THE OHIO STATE UNIVERSITY

Book Recommendation System Using Goodreads Dataset

Project Category: Class-Defined
Physics 5680, Autumn 2024

Author(s): N. Ghugare

December 11, 2024

Abstract

Finding books based on personal tastes can sometimes be a uphill battle. Having very little information at our disposal, like a book's title, cover, and description, it can be difficult to know if one would find a book enjoyable. This project attempts to provide book recommendations by combining multiple different machine learning and NLP methods. We repurposed the Goodreads raw English review subset for spoiler detection in order to fit sentiment analysis and collaborative filtering tasks. We used the review text in the dataset to tokenize and train a BERT-Tiny model for sentiment analysis, where the model predicts the reviewer's rating score. Training the BERT-Tiny model yielded a training accuracy of 55% and a testing accuracy of 53%, a better result than random guessing (20%). For collaborative filtering, we slimmed down the dataset to only include user information, rating information, and book information, storing it as a matrix, and then we applied two models to the matrix: SVD and KNN clustering. Iterating over all books for a user id, we weight the SVD and KNN recommendation scores and then return the highest scores, thus provided the user with personalized book recommendations. Thus, we created a pipeline, where we can pass in user id, book id, and review text into our BERT-Tiny model, and then use the predicted rating to get recommendations from SVD and KNN for the user.

1 Introduction

Book enthusiasts all have one common problem: it is extremely difficult to find new, enjoyable books. Not only does it waste time trying to read book descriptions and titles trying to conclude if they would enjoy that book, but it could also cost a lot of money. Being a fellow book enthusiast, it would be extremely useful if some sort of program could take book reviews or ratings, and then spit out books that others have found enjoyable based on that singular book (or collection of books). Therefore, this project seeks to solve this problem using multiple machine learning and natural language processing methods. We repurpose the Goodreads raw English review subset for spoiler detection dataset to fit our needs. We first made a subset of this dataset only containing one feature (review text for some book) and with one target (the rating score to predict). Since the dataset contains over 1.3 million books (meaning 1.3 million reviews), it would take ages to train even a BERT-Tiny model we trained. Therefore, we stratify-sampled the main dataset to get the same fraction of inputs for each rating value, leaving us with about 130,000 reviews for training, testing, and validation, which we then pre-processed for fitting. We tokenized these reviews using a BERT-Tiny tokenizer to a max vector length of 200 (with padding or truncation if needed), then used the tokenized dataset to train a BERT-Tiny model. With this model, we can take a new user's review of a book and assign it a tangible rating score, just based on the content of the review. Using a copy of the original dataset, and

cutting out all columns except the user id, book id, and rating value, we create a compressed sparse row matrix with rating value being the values in the matrix, and the ids being their own axis. Following that, we applied SVD and KNN clustering to the matrix, which allows us to get collaborative filtering predictions for a user. These models take into account other user's reviews who have read similar books to the target user to make predictions. Putting all of this together, a user's text review of a book can be predicted as a rating score, then filtering through collaborative filtering to receive a hybrid recommendation of books they'd likely enjoy.

2 Related Work

There is a lot of related work when it comes to recommendation systems using sentiment analysis, content filtering, and/or collaboritive filtering. The paper that is the closest to resembling this project is the enhanced Goodreads book discovery paper by Lee Choo Hui [7]. The paper uses a "hybrid recommendation" system which combines sentiment analysis, collaborative filtering, and content filtering to make recommendations, which is exactly what I wish to do for this project. There are some differences in the project. For example, for collaborative filtering, the paper only used SVD and reconstructed the data, while this project sought to use KNN clustering *and* applying SVD to the collaborative filtering data as an extra step to try to boost the recommendations. For sentiment analysis, the paper used two tokenization techniques, "stemming" and "lemmatization", and used this to train on a VADER sentiment analysis model. In contrast, this project used BERT to test sentiment analysis and did not use either of the tokenization techniques, rather opting to use BERT's custom-designed tokenizer. The paper noted that the model would not be able to pick up on language-specific aspects, like sarcasm, which is likely to be problematic in this project as well. For the hybrid filtering the paper used, they weighted the content-based and collaboritve-based filtering using a weighted average, however this project does *not* include content-based filtering, instead calculating a weighted average on the two collaborative filtering models.

