Improving Book Recommender Systems Using Reading Tags and Book Covers

Cinny Lin (ycl461), Yingrong Mao (ym1780), and Zhihan Yang (zy1229)

***Abstract*—Our project aims to use the available features about users and books to improve recommendation accuracy for books. Initially, we planned to recommend books based on user features. However, after implementing k-means clustering, we realized that the customers purchase habits do not vary significantly according to demographics. Then, we learned about item-based collaborative filtering, but also its limitation regarding scalability and cold-starts. Therefore, to offer more useful suggestions, we turned our task to a supervised learning problem. In terms of dimensionality reduction, we tested stochastic gradient descent (SDG), alternating least squares (ALS) and singular vector decomposition (SVD) matrix factorization methods. While for the cold-start problem, we used popularity model and content-based filtering as complement. Regarding content-based filtering, we built two models. We used Term Frequency-inverse Document Frequency (TF-iDF) on book tags. We also employed neural network models, incorporating the categorical and numerical book features as well as the images of book covers. The aggregated performance of our models turned out pretty interesting. To our surprise, SVD did not yield very good results. We discovered that neural based model gives the best performance, reducing RMSE by 94% than the basic k-nearest neighbors (kNN) model. This result reveals that we could efficiently reach a larger customer base since we are able to recommend books accurately even when we know very little about our new users. More specifically, considering cover images when making book recommendations can yield better results than only using text or numbers.**

***Index Terms*—recommender systems, collaborative filtering, matrix factorization, content-based filtering, neural network**

I. Introduction

A good recommender system is essential for any platform to become popular today, as seen in Spotify music, Netflix film streaming, or Tiktok video-based social media. Online shopping sites are no exception. McKinsey estimated that 35% of what consumers purchase on Amazon comes from product recommendations [5]. With the growth of consumerism as well as increasing supply of new products, e-commerce platforms today face two challenges when adopting item-based collaborative filtering (CF) as their recommender system: how to give recommendations efficiently when we have a lot of users and products, and what products we can recommend when we know very little about our users.

CF algorithm recommends products based on user ratings. User-based CF means recommending products based on similar rating patterns between users. Item-based CF was developed by Amazon [3] and has been proven to be more effective. Item-based is faster when there are more users than items. It is also more stable because the average rating received by an item usually fluctuates less than the average rating given by a user.

CF uses a item-user matrix where the size of the matrix is the number of the users times the number of items[10]. Therefore, when the number of users and items increase, scalability becomes a huge problem. The cold-start problem is a challenge when the product has not been rated by many users, in the case of item-based model, or when the user has not rated many products, when considering a user-based model.

Our project focuses on recommending books to readers. We tested three different matrix factorization methods to reduce dimensionality and predict user ratings: training kNN with SDG, training kNN with ALS, and SVD. As for the cold-start problem, we tested five methods: popularity model, TF-iDF model with reading tags, multi-layer perceptron (MLP) model using categorical and numerical book features, convolutional neural network (CNN) using book cover images, and a combination of MLP and CNN. We are especially interested in seeing the role of reading tags and book cover images in improving our recommendation results.

Thank you Prof. Li Guo for offering this course at NYU Shanghai.
Thank you Prof. Wilson Tam for supporting the course and our project.

II. Dataset and Features

A. Data Acquisition

We have three datasets in this project. Our main dataset is scraped from Book Crossing, a leading online book club. We also used two other datasets accessed from Good Reads API as supplement. All of the datasets are developer-friendly and can be obtained from Kaggle.

B. Data Features

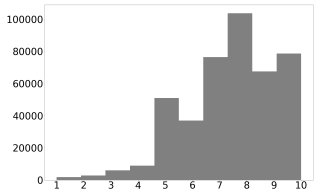
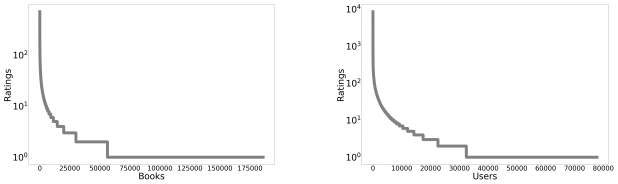


Fig. 1: Rating Distribution

Our main dataset contains information of 278,858 users, providing 1,149,780 ratings about 271,379 books. The demographic information we have about users include their age and location (city, state, country). The book features we have are title, author, publisher, year published, and book cover image. There are Book ID and user ID that help us merge the users' features and book features with ratings.

