Music Recommendation System

Yuwei Tu, Yiran Xu, Weisen Zhao

May 12, 2017

Abstract

In this project, we are trying to build a music recommendation system based on the predicted score of rating on a specific song for a user. We employed Matrix Factorization using Alternating Least Square to generate features, and trained Matrix Factorization/Regression/Ensemble models to make predictions.

1 Introduction

With the growth of world wide web, a large amount of music is available for listening on the Internet. Listening music on the Internet serves as an essential entertainment in people's daily life. In order to have a effective method for users to find their favorite music, it becomes necessary to develop a personalized music recommendation system nowadays. In this project, we build a music recommendation system based on the prediction of rating, using user's features that we extracted from the original raw data by Alternating Least Square. We employed and evaluated different machine learning models like Matrix Factorization and Regression to make better prediction of user ratings.

2 Dataset

We used Yahoo! Music rating data provided by Yahoo Research Lab for this project. There are two data that are useful for us: rating data and track info data. The rating data lists the ratings of users that are grouped by user id. Users gave ratings to one of the four different items: tracks, albums, artists, and genres. For each user, we have the user's total number of ratings. For each rating of a user, we have item id, score and time of rating. The scores are integers between 0 and 100. The track info data has hierarchical information of tracks on which album, artist, and optional genres a track belongs to. All the users and items (track, album, artist) are represented as an id number.

2.1 User Information

We have $249,013$ distinct users and those users gave $61,944,607$ ratings in total in the raw dataset. Numbers of distinct users that gave ratings on each type of item are summarized as table 1.
The average rating users gave is 50.15. The distribution of ratings is shown in detail in section 2.3.

Item Type	Number of Users
track	178,131
album	148,852
artist	247,635
genre	208,376

Table 1: Summaries of distinct number of users on each type

To find similarities among users, we used K-means clustering technique based on users' features. We ran K-means clustering for K = 8 to K = 48. We used formula(1), total within-cluster sum of squared distances, to evaluate the result of K-means clustering.

$$\text{Z(x,u)} = \sum_{i=1}^{n} ||x_i - u_{c(x_i)}||_2^2 \tag{1}$$

where $u_{c(x_i)}$ is the cluster point associated to x_i.
Although the Z-score for K-means clustering is decreasing as K value increases from 8 to 48, the smallest Z-score that we achieve is still Incredibly huge. So in conclusion, there is no strong similarity between users.

2.2 Item Information

We have $624,961$ items in total in the raw dataset and $296,112$ of them were given ratings. The items include track, album, artist and genre. Among items given ratings, number of ratings given on each type of item are summarized as table 2.

Item Type	Number of Ratings
track	27,167,857
album	11,928,316
artist	19,289,882
genre	3,558,351

Table 2: Summaries of number of ratings on each type

Numbers of distinct items of each type are summarized as table 3.

Item Type	Number
track	224,041
album	52,829
artist	18674
genre	567

Table 3: Summaries of number of distinct items

We are targeting on the rating of track in this project and ignore ratings on genre.

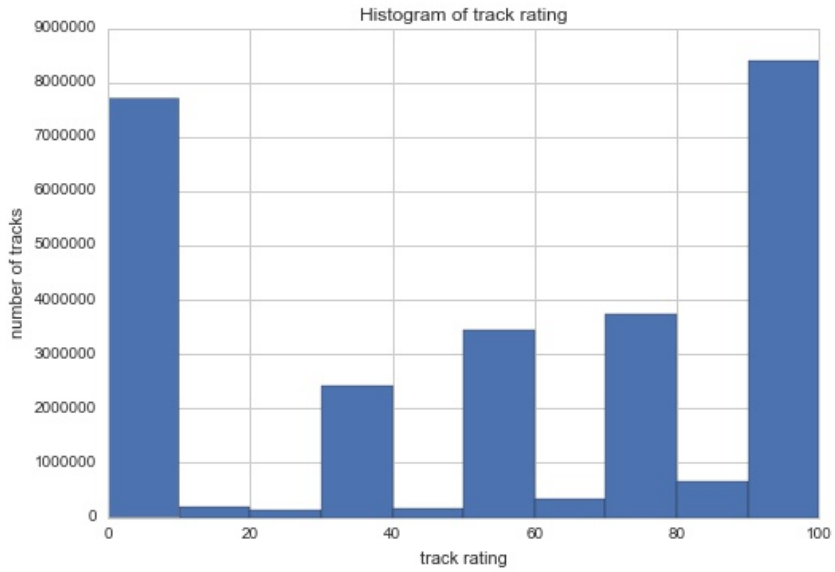

Figure 1: Histogram of ratings for tracks.