A paper by Erin Cho and Meng Han [4] also attempted a similar hybrid analysis, putting personally collected review data through popularity rankings, collaborative and content filtering, and lexile similarities. Although including popularity rankings is an interesting aspect to consider, popularity may not always correlate with enjoyment, which is why this project didn't consider popularity. Additionally, not including sentiment analysis of text reviews means there was no distinct metric to consider "how much" a person enjoyed a book.

The final main paper that we considered was a paper involving different recommendation algorithms by Clemens Tegetmeier [19]. This paper considered all the different AI techniques that could be used for book recommendation systems, including performance metrics, KNN clustering, SGD on matrices, and more. The data the paper used to apply and conclude their results was not the Goodreads dataset, and the paper does not talk about sentiment analysis or hybrid filtering methods, which the other papers considered. The paper concluded that SGD worked better for collaborative filtering than simply doing KNN clustering, but the paper never talked about using SVD on a matrix, which is what this project did alongside KNN clustering, which allows for possible performance improvement on recommendations when weighting with KNN recommendations. We also examined more articles that discuss the exact methodologies to apply techniques like SVD or content-based filtering, which can be found in the References section.

This project seeks to put all of these works together. This project takes inspiration from the paper by Lee Choo Hui [7], by integrating sentiment analysis with hybrid filtering. This project uses BERT instead of VADER and opted to use KNN and SVD collaborative filtering instead of just SVD filtering, and we did not explore content filtering, unlike the paper. We considered the paper by Cho and Han [4] for using popularity ratings, but ultimately decided against it. That same paper's collaborative filtering uses Pearson correlation values in order to make its predictions, but we instead opted to try KNN and SVD, and we built on top of this paper by integrating the rating values from our BERT model to get a distinct classification of book enjoyment, which this paper did not do so. Finally, we saw that the Tegetmeier paper [19] analyzed the differences between SGD and KNN for collaborative filtering. Using the Hui paper [7], which used SVD, we combined this paper with that to use both SVD and KNN, opting not to use SGD as KNN is simpler to understand and easier to implement. Thus, this project becomes a combination of methodologies from these three papers (alongside other resources).

3 Dataset

The dataset is the Goodreads raw English review subset for spoiler detection. The data contains over 1.3 million book reviews from around 25,000 books and 19,000 users. While the dataset was initially created for spoiler detection (checking if a spoiler is in a text), we repurposed the dataset to work for our needs. We used the dataset in multiple ways. First, we will discuss the ways we used the dataset for sentiment analysis on BERT-Tiny. We took a copy of the dataset and cut out all the columns except for the review text and the rating given for said review text. We then stratify-sampled the dataset to receive a smaller, easier to train dataset, with the sampled dataset having the same fraction of each review score as the original dataset. We did this because training 1.3 million reviews would be CPU and RAM intensive, and could take a lot of time. This reduced our dataset to 84,202 training samples, 21,051 validation samples, and 26,314 test samples. When splitting the train and validation sets from the testing set, we opted to use group shuffle splitting based on the book id. This meant that the data points with the same book id must all exist either in the test set or in the training/validation set. They may not exist in both. We did this to ensure the models performance on the test set, as it is full of book reviews the training model doesn't know about. We then pre-processed the review text by lowercasing the review text, removing excess whitespace, and dropping exact equal review texts. Finally, we used a BERT-Tiny tokenizer to tokenize the sets to a max vector length of 200, with truncation or padding if necessary. The dataset can be downloaded from `https://mengtingwan.github.io/data/goodreads#datasets`.

