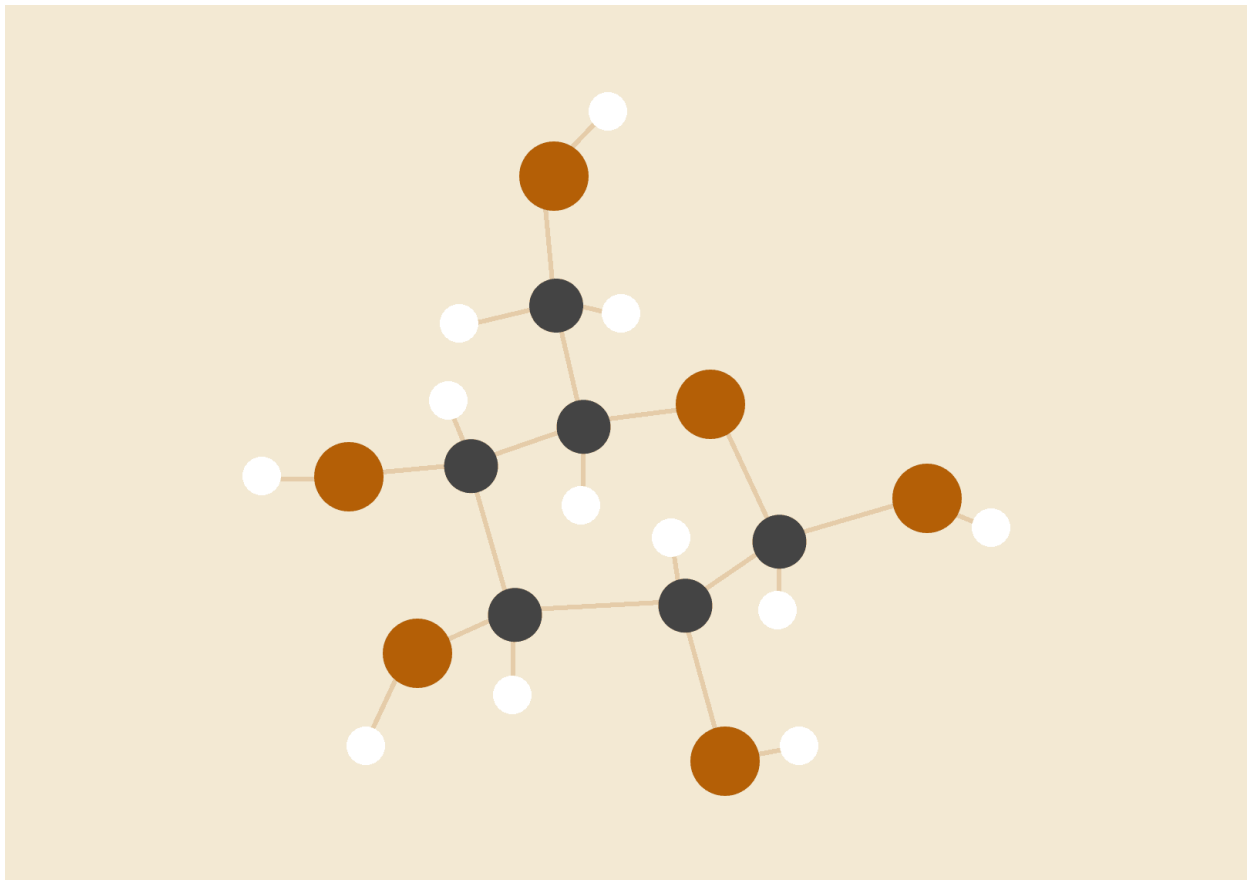


# CSE 740: Project Report

Analysis of Million Song Dataset using Spark



**Harsha Sudarshan [50169593]**

**Sahil Dureja [50168872]**

**Yashas Gowda[50168854]**

## **Index of contents**

<b>Index of contents</b>	<b>2</b>
<b>PROJECT GOALS</b>	<b>3</b>
<b>HIGH LEVEL COMPONENTS OF THE PROJECT</b>	<b>3</b>
<b>PROJECT LIFECYCLE</b>	<b>4</b>
<b>DATA</b>	<b>5</b>
<b>ANALYSIS OF DATA USING ML</b>	<b>10</b>
<b>CONFIGURATION, SET_UP AND DEPLOYMENT USING GOOGLE CLOUD</b>	<b>24</b>
<b>CONCLUSION</b>	<b>28</b>
<b>REFERENCES</b>	<b>29</b>

## 1. PROJECT GOALS

Use Machine learning methods to understand and visualize the trends of features of popular songs in the past few decades using a publicly available large dataset and implement it using modern Big Data tools on local as well as Cloud platforms

## 2. HIGH LEVEL COMPONENTS OF THE PROJECT

At a high level, the project components can be listed as:

