Crystal Discoball

Bishwajeet Dey, Rashmi Dwaraka

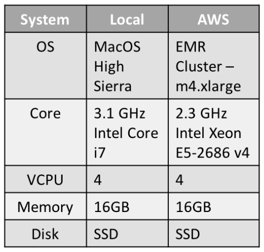
12/7/2017

Problem Statement

Using the Million Song Dataset, create a model capable of predicting song download numbers. The entire problem description can be found here - http://janvitek.org/pdpmr/f17/project.html. The dataset used in this project is Million Song Dataset ('MSD') which is a freely-available collection of audio features and metadata for a million contemporary popular music tracks. Reference - https://labrosa.ee.columbia.edu/millionsong/

Execution Environment

Certain tasks like extracting small datasets and feature selection were executed locally. Building the training model for the entire dataset and parameter tuning was performed on AWS platform for faster execution. System specifications are as per figure 1.



System	Local	AWS
OS	MacOS High Sierra	EMR Cluster – m4.xlarge
Core	3.1 GHz Intel Core i7	2.3 GHz Intel Xeon E5-2686 v4
VCPU	4	4
Memory	16GB	16GB
Disk	SSD	SSD

Figure 1 :
Execution Environment

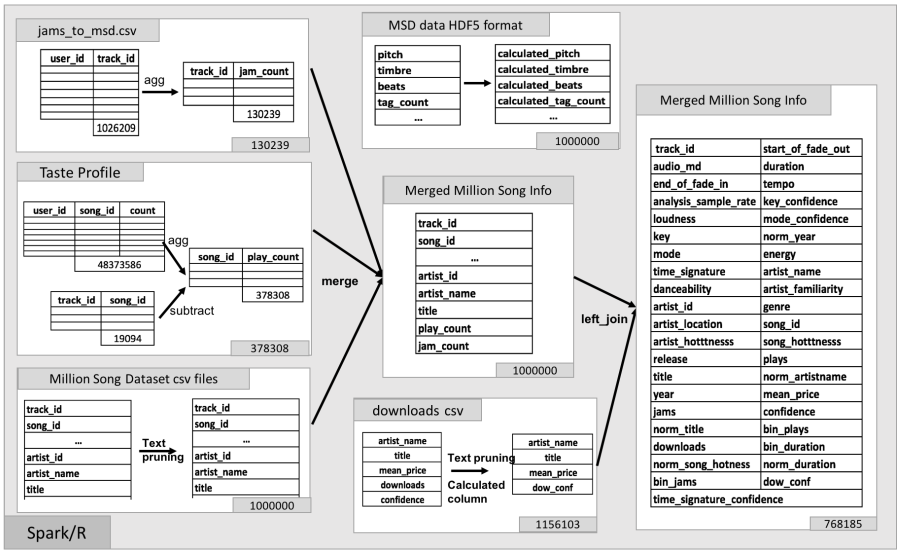


Figure 2 : Data Preparation

1. Data Preparation

To build the final dataset, we combine data from multiple sources, prune and transform relevant dataset. In order to identify the relevance of the data features, we used subset of the dataset to train and analyse the variations and correlation to download numbers.

With reference to figure 2 which represents the Data Preparation phase, we explored the data from the below sources:

1. **Jam Count** - Jams are equivalent of likes (https://labrosa.ee.columbia.edu/millionsong/thisismyjam). We computed the popularity of the tracks by taking the jam count into consideration. Also, we created a binary field to indicate if the track was part of jam or not.
2. **Taste Profile** - The number of times a song was played by a user (https://labrosa.ee.columbia.edu/millionsong/tasteprofile) is captured here. The taste profile compiled by users includes many mismatches between track_id and song_id of MSD and hence were removed. The play_count was aggregated in terms of song_id. We performed this operation using bash scripts as loading and processing in memory was not feasible and efficient. Also, we computed binary field for plays to indicate if it was part of a playlist.
3. **Million Song Dataset csv file** - We downloaded the csv file which included the summary of the audio features with artist and song information. Here, we normalize the textual data by removing any punctuation, additional spaces and lowering the case. Additionally, we transformed the numerical values as below:
   - *norm_song_hotness* - the range of song_hotness is (0,1), we found many of the values were outside this range, we converted these values to 0.0
   - *duration* - converted the duration into bins by computing difference of start-of-fade-out and end-of-fade-in. We tried giving higher weight to the songs with duration between 1 and 3 minutes.
   - *loudness* - the range of values for loudness include negative and positive values based on a reference point 0. In order to nullify the effect of negative and positive values, we squared the loudness value.
   - *other numeric values* - All the NAs are replaced by median of that column instead of removing the entire row. This helps us retain other good features of that track. Since it is replaced by median, it would not induce much error in the model.

