Recommending Music with Spark ML

Implementing a large-scale song recommendation system for music streamers

Dee Ham

New York University Center for Data Science, drh382@nyu.edu

Kathleen Young

New York University Center for Data Science, ky2132@nyu.edu

In the world of big data, recommendation systems are an immensely important tool for introducing users to content that is both novel and matches their unique taste. There is simply too much content for a user to sort through themselves--whether it be music, TV shows, movies, tweets, YouTube videos, Tiktoks, etc. A good recommendation system can open up new worlds to users, exposing them to content they enjoy but would not have discovered on their own. In the following paper we outline our music recommendation system that harnesses the power of thousands of users' listening habits to generate song recommendations. The system is built using Spark ML's implementation of alternating least squares and is trained on the implicit music streaming preferences of 110,000 unique users. We also examine a single machine implementation of such a system using LightFM.

CCS CONCEPTS • Recommender System • ALS • Implicit Feedback Data

Additional Keywords and Phrases: Spark, LightFM, Million Song Dataset

ACM Reference Format:
Dee Hahm and Kathleen Young. 2021. Recommending Music with Spark ML: Implementing a large-scale song recommendation system for music streamers. New York, NY, USA, 5 pages.

1. INTRODUCTION

1.1 Overview

In 2021, the Recommender System is a part of daily lives for everyone whether they realize it or not. A recommender system typically recommends a new product to its user based on their own history and preference. There are many different algorithms to implement a recommender system. In this project, we specifically focus on Spark's alternating least squares (ALS) model to learn latent factor representations for users and items using the Million Song Dataset. Additionally, we explore LightFM, a single-machine implementation and compare efficiency and resulting accuracy against Spark ALS model.

2. DATA

2.1 Description of Data

The Million Song Dataset is a freely-available collection of audio features and metadata for a million contemporary popular music tracks [1]. The subset of data used in this project includes user_id, track_id, and count. This is implicit feedback data since there are no explicit ratings from users; rather the user's feelings about a song are implied based on the number of times a user listened to that song, which is represented as count. Data was pre-split into train, validation and test sets as three separate Parquet files.

2.2 Preprocessing Data

Prior to implementation, we preprocessed data in two different methods since using the whole dataset consumed too much time and resources. Per suggestion from Professor McFee, we filtered our data so that we only include users represented in the test dataset. This did not work well for the users in the validation dataset (there was no overlap between the test set users and validation set users); therefore, we filtered our data to include all users that are present in either validation or test data. Additionally, in order to use Spark's implementation of ALS, it was necessary to re-map the user and track IDs, originally alpha-numeric strings, to 32-bit integer values. We accomplished this through a series of Python scripts that created a new ID for each user and track and joined the new IDs to the training, testing, and validation data sets. We quality-checked the process by implementing a series of count queries to ensure no data was lost or mis-mapped. The newly mapped datasets were saved to our personal HDFS directories. Table 1 shows the unique number of users and songs for each dataset:

Dataset	Original data	After filtering on validation and test users
Train	Users=1,129,318 songs=386,213	Users=110,000 songs=163,206
Test	Users=100,000 songs=159,717	Users=100,000 songs=159,717
Validation	Users=10,000 songs=50,074	Users=10,000 songs=50,074

Table 1: The unique number of users and songs for each dataset

3. IMPLEMENTATION

3.1 Implementing Spark ALS

We trained our recommendation system using the alternating least squares class in Pyspark ML Recommendation (Spark version 2.4.0). Our data, the Million Song Dataset, was stored as parquet files in our personal HDFS directories and our model was run on the NYU Peel high performance computing cluster. The data was pre-split for us into validation, test, and train sets. We further filtered the training set to include only those users that appeared in the testing or validation set, as described above.

Step one: Training and hyperparameter search

The new training set, with 32-bit user and track IDs and filtered for test and validation users, was used to train an initial series of 24 models with unique sets of hyperparameters. These initial hyperparameters included rank (10, 50,100), the regularization parameter (0.001, 0.01, 0.1, 1), and the maximum number of iterations (10, 20). Each of the 24 models was applied to the validation data set and evaluated on a series of metrics, described below in section 4. We chose the best performing model based on mean average precision, and then continued the hyperparameter search with 8 additional models, tuning alpha (0.01, 0.1, 1, 1.5, 5) and trying more maximum iteration values (30, 50, 100), resulting in a total of 32 potential models to choose from. In this way, the hyperparameter search was a semi-greedy process because it started with a thorough search and then tested more hyperparameters on the best initial model. Checking every single potential model given the hyperparameter ranges we were interested in would have been too computationally expensive and time consuming.

