# CZ4125 Developing Data Products

# Team Project:

# Semantic Search and

# Personalised Recommendation of Books

| | |
|---|---|
| Tan Ching Fhen | U1920787D |
| Kusalavan Kirubhaharini | U1921635E |
| Inbanathan Vaishnavi | U1921223D |
| Sim Shi Qian | U1920856E |

## 1.1 Introduction

With the exponential growth of data in recent years, consumers are faced with the problem of information overload - it is challenging to sieve through big data for the most useful information. Search engines and recommendation engines are valuable tools for this reason, because they help businesses and users find the information they need. Therefore, in this project, we want to understand search and recommendation better by developing a data product that utilises both these tools.

## 1.2 Ideation

Though there are many areas of application for search and recommendation, finding an area of application was not trivial because it was difficult to find the right data for both search and recommendation. The particular recommendation technique we wanted to focus on was collaborative filtering - a method of making automatic predictions about the interests of a user by collecting preferences or taste information from many users. Acquiring the user-item interaction data required to train collaborative filtering models was rather challenging; since we do not have a large user base, we can only rely on publicly shared data, which limits our options significantly.

Ultimately, we settled on developing a search and recommendation data product for books. The training data for collaborative filtering could be downloaded from Kaggle, while additional textual data could be crawled or called from an API.

In particular, our objective was to develop a fully semantic-based search engine. The reason being that semantic-based search for books is much less common; many existing search engines for books such as Goodreads [5] or Amazon Books [6] utilise lexical-based search.

## 2.1 Application Objective

With that being said, our goal was to develop a book application that performs semantic search, personalised recommendations, and sentiment analysis of reviews.

Links to the application demo and development codes are in Sections 7 and 8.

## 2.2 Features

Our data product has the following features:

1. Semantically search books based on text
2. Recommend personalised books based on past user interactions
3. Display book information - summaries, genre, review, and review sentiments
4. Filtering of books based on genre
5. User login and signup

## 3.1.1 Data Sources

In this chapter we describe the various ways we acquired the raw data required for our book application.

## **3.1.2 Kaggle - Books Dataset**

To train collaborative filtering models, we require user-interaction data. In the Kaggle Books Dataset [1], the ratings.csv file contains the ratings of each user to a particular book. We directly downloaded this data to train our collaborative filtering models.

We also utilised the books.csv file, which contains basic information of the books such as book title and book author. Each book has an identifier, ISBN, which we will use to obtain additional information like summary and reviews from other data avenues.

## **3.1.3.1 GoodReads**

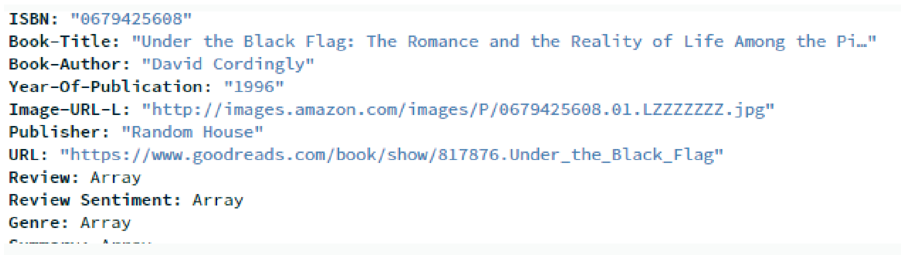
Initially, Goodreads data was to be scraped through their API which allows excess to approximately 10 million reviews for their accumulated 700,000 books, however the API methods require a developer key. As of December 8th 2020, Goodreads no longer issues new developer keys for their public developer API since they planned to retire the current version of these tools. This makes scraping the next viable step for collecting data on goodreads. The relevant URL was obtained from the aforementioned csv file, however, there were various challenges that would be shared below. Being a resource where the main audiences come from online sites, Goodreads was aggressively anti-scraping, which resulted in various strategies to overcome their policies.

An initial comparison of the python libraries used for scraping indicated that the speed of using Scrapy over Beautiful soup was faster by approximately 7-13 times, therefore scrapy was periodically used. Additionally, there is the added benefit of implementing parameters in requests to attempt to combat anti-scraping policies. One major drawback of the URL link in the initial CSV is the amount of redirects it will cause. After manual examination, it was found that attempting the scraping on the URL: "https://www.goodreads.com/" followed by the ISBN titles of the different books allowed lesser redirecting. Even with the correct redirected URLs, a strategy goodreads has is to redirect to the same exact link, which causes constant, aggravating redirection.

| **Common errors** | **Strategies used** |
|---|---|
| Redirection | Use altered ISBN links, add dont_redirect=False on the request parameter |
| HTTP Status Code 301: "Moved Permanently". Status code 301 is shown when a page has been moved. They are colloquially known as "301 redirects". Since the amount of redirects are limited by Goodreads, getting this error is as good as not obtaining the data | Using a FIFO queue, push this to the bottom of the stack to retry again later. Note that "handle http status" list does not work for this error. |
| HTTP Error 302 indicates a temporary redirection. One of the most notable features that differentiate it from a 301 redirect is that, in the case of 302 redirects, the strength of the SEO is not transferred to a new URL. This is better than 301 empirically as it is still possible to obtain data. | Add error code to "handle http status" list and put a callback to retry on the newly obtained link. |
| HTTP Error 429 is an HTTP response status | Use alternative IP and rest the system for |

| | |
|---|---|
| code that indicates the client application has surpassed its rate limit, or number of requests they can send in a given period of time. | 1-day. |

Note that from the scraping, the following information is attained: ISBN, Book Title, Book Author, Year of Publication, Image URL for the book, book publisher, static book URL, Review(top15), Genres associated, Summary/Synopsis of Book. A real time calculation of review sentiments(section 4.3) and topic modelling(appendix) is performed.



```
ISBN: "0679425608"
Book-Title: "Under the Black Flag: The Romance and the Reality of Life Among the Pi…"
Book-Author: "David Cordingly"
Year-Of-Publication: "1996"
Image-URL-L: "http://images.amazon.com/images/P/0679425608.01.LZZZZZZZ.jpg"
Publisher: "Random House"
URL: "https://www.goodreads.com/book/show/817876.Under_the_Black_Flag"
Review: Array
Review Sentiment: Array
Genre: Array
```

*Figure 1 - Books2 in MongoDB(Summary truncated at bottom)*

Each of the acquired data is subsequently added to MongoDB *real time*, where every book contains a dictionary item of the elements mentioned above. As long as a book is complete, it will append itself to the database. An interesting thing to note is when performing the scraping empirically in initial trials, using lists would cause the scraping to be inconsistent and items from one book may fall into another, highly due to the parallelity of scraping. Therefore only dictionaries are used to store items during scraping.

#### 3.1.3.2 Auto-Updating

We implemented an auto-scraping mechanism that periodically scrapes Goodreads.

This function required tmux installation in the system. Using bash scripting, a detached tmux session can be called to send signals before using the system editor to attach the session. In the script, the python file can be set to run every weekend night and stop running after 8 hours. Continuously run a python script in tmux at a constantly operating system accessed via ssh, and set datetime for the system within the python file(this is not recommended as its running forever, need to account for crashing et cetera). Alternatively, at the constantly running computer accessed via ssh, using Cron Job Scheduler, schedule the job with a shell script(.sh) to run the scraping python file present in the system. A daily update can even be made at 11pm daily: * 23 * * * /root/daily_updates.sh

### 3.1.4 Amazon API

The actual product API that Amazon Web Services (AWS) offers requires us to sign up for a paid AWS membership. Thus, we had to source for free alternatives instead. RapidAPI is a platform which allows free subscription to several API services. Three APIs from that platform were explored and the best source was used to collect the Amazon book review details.

The first source explored was the Amazon Product Scraper [10]. The second source explored was the Amazon Kindle Scraper [11]. Both these sources had a hard limit of 1000 API calls per month. Nevertheless, further exploration was done for these two sources. The API endpoints were able to return details about the book such as the synopsis and top reviews based on book ISBNs. However, they were missing the essential information on the genre of the books. Thus, a third API, Amazon by Gautruche [12] was explored. In addition to the information about the book and its reviews, this API was able to return the category that the books belong to, as well as how it ranks in those categories. Through some simple text manipulation, we were able to obtain the genres through these categories. Thus, this Amazon API was chosen to be the best choice out of the options available in the RapidAPI platform. The scraped results were temporarily saved into a json file and then moved to the MongoDB database described in Section 3.2.

## 3.2 Database - MongoDB

Our database of choice was MongoDB, because it allows for flexibility of document schemas, it has good community support and is easy to access with *pymongo*, a python API package for MongoDB [4].

Our MongoDB database has three main collections: *books*, *genres* and *users*. The *books* collection contains information about each book, such as title, summary and reviews (Figure 7). Each document in the *books* document was acquired via crawling or call from APIs (as described in sections 3.1.3 and 3.1.4).

The *Books* collection consists of 2 sets: Books and Books2. The Books collection contains more than 40000 data and the Books2 collection contains about 16000 data. Books contain mainly reviews and summary, where the word embeddings are trained. Books2 contain the full set of data mentioned in 3.1.3. The database for this set was stored separately from the other databases since this collection maximised(and exceeded slightly) the available free storage in a single Atlus account(500MB) (Figure 10) The auto-updating function will thus remove any duplicates using the ISBN to allow more storage in the update.

The *genres* collection (Figure 9) contains more than 900 unique genres and an embedding matrix **G** of size |G| × d, where |G| is the total number of genres and d is the dimensionality of each embedding. Each embedding vector in **G** represents a genre. This embedding matrix, **G**, is the output of Phrase-BERT and it plays a key role in our semantic search pipeline.

Lastly, the *user* collection contains user credentials and book interactions (Figure 8). Note that the initial set of book interactions are derived from the Kaggle Books Dataset [1]. Subsequently, book interactions are *automatically* updated as users use our application; while users explore new books in our application, the ‘interactions’ field is updated.

## 4 Backend Algorithms

The following sections will elaborate on our backend algorithms. The following sections will describe the implementation of semantic search, collaborative filtering, and sentiment analysis.

## 4.1.1 Semantic Search Algorithm

The semantic search algorithm that we developed consists of two stages: filtering by genre and reranking of book documents (Figures 2.1 and 2.2).

