# Predicting Song Popularity on the Million Song Dataset

**Rohan Bansal**
Carnegie Mellon University
Pittsburgh, PA
rohanb@andrew.cmu.edu

**Siddhant Jagtap**
Carnegie Mellon University
Pittsburgh, PA
sjagtap2@andrew.cmu.edu

**Vaidehi Joshi**
Carnegie Mellon University
Pittsburgh, PA
vaidehij@andrew.cmu.edu

## 1 Introduction

### 1.1 Motivation

Large scale production level systems often deal with multiple petabytes of data, a reality for multiple music streaming services such as *Apple Music, SoundCloud, Spotify, etc.* With the explosion of such services it has become increasingly pertinent for consumers, producers and artists to understand which song will be a hit and where improvements could be made to make a song more appealing to an audience.

### 1.2 Problem Statement

The goal of our project is to investigate characteristics which propel a song to the top of trending charts while optimising for effective use of the large dataset. We undertake this analysis by building a pipeline capable of handling large amounts of data in the *Million Song Dataset(MSD)* and delivering reasonable results in terms of error.

The goal of the pipeline is to successfully predict and subsequently rank which song would be popular. To solve this problem we posed a multitude of questions: Which features are pertinent in the analysis of the song popularity? What degree of importance can each parameter play in this analysis? Which algorithm is best suited for the task at hand? Which algorithm has the best performance? What are the trade offs between performance and computational complexity between the algorithms?

### 1.3 Dataset

The MSD [1] has been a staple of academic research and following instructor guidance we decided to isolate our analysis to this dataset. The dataset has been made publicly available courtesy of Columbia University. They have hosted a version of the complete dataset on *Amazon Web Services(AWS)* S3.

The dataset is rich with an expansive set of meta-data, audio features, similar artist data among a large list of supporting features. The label parameter for our machine learning pipeline was the feature *'song hotness'*. This label gives a good representation of how popular a song is [1].

The raw dataset contains about 270GB of .h5 files. While experimenting, we used a small subset of this full dataset which contains 10k songs. During this experimentation phase we had considered all the features available in the dataset and for our final analysis we selected a fixed number of features, which is described in methodology.

The schema consists of 35 features, listed on the MSD website [1]. We have additionally used the library code provided by [1] for the processing of .h5 files. A complete list of features for the given schema is given in Fig.4

## 2 Methodology

### 2.1 Machine Learning Pipeline

In our experimentation we have conducted two separate process flows. The first flow is used to determine which model and what parameters for a given model works best, an overview described in Fig 1. The second one is to investigate how feature extraction techniques (i.e. Principal Component Analysis(PCA)) play a role in the model performance for our regression task and the relevant trade-offs from a computational standpoint can be found in Fig 5.

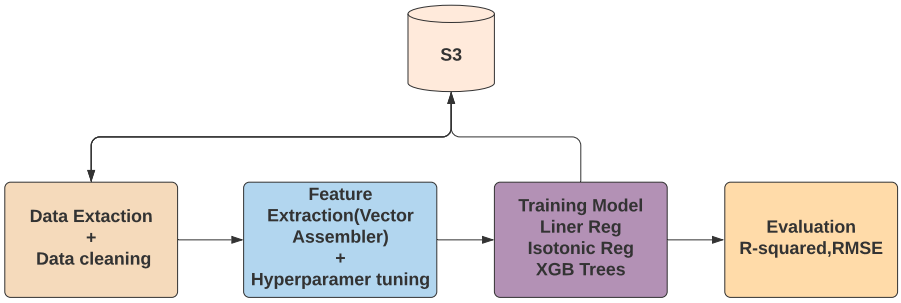



Figure 1: Process to evaluate three models: Liner Regression, Isotonic Regression, and Gradient Boosted Trees (XGB); Subsequently determine best parameters for each model.

Processing h5 files was the first step of the pipeline. With the processed data available in machine readable format, the data was cleaned by removing duplicate rows. For rows with missing/corrupted values, the cells were filled with the average value for that column. In our initial experimentation, we used scikit-learn to determine the best parameters to predict song hotness in Fig 6. Following which, the data was vectorized and a range of parameters were selected to perform grid search using four fold cross validation. Optimal hyper-parameters were deployed to train each of the models. Finally, the evaluation was done on this optimal model.

