Data301 Final Project: Recommender System

Evan Richards-Ward
11363270

Summary

The recommender system uses the Goodreads dataset of reviews and books to construct a recommendation for an input user of books that similar users have reviewed. In contrast, the input user has not reviewed the book. The algorithm is similar to the Netflix recommender which uses *cosine similarity* to compute how close users are together. The output is a set of book recommendations (book IDs or, using the book title function, book titles) where the amount can be specified. The result should be able to perform optimally over large data amounts, including the full Goodreads dataset of reviews. Once this is achieved the optimisation of the algorithm and methods can be done when the output is correct. Data handling and data accessing methods may want to be optimised to handle the large amounts of data being accessed, as this is a major bottleneck for the recommender.

Introduction

The research question is: *"Given a database of reviews and books, can new books be recommended to those who have reviewed a set of books that they might be interested in?"*

The datasets used are the Goodreads reviews and the Goodreads book data. These datasets are used to recommend a group of book titles, given a user's reviews. The book titles recommended will be books that similar users have reviewed, but not one that the user has reviewed. The two datasets are linked together by the *book_id*, which is used to identify the titles once the review database has been processed into the recommended book ids. The data used within the review data is: *user_id, book_id and rating*. The data used within the book data is: *book_title* and *book_id*.

I read a lot and knowing how the recommendations for books could be done was an interesting topic.

Methods/ Process

The process for creating a recommendation makes use of five functions. They are: *create_book_array()*, *cosine_similarity()*, *closest_users()*, *book_recommendations()* and *get_book_titles()*. The data gets loaded into two Dask data frames, where one is the **review data** and the other is the **book data**. The data is filtered for the following variables: *user_id, book_id* and *rating* in **review data** and *book_title* and *book_id* in **book data**. The data is then

cleaned, removing commas and extra whitespace. It is here that the data is partitioned into different amounts depending on how much was being tested/used at the time.

The flow of the data is as follows: the review data gets folded to make each *user_id* store with all of the reviews that user had done. The *create_book_array()* function is then mapped onto each entry and a Boolean array is created. These arrays are used to compute the cosine similarity through *cosine_similarity()* through the *closest_users()*. The reviews and a user are passed into the *closest_users()* function in order to retrieve the closest users by mapping the *cosine_similarity()* function onto each user's review array. From there the data enters the *book_recommendations()* function, filtering out those that are not similar to the user. From there *book ids* are collected if the closest users have reviewed it but not the user. After the list is made, it is passed into the *get_book_titles()* to retrieve the book title by id.

The libraries used are:

- Dask - bags and dataframes for data processing and main functions for parallelization.
- Numpy - assistance with arrays.
- Time - testing
- Matplotlib - graphs

Results

Due to Google Cloud not working as intended and Google Colab constantly crashing, the code was run in Visual Studio Code, with the tests being manually done. VSCode was used as it is a standard industry IDE, as well as its ease of use. The multiprocessing within VSCode did not work due to being cancelled by the libraries which were Dask client and multiprocessing. The following data was collected on a Mac Studio with the following hardware pc stats: 64Gb, 12 processor cores, 34 GPU cores and a M2 chip.

The time taken for the algorithm to compute the recommendations was recorded as the amount of data used increased. This is shown in Figure 1. The data was partitioned into test sets to run through the algorithm increasing by 20% each time.

