Yahoo Music Recommendation
Advanced Machine Learning
Progress Report

Kirk Hunter, Mrunmayee Bhagwat, Swetha Reddy

February 18, 2016

Summary

After writing a simple collaborative filtering algorithm from scratch earlier in the course and observing that its computational complexity is $O(N^2)$, we sought to implement a recommendation system at scale. The data we are using is the Yahoo! music ratings data set that was recently released. For our implementation we created an Elastic MapReduce instance on Amazon AWS with five nodes, and we stored the data in an S3 bucket. We used Spark to analyze the data and implement any machine learning algorithms.

Exploratory Data Analysis

The size of the data set is around 11 GB. For our initial analysis we have used 10% of the data. The table 1 shows the songs with highest number of ratings. Genre for 4 of these 5 songs is unknown while the 3rd highest song is from the genre rock. The distribution of the songs by genre is represented in the figure 1. It shows that rock and pop are the two most represented genres in the data set. The unknown genre was omitted from the plot.

Table 1: Songs with high number of ratings

Song Id	Number Of Ratings
72309	35,682
105433	33,954
22763	33,794
123176	32,393
36561	32,074

Table 2: Ratings per genre

Genre	Number Of Ratings
Unknown	65,499,138
Rock	5,293,264
Pop	1,744,753
R&B	1,295,667
Country	567,461