### 2.2 Modelling Techniques & Evaluation Metrics

Liner Regression, Gradient Boosted Trees[3] and Isotonic Regression[5] were the three key algorithms that were evaluated. Linear Regression served as a baseline for our evaluation task. The algorithm also allowed us to infer the degree of linearity in the dataset and how much of this linearity could be effectively captured from features such as *artist_familiarity* and *year*. Subsequently, we wanted to train a model which generalises to our broader dataset. Here, we considered two techniques. XGB allowed to evaluate how well a model could mitigate the dispersion of predictions and minimize spread[3]. However, in this large dataset case, we also wanted to evaluate how a less computationally expensive regression model, like isotonic regression, performs and subsequently evaluate the trade-offs.

For predicted values $y'$ and labels $y$, for $N$ data points we define our first evaluation metric Root Mean Squared Error($RMSE$) eq(1). We also define $R^2$ eq(2). Here, RSS is the sum of the residuals and TSS is the sum of squares for a set of predictions.

$$RMSE = \sqrt{\frac{\Sigma_{i=1}^{N}\left(y_i' - y_i\right)^2}{N}} \tag{1}$$

$$R^2 = 1 - \frac{RSS}{TSS} \tag{2}$$

$R^2$[2] is a statistical measure of how close the data is to the fitted regression line. It is also known as the coefficient of multiple determination for multiple regression and describes how well the model is able to capture the variability in the data. RMSE is a standard benchmark used for evaluation of regression problems and describes the net error in each of the predictions.

# 3 Computation

## 3.1 Initial computational challenges

Initially, a small subset of the Million Song Dataset consisting of around 10,000 rows of songs was used to set up a pipeline for testing different models on 'Runtime 6.6 ML cluster' on the community edition of Databricks. Prior to this, we converted the .h5 files in the dataset to .csv files on our local machine and found out the top 10 features in the data which are most correlated to our target feature 'song hotness' on Google Colab (12 GB RAM).

While working with the small data subset on Databricks, we tried explicitly caching data from pre-processing steps only to later discover that pyspark.ml package does this caching in a much more efficient manner anyway; thus leading to multiple memory errors and cluster timeouts multiple times because of the size of the data we were dealing with. This was exacerbated by the cluster capacity on Databricks. Later, the data was structured and processed in a way to avoided caching. Utilising the 'Pipeline' class solved issues related to cluster timeouts and made computations on the smaller subset easier. We faced similar issues while caching, on EMR clusters as well. Saving test and train data in parquet files mitigated issues faced while caching objects in memory. This solution also improved the overall computation time and performance.

Table 1: AWS Services used to build, train and evaluate the machine learning pipeline

| Service | Instance | Specification |
|---|---|---|
| S3 | – | 500 GB (Volume) |
| EC2 | t3.2xlarge | 8vCPU, 32GB RAM |
| EMR | 4 - m5.xlarge | 4vCPU, 16GB RAM |

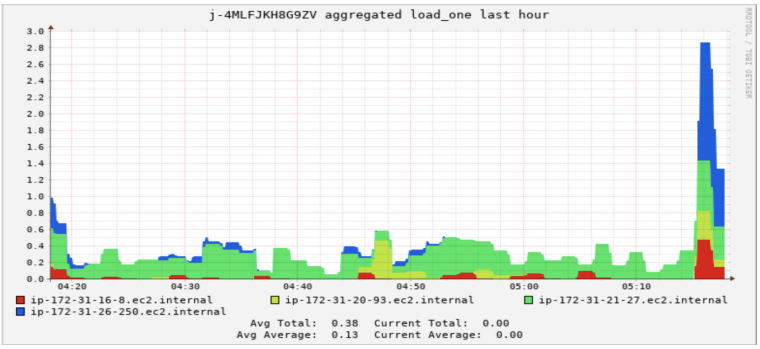



Figure 2: Aggregated load over one hour across all nodes in the EMR cluster visualized using Ganglia

## 3.2 Language/ Libraries used

The conversion of data from .h5 format to .csv format was done using python 3.7. The feature extraction to get the top 10 most relevant features was done using *scikit-learn* and *pandas* libraries in python. Reading and relevant pre-processing of .csv files was done using PySpark dataframes. Linear regression, Isotonic regression and Gradient boosted tree models were run using ml library in *PySpark*. We used Ganglia installed on the EMR cluster to visualize the cluster memory and RAM usage while training these models; an example is shown in Figure 2.