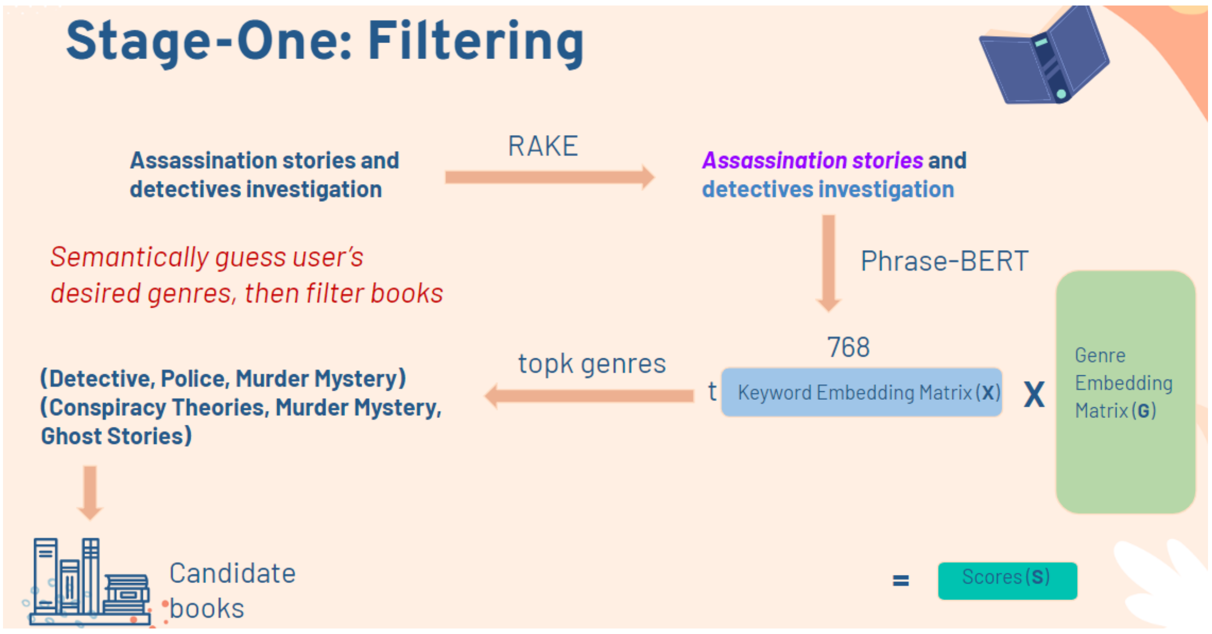

Stage-One: Filtering
Assassination stories and detectives investigation
RAKE
Assassination stories and detectives investigation
Phrase-BERT
Semantically guess user's desired genres, then filter books
768
topk genres
t
Keyword Embedding Matrix (X)
X
Genre Embedding Matrix (G)
(Detective, Police, Murder Mystery)
(Conspiracy Theories, Murder Mystery, Ghost Stories)
Candidate books
=
Scores (S)

*Figure 2.1 - Semantic Search Stage One: Filtering*

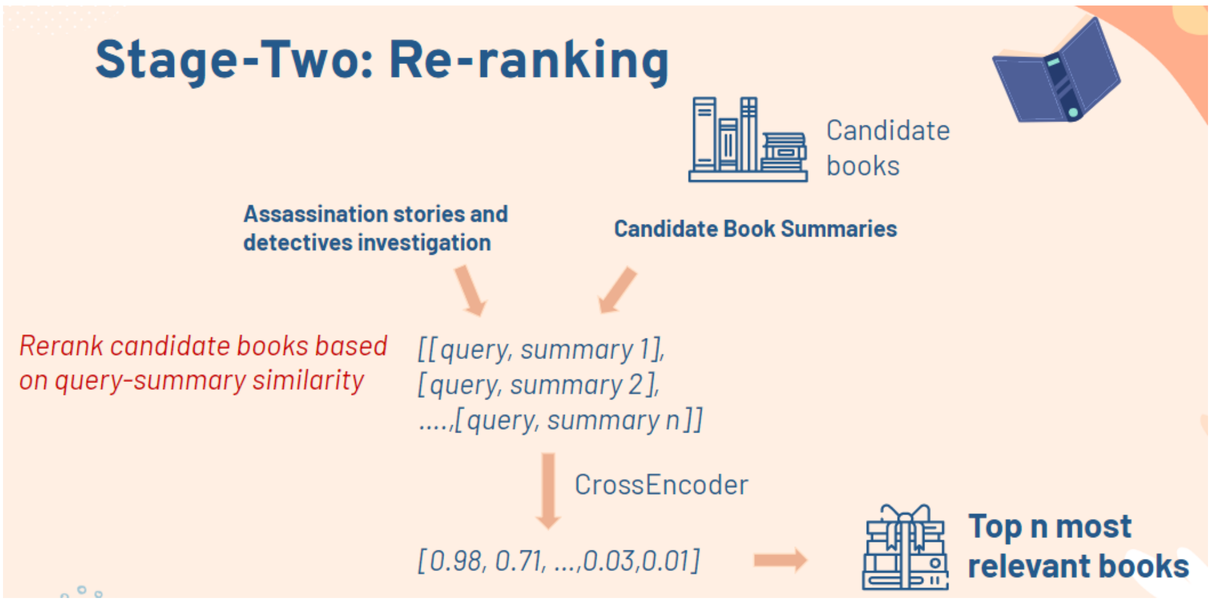


*Figure 2.2 - Semantic Search Stage One: Re-ranking*

In stage-one (Figure 2.1), given a textual query, extract the t most relevant keywords using the Rapid Automatic Keyword Extraction (RAKE) algorithm [2]. Pass each of those keywords through Phrase-BERT [3] to acquire a contextual embedding matrix, **X** of size t × d, where t is at most 3 and d is the embedding dimension 768. From MongoDB, call the genre embedding matrix **G**, of size |G| × d, where |G| is the total number of genres. Perform matrix multiplication to obtain the similarity scores $\mathbf{XG}^T = \mathbf{S}$. Sort the similarity scores **S**, and obtain the **topk** most similar genres for each keyword, **desired_genres,** of size t × **topk**. Using *pymongo* [4], find all books that have at least one genre in all sets in **desired_genres**. In essence, stage-one obtains a candidate set of books, *C*, by filtering via semantically similar genres.

In stage-two (Figure 2.2), the book documents in *C* are ranked using the CrossEncoder class from SentenceTransformers, a Python framework for state-of-the-art sentence, text and image embeddings [7]. In particular, we pair the query with every book summary from *C*, then pass each pair through the CrossEncoder, which returns the similarity scores. Finally, return the top n most semantically similar books.

In summary, a book must fulfil the following criterias to be recommended based on the user's textual query:

- Criteria (1): The book must contain at least one genre from *each and every* set in **desired_genres**
- Criteria (2): The summary of the book must be within the top n most semantically similar to the user's textual query

### 4.1.2 Semantic Search Discussion:

In this section, we discuss the problems we faced, and strengths and weaknesses of our semantic search pipeline.

The key advantage is that our pipeline is efficient. The embedding matrix **G**, can be pre-computed before inference, thus reducing computational requirement. Also, stage-one performs an efficient matrix multiplication $\mathbf{XG}^{\mathbf{T}}$. During our initial developments, we tried storing each genre and genre embedding in separate documents, but this was incredibly inefficient because looping through the MongoDB Cursor turnt out to be time-consuming. Instead, we found that storing the entire genre embedding matrix **G** in one Mongo document is most efficient.

Furthermore, stage-one significantly reduces the number of pairs required for computation in stage two, which significantly reduces inference time because the Transformer-based CrossEncoder is computationally demanding. The hyperparameter **topk** can also be used to control $|C|$ - increasing **topk** increases the number of genres that can be matched, which increases $|C|$. Thus, we have some flexibility in controlling the inference time of our application.

Another advantage is that, our semantic search accounts for lexical dissimilarity. Phrase-BERT produces powerful phrase embeddings such that similar phrases that are lexically dissimilar can have high embedding similarity. In contrast, the original BERT heavily relies on lexical similarity to determine semantic-relatedness [3,8]. For instance, the similarity score between words “werewolves” and “historical fiction” are lexically dissimilar, but are semantically similar because werewolves are a form of historical paranormal fiction genre. Phrase-BERT embeddings tend to give such pairs greater similarity scores, which is our desired outcome - to match the user's query to the right genres.

There are a couple of reasons for extracting the t most relevant keywords from the user's query. Firstly, some genre types are dominating. Initially, we used the entire query as input to Phrase-BERT. However, because certain genre types are much more common, other relevant genres do not make it to **desired_genres**. This may result in the loss of some genres that the user might desire. Also, the reason we kept the value of t to at most 3 is to reduce the impact of irrelevant keywords extracted by RAKE, thus documents in *C* tend to be more relevant.

Some limitations of this approach are that it is highly reliant on genre tags. For instance, books that do not have assigned genres can never be found by a user. There are also cases where no books fulfil criteria (1). In such cases, criteria (1) is relaxed; the book must contain at least one genre from *any* of the sets in **desired_genres**.

Another limitation is that our pipeline is highly reliant on the quality of keyword extraction. We experimented with both RAKE and KeyBERT, eventually deciding on RAKE because it is much faster and produces similar results. However, when the length of a user's query is long, noisy keywords tend to be produced by RAKE, which is detrimental to the quality of the results. This is another reason, the number of keywords extracted per query is capped to at most 3.

Due to the use of Transformers, our pipeline can be computationally demanding. The time and space complexity of the self-attention mechanism in Transformers quadratic to the input sequence length. Since genre embeddings were pre-computed, this isn't a problem for stage-one. However, in stage-two, though the use of CrossEncoder makes our semantically retrieval accurate, the embeddings cannot be pre-computed. One way to mitigate this issue is to reduce **topk**, the number of semantically similar genres for each keyword because it reduces the candidate size, |C|. Nonetheless, we still kept the use of the Transformer-based CrossEncoder in stage-two because it contributes significantly to the quality of recommendations.

### 4.2.1 Personalised Recommendation

To make personalised recommendations, we used a model-based collaborative filtering model, Alternating Least Squares, from a Python Collaborative Filtering library called *implicit* [9]. Given a user's ID, return the **num_rec** number of personalised books (that the user has not interacted with before).

Note that only users that have *at least 10* interactions are able to access this feature. Reason being that collaborative filtering models require sufficient historical interactions in order to produce reliable recommendations.

### 4.2.2 Training of collaborative filtering models

From the user-item interactions data (as mentioned in section 3.1.2), we remove all users and items that do not have at least 10 interactions. Next, split the dataset into training and test sets of ratio 7:3, then convert the training and test sets into matrices of compressed sparse row (CSR) format. Lastly, train collaborative filtering (CF) models on the train CSR matrix and evaluate the performance on the test CSR matrix.

We compared the performances of 3 different CF models namely: alternating least squares, bayesian personalised ranking and logistic matrix factorization. Furthermore, we performed hyperparameter tuning to optimise the performances (Figures 4 and 5). Evaluation results are illustrated in Section 6.

## 4.3 Training of Sentiment models

This section outlines the training on sentiment models that was used to predict sentiment for each review.