- Data Source
  - Million Song Dataset (MSD)
  - Billboard weekly chart Data
- Big Data Tool
  - Apache Spark
    - Pyspark API
    - Spark version 1.6
- Implementation platforms
  - Local Machines
    - Windows
    - Linux VM
  - Cloud platform
    - Databricks community edition
    - Google cloud Dataproc
- Data exploration tool
  - Jupyter notebook with python
- Machine Learning Algorithms
  - Linear Regression
  - Logistic Regression

### 3. PROJECT LIFECYCLE

The project was developed and implemented in the following steps:

1. Collect and store required data
  - a. MSD dataset available online at [labrosa.ee.columbia.edu/millionsong/](http://labrosa.ee.columbia.edu/millionsong/)
  - b. Store MSD dataset in cloud storage
  - c. Crawl Billboard weekly chart data
2. Perform EDA on data sources using data exploration tools
  - a. Download sample of MSD dataset
  - b. Load MSD sample data and Billboard collected data
  - c. Check viability of using Billboard data as supplementary data source
3. Build code to extract required data into best suited format
  - a. In the absence of native support in spark for HDF5 files, it was converted to csv format
4. Prototype development on Spark using Local and Databricks platform
  - a. Databricks Community Edition was the main development platform
  - b. Spark installation on local machines(Windows and Linux VM) acted as supplementary development platform
5. Build and test code for cluster initialization on Google Cloud
6. Deploy the prototype code on cloud and fix any issues
7. Change the cluster configuration and check processing time

More details on components and their usage/context are provided below.

## 4. DATA

### Million Song Dataset (MSD):

Million Song Dataset was selected as the primary data source. It was selected due to its volume, variety, popularity in academia as well as industry and no-cost availability.

Major statistics of the dataset:

Parameter	Quantity
Data	273 GB
Songs/Files	1,000,000
Unique Artists	44,575
Year range	1922-2012
File format	HDF5(.h5 files)

More details are available here: <http://labrosa.ee.columbia.edu/millionsong/faq>

### Individual File details:

Each of the 1M files are stored in HDF5 format. HDF5 is a hierarchical data model and format that supports wide variety of datatypes and is designed for high volume and complex data.

Tools/API used to access the files:

- HDFView GUI

- Python libraries such as pytables and h5py

### Field list:

Each file has 55 fields per-song.

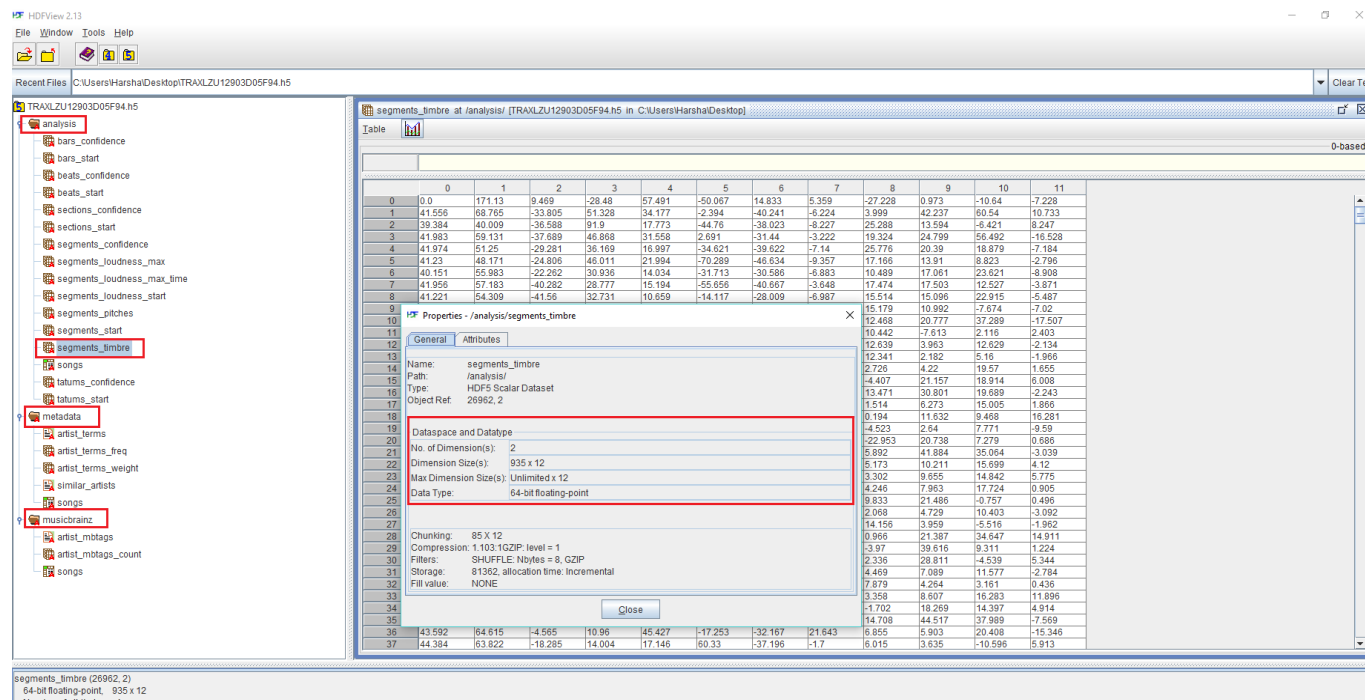
analysis_sample_rate	artist_7digitalid
artist_familiarity	artist_hotttnesss
artist_id	artist_latitude
artist_location	artist_longitude
artist_mbid	artist_mbtags
artist_mbtags_count	artist_name
artist_playmeid	artist_terms
artist_terms_freq	artist_terms_weight
audio_md5	bars_confidence
bars_start	beats_confidence
beats_start	danceability
duration	end_of_fade_in
energy	key
key_confidence	loudness
mode	mode_confidence
num_songs	release
release_7digitalid	sections_confidence
sections_start	segments_confidence
segments_loudness_max	segments_loudness_max_time
segments_loudness_start	segments_pitches
segments_start	segments_timbre
similar_artists	song_hotttnesss
song_id	start_of_fade_out
tatums_confidence	tatums_start
tempo	time_signature
time_signature_confidence	title
track_7digitalid	track_id
year	