4. **Price and download information -** This dataset provide aggregated information about song downloads from various sources. Here, we normalize the textual data by removing any punctuation, additional spaces and lowering the case. In order to consider the confidence of downloads into account, we assigned a fixed range of weights between 0 and 1 to each confidence level (excellent - 1.0, very good - 0.9, good - 0.75, average = 0.5, poor = 0.1). We created a calculated column "dow_conf" by multiplying the confidence weight and downloads.

5. **MSD HDF5 format with Additional fields -** MSD in HDF5 format includes many more detailed audio features like beats, timbre, pitch information of the segments of a song. We wrote java extractors to extract required fields. We compute few additional calculated fields like average of the segment features. We experimented on these extracted fields for the subset of files in order to find their correlation to the song downloads. We did not find any satisfactory correlation between them. Also, the extraction of these fields for subset of files required approximately 40 minutes. Hence, we decided to not use any of these additional fields.

We merged all the data sources - jam, taste_profile and downloads using left_join on million song data. The final training model dataset included 1 million song information. After eliminating the rows where dow_conf is na, we had the total row count of 768185.

Performance

The extraction process was simple to implement and iterative in nature. Hence, we executed the extraction process on the local machine using R. We found that executing it in the R script was faster than SparkR implementation of the same. Since the dataset includes various formats, we used multiple frameworks to extract and prune the dataset. Extraction of each dataset and combining them takes around 2.04 minutes to complete and write the model to file.

2. Feature Selection

The data features that we use to train machine learning models have a huge influence on the performance and accuracy. Irrelevant or partially relevant features can negatively impact model performance. Out of 41 features extracted from the data preparation, we decided on using 6 features. In this phase, we also revisited data preparation phase to modify features to increase the correlation to song downloads.

Feature Elimination - We eliminated few of the features like Danceability, Energy, Genre as their data distribution indicated majority of the value to be either null or 0.0. Analysis Sample Rate was a constant and we eliminate this feature as it would not give us any correlation with song downloads.