2.3 Rating distribution

We made two histograms to see distributions of the target variable, ratings of tracks, and distribution of ratings of all items (tracks, artists, albums and genres).
As we can see from both histograms that follow, Figure 1 and Figure 2, both ratings have similar distributions. Both have five distinct groups. Around 30% of ratings fall into the range of 0 to 20. Another 30% of ratings fall into the range of 80 to 100. Each of the remaining ranges, 20 to 40, 40 to 60, 60 to 80, has around 10% of ratings.

2.4 Binning/Grouping

Because the target variable, ratings of tracks, forms five distinct groups as we can see from Figure 1, we decided to bin ratings into five classes and treated our problem as a regression or multi-class classification problem.
We grouped ratings from 0 to 20 as "1"; ratings from 20 to 40 as "2"; ratings from 40 to 60 as "3"; ratings from 60 to 80 as "4", ratings from 80 to 100 as "5".
Since the results for multi-class classification are not ideal, we would talk about classification models here briefly and focus on regression models for the rest of the report.
For multi-class classification models, we used Multinomial Logistic Regressions (MLR) and Decision Tree classification models to predict class of ratings. We chose precision and recall as evaluation metrics. By applying these models to data of size one million, we got the results summarized as table 4 and 5:

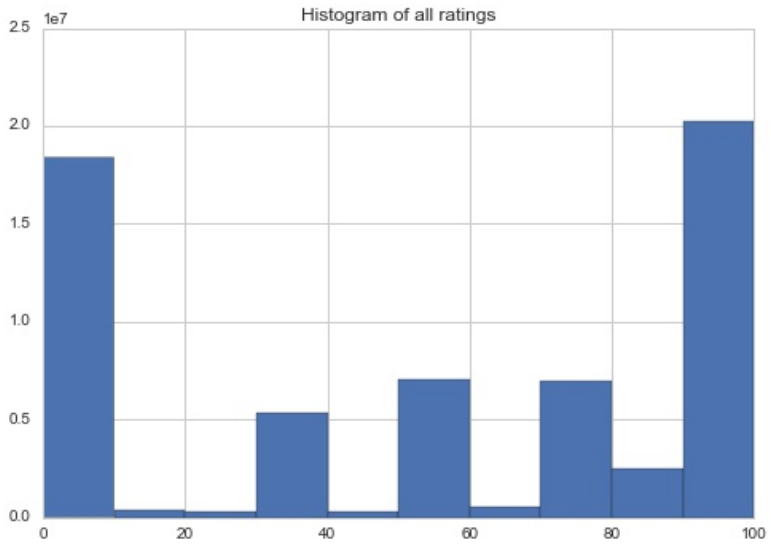

Figure 2: Histogram of ratings for all items.

precision	overall	class “1”	class “2”	class “3”	class “4”	class “5”
MLR	0.327	0.321	0	0	0	0.329
Decision Tree	0.338	0.339	0.207	0.195	0.264	0.344

Table 4: Summaries of precisions

As we can see, precisions are quite low for all classes while recall is high for class ”5”. The low precision for class ”5” means among all the instances classified as class ”5”, only around 33% are really class ”5”, the highest rating group. In the context of recommender system, this means among all the tracks we recommend, only 33% are tracks a user would really like, which is not ideal. The high recall indicates that too many classes are classified as class ”5” and the classifiers failed to distinguish different classes. Therefore, multi-class classification approach is not very suitable for our problem. We will focus on regression approach from now on.

3 Evaluation Metrics

The evaluation metrics we chose for regression models are Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

recall	overall	class “1”	class “2”	class “3”	class “4”	class “5”
MLR	0.327	0.321	0	0	0	0.732
Decision Tree	0.338	0.4	0.01	0.028	0.0196	0.656

Table 5: Summaries of recalls

3.1 MAE

Mean Absolute Error (MAE) is the mean value of absolute difference between the true values and our predictions. The formula is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|\,, \tag{2}$$

where n = sample size, true value = y_i and prediction = $\hat{y}_i$ and error=$y_i - \hat{y}_i$.
We want the errors to be small, so we want MAE to be as low as possible.