**Table 2.** List of the 55 fields provided in each per-song HDF5 file in the MSD.

The fields are not available in flat file-like structure but instead are grouped into three main groups in HDF5 file:

1. Analysis
2. Metadata
3. Musicbrainz

A view of the file structure using the HDFView GUI is given below.



The grouping tree structure is visible on the left pane, with the three groups highlighted. Also highlighted in the left pane is a field - segments\_timbre - the contents of which are visible in the background on the right pane. On the foreground in the right pane is a window that is displaying the properties on the field.

Unfortunately, the Echonest API documentation, using which the dataset was created, is no longer accessible. However relevant information on the fields in each file can be found in the paper by Thierry Bertin-Mahieux, who created the dataset - [http://tbertinmahieux.com/Papers/tbm\\_thesis.pdf](http://tbertinmahieux.com/Papers/tbm_thesis.pdf)

### Implications of file format:

- No native read support for HDF5 file format in spark
  - Attempt to perform parallel read .h5 files hosted on cloud storage in a cloud instance was not successful
  - Necessitated change in ETL strategy
    - Python was used to extract required fields and convert it into .csv format

- In the project, 24% of the original dataset was sampled and used for analysis
- Dimensions of fields not constant
  - Fields with vector data with variable number of rows
    - Ex. segments\_timbre - a vector with dimensions (m,12) where m was the number of segments
  - Necessitated usage of descriptive statistics to quantify the vector data
    - Mean was used as the descriptive measure
- Field selection required to consolidate data for analysis
  - Fields were selected from songs vector(metadata group) and segments\_timbre
    - Segments\_timbre was selected as it was noted as one of most important quantitative information on a song in the above paper
  - Further details of the features used for analysis are provided in the next section

### **Billboard weekly chart dataset (BB):**

The idea was to use the ranking of songs from Billboard weekly charts ('Hot 100') to filter and select the popular songs from MSD and use the ranking as one of the features to analyze trends in popular music.

The data was crawled using an unofficial python API hosted at <https://github.com/guoguo12/billboard-charts> . The data collected consisted of:

1. 292,244 records
2. Date range: current date backwards to 1960

### To check viability of using BB to filter MSD data:

- Take subsample of 10k records from MSD data
- Group BB data based on artist and song title
- Merge subsample MSD data with BB
  - With condition as artist and song title has to match



- Measure the size of the filtered data

Using the above procedure, the size of filtered dataset was 192 songs out of 10k (less than 2%). Because the size of merged dataset would result in elimination of majority of the MSD data, we decided against using the BB data in our further analysis.

## ANALYSIS OF DATA USING ML

Columns extracted from h5 format to csv

Info features:	Scaler features	2D feature for each song: Timbre
'SongNumber', 'SongID', 'AlbumID', 'AlbumName', 'ArtistID', 'ArtistLocation', 'ArtistName', 'Title',	'ArtistLatitude', [High missing values] 'ArtistLongitude', [High missing values] 'Danceability', [High missing values] 'Duration', 'KeySignature', 'KeySignatureConfidence', 'Tempo', 'TimeSignature', 'TimeSignatureConfidence', 'Year', [High missing & 0 values] 'Song_hottnesss', [ LABEL ] 'Artist_hottnesss', [IMP FEATURE] 'Artist_familiarity', [IMP FEATURE]	Org. Size: 1000 X 12 Mentioned as one with important information about the songs segments  Converted to: 1 X 12 taking mean column wise  12 rows for Timbre 0-11 features columns

Process followed in cleaning and ML overview

- > Removed null, nan, empty.
- > Converted to float

### 1) First Study

- > Used song hotness as label
- > Classified as 0 and 1 above and lower the median value for classification
- > used actual value for regression and RMSE for error
- > Split 80 10 10 for train test validation set.

