DSGA 1004 Final Project

Bofei Zhang(bz1030), Shaoling Chen(sc6995), Yuxuan He(yh2857)

In this project, we built a recommendation model using Spark's alternating least squares (ALS) method and tuned the hyper-parameters to optimize performance on the validation set together. On top of the baseline collaborative filter model, we implemented three extensions: alternative model formulations, fast search, and cold start strategy.

Model Implementation

We used three datasets in this model: training, validation, and testing data. Each contains a table of triples (user_id, count, track_id) which measure implicit feedback derived from listening behavior.

First, we transformed the user and item identifiers of three datasets into numerical index representations. Then we downsampled the training set with a sampling rate of 0.02 to more rapidly prototype the model. An ALS model was used to learn latent factor representations for users and items.

Next, we turned the model's hyper-parameters, including the rank of the latent factors, the regularization parameter, and alpha, the scaling parameter for handling implicit feedback data, to optimize the Mean Average Precision(MAP) on the validation set. The following figures show the results of different configurations and through grid search. The best performance model has parameters [rank = 10, alpha = 0.1, reg = 0.01] and it reaches MAP of 0.0138.

Rank	Alpha/Reg	0.01	0.1	1
5	0.1	0.01346348184	0.008197601807	0.006334559663
5	1	0.009677114249	0.009743009	0.007064216155
5	5	0.009066671005	0.009164012665	0.00902802082
10	0.1	0.01384084308	0.01033918896	0.005907380837
10	1	0.009608153937	0.009416008091	0.007292397102
10	5	0.00846588898	0.008718621844	0.00914954526
20	0.1	0.01011272356	0.008883673076	0.003975521849
20	1	0.008426382166	0.008192677335	0.006799765655
20	5	0.007513700891	0.007775421753	0.007722513322
50	0.1	0.00664	0.006468278138	
50	3	0.006586653149	0.006793327703	
80	0.1	0.005131349198	0.005146981024	
80	3	0.006339025046	0.00628977159	

Note: Please refer to the code 'preprocess.py', ''train.py', 'evaluate.py' and 'baseline.sh ' for implementation. Due to the limitation of the resources, we evaluated the grid-search models separately rather than using single for loop.

Evaluation results

After grid search, we trained the model (rank 10, alpha 0.1, regularization parameter 0.01) on the whole training set and evaluated it on the test set using the ranking metrics. Precision on predictions of the top 500 items for each user is 0.00679346, which takes 571.997 seconds.

Extensions

1. Alternative model formulations (extension 1.sh, preprocess.py)

We adopted two different modification strategies: log compression and dropping low count values and evaluated their efficiency and impact on overall accuracy.

1.1 Log compression

We transformed the count data into $log(count + 1)$ and performed grid search on 2% training set again. The following results show the best configuration will be rank 10, reg 0.01, alpha 0.1.

Rank	5	10	15
MAP	0.013463481837255155	0.013840843083129712	0.010966944943705897

After training on the whole training set, we obtained the best model for log compression with precision for 500 items of 0.00665964, which takes 536.68 seconds. Compared to the best baseline model, the model with log transformation takes a shorter time in training session slightly but the performance is still quite similar.

1.2 Dropping low count values

We dropped records in training set with count value lower than 1 and performed grid search on 2% training set again. The following results show the best configuration will be rank 5, reg 0.01, alpha 0.1.

Rank	5	10	15
MAP	0.0122603820542006	0.010907520634974277	0.009713541076264539

It takes 489.217 seconds to get the best model, with precision at 500 of 0.004899372076062402 on the test set. Although it saves training time to use drop data with low count value, it threatens the prediction results of the model greatly. Looking into the data, we found that the average count was just 2.88 for the whole dataset. If we dropped records with count no more than 1, only around 40.60% data remained. Thus it's better not to drop low count values for this dataset.

2. Fast Search (extension 2)

One of the key factors for Collaborative Filtering is searching for similar users or items given a set of dense and real-value vectors. In this section, we studied the performance gains provided by the spatial tree implementation from Annoy (Approximate Nearest Neighbors Oh

Yeah) and graph-based algorithm from NMSLIB (Non-Metric Space Library) compared to a brute-force method.

