Music Recommender System with Implicit Feedback

Nhung Le (nhl256), Shirley Xu (xx852), Hengyu Tang (ht1162)

May 18, 2019

1 Implementation

1.1 Transformation

In order to develop the music recommender system, we used Spark's Alternating Least Squares (ALS) method to learn latent factor representations for users and items. In order to develop a baseline ALS model, we first transformed the data in the following steps:

- Drop the unnecessary column *__index_level_0__* for storage efficiency.
- Down-sample the training data to 6 millions records, which include all the user IDs that are in the validation and test set and approximately 10% of the other records in the original training data file.
- Transform the user and item identifiers (strings) into numerical index representations for Spark's ALS model. The inclusion of the last 110K records of partial history ensures us to have string representations for all the users in the validation and test set. Additionally, we integrated *metadata.parquet* file provided to index every track using *track_id*.
- Re-partition our down-sampled training data with respect to *user_label* for 5000, and re-partitioned validation and test data with respect to *user_label* in order to store intermediate data for training models.
- Build model using the *pyspark.ml* package. Given counts as implicit feedback, we set our *ImplicitPrefs* as *True* to better accommodate our model settings.

1.2 Hyper-parameter Tuning

After running a baseline model, we started tuning some hyper-parameters to optimize performance on the validation set. These hyper-parameters are 1) the rank (dimension) of the latent factors, 2) the regularization parameter, and 3) alpha, the scaling parameter for handling implicit feedback (count) data. We used **Mean Average Precision (MAP)** from *pyspark.mllib* package on 100 recommended tracks for validation users to evaluate the model performance. The ranges chosen for the hyper-parameters are $[0.1, 0.2, 0.4]$ for *alpha*, $[10, 20, 40, 60]$ for *rank*, and $[0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$ for *regularization parameter*. Part of our hyper-parameter tuning results are shown in the table 1 below. The complete hyper-parameter tuning results are attached in the appendix.

Rank	Alpha	Regularization Parameter	Validation MAP
10	0.1	0.1	0.027830808611560166
20	0.1	0.1	0.03203915601058416
40	0.1	0.5	0.014768179078776377
60	0.2	0	0.030856037736286174
60	0.2	0.1	0.03709295725994317
60	0.2	0.5	0.02342892689475009

Table 1: Selected Hyper-parameter Tuning Results

2 Evaluation Results

The default setting model before tuning has **MAP = 0.018402243511908077**. The best configuration recorded after tuning is **rank: 60, alpha: 0.1, reg: 0.1, MAP=0.03894831062718834**. We further tested our best baseline model on test set to recommend 500 tracks for each user and calculated the MAP: **MAP = 0.0367593620020977**. From the hyper-parameter tuning results shown in table 1, we can see that:

- For the same rank and alpha, the models with regularization of 0.1 outperform the models with regularization of 0 and 0.5.(rank 60 and alpha 0.2)
- Rank seems to play the most important role as for the same alpha and regularization. The models with higher rank significantly outperform the other models with lower rank. Therefore, the more number of latent factors we use, the more likely the recommended tracks are relevant to the users (i.e., tracks recommended are those that users actually listen to).

3 Extensions

3.1 Alternative Model Formulations

We performed 2 modification strategies: log transform counts, and drop lower counts. The core concept for implicit feedback, *count* in our dataset, is that it represents confidence level rather than explicit user preferences. The large counts do not necessarily represent users' higher preference. We conducted the following manipulations on our training data, leaving validation and test data intact as the baseline for comparison purpose.
For each method, we trained the model under the same validation metrics (MAP for 100 recommended tracks for each user). The last 3 models conducted small tuning on each parameter by changing 1 parameter at a time while fixing the other two parameters:
1). Baseline model using default setting: reg = 0.01, rank = 10, alpha = 1; 2). Model with best configuration from baseline tuning: reg = 0.1, rank = 60, alpha = 0.1; 3).reg = 0.01, rank = 10, alpha = 0.1;4).reg = 0.01, rank = 60, alpha = 1;5).reg = 0.5, rank = 10, alpha = 1.

