Final Project Report for DS-GA 1004 Big Data

Nimi Wang (nw1212), Ruoyu Zhu (rz1403)

May 18, 2019

1 Introduction

This project report is dedicated to construction and implementation of recommendation models using Spark's Alternating Least Squares (ALS) method. The objective of the recommendation system is to learn latent factor representations for users and items, and finally give top 500 recommendations of song tracks for each user.

2 Data Description

Here is a brief summary of the dataset we used for our recommendation system. There are three sets of data we used:

- cf_train.parquet
- cf_validation.parquet
- cf_test.parquet

The schema for these files is as follows:

- schema='user_id STRING, count LONG, track_id STRING, __index_level_0__ LONG'

Specifically, each file contains a table of triples(user_id, count, track_id) which measure implicit feedback derived from listening behavior. The first file cf_train contains full histories for approximately 1M users, and partial histories for 110,000 users, located at the end of the table. cf_validation contains the remainder of histories for 10K users, which is used for validation set to tune the hyper-parameters. cf_test contains the remaining history for 100K users, which is used for final evaluation.

3 Baseline model

3.1 Data Transformation and Handling

DATA TRANSFORMATION VARIABLES

We transform the training and validation data with StringIndexer transformer. We use pipeline before the for loop which is used for hyper-parameter tuning for faster computation.

- *user_id_indexed* : user_id after StringIndexer transformation, which transform DataFrame columns from type STRING to type FLOAT
- *track_id_indexed* : track_id after StringIndexer transformation, which transform DataFrame columns from type STRING to type FLOAT

PREDICTION VARIABLES

- *userRecs*: use *.recommendForAllUsers(500)* to get the top 500 recommendations for each user. The returned result includes both the recommended tracks column and confidence score column for each track, arranged in a descending order.
- *pred_list*: select the recommended tracks column for '*userRecs*' and renamed it as '*pred_list*'

3.2 ALS model and Evaluation Metrics

The method we use for the recommendation system is collaborative filtering, in which users and song tracks are described by a small set of latent factors that can be used to predict missing entries. In our model, we use alternating least square (ALS) algorithm to learn the latent factors.

For evaluation, we choose to use Mean Average Precision (MAP) and precision (at 500) as our ranking metrics. We collect song track ids for each user on validation set using *GroupBy* and get the variable '*val_groupby*'. Then the ground truth song tracks for each user is the concatenated list of track ids we get from the previous step. By mapping the ground truth list and our prediction result to a tuple '*predLabelsTuple*' , we then can compute the MAP and the precision for each user.

3.3 Hyper-parameter Tuning and Evaluation Result

There are three parameters we need to tune in our ALS model to optimize performance on the validation set, notably:

- the **rank** (dimension) of the latent factors
- the **regularization parameter** λ , which is used to prevent over-fitting, and
- **alpha**, the scaling parameter for handling implicit feedback (count) data.

The set of hyper-parameters we choose are shown in the table below:

rank	[4, 6, 8]
regularization Parameter λ	[0.01, 0.105, 0.2]
alpha	[0.5, 1.25, 2]

We build a parameter grid for the three hyper-parameters. By using the built-in function *itertools.product*, we get 27 combinations of the (rank, regParam, alpha) and feed each combination into the ALS model.

After tuning on the validation set, we get the most optimal hyper-parameters setting: rank = 8, regParam = 0.01, alpha = 2.0, which resulted the highest recorded MAP score = 0.151515 on validation set and MAP score = 0.142532 on test set. The precision score on the validation set is 0.002 and 0.00183 on the test set.

4 Extension

We choose alternative model formulations as our extension model. We modify the ***count*** data to alternate the ALS model. We conduct both log compression modification and drop-low-values modification on the count data.

4.1 Log Compression

We apply natural log function on the '*count*' feature in the training dataset. By using the build-in function *.withColumn*, we add a new feature call '*log_count*' to the replace '*count*' feature.

With the optimal hyper-parameter setting, our model achieves an MAP score = 0.163912, precision = 0.00184 on test dataset using the same optimal hyper-parameters we get in the baseline model. The MAP improves 15% and precision improves 0.87% by applying log compression.

4.2 Drop Low values

We filter out the lowest 0.1% of the count value by using the build-in function *.approxQuantile*. i.e. If the count is smaller or equal to 1, we eliminate it from the training dataset.

With the optimal hyper-parameter setting, our model achieves an MAP score = 0.1733889, precision = 0.00221 on test dataset using the same optimal hyper-parameters we get in the baseline model. The MAP improves 22% and precision improves 21.1% by dropping the low count values.

	MAP	precision at 500	MAP improvement	precision improvement
baseline	0.142532	0.00183	NA	NA
log	0.163912	0.00184	15%	0.87%
drop low count	0.173889	0.00221	22%	21.1%

5 Summary

We use ALS model for the construction of our recommendation system. As we can see from the results above, the baseline model achieve a decent MAP and precision score on the test set with best parameter setting. With further pre-processing on the "count" column by either log-compression and dropping values lower than the lower 1% of the data, we achieve higher MAP and precision scores. This shows that these pre-processing is rewarding in terms of getting more accurate recommendations for users.

However, due to computation constraint, the hyper-parameters are not trained fully and intensely. If given more resources, the model could have found a better hyper-parameter setting which might achieve a higher MAP and precision score.

Note that we use baseline_for_loop.py for training and test.py for testing. For extended modifications on "count" column, we used extensions.py. For more details about the exact implementations, please refer to `github.com/nyu-big-data/final-project-team-tw`.

Collaboration Statement:
Nimi Wang implemented the recommendation model and Ruoyu Zhu improve the extension model. For the writing part, Ruoyu Zhu focused on the introduction, methodology and extension of the model. Nimi Wang focused on results and analysis.