3.2 RMSE

Root Mean Square Error (RMSE) is square root of the mean value of square of all of the errors. The formula is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{3}$$

where n = sample size, true value = y_i, prediction = $\hat{y}_i$ and error=$y_i - \hat{y}_i$.
Compared to MAE, RMSE severely punishes large errors because errors are squared before taking average.
Similarly as MAE, we want RMSE to be as low as possible.

4 Feature Engineering

4.1 User factors and Item factors

We generated latent factors for users and items using matrix factorization.Those latent factors will be used as features in our models.
Matrix factorization is a classic approach to learn latent factors for recommender systems. The matrix to decompose is the utility matrix, M, where each row is a user and each column represent an item. Element M_{ui} is the rating that user u gives to item i. The utility matrix is decomposed into a user matrix and item matrix. We want to choose the matrices such that the product of these two matrices best approximates utility matrix, i.e. having the least RMSE[2]. To avoid overfitting, we can regularize the square loss function. The regularized loss function is:

$$\underset{q,p}{min} \sum_{(u,i)\in k}(r_{ui} - q_i^T p_u) + \lambda(\|q_i\|^2 + \|p_u\|^2), \tag{4}$$

where k is the set of the (u,i) pairs for which r_{ui} is known.

userID	rating	rating_categories	trackID	AlbumId	ArtistId
118756	0	1	80	245620	228132
118756	50	3	164231	160998	244384
118756	30	2	52394	277474	234794
118756	30	2	52233	250197	278913

Figure 3: First four rows of table for hierarchical track information.

After matrix factorization, we got a user matrix and an item matrix. Each row of the user matrix represents a user factor $p_u \in \mathbb{R}^n$. Each column of the item matrix represents an item factor $q_i \in \mathbb{R}^n$, where n is selected by minimizing the squared loss. For a given item vector(factor) $q_i \in \mathbb{R}^n$, the i^{th} element of the factor represents the degree to which this item has on i^{th} latent factor. For a given user vector(factor) $p_u \in \mathbb{R}^n$, the i^{th} element represents the degree of interest to which this user has on i^{th} latent factor [1]. The inner product $q_i^T p_u$ approximates the rating of user u on item i.
By tuning parameters of the loss function, we got our user vectors and item vectors. Length for both vectors is 50, i.e. n =50. For example, item No.40 has item factor $q_{40} = [-0.23171799, 0.15220039, ..., 0.98501605, 0.2750264]^T$. User No.40509 has user factor $p_{40509} = [-0.17391361, 1.0280728, ..., 0.37988117]^T$.
We will then use user factors to represent users, item factors to represent items. Since users are able to give ratings on track, artist, album, item factors are in fact track factors, artist factors and album factors.

4.2 Merge and Sample(20 Million to 4 Million)

To fully utilize the hierarchical information we have on track, we merged a table as Figure 3. For instance, the second row means: user No.117856 gave rating of 50, labeled as "3" on track No.164231 and track No.164231 is in album No.160998 and created by artist No.244384.
We then substituted user id, track id, album id and artist id as its corresponding user factor, track factor, album factor and artist factor. By unwrapping each factors to 50 float numbers and merge them together, we got the whole dataset for modeling with dimension $24,547,016 \times 202$, i.e. $24,547,016$ rows and 202 columns.
However, this whole dataset is too big to run on the available machines. Hence, we randomly sampled four million data points from the whole dataset as our sample data.

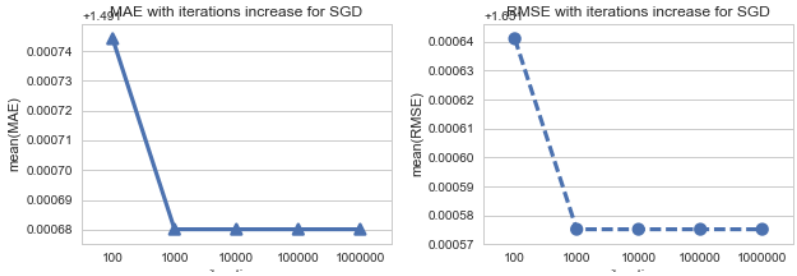

Figure 4: MAE and RMSE with iterations increase for SGD

5 Modeling

5.1 Matrix Factorization

The first model to predict rating is matrix factorization, as the method we explained in the section "User factors and Item factors". By minimizing the loss function Equation, see equation (4), the product of user matrix and item matrix approximates the true utility matrix M. Apache Spark, where we did data engineering and model training, implements alternating least squares (ALS) as the learning algorithm for minimizing the loss function Equation (4). ALS rotates between fixing the $q_i \in \mathbb{R}^n$ and fixing the $p_u \in \mathbb{R}^n$[1]. In each rotation, the system solves the regularized least square problem.

