Book Recommandation System Based on ALS and LightFM models

Guyu Liu
New York University
gl1813@nyu.edu

Naisheng Zhang
New York University
nz862@nyu.edu

1 Overview

Within only a few years, deep learning and machine learning techniques have started to dominate the landscape of algorithmic research in recommender systems. Novel methods were proposed for a variety of settings and algorithmic tasks, including top-n recommendation based on long-term preference profiles or for session-based recommendation scenarios. Given the increased interest in machine learning in general, the corresponding number of recent research publications, and the success of deep learning techniques in other fields like vision or language processing, one could expect that substantial progress resulted from these works also in the field of recommender systems.

1.1 Dataset

The datasets were collected in late 2017, details of the dataset are available on the goodreads Web at

https://www.goodreads.com/

UCSD only scraped users' public shelves, which was collected for academic use only. We use one of the genre subsets to run on our local machine and the entire dataset to run on the dumbo.

1.2 Models and evaluation

We use Spark's parallel ALS model and the single-machine LightFM model to build our recommander system.

We only use the RMSE to fine-tuning the ALS model because it can converge fast when we run our program on the dumbo, and we use the Ranking Metrics to build the precision@k, MAP, NDCG@k to evaluate the ALS model's performance.

For the LightFM model, we use the built-in function ROC_SCORE to fine-tuning our hyperparameter, we adopt the precision at k and ROC_SCORE to evaluate the model's final performance.

2 Data Processing

This part, we will talk about how We filter the data by interaction numbers and the ratings, and how we use the spark tools to split the data.

2.1 Data Filtering

Before the data splitting, we filter the users whose interactions are less than 10, we choose to ignore these users because they cannot provide sufficient data for evaluation, especially after partitioning data into train/test, we use spark to filter the user_id which number is less than ten and join these filtered users to original dataset in our experiment. In the local machine, we use the smaller dataset (goodreads_interactions_poetry.json) to run our model. And in that dataset, because the type of data

is string, we use the indexer and pipeline to convert the data into the numeric type. However the data type of the whole dataset in dumbo is numeric, so there is no need to convert the data type to numeric here.

2.2 Data splitting

We construct train, validation, and test splits of the data with a fixed random seed, so that our validation scores are comparable across runs.

First of all, we filter 60% of users and all of their interactions to form the training set, and 20% of users to form the validation and test set. Because the recommender system cannot predict items for a user with no history, so then for each validation and test user, we use half of their interactions for training, and the other half should be held out for validation(Actually 100% users to train).

Our method for splitting data is to select the distinct user firstly, and randomly select the user_id from the distinct user based on the percentage(60%, 20%, 20%), and join these selected users to the original dataset to split the train, validation and test by user_id. In addition, we use window function to add row number to every distinct user_id in validation and test dataset, and select odd row number to union with training data, which ensures the user_id in test and validation set also exist in the train set.

3 Models and Experiments

In this part, we use the 1%, 5%, 25%, 100% of the whole dataset to train and test. We compare and anlysis the result from different angles.

3.1 Introduction to ALS model

Consider a feature matrix $X \in \mathbb{R}^{m \times d}$ and target value then regularised least squares optimises $arg\,min = \|y - Xw\|^2$, One of the interesting property of linear regression is that the optimal solution w can be defined analytically as $w = (X^T X + \lambda I)^{-1} X^T y$ In multi-regression, we perform regression over multiple target, using the same feature matrix. The solution is given as $W = (X^T X + \lambda I)^{-1} X^T Y$, In general, we rarely get to use this elegant analytical solution as it involves inverting $d \times d$ matrix where d ranges from thousands to millions in practice. Note that inversion of a large matrix is not only computationally expensive but also numerically unstable. However, despite being rarely used in real world problems, matrix factorisation (MF) framework optimises is defined by the equation $arg\,min = \left\|R - U^T V\right\|^2 + \lambda(\|U\|_2^2 + \|V\|_2^2)$ where R is partially observed the user-item preference matrix $U \in \mathbb{R}^{k \times m}$, and $V \in \mathbb{R}^{k \times n}$, For rating prediction, the gradients wrt U and V are computed only using the observed entries. Typically, R is extremely sparse, and hence the algorithm scales linearly with the number of observed entries.

