RECOMMENDER SYSTEM FOR MILLION SONG DATASET

ANKITA VIJAYA KUMAR KOTTUR, New York University, USA
AYESHA NAZNEEN AHMED, New York University, USA
TAMOGHNA CHAKRABORTY, New York University, USA

1 INTRODUCTION

Recommendation systems have been a significant part of today's world. From Netflix, e-commerce to online dating, everybody likes it customized. Recommendation systems are basically just filtering with predictions of rating or preferences a user would give to a particular item. Well, there are many approaches to build one, namely, collaborative filtering, content based filtering, multi-criteria, risk aware, reinforcement, session-based, Hybrid and Mobile. The ability to accurately recommend complex items without knowing and understanding the item itself is commendable. In this project we have adopted the collaborative filtering method , a matrix factorization method to decompose user-item interaction information to recommend users' most relevant items, with ALS. We have further tried to build extensions by implementing the model on a single machine to compare the performance from HPC settings and also tried to develop a popularity based model which is not provided by spark.

2 DATA

For the baseline recommendation system, we are using the 1 Million Song Dataset which consists of 1129318 unique users and 386213 unique tracks. The dataset consists of 3 attributes namely the 'user_id','track_id' and the 'count', where count is numeric and the rest are alphanumeric values. Each row in the dataset represents the interaction between the user and the track. Basically, count gives the no. of times , say customer x has heard the track y. On these basis we have tried to build a popularity based model as well as a collaborative one.

3 DATA PROCESSING

A good approach to process the data is to first down sample the data into some percentages and then try building the model and test it on a small to large dataset while comparing the results. This also helps in understanding the constraints on memory. We first sampled the data to 1% then 5% and then used the entire dataset. We then created hashed values for user_id and track_id for the data since our model required to be in integer format. We now have 3 sets of data with all numeric values, ready to be included in the model.

4 MODEL IMPLEMENTATION

The baseline model for this project mainly uses libraries pyspark.ml (has several predefined recommender functions based on Spark DataFrame) and pyspark.mllib (RDD based APIs). The former is used to train the model and the later is mainly used for evaluation. We use the collaborative filtering technique for our recommender system [1], which as we know, uses a small set of latent factors to predict missing entries in our utility matrix. The ALS or Alternating Least Squares algorithm is used by us to fit our model. The ALS algorithm factorizes the utility matrix into two matrices U(user matrix) and V(item matrix).The latent factors are uncovered by the algorithm which explains user to item ratings(here, play count) and the algorithm tries to minimize the least squares between predicted and actual ratings.

5 EXPLANATION OF THE PARAMETERS AND RANKING METRICS

Implementation of ALS in spark.ml requires these parameters [1]
1. numBlocks- the number of blocks the users and items will be partitioned into in order to parallelize computation (defaults to 10).
2. rank - the number of independent rows and columns of user and item matrices, or the number of latent factors in the model (defaults to 10).
3. maxIter - the maximum number of iterations to run (defaults to 10).
4. regParam- the regularization parameter in ALS (defaults to 1.0).
5. implicitPrefs- TRUE for implicit feedback Collaborative filtering (our case)
6. alpha- applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations (defaults to 1.0).
7. nonnegative- specifies whether or not to use nonnegative constraints for least squares (defaults to false).

The ranking metrics we have used in our model are Precision@K and MAP (Mean Average Precision). Here K refers to the number of recommendations per user, which in our case is 500. Precision@K simply returns the score as calculated from the formula True predictions@k for the user/Total predictions@K) and MAP, as the name suggests is the mean of all the average precision scores which is a metric that tells us how a single sorted prediction compares with our ground truth. [2]

6 HYPERPARAMETER TUNING

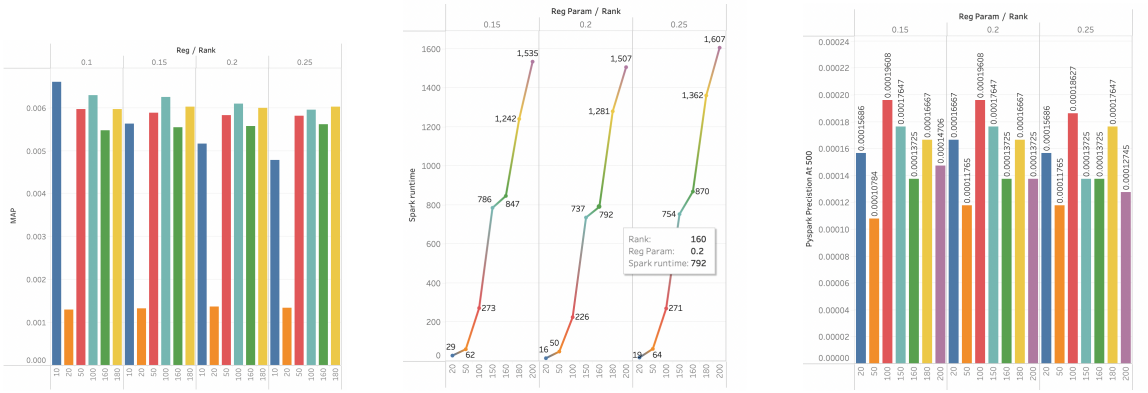
We have run our base model on 1%, 5% and 100% of our dataset. To increase the performance of the ALS model we decided to tune some of the hyperparameters. We experimented with various ranges and values of rank and regularization parameters. We also played with values of maxIter. We ran several models having different values of these hyperparameters on our subsamples,loosely basing our code on the code pprovided here [3] and the parameters that gave us the max MAP score. The metric scores were always greater for higher rank models, but with increase in rank, the runtimes also increased. We got our best results for the subsample datasets with rank= 200, regParam= 0.25, maxIter= 10. So, we ran our model on the entire MSD dataset using these values for our hyperparameters. All results for our larger datasets are available on our GitHub repository

7 BASELINE MODEL CHALLENGES

To run the baseline model on the entire dataset with such high values for rank parameter, we had to repartition the dataframes for parallel computing.