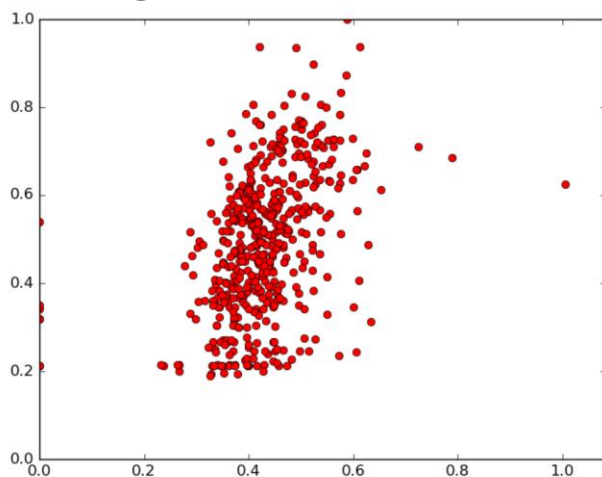
### 2) Second Study

- > Used Year as Label
- > Used actual value for regression and RMSE for error
- > Split as 0 label for 1990-2000 and 2001-2010.
- > Equal sampling for 0 1 classification.

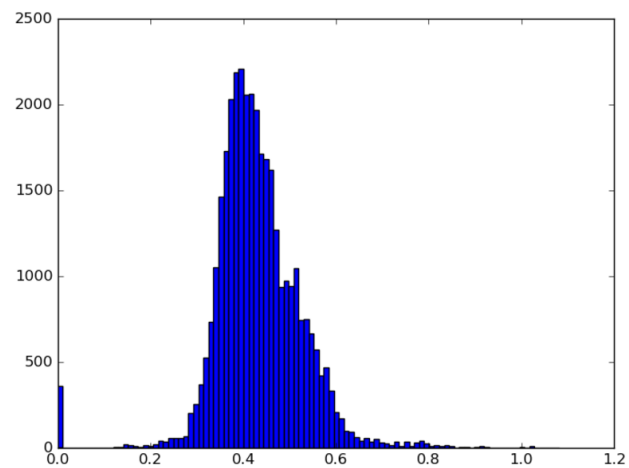
## 1) First Study

Some Plots between Song Hotness vs some of other features

### a) Song Hotness vs Artist Hotness

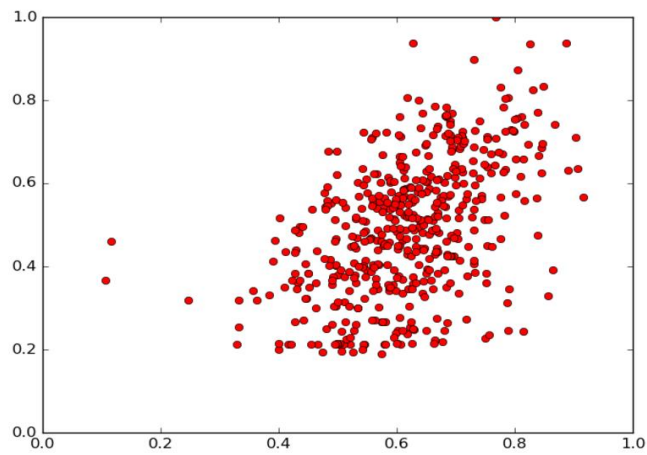


