An Implicit Feedback Model for Goodreads Recommendations

Samantha Lee
skl384@nyu.edu
Center for Data Science, New York University

***Abstract*— Recommendation systems are one of the most popular applications of machine learning technologies, and are commonly used to personalize suggestions for users. In this project, I utilized the Spark engine to build and evaluate a model based on the Goodreads dataset and the ALS algorithm to recommend books to users according to implicit feedback. To further extend the model, I also visualized clusters of the books using the fuzzy genres metadata and the t-SNE algorithm.**

I. INTRODUCTION

A perfect example of the application of data, recommendation systems provide easy, personalized suggestions for customers, suited to their tastes and needs. One particular strategy for recommendation systems is known as collaborative filtering[1]. This method relies on past user behavior, and identifies relationships between the user and their interactions with certain products in order to find associations. These associations can be used for like-minded users with similar features to discover unknown relationships between the users and items.

In practical situations, recommendation systems utilize implicit feedback to determine a user's preferences. This can include purchase or browsing history, mouse movements, or search patterns. Although it is not as clearly defined as explicit feedback, implicit feedback is more readily available and accurately reflects the opinions of a specific user.

To build the model, I utilized the alternating least squares (ALS) algorithm, which is an iterative technique that fits the rating matrix R as the product of user-item embedding matrices - in our case the user-book matrices U and V. After initializing each set, at each iteration, one matrix is fixed and the other is solved using least-squares:

$$(R^*, V^*) = argmin_{(R,V)} \|R - U^T V\|^2 + \lambda(\|U\|^2 + \|V\|^2) \quad (1)$$

The updated factor matrix is then held constant to optimize the former matrix. This process is repeated until convergence.

The hyperparameters are the rank of U and V, the regularization parameter λ, and a constant alpha. These are used to tune the model in Section 2.

II. IMPLEMENTATION

A. Dataset

Each row of the dataset represents an interaction and contains the user id, book id, rating, booleans for whether or not the book has been reviewed or read. There are 876K total users and 223M user-book interactions.

B. Preprocessing

Due to the limited computing resources, there was significant preprocessing done to efficiently run the model. Any books that have not been read or were not reviewed were removed from the total dataset, and any user who read less than 10 books was removed. I also excluded ratings with a value of 0, since it corresponded with having no review. Then the data was downsampled to include only 25% of the total interactions, and the users split into 60% training, 20% validation and 20% testing. I made sure to include all users that ended up in the validation and testing sets in the training set as well, by taking half of those users' interactions and adding them into the training data. This guarantees that we do not have unseen users in the validation and test sets. See Figure 1 for an overview of the records in each dataset, respectively.

Dataset	Interactions	Users
Train	2, 916,694	177,866
Validation	379,967	36,257
Test	369,833	35,782

Fig. 1. Dataset Overview

C. Modeling

From Spark's machine learning library MLlib, the ALS method was used to estimate the user-item matrices. After fitting the model, the recommendForUserSubset method was used to return the top 500 predicted items for each user. Another DataFrame was also produced consisting of the actual rank of items for each user, ordered by the count. These two DataFrames were joined and converted to a resilient distributed dataset (RDD), which became the input of the RankingMetrics method. This produced the values for the three metrics that evaluated the recommendations, so the best fit can be chosen for the test set.

D. Hyperparameter Tuning

There are different hyperparameters that were tuned on the validation set to create optimal performance: rank, regParam, and alpha. For the ALS model specifically, rank represents the number of latent factors, or features in the lower dimension that have been projected from the user-item matrix[2]. The regParam specifies the regularization parameter, used to avoid overfitting. Alpha is applied to the implicit feedback variant of ALS; it governs the baseline confidence in preference[3].

They are assessed using mean average precision, precision at k, and normalized discounted cumulative gain as shown in Section 3.

III. Evaluation

First, we define M^4 as the number of users $U = [u_0, u_1, ...u_{M-1}]$. Each user ($u_i$) has a set of N ground-truth relevant documents $D_i = [d_0, d_1, ...d_{N-1}]$ and a list of Q ordered recommended documents based on the relevance $R_i = [r_0, r_1, ...r_{Q-1}]$. The metrics used to evaluate here compare the true relevant items with the recommended items to determine the effectiveness of the model.