## 3.3 Overall Costs/ Time

We spent $49.27 overall on AWS as combined costs for compute and S3 storage. Data conversion (reading from S3, processing and writing back to S3) using an EC2 instance took around 2.5 hours to process a single alphabetical folder. Model training took within 1 hour initially per model. Later we ignored caching and saved our train and test split data as parquet files to s3 and read it again before running models. This reduced the time to train each model on the full dataset to around 5-10 minutes which increased the efficiency of training and testing models on the large dataset.

# 4 Results

## 4.1 Model Performance

As described in sections 1 and 2, we created an end-to-end machine learning pipeline to predict song hotness using the Million Song Dataset. The target feature (song hotness) in the dataset lies in $y \in [0, 1]$, where the upper bound represents a popular song. In this project, we have focused on three regression algorithms, namely Vanilla Linear regression, Gradient Boosted Trees regression and Isotonic regression. Table 2 reports the metric scores on the test dataset.

Table 2: RMSE and $R^2$ scores for all models; The scores have been computed by evaluating the trained models on the test set which is 25% of the full dataset.

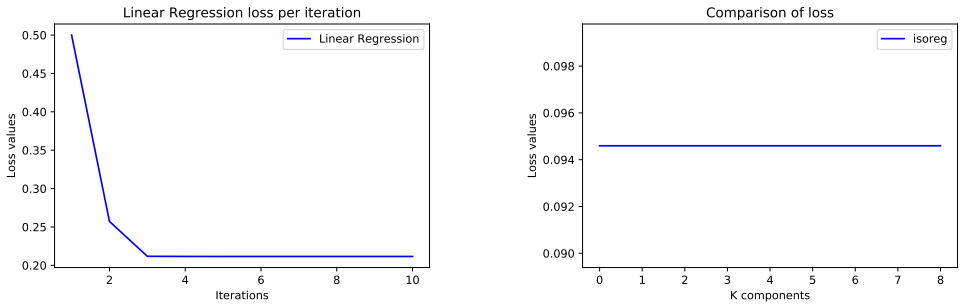
| Regression Model | RMSE | $R^2$ |
|---|---|---|
| Vanilla Linear Regression | 0.0619962 | 0.67087 |
| Gradient Boosted Trees Regression | 0.0578572 | 0.62703 |
| Isotonic Regression | 0.0578205 | 0.62746 |

From the above results it is evident that all three algorithms have a similar performance on the test dataset. However, isotonic regression appears to outperform the other two algorithms by a small margin. Therefore, we shortlist the isotonic regression model to conduct further experiments.

It is interesting to analyse why the isotonic regression model performs better as compared to other methods. Unlike vanilla linear regression, which attempts to fit a linear hyperplane to the data sample, isotonic regression fits a monotonic (non-increasing or non-decreasing) line/surface to the data. This ensures that the final hypothesis includes some non-linearity while evaluating the model on held-out data. Gradient Boosted trees method makes use of decision tree algorithms such as Classification and Regression Trees(CART)[4] which are computationally expensive.
Hence, we posit that despite the marginal difference in RMSE values, isotonic regression proves to be superior in terms of capturing non-linearity in the dataset as well as computational cost.

## 4.2 Visualization

This subsection explains the results obtained with the help of plots. Moreover, we also make interesting observations and discuss possible explanations for the same.




(a) (b)

Figure 3: Linear Regression and Isotonic Regression Loss Plots

Fig 3a. shows the reduction in training loss against number of iterations for the Vanilla Linear Regression Algorithm. As the model appears to converge in less than five iterations, we can conclude that the dataset has an approximately planar structure with respect to the features used in this project.

We also performed Principal Component Analysis on the dataset and used the lower-dimensional data to train the isotonic regression model. Fig 3b. illustrates the effect of k ,i.e number of principal components, on the RMSE value for the test dataset. It is worth noting that the RMSE remains constant irrespective of the number of principal components used to train the model. This implies that one of the features in the dataset is heavily correlated to the target feature, corroborating our initial analysis as seen in Fig 6.

## References


[1] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[2] A. Colin Cameron and Frank A. G. Windmeijer. R-squared measures for count data regression models with applications to health-care utilization. *Journal of Business  Economic Statistics*, 14(2):209–220, 1996.

[3] Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016.

[4] Dan Steinberg and Phillip Colla. Cart: classification and regression trees. *The top ten algorithms in data mining*, 9:179, 2009.

[5] Ryan J. Tibshirani, Holger Hoefling, and Robert Tibshirani. Nearly-isotonic regression. *Technometrics*, 53(1):54–61, 2011.

