CAPSTONE PROJECT: FINAL REPORT

Elyse Renouf | July 31, 2020

Problem Statement & Background:

Tell Me A Story (TMAS) is a content-based young children's book recommender using NLP that is derived from a Kaggle dataset of Goodreads.com book reviews. It was created so parents could input one of their child's favourite books and TMAS can recommend other similar books based entirely on similarities between key words in the book descriptions.

As a parent, I have tried to find new books about firetrucks that are similar to the ones my son currently loves, and that are aimed at kids in his age group. This has required a lot of research on parents' forums and expensive trial and error; especially with libraries and bookstores being closed during COVID, the cost to try a new book in both time and actual monetary cost can be fairly high. After speaking to other parents, and exploring kids' books on Amazon.com and GoodReads.com, I realized this was a common issue. There isn't really a place where you can say "my kid is this age and loves this book, give me more like it!".

So, I decided to explore NLP and content-based recommenders for my capstone project, in hopes of creating a content-based recommender that would give parents age-appropriate book suggestions based entirely on the text/subject matter included in the book itself, not on previous sales or on sponsored ads.

About the Data:

Finding a large dataset solely made up of kids' books without the ability to get API access proved difficult. I was able to find a series of .csv files on Kaggle which were originally web scraped from GoodReads.com.

The dataset included book name, author, publisher, and the book ISBN (a unique identifier number), page number, average rating given by users, and a short book description – as well as few other columns that were not used in the model including publishing day/month/year.

In order to make this text-based recommender work, I could only include the files where there was a book description, which included 11 files in total, of approx. 100k book titles per file.

Unfortunately, there was no book category/genre or age category available in this dataset. In order to ensure TMAS would only return/recommend only children's books, I researched publisher standards on the average page numbers recommended for children's books and used these as a first stage filter to remove any books over 38 pages, merge all the .csv files into on single kidsbooks.csv and to create a column called 'is_preschooler' to separate out the books into two age categories:

- 'Is_preschooler': 0 = infant/toddler ages 0-2 years old with books 30 pages and under
- 'is_preschooler': 1 = preschooler ages 3+ with books 30-38 pages (the dataset page max)

I also removed all books under 4 pages to exclude a large majority of audiobooks and study guides.

Data Cleaning and Exploratory Data Analysis:

After collecting only books with page counts between 4-38 pages, merging them into a single dataframe and adding the age category, the bulk of the cleaning was on removing rows with null values in the description column and then exporting the file to Excel to manually review all book titles and create a list of approximately 500 publishers that were children's publishers. I also standardized all publisher names to remove all variation i.e. harper books, harper TM, harper UK, etc.

After all cleaning steps and the removal of all duplicates based on the ISBN (a unique book identifier number), I was left with a dataset of just over 10,000 books.

Then, after all cleaning & preparation steps, I used a CountVectorizer to count and create a list of the most frequently appearing words. This practice mostly confirmed my decision to use Term Frequency – Inverse Document Frequency (TF-IDF) in order to weight the results toward more unique terms which helps to discover similarity between the core/key topical words in book descriptions, instead of just the most frequently appearing words.

Also for curiosity's sake, I attempted to find any trends in the most frequently appearing year, month, and day that the majority of the books in this dataset were published on but quickly discovered some data entry errors where month and day were swapped in a large portion of the dataset that prevented any definitive insights. I also found the largest number of books had exactly 32 pages and that, with my page count / age group breaking point at 30 pages, I had assigned the majority of books into the preschooler age group, which seems to be a fairly accurate reflection of the quantity of books available for each age group in any bookstore.

Modeling:

Because I knew I wanted to do a content-based recommender with this particular dataset, there was very little required in the way of additional modelling/evaluation outside of setting up for my recommender.

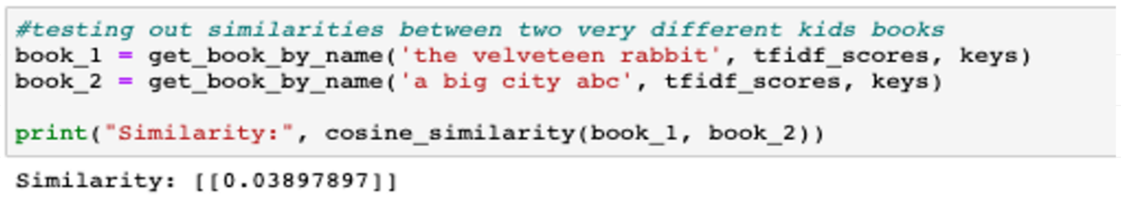
I built my tokenizer function to remove punctuation, split the sentence into individual words, stem the verbs to their root word, to remove English stop words, and to remove words shorter than 2 characters to remove any lingering html tags from the original web scraping. I then tested it on a simple sentence to ensure it worked as required before applying it within my model on the entire dataset.

I used a helper function that would look up a books TFIDF score by the book name, then instantiated and fit the model and transformed the data. Once all the book description text was transformed, I used a function to calculate cosine similarities to determine just HOW similar one book description is to that of other books in the dataset.

Model Evaluation & Conclusions:

Content-based recommenders are unique in that the key measurement used to evaluate performance is a similarity score using cosine similarities. The ultimate goal is for TMAS to take in a single book input by the user and return a list of similar books – ranked on how closely related they are to the user's book by using only the text in the book description.