In One-Class CF(OC-CF) the user-item interaction consists of positive only preferences, such as item purchase. In this scenario, $R \in \{?, 1\}^{m \times n}$ where 1 indicates purchase and ? indicates unobserved entries, if we do SGD using the observed entries then we learn a useless model that always predicts 1. To address this, most OC-CF algorithms treat unobserved entries as 0, But we have too many entries in matrix for gradient computation which makes it extremely hard if not impossible in any real world dataset. This can be addressed by negative sub-sampling.

For the ALS algorithm, Repeat these steps: (a) Fix U and solve for V, (b) Fix V and solve for U, If V is fixed, the MF objective is equivalent to multi-regression problem with U as a feature matrix and R as a target, and the optimal value for V is given as $V = (UU^T + \lambda I)^{-1} UR$, here we can use the analytical solution as it involves inverting a small $k \times k$ matrix, and sparse matrix multiplication. Similarly, the solution for step (b) is given as: $U = (VV^T + \lambda I)^{-1} VR$, ALS is a big idea for Implicit/OC-CF setting. ALS can be applied for rating prediction because SGD scales very well in rating prediction setting.

3.2 Hyper-parameters

Besides the learning rate, we choose rank and regParam to be our hyper-parameters to fine tuning our model.

3.2.1 Rank

Rank refers to the presumed latent or hidden factors, for example, if you were measuring how much different people liked movies and tried to cross-predict them then you might have three fields: person, movie, number of stars. Now, let's say that you were omniscient and you knew the absolute truth and you knew that in fact all the movie ratings could be perfectly predicted by just 3 hidden factors, sex, age and income. In that case, the "rank" of your run should be 3, so we need to set enough hidden factors to fine-tuning our model.

3.2.2 regParam

regParam specifies the regularization parameter in ALS, it controls performance of the model fitting, it prevent the model from over fitting, and it also adjust the model to fit data in a good performance.

3.2.3 Evaluation Metric

We use RMS error metric and Ranking Metric on this model. RMSE is root mean square error, it has an equation that is

$$RMSE = \sqrt{\frac{\sum_{(i,j)\in Test} |r_{ij} - R_{ij}|}{|Test|}} \tag{1}$$

where r_{ij} is ground truth for rating and R_{ij} is the precision of the rating. Ranking metrics has three metric, precision@k compute percentage of relevant in top K, and ignores documents ranked lower than K, MAP is Average Precision across multiple queries/rankings, DCG is the total gain accumulated at a particular rank p, which is $DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2 i}$, NDCG Normalize DCG at rank n by the DCG value at rank n of the ideal ranking, The ideal ranking would first return the item with the highest relevance level, then the next highest relevance level.

3.2.4 ALS Model Implementation

Design Model: We include all range of the rating in the train and validation dataset. We found that the RMSE is much high. We think it is because when a book at rating <= 0, the book will be recognized as not read or not like, so it is a bias when we compute the RMSE. After drop them, the RMSE result is better enough. To avoid NaN in results, we set ColdStartStrategy to drop for ALS model.

Fine-tuning: Hyper-parameters are chosen as: ranks = [5,10,15,20,25,30], regParams = [0.01, 0.05, 0.1, 1]. The fine-tuning progress is in Table 1 with the whole dataset.

Best model: latent factors = 25, regularization = 0.1, RMSE = 0.825.

Evaluation: We use the ranking metrics to evaluate the model. At first, we found that the NDGC = 1.0. It is because the model does not care the sorted order and recognize all the book as the truth data. So we set a relevance threshold which means drop rows with the rating = 0, and solve the problem. In the Table 3, as we can see, the precision@k is low when k = 500. The reason here is that most of users chosen as test users have much less than 500 books in the list and the ground truth of books for each user could be much smaller than 500. Then we set k = 10, 100 and find that the precision@k increase with the k decrease, and we use the pyspark to select every user's book, we found that more than half of the user read less than 100 books. So those are the reasons why precision@k is low when k is large.

3.2.5 Comparison of Two Models:

In the respect of time, evaluation time will increase with the growth of the data set, especially in LightFM. If the size of data set is large, the training process of LightFM would be quite slow. ALS model's evaluation is faster. However, the speed of LightFM fitting is much faster than ALS. In the respect of accuracy, the LightFM's accuracy is comparable low, we believe that is because we did not give it enough features to evaluate, so matrix of test is too sparse to get a good performance.

