DSGA1004: Book Recommendation Engine

Anjali Agrawal
New York University
NetID: aa7513

Praxal Patel
New York University
NetID: psp334

1 Introduction

Recommendation systems aim at providing the most relevant information to a user by discovering patterns in the dataset. In an era where there is an abundance of advertisements of a vast range of commodities directed towards the user, providing personalized recommendations based on the user's past behavior is of prime importance. The project aims at recommending books to the readers based on their prior experience and feedback on the books they read. We predict the recommendations using Apache Spark ML's implementation of Alternating Least Square. We employ The Goodreads Interaction dataset which has 228,648,342 distinct interactions. Owing to the space constraints for implementation, we have utilized NYU Dumbo cluster and Google Cloud Platform(Challita et al., 2018) for data modelling and hyper parameter tuning.

2 Data Processing

The datasets used for the implementation are:

- Goodreads_interactions.csv
- Book_id_map.csv
- User_id_map.csv
- goodreads_interactions_children.json (Goodread, 2017)

We used goodreads_interactions_children.json for our local implementation. The difference between the full dataset and the subset is the value of user_ids and book_ids. The goodreads_interactions_children.json dataset had the actual user_ids and book_ids in the string format which we mapped using the Book_id_map.csv and User_id_map.csv.

The steps we followed for the preprocessing of our final data is as follows: We removed users that had less than 20 interactions and rows with book_ids that were occurring less than 15 times. We also observed users that had read the books (is_read = 1) and still rated the book 0. Considering the rating 0 as being not rated, we removed all the rows that had ratings = 0.

2.1 Data Splitting

For preparing the data for our data modelling step, we extracted all the user ids from our filtered data. In the first step, we randomly sample 60%, 20%, 20% users for the training, validation and test sets. We then joined the training, validation and test sets with the filtered data based on the user ids.

In the second step, we sorted the validation and testing sets on the basis of the user_ids and provided additional column rows numbers. Subsequently, we extracted interactions that had odd row numbers from the validation and test sets and added back those interactions to our main training set after dropping the column row numbers. The remaining interactions formed our final validation and testing sets respectively. We stored the resulting files in parquet format for more efficient access.

We down-sampled the data to 5%, 2%, 1% and 0.1%. The down-sampling was performed based on the user_ids i.e. We extracted 5%, 2%, 1%, 0.1% users from the full dataset and performed the steps mentioned above.

3 Models and Recommendation

3.1 Spark ALS

For our data modelling part, we use Apache Spark ML's implementation of Alternating

Least Square. The implementation tries to describe the user(readers) and books by a small set of latent factors. The rank of these latent matrices is extremely low in comparison to the actual user-item matrix. The methodology tries to recreate the actual matrix by alternatively minimizing the Least Square loss for the item and user matrix respectively. In the process, it gains the ability to predict the missing values in the matrix. The hyper-parameters for the ALS algorithm that we have used are Rank of the latent matrices, Maximum Iterations (fixed to 10 because of memory constraints), and Regularization. After training the model using ALS, we find the top 500 predictions using the trained model for the validation set. We filter those predictions in a way such that we don't recommend the books already read by the user again to make diverse recommendations. Therefore, we remove the predictions of books already read by the user and recommend the remaining books. Then, we evaluate the model using the metric for different hyper-parameters to obtain the best model. We report the test results using this model. We also obtain the user factors and items factors to obtain the latent features of the model for analyse what the model is learning.

3.2 Evaluation Techniques Used

Precision at K (paK): It is the ratio of the items in top-K recommendations that are relevant to the value of K. For instance, for our implementation, after extracting the top 500 predicted recommendations, if we obtain 20 documents that are present in the validation set (relevant documents), we say that the precision at K is 20/500. Precision at K is similar to the normal precision evaluation metric but for recommendation systems it is important to have relevant items at a higher position so we set a cutoff at the value K. However, the metric does not take into consideration the order of our 500 recommended items. This drawback is addressed by the evaluation metric MAP.