In addition, Good Reads is the world's largest social network for book lovers to search books and reviews. One dataset contains reading tags generated from users, which has around 39,800 records. The other dataset includes the following additional book features: language, number of pages, average rating, and the counts of numerical ratings and text reviews, which has about 55,412 records.

C. Data Preprocessing



(a) Book (b) User

Fig. 2: Rating Frequency (log)

We processed our data before constructing our machine learning models. Except the regular data cleaning of mistakes and outliers, our data preprocessing involves three more steps. First, as we discussed before, collaborative filtering requires users to give ratings and products to have ratings. Therefore, we filtered out books that received less than 10 ratings and users that gave less than 5 ratings. This step leaves us with around 40% of our original data. Moreover, we used MinMaxScaler to bring all the numerical features to the same level of magnitude. Last but not least, we used LabelBinarizer to one hot encode our categorical variables.

III. Explanation of Method Used

A. Baseline Models

1) K-Means Clustering: Initially, we wanted to segment users to see whether we can recommend books based on the preferences across different user groups. We applied k-means clustering on users' age and location features, and observed how it affects the books they read as well as the ratings they give. Using this method, x is the user or book features, while c_i is the average value of feature x in group S_i.

$$argmin \sum_{i=1}^{k} \sum_{x \in S_i} ||x - c_i||^2 = \sum_{i=1}^{k} |S_i| Var S_i$$

We encountered two problems while using this method. First, we realized that users' reading and rating habits did not differ significantly across users and book features. Secondly, since k-means clustering is an unsupervised learning method, it is difficult to evaluate the results of our model. Therefore, we turned to supervised models for our following experiments.

2) Popularity: The popularity-based model directly recommends the top n books to all users, sorted by the number of ratings the books received. As shown in our exploratory data analysis, we observed the "long-tail property" in our distribution of book rating frequency, where only an extremely small fraction of the items are rated frequently. We refer those frequently rated items as popular items. Around 1% of books that have more than 700 ratings.

3) K-Nearest Neighbor with Means: Here we set memory-based collaborative filtering as our baseline model. It is essentially linear algebra calculation for similarities between items and can be implemented using k-nearest neighbors. We use root mean squared error (RSME) as our minimizing objective in our item-based loss function. We also took into account the mean ratings of books μ_i and μ_j when considering the similarities between two books i and j. We chose 10 and 20 as the values of k while we tuned our hyperparameters.

$$\hat{r_{ui}} = \mu_i + \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - \mu_j)}{\sum_{j \in N_u^k(i)} sim(i,j)}$$

B. Collaborative Filtering with Matrix Factorization

Inspired by examples of Netflix[7][11] and other previous works in this field[9][10] in this field, we improved our collaborative model with matrix factorization. Item-based collaborative filtering makes recommendations based on similarities between the ratings that two products received. Here, $\hat{r_{ui}}$ represents our predicted rating, b is the bias term, and L is our objective function.

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T p_u$$

$$L = \sum_{r_u i \in R_t rain} (r_u i - \hat{r_u} i)^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2)$$

In this process, we used matrix factorization to reduce the user-item matrix dimension. We assume that there exist d latent features such that our $U * I$ rating matrix can be represented as the product of two lower-dimension matrices. Here, d is the hyperparameter, which we tune using three methods: training kNN wth SDG, training kNN with ALS, and using SVD.

1) Stochastic Gradient Descent: The general formula of SDG is given as follow. γ refers to the learning rate and λ refers to the amount of regularization we apply to our model, both of which are hyperparameters that we tune.