X: Artist hotness vs Y: Song Hotness

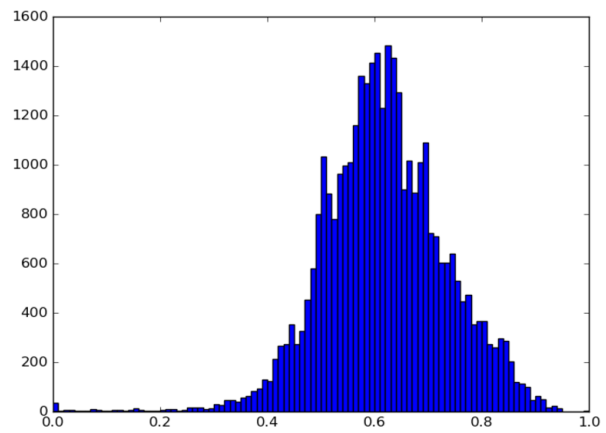


Artist hotness histogram

## b) Song hotness vs Artist Familiarity

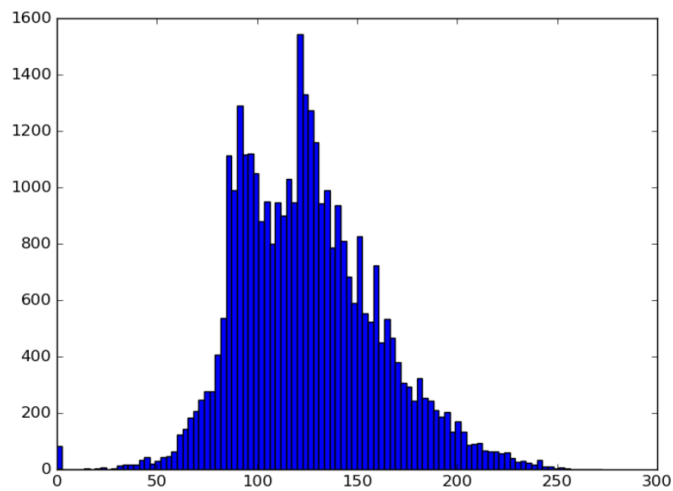


X: Artist Familiarity & Y: Song hotness

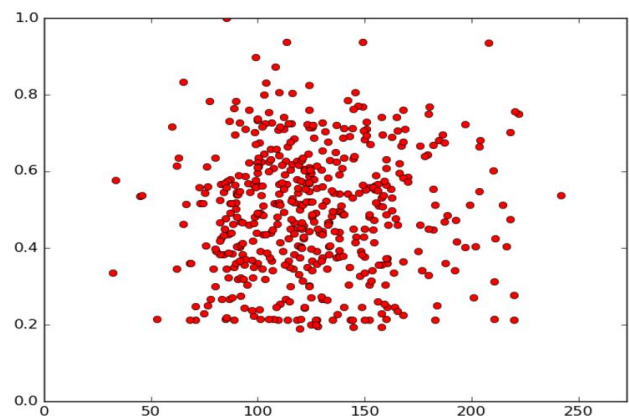


Artist familiarity histogram

## c) Song hotness vs Tempo

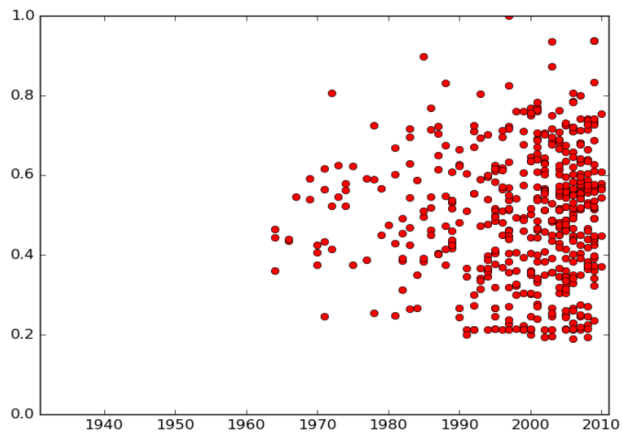


X: Tempo vs Y: Song hotness

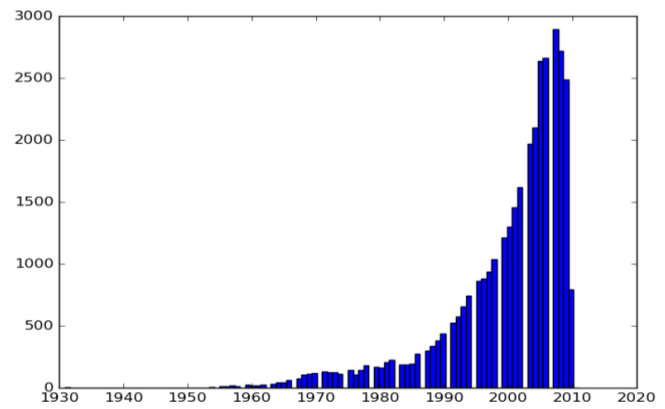


