Crystal Discoball

Aditya, Apoorv, Dhvani

December 7, 2017

Introduction

This report discusses the design decisions and the result for solving Crystal Discoball.

Approach

We divded the project implementation into two phases

- Feature Computation and Selection
- Using Spark MLlib's Random Forest Implemenetation and tuning the parameters

Feature Selection

We analyzed the Million Song Dataset, Taste Profile and This Is My Jam data. Following are the fields we analyzed from Million Song Dataset: **Artist familiarity**, **Artist hotttnesss**, **Artist terms**, **Artist terms freq**, **Artist terms weight**, **Danceability**, **Duration**, **Energy**, **Loudness**, **Similar artists**, **Song hotttness**, **Tempo**, **Year**

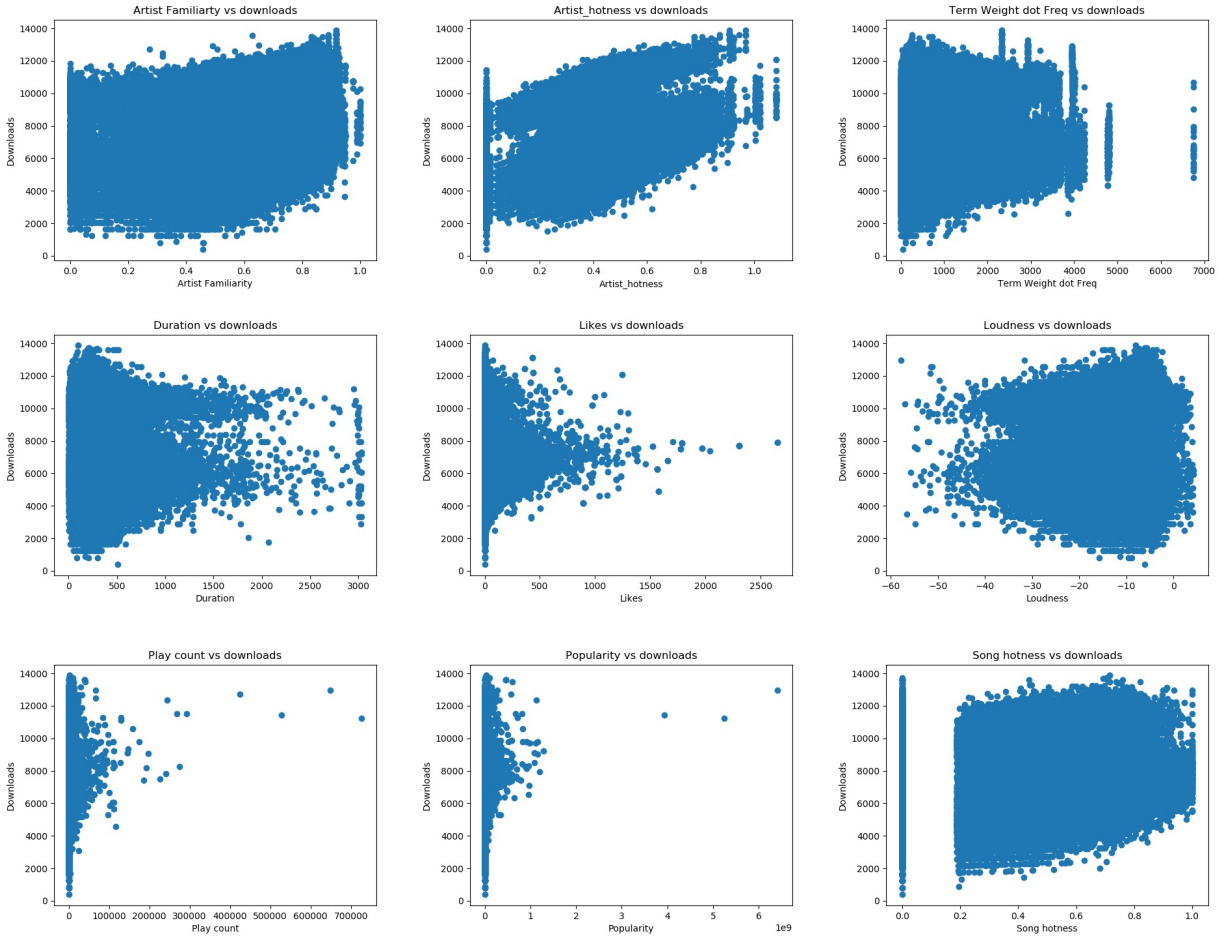
Following are the details of how we gathered data for the above mentioned fields:
artist terms/genre: join between artist_id, artist_hotness and artist_id, genre, followed by calculating the mean scores of all genres, then sorting and finally obtaining the top 1000 results. Then count the number of genres for each artist that match the top 1000 genres. We started with top 5 but found that there are lot of genres and thus all the values were coming as zero. Then we increased the top value to 15, 20, 100, 500, 1000 and decided 1000 was a good value based on the correlation of this value to downloads.
artist terms freq and artist terms weight: multiplied the freq and weight of artist terms, renamed as weight dot freq
danceability and energy: all values are zero
similar artists: count of number of similar artists for each artist
song count: total number of songs for each artist
popularity: popularity for each artist is the product of artist familiarity, song count and the number of similar artists
Taste Profile: We got the total play count for each song (play count)
This Is My Jam Data: We got the number of likes for each jam id and matched that with track id (likes)