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$$

$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i)$$

$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u)$$

$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i)$$

2) Alternating Least Squares: While implementing ALS, we start by treating a set of latent factors as constant. For example, in the formula above, we pick the item vectors, y_i. We then take the derivative of the loss function with respect to the other set of vectors, the user vectors, x_u and solve for the non-constant user vectors.

$$\frac{\partial L}{\partial x_u} = -2\sum_i (r_u i - x_u y_i^T) y_i + 2\partial x_u = 0$$
$$= -(r_u - x_u Y^T)Y + \partial x_u = 0$$
$$= x_u(Y^T Y + \partial I) = r_u Y$$
$$= x_u = r_u Y(Y^T Y + \partial I)^{-1}$$

Then, we alternate back and forth and carry out the two-step process until convergence. The reason why we alternate and keep one vector fixed each time is because optimizing both vectors simultaneously is difficult.

$$\frac{\partial L}{\partial y_i} = y_i = r_i X(X^T X + \lambda I)^{-1}$$

3) Singular Vector Decomposition: In linear algebra, SVD is a matrix factorization method. SVD algorithm was popularized by Simon Funk during the Netflix Prize [11] and is known to be effective in predicting ratings in recommender systems. The minimization process of SVD is also performed by SDG.

However, for new users or those who didn't rate enough books, the collaborative filitering method cannot provide desired recommendations to them. Also, it requires all user community to be active and may suffer from sparsity problem. Thus, in the next part, we use natural language processing algorithms and neural network based models to include reading tags and book cover images to improve our recommemder system.

C. TF-iDF for Item Similarity

TF-iDF (term frequency-inverse document frequency) is a numerical statistic to process text-data that the importance of a word is directly proportional to its frequency in the text and inversely proportional to its frequency in the corpus. We calculate the similarity of authors and book tags respectively and combined as well. tf-idf(t,d) is

$$tf(t,d) \times idf(t)$$

And we have

$$idf(t) = \log \frac{1+n}{1+df(t)}$$

D. Neural Network based Model

Neural Networks have proved their effectiveness for almost all machine learning problems as of now and they also perform exceptionally well for recommendation systems. Learned from the related work about House Pricing experiment[2]and the box-office success prediction[6], we constructed our neural network based model of multi-layer perceptron, convolutional one, and the combined one.

1) Multi-Layer Perceptron: We built a MLP with 8- and 4-hidden layers and included only the numerical and categorical features: author, title, publisher, year of publication, pages, language, ratings count, reviews count. After one-hot encoding the categorical data, we have a total of 13794 features. We used ReLU activation function for the hidden layers and a linear function for the final output layer.

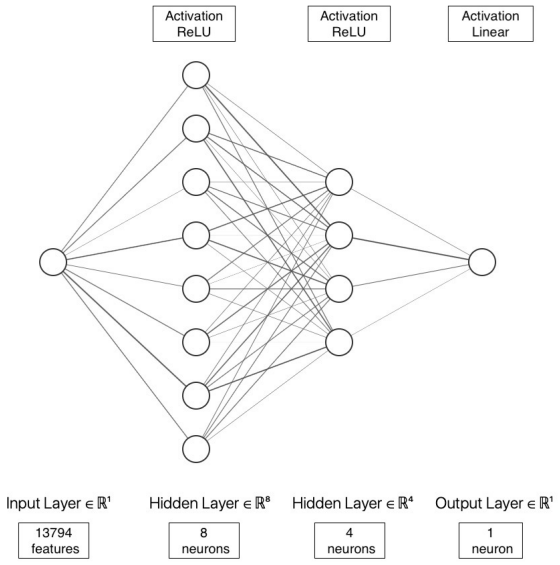


Fig. 3: Multi-Layer Perceptron Diagram

2) Convolutional Neural Network: We built a CNN model to only include book cover images. The input image size has height and width 32 and depth 3, with filter sizes 16, 32, 64.