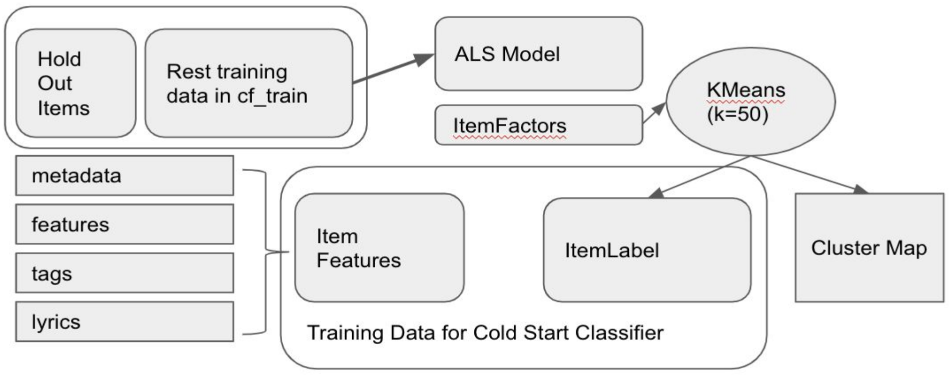
Firstly, the user factor matrix was obtained from the best model evaluated from the test set. There are a total of 1129318 users, and each of them is represented by a vector with dimension 10. The Euclidean Distance is used as metrics for the similarity of vectors. For the brute-force method, we first computed the Euclidean distance between a giver user vector and all other users. Then, we sort the distance descendingly and obtains top K users with the closest distance with the given user vector. The sorting algorithm guarantees $O(nlog(n))$ complexity. The results from the brute-force method are treated as ground truth to evaluate the performance of the spatial tree. Then, we evaluated the spatial tree algorithm such as Annoy and graph-based algorithm such as hnsw (NMSLIB), sw-graph (NMSLIB). Their hyperparameters are tuned to see how the performance will change.

	Query 1/s For top 100	Query 1/s For top 500	Recall at 100	Recall at 500
Brute Force	4.99	2.43	1	1
Annoy (Tree = 10)	46	31	0.92	0.95
hnsw (W = 40)	618628	423667	0.7966	0.3198
sw-graph (NN = 40)	108492	92426	0.9964	0.8816
Annoy (Tree = 50)	16	9.7	0.9986	0.999
hnsw (W = 80)	344359	381994	0.8012	0.3477
sw-graph (NN = 40)	38346	46417	1	0.99476

We randomly sampled 50 user vectors and running similarity search on them. Then, we computed the average recall and elapsed time over these 50 searches. The recall is computed by the following equation: $Recall = \frac{True\ Positive}{True\ Positive + False\ Positive}$. For Annoy, the number of spatial trees can be tuned. It is the slowest one among three methods. Increasing the number of spatial trees will lead to better recall but slower query speed. A larger number of trees also means longer time to build trees and more space to store the index. According to the NMSLIB documentation, there exists a perfect value for M and NN for hnsw and sw-graph respectively. Increasing them to a certain degree will improve recall and shorten retrieved time at the cost of indexing time. For hnsw, this algorithm is fastest among the three algorithms. Larger M means slower speed and slightly higher recall. For sw-graph, it is slower than hnsw but much faster than Annoy. Increasing NN improves recall and slows down retrieved time. In experiments, sw-graph takes the longest time to build the index.

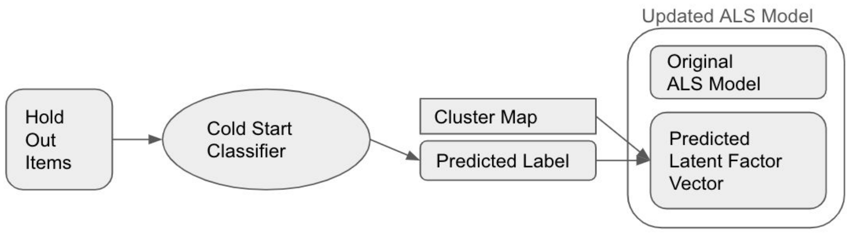
3. Cold Start (extension3)

Since our goal is to build a classifier that could predict item latent factor vector using item features, the implementation contains three steps: obtain data, train classifier, and performance evaluation.



In the first step, we obtained feature variables for items by processing and joining the four supplementary datasets. Then, to obtain the target value for items, we trained the latent factor model (using the best parameters found in the previous section) on full training data less the holdout items and converted the itemFactors parquet in the learned model to the target variables. Since spark MLlib does not support learning task with the 10-dim target variable, we reduced the 10-dimension item factor vector to 1-d target variable using k-means cluster model. we collected 50 clusters on itemFactors data and use the learned cluster label to represent each item's latent factor vector. Finally, we merged the feature variables and target variables to build the final dataset.

In the second step, we built and tested different classifiers performance and different parameters to obtain the best classifier. MLlib supports logistic regression, decision trees, and random forests for a multi-class classification task. we tested the model performance for logistic regression and random forests and random forests reached higher accuracy. Then we tuned hyperparameter for treeDepth and numtree using cross-validation to obtain the best set of parameters. [treeDepth=20, numTree=50]



Finally, we used the classifier we obtained in the step above to make predictions for items in the holdout dataset and then transformed the categorical prediction to latent factor vectors using the saved k-means cluster centers. Then we obtained an updated ALS model by combining the original ALS model with the predicted values we get and run evaluation tests on two ALS models. The updated model's MAP metrics improved from 0.0297529 to 0.02997290, which indicates that our cold start classifier slightly improves prediction accuracy.

4. Contribution

We worked together on building the baseline recommendation system, each implemented one extension: alternative model formulations(Shaoling Chen), fast search(Bofei Zhang), and cold start strategy(Yuxuan He) and then wrote the report together.