DSGA 1004 BIG DATA FINAL PROJECT

Jeewon Ha (jh6926), Anthony Lanzisera (acl673)

Introduction

Not only as relief for customers drowning in a sea of information but also as a covert collector of the user information, recommender system has become one of the most essential services that business should provide. In this paper, we emulate a glimpse of the technology using Spark ML's ALS module. Our goal is to evaluate the model performance and explore means to optimize it through hyperparameter tuning. We also plan to compare the model performance of ALS to that of lightfm.

We use the Goodreads dataset (Wan and McAuley, 2018) for creating train, validation, and test set. The dataset contains 3 files: 1) goodreads_interact.csv, 2) user_id_map.csv, and 3) book_id_map.csv. The first file, good_reads_interactions.csv, contains tuples of features representing user-book interactions: user_id, book_id, is_reading, rating, and is_reviewed. The second and third files, user_id_map.csv and book_id_map.csv, contain unique numeric identifiers and alphanumeric strings representing user and book object, respectively. Overall, there exist 876K users, 2.4 books, and 223M interactions.

Data Processing

A. Due to the computational limitation of the Dumbo cluster, we downsampled the user data by 2%; the downsampled data has 17479 users. For experiment reproducibility, random seed was set to have the value of 42.

B. Interaction data was inner joined with downsampled user data to filter unnecessary interaction records. Similarly, book data was inner joined with interaction data.

C. All three datasets were saved as parquets for efficient data accessing.

D. Users with less than 10 interactions were removed since they may not provide sufficient data for evaluation.

E. For training set, 60% of the users and all of their interactions were selected. For validation set, of remaining 40%, 50% of the users and all of their interactions were selected. For each validation user, half of his/her interactions was used for training, and the other half was used for validation. For test set, resting 20% of the users and all of their interactions were selected. Similar to the validation set, for each test user, half of his/her interactions was used for training, and the other half was used for test. Table 1 demonstrates the resulting composition of the training, validation, and test sets.

To replicate the data processing, refer to Train_Validation_Test_Datasets_Creation.py from the github.

Model

Alternating Least Squares (ALS) is a matrix factorization algorithm commonly used for collaborative filtering. The goal is to find optimal user matrix U and item matrix M by minimizing the following squared loss function:

$$L_\lambda(R,U,M) = \sum_{(i,j)\in I}\left(r_{ij} - u_i^T m_j\right)^2 + \lambda\left(\sum_i n_{u_i} \parallel u_i \parallel^2 + \sum_j n_{m_j} \parallel m_j \parallel^2\right);$$

$R \approx U^T M$, I = index set of known ratings in R, n_{u_i} = number of ratings by user i, and n_{m_j} = number of rating by item j (Winlaw et al., 2016). General procedure of the ALS can be summarized into two sequences. While U is held constant, $\frac{\partial L_\lambda}{\partial u_i} = 0$ ∀i, M is updated through gradient descent. Similarly, while M is held constant, $\frac{\partial L_\lambda}{\partial m_j} = 0$ ∀j, U is updated through gradient descent. Until convergence, these two sequences are repeated for each iteration. In this project, we evaluate the performance of Apache Spark ML's ALS module and optimize the validation accuracy by tuning two hyperparameters – rank and regularization parameter. Due to the large dataset size and computational cost, we bypass the cross-validation.

Rank is the number of presumed latent factors in the model. While higher rank may help with larger dataset, too high rank may overfit the model, degrading the validation accuracy. We attempted values including 10, 20, 30, 40, 50, and 100. Regularization parameter, lambda, is a constant used to penalize the loss function in order to avoid overfitting. Similar to rank, while higher lambda may help reduce overfitting, too high lambda may increase model bias due to bias-variance tradeoff. We attempted values including 0.001, 0.01, 0.1, 1, and 10.

The validation accuracy of the ALS model is measured using 4 different metrics 1) RMSE, Precision, MAP, and NDCG. Following definitions of the metrics are referred from (Evaluation Metrics - spark.mllib - Spark 1.6.3, 2020).