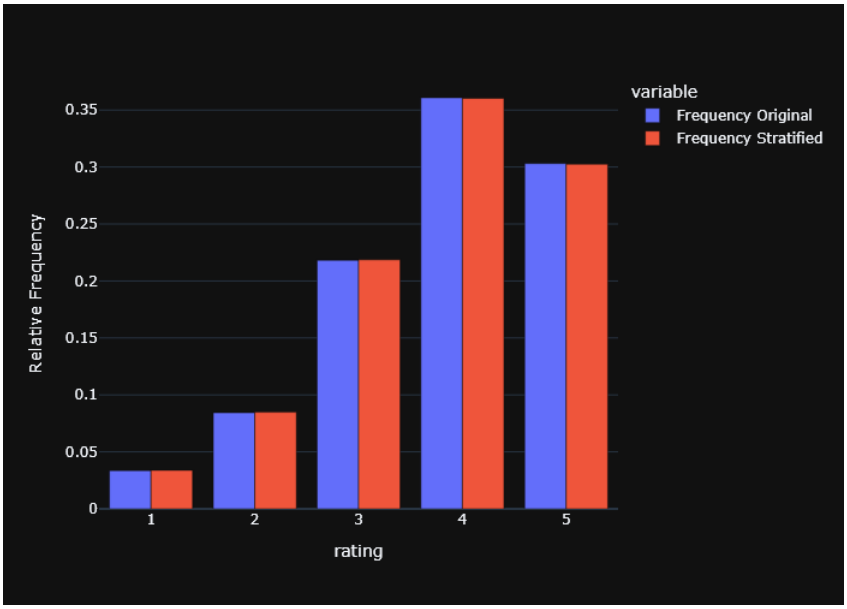


Figure 1: Stratified sampling of original dataset based on relative frequency of rating score.

For collaborative filtering, a copy of the same dataset was made, and then a subset was made only including the user id, the book id, and the rating value columns. Using this we created a compressed sparse row matrix (CSR matrix), with the user id on one axis and the book id on the other, and the matrix components containing the rating. We filled the missing ratings with the value of 0, and we can crudely visualize the non-zero ratings as a 2D image. Even with the large amount of missing ratings, we still have a lot of daata points / review values. Using this sparse matrix, we can fit SVD and KNN models to use for collaborative filtering.

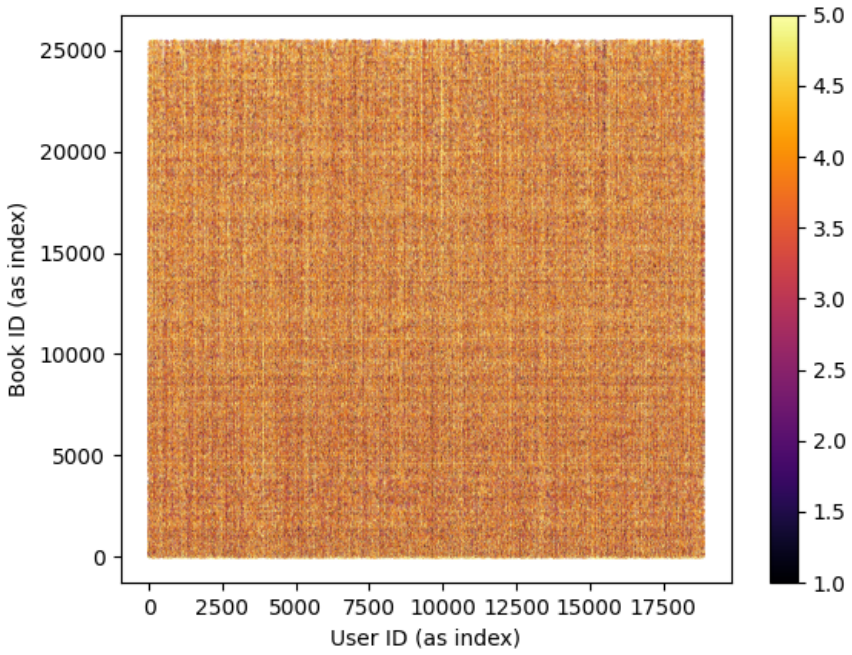


Figure 2: 2D scatter plot matrix of non-zero rating values for varying book id and user id indices.