The main datasets used for the training are as follows: Amazon[13], Twitter [14][15][16][17][18]. Due to the nature of the neural network model explored, the data was prepared in the following manner: Lemmatization→ tokenized by mapping each word to its unique identifier→ padding.

Lemmatizing was preferred over stemming since lemmas which are based on a canonical dictionary approach are context dependent which improves accuracy. For all the neural network architecture used, the data had to be tokenized and padded as network input was set with an initial length of 70. Compared to the other datasets which contained (-1,0,1) based sentiments, the Amazon dataset implemented binary values of true/false in recommendation, which will be subjected to a binary conversion to become an indicator of the overall sentiment.

Post conversion, Amazon data has an approximate of 200 negative tags out of 5000 samples which is only 4% of the total set. The difference of data between classes may be problematic during the training phase, but this issue will be downplayed with the addition of the other datasets. Comparatively, the data from the other sets have a close to 33% negative sample, an improvement from the earlier 4%. To make the training not be affected too much by a single training set, an equal amount of training data will be randomly sampled from the different sets so that the results yielded are not affected by the number of samples obtained; this will hold for the test set as well. A three-way split with a holdout of 30% will be used since the model selection and true error estimates are to be computed simultaneously. The percentage of the three disjoint sets from the datasets are as follows: Training (0.9*0.7=0.63), Validation (0.1*0.7=0.07), Test (0.3). Stratified sampling is used especially since there is not an equivalent number of samples of false and true values in recommendation.

The maximum sequence length differs across all dataset, where the Amazon reached up to 1559 in length(outlier). The other datasets contain a maximum sequence of less than 70. In consideration of exploding and vanishing gradients (even with clipping), loss of viable data and sparsity, to obtain a richer representation, the maximum sequence length was eventually capped at 70 (Pure models did not perform well with a padding up to1559). The amount of vocabulary to tokenize was also consistently capped at 5000 across datasets.

Traditional methods include implementing N-gram and sliding window approach to create a 1-dimensional convolutional network which utilises the stochastic properties of N-gram while using elements of Markov model for scaling. However, in the neural network created here, neither N-gram of bag-of-words were selected; an embedding layer was used directly after the input layer since they can better embed the tokenized sentences into vector spaces to obtain a richer representation of the data: it places semantically similar inputs close together in the embedding space. Keras was used for the pure models, where variations of the following architecture will be applied: Input layer→ Embedding→ Bidirectional LSTM/GRU→ Final Dense layer with softmax(3 classes) → compilation with categorical cross entropy and Stochastic gradient descent(with momentum+decay). Subsequently, these models are finetuned with kerastuner's hyperband algorithm which performs random sampling while conserving time via optimising best results through killing bad configurations which does not converge well early. In the next phase, the model used by transfer learning implements sentence embedding since word embedding forms an N-dimensional vector for every word whereas sentence embedding can embed phrases and sentences, which makes it stronger.

## 5 UI and Demo

We used Stream-lit to build our UI. There are two ways in which users can use our app: with login (signed in user) and without logging in (guest user).

The first page is the login screen (Figure 12). If the user does not want to login, they can continue as a guest user.

Once the user clicks on 'continue as guest', they will be directed to the main page where they can see a search bar, a sidebar filter for genres and a list of books under the 'Popular Reads' section (Figure 16). The books here are the ones that have the highest user interactions, which is calculated by the greatest number of clicks for the books.

The guest users can search for their desired genres with the sidebar filter for genres (Figure 19). This does a string matching for genres. The filter is a multi-select box which gives suggestions based on what the user types in. The suggested genres are the list of all genres from the total book collection.

The search bar in the main page (Figure 20) filters books using semantic search instead of plain string search. For example, the query "assassination stories and detectives investigating" returns books with relevant genres such as Murder Mystery, Detective etc accurately. Moreover, the genres added via the semantic search are automatically added to the sidebar filter as well.

It is possible to add both the semantic search bar and the sidebar for filtering by genre together to get a combined search output as well: "assassination stories and detectives investigating" in search bar and ‘music’ in sidebar filter tool. Both filters get applied and the results are shown accordingly (Figure 21).

If the user already has an account, they can login to get customised book recommendations. If the user enters the wrong credentials while logging in, he’ll get an error message asking to try again (Figure 13). Taking reference to one of the existing users in the database with the username, ‘*210959’*, the username appears along with the search bar and filtering options. These search functions are similar to the ones mentioned for guest users. The additional component in the personal dashboard is the personalised recommendations. Here, these recommendations differ from user to user and are optimised using collaborative filtering algorithms. (Figure 17).

When the user clicks on the book title under the book cover image, they’ll be brought to the book dashboard page for that particular book (Figure 22). Firstly, the book title, image and details such as author name, publisher and year are displayed. Additionally, the scraped summary and genres for the book are also shown. The bottom of the page shows the top few reviews for the book as well as the overall sentiment for the reviews, as shown by an indicator chart. This gives the user a sense of how positive or negative the overall sentiment is for that particular book. There is a back button to go back to the main menu page.

Furthermore, users who do not have an account can sign up as a new user via the sign up function (Figure 14). If the user tries to sign up with a username  that an existing user already has, he’ll get an error message asking him to try a new username (Figure 15). Since the newly signed up user won’t have any previous book interactions recorded to generate personalised recommendations, his personalised recommendations would be similar to the popular reads section of guest users (Figure 18).

## <u>6 Results and Evaluation</u>

For the following sections, we show quantitative and qualitative evaluations of our algorithms.

### <u>6.1 Semantic Search Results</u>

Based on observation, semanticSearch() returns relevant results in a short amount of time - an average of 4.14 seconds, given >40000 book documents. For instance, given the query “assassination stories and detectives investigating” the top result was *2nd Chance* by James

Patterson. Given the query “Find me storybooks about self growth” the top result was *Love, Medicine and Miracles* by Bernie S. Siegel.

Due to time and difficulty, we do not quantitatively evaluate this pipeline. The best way to evaluate this pipeline is to deploy and allow real users to use this application, following that, obtain feedback by asking users to rate the quality of the results. Another way is to synthetically generate user queries, however there will still be a significant deviation from the true performance of the pipeline, thus we do not pursue this task.

## 6.2 Personalised Recommendation Results

Our personalised recommendation feature is very efficient, with an average computation time of 0.33s.

For recommendation quality-wise, we evaluated our collaborative filtering (CF) models on the unseen test CSR matrix of user-item interactions. The metric used for evaluation was recall@20. In layman terms, given that the CF model recommends 20 items, recall@20 is the fraction of true items/books that were recommended.

We compared 3 different collaborative filtering models namely: alternating least squares, bayesian personalised ranking and logistic matrix factorization. In addition, as a benchmark, we evaluated the performance of a model, “pop”, that only recommends the most popular i.e most interacted, books (Figure 3). In Figure 3, we can observe the clear effectiveness of collaborative filtering over a simple popularity-based recommendation. Alternating least squares model achieved the best performance, thus we utilised this model for our final application.

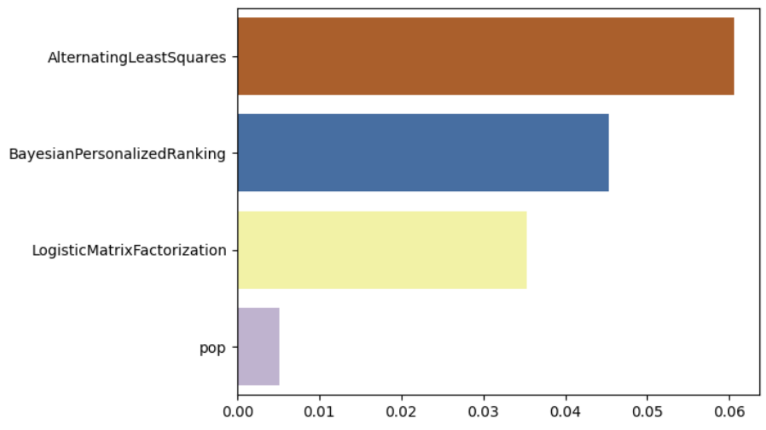


*Figure 3 - Comparison of recommendation models with recall@20*

Lastly, we performed hyperparameter tuning of *alpha* and *factors*, which are the additional weights to positive examples and the embedding dimensions respectively (Figures 4 and 5). For the final CF model, we use the alternating least squares model with alpha value of 5.0 and with 128 latent factors. We chose 128 factors over 256 factors to balance memory space, inference time and performance.

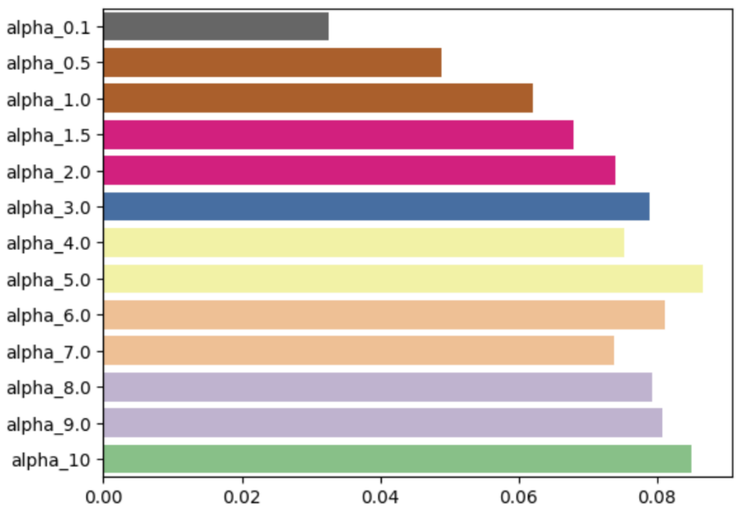


*Figure 4 - Hyperparameter tuning of alpha, weights to give positive examples*

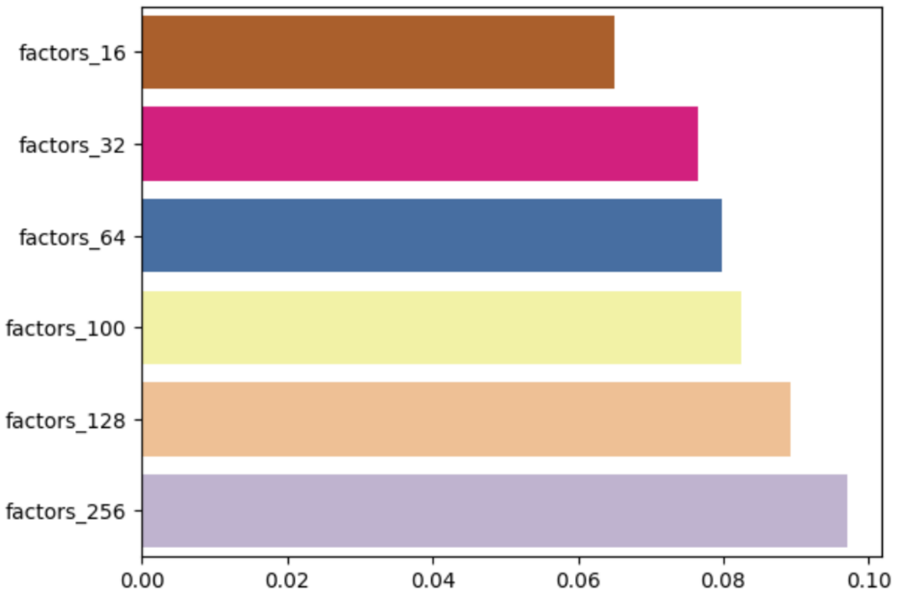


*Figure 5 - Hyperparameter tuning of factors, latent factors to compute*

## 6.3 Sentiment Analysis Quantitative Results

In initial model training, validation loss was constant after the second epoch, which indicated that the model is too strong and/or the model has been stuck in a local minima. This called for several strategies. The model strength was first reduced and the number of regularisation techniques increased. The BiLSTM+GRU layers were reduced to a single BiLSTM layer with reduced neurons. The convolution layer remained and dropouts were added to allow better generalisation and reduce overfitting. Weight penalties were added to enable better back-propagation via increasing cost penalty. These strategies resulted in a model with a better loss curve and higher overall accuracy.