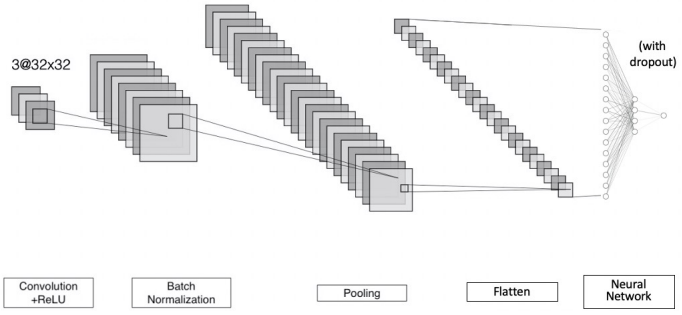


Fig. 4: Convolutional Neural Network Diagram

3) Combined Model: We built a model that combined the MLP and CNN model explained above.

IV. Results

In our evaluation, we focus on comparing the RMSE across different models. Since we could not evaluate k-means clustering for rating prediction, we left the model results in the appendix. From our result tables, we can see that our model scores improve as we progress.

Model	Popularity	kNN
RMSE	4.112	3.664

TABLE I: Baseline Models

We set the popularity model and the simple kNN model as our benchmarks. Explicitly speaking, the RMSE of the two models is used as the measuring standard for content-based and collaborative filtering respectively.

The best results was yielded from the kNN model when we set k=20 neighbors, and used mean-squared distance (msd) to calculate similarities between users and books.

Model	SDG	ALS	SVD
RMSE	1.663	1.663	3.438

TABLE II: Matrix Factorization Models

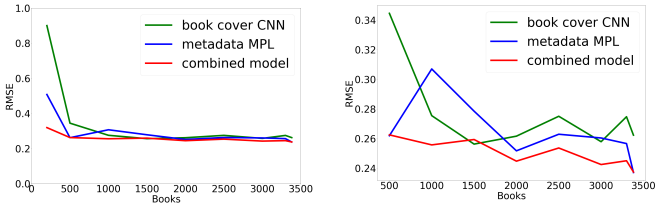
The best results for both SDG and ALS were also yielded when we set k=20 neighbors, but when using cosine similarity between users and books. The best results for SVD was when we set γ=0.005 and λ=0.4.

Looking into the matrix factorization results, we see that there is little difference between using SDG or ALS to train our kNN model in terms of their accuracy in predicting results. Both models were able to improve accuracy from the baseline kNN model by 54%.

However, we were surprised to see that using the SVD matrix decompsition method and training parameter estimation using SDG barely improved from the baseline model in our recommender system, despite implementing GridSearchCV and re-tuning our hyperparameters multiple times.

Model	MLP	CNN	Combined
RMSE	0.237	0.262	0.237

TABLE III: Neural Network Models



(a) Books 0-3500 (b) Books 500-3500

Fig. 5: RMSE across neural network models

Probing into implementations of neural networks for content-based recommender systems, we can confidently say that it yielded the best results out of all the models, improving from the baseline popularity model by 94%. In 5a, we plotted our accuracy as a function of the size of our input book data size. We observed that our neural network model is sensitive to the input size, with approximately 76% improvement from 200 books to the full dataset. The RMSE converges to around 0.25 after 500 books.

In 5b, we look closer into the performance of each model, focusing on the part where they converge (i.e. after 500 books). Even though the results fluctuates, it generally aligns with our intuition. We see that adding book cover images to our model improves our recommendation results (red line) from using book meta data alone (blue line). However, using book images alone is not enough to make good recommendations, reaffirming our common sense that we cannot judge a book only by its cover.

It is important to note that we only tested on approximately 3,000 books, which is around 1% of our original dataset. Even so, we believe that we are able to make relatively accurate inferences based on these preliminary results, since we see the RMSE for all three models flatten after 500 books.

V. Conclusion and Future Work

In this project, we studied recommender system in depth and used multiple models and compared their advantages and disadvantages. We went from naively thinking that we could making recommendations based on clustering user features, to learning about item-based collaborative filtering as well as its limitations.