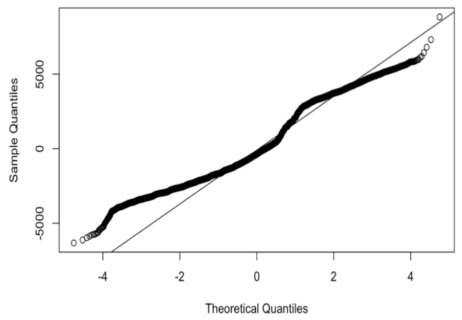


Figure 3: Linear model residual plot

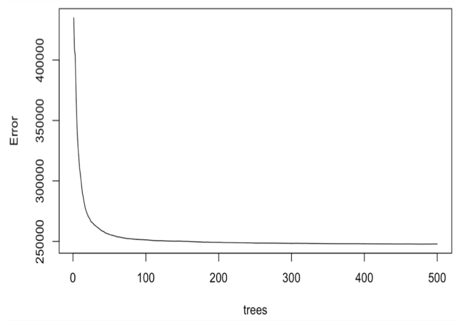


Figure 4: Random Forest – Error rate vs number of trees

Linear Model - In order to find correlation between the features with good data distribution and dow_conf, we tried fitting the data to Linear model. Based on our intuition, we performed trial and error method by selecting a subset of features and fit it to linear model to understand the correlation. Figure 3 shows linear model curve for one of the various subsets. Based on the graph, we conclude that the data is not fit for linear model, as the line is curved. After some research on ML algorithms and data analysis, we decided RandomForest would be a better option, as it fits the model based on several combination of feature subsets.
But, experimenting with linear model, we could eliminate few features, as they had no correlation or some feature suppressing the correlation of other features. For example, jam_count had negative correlation with dow_conf. The dataset of jam_count was around 100k and was not providing us with much improvement in RMSE. After validating the same with Random Forest, we decided to drop jam_count.

Random Forest - Random Forest constructs multiple decision trees from a subset of feature set. Random forest was slow compared to linear model, so we decided to test with 10% data for feature selection purpose. It took approximately 40 minutes to compute the model for this subset of data locally. We used default parameters for feature selection as per the RandomForest r package. We tested each feature by including them in the model and testing if the variance explained various negatively or positively.

- *artist_familiarity* - artist familiarity provided good improvement in the RMSE and explained variance. Also, intuitively this feature should have good correlation with song downloads.
- *norm_song_hotness* - We found the song hotness data to be error prone in [0 - 0.1]. This property was reducing the model variance percentage. We experimented by converting < 0.1 values to 0 and training the RF model. This provided us good results in our model variance.

- *mean_price* - Mean_Price had a very high correlation with song downloads in linear model and provided us good results in random forest model as well
- *artist_hotness* - Artist Hotness provided us good correlation with song downloads in both linear model and random forest model.
- *norm_duration* - Raw values of song duration does not add much value to the model, so we tried creating range of duration and also filtered with start and end of fade_in and fade_out information. With this, we could achieve negligible improvement in the model and hence dropped this feature
- *norm_year* - Logically, Year should have good effect on downloads as old songs would have comparatively low downloads compared to the songs during the internet era. But we found the year data includes lot of null values. After removing na from the data, we could achieve negligible improvement. This might be due to lack of year information for most of the songs and hence we dropped this feature.
- *bin_plays* - The raw play count feature was not a good feature as many popular songs did not have high play count. But we had good amount of data for plays and hence tried creating binary field of play. If a song had a play count of greater than 1, then we consider 1, 0 otherwise. This provided good improvement in the model.

After lot of experiments on different combinations we finalized the following features - norm_song_hotness, artist_familiarity, artist_hotness, bin_play, mean_price and loudness with **% Var explained: 93.51** for the random forest model. In order to validate the feature set, we ran the random forest model on a subset of data which contains only downloads for excellent confidence. This dataset with the final feature set provided us **% Var explained as 99.78**. We compressed the entire dataset size by rounding the decimal points of numerical features to 6 digits as this would not affect the accuracy. and removed the quotes. The next step was to train the entire dataset and tune the RF model parameters accordingly.

Random Forest Parameter Tuning - After finalizing the feature set, the aim was to reduce the model size with the best accuracy possible as the deployment package size limitation is 100MB. We could improve the accuracy of the model and control the size of the model, by choosing appropriate parameters for the random forest function. The 2 parameters to be tuned were the *number of trees* and the *depth of the tree*. The plot of *error vs trees* in figure 4, shows that the error rate is minimized and saturates for ***number of trees*** **> 100**. In order to find the exact tuning parameters for entire dataset, we trained the model with a range of 100-200 trees as initial configuration. The initial parameter range was determined by the analysis of 10% of dataset. In order to find the correct tuning parameters we had to train entire dataset. Since it is computation intensive, we ran these jobs on AWS EMR cluster. We used Spark MLLib - Random Forest to get the random forest model.