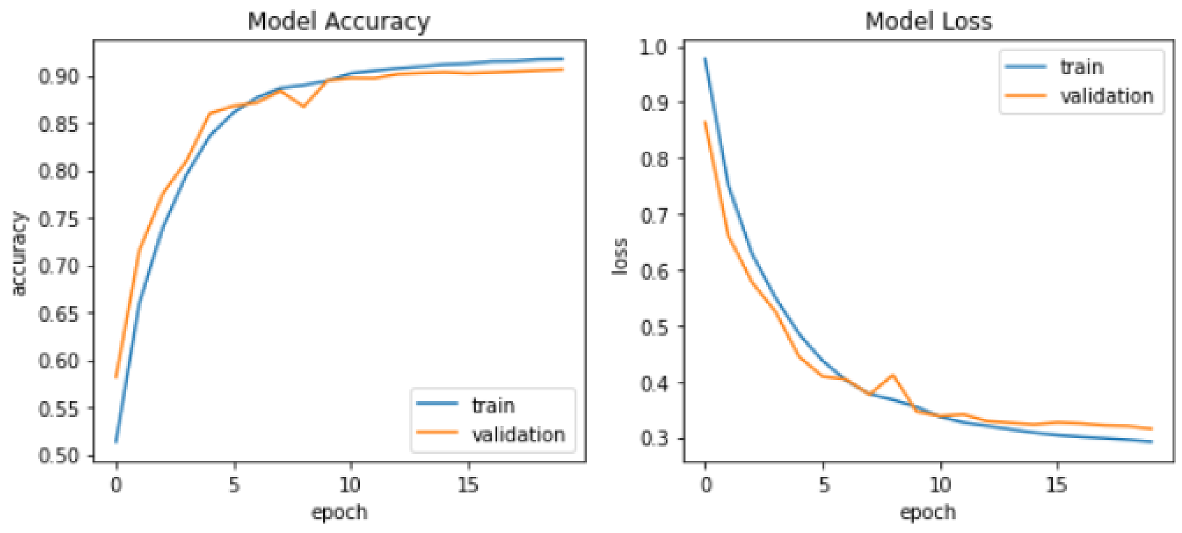


*Figure 23 - Train loss and accuracy*

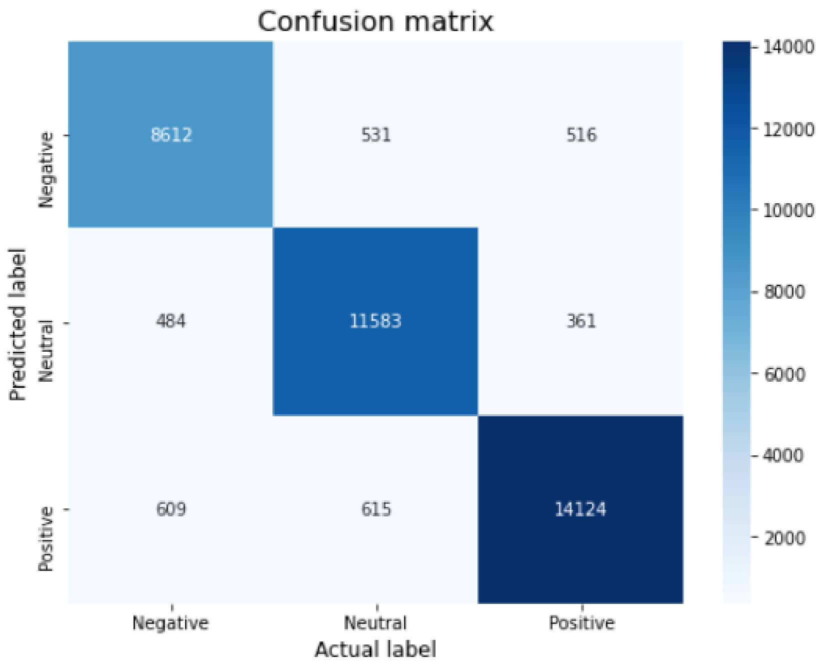
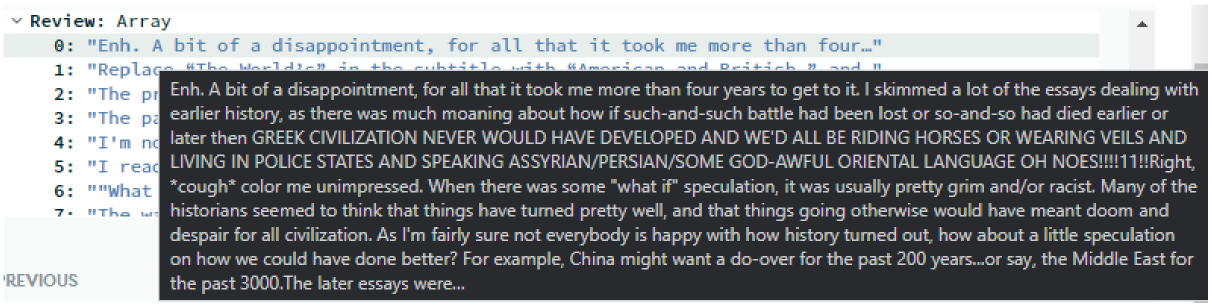


*Figure 24 - Confusion matrix of test results*

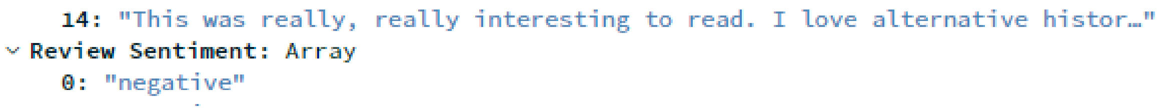
The model was able to work well with the prediction when went through manually. For instance, in a randomly sampled book(ISBN: 0425176428), the first review was of negative sentiment and the last(14th) review had positive sentiment. This resulted in a correct classification of sentiment where review 0 had negative score and review 14 had positive score.



```
⌄ Review: Array
    0: "Enh. A bit of a disappointment, for all that it took me more than four…"
    1: "Replace "The World's" in the subtitle with "American and British," and…"
    2: "The pr
    3: "The pa
    4: "I'm nc
    5: "I reac
    6: ""What
    7: "The w
```

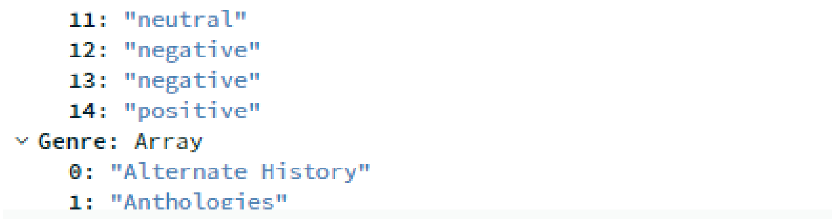
Enh. A bit of a disappointment, for all that it took me more than four years to get to it. I skimmed a lot of the essays dealing with earlier history, as there was much moaning about how if such-and-such battle had been lost or so-and-so had died earlier or later then GREEK CIVILIZATION NEVER WOULD HAVE DEVELOPED AND WE'D ALL BE RIDING HORSES OR WEARING VEILS AND LIVING IN POLICE STATES AND SPEAKING ASSYRIAN/PERSIAN/SOME GOD-AWFUL ORIENTAL LANGUAGE OH NOES!!!!11!!Right, *cough* color me unimpressed. When there was some "what if" speculation, it was usually pretty grim and/or racist. Many of the historians seemed to think that things have turned pretty well, and that things going otherwise would have meant doom and despair for all civilization. As I'm fairly sure not everybody is happy with how history turned out, how about a little speculation on how we could have done better? For example, China might want a do-over for the past 200 years...or say, the Middle East for the past 3000.The later essays were...

*Figure 25 - Review 0*



```
    14: "This was really, really interesting to read. I love alternative histor…"
  ⌄ Review Sentiment: Array
    0: "negative"
```

*Figure 26 - Review 14 and review 0 sentiment score*



```
    11: "neutral"
    12: "negative"
    13: "negative"
    14: "positive"
 ⌄ Genre: Array
    0: "Alternate History"
    1: "Anthologies"
```

*Figure 27 - Review 11-14 sentiment score*

## 6.5 Backend Algorithm Discussion

In this section, we further elaborate on some of the pros and cons of our pipeline, as well as proposing some areas for improvement.

As mentioned in section 4.1.2, our semantic search pipeline is designed to be efficient yet performant. Another advantage is that users do not have to search through a long list of genres, because our algorithm essentially guesses the user's desired genres based on text. At the time of writing, our database has more than 900 genres, thus this feature provides a lot of utility for the user.

Our personalised book recommendation algorithm is dynamic and fast. Due to the use of CSR matrices, the CF recommendation models are very compact and efficient. Furthermore, our application constantly records the interactions users perform, thus the CF models can be retrained on new interactional data.

One downside to the CF models is the problem of cold start users. New users or users without sufficient historical interactions cannot receive personalised recommendations. A possible solution is to obtain more user information like demographic data, which can be used to cluster similar users together, then recommending items based on clustering results.

As mentioned in section 4.1.2, our semantic search pipeline is highly reliant on genre tags. In cases where books do not have genres tagged or are only partially tagged, they may not be retrieved by the pipeline. It is possible to automatically assign genre tags for books that have not been assigned genres by computing the semantic similarity between every genre and the book summary.