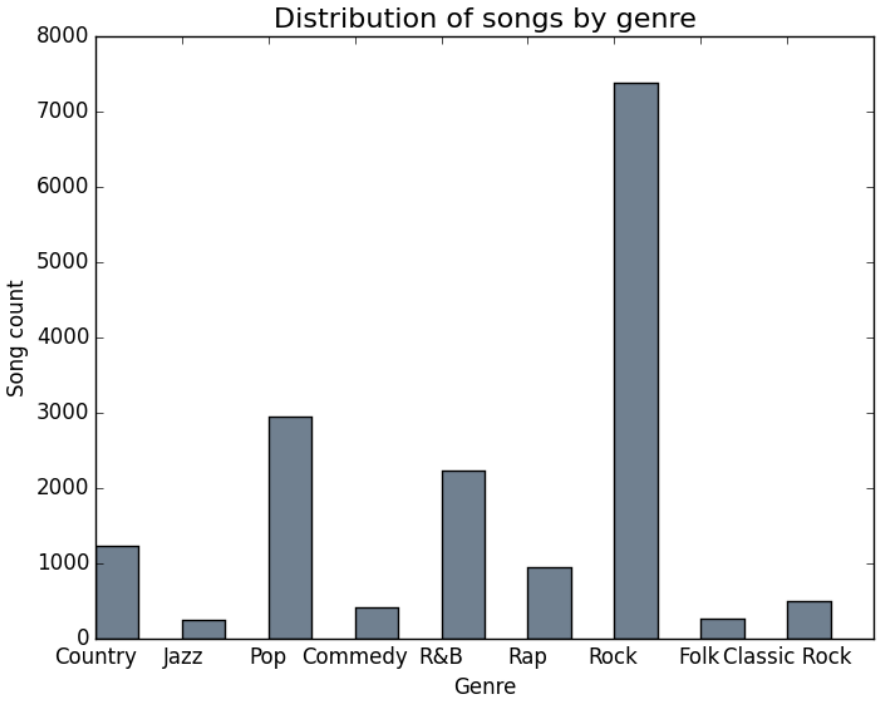

Figure 1: Distribution of songs

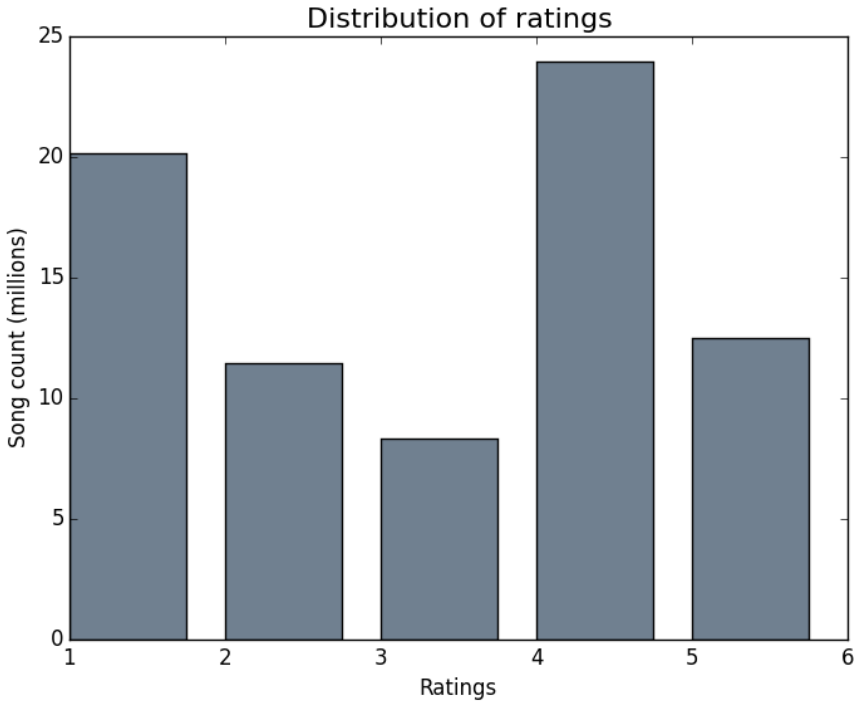

Figure 2: Distribution of ratings

Algorithm Overview

Similarity matrix using Map Reduce technique

To implement an item to item similarity matrix in collaborative filtering algorithm using MapReduce technique, we have referred to the 'Algorithm 3' described in the paper Scalable Similarity-Based Neighborhood Methods with MapReduce by Sebastian Schelter, Christoph Boden, Volker Markl. We have constructed the item by item matrix using Jaccard, Cosine similarities and Pearson correlation coefficient.

One important feature of the algorithm is that it employs an interaction cut, whereby a given user is restricted to giving p ratings in the data set. This greatly reduces the computational complexity since runtime is dominated by these so called 'power users' who give a disproportionately large number of overall ratings. For a given power user, if the number of ratings considered is restricted to p ratings, then the number of item-pairs is restricted to $\binom{p}{2}$, whereas without this interaction-cut the number of item-pairs grows rapidly and consumes computational efficiency.

In figure 3 we see that, although the increase in runtime does not appear to be too severe following an increase in the number of users, it is not quite linear. This may be an indication that the interaction-cut paramter p should be reduced.