A. Metrics

Mean Average Precision (MAP)

This is a measure of how many of the predicted items will appear in the true relevant item set. Since MAP accounts for the order of the recommender, the nuances here are that it will impose a penalty for relevant items that were not recommended with high relevance by the model. For M users and Q top recommended items, MAP is calculated by:

$$MAP = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{\mid D_i \mid}\sum_{j=1}^{Q}\frac{rel_{D_i}(r_j)}{j} \tag{2}$$

$$\text{where } rel_D(r_i) = \begin{cases} 1, r \in D \\ 0, \text{ otherwise} \end{cases} \tag{3}$$

Precision at k

To account for the penalization of MAP, precision is also used to evaluate the model. This is a measure of how many of the predicted items are relevant and appear in the item set, regardless of the ordering of both the predictions and the item set. For M users, precision is defined by:

$$p(k) = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{k}\sum_{j=0}^{min(|D|,k)-1} rel_{D_i}(R_i(j)) \tag{4}$$

Normalized Discounted Cumulative Gain (NDCG)

NDCG is computes how relevant the recommendations are, when normalized between users. We consider the cumulative gain, which sums up the ratings up to a specific rank position k: $CG_k = \sum_{i=1}^{k} rel_i$ We then consider the discount D by dividing the cumulative gain by the rank k: $DCG_k = \sum_{i=1}^{k}\frac{rel_i}{log_2(i+1)}$ To normalize, we divide the DCG by the ideal DCG, which is the discounted cumulative gain of the best possible results based on the ratings: $IDCG_k = \sum_{i=1}^{|REL_k|}\frac{2^{rel_i}-1}{log_2(i+1)}$ The final calculation for NDCG is:

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{5}$$

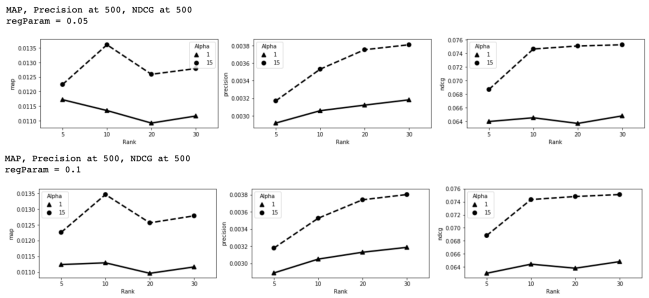

Fig. 2. Evaluation Metrics

MAP	Precision at 500	NDCG at 500
0.01262	0.003846	0.075214

Fig. 3. Evaluation on Test Set

B. *Final Results*

I performed grid search over the following hyperparameters to achieve the ideal combination to train on the final model. See Figure 2 for the evaluations based on each setting.

- regParam: [0.05, 0.1]
- rank: [5, 10, 20, 30]
- alpha: [1, 15]

The best model on the validation set, based on precision at k and NDCG, is achieved with rank = 30, regParam = 0.05, and alpha = 15. We see a MAP of 0.012786, precision score of 0.003809, and ndcg of 0.075277 in the holdout set with these hyperparameters. Note that the best MAP is found with rank = 10, regParam = 0.05, and alpha = 15, but it was not used since the other metrics had matching best hyperparameters.

After evaluating our model on the test set, we see the results shown in Figure 3.

The scores are similar to that of our holdout set, which indicates that the model does not overfit the training data. With more time and computing resources, I could improve the evaluation metrics by tuning on more combinations of hyperparameters and include a larger sample of data while training.

IV. Extension

To extend the baseline model, I developed a set of visualizations using t-SNE; the goal was to depict clusters of the users and books using the fuzzy genres[5] metadata. Taking the best model with the rank of 30, I applied the t-SNE algorithm to map

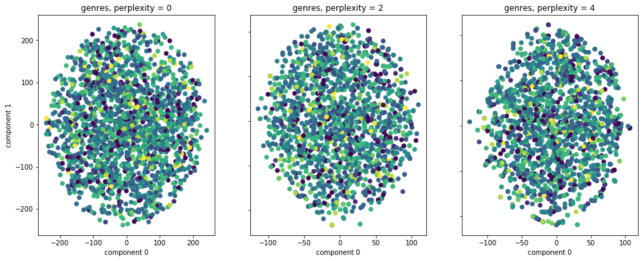

Fig. 4. Learned item factors colored by genre

the data into two dimensions, and color based on the genres from the metadata. The genres metadata includes multiple possibilities that each book could be categorized as, based on tags extracted and matched from users' popular shelves, so I assumed that the highest value of genre for a certain book equated the most matches and considered that the specific genre for each book.

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear dimensionality reduction algorithm used for exploring high-dimensional data[6]. It find patterns by identifying observed clusters based on the similarity of data points within multiple features; however the input features no longer become identifiable as the multi-dimensional data is mapped into a lower dimensional space. The algorithm can be tuned using perplexity, which is a guess about the number of close neighbors each point has[7], and can balance the attention between local and global aspects of the data. The main use case for t-SNE is for exploratory data analysis and as an input parameter for further classification and clustering tasks.