Another downside is the use of CrossEncoder to compute semantic similarity. Due to O($n^2$) complexity(n is the number of input tokens), stage-two of our semantic search algorithm is not scalable. Future work could integrate lexical-based search for stage-two instead.

## 7 Conclusion

In this project, we developed a book recommendation application that can perform semantic search, personalised recommendations and review sentiments. We implemented our front-end user interface using stream-lit. For the backend algorithms, we utilised collaborative filtering models, and language models. Our application is dynamic as user interactions with the application are captured for future use, and book data can be automatically scraped periodically.

The demo our our final application can be found in this link: Book Recommendation System Demo Video

## 8 Contributions

| Name | Contributions |
|---|---|
| Tan Ching Fhen | ● Developed and implemented semantic search algorithm<br>● Implemented personalised recommendation algorithm<br>● Trained, evaluated and optimised recommendation algorithms<br>● Implemented product quantization for search (appendix)<br>● Managed *user* and *genre* collections in MongoBD<br>● Implemented PriorityQueue to assist semantic retrieval tasks<br>● chingfhen/Semantic-and-Collaborative-FIltering-Recommendation (github.com) |
| Sim Shi Qian | ● Developed and Implemented Goodreads Scraping<br>● Develop real time data insertion framework and an auto-update function(Goodreads→MongoDB)<br>● Managed *books* collection in MongoDB (~70000)<br>● Developed, trained, optimised and implemented review based sentiment analysis<br>● Trained and implemented topic modelling (Appendix)<br>● https://github.com/Veracitea/GoodreadsScraping<br>● https://github.com/Veracitea/SentimentAnalysis_TopicModel |
| Kusalavan Kirubhaharini | ● Explored and tried out different API sources<br>● Implemented Amazon API from RapidAPI [12]<br>● Created functions to get book details and review information respectively from the API<br>● Updated *books* collection in MongoDB from RapidAPI<br>● Code for API implementation: https://github.com/kirubhaharini/Get-Book-Information---Amazon-API<br>● Worked on developing Streamlit UI<br>● Code for Streamlit UI: https://github.com/kirubhaharini/CZ4125-Data-Products-App<br>● Demo Video |
| Inbanathan Vaishnavi | ● Explored SBERT network and tested it's suitability for this project<br>● Created functions and used pymongo queries to read information from *user*, *genre* and *books* collection<br>● Ideated and developed Streamlit UI<br>● Worked on the scripting and creating the flow of the demo video<br>● Code for Streamlit UI: https://github.com/kirubhaharini/CZ4125-Data-Products-App |

## References

[1] Books Dataset - Kaggle. Books Dataset | Kaggle

[2] Rapid Automatic Keyword Extraction. rake-nltk · PyPI

[3] Phrase-BERT. [2109.06304] Phrase-BERT: Improved Phrase Embeddings from BERT with an Application to Corpus Exploration (arxiv.org)

[4] Python API for MongoDB. PyMongo 4.3.2 Documentation — PyMongo 4.3.2 documentation

[5] Goodreads | Meet your next favorite book

[6] Amazon.com: Books

[7] Python Package for Transformers. SentenceTransformers Documentation — Sentence-Transformers documentation (sbert.net)

[8] Original BERT. [1810.04805] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (arxiv.org)

[9] Python Package for Collaborative Filtering. Implicit — Implicit 0.6.1 documentation (benfred.github.io)

[10] Amazon Product Scraper. https://rapidapi.com/animohost/api/amazon-product-scraper5

[11] Amazon Kindle Scraper. https://rapidapi.com/Lagrafabdessalem/api/amazon-kindle-scraper

[12] Amazon by Gautruche API. https://rapidapi.com/b2g.corporation/api/amazon24

[13] Amazon dataset https://data.world/datafiniti/consumer-reviews-of-amazon-products

[14] “Covid 19 Indian Sentiments on Covid19 and Lockdown  | Kaggle.” Kaggle: Your Machine Learning and Data Science Community, https://www.kaggle.com/datasets/surajkum1198/twitterdata.

[15] “Twitter Tweets Sentiment Dataset | Kaggle.” Kaggle: Your Machine Learning and Data Science Community, https://www.kaggle.com/datasets/yasserh/twitter-tweets-sentiment-dataset.

[16] Kaggle: Your Machine Learning and Data Science Community, https://www.kaggle.com/datasets/surajkum1198/twitterdata.

[17] “Twitter and Reddit Sentimental Analysis Dataset | Kaggle.” Kaggle: Your Machine Learning and Data Science Community, https://www.kaggle.com/datasets/cosmos98/twitter-and-reddit-sentimental-analysis-dataset.

[18] “Apple_twitter_sentiment_texts | Kaggle.” Kaggle: Your Machine Learning and Data Science Community, https://www.kaggle.com/datasets/seriousran/appletwittersentimenttexts.

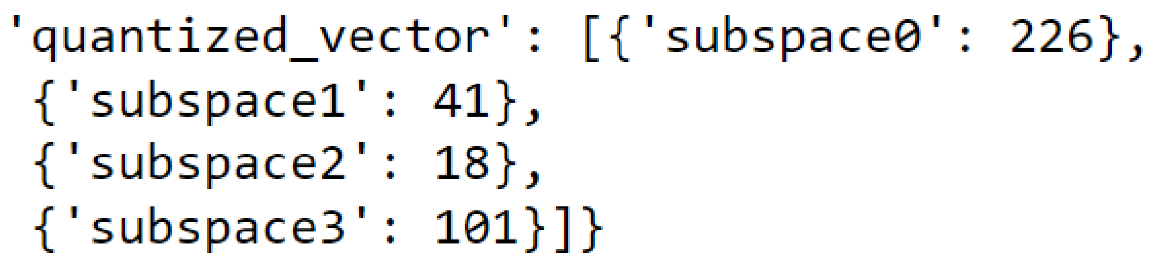
## Appendix

In this section we append all works that did not end up in the final application. Images referenced in the report are also appended here.

## Product Quantization

In previous sections, we mentioned that the semantic search pipeline is overly reliant on quality genre tags. This can lead to poor results in some cases when book documents do not have genre tags or are not properly tagged. We attempted to mitigate this issue by using an additional algorithm that does not rely on genre tags -  product quantization.

For every book document, acquire its contextualised document embedding using Transformers and then perform product quantization on these embeddings. This means that every book document was also assigned with a set of quantized ids (Figure 6). Given a text query, perform the same processing steps, then retrieve books that contain at least one matching quantized id. The problem with this approach is that obtaining quality quantized ids is a non-trivial task. Length of book summaries are usually long, which makes it difficult to generate high quality sentence embeddings. Furthermore, the clustering step performed during product quantization may not be optimal. Therefore, we did not include this algorithm in our final application.



```
'quantized_vector': [{'subspace0': 226},
 {'subspace1': 41},
 {'subspace2': 18},
 {'subspace3': 101}]}
```

*Figure 6 - quantized ids of a book document*

## Review Keyword Extraction

During our development phase, we also attempted to generate more keyword tags, in addition to genre tags. The idea was to produce a wider variety of tags, which could be used in stage-one. In particular, the reviews contain a rich source of keywords such as “magnificent book” or “thrilling book”, so we extracted keywords using RAKE and KeyBERT from the reviews and summaries. However, it turnt out that keyword extraction generates a lot of noisy keywords, which is detrimental to the performance of our pipeline. Thus, we did not include this in our pipeline.

## Semantic Search Version 1

Initially, our stage-one pipeline was quite different. Instead of pre-computing genre embeddings, we pre-computed the summary embeddings. This turnt out to be incredibly time-consuming because we will need to loop through 40000+ summary embeddings, compute the dot product similarity, then update a priority queue to keep track of the most similar books. We improvised and adapted this to the current stage-one algorithm - leveraging genre tags for semantic retrieval.

## Topic Modelling

The motive of performing topic modelling was to apply it to every review in a book such that a book could contain additional information of the topics which is derived from the reader's comments. This would allow potential filtering or search functions to be created from only reviews based on topic keywords. However, after applying them to the data in real time(during scraping) it was found that there are some issues: firstly, most of the books do not have any results due to having little to no comments or having comments which are unrelated, causing keywords to not be picked up (sparse matrix formed). Even if there are topics, due to insufficient length of the comments, the model was only able to pick up topic -1 which represents the stopwords, which is not useful. Due to the limited amount of topics picked up for all books, where most returned empty information, this implementation was not included in the final GUI.

BERT-based topic modelling was done successfully whereas Top2Vec topic-modelling was attempted but failed (Figure 11). The BERT-topic modellers are state-of-the-art token-based embeddings(as opposed to sentence-based embeddings). This means that they are pre trained with a defined amount of lexicon(a very big, generalised set) to be able to generate embeddings for given words or phrases. By using the pretrained embeddings as initialization, the fine tuning would only cause the samples to work better on the dataset that we are looking at. Embedding spaces are stronger than traditional Bag-of-words approach since they can better embed the tokenized sentences into vector spaces to obtain a richer representation of the data where semantically similar inputs will be generated closely in the embedding space. As the name BERT suggests(Bi-directional Encoder representation), the embeddings are trained with transformers, which uses a multi-headed attention mechanism– this means that it does not have the flaws of RNNs used in the previous section, which has concerns over elongated timespaces on gradient effects. With strong embeddings, the model then creates dense clusters to allow easily interpreted topics via including important keywords in the topic descriptions, where each topic-term matrix are calculated through c-TF-IDF:

**c-TF-IDF**

For a term **x** within class **c**:

$$W_{x,c} = \|tf_{x,c}\| \times \log\left(1 + \frac{A}{f_x}\right)$$

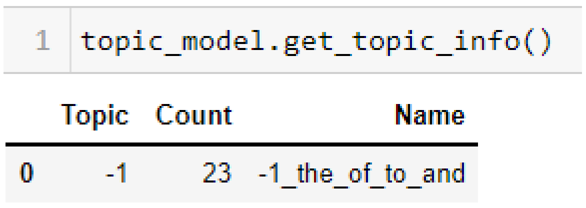
$tf_{x,c}$ = frequency of word **x** in class **c**

$f_x$ = frequency of word **x** across all classes

**A** = average number of words per class

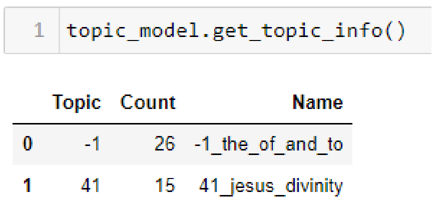
Comparatively, top2vec implements joint embeddings which allows word-embeddings to be in the same space as document embeddings. This is slightly different from the token-embeddings that were used by BERTopic. Initially, word2vec was attempted since it was the one mentioned in the discussions during meetings, however there were some difficulties: even after pip installation, paths do not match and manual installation of universal sentence encoder from google( https://tfhub.dev/google/universal-sentence-encoder/4) did not work.