Correlation

We analysed the correlation between the above features and downloads in Python and plotted graphs of downloads vs each features. Initially we got bad correlation values. While trying to understand this we saw artist familiarity graph and realized that the data is bad as the values had to be in the range of 0 to 1 but we were getting value near 4000 as well.

Thus, we cleaned the data again. This time, we cleaned the artist name and song title i.e. removed alphanumeric characters and converted to lowercase and checked for row length consistency. Except mean price, no other feature had a promising correlation factor. Thus, we plotted graph to analyse the correlation more thoroughly and we realized that as artist hotness increases, downloads increases.

Visualizing features against downloads



Using Spark MLlib's Random Forest Implementation and tuning the parameters

We have used Random Forest regression algorithm to predict the downloads using Spark's ML Library. **Data**: downloads.csv file which contained information about the downloads, confidence, and the mean price had more than million rows. So, we took left join with million songs data (henceforth while referring to million song data, it has all the above features mentioned during feature selection). However, we realized that almost half of the rows had null values for almost all of the columns. Next step was to decide whether to substitute the null values or remove the rows. One option we thought of was to substitute the null values with average or median depending on the column. However, since many of the columns and rows had null values, we realised substituting it with data ourselves was not a good idea.
Thus, we eliminated such rows and we did an inner join on downloads data and million songs data. We divided data into two parts in 70-30 ratio, 70% for training the random forest model and 30% for testing it. The data was divided in such a way that we avoid the **producer effect** by making sure no song from a given artist ends up in both the train and test set [https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD]

Based on the above analysis, we took the below **15 features** and ran RandomForest algorithm (20 features and depth also as 20) and got **RMSE as 77.77**. To check which feature is more/less important, we ran RandomForest algorithm with each of the 15 features at first and then removed each feature one by one.

Below is the table which has the RMSE values after removing that particular feature. The general idea here was that if the RMSE value decreased, means we need to remove that feature and if it increased, we can consider keeping or removing it.