4 Extension

In this part, we implemented a single-machine model named LighFM, and compare the result to ALS mdoel.

4.1 Introduction to LightFM Model

LightFM is a mixed content-collaborative model, it will make resemblance for FM, in this model, users and items was presented in embeddings, it combines the advantages from Filtering-based Recommendation and Content-based Recommendation, it can be used to cold start problem, and when we have warm-start, dense user-item matrix, it can perform as good as MF model, embeddings can be used to produce important features, which can be used in label recommending.

4.2 LightFM Implementation

Downsample: Firstly we use spark functions to filter data as ALS model. Then, we use to_pands function to convert the spark dataframe to pandas dataframe, but we find this process is too slow to finish and the program could throw OutOfMemoryError if using more than 5% of the whole dataset. So we decide to write the spark dataframe into parquet files, and then using the pandas to read the parquet to speed up the conversion. However, if we use the whole dataset, the fit and evaluation could be quite slow. In our trial, it cost us about 21 hours to fit the model. And evaluation could be much more time-consuming. So we downsample 25% of the dataset to train and evaluate.

Covert pandas dataframe to matrix: Normalize the rating label can help us to make the data matrices for train, test and validation into same shape, we use user_id and book_id as rows and columns to fill the ratings in the sparse matrices, and then we use an build-in function coo_matrix to convert pandas dataframes into matrix to fit our model. Then we filter the book_id in our pandas dataframe to ensure the number of distinct books is same in our three dataset.

Fine-tuning: We use the ROC score to fine-tuning our model, and the ROC score and precision@k to evaluate our model. Hyper-parameters are chosen as: learning_rates = [0.01,0.05,0.1], learning_schedules = ['adagrad','adadelta'].

Best model: When the learning rate =0.1 and the learning schedule is adadelta, the best score = 0.9108692407608032.

Evaluation: We find that precision@k is low and the ROC score is good, we compare the three different k in precision@k(See Table 4), we found that that is low when k is high, so we think it is because not every user would like to read more than 100 books, so the user who read less than 100 books will be a bias when we compute the precision@k when k is high. We compare the results of precision@k for k = 500, 100, 10 to ALS model, we find that ALS model performs better than LightFM model. It might be because the matrices is not very dense for model to make a good performance in this model.

5 Contribution of Team Members

We have two team members: Guyu Liu and Naisheng Zhang, both of us works from start to end, all the works is finished by two of us which includes: Data processing strategy, Algorithm design, model fine tuning, extension implementation, code writing, and report writing. We make same effort to solve the problem we met. Under the clear goal, our team members collaborate and assign responsibility to each member until we finish our total project. We are very appreciate our TA Junge Zhang and Weicheng Zhu helping us in this final project, they give us many useful information when we stuck in some problems.

Appendix

Table 1: ALS Validation

Hyperparameters		
Latent Factors	Regularization	RMSE
5	0.01	0.935
5	0.05	0.848
5	0.1	0.837
5	1	1.31
10	0.01	0.960
10	0.05	0.845
10	0.1	0.832
10	1	1.33
15	0.01	0.980
15	0.05	0.846
15	0.1	0.830
15	1	1.32
20	0.01	0.983
20	0.05	0.846
20	0.1	0.828
20	1	1.317
25	0.01	0.986
25	0.05	0.842
25	0.1	0.825
25	1	1.312
30	0.01	0.999
30	0.05	0.843
30	0.1	0.827
30	1	1.316

Table 2: ALS Evaluation

Metric					
Scale of Dataset	MAP	precision@10	precision@100	precision@500	NDCG@500
100%	0.991	0.976	0.470	0.139	0.995

Table 3: LightFM Validation

Parameters				
Scale of Dataset	Learning Rate	Learning Schedule	AUC Score	Best Model
25%	0.01	adagrad	0.820	
25%	0.01	adadelta	0.917	
25%	0.05	adagrad	0.917	
25%	0.05	adadelta	0.913	
25%	0.1	adagrad	0.935	True
25%	0.1	adadelta	0.917	

Table 4: LightFM Evaluation

Metric				
Scale of Dataset	percisoin@10	percisoin@100	percisoin@500	ROC score
25%	0.16	0.07	0.03	0.909