Step two: Evaluation

We applied each of the 32 models to the validation set and recorded the mean average precision, root mean square error, precision at k, and normalized discounted cumulative gain. The results can be found in table 2. Based on our discussion in lecture, we decided to use mean average precision as our primary evaluation metric, though it should be noted that the other metrics we calculated did not universally agree with MAP on the “best” model. The model with the parameter settings that resulted in the best MAP on the validation set was then applied to the test set. The final hyperparameters were rank = 100, regularization parameter = 0.001, maximum iterations = 20, and alpha = 1. Test set evaluation metrics are reported below in table 3.

3.2 Challenges

There were some challenges we faced while implementing the Pyspark ALS model. (1) The first problem was working with a full train dataset. Since the full data prior to preprocessing was so big, we got numerous OutOfMemory error messages. This also did not seem ideal since we had to share resources in Peel Cluster with other classmates. (2) User_id and track_id had to be converted to integers in order to train in the ALS model. Our initial approach successfully converted both IDs to integers, however, they were 64-bit, which is not compatible with the Pyspark ALS version we were working with.

4. EVALUATION

4.1 A metric used to evaluate the result

To assess the accuracy of our models, we turned to four common evaluation metrics for recommendation systems: mean average precision, root mean square error, precision at k, and normalized discounted cumulative gain. Mean average precision (or “MAP”) was used to make decisions between models with different hyperparameters, and we will define it here as it is calculated in Pyspark ML[2]:

$$MAP = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{|D_i|}\sum_{j=0}^{Q-1}\frac{rel_{D_i}(R_i(j))}{j+1}$$

where $U = \{u_0, u_1, \ldots, u_{M-1}\}$, $D_i = \{d_0, d_1, \ldots, d_{N-1}\}$, $R_i = [r_0, r_1, \ldots, r_{Q-1}]$, and $rel_D(r) = \begin{cases} 1 & \text{if } r \in D, \\ 0 & \text{otherwise.} \end{cases}$.

To implement MAP as well as the other ranking metrics in Pyspark, we were required to generate a ranked set of recommendations, as well as a "relevant set". The ranked set was easily obtained from the model and the validation set itself, using the recommendForUserSubset function. Based on our discussions in class, we examined the top 500 recommendations.

Rank	Regularization parameter	Maximum iterations	Alpha	**MAP (mean average precision)**	Precision at k=500	NDCG (normalized discounted cumulative gain)	RMSE (root mean square error)
10	1	10	1	4.85E-05	5.90E-05	0.001573	7.08
10	1	20	1	5.60E-05	5.61E-05	0.001577	7.068
10	0.1	10	1	0.0001163	9.65E-05	0.002638	8.659
10	0.1	20	1	0.0001346	0.0001313	0.003677	8.352
10	0.01	10	1	0.0001749	8.47E-05	0.002396	13.949
10	0.01	20	1	0.0001344	9.12E-05	0.002505	12.323
10	0.001	10	1	0.0001705	6.97E-05	0.002079	26.34
10	0.001	20	1	0.0001307	7.69E-05	0.002254	23.997
50	1	10	1	0.0001136	7.50E-05	0.002113	6.865
50	1	20	1	0.0001021	6.71E-05	0.001949	6.83
50	0.1	10	1	0.0001147	0.000101	0.002815	7.244
50	0.1	20	1	0.0001293	0.0001121	0.003135	7.09
50	0.01	10	1	0.0001925	0.0001349	0.003892	7.737
50	0.01	20	1	0.0001848	0.0001277	0.003716	7.506
50	0.001	10	1	0.0002577	0.0001747	0.004907	8.786
50	0.001	20	1	0.0002355	0.0001662	0.004616	8.304
100	1	10	1	0.0001396	8.99E-05	0.002591	6.84
100	1	20	1	0.0001418	7.79E-05	0.002229	6.805
100	0.1	10	1	0.0001899	0.0001343	0.003834	7.075
100	0.1	20	1	0.0001901	0.000147	0.00407	6.956
100	0.01	10	1	0.0002735	0.0001942	0.005413	7.246
100	0.01	20	1	0.0002228	0.000177	0.004833	7.154
100	0.001	10	1	0.000511	0.0002871	0.008279	7.68
100	**0.001**	**20**	**1**	**0.0005843**	**0.000276**	**0.007946**	**7.434**
100	0.001	30	1	0.0004336	0.000248	0.007031	7.373
100	0.001	50	1	0.0004932	0.0002418	0.00697	7.276
100	0.001	100	1	0.000374	0.0002249	0.006394	7.169
100	0.001	20	0.01	0.0005844	0.000277	0.007974	7.434
100	0.001	20	0.1	0.0003635	0.0002653	0.007636	7.507
100	0.001	20	0.5	0.0003632	0.0002656	0.007639	7.507
100	0.001	20	1.5	0.0003632	0.0002653	0.007636	7.507
100	0.001	20	5	0.0003633	0.0002653	0.007635	7.507