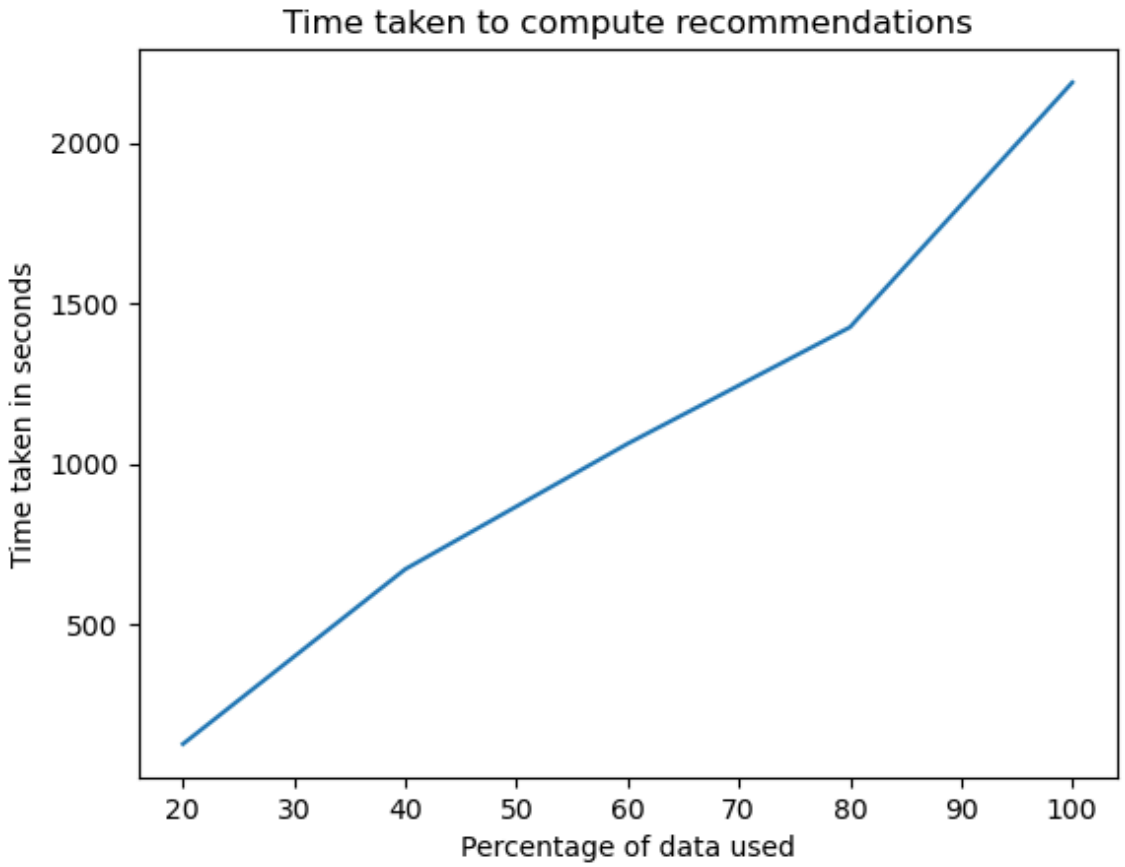

Figure 1: Time taken to compute recommendations (seconds)

As seen in Figure 1 as the percentage increases so does the time in a linear fashion. The dataset being used, Goodreads review data, has 1.3 million entries. The percentages are of that 1.3 million. The above data includes loading, partitioning and cleaning the data before actually running the recommender system through it. The majority of the time comes from handling this but due to the implementation it had to be done like this each time so was recorded as such. This implementation doesn't include the final call where the book ids are used within the book database as this wouldn't represent the recommender fully due to it already making a recommendation. That said there are more efficient ways that can be done to reduce these issues which will be talked about in the Reflection.

The recommender was able to recommend books that a given user might be interested in based on reviews that similar users to themselves had reviewed that they had not reviewed. This can be refined further but as a basis for the algorithm is a good start and performed well within the given tests.

Conclusion

The recommendation system was successfully made meeting the initial hypothesis. Given a user a set of books can be recommended based on similar user reviews. This will choose the

top n reviews based on how many are desired. This allows for reviews which may not be the best due to there being no good books that similar users have enjoyed. This system allows for more specific conditions due to having the amount of recommendations desired.

The recommender system shows how large data can be processed quickly through the use of Dask. This could then be used for more complex or more important data that is required to have a lot of calculations quickly as long as the access of the data is quick and easy. Not having the quick and easy access of data the algorithm would significantly slow down. This highlights the importance of data management and access the algorithm has.

Next directions this project could be taken would be to allow for more specific requests such as specifying the min and max rating that should be recommended. Another aspect would be inspecting how to optimise the data loading and access of the data to make applying the algorithm quicker instead of needing to wait on the loading first.

Critique

Having a different approach to designing the algorithm to work on the multiprocessing properly would've been a better idea as during testing it was unable to work as intended. The errors that came up during testing the multiprocessing was an error with the client, the function cancelled due to arbitrary reasons and the different cores/ processors crashed during dataloading. This should have been checked when the colab methods broke and were unable to be fixed as others did not know why it broke or how to fix it.

Changes that could be made to make the algorithm/ recommender better would be to improve data loading to speed it up or allow for better access of data. As mentioned previously allowing for a range of ratings to be allowed in the recommendation to provide more flexibility. The current one does work but this addition would enable bad reviews to not be recommended if they're the only ones available.

Reflection

Course concepts that helped were the Dask functions and applications behind them. Functions such as map, reduce and foldby were integral in the use of the recommender system by allowing a function be applied across all entries at once/ in batches to speed up the process. I learned that identifying and planning what the data is and what it needs to go in each process is important and being able to inspect and analyse the data is required otherwise nothing gets done. Having a dataset that is correctly implemented and not broken helps immensely and switching sooner than later if the dataset is broken is the best course of action.

References

Item recommendation on monotonic behavior chains
Mengting Wan, Julian McAuley