However, as aforementioned, upon usage, the BERTopic was extremely limited due to the lack of useful word in reviews:



```
topic_model.get_topic_info()
```

|   | Topic | Count | Name |
|---|---|---|---|
| 0 | -1 | 23 | -1_the_of_to_and |



```
topic_model.get_topic_info()
```

|   | Topic | Count | Name |
|---|---|---|---|
| 0 | -1 | 26 | -1_the_of_and_to |
| 1 | 41 | 15 | 41_jesus_divinity |

*Figure 25 - Examples from BERTopic implementation on the Book reviews*

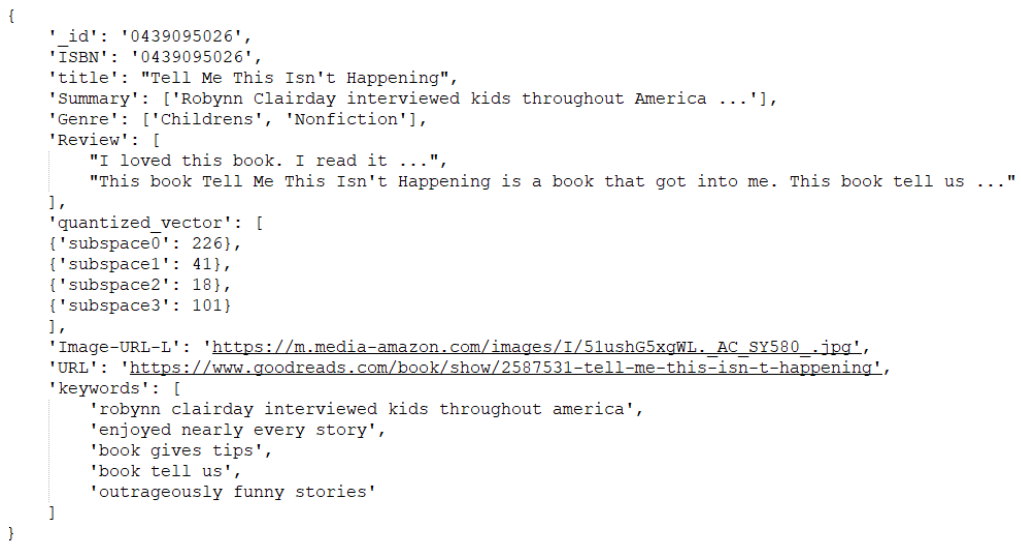
Majority of the reviews were not long enough for a thorough fit, which gives back an error since there are no repeating words to form a viable identity matrix:

```
ValueError: zero-size array to reduction operation maximum which has no identity
```

Even if the reviews are long enough, most contain only the simple words, which falls under class -1. Only occasionally, with biblical books, there are more chances for other words to come up. Therefore, it was not useful due to its sparse occurrences in the dataset and were thus abandoned in the user interface.

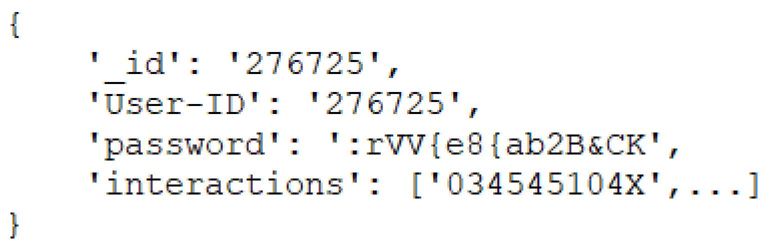
---

## **<u>Images and Figures</u>**



```
{
    '_id': '0439095026',
    'ISBN': '0439095026',
    'title': "Tell Me This Isn't Happening",
    'Summary': ['Robynn Clairday interviewed kids throughout America ...'],
    'Genre': ['Childrens', 'Nonfiction'],
    'Review': [
        "I loved this book. I read it ...",
        "This book Tell Me This Isn't Happening is a book that got into me. This book tell us ..."
    ],
    'quantized_vector': [
    {'subspace0': 226},
    {'subspace1': 41},
    {'subspace2': 18},
    {'subspace3': 101}
    ],
    'Image-URL-L': 'https://m.media-amazon.com/images/I/51ushG5xgWL._AC_SY580_.jpg',
    'URL': 'https://www.goodreads.com/book/show/2587531-tell-me-this-isn-t-happening',
    'keywords': [
        'robynn clairday interviewed kids throughout america',
        'enjoyed nearly every story',
        'book gives tips',
        'book tell us',
        'outrageously funny stories'
    ]
}
```

*Figure 7 - a document in the MongoDB book collection*



```
{
    '_id': '276725',
    'User-ID': '276725',
    'password': ':rVV{e8{ab2B&CK',
    'interactions': ['034545104X',...]
}
```

*Figure 8 - document in MongoDB user collection*

```
{
'_id': ObjectId('63550b3400bedcfac4b69560'),
'genre': ['Star Wars','Romanticism',..., 'Theology','Humanities']
'embedding':
    [[0.4147, 0.7237,...],
    ...,
    [-0.4584,-0.3833,...]]
}
```

*Figure 9 - a document in MongoDB genre collection*

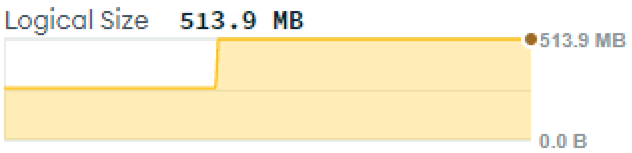
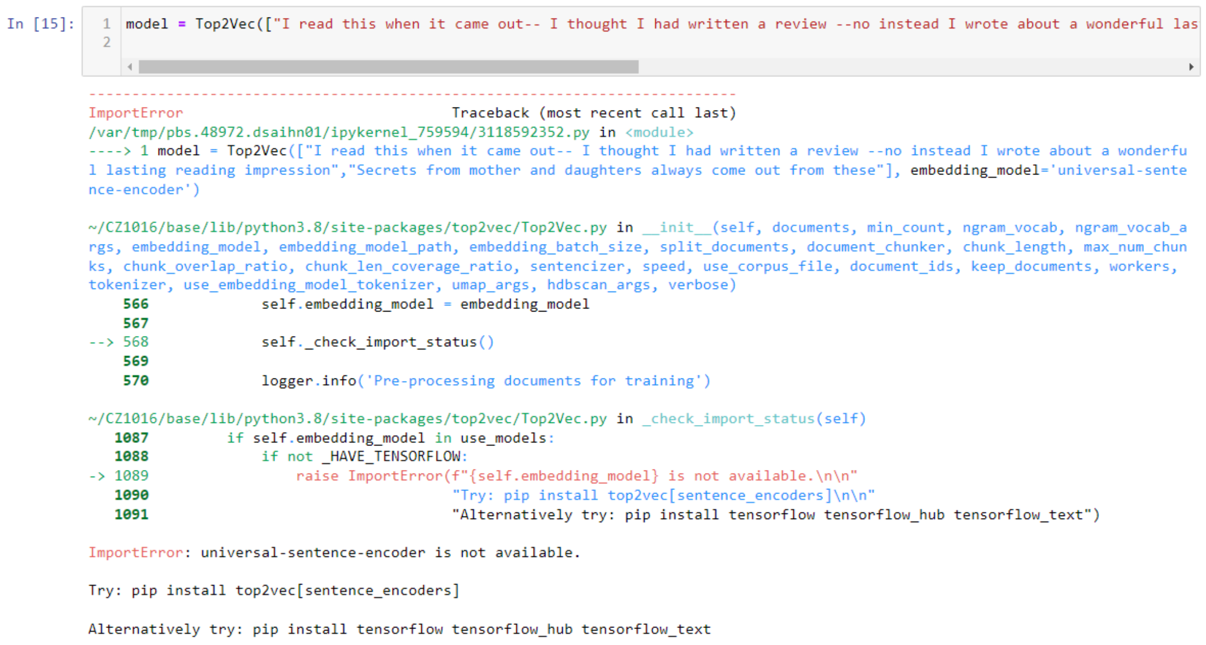
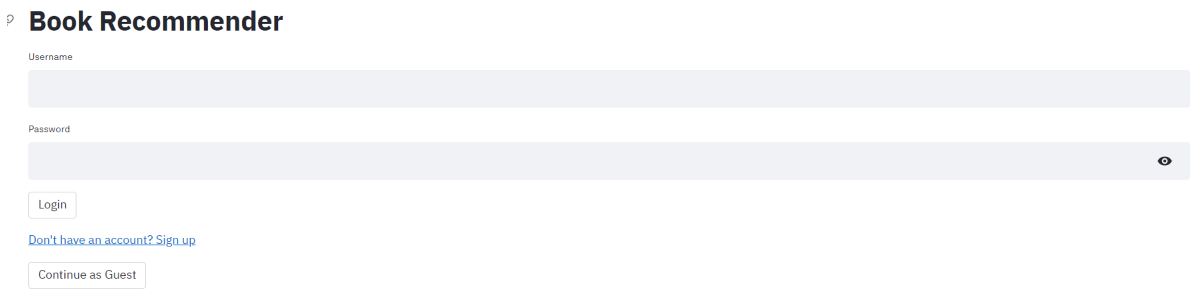


*Figure 10 - Books space in MongoDB*



```
In [15]:  model = Top2Vec(["I read this when it came out-- I thought I had written a review --no instead I wrote about a wonderful las

---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
/var/tmp/pbs.48972.dsaihn01/ipykernel_759594/3118592352.py in <module>
----> 1 model = Top2Vec(["I read this when it came out-- I thought I had written a review --no instead I wrote about a wonderfu
l lasting reading impression","Secrets from mother and daughters always come out from these"], embedding_model='universal-sente
nce-encoder')

~/CZ1016/base/lib/python3.8/site-packages/top2vec/Top2Vec.py in __init__(self, documents, min_count, ngram_vocab, ngram_vocab_a
rgs, embedding_model, embedding_model_path, embedding_batch_size, split_documents, document_chunker, chunk_length, max_num_chun
ks, chunk_overlap_ratio, chunk_len_coverage_ratio, sentencizer, speed, use_corpus_file, document_ids, keep_documents, workers,
tokenizer, use_embedding_model_tokenizer, umap_args, hdbscan_args, verbose)
    566             self.embedding_model = embedding_model
    567 
--> 568             self._check_import_status()
    569 
    570             logger.info('Pre-processing documents for training')

~/CZ1016/base/lib/python3.8/site-packages/top2vec/Top2Vec.py in _check_import_status(self)
   1087         if self.embedding_model in use_models:
   1088             if not _HAVE_TENSORFLOW:
-> 1089                 raise ImportError(f"{self.embedding_model} is not available.\n\n"
   1090                                   "Try: pip install top2vec[sentence_encoders]\n\n"
   1091                                   "Alternatively try: pip install tensorflow tensorflow_hub tensorflow_text")

ImportError: universal-sentence-encoder is not available.

Try: pip install top2vec[sentence_encoders]

Alternatively try: pip install tensorflow tensorflow_hub tensorflow_text
```