Table 2: Results of the hyperparameter search with the validation data. The best performing model had hyperparameters rank = 100, regularization parameter = 0.001, maximum iterations = 20, and alpha = 1.

The "relevant set"

Determining the relevant set, on the other hand, depended on our own intuition of what constitutes a good song recommendation in our data. We chose to include every song that was listened to more than 5 times as "relevant." The number was chosen by the team based on our own experience--if someone recommended us a song, and we listened to it five times or more, we would consider that a successful recommendation. We also considered defining the relevant set as the top 500 most listened-to songs for each user, but this definition performed worse for all models we tried and, in our opinion, makes less sense (that is, just because a song isn't in your 500 most listened to doesn't mean that recommending it would be a failure). Plus, users vary in their listening habits and using listen count to determine relevance should be more individualized and flexible to different user types than a hard song number cutoff.

The results from our hyperparameter search for each of the evaluation metrics described above can be found in table 2. The evaluation results for the test set are in table 3.

Rank	Regularization parameter	Maximum iterations	Alpha	**MAP (mean average precision)**	Precision at k=500	NDCG (normalized discounted cumulative gain)	RMSE (root mean square error)
100	0.001	20	1	0.0004054	0.0002587	0.007375	7.78

Table 3: Test data evaluation metrics based on the best performing hyperparameters, rank = 100, regularization parameter = 0.001, maximum iterations = 20, and alpha = 1.

4.2 Discussion

The most striking feature of our results is how poorly even the best model performs. It's mean average precision is about 0.0006 on the validation set, and 0.0004 on the test set. There are a few potential explanations for this poor performance.

It's possible that the model could perform much better if more data was incorporated into the training set. We chose to include only users that appeared in either the test or validation set--110,000 users in total, just about 10% of the total number of users in the dataset. The tradeoff for faster-trained models and quicker iteration might be lower validation and test accuracy. Ideally, future iterations of this model would include more users. We would recommend adding the most active users first, as their extensive listening histories may have more utility in predicting others' habits. A learning curve could be used to examine the benefits of additional data via mean average precision.

Another possible solution to the poor performance could be a more extensive hyperparameter grid search. Although we tested 32 models, we were limited in the hyperparameter ranges we could try and it is possible that we missed a hyperparameter "sweet spot."

Perhaps most interestingly, we must consider how our choices in model evaluation affected our beliefs about how the model performed. To calculate mean average precision, we decided that the true "relevant set" of songs for a given user should include all tracks that had more than five listens. This was not a random choice--we believe that if a friend recommended a song, and we listened to it five times or more, that would be a successful recommendation--but it is anecdotal at best. After we determined the best model (which can be found in table 2), we investigated a few other relevance thresholds and evaluated the same model again. That is, we recalculated MAP eleven times, each time considering the relevant set to include songs with more than 10, 30, 40, 45, 50, 55, 60, 65, 70, 75, and 100 listens respectively. Obviously it is not an advisable practice to change the parameters of the evaluation metric after the model selection process is complete, but here it granted us some interesting insights. We saw that the MAP improved steadily, peaking at 60 or more listens with 0.006, a more than 10x increase over the original relevant set which included all songs with 5 or more listens.