- **Log Transformation**: We conducted the log transform using the following formula:

$$r_{ij} = log(1 + c_{ij})$$

c_{ij} represents the original counts of user i over track j. The addition of bias 1 is to prevent scenarios where original count is 1, and will be considered non-interactions after performing log

Rank	Alpha	Regularization Parameter	Validation MAP
10	1	0.01	0.027122468883603246
60	0.1	0.1	0.04611925243846317
10	0.1	0.01	0.025222959506148925
60	1	0.01	0.031225145871043823
10	1	0.5	0.024573955573833302

Table 2: Hyper-parameter Tuning Results for Log Transformation

transformation($log(1) = 0$). We transformed the training set prior to fitting the model, chose the best configuration using validation data, and tested the best model performance using test data. The tuning results are shown in the table 2 above: the best validation performance is achieved at rank = 60, alpha = 0.1, reg = 0.1. The test performance recommending 500 tracks is **MAP= 0.04949524135836173**. Coincidentally, we achieved the best performance using the same model configuration as our best baseline model. In general, we can see that with increased ranking number, the effects of boosting MAP score is obvious and significant. It is reasonable as we are offering model more space to represent each user and track. We also observed that the log-transformed models respond less sensitively to increased ranks. We also witness a decrease in MAP validation performance when we lower down alpha, which is the confidence parameter the ALS model time with the implicit feedback counts. With smaller alpha, we weaken the contribution of each counts to make final predictions. Given the same training size, it is plausible that with restricted information feed, we will not get the same performance as before. In general, after log transformation, we can see obvious improvements in both validation and testing model performances. It is likely due to the fact that we compress the count value ranges, hence alleviating the extreme influences of very large counts and very small counts. A very large count may not indicate that the user prefers this particular tack as it could be an outlier. Log transformation focuses more in the middle range of counts, which is more likely to be real reflection of user behaviors and does help the model recommend relevant tracks to users.

- **Dropping Low Count Values**: We further dropped the lower counts for our already-down-sampled training sets. We believe that by dropping only the lower counts in our baseline training data, we will be able to make reasonable comparisons with baseline model performances. After several tests, we found out that there are only 2,450,810 out of 6M records that have count > 1. By dropping count as low as 1, we will prune almost 60% of the original training data. Moreover, we found out that by pruning low count 1, we are still able to include all test and validation users. The tuning results using validation data are shown in the table 3 above: the best model is achieved at rank 60, alpha 1, and regularization parameter 0.01. We further tested the best dropped model using test set and the test **MAP = 0.03266172154133633**. It is also obvious that the parameter settings is different from our best baseline model, validating that model configuration differs greatly with different training settings. We observe that with increased rank numbers, the model performs better, given that we have more freedom to encode the user behaviors and track features. When the regularization parameter increases, we can still witness a small increase in MAP albeit slight. With decreased alpha, we can witness a drop in MAP, as we are putting fewer weights on the count, hence weakening the contributions of each count to depict users. Combined with

Rank	Alpha	Regularization Parameter	Validation MAP
10	1	0.01	MAP = 0.021792298643234374
60	0.1	0.1	MAP = 0.02571965352995248
10	0.1	0.01	MAP = 0.020040483588673594
60	1	0.01	MAP = 0.030236041450228034
10	1	0.5	MAP = 0.022659684151149665

Table 3: Hyper-parameter Tuning Results for Dropping Low Count Values

smaller data sizes, weakened contribution of counts provides less information to the model.

In general, compared with baselines, the MAP score after dropping lower counts decreases. Possible contributing factors include the significant shrinkage of the training samples, hence the data may not be able to provide enough information to fully represent users and tracks in the factor space. Additionally, different from explicit feedbacks, low counts do not necessarily represent dislikes of users to this particular track. It is possible that a user favored a track but lost the track file after one interaction, which is very hard to be captured. Therefore, dropping lower counts does not provide much help in predicting the user preferences better.