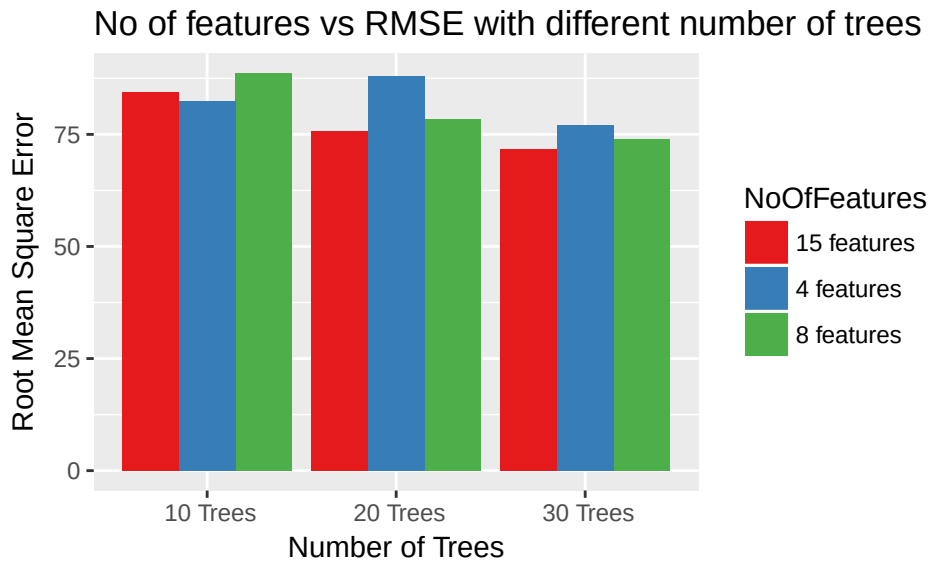
Feature	RMSE without this feature
artist_hotness	280
popularity	87.91
genre	90.53
artist_familiarity	88.76
year	89.5
play_count	101.23
mean_price	2188.88
loudness	84.32
tempo	90.71
song_count	83.63
duration	84.82
likes	88.55
weight_dot_freq	83
song_hotness	96.78
confidence	109.78
similar_artist_count	92.02

Features	RMSE
song_hotness	73.92
confidence	
simlar_artist_count	
tempo	
play_count	
mean_price	
genre	
artist_hotness	

Features	RMSE
song_hotness	101.34
play_count	
mean_price	
artist_hotness	
confidence	

Features	RMSE
play_count	90.01
mean_price	
artist_hotness	
confidence	

All the above runs are with 30 trees and depth as 20. Below is the graph comparing the RMSE for 4, 8 and 15 features with 10, 20 and 30 trees (depth as 20)



RMSE for testing and training data for 4 and 8 features

	4 features	8 features
Testing Data RMSE	77.17	77.37
Training Data RMSE	78.51	41.42

Based on the above table, as the difference between testing and training data RMSE for 8 trees is more than that of 4 trees, it means that the data is overfitted slightly for 8 features. Also, since the testing data RMSE difference between 4 and 8 features is less, we decided to use only 4 features (song_hotness, confidence,

mean_price, artist_hotness) for our model to predict the downloads (30 trees and depth as 20).

Implementation

The confidence feature from download file was converted to a numerical value based on the below criteria: terrible => 0, poor => 1, average => 2, good => 3, very good => 4, excellent => 5 We pass the confidence feature values to the Random Forest algorithm as categorical features while training the model. While running the model, we have a file that maps artist name and song title to the 4 features. This file is left outer join of million songs and downloads. Before matching the artist name and song title, we remove all alphanumeric characters from it and convert to lowercase.
If we get multiple rows, we take average of artist_hotness, song_hotness, mean price and median for confidence. If we don't get any mapping (unknown input data), we first try to get features based on only artist name. We then take average for artist_hotness, song_hotness, mean price and median for confidence. If we still don't get features, we do similar approach with song title. If we still don't features, then we take average for artist_hotness, song_hotness, mean price and median for confidence computed using the whole dataset. We pass these features to the model and get the predicted download value.

Execution Time

The execution time for 4, 8, and 15 features with 30 trees and depth as 20 on AWS (1 master and 4 nodes) was 9 minutes, 26 minutes, and 32 minutes respectively for training the model. Configuration of the machines used (master and node): 4 vCPU(2 cores), 15GB RAM, 80GB SSD, Intel Xeon E5-2670 v2 2.5 GHz

Result

Based on our testing data, we categorized the difference between the actual and predicted values into 3 categories with their counts is as follows: low - 243342 medium - 3423 high - 37 We categorized as follows Sorted the difference in ascending order. Then removed duplicate numbers. Thus, now we have the range of the difference. Split this range into 3 categories with top 33.33% falling in low category, next 33.33% in medium category and the last 33.33% in high category. Thus our model has good accuracy as the number of predictions in low category is more.

Conclusion

- We learnt that data has to be clean.
- After exploring at least 15 features, we found that 4 features were good enough to predict the downloads.

Future Scope

- Compare execution time of random forest using RDD vs Data Frame
- Use other regression algorithms and compare them with this model