DS-GA 1004: Final Project

Annika Brundyn (ab8690)
Camille Taltas (ct2522)

Introduction

In this project we build and evaluate a basic recommender system in order to recommend books to readers. The recommender system we use is collaborative filtering. This model trains from a user-item matrix, which provides information about each reader and their interactions with the books they read. The specific collaborative filtering method we use, alternating least squares, then decomposes the user-item matrix into a product of a user matrix and item matrix which represent lower dimensional latent factors. This algorithm then uses this decomposition and runs itself in a parallel fashion in order to produce a ranking of each unread book for each user and recommend the highest ranking ones.

In order to build this recommender system, we first tune two hyperparameters, namely dimension and regularization, using a validation set. Once we find the optimal set of hyperparameters, we run our trained model on our test set and evaluate its performance.

This second goal of this assignment is to compare a single machine implementation to the parallel implementation of ALS. The single machine implementation we chose was lightfm.

Implementation

Dataset

We use the Goodreads dataset collected by Wan and McAuley. This dataset contains Goodreads user-book interactions that show which users have read, rated and/or reviewed each book. In this dataset there are 876K users, 2.4M books, and 223M interactions.

Data Preprocessing, Splitting, and Subsampling

In our data preprocessing we excluded the users who had less than 10 interactions as they did not have enough data for us to make meaningful predictions, especially after splitting them between the validation, test, and training dataset. In order to split our dataset into train, validation, and test, we used 60% of the filtered users with all of their interactions in the training dataset. We used 20% of the remaining users for the validation set and put half of their interactions in the training dataset. Similarly, we used the last 20% of the users in the test dataset and added half of their interactions to the training dataset. This allowed us to have at least 5 interactions for each user we are predicting for in the pretrained models. Additionally, we converted all of the csv files to parquet in order to speed up the computations. Lastly, we ran all of our work on subsamples of 1% and 5% of the data as getting results was extremely slow or our jobs were dying due to memory overloads and having small comparable results was more important to us than one large result in order to understand the learning curve of our model.

Training and Tuning the parameters of the ALS model

Our recommendation model uses Spark's alternating least squares (ALS) method to learn latent factor representations for users and books. We tune two hyperparameters of this model, namely:

- Rank (dimension) of the latent factors, and
- Lambda, the regularization parameter

The rest of the parameters were kept at the default values for Spark ML ALS.

Evaluation

We explored a number of evaluation metrics, including mean average precision and normalized discounted cumulative gain on a 1% subset of the data but ultimately, the evaluation metric we chose for our data was precision at k, specifically k = 50. We chose k = 50 since given the small sizes of our subsamples, the number of true relevant books for each user that appeared in the validation or test set was most likely closer to 50 than to 500 and using the standard 500 would have given us a greater error than we actually had. The reason why we ran precision at k is that it allowed us to compare the first 50 books which are recommended rather than the whole set of recommended books. This was important as the number of books a reader reads is a lot smaller than the number of books evaluated by the model and potentially recommended. Thus, restricting this recommendation set for our evaluation makes it more accurate. Moreover, we did not want to give importance to ordering or ranking as we were forced to train and test on small datasets, which made ranking very variable and too complicated of a task to evaluate for the amount of information we were using. Lastly, using precision at k was convenient for our extension as we had to compare metrics and used precision at k for lightfm.

Training Results

We restricted our hyperparameter search to the following values on datasets of different size - rank: [5, 10, 20] and regularization strength [0.001, 0.01]. We observed that performance always increased as rank increased, but due to computational constraints where unable to try a rank value greater than 20. The tables are included in the appendix but overall we found our best model to have rank 20 and lambda 0.001. We chose the best model based on the precision at 50. Although we didn't have time to, we were curious to explore the effect of stronger regularization.

Although we experimented with various hyperparameters and dataset sizes, due to time constraint, we were only able to provide the results of our full hyperparameter grid search ran on downsampled datasets of size 1% and 5% of the full dataset. See appendix for result tables.

Evaluation

Evaluation of implicit-feedback recommender systems requires appropriate measures. We used the ALS recommendForUserSubset method to generate the top 500 recommended items. Again we looked at the validation and set precision at k for our best hyperparameter combinations (i.e. rank = 20; lambda = 0.0001).