Tempo histogram

#### d) Song hotness vs Year

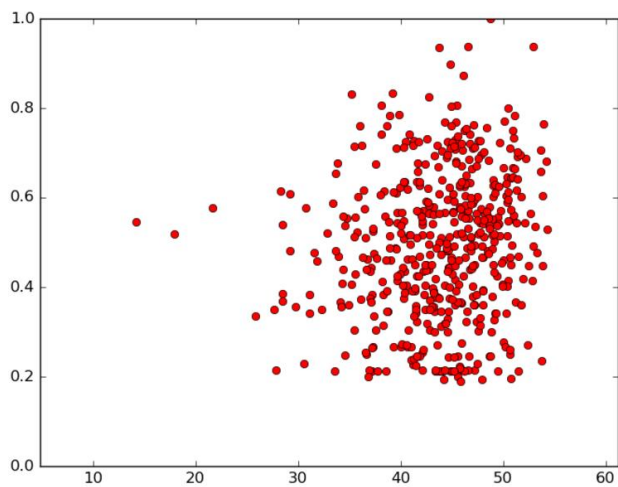


X: year vs Y: Song hotness

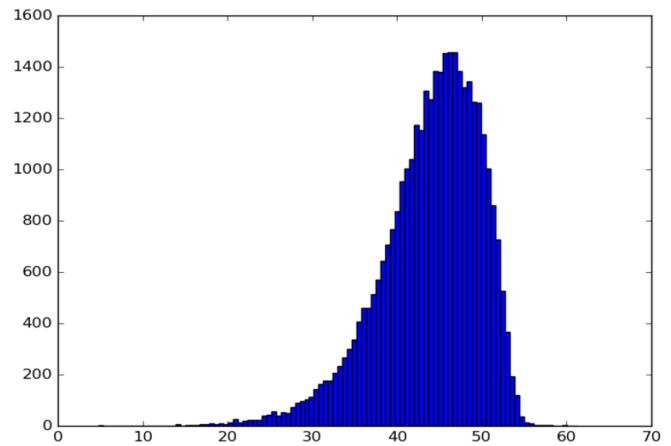


Year histogram

#### e) Song hotness vs Timbre[0]

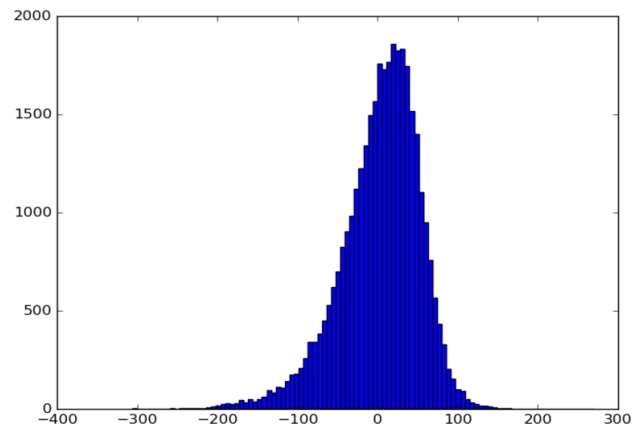
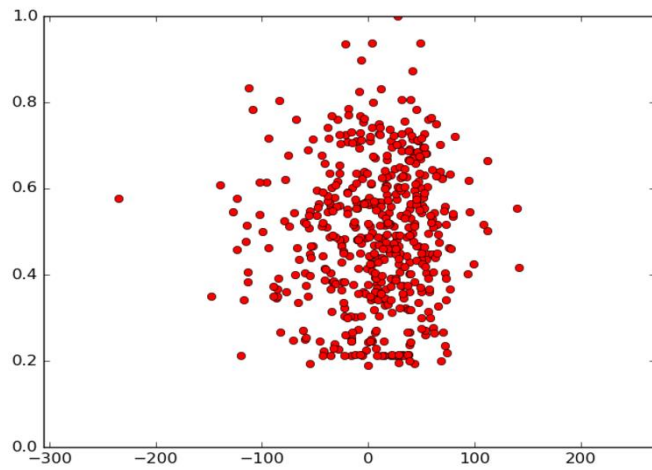


X: Timbre[0] vs Y: Song hotness



Timbre[0] histogram

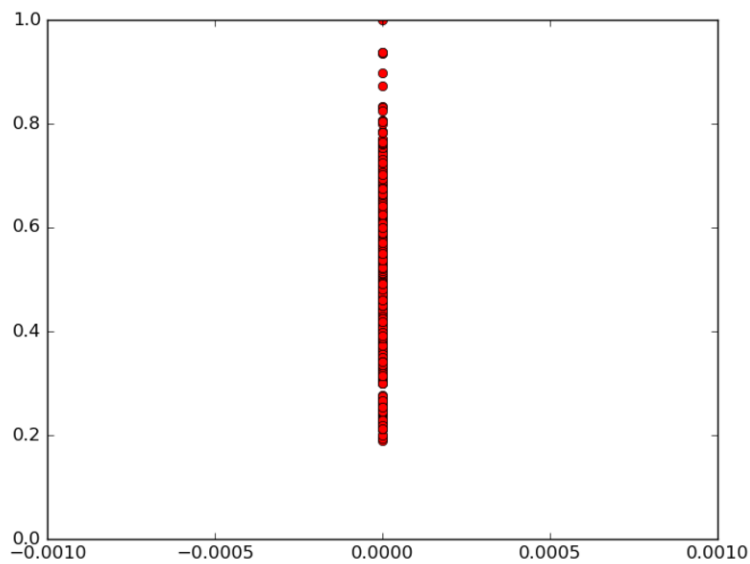
f) Song hotness vs Timbre[1]