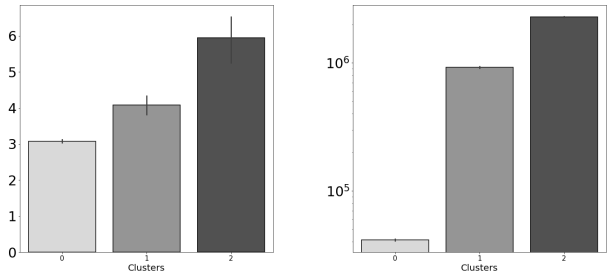
In terms of reducing the dimensionality of the item-user rating matrix, we tested three matrix decomposition and factorization methods.

When it comes to mitigating the cold-start problem, we used popularity model as the baseline model. We experimented how text and images can influence our recommendation results. For text, we incorporated reading tags labeled by GoodReads users, and tested the results using TF-iDF. One limitation of our experiment was that the GoodReads dataset with reading tags did not have enough overlapping books with our main dataset. After merging the datasets, we are left with less than 0.1% of our original data size (approximately 300 books), making the model training extremely difficult. Therefore, we did not proceed further with evaluating the role of user-generated text data in our recommender system. However, from our very preliminary findings, we believe that we would be able to see a significant improvement in recommendation accuracy if we collected more text data. We also acknowledge that there are other natural language processing methods that we can try out to improve our results, such as Rapid Automatic Keyword Extraction (RAKE) and TextRank.

Additionally, we tested three different neural network models, using categorical and numerical book features, using book cover images, and adding all book features into our model. To evaluate the role of cover images on book recommender systems, we built a simple CNN model. we are aware that there are more advanced models such as AlexNet or VGG that we can try out in our future experiments.

Looking forward, we can improve our project by adding more books and users features, augmenting our dataset size, and further testing more advanced models in order to understand more accurately the role text and images play in making better recommendation for users.

APPENDIX A
K-MEANS CLUSTERING RESULTS



(a) User Ratings (b) Book Rating Counts

Fig. 6: Clustering Results

An interesting finding we observed from the clustering results is that people who tend to give higher ratings to books also tend to read books that have more ratings. This may be understood because people who tend to give higher ratings probably also give ratings more often and care more about ratings of book in general, while users who give lower ratings probably only give book ratings when they really did not enjoy it, hence the lower rating counts for those books.

However, we did not observe that users who give higher ratings also read books with higher ratings. All users read books with an average rating of 4. This may be explained by the fact that 4 is the average rating of all books in general, as we can observe from the average rating given across users.

REFERENCES

[1] A. Dosoviitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In Advances in Neural Information Processing Systems, pages 658-666, 2016
[2] E.H. Ahmed, M. N. Moustafa "House price estimation from visual and textual features", NCTA 2016
[3] G. Linden, B. Smith, J. York, "Amazon.com recommendations: item-to-item collaborative filtering" IEEE, 2003
[4] I. Kanellopoulos and G. Wilkinson. Strategies and best practice for neural network image classification. International Journal of Remote Sensing, 18(4):711-25, 1997
[5] McKinsey & Company, "How retailers can keep up with consumers" (2013)
[6] R. Sharda, D. Delen "Predicting box-office success of motion pictures with neural networks", Expert Systems with Applications 30, 2006
[7] S. Funk, "Netflix Prize blog post", 2006
[8] S. Bell and K. Bala. Learning visual similarity for product design with convolutional neural networks. ACM Transactions on Graphics(TOG), 34(4):98, 2015
[9] Y.Koren, R. Bell, C. Volinsky, "Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems" AT&T Labs Research, 2007
[10] Y. Koren, R Bell, C. Volinsky, "Matrix Factorization Techniques for Recommender Systems", IEEE, 2009
[11] Y. Zhou, D. Wilkinson, R. Schreiber, R. Pan, "Large-scale Parallel Collaborative Filtering for the Netflix Prize" AAIM, 2008
[12] Libraries and resources: pandas, numpy, sklearn, scipy, matplotlib, NLTK Corpus, keras, cv2(open cv), urllib
[13] Neural network diagram: NN-SVG