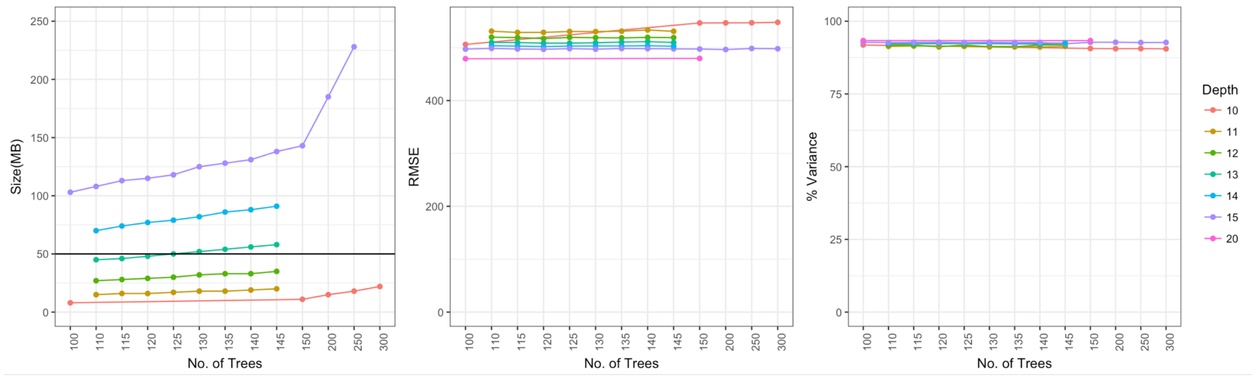


Figure 5: Random Forest Parameter Tuning

- **Initial-Configuration** - Number of trees - {100,150,200,250,300} and depth - {10,15,20}
  We executed the jobs with the above configuration with a split of 9:1 for training and testing. It took 11 hrs 57 minutes to compute for all combinations of the parameters on entire dataset. We noticed that the model size increased exponentially as the depth parameter is increased and compute time increases with number of trees and depth. We reduced *number of trees* range to 100-150 and *depth* range to 11-15.

- **Tuned-Configuration** - Number of trees - {110,115,120,125,130,135,140,145} and depth - {11,12,13,14,15}

  Within the reduced range, we ran with above combination of parameters. The execution time was 12 hrs 2 minutes for all the combination of parameters. The resulting metrics are plotted as shown in the figure 5. Here, we can notice that the RMSE and variance doesn't vary much as we increase the number of trees and depth. Hence, based on the size of the model generated, we decided on the parameters as **Number of trees = 110** and **depth = 13** as the size of the model is 45MB.

With number of trees = 110 and depth = 13, the training model for the entire dataset was generated in 16 minutes on AWS EMR cluster of 4 nodes.

3. Predicting the Song Downloads

Given the input data that is <*artist name; title*>, we need to build the test model in order to predict the song downloads. We normalize the text by removing any punctuation, additional spaces and lowercase. Considering the cases where we don't have a match for the artist name or title, we built a fuzzy model to approximate any small mismatch in the text matching. To get the best possible search with high accuracy and performance, we used Apache Lucene to build a fuzzy search model. Apache Lucene is a full-featured text search engine library written entirely in Java. We use levenshtein distance for string matching with edit distance of 3 atmost. The fuzzy search model works as per the flow chart described in figure 6. If we do not find any match with the given artist name and title, we predict 0 as the downloads because we assume it as an invalid input.

For Example, for the input "the pointer sister;who do you love", an exact match search was not found for the artist name, it did a fuzzy match with "the pointer sisters" and predicted 9135 as download value. The actual computed download value for the combination is 9261.