It's worth considering what this says about our model. It is possible that the MAP benefited from smaller relevant sets--however, those benefits are limited, as the model performed very poorly when the relevant set was determined by a hard 500-song cut off, as mentioned in the Evaluation section. Perhaps our model is more powerful than our original evaluation method gave it credit for, as it appears to perform better when the bar for a successful recommendation was set *higher*, not lower. At the very least, this finding warrants more thought and investigation. It leads to questions about the timeframe this dataset was collected over--how long does it take for a novel song to amass 60 listens? Is this a realistic goal for a newly recommended song? For the average user, how many songs reach this 60 count threshold? And, perhaps most importantly, does a user really want a song like one they've listened to 60 times before? These are fascinating behavioral questions, and the answers could inform more effective methods to evaluate recommendation systems.

5. EXTENSION

5.1 LightFM model

For the extension, we picked the comparison to single-machine implementations, specifically LightFM. This was implemented in the Greene Cluster using JupyterLab. In order to access the same dataset, all files were re-partitioned to one, saved as csv files and CopyToLocal. The number of unique users and songs were checked and confirmed that ALS and LightFM had access to the same dataset. After creating a working LightFM model, we looked for the most optimized hyperparameters using randomized numbers. [3] Random search is an effective method to find the best hyperparameters, and since LightFM has several different parameters, this has been deemed to be the best method to optimize hyperparameters for LightFM. The function used to find the hyperparameters is from Stack Overflow. [4]

5.2 Results

The best hyperparameters for the LightFM is no_components=19, learning_schedule='adagrad', learning_rate=0.0932200406391406,loss='warp-kos', item_alpha=1.1492474344679673e-09, user_alpha=1.2606663566972156e-08, max_sampled=9 with epochs of 25. We fitted the train data with these hyperparameters, and on test data, the final AUC was 0.81485515832901, Precision at k=500 was 0.00032400048803538084, and Recall at k=500 was 0.06808885111568418.

5.3 Comparison

We compare the efficiency and accuracy of the ALS model and LightFM model. Using the exact same train dataset, the ALS model took about 19.3207 minutes while the LightFM model took about 9.6645 minutes. It took less time to train the LightFM model compared to the ALS model. However, higher efficiency could have resulted from most of the classmates utilizing Peel cluster over Greene cluster. In addition, we used one node, 8 core and 16GB to run the LightFM model, meanwhile the ALS model was run in Peel cluster with the driver memory and executor memory of 4GB with executor cores of 50. Since ALS and LightFM models were not trained on the same specification and resources, it is hard to conclude that the LightFM model's training time will always be shorter than the ALS model. For accuracy, we used precision at 500 to compare. The ALS model has precision at 500 of 0.0002760 while the LightFM has precision at 500 of 0.0003332. Therefore, the ALS model outperforms the LightFM model based on the precision at 500.

6. CONCLUSION

6.1 Conclusion

In this project, we implemented the Spark ALS model and LightFM model on the Million Song Dataset. When both models were trained using the same train dataset, the LightFM model was more efficient while the ALS model had higher accuracy based on the precision at k=500. Comparing efficiency can be questionable given that these models were trained using different resources. However, given how large the Million Song Dataset is, Spark has been proven to be efficient at handling such large data. We recommend Spark ALS model as a better model for Million Song Dataset.

To improve model accuracy, future iterations of this project should include more users in the training dataset, and in particular we would recommend testing the benefits of adding "power users" whose extensive listening habits may have more utility in informing other users' preferences. If time and resources allow, we would also recommend a more exhaustive hyperparameter grid search, as well as further exploration into how to best use listen count data as a meaningful representation of a user's true taste to effectively evaluate the system's performance.

6.2 Contribution

Both members have worked together to research and implement a baseline model, sharing feedback and working together to create the best Recommender System model. Specifically, each member has demonstrated leadership in segments listed below.

Dee Hahm	Preprocessing Data, Baseline Evaluation, Implementing LightFM Model
Kathleen Young	Preprocessing Data, Baseline Model, Baseline Evaluation

REFERENCES

[1] Welcome! | Million Song Dataset Retrieved May 18, 2021 from http://millionsongdataset.com/

[2] “Evaluation Metrics - RDD-Based API.” Evaluation Metrics - RDD-Based API - Spark Documentation, spark.apache.org/docs/latest/mllib-evaluation-metrics.html#ranking-systems.

[3] James Bergstra, Yoshua Bengio. Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research 2012

[4] Maciej Kula. How do I optimize the hyperparameters of LightFM? (August 2018). Retrieved May 18, 2021 from https://stackoverflow.com/questions/49896816/how-do-i-optimize-the-hyperparameters-of-lightfm