4 Methods

Since this project encapsulates multiple different machine learning algorithms, we will break up the explanations into sections.

4.1 Sentiment Analysis

BERT-Tiny is a version of Google's BERT model, which is specifically designed and trained for natural language processing (NLP) tasks. The specific BERT-Tiny model we used was accessed on Hugginface at `https://huggingface.co/prajjwal1/bert-tiny`. BERT's advantages over other NLP models like VADER or doing sentiment analysis using an RNN is that is that it is more accurate compared to other models and is able to do more tasks than other models (more generalizable). The drawback of it is that it is more computationally expensive and more complex. BERT comes in many different pre-trained models. To save training time, we opted to use the smalles of the BERT models called BERT-Tiny. One of the main advantages of BERT is that it is completely open source: `https://github.com/google-research/bert`. BERT stands for Bidirectional Encoder Representations from Transformers. BERT uses the following methodology to train NLP tasks [12]:

- Data: training on a large dataset containing 3.3 billion words!
- Masked language modeling: hiding a word in a sentence and forcing BERT to use the words around it to predict the word.
- Bidirectionality: BERT uses words on both sides of the masked words for prediction.
- Next sentence prediction: BERT is trained to predict if a sentence follows another.
- Transformers [23]: BERT uses transformers, a specific and powerful deep-learning algorithm.

Transformers [12] work by leveraging attention, creating weights that identify which words in a sentence are the most important for further processing, thus paying "attention" to only the important parts of the data. A transformer uses an encoder to process an input and can efficiently process large amounts of data. After training this BERT model, a user's review text can be passed into the model to retrieve a predicted rating score of the text.

4.2 Collaborative Filtering

The first part of the collaborative filtering section of the project utilizies a linear algebra component called singular value decomposition, or SVD. If we have an mxn matrix A, then SVD on A is defined by the following equation [6]:

$$A = U\Sigma V^{\mathrm{T}}, \tag{1}$$

where U is a mxm matrix of the orthonormal eigenvectors of the matrix $A \cdot A^{\mathrm{T}}$, Σ is also a matrix (not a sum) which is diagonal and with r elements equal to the root of the positive eigenvalues of U. Finally, V^{T} is the transpose of an nxn matrix containing the orthonormal eigenvectors of $A^{\mathrm{T}} \cdot A$. In short, SVD is a way to factorize a given matrix into three matrices. The use of doing so is to reduce dimensionality and noise, and given a rating matrix, it can be used to help predict the missing values in the ratings [8].

KNN is a nearest-neighbors algorithm that can be used for clustering or regression tasks. In some N-dimensional vector space, the KNN algorithm finds the K closest points to a given vector, using some provided distance calculation [5]. Thus, the algorithm is called K-nearest neighbors, or KNN for short, since the closest K "neighbors" to a point are being found. For this project, we used Scikit-Learn's NearestNeighbor class, defaulting the algorithm to "brute", which means that the distance formula was calculated for all points in the space to get a list of all distances between points [24]. The method used for this calculation is simply the Euclidean distance:

$$d_{\mathrm{E}}(x, X) = \sqrt{\sum_{j=1}^{N}(x_j - X_j)^2}, \tag{2}$$

where x is the given point in the N-dimensional space, and X is some other "neighbor" in the space. The amount of neighbors to find is a hyperparameter that can be tuned. We opted to use 10 neighbors. So for each data point, we find the 10 points (not including the point we pass in) with the *shortest* (or smallest) Euclidean distance.