5.2 Regression Models

We randomly sampled 0.5 million user records from the whole dataset, merged each (user,item) pair with correspondingly unwrapped user factor and item factors(including track factor, album factor and artist factor). With all those features, and grouped 1-5 ratings, we try to build a effective regression model to predict corresponding rating for each (user,item) pair. The training/testing datasets are proportionally split by 8:2, and each record in testing data has never been shown in training data before.

5.2.1 Linear Regression

The first and simplest model we tried is linear regression with Stochastic Gradient Descent algorithm. As we can see from Figure 4, the algorithm almost reached its optimum after 1000 iterations, so we use iteration = 1000 as our SGD setting during parameter tuning.
By adding regularization parameter λ, the best λ we can find for both Lasso and Ridge regression are both 0.01 from Figure 5.
The minimal error we can achieve by linear regression is MAE = 1.492, RMSE = 1.651 with L2(Ridge) regression where $\lambda = 0.01$, iteration = 1000.

5.2.2 Decision Tree Regression

Linear regression can't well define the relationship between user/item factors and ratings, so there might be no linear relationship between features and tar-

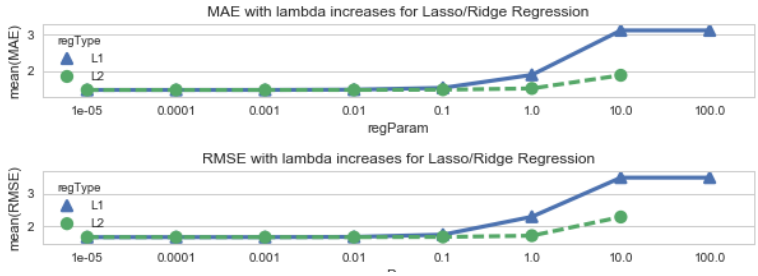

Figure 5: Lambda increases for Lasso and Ridge Regression

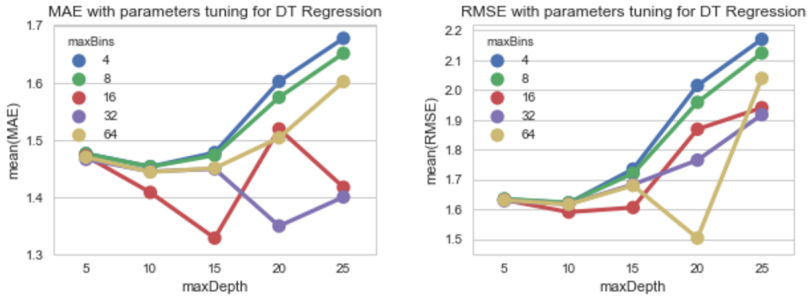

Figure 6: MAE and RMSE with parameters tuning for Decision Tree Regression

gets. The next model we try is Decision Tree regression, which is also simple to explain, but better at modeling non-linear pattern.
From parameter tuning form Figure 6, we can tell the best parameter setting for minimizing MAE is maxDepth = 15, maxBins = 16; And the best parameter setting for minimizing RMSE is maxDepth = 20, maxBins = 64. The minimal error we can achieve by decision tree regression is MAE = 1.328, RMSE = 1.503, which is much better than linear regression.

5.2.3 Gradient Boosted Regression

Inspired by performance of decision tree regression, we also tried different ensemble models which can amplify the benefit of tree models. Combining the performance of decision tree regression on both MAE and RMSE, we set maxDepth = 15, maxBins = 16. However, given the limited computing power, we can only build 10 iterations with learning rate = 0.1. The minimal error we can achieve is MAE = 1.303, RMSE = 1.587, which is just slightly better than decision tree regression. We believe the benefit of Gradient Boosted Regression is limited by the sample size and computing power, so we tried to find a more effective algorithm to improve model performance.