A. Root Mean Square Error (RMSE) represents the average squared difference between true rating and predicted rating. Let I be the index set of ratings in R.

$$\text{RMSE} = \sqrt{\frac{\sum_{(i,j)\in I}\left(r_{ij} - u_i^T m_j\right)^2}{|R|}}$$

B. Precision at K represents the number of top K recommended documents that are also the true relevant documents averaged across all users. For each user i, D_i be the set of N true relevant documents and C_i be the list of Q recommended documents. Suppose there exist M users.

$$\text{p(K)} = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{K}\sum_{j=0}^{\min(|D|,K)-1}\text{rel}_{D_i}(C_i(j)); \text{rel}_D(c) = \begin{cases} 1\ if\ c \in D \\ 0\ otherwise \end{cases}$$

C. Mean Average Precision (MAP) represents the number of recommended documents that are also the true relevant documents penalized by the order of the recommendations.

$$\text{MAP} = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{|D_i|}\sum_{j=0}^{Q-1}\frac{\text{rel}_{D_i}(C_i(j))}{j+1}$$

D. Normalized Discounted Cumulative Gain at K (NDCG) represents the number of top K recommended documents that are also the true relevant documents averaged across all users and penalized by the order of the recommendations. Suppose $n = \min(\max(|C_i|, |D_i|), K)$.

$$\text{NDCG(K)} = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{\text{IDCG}(D_i,K)}\sum_{j=0}^{n-1}\frac{\text{rel}_{D_i}(C_i(j))}{\log(j+1)}; \text{IDCG(D, K)} = \sum_{j=0}^{\min(|D|,K)-1}\frac{1}{\ln(j+1)}.$$

For precision and NDCG, K was set to have the value of 500.

Result

When choosing the optimal parameters, we prioritize the model performance on precision, MAP, NDCG, especially NDCG, since RMSE is solely based on explicit feedback while the other metrics depend on implicit feedback. Table 2 demonstrates validation accuracies of ALS depending on the rank values when regularization parameter is held at constant ($\lambda = 0.01$). While Rank = 10 reported the best RMSE, since Rank = 50 reported better precision, MAP, and NDCG, we choose Rank = 50 as the optimal value. Holding rank at optimal constant (Rank = 50), Table 3 demonstrates validation accuracies of ALS based on the changes in regularization parameter values. According to the same logistics used for optimal rank value selection, we choose $\lambda = 0.01$ as the optimal value. Therefore, Rank = 50 and regularization parameter = 0.01 is the optimal configuration for the ALS model. Test accuracy under optimal condition is reported in Table 4.

To replicate the experiment, refer to ALS.py from the github.

Extension

Topic: Comparison to single-machine implementations

LightFM is a hybrid recommendation algorithm based on both content-based filtering and collaborative filtering. Differentiating from ALS, LightFM utilizes latent representations of user u and item m, denoted as the sum of their features' embeddings, for model prediction. Let U be the set of users and M be the set of items. Each user u is associated with features f_u, and each item m is associated with features f_m. Using user and item feature embeddings, e^U and e^M, LightFM computes latent representations of the user u and item m as $q_u = \sum_{j \in f_u} e_j^U$ and $p_m = \sum_{j \in f_m} e_j^M$, respectively. The model prediction for user u and item m is $\hat{r}_{um} = f(q_u \cdot p_m + b_u + b_m)$, where f is the sigmoid function, and b_u and b_i are user and item feature biases. Based on the given information, LightFM aims to find feature embeddings (e^U, e^M) and feature biases (b^U, b^M) such that the following likelihood function is maximized:

$$L(e^U, e^M, b^U, b^M) = \prod_{(u,m) \in S^+} \hat{r}_{um} \times \prod_{(u,m) \in S^-} (1 - \hat{r}_{um})$$ (Kula, 2015).

In this section, we evaluate the performance of LightFM using Weighted Approximate-Rank Pairwise (WARP) loss and compare the model performance between Spark ML's ALS module and LightFM. WARP loss maximizes the rank of positive items by sampling negative items at random until a violating negative item is found. Then, it performs gradient update; the amount of update is inverse-proportional to the number of samples took to find the violation. The procedure is repeated until the satisfactory condition is met. More information about LightFM can be found at https://making.lyst.com/lightfm/docs/home.html.