Mean Average Precision (MAP): For recommendation systems, it is of prime importance that the relevant items are present higher in the list of recommended items. In Mean Average Precision we use a weighted sliding window. Every time we encounter a relevant item, we extract the sub-list till that item and compute the precision of that sublist. We continue doing so till we encounter the last relevant item from our predictions. For our task, the final MAP value is the average of product precision values of all the sublists and the reciprocal of total relevant items. Thus, for our task, on recommending books, we will get a higher MAP value if the relevant books appear higher in the list of recommended items. One drawback of MAP is it performs better with binary data but fails to perform well on numerical data as it does not take into account the ratings (1-5) of the recommendation.

Normalized Discounted Cumulative Gain (NDGC): NDCG takes into account graded relevance values of the items. Thus, it is better at handling the evaluation of the ranked items. One issue we may face in evaluating NDCG at 500, is that the user might have consumed less than 500 books in the validation set. In real life scenario, we can deal with this issue by padding the remaining values with the minimum scores (ie. 1 in our case). Thus, NDCG takes into account that Highly relevant documents (books that are highly rated by the user) are more important than moderately relevant and irrelevant documents.

3.3 Hyper-parameters

- **Rank of the latent matrices:** Rank of the reader and book matrix from which we try to recreate the sparse observations. The rank of matrix controls the model complexity and can be tuned to overcome underfitting or overfitting. Moreover, it also depends on how much space we want to allot to our matrices.

- **Regularization Parameter:** Regularization parameter is a hyper-parameter that aids in increasing model generalizability and thus control overfitting.

For hyper-parameter tuning, we fix the value of seed to 42 for reproducible results and fair comparison. For the fixed value of maxIter i.e.

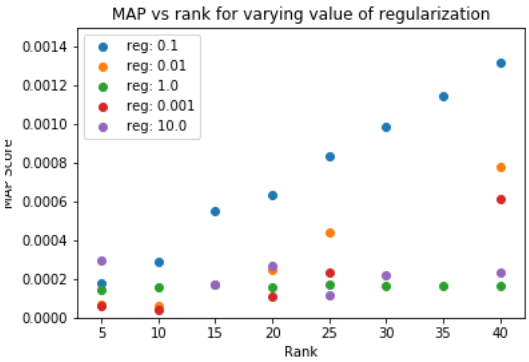

Figure 1: MAP scores of ALS with varying hyper-parameters for 5% data

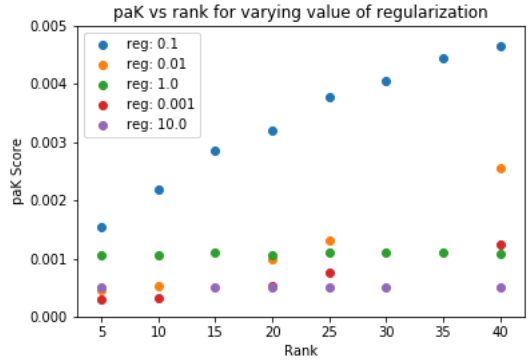

Figure 2: paK scores of ALS with varying hyper-parameters for 5% data

10, and for each version of down-sampled data (5%, 2%, 1% and 0.1%), we vary the rank of the latent matrices from 5-40 with a step size of 5 and we vary the regularization parameter λ from 0.001-10[1] increasing in multiples of 10. We tune these hyper-parameters for our model by evaluating the learned model on the validation set using the evaluation metrics described in the above section. After analysis of the evaluation metrics for all the subsets of data, we observe that the best performance in terms of MAP is generally achieved for the rank of 40 and when the regularization parameter is 0.1. The general trend we observe is that the performance of the model in terms of most of the metrics increases with the increase in rank. This can be observed in Figure 1, 2, 3.

The results for paK, MAP, and ndgc, on the validation set, and the held-out set (test set),

[1]For some ranks, we were unable to obtain results for some values of the regularization due to unavailable resources.