5.2.4 XGBoost Regression

XGBoost is short for "Extreme Gradient Boosting". Actually, Both XGBoost and Gradient Boosted Machine follows the principle of gradient boosting. There are however, the difference in modeling details. Specifically, XGBoost used

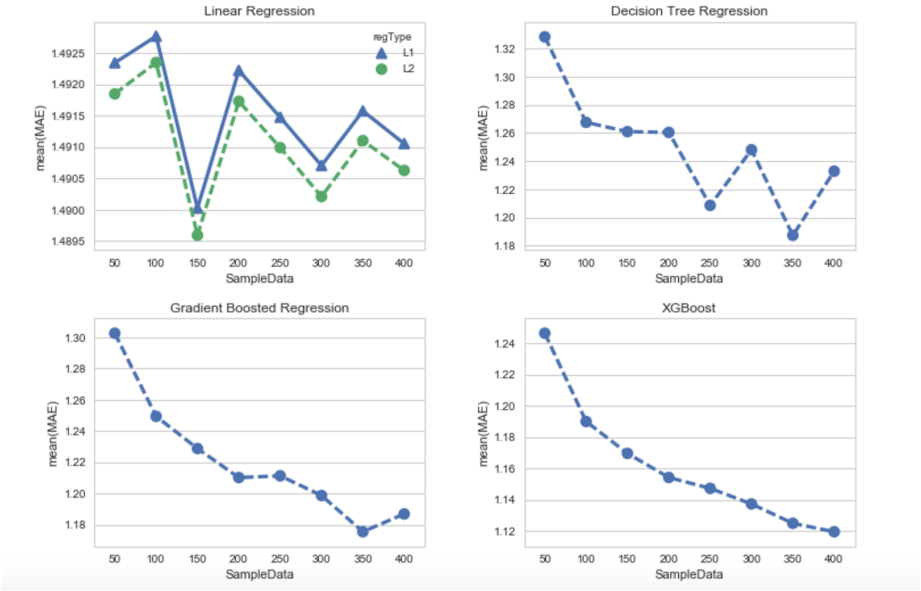

Figure 7: MAE with sample size increasing

a more regularized model formalization to control over-fitting, which gives it better performance. The computing rate is more than 10 times higher than Gradient Boosted Regression, we can easily set iteration=100, and achieve MAE = 1.246, RMSE = 1.438.

5.3 Learning Curve

Given the limitation of sample size, the next we need to figure out, are whether we can get better model performance by increasing sample size, and where is the convergence. So we increased our sample size from 0.5 million to 4 million, with 0.5 million as a step, plotted the error curve with sample size increasing in Figure 7 and Figure 8.
As is shown, the MAE and RMSE kept dropping down as sample size increasing. Even though it seems like Gradient Boosted regression and XGboost regression has already reach their stabilities, the Decision Tree regression still have a large space to improve. Given those two ensemble models are combination of the predictions from base tree estimators, we can assume their performance will also improve with the improving of Decision Tree regression model performance.

5.4 Model Evaluation

The Root Mean Square Error is a frequently used measure between values predicted by a model. And Mean Absolute Error is one of most effective ways to compare forecasts and eventual outcomes. So we use RMSE and MAE to evaluate the performance of different methods that we propose. However, different models didn't present strongly preference to any one of these two indicators. The performance of each model with 4 million sample data has been concluded