# Appendix

## A: List of Features

| Field name | Type | Description | Field name | Type | Description |
|---|---|---|---|---|---|
| analysis sample rate | float | sample rate of the audio used | loudness | float | overall loudness in dB |
| artist 7digitalid | int | ID from 7digital.com or -1 | mode | int | major or minor |
| artist familiarity | float | algorithmic estimation | mode confidence | float | confidence measure |
| artist hotttnesss | float | algorithmic estimation | release | string | album name |
| artist id | string | Echo Nest ID | release 7digitalid | int | ID from 7digital.com or -1 |
| artist latitude | float | latitude | sections confidence | array float | confidence measure |
| artist location | string | location name | sections start | array float | largest grouping in a song, e.g. verse |
| artist longitude | float | longitude | segments confidence | array float | confidence measure |
| artist mbid | string | ID from musicbrainz.org | segments loudness max | array float | max dB value |
| artist mbtags | array string | tags from musicbrainz.org | segments loudness max time | array float | time of max dB value, i.e. end of attack |
| artist mbtags count | array int | tag counts for musicbrainz tags | segments loudness max start | array float | dB value at onset |
| artist name | string | artist name | segments pitches | 2D array float | chroma feature, one value per note |
| artist playmeid | int | ID from playme.com, or -1 | segments start | array float | musical events, ~ note onsets |
| artist terms | array string | Echo Nest tags | segments timbre | 2D array float | texture features (MFCC+PCA-like) |
| artist terms freq | array float | Echo Nest tags freqs | similar artists | array string | Echo Nest artist IDs (sim. algo. unpublished) |
| artist terms weight | array float | Echo Nest tags weight | song hotttnesss | float | algorithmic estimation |
| audio md5 | string | audio hash code | song id | string | Echo Nest song ID |
| bars confidence | array float | confidence measure | start of fade out | float | time in sec |
| bars start | array float | beginning of bars, usually on a beat | tatums confidence | array float | confidence measure |
| beats confidence | array float | confidence measure | tatums start | array float | smallest rythmic element |
| beats start | array float | result of beat tracking | tempo | float | estimated tempo in BPM |
| danceability | float | algorithmic estimation | time signature | int | estimate of number of beats per bar, e.g. 4 |
| duration | float | in seconds | time signature confidence | float | confidence measure |
| end of fade in | float | seconds at the beginning of the song | title | string | song title |
| energy | float | energy from listener point of view | track id | string | Echo Nest track ID |
| key | int | key the song is in | track 7digitalid | int | ID from 7digital.com or -1 |
| key confidence | float | confidence measure | year | int | song release year from MusicBrainz or 0 |

Figure 4: List of features and their description in the Million song dataset

## B: Machine Learning Pipeline

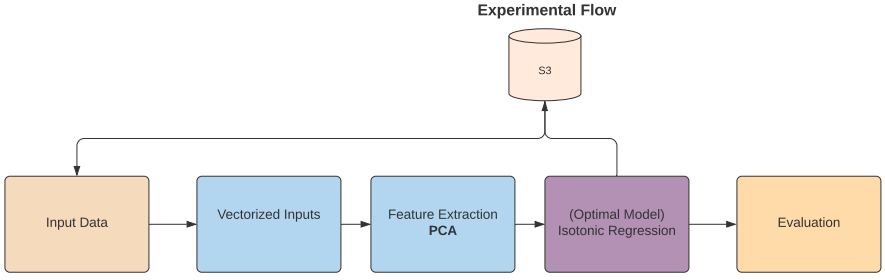



Figure 5: Process to evaluate feature extraction techniques(PCA) for the dataset.

**C: Feature Importance**

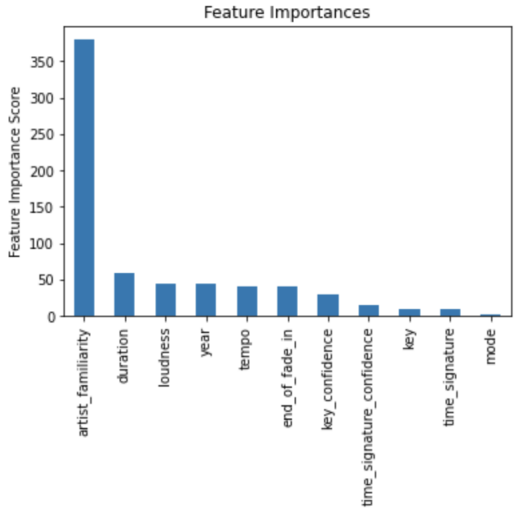



Figure 6: List of features and their importance's using Scikit-Learn feature importance API