4.2.1 Making Predictions

Making predictions for the KNN model is relatively simple. Using a rating score (whether predicted or true) and given a user id, the specific data point including the user id, book id, and rating score can be placed in the vector space with the other data points from fitting the KNN model on the sparse matrix. From there, the model finds the 10 closest neighbors to that data point and calculates the average rating score. We then iterate this for each book and find the books with the highest average rating score. KNN models for this usage have two main drawbacks: they may be calculating scores based on 0-values filled that happen to be near the data point, and the closest data points to the data point may be from the user themselves. Fixing the latter was simple, ignore any data points from the user, which would change the average rating score. Fixing the former was harder, as it is a built-in problem with KNN. Therefore, out of SVD and KNN, we opted to weight the KNN ratings lower at 1.5x and 0.5x respectively. Defined mathematically, where S is the final score and R is the rating at the current data point:

$$S_{\mathrm{knn}} = 0.5\frac{\sum_{j=1}^{N} R_j}{N} \tag{3}$$

When fitting SVD on the matrix, we can extract the two factors (defined as U and V^{T} above), which represent a list of the reduced-dimension vectors of the user ids and the book ids. We can calculate a predition for a given user id and book id by taking a dot product of the user factors vector for the given user id and the transposed item factors vector for the given book id. We iterate over all book ids to get all predicted ratings.

Mathematically, where $\mathbf{u}$ is the vector from the vector list U for the user id, and $\mathbf{v}$ is the vector from the vector list V^{T} for the book id:

$$S_{\mathrm{svd}} = 1.5\left(\mathbf{u}\cdot\mathbf{v}^{\mathrm{T}}\right) \tag{4}$$

Thus, to get a score for a specific book id and user id, S, we can say that:

$$S = S_{\mathrm{knn}} + S_{\mathrm{svd}} = 0.5\frac{\sum_{j=1}^{N} R_j}{N} + 1.5\left(\mathbf{u}\cdot\mathbf{v}^{\mathrm{T}}\right) \tag{5}$$

4.3 The Pipeline

The pipeline for the project combines these two aspects together. We are given user information like the user id, book id, and the review text (no rating score is provided). From the review text, the BERT-Tiny model does the sentiment analysis providing a predicted rating score. After that, the predicted rating score is sent to collaborative filtering (specifically to the KNN model), retrieving the needed list of SVD and KNN recommendation scores. We find the highest recommendation scores and return that to the user.

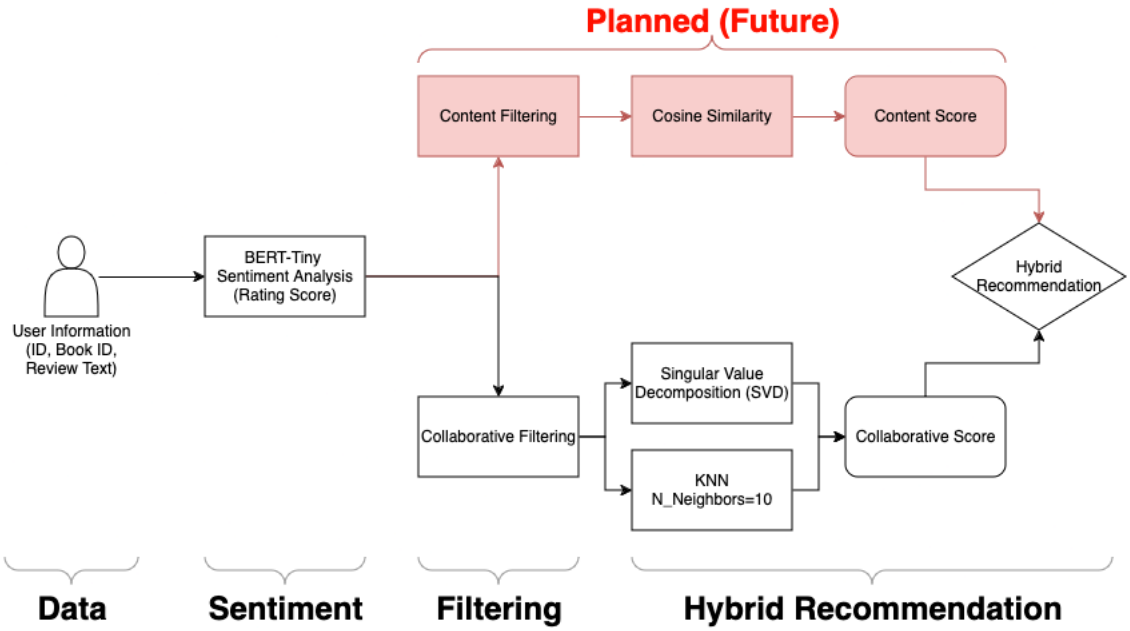


Figure 3: Project pipeline of sentiment analysis and hybrid filtering.