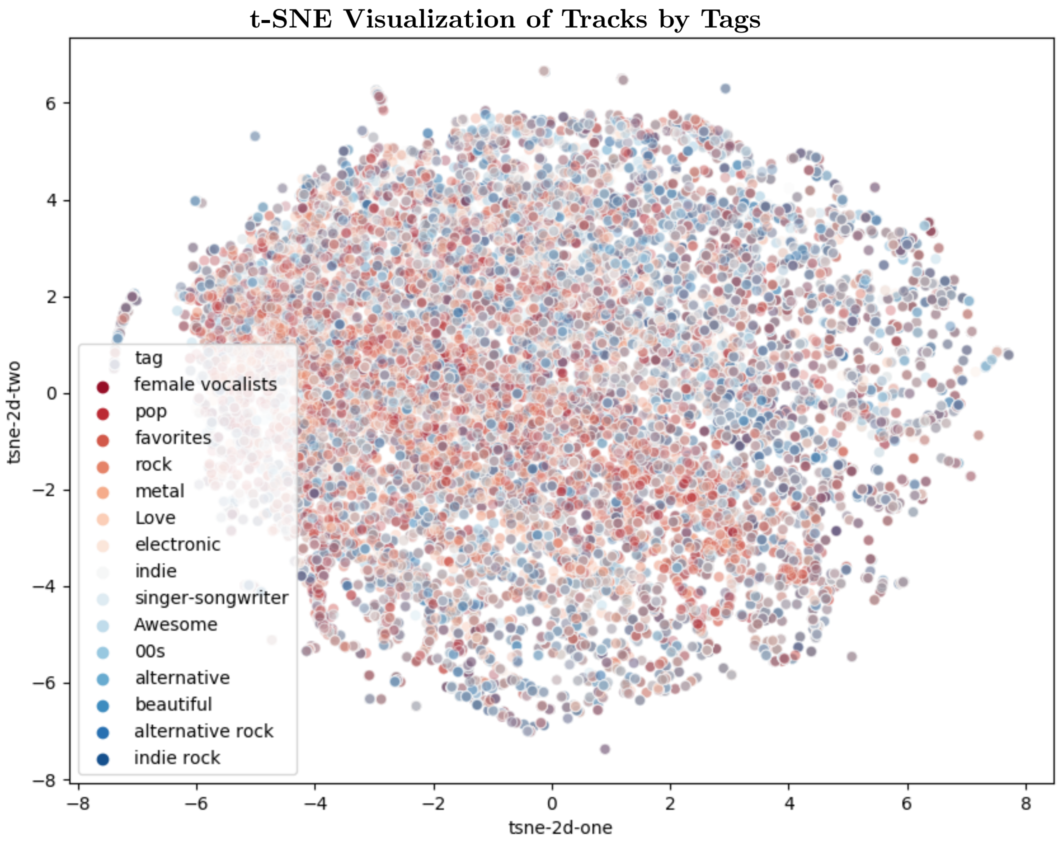
3.2 Exploration

After implementing the Alternating Least Squares model, we proceeded to visualize the learned representations of the items using t-Distributed Stochastic Neighbor Embedding (t-SNE), a technique for visualizing high-dimensional data. Specifically, we visualized the learned latent factors of tracks from the fine-tuned baseline model. The optimal model has the parameter rank of 60, so each track is represented by a 60-dimensional vector in the latent factor space. The t-SNE algorithm embeds the 60-dimensional feature vectors in a 2-dimensional space and models each track by a point in such a way that the similarity between two tracks is modeled by the distance between the two corresponding points.

For visualization, we read the 10 Parquet files for the item factors learned by our optimal baseline model and then employed the Scikit-Learn implementation of the t-SNE algorithm to visualize the tracks represented by 60-dimensional vectors in a 2-dimensional space. In addition, we incorporated the tags information for each track in the visualization.

Since there are 522,366 unique tags in the dataset and a number of tags only apply to very few tracks, we selected the most popular 15 tags for visualization. The selected tags are representative in terms of the popularity, but the set of 15 tags only covers approximately 8% of the tracks in the dataset. In the 2D scatterplot, each track is colored by its respective tag. Since each track is likely to have more than one tag in the 15 tags of interest associated with it, points with different colors representing the same track could stack together in the 2-dimensional space.