X: Timbre[1] vs Y: Song hotness

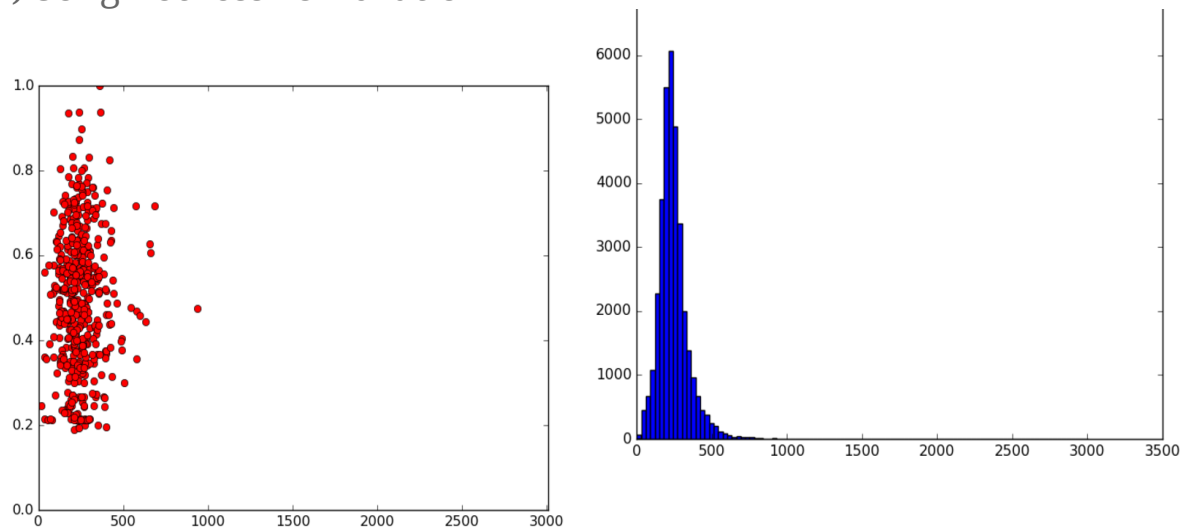
Timbre[1] histogram

g) Song hotness vs Timbre[1]



X: Danceability vs Y: Song hotness

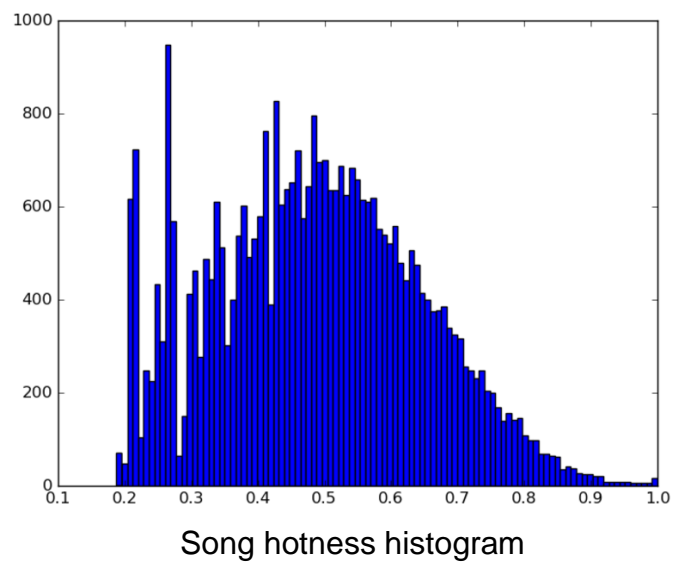
## h) Song hotness vs Duration



X: Duration vs Y: Song hotness

Duration histogram

## i) Labels: Songs hotness histogram



Song hotness histogram

## Weights of 21 features with song hotness as label

### Regression: Linear Regression

Coefficients: [0.394833778939,0.390069363361,2.00462651292e-05,-0.000193330580703,-0.00086810982572,0.000475925745666,0.00331502483882,0.000370778821461,0.00229141243298,-0.000188453739895,-4.92022178629e-05,-0.000114187263938,-0.000273463825,-0.000700790637103,0.000135684362559,-0.000216666981654,-2.00530469204e-05,2.72570278361e-05,-4.77269565176e-05,1.56787215113e-05,-1.92509361716e-05]  
Intercept: -0.774767897738

### Classification: Logistic Regression

Coefficients: [5.57505159267,5.50824262773,-4.70935247502e-05,-0.000463902158376,-0.0355307325631,0.0243533736022,-0.0343392233388,-0.00374355466338,0.0365716178706,-0.00295087221929,-0.0017739955206,6.12438651667e-05,-0.00422454645384,-0.0113599522766,0.00226237222224,-0.0046473993618,0.000273080030751,0.00459172753331,0.00083095229351,0.00118809111422,-0.000425547758531]  
Intercept: -0.0247471236914 Accuracy=71.44