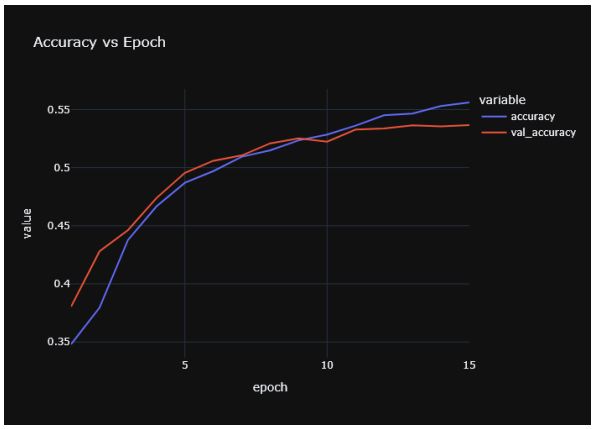
5 Results/Discussion

We compiled BERT-Tiny model using an Adam optimizer with a learning rate of 1e-5. We chose this learning rate because literature indicates the model is harder to converge with a higher learning rate [18]. We used a batch size of 16, as that batch size seemed to optimize training time vs. computation resource requirement through testing of multiple batch sizes. The loss function we wished to minimize is the sparse categorical crossentropy function, defined as follows:

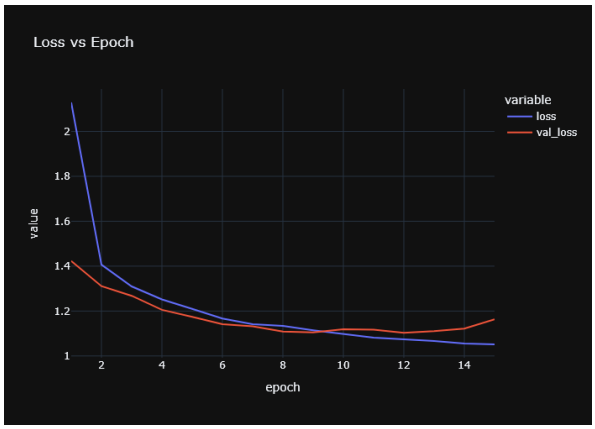
$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log\left(\hat{y}_i\right) + \left(1-y_i\right)\log\left(1-\hat{y}_i\right)\right], \tag{6}$$

where $\mathbf{w}$ is the model parameters (weights, etc.), y_i is the true label, and $\hat{y}_i$ is the predicted label [10]. This is the same as the typical categorical crossentropy formula. The reason we use sparse categorical crossentropy

is because our labels are *not* one-hotencoded, and are, in fact, integers like [1] or [5] [10]. Since we are not super concerned about slight deviations in predicted rating from true rating, we opted to choose the simple-to-understand accuracy as our metric. We trained the model for 30 epochs, monitoring the validation loss to stop training if the validation loss does not decrease for three epochs, and saving the best model weights based on the validation loss.



(a) Training and Validation Accuracy per Epoch.



(b) Training and Validation Loss per Epoch.

Figure 4: Training and Validation Metrics as a Function of Epoch.

The best model from training had a training/validation accuracy of about 55%. The testing set did not contain any books from the training set. Evaluating the best model on the test set yielded an accuracy of 53%, indicating that our model makes good generalizations of the data (no over- and underfitting). Since the accuracy curve did not completely flatten, it is possible that we could have attained a higher accuracy, but we decided not to do so in order to save computational resources. Additionally, an accuracy of about 55% is better than random guessing a rating score from 1-5 (20%), so we claim that this model is sufficient.