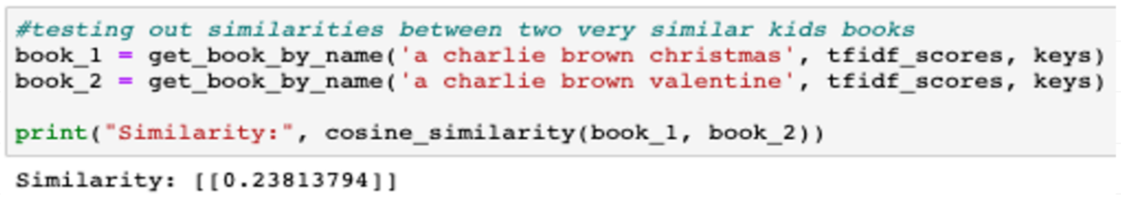
In order to test to determine if TMAS was working well, I input what I knew to be two very different books (in both topic and word choices) and asked it to output the similarity score, which returned a very low score of under 4%. I then input two books I knew would be very similar and got out a similarity score of 24%, so that was the score I used as a benchmark to determine what a "good" similarity score would be when used to compare other books in this dataset.



```
#testing out similarities between two very different kids books
book_1 = get_book_by_name('the velveteen rabbit', tfidf_scores, keys)
book_2 = get_book_by_name('a big city abc', tfidf_scores, keys)

print("Similarity:", cosine_similarity(book_1, book_2))
```

```
Similarity: [[0.03897897]]
```



```
#testing out similarities between two very similar kids books
book_1 = get_book_by_name('a charlie brown christmas', tfidf_scores, keys)
book_2 = get_book_by_name('a charlie brown valentine', tfidf_scores, keys)

print("Similarity:", cosine_similarity(book_1, book_2))
```

```
Similarity: [[0.23813794]]
```

Further Applications & Potential Next Steps:

I was then able to take this first version of the model and put it into a Streamlit app as a working model, though run/loading time was entirely too slow to use as a live demo at this point.

TMAS Version two, a working model in my Jupyter notebook allows the user to filter the books that are returned by age group – so the recommender only compares the description text of all books in that particular age group. Now that this filter is working well in notebook form, my next step would be to create a working second version of the Streamlit application using this age filter. This draft is included in the project folder as TMAS-app.filtered.py but is not yet completed.

Just separating out the results by age group has increased the returned similarity scores. So TMAS is already returning more closely related books, with top scores around 32% in version two, compared to top scores of approximately 24% in version one. This filter will also likely increase response time when loaded into app format, which is much needed as the original app took so long to load, it couldn't be demonstrated live during presentations because adding it would have exceeded the 3-minute presentation time limit.

If given access to a dataset of books or book API that also has already designated age recommendations, book type (board, picture, reader), there is potential to make my age groups more specific and add a book category/type filter option. From a UX/UI perspective, having access to the book covers and links on where to buy would also be helpful and more engaging for the end user.

Once a more robust dataset is in play, I could also have the user select the starting letter of their book first and populate the select box with only books starting with that letter. Or move away from a dropdown select box altogether and allow users to free type in the books they are currently reading, providing an auto-complete/text type recommendation function that suggests the names of books that exist in the database, with a custom error message if that book is not found in the database. This adds a level of complexity that would require much more research and intelligent app programming.

I would also like to remove more book-driven common words (illustrated, reader, author, etc.) and streamline the recommender to specifically target parts of speech (nouns, adjectives, and verbs). This may also increase recommender accuracy and decrease run time required to make these comparisons.

Another way to improve accuracy would be to get larger excerpts of each books' text (or the whole story) to run through the recommender to as some books have very generic descriptions but are content-rich in the story itself.

If that level of data/intelligence is possible then I would like to continue my research and work into hosting the final app online, potentially integrating Amazon affiliate links and building out any major improvements to runtime efficiency in order to make this project useable for the public.

APPENDICIES

Attachments:

1. README.txt – a readme file outlining all files included in this project folder (also included below)

Project Summary Documents:

2. 1 - ElyseRenouf_Capstone_Findings.pdf - This is the final report that summarizes the project
3. 2 - ElyseRenouf_Capstone_Presentation.pdf - This was the capstone slide deck as presented to the data science cohort in July 2020
4. 3 - TMAS Demo.pdf - This was the slide deck as presented during the initial TMAS demo video

The process in Jupyter Notebooks:

5. Loading_Merging_Datasets.ipynb
6. Cleaning & EDA.ipynb
7. Vectorizing & Modelling.ipynb

Streamlit App Files (2 versions):

8. TMAS-app.py - This is version one of the TMAS Streamlit app
9. TMAS-app-filtered.py - This is version two of the TMAS Streamlit app is still a work in progress

Data Folder Files:

10. Raw Data/Original .csv files named in numerical order starting with book600-700k.csv up to book1700k-1800k.csv could not be added to the submitted folder on Synapse due to file size restrictions. Please visit Kaggle for the original data files.
11. kidsbooks.csv - csv of all merged kids' books .csv files, before cleaning
12. master.csv - csv of cleaned, finalised kids' books data and all columns
13. books.csv - csv of cleaned kids' books data with only columns needed for recommender
14. infbooks.csv - books filtered by is_preschooler column as 0 (30+ pages for kids aged 0-2 years old) for potential delineation for Streamlit app version 2
15. preschoolbooks.csv - books filtered by is_preschooler column as 1 (31-38 pages for kids aged 3-6 years old) for potential delineation for Streamlit app version 2

Data Sources:

The outside resources that I've used throughout this project:

1. Datasets gathered from:
   https://www.kaggle.com/bahramjannesarr/goodreads-book-datasets-10m

2. Children's Book page range averages and ages estimated based on info from:
   https://www.jennybowman.com/what-genre-is-my-childrens-book/

3. A deeper insight into TF-IDF:
   http://www.tfidf.com/