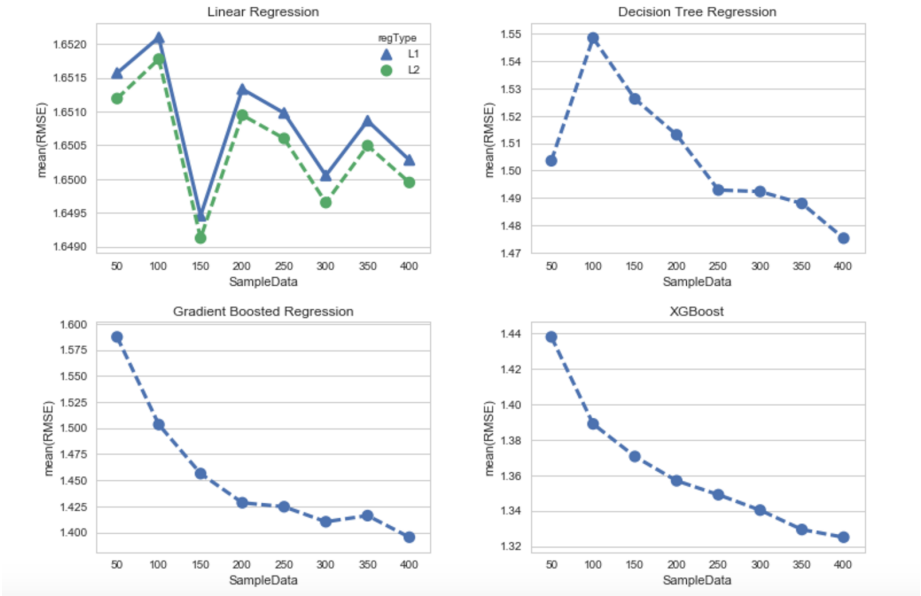

Figure 8: RMSE with sample size increasing

in Figure 9. The minimal MAE we can achieve is MAE = 1.120 by XGBoost, while the minimal RMSE we can achieve is RMSE = 1.325 by XGBoost as well. However, the minimal RMSE we can achieve by ALS with 20 million sample data and $\lambda = 0.01$ L2 regularization is 1.130, which is much better than what we can achieve with partial 4 million sample data.

6 Discussion

First of all, based on the performance of multi-label classification methods, we can there is a strongly unbalance between different labels. The majority of ratings grouped on class 1 and class 5, while only small percent of ratings is distributed between class 2,class 3 and class 4. To solve the unbalanced target distribution problem, we plan to replace random sampling with stratify sampling to make sure our sample data is within the same distribution as original data. There is also another sampling method called down sampling, which is balancing data by keep the same size in each class. However, it might be a bad solution as it biases our model and throws out potentially useful data.
Besides that, given the computing power limitation, we only train 4 million records at present. However, our features are composed with 20 million records. One possible solution might be split original dataset into 5 partitions, train different models separately on each partition, and ensemble these five models by using common label or averaging to get our final prediction.
Next, our model focus on make prediction with user/item features created by matrix factorization. And we didn't pay much attention to original sparse user-item rating matrix. Given a sparse user-item matrix, we can try several Big Data techniques like MinHash and Locality-Sensitive Hashing to find similar items, and build a recommendation system by user-based filtering or item-based

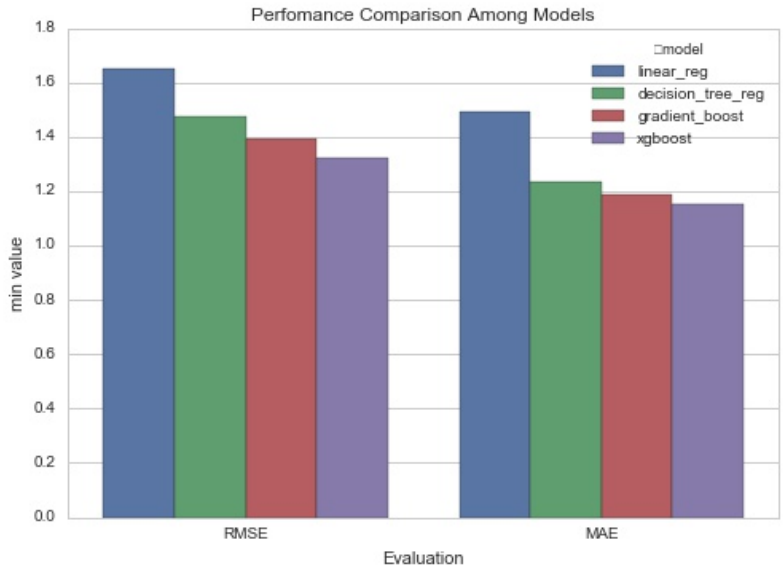

Figure 9: Model Evaluation between all regression models

filtering.
It may also be worthwhile to try other new model families, like co-clustering. Unfortunately due to engineering difficulty and time limitation, we are not able to explore more sophisticated models for this write-up in time.

References

[1] Koren, Yehuda, Robert Bell, and Chris Volinsky. *"Matrix Factorization Techniques for Recommender Systems." Volinsky 42.8 (2009): 30-37.*. Web. 11 Apr. 2017.

[2] Leskovec, Jurij, Anand Rajaraman, and Jeffrey D. Ullman. "Recommendation Systems." Mining of Massive Datasets. Cambridge: Cambridge UP, 2015. N. pag. Print.