8 EXTENSION1-IMPLEMENTING A BASELINE MODEL

Unfortunately Spark doesn't provide us with a popularity-based baseline model. Baseline models are usually a good starting point and help us make better changes to the model we eventually want to implement. Implementation of the baseline model will help us compare the performance of the ALS model that we are implementing. As there isn't one we use the Lenskit package [4]. LensKit is an open-source set of tools for building recommender systems. We used the bias implementation provided to generate personalized predictions. We tuned the damping parameter of the model. Damping parameter is especially important; it doesn't allow the results not to be skewed by few results which



(a) MAP vs params and ranks (b) Runtime vs params and ranks (c) Precision at 500 vs params and ranks

Fig. 1. Plots corresponding to 1% data sample

are extreme values. It does this damping low-information users and items into the mean rather than allowing them to take on extreme values dependent on few ratings. After testing out various values we found that value 10^9 gave us the best possible results. Results were then stored in a csv file. We also faced issues when trying to use pyspark's ranking metrics. To mitigate this we converted our parquet files to a pandas dataframe. We calculated the mean average precision, NDCG values. We then wrote out our results in a csv file. We also calculated the MAP and NDCG values a k = 500. As we were finding the cluster to be too busy at times we installed Spark and ran the model on 1% data locally.

Lenskit Metrics	
Metric	Score
MAP	0.007144889759509924
Precision@500	0.00019052470708099837
NDCG@500	0.01970656797561984

9 EXTENSION 2: SINGLE MACHINE IMPLEMENTATION USING LIGHTFM

Lightfm is a collection of various recommender algorithms in python [5]. It includes explicit feedback, implicit feedback and cold start and hybrid models. We read parquet files to pandas dataframe using pyarrow package. We converted the initial utility matrix using pandas.pivottable and then that into an acceptable formal for lightfm using scipy.sparse.crsmatrix. We split the dataset into a 70-30 ratio. We experimented with WARP (Weighted Approximate-Rank Pairwise) loss and BPR (Bayesian Personalized Ranking) and various rank and regularization parameter. Please refer to the Lightfmresult VS Baseline.pdf for comparison of various aspects lightfm models to that of the baseline.

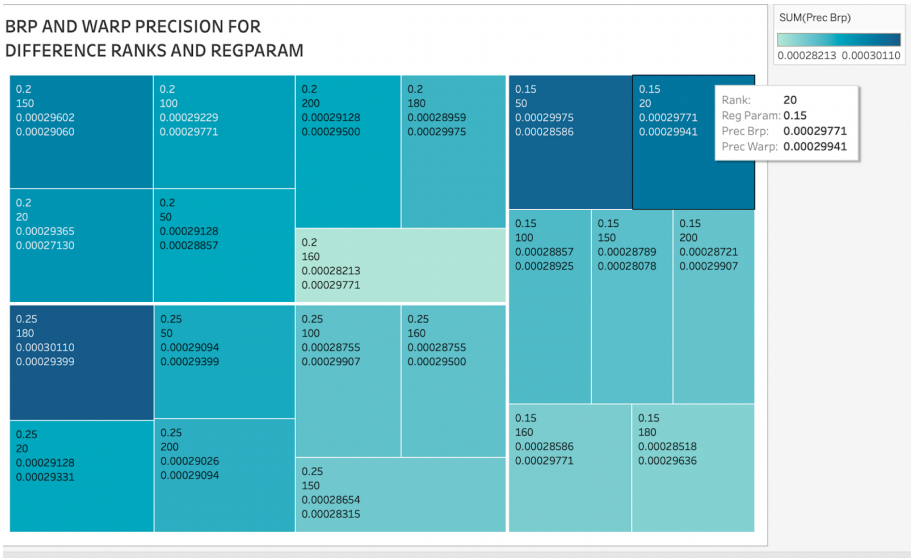


Fig. 2. LightFM model results with different ranks and regParams

10 CONTRIBUTIONS

ALS Model:-Tamoghna, Ankita , Ayesha

Extension1-Tamoghna, Ayesha

Extension2-Ankita , Ayesha

Report-Tamoghna, Ankita , Ayesha

REFERENCES

[1] https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html
[2] https://queirozf.com/entries/evaluation-metrics-for-ranking-problems-introduction-and-examplesmap-mean-average-precision
[3] https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3175648861028866/48824497172554/657465297935335/latest.html
[4] https://lkpy.readthedocs.io/en/stable/
[5] https://making.lyst.com/lightfm/docs/lightfm.html