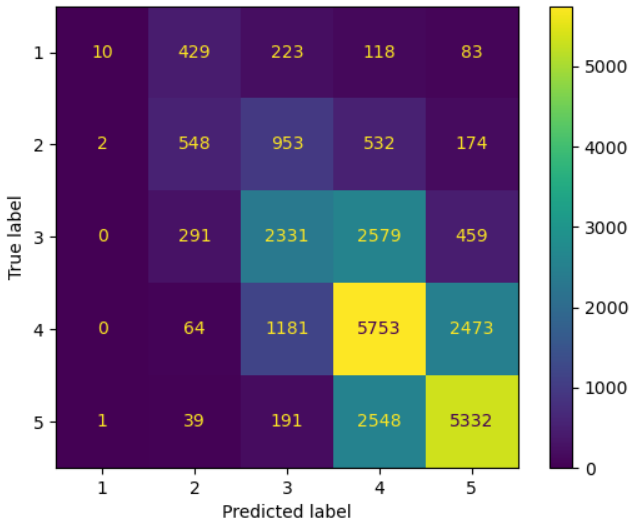


Figure 5: Confusion matrix of predicted rating vs. true rating on the test set on the best model.

From inspecting the confusion matrix on the test set, it seems that most predictions are either the true prediction are stray from the true value by only 1 rating point, which means that the model is able to understand the general sentiments from a review text, although it may not always pinpoint the exact rating value. There are few values where the true label differs greatly from the predicted label (only 1 predicted score of 1 when it was actually 5, and only 83 predicted labels of 5 when they were actually 1).

6 Conclusions/Future Work

In conclusion, this project sought to make a pipeline to make book recommendations to a user, given their user id, and given a book id and the user's review of the book. This review text is passed into a BERT-Tiny model to predict what the user would rate the review on an integer scale from 1-5. This predicted rating is sent for KNN filtering to find similar books and user ratings and provide recommendations for the user. We also use SVD to find latent factors of the books and users to then provide another recommendation scores for books for the user. We weight the SVD scores at 1.5x and the KNN filtering scores at 0.5x, since the KNN scores have intrinsic flaws when scoring, which motivated us to weight it less. We tabulate the sum of the weighted rating scores per book, and then return the highest rating scores to the user.

One of the best aspects of this project is its scalability; There is a lot of work that could be done (and will be done) in the future. There are three main aspects that could be done in the future: content filtering, a user interface (UI), and a larger BERT model. Leveraging another goodreads dataset, we could use content filtering to create a larger hybrid recommendation score, like in other projects [7]. Content filtering would take book descriptions and use metrics like cosine similarity to find book descriptions that are more similar to each other to provide a recommendation score. As of right now, a person's review information would need to be programmed into python to then use to get recommendations. A great addition to the project would be to create a UI where a person can directly interact with a website or app to put in information and reviews, and have a backend process the ratings using this project, and then return the recommendations back to the UI. Finally, it would be beneficial with more computaitonal resources to investigate a larger BERT model with more trainable parameters (like DistilBERT), or training the BERT-Tiny model on more data, in order to see if we can increase the accuracy of the rating scores provided.

References

[1] Abdulaziz, Monirah. "Goodreads Books Recommendation System." Medium, 28 Feb. 2021, monirah-abdulaziz.medium.com/goodreads-books-recommendation-system-619dc190c81.

[2] Arora, Mansi. "Collaborative Based Recommendation System Using SVD." Analytics Vidhya, 30 Mar. 2020, medium.com/analytics-vidhya/collaborative-based-recommendation-system-using-svd-9adc5b6b3b8.

[3] Bhargava, Prajjwal, et al. "Generalization in NLI: Ways (Not) to Go beyond Simple Heuristics." ArXiv (Cornell University), 1 Jan. 2021, https://doi.org/10.48550/arxiv.2110.01518. Accessed 17 Nov. 2024.