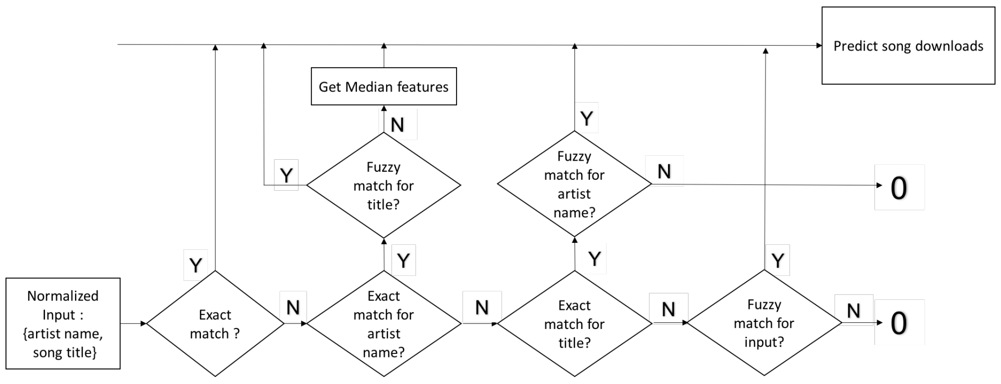


Figure 6: Fuzzy search model

4. Validation - Performance and Accuracy

To evaluate the model accuracy, we created a input data set with 100 songs in each confidence level randomly picked from the entire dataset. We bucket these confidence levels based on the calculated download values i.e, < 5000, 5000 - 10000, > 10000. We measured the absolute error between the predicted and our computed dow_conf values. Figure 7 represents the graph of the distribution of error in the prediction, where the x-axis represents the bins of confidence level and the download value range and the y-axis represents the absolute error. For *very good* and *excellent* confidence levels, the absolute error is comparatively low. But, for lower confidence levels like *good* and *average*, the absolute error rate is high. From the error rate distribution of average and good confidence, we note that the calculated downloads of those confidence levels is not captured by the model.

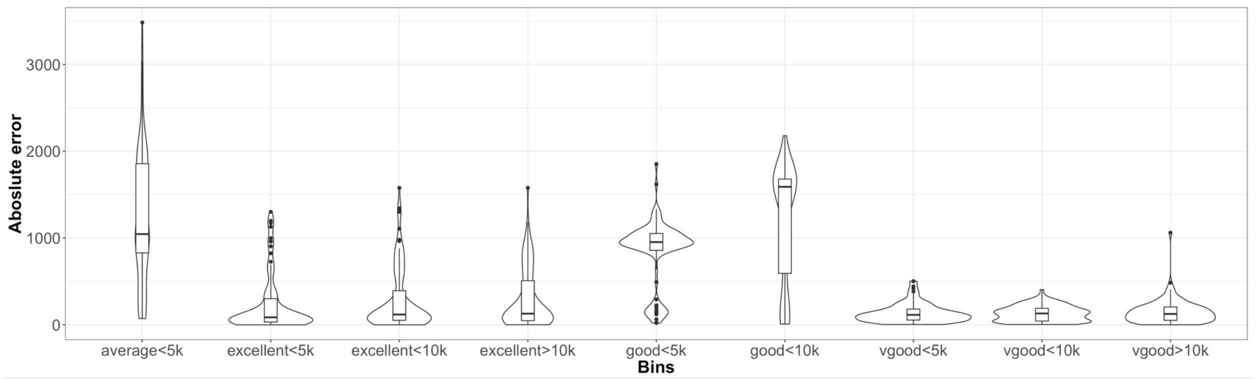


Figure 7: Error distribution of Song download across confidence levels

The above input data had 900 values across all confidence levels. We did not have sufficient data in average >5000 and good >10000 bins.The execution time for the same is 100 minutes on our local machine. Higher execution time mainly due to the fact that we read the input values from the standard input, and prediction for each input is a separate spark job.

5. Conclusion

Predicting Song downloads given the downloads with various confidence levels was little tricky. We had to incorporate the confidence level of the downloads in the trained model with an assumption that the download numbers are amplified for lower confidence levels. Extracting various features of varied dataset involved some effort. Collating data from multiple sources represented in various formats was time consuming. Many of the features did not have linear correlation with the download numbers and we had to consider combination of features to train the model. Random forest was one of the good model for our case.

After Training the model our next task was to test the model with artist name and title as the input, we had to build a fail-safe fuzzy model to in-corporate minute mismatches with the name and title. Lucene gave us good text matching capabilities to build this feature. The last challenge was to deploy the jar with the model and lookup data within 100MB limit. We had to let go some level of accuracy of the model to keep the model size within the limit.