Data size	**1%**	**5%**
Validation: Precision @ 50	0.00134548	0.0002730
Test: Precision @ 50	0.00082988	0.0002618

As can be seen from the table above the results from this analysis were very poor. This could be attributed to a number of factors - including the fact that the sample might be too small to extract meaningful recommendations, using a more appropriate evaluation metric, training the models for more epochs or searching for more suitable hyperparameter combinations. Given more time, we were most interested in exploring how designing custom metrics might lead to more appropriate results for our given dataset since all the ranking metrics explored led to quite extreme results.

Single Machine Extension:

For our extension we worked on implementing lightfm in order to compare Spark's parallel ALS to a single-machine implementation. Both models implement the collaborative filtering method in order to recommend books to users in that they use the user-item matrix as input and output a ranking of books most likely to have a positive interaction with the user. The goal of this implementation was to identify how the use of one machine compares to running on parallel machines both in terms of compute time and precision.

In order to make the comparison as accurate as possible. We first tuned the rank and regularization parameters by using scikit-learn's forest minimizer. Then we tested the time and precision for our test sets, given optimal rank and regularization parameters, and different subsample sizes of our data. We kept every other hyperparameter the same as they were defined for ALS.

The results we got were the following:

Data Size	**0.1%**	**0.5%**	**1%**
Time ALS (min)	2.3463	2.69306	4.656
Precision ALS	0.0057	0.00224	0.0013
Time Lightfm (min)	1.1300	13.74	49.64
Precision Lightfm	1.3e-4	2.3e-05	7.5e-06

Here, we can see that lightfm is faster for smaller datasets but ALS is much faster as the dataset increases in size. However, ALS is consistently more precise than lightfm.

To make this comparison more valuable it would have been interesting to look at at least 20% of the data. However, in terms of memory and speed for lightfm on our local machine this would have taken too much time and space.

As for our results, given that we ran similar optimizations and parameters, we can trust that our conclusions are accurate in that lightfm seems to be faster when the dataset is reasonably small but is very inaccurate. On the other hand ALS seems to be running exponentially faster in function of time and has reasonable precision. However, note that restricting the data to such small datasets makes our precision results very volatile and could arguably be considered insignificant.

Conclusion

In conclusion, our recommender system was not very successful. We did not manage to recommend books to users in a very accurate manner. One of our main considerations during this project, and a question that remains unanswered, is deciding on a more appropriate metric than precision at k for generating good recommendations. We also saw little to no improvement with our slightly bigger data sample which does not allow us to make positive predictions about what would have been given that we had more data. This is mostly due to the fact that in general, we were comparing data sets that were too small and so our results had little significance. With more time and better resources, our model could have been improved in a few key ways. First, we would like to have spent more time investigating custom metric functions that were more in line with our goal. Ideally, we wanted to train and run a hyperparameter grid search on the full training set instead of small subsamples. Additionally we would have searched over a broader range of hyperparameters and then narrowed once we found an optimal search range. Lastly, we would have been able to implement a better extension and gotten overall more significant results. Despite the poor performance of our model, this project helped us build a much more practical understanding of the considerations, constraints and complexities that working with very large sets involves.

Resources

Collaborative filtering:
https://www.ethanrosenthal.com/2015/11/02/intro-to-collaborative-filtering/
Goodreads dataset:
https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home
Evaluation metrics:
https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html#ranking-systems
Extension optimization:
https://www.ethanrosenthal.com/2016/11/07/implicit-mf-part-2/
Lightfm implementation:
https://making.lyst.com/lightfm/docs/quickstart.html

Contributions

Annika Brundyn: basic recommender system, writeup
Camille Taltas: extension, writeup

Appendix

1% of data

Rank	regParam	Precision at 50	Time elapsed (min)
5	0.001	0.0000981071058	3.443496724
5	0.01	0.0002382620883	4.080622169
10	0.001	0.0006447091801	4.082197350
10	0.01	0.0006166783462	4.072256514
20	0.001	0.0013454800280	5.084602592
20	0.01	0.0009250175193	5.078675601

5% of data

Rank	regParam	Precision at 50	Time elapsed (min)
5	0.001	0.00003940332	18.565657447
5	0.01	0.00005629046	18.135551223
10	0.001	0.00016042781	20.135086584
10	0.01	0.00010976639	20.135117023
20	0.001	0.00027300873	22.149682309
20	0.01	0.00025330706	22.783062001