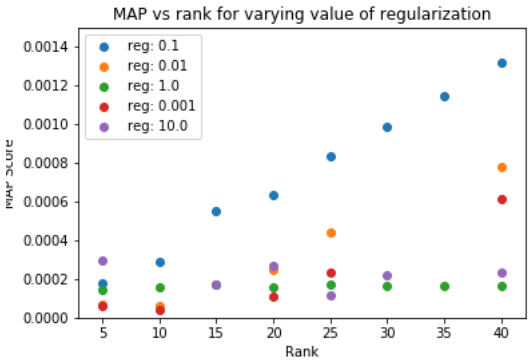

Figure 3: NDCG of ALS with varying hyper-parameters for 5% data

Data Percentage	MAP	paK	NDCG
5% data	0.001509	0.027251	0.004867
2% data	0.008140	0.088262	0.013464
100% data	3.48e-5	9.09e-5	1.25e-5

Table 1: Evaluation Results for ALS on the test set

for the our best model are outlined in Table 1[2].

4 Extension

In this section, we use the LightFM(Kula, 2015) package to give recommendations and compare the performance of the single-machine implementation (LightFM) with the ALS model we trained in the previous section. We aim to measure the efficiency gains achieved from this method both in terms of training time as well as the resulting accuracy of the recommendations.

4.1 LightFM

Content-based models and collaborative models are used widely and both of them have their pros and cons. LightFM(Kula, 2015) is a hybrid based approach that gets the best of both worlds by uniting the benefits of the content-based models and the collaborative-filtering approaches. This recommendation algorithm is basically a hybrid matrix factorisation model that represents the users and the items as the

[2]The values for 100 percent data are for rank 10 and regularization parameter 1. We were not able to tune on the full dataset because of the lack of available resources on Dumbo. Therefore this value is not indicative of the best value

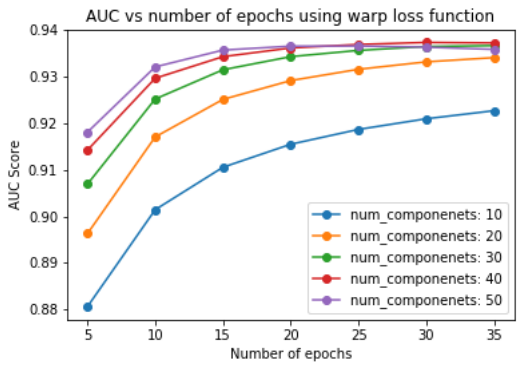

Figure 4: Performance of LightFM with varying hyper-parameters for warp loss

linear combinations of the latent factors of the content based features.

4.2 Experimental Setup & Results

For our experiments to analyse the best model using LightFM(Kula, 2015; Rubtsov et al., 2018), we have used three loss functions- warp(Rubtsov et al., 2018), bpr(Rubtsov et al., 2018), logistic(Rubtsov et al., 2018). The description of these three losses are outlined below-

- **WARP(weighted approximately ranked pairwise):** The loss function randomly samples output label pairs until it finds a pair that was wrongly labelled. WARP(Rubtsov et al., 2018) is extremely memory efficient. Thus, in the process of random sampling, it will keep a note about how many output label pairs it extracted before finding an incorrect pair and will multiply this value to the loss. Loss in our case would be the difference in ratings.

- **BPR(Bayesian Personalized Ranking):** Bayesian Personalized Ranking(Rubtsov et al., 2018) optimization criterion involves pairs of items(the user-specific order of two items) to come up with more personalized rankings for each user.

- **Logistic Loss:** More desirable when Positive and Negative interactions both are present.

Model	paK	Training Time	Evaluation Time
LightFM	0.050646	17.659	2.093
ALS	0.013057	17.045	17.647

Table 2: Comparison of ALS & LightFM

The hyper-parameters for the LightFM algorithm that we have used are the number of epochs and number of components. For each of these losses, we vary the number of components from 10-50 with a step size of 10 and we vary the number of epochs from 5-35 with a step size of 5. We tune these hyper-parameters for our model by evaluating the learned model on the validation set using three metrics supported by LightFM- area under curve (AUC), precision at K (pak) for 500 predictions, and recall. We also evaluate the model in terms of training time and evaluation time using 0.1%, 1%, and 2%, of the original dataset. The best performance in terms of AUC is achieved using warp loss when the model is trained for 20 epochs with 50 components. The general trend we observe is that the AUC increases both with the number of epochs and the number of components. However, the curve starts flattening with an increasing number of epochs. This can be observed in Figure 4. Surprisingly, for 2% data, the time taken to converge was relatively less than that on 1% data. Moreover, it obtained relatively higher accuracy on 2% data. The reduction in time might be due to the way the users were sampled. The performance results for logistic loss and bpr loss can be found in Figure 5 and Figure 6 which is outlined in the Appendix