[4] Cho, Erin, and Meng Han. "AI Powered Book Recommendation System." Proceedings of the 2019 ACM Southeast Conference, 18 Apr. 2019, https://doi.org/10.1145/3299815.3314465.

[5] GeeksforGeeks. "K-Nearest Neighbours - GeeksforGeeks." GeeksforGeeks, 13 Nov. 2018, www.geeksforgeeks.org/k-nearest-neighbours/.

[6] GeeksforGeeks. "Singular Value Decomposition (SVD)." GeeksforGeeks, 18 Sept. 2021, www.geeksforgeeks.org/singular-value-decomposition-svd/.

[7] Hui, Lee Choo, et al. "Sentiment Analysis and Innovative Recommender System: Enhancing Goodreads Book Discovery Using Hybrid Collaborative and Content Based Filtering." Lecture Notes on Data Engineering and Communications Technologies, 1 Jan. 2024, pp. 97–111, https://doi.org/10.1007/978-3-031-59707-7_9. Accessed 16 Nov. 2024.

[8] Jagdeesh. "Singular Value Decomposition - a Comprehensive Guide on Singular Value Decomposition." Machine Learning Plus, 31 Aug. 2023, www.machinelearningplus.com/linear-algebra/singular-value-decomposition/.

[9] Lee, Ernesto. "Building a Collaborative Filtering Recommender System with Python and Google Colab." Medium, 14 Nov. 2024, drlee.io/building-a-collaborative-filtering-recommender-system-with-python-and-google-colab-91a623064b21. Accessed 16 Nov. 2024.

[10] "Machine Learning - Cross Entropy vs. Sparse Cross Entropy: When to Use One over the Other." Cross Validated, stats.stackexchange.com/questions/326065/cross-entropy-vs-sparse-cross-entropy-when-to-use-one-over-the-other.

[11] Matplotlib. "Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation." Matplotlib.org, 2012, matplotlib.org/.

[12] Muller, Britney. "BERT 101 - State Of The Art NLP Model Explained." Huggingface.co, 2 Mar. 2022, huggingface.co/blog/bert-1011-what-is-bert-used-for.

[13] Numpy. "NumPy." Numpy.org, 2024, numpy.org/.

[14] Pandas. "Python Data Analysis Library." Pydata.org, 2018, pandas.pydata.org/.

[15] Plotly. "Plotly Python Graphing Library." Plotly.com, 2023, plotly.com/python/.

[16] Scikit-learn. "Scikit-Learn: Machine Learning in Python." Scikit-Learn.org, 2019, scikit-learn.org/stable/.

[17] SciPy. "SciPy.org — SciPy.org." Scipy.org, 2020, scipy.org/.

[18] Sun, Chi, et al. "How to Fine-Tune BERT for Text Classification?" ArXiv:1905.05583 [Cs], 5 Feb. 2020, arxiv.org/abs/1905.05583.

[19] Tegetmeier, Clemens, et al. "Artificial Intelligence Algorithms for Collaborative Book Recommender Systems." Annals of Data Science, 8 June 2023, https://doi.org/10.1007/s40745-023-00474-4.

[20] TensorFlow. “TensorFlow.” TensorFlow, Google, 2019, www.tensorflow.org/.

[21] “Transformers.” Huggingface.co, huggingface.co/docs/transformers/index.

[22] Turc, Iulia, et al. “Well-Read Students Learn Better: On the Importance of Pre-Training Compact Models.” ArXiv.org, 25 Sept. 2019, arxiv.org/abs/1908.08962.

[23] Vaswani, Ashish, et al. “Attention Is All You Need.” ArXiv.org, 5 Dec. 2017, arxiv.org/abs/1706.03762.

[24] “1.6. Nearest Neighbors — Scikit-Learn 0.21.3 Documentation.” Scikit-Learn.org, 2019, scikit-learn.org/stable/modules/neighbors.html.