When choosing the optimal parameters, we prioritize the model performance on precision since it is the common evaluation metric between ALS and LightFM. Table 5 demonstrates validation accuracies of LightFM depending on the number of components when the number of epochs is held at constant (# Epochs = 10). Since # Components = 30 reported best precision, recall, and AUC, we choose it as the optimal value. Holding the number of components at optimal constant (# Components = 30), Table 6 demonstrates validation accuracies of LightFM based on the changes in the number of epochs. According to the same logistics used for optimal component size selection, we choose # Epochs = 30 as the optimal value. Therefore, # Components = 30 and # Epochs = 30 is the optimal configuration for the LightFM model. Test accuracy under optimal condition is reported in Table 7.

Comparing the model performance between ALS and LightFM under their optimal configurations, LightFM has much lower precision than ALS. However, in terms of efficiency, as shown in Table 8, ALS

reported about 1467% increase in train fitting time, LightFM reported only 357% increase in train fitting time when the dataset size doubled.

To replicate the experiment, refer to LightFM.py from the github.

Contribution Statement

Jeewon: ALS model, LightFM model, Hyperparameter tuning, Report generation.

Anthony: Dataset creation, ALS model, LightFM model, Hyperparameter tuning.

Reference

Evaluation Metrics - spark.mllib. https://spark.apache.org/docs/1.6.3/mllib-evaluation-metrics.html (accessed May 11, 2020).

Kula, M. Metadata Embeddings for User and Item Cold-Start Recommendations. Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems 1448, 14–21.

Mengting Wan, Julian McAuley, "Item Recommendation on Monotonic Behavior Chains", RecSys 2018

Safdari, N. If You Can't Measure It, You Can't Improve It!!! https://towardsdatascience.com/if-you-cant-measure-it-you-can-t-improve-it-5c059014faad (accessed May 11, 2020).

Winlaw, M.; Hynes, M. B.; Caterini, A.; Sterck, H. D. Algorithmic Acceleration of Parallel ALS for Collaborative Filtering: Speeding up Distributed Big Data Recommendation in Spark. 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS) 2016.

Appendix

A. Tables

	Users	Interactions
Training	9170	~ 3680035
Validation	3057	~ 460004
Test	3057	~ 460005

Table 1. Composition of training, validation, test sets.

Rank	RMSE	Precision	MAP	NDCG
10	2.15169	0.22369	0.98522	0.99676
20	2.19004	0.22373	0.98516	0.99683
50	2.17181	0.22379	0.98537	0.99693

Table 2. Validation accuracy of ALS based on rank; $\lambda = 0.01$.

λ	RMSE	Precision	MAP	NDCG
0.001	2.79123	0.22358	0.98506	0.99671
0.01	2.17181	0.22379	0.98537	0.99693
0.1	1.77100	0.22379	0.98517	0.99682

Table 3. Validation accuracy of ALS based on regularization parameter; Rank = 50.

	RMSE	Precision	MAP	NDCG
Rank = 50, $\lambda = 0.01$	2.14167	0.22027	0.98302	0.99723

Table 4. Test accuracy of ALS based on optimal configuration.

Number of Components	Precision	Recall	AUC
10	0.03235	0.35553	0.93535
20	0.03450	0.38110	0.94482
30	0.03700	0.40607	0.95058

Table 5. Validation accuracy of LightFM based on the number of components; # Epochs = 30.

Number of Epochs	Precision	Recall	AUC
10	0.03700	0.40607	0.95058
20	0.03975	0.44589	0.95575
30	0.04160	0.46162	0.95640

Table 6. Validation accuracy of LightFM based on the number of epochs; # Components = 30.

	Precision	Recall	AUC
# Comp = 30, # Epoch = 30	0.04371	0.45239	0.95662

Table 7. Test accuracy of LightFM based on optimal configuration.

	1%	2%
ALS	170s	2495s
LightFM	103s	368s

Table 8. Train fitting time of ALS and LightFM based on dataset size under optimal configuration.

B. Supplementary link

Github repository for the project: https://github.com/nyu-big-data/final-project-dsga1004_jh_al