Note: 1) **The results is for neither l1 or l2 norm**

**2) As we see only first two weights have some significant values. Two columns are artist hotness and artist familiarity respectively**

### Norm Used

ElasticNet: it is hybrid of lasso and ridge regression both

$$\alpha(\lambda\|w\|_1)+(1-\alpha)((\lambda/2)\|w\|_2^2), \alpha \in [0,1], \lambda \geq 0$$

Example: Lambda=0 means  
Neither L1 norm nor L2 norm

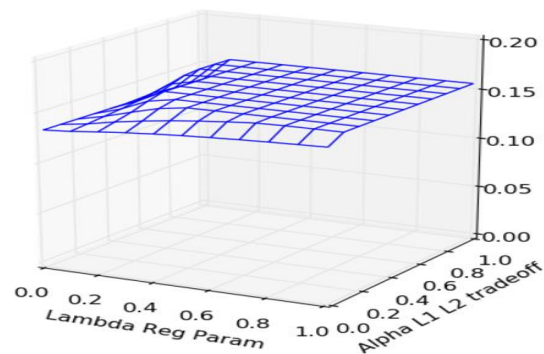
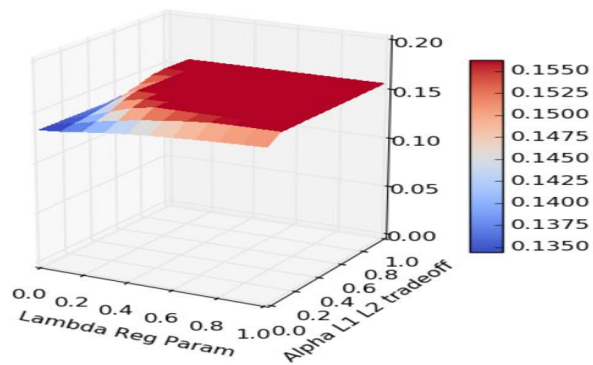
$\alpha$  set to 1, it is equivalent to a Lasso model. On the other hand, if  $\alpha$  is set to 0, the trained model reduces to a ridge regression model

In our results the applying any of L1 and L2 norm decreasing the accuracy or increase the RMSE error.

RMSE before linear regression just using the median as training set labels as prediction label  
0.156166121651

RMSE with linear regression

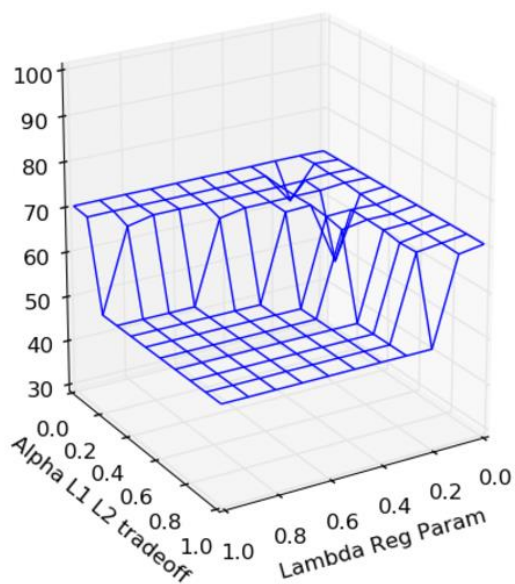
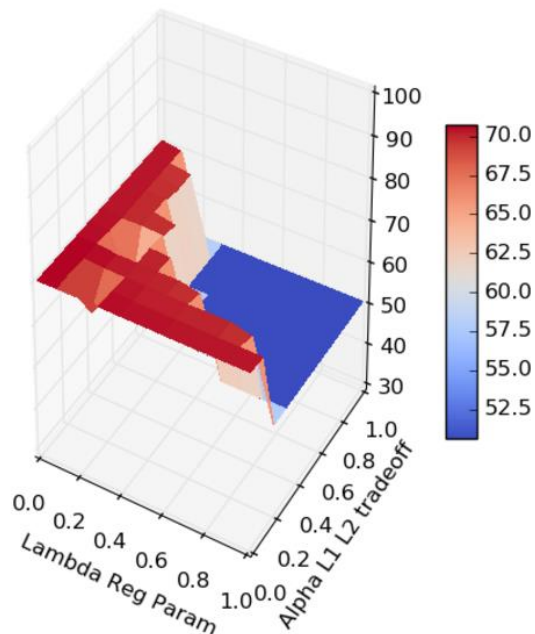
Lambda 0.0, Alpha 0.0 , RMSE value 0.133355694947



## Logistic Regression

Accuracy 70.94 % for Lambda 0.1, Alpha 0.1 gives the maximum for validation set

Test Set Accuracy 70.73 %





## Running just for two features Artist Hotness and Artist Familiarity

▶ (553) