notified to initialize the user's model. The 'get' method is called when the front end needs 5 more book recommendations. The front end sends a 'get' request by using the path '/book/<user_id>' with the specified user id. This function calls a get_recs() function from the

back end, converts the returned object into a JSON object, and returns that JSON object back to the server.

Dataset

The dataset is created from the “goodbooks-10k” dataset on GitHub and consists of two parts: the user/book/rating tuples, and the data associated with each book (book title, author, publication date, etc.). Both parts of the dataset have been processed to not include books with no title, not include books in other languages, and have a unique id for each entry. For the book data part of the dataset, we have removed all categories we don’t need, and have stored the book id, book title, author, and image URL for each book in the dataset on the Mongo server. We did this by reading the csv file to a Pandas dataframe, removing the unnecessary columns, converting the data frame to a dictionary, creating a Mongo collection for the book data, and inserting the entire dictionary to that Mongo collection. This improves the overall efficiency of the application.

Embedding Model

The Embedding Model is the first machine learning component of the project and is used for the foundation of much of the work the later Reinforcement Learning (RL) model does. The Embedding Model is used to generate both the user vectors and the book feature matrix that the RL model uses to make its predictions. Additionally, it produces the 2 dimensional matrix that is used in the pairing process.

The Model itself consists of two Embedding Layers and a Dropout Layer used in training. The first embedding layer is used to generate and store the embeddings for each user in the dataset, while the second embedding layer does the same but for books. The result of the forward call on the model is computed by taking both user and book indices as input, and calculating the dot product of their resulting embedding. This dot product is then used as input to a sigmoid function, and multiplied by 5.5 to bring it to scale with the ratings.

During training a stochastic gradient descent optimizer was used, alongside a learning rate scheduler which adjusted the learning rate after each batch. We found the best training parameters for this task to be an initial learning rate of 10^{-6}, an optimizer momentum of 0.75, a max learning rate of 0.1, a batch size of 32, and a total training duration of 20 epochs. To combat overfitting observed during initial training sessions, the Dropout layer was set to a dropout probability of 0.25 and applied to both embeddings. Using Mean Square Error as our loss function, we were able to minimize our loss to 0.67, meaning the model predicted ratings within 0.8 stars on average.

From this model, the embedding layers are extracted as 2 dimensional tensors, and after some dimension manipulation, multiplied together to produce a 2 dimensional tensor of ratings, with book_id, and user_id as indices. This matrix is necessary for the pairing logic, which compares the absolute difference between the swipes sent by the frontend user during pairing, and how all the users in our dataset rated that book. Once the paired user is found, we index into the user embedding layer of the model using the paired user’s id as an index to extract their

embedding; we call this embedding the user feature vector. Additionally, the two dimensional tensor representation of the book embedding bag is extracted to allow for mappings between a book's id and it's embedding in the RL model.

Reinforcement Learning Model

The Reinforcement Learning Model is similar to the embedding model, but is much simpler and is actively used when the app is running. The RL model has only one layer: an embedding layer initialized with the weights of the user feature vector found above. The RL model also uses an external matrix, the aforementioned tensor representation of the book embedding bag, which it takes as input to it's forward function. This matrix is kept external to the model since it's a large unchanging matrix common to all users. During the forward operation of this model, the dot product between the embedding for the book, which is extracted from this matrix, and the stored user feature vector is computed, and put through a sigmoid function before being output.

Upon setting up a new user, the RL model is used to compute the ratings for all books in the dataset - ratings which are then used to order the books. As the app is used, a subset of 300 books are repeatedly re-ranked and reordered to ensure the user is always getting the best recommendation. A subset of 300 is used as it is large enough to ensure that the first books will be rated well but small enough to not be too costly to continuously resort. Books are sent from this list to the front end in batches of 5, and when this subset is 200 books long or less it grabs an additional 100 from the head of the ranked list of all books. This ranking occurs without input from the user, and with the model in evaluation mode.

As the user swipes, the model runs in training mode to learn by comparing its predictions against their swipe, and improve it's rating. Using a stochastic gradient descent optimizer with a learning rate of 1 and a momentum value of 0.3, the model trains by comparing its current prediction for a book, the probability a user will like it, with whether or not the user actually swiped. This loss is used immediately to retrain the model before the next swipe.

Design Tradeoffs

In designing our app, there were competing needs between different elements of the project that we needed to balance. Maintaining this balance led to some of our design choices. Some detail on some of these design choices is provided here.

For the initial ratings we chose to go with a constant 18 books used as both a tutorial period and for pairing. This decision was made as it allowed us to have a tutorial period that helped orient the users to the front end while also having functionality for the backend. Additionally, by using the same swiping here instead of user input ratings we were able to maintain the same interface and prevent the frontend from being too clustered.

For the dataset we used, we initially considered including much larger datasets, but the ones we found would've taken too long to process. We had found a dataset from amazon.com containing over 50 million book ratings, but processing this dataset would require us to scrape information from Amazon. Doing this scraping was something we tested but determined, due to

the dataset size, would be too costly. Additionally this dataset did not merge well with our already selected GoodBooks dataset due to a higher potential for duplicates, as the Amazon dataset treated different editions of a book as different books. Also, not every book entry contained information such as the ISBN number, which is required to connect multiple datasets.