Figure 4 is a plot of the low-dimensional representation of the item latent factors, colored by the genres in the metadata, as well an extra category if no specific genre was found. There is no visible clustering in the genres at each perplexity value. This could be due to the fact that the item latent factors are both positive and negative in the model, which consequently cancel each other out and make it difficult for t-SNE to make sense of.

In Figure 5 and 6, I further explored the hypothesis above by including only non-negative latent item factors, and only negative latent item factors respectively. We can see the embeddings contribute to the representation individually and tend to cluster to opposite portions of the plot

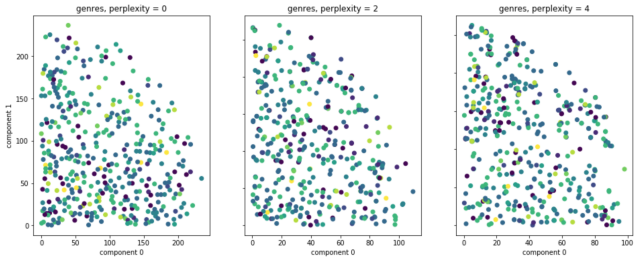

Fig. 5. Learned non-negative item factors colored by genre

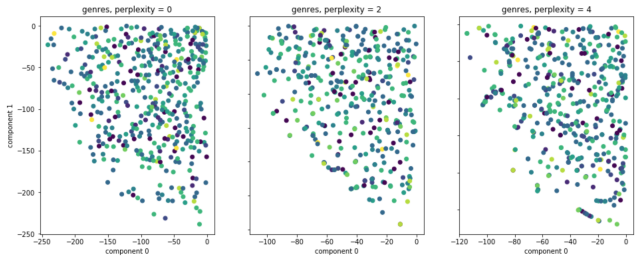

Fig. 6. Learned negative item factors colored by genre

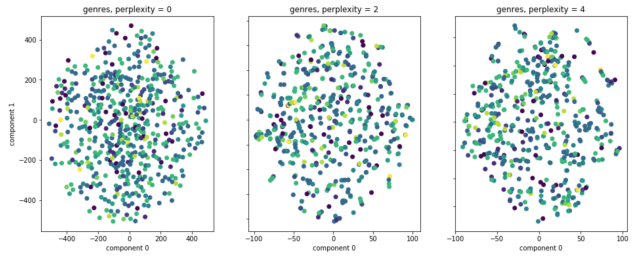

Fig. 7. Learned user factors colored by genre

depending on the scale. Although there are still no definitive clusters that can be identified, there is a visible impact of the latent factors' magnitude on the plots.

Figure 7 is a plot of the low-dimensional representation of the user-latent factors, colored by the 10 genres and if there was not one genre found. We see more sparsity as the perplexity increases, but there is still no identifiable clustering, most likely due to the positive and negative user latent factors. I posit the visualizations will have more scale and patterns will be able to be identified if there were only non-negative or only negative latent factors.

V. Discussion

The final ALS model is fit with hyperparameters rank = 30, regParam = 0.05, and alpha = 15. The achieved evaluation metrics are MAP = 0.01262, precision at 500 = 0.003846, and NDCG at 500 = 0.075214. This was done post downsampling, preprocessing and hyperparameter tuning. In the extension, the t-SNE algorithm is applied to attempt to identify clusters of the item and user latent factors based on the genre.

The limitations of building this recommendation system were due mostly to the constraints of running jobs on the HPC Cluster. If possible, I would have liked to include a higher percentage of the total data in the model, and conduct a more thorough hyperparameter grid search to achieve better results. I would have also liked to further explore the visualizations with t-SNE, and experiment with the latent factors and metadata to deliver some clustering in the plots.

References

[1] Y. F. Hu, Y. Koren, C. Volinsky, Collaborative Filtering for Implicit Feedback Datasets, IEEE International Conference on Data Mining (ICDM 2008), IEEE, 2008.

[2] Sammut C., Webb G.I. Latent Factor Models and Matrix Factorizations. Encyclopedia of Machine Learning. Springer, Boston, MA, 2011.

[3] Collaborative Filtering - Spark 2.4.5 Documentation. Spark.apache.org. https://spark.apache.org/docs/latest/ml-collaborative-filtering.html

[4] Evaluation Metrics - RDD-based API - Spark 2.4.5 Documentation. Spark.apache.org. https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html

[5] Mengting Wan, Julian McAuley, "Item Recommendation on Monotonic Behavior Chains", in RecSys'18.

[6] Saurabh Jaju, Comprehensive Guide on t-SNE algorithm with implementation in R Python. 2017. https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/

[7] Wattenberg, et al., "How to Use t-SNE Effectively", Distill, 2016. http://doi.org/10.23915/distill.00002