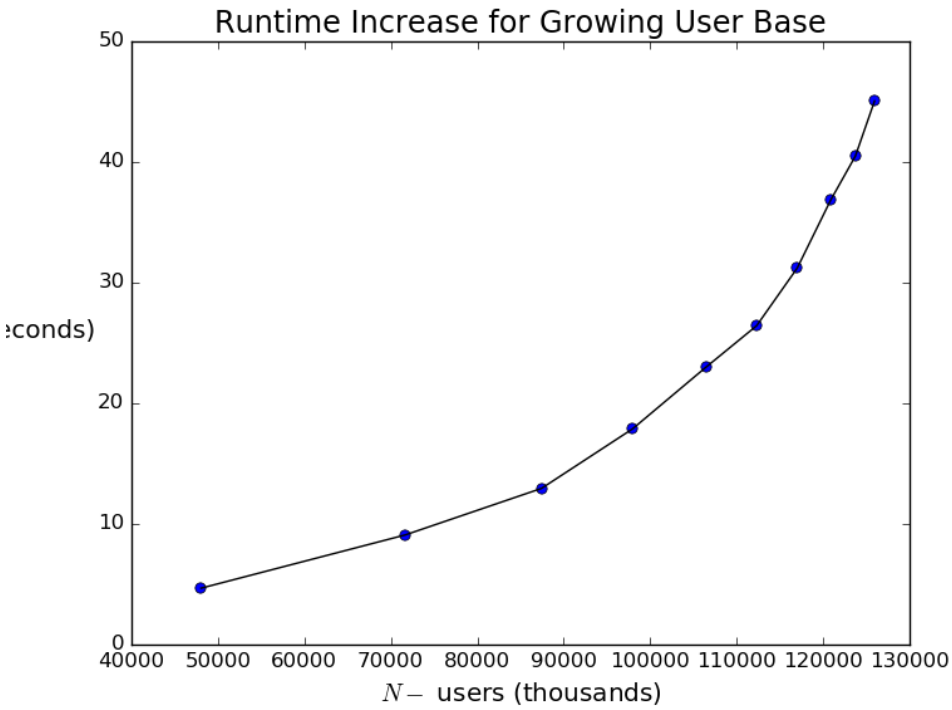

Figure 3: Runtime using Jaccard similarity

Prediction using Alternating Least Squares(ALS) algorithm

ALS algorithm uses matrix factorization to predict unobserved ratings in the user to item association matrix. In certain situations, this method gives better recommendations than the nearest neighbor algorithm. This technique aims to fill the missing entries of a user to item matrix.

Spark's machine learning library MLlib provides an implementation of ALS algorithm. We implemented it to test on different data sets. It was observed that the test Root Mean Square Error (RMSE) significantly decreased as more records are used to train the model. The figure 4 represents this.

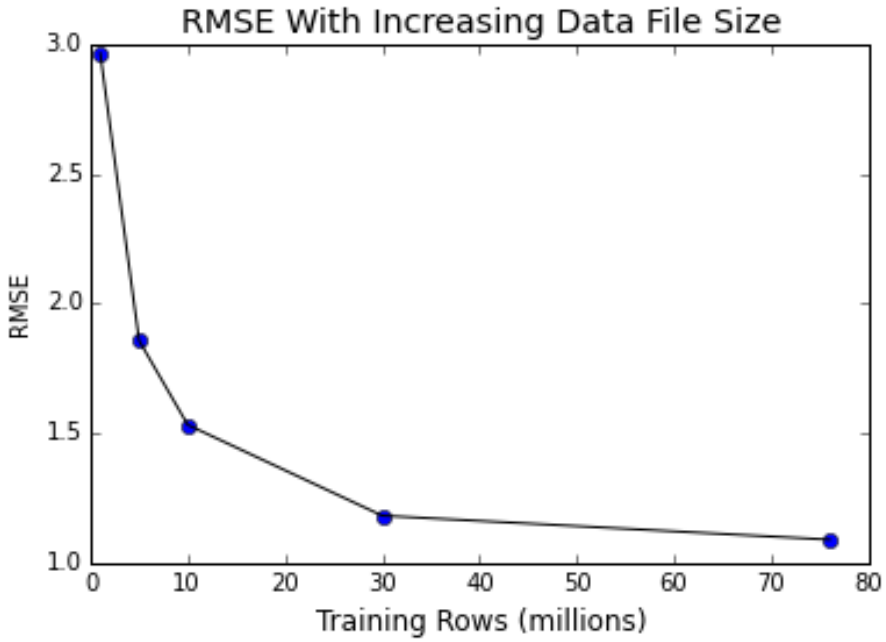

Figure 4: RMSE for ALS algorithm

Further Analysis

In the following weeks we will implement locality sensitive hashing (LSH) to further speed up the item similarity calculations. Under LSH, rough similarity measures are used to partition the data into bins, and more precise similarity measures are calculated within each bin. With this method we will lose some information by partitioning the data into separate bins, but we gain an overall decrease in runtime since the similarity calculations can be run in parallel on each bin.

Our ultimate goal with the project is to build a recommendation system that utilizes the entire Yahoo! music ratings data (11GB). Thus far we have only used as much as 10% of the overall data for our implementations. In order to achieve this goal we must partition the data prior to the similarity calculations. This will surely reduce the accuracy of the predictions somewhat, but with data of this size the increase in efficiency may outweigh the decrease in prediction accuracy.