*Figure 11 - Error with Top2Vec*



**Book Recommender**

Username

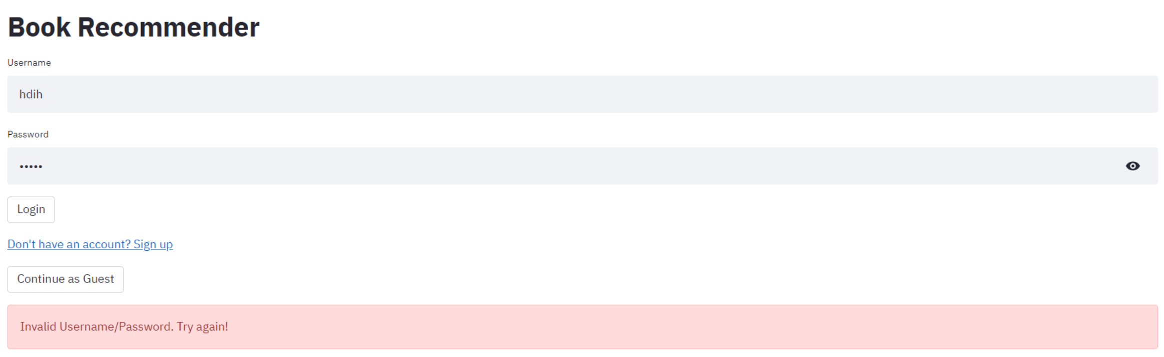
Password

Login

Don't have an account? Sign up

Continue as Guest

*Figure 12 - Login page*


Book Recommender

Username

hdih

Password

•••••

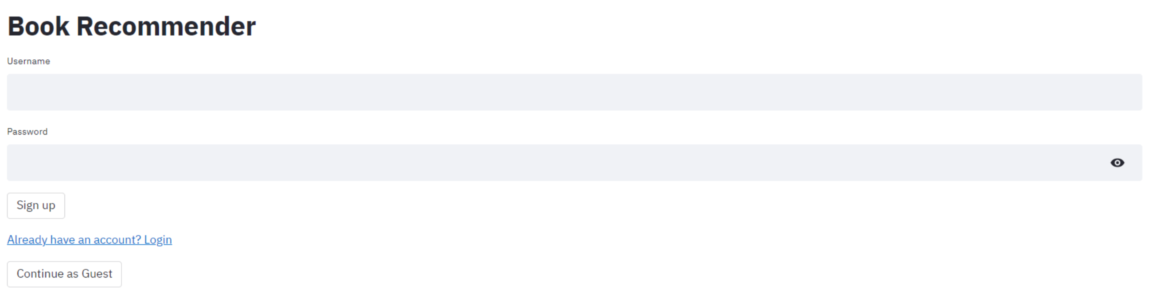
Login

Don't have an account? Sign up

Continue as Guest

Invalid Username/Password. Try again!

*Figure 13 - Invalid credentials at login page*


Book Recommender

Username

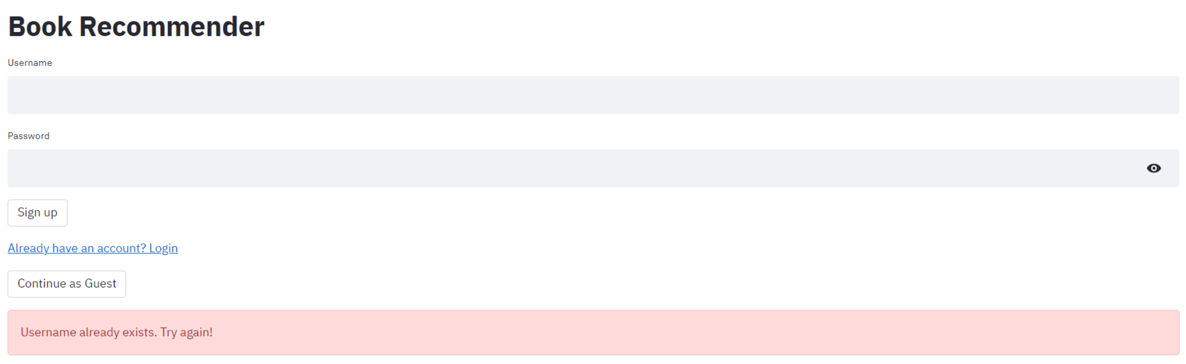
Password

Sign up

Already have an account? Login

Continue as Guest

*Figure 14 - Sign up page*


Book Recommender

Username

Password

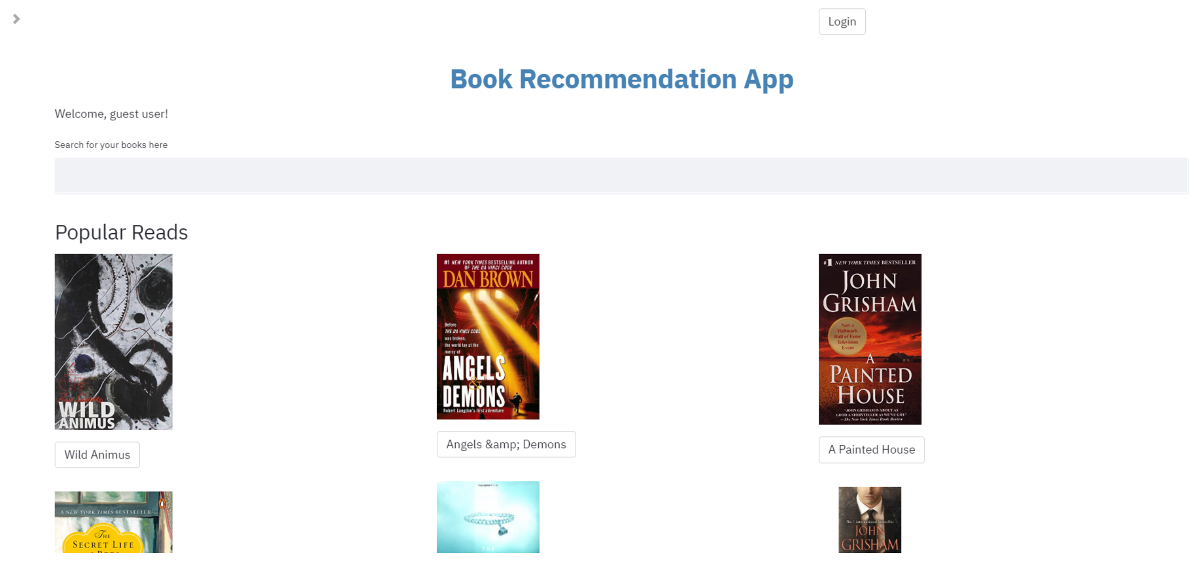
Sign up

Already have an account? Login

Continue as Guest

Username already exists. Try again!

*Figure 15 - Error message for signing up using an existing username*



Login

Book Recommendation App

Welcome, guest user!

Search for your books here

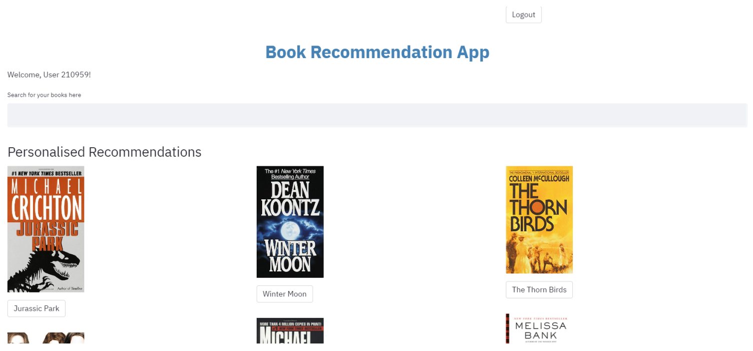
Popular Reads

Wild Animus

Angels &amp; Demons

A Painted House

Figure 16 - Guest user main page showing popular reads



Logout

Book Recommendation App

Welcome, User 210959!

Search for your books here

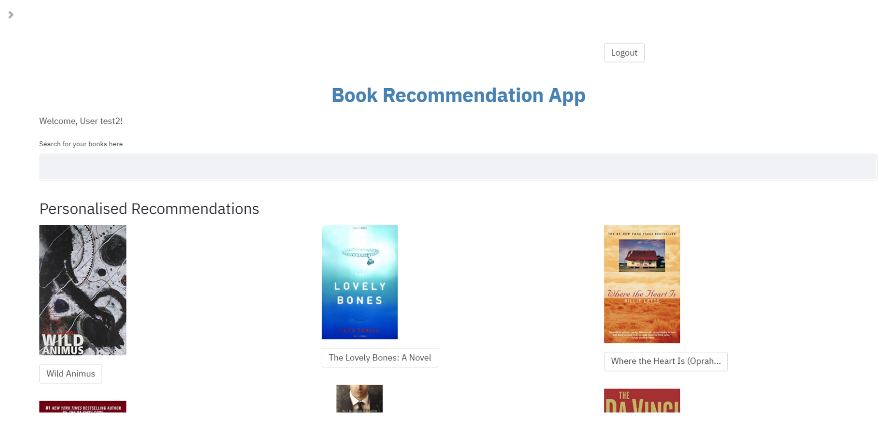
Personalised Recommendations

Jurassic Park

Winter Moon

The Thorn Birds

Figure 17 - Logged in user page showing personalised recommendations



Logout

Book Recommendation App

Welcome, User test2!

Search for your books here

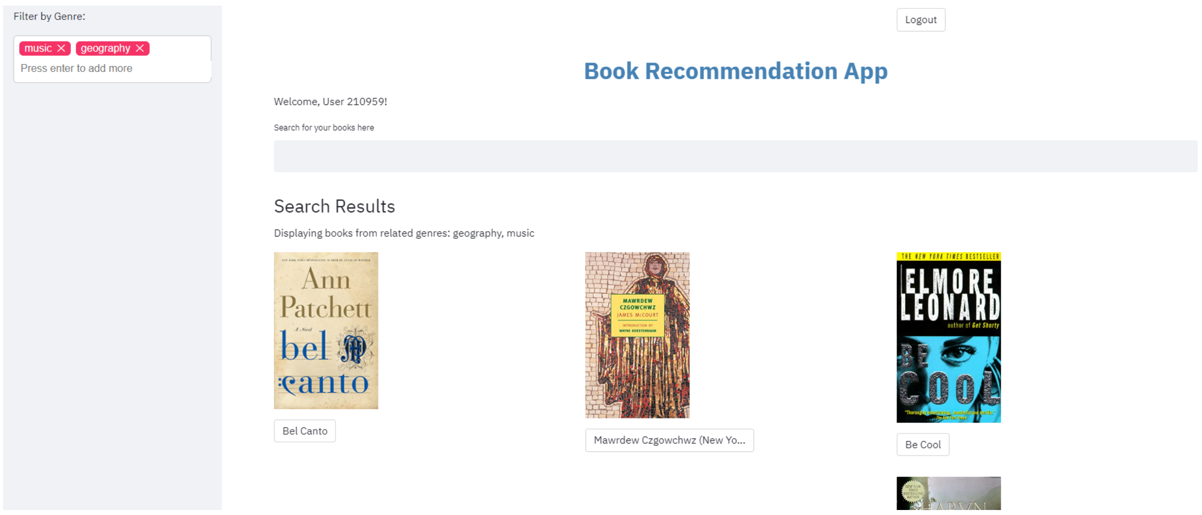
Personalised Recommendations

Wild Animus

The Lovely Bones: A Novel

Where the Heart Is (Oprah...

Figure 18 - First time logged in user page showing personalised recommendations which are the same as popular reads


Filter by Genre:

music geography

Press enter to add more

Logout

Book Recommendation App

Welcome, User 210959!

Search for your books here

Search Results

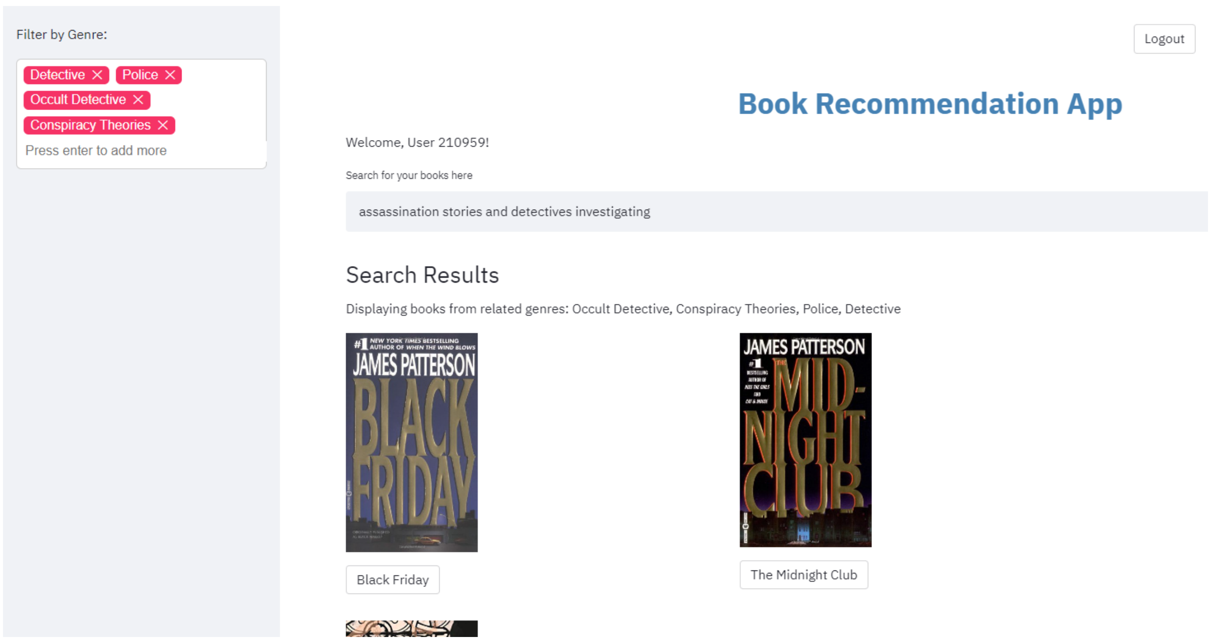
Displaying books from related genres: geography, music

Bel Canto

Mawrdew Czgowchwz (New Yo...

Be Cool

*Figure 19 - Genre string matching filter in sidebar*


Filter by Genre:

Detective Police Occult Detective Conspiracy Theories

Press enter to add more

Logout

Book Recommendation App

Welcome, User 210959!

Search for your books here

assassination stories and detectives investigating

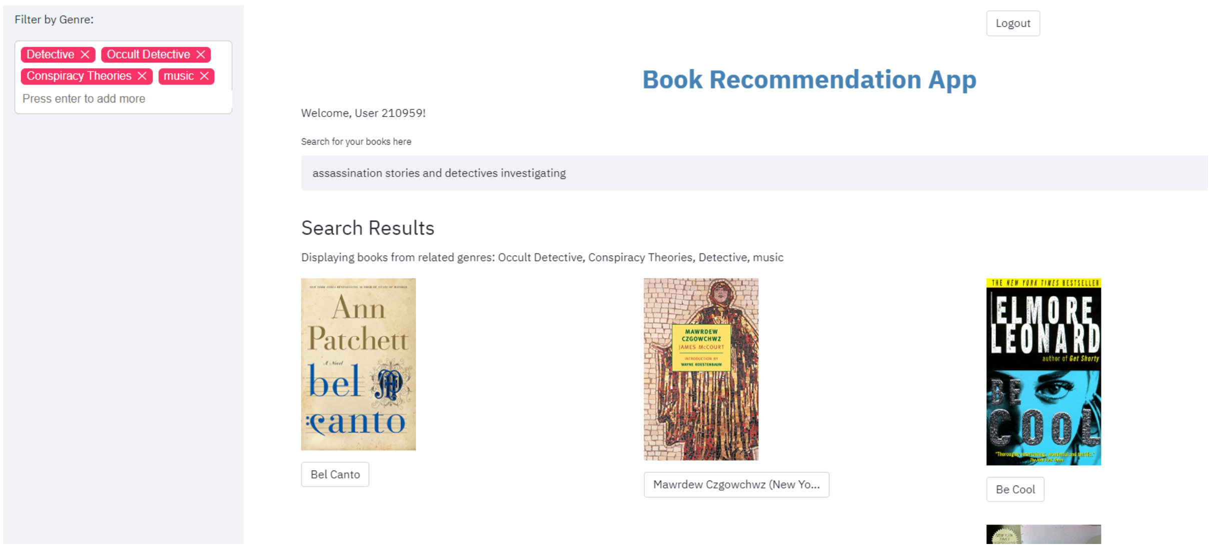
Search Results

Displaying books from related genres: Occult Detective, Conspiracy Theories, Police, Detective

Black Friday

The Midnight Club

*Figure 20 - Genre semantic search (‘filter by genre’ filters applied automatically)*



Filter by Genre:

Detective ✕ Occult Detective ✕ Conspiracy Theories ✕ music ✕

Press enter to add more

Logout

Book Recommendation App

Welcome, User 210959!

Search for your books here

assassination stories and detectives investigating

Search Results

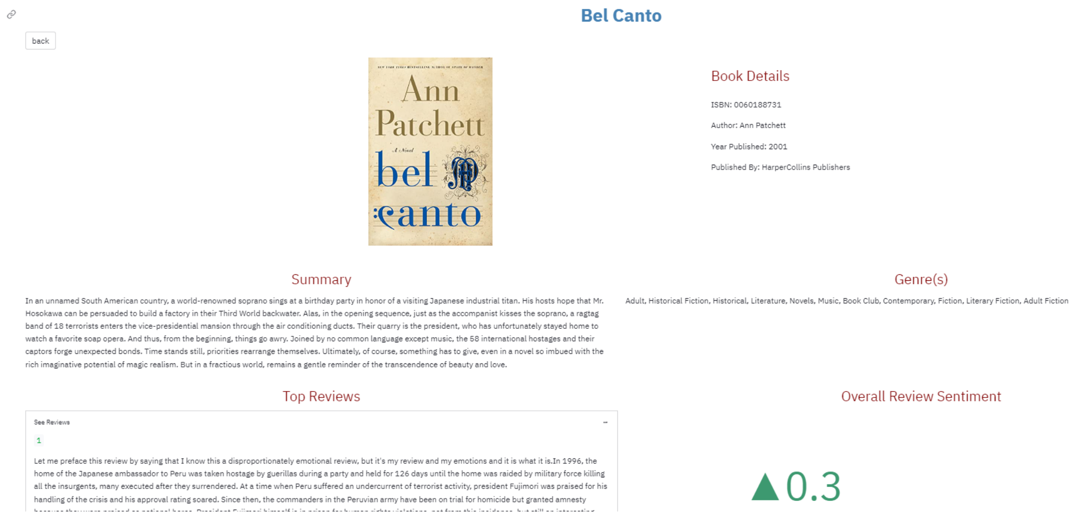
Displaying books from related genres: Occult Detective, Conspiracy Theories, Detective, music

Bel Canto

Mawrdew Czgowchwz (New Yo...

Be Cool

*Figure 21 - Genre semantic search with additional filters in 'filter by genre'*



Bel Canto

back

Book Details

ISBN: 0060188731

Author: Ann Patchett

Year Published: 2001

Published By: HarperCollins Publishers

Summary

In an unnamed South American country, a world-renowned soprano sings at a birthday party in honor of a visiting Japanese industrial titan. His hosts hope that Mr. Hosokawa can be persuaded to build a factory in their Third World backwater. Alas, in the opening sequence, just as the accompanist kisses the soprano, a ragtag band of 18 terrorists enters the vice-presidential mansion through the air conditioning ducts. Their quarry is the president, who has unfortunately stayed home to watch a favorite soap opera. And thus, from the beginning, things go awry. Joined by no common language except music, the 58 international hostages and their captors forge unexpected bonds. Time stands still, priorities rearrange themselves. Ultimately, of course, something has to give, even in a novel so imbued with the rich imaginative potential of magic realism. But in a fractious world, remains a gentle reminder of the transcendence of beauty and love.

Genre(s)

Adult, Historical Fiction, Historical, Literature, Novels, Music, Book Club, Contemporary, Fiction, Literary Fiction, Adult Fiction

Top Reviews

See Reviews

1

Let me preface this review by saying that I know this a disproportionately emotional review, but it's my review and my emotions and it is what it is.In 1996, the home of the Japanese ambassador to Peru was taken hostage by guerillas during a party and held for 126 days until the home was raided by military force killing all the insurgents, many executed after they surrendered. At a time when Peru suffered an undercurrent of terrorist activity, president Fujimori was praised for his handling of the crisis and his approval rating soared. Since then, the commanders in the Peruvian army have been on trial for homicide but granted amnesty

Overall Review Sentiment

▲ 0.3

*Figure 22 - Books dashboard page showing all the relevant details of the book selected*