Finally, at the end of the tutorial period, we needed to find a way to get the user to wait before proceeding, as the frontend advanced faster than the backend. To do so, we created an info card asking the user to wait, and set up the frontend and backend to alert the user when the books were ready. This slowed down the users flow through the app, but it allowed us to ensure they were being paired to a user before proceeding.

Future Improvements

While we met our project goals, the project as is would need to undergo some changes prior to deployment. To officially deploy this product, we must first implement a multi-user system since currently the app does not store user info. The app can be easily upgraded to be able to handle multiple users by storing all user specific information in the database that we have already established. User specific information that will be stored include: personal information, a personalized model for each user, and their reading list. Another improvement that we could incorporate is the ability to constantly update the book database so that the program will not run out of books to recommend to the user. Currently, the database contains less than ten thousand books to recommend to each user. We are able to increase this amount of books by either appending other databases to the current working database or even creating our own either manually or automatically, using Python scripts.

Moreover, we could also implement some improvements to the UX design of the front-end. For example, the book images that are shown on the book cards are quite low resolution and some are even missing images. To resolve this, we would either manually update the database with new images of each book, or use a Python script and the Beautiful Soup Library to do this automatically. Another feature that we would implement would be providing brief descriptions of each book on the back of the book cards. Essentially, the user would be able to click on the card and be able to read a short description of the contents and genre of the book, so they are able to make a more educated decision on their interest in the book.

Overall Team Member Contributions

Byrne, Declan

Declan’s contributions for this project were primarily focussed on the backend. He began by developing the embedding model, testing many different parameters to minimize loss. He then began to look into different methods of implementing reinforcement learning, before settling on the approach described above, which he and Harvir both worked on. He spent time making sure this model was working on a small scale before extending it to the larger scale project. He

then worked to implement the rest of the backend to interact with the models. Once implemented he began integrating the backend and the API with Anna, and after that worked on debugging the backend code as API calls were made both in isolation, and in the context of the frontend. Finally he reprocessed the dataset to eliminate foreign language books, and other books we did not want to be recommending, as well as to reduce the amount of data needed to store in the MongoDB database, he then retrained a new final model with this reprocessed dataset.

Dhaliwal, Harvir

Harvir's contributions to the project were mainly focused on backend development. He began by developing the initial Embedding Model that would be used to bootstrap a new user as well as assist with the Reinforcement Learning Model. He then began researching different methods to implement the Reinforcement Learning model as this was something we were not very familiar with. As a lot of the methods found were very complex and difficult to implement in the context of this project, Declan and Harvir designed their own version of a Reinforcement Learning Model which they thought would work for this project. They both worked together to implement and test this model to ensure that it was actually learning. After completing this, Harvir worked with Ritchie to help design the UI and help with some bugs that appeared when Ritchie was implementing the front-end. Lastly, he spent the rest of his time improving the Embedding model, to improve the overall quality and accuracy of recommendations.

Riley, Anna

Anna started working on obtaining and processing the dataset. She worked with multiple datasets to see which ones would be feasible and helpful for the project. After using various methods to make multiple datasets work for the project, some datasets were too large or had insufficient data so we decided to stop working with this dataset. Anna then started working with the API and setting up a Mongo database to store user data and models. She created a REST API using Flask. She initially created the functionality of the API to send 5 book recommendations to the front end, store a user's initial ratings of 5 random books, and update user data each time a user swiped, respectively. She initially created a Mongo database for the user's initial ratings and swipe history. After the team decided to make some changes to the structure of the app, she changed the API functions to be just get and put (and a simple delete user function) to get 5 updated book recommendations, and update the model of a user using their swipe data. She made simplifications to the Mongo database to account for the overall design changes, then explored other ways to utilize the Mongo database. She stored the book data into the Mongo database and adjusted the dataset to retrieve the data about each book from the database. She helped with debugging the API calls from the front end, and this led to deciding to change the format of the API so that the calls from the front end could route to a specific path for requesting data and sending data to the back end using @app.route(). She then helped the overall debugging process, and she contributed to researching the best design choices throughout the project. Additionally, she selected the representative 18 books to be hard-coded into the initial user tutorial period, and found the goodreads ids of those books.

Xia, Ritchie

Ritchie started working on processing the dataset. Although only the GoodBooks dataset is used in the final product, Ritchie also worked on the Amazon dataset. For the GoodBooks dataset, Ritchie helped extract information from a given GoodBooks ID through searching for the matching table entry in the separate csv file. For the Amazon dataset, he extracted information from elements on Amazon's website for each book ID by appending zeros to the concatenated Amazon UIDs and using Beautiful Soup to read the HTML. However, the dataset was determined to be too big to use, and many entries on Amazon's website were missing crucial information such as the ISBN number which would be required to connect books with different editions, as well as connecting to the GoodBooks dataset. He assisted in sorting the matrix of all user and book combinations so that each row would be sorted in descending order by rating. For the latter half of the project, Ritchie created the web app. He utilized React components and states to store the data of the recommended books and used CSS to stylize the layout of the cards and the saved list. He debugged the API calls to ensure the proper information was being passed, and unpacked the 5 recommended books from their JSONs.

Video Link

https://youtu.be/B6URcxePV0s