The figure below is the t-SNE visualization of a subset of the tracks (part 0) in the learned latent factor space. The complete visualization is available on GitHub. The figure shows how the tracks of different tags are distributed in the learned space. In the scatterplot, the tracks are not obviously grouped together with their respective tags, but some patterns are clearly identifiable: tracks of “rock” and “metal” tend to be close in the learned space; tracks of “awesome”, “00s”, and “alternative” share certain similar factors as well since they are not too far away from each other in the learned space.



4 Collaboration Statement

- Nhung Le: Developed ALS model; Developed MAP evaluation; Tuned 1/3 of the hyper-parameter sets; Worked on Exploration extension.
- Shirley Xu: Developed ALS model; Subset and transformed train/validation/test data; Tuned 1/3 of the hyper-parameter sets; Worked on the Exploration extension.
- Hengyu Tang: Developed ALS model; Subset and transformed train/validation/test data; Tuned 1/3 of the hyper-parameter sets; Worked on the Alternative Model Formulation extension.

5 Appendix

The table below shows the complete hyper-parameter tuning results.

Rank	Alpha	Reg Param	Validation MAP	Rank	Alpha	Reg Param	Validation MAP
10	0.1	0	0.02496960683810706	10	0.2	0	0.026629069504452943
10	0.1	0.01	0.024976450965516355	10	0.2	0.01	0.025527879997634298
10	0.1	0.05	0.027559081607352008	10	0.2	0.05	0.027149015436463256
10	0.1	0.1	0.027830808611560166	10	0.2	0.1	0.02721577527808156
10	0.1	0.5	0.010868089975593477	10	0.2	0.5	0.015238430929050573
10	0.4	0	0.027555906152516404	20	0.1	0	0.026119697318329197
10	0.4	0.01	0.02646446382237515	20	0.1	0.01	0.025720089803117293
10	0.4	0.05	0.027099456882029783	20	0.1	0.05	0.02994246377767248
10	0.4	0.1	0.0271174602685097	20	0.1	0.1	0.03203915601058416
10	0.4	0.5	0.020046176230368536	20	0.1	0.5	0.014146936514576283
20	0.2	0	0.027555906152516404	20	0.4	0	0.02983572693841939
20	0.2	0.01	0.026466099469753018	20	0.4	0.01	0.028248948309939285
20	0.2	0.05	0.02935040362306495	20	0.4	0.05	0.02933781538927694
20	0.2	0.1	0.031377086243036444	20	0.4	0.1	0.03072501203882944
20	0.2	0.5	0.01638932582737609	20	0.4	0.5	0.02382288806151633
40	0.1	0	0.027116821149713925	40	0.2	0	0.029401407683277255
40	0.1	0.01	0.025757053213276825	40	0.2	0.01	0.027470659657536877
40	0.1	0.05	0.03218147053670535	40	0.2	0.05	0.0313945584450698
40	0.1	0.1	0.03574152131784452	40	0.2	0.1	0.03445899296739989
40	0.1	0.5	0.014768179078776377	40	0.2	0.5	0.020517998791758123
40	0.4	0	0.0323216097750259	60	0.1	0	0.028272115342033322
40	0.4	0.01	0.031058351172231293	60	0.1	0.01	0.026285129360736455
40	0.4	0.05	0.032130188230165765	60	0.1	0.05	0.03425935649006713
40	0.4	0.1	0.034401920952167486	60	0.1	0.1	0.03894831062718834
40	0.4	0.5	0.029675602413815907	60	0.1	0.5	0.01593698030087145
60	0.2	0	0.030856037736286174	60	0.4	0	0.024072332837863076
60	0.2	0.001	0.028933596043486096	60	0.4	0.001	0.03243451964254617
60	0.2	0.005	0.03329280728484411	60	0.4	0.005	0.034173316728372045
60	0.2	0.1	0.03709295725994317	60	0.4	0.1	0.03672794317500746
60	0.2	0.5	0.02342892689475009	60	0.4	0.5	0.0327403899829543