4.3 Comparison to ALS

Next, we perform the comparison of LightFM to Spark's parallel ALS algorithm in terms of precision at K (paK) for 500 predictions on the held-out set (test-set), training time, and the evaluation time on 2% of the data. To make this comparison, since we have different hyper-parameters for both the algorithms, here, we just compare the performance of the best models we obtain after hyper tuning on the validation set. The paK values, training time, and the evaluation time, for both the models are outlined in Table 2.

For the hyper-parameters searched in our experiments, we observe that LightFM with nearly the same time gives much higher results for precision at K in comparison to ALS. In fact, even for other configurations of the LightFM which takes very less time to train in comparison to ALS achieves higher quality results than the ALS algorithm. Moreover, the computation time to get precision at K for LightFM is much faster than ALS. This shows the efficiency gains that are achieved using LightFM in terms of accuracy or relevance of the predictions as well as the training time.

5 Contribution

- Praxal Patel(psp334): Data Preparation and Data Filtering, LightFM (Extension), report, proofreading, ALS implementation, Hyperparameter Tuning
- Anjali Agrawal(aa7513): Data Preparation and Data Filtering, LightFM (Extension), ALS implementation, Hyperparameter Tuning, report, proofreading

References

Stéphanie Challita, Faiez Zalila, Christophe Gourdin, and Philippe Merle. 2018. A precise model for google cloud platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 177–183. IEEE.

Goodread. 2017. UCSD Book Graph. https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home. Last Accessed: 2020-05-08.

Maciej Kula. 2015. Metadata embeddings for user and item cold-start recommendations. In *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.*, volume 1448 of *CEUR Workshop Proceedings*, pages 14–21. CEUR-WS.org.

Vasiliy Rubtsov, Mikhail Kamenshchikov, Ilya Valyaev, Vasiliy Leksin, and Dmitry I Ignatov. 2018. A hybrid two-stage recommender system for automatic playlist continuation. In *Proceedings of the ACM Recommender Systems Challenge 2018*, pages 1–4.

A Appendices

Figure 5 and 6 specifies the performance of LightFM for different loss functions.

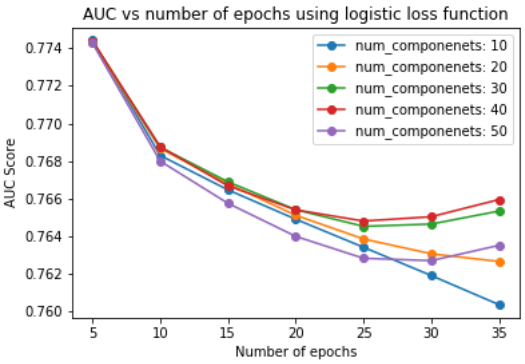

Figure 5: Performance of LightFM with varying hyper-parameters for logistic loss

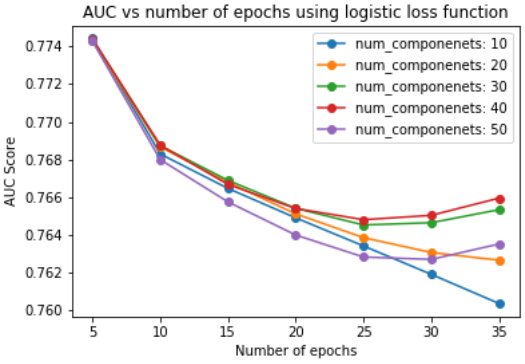

Figure 6: Performance of LightFM with varying hyper-parameters for bpr loss

B Reproducibility

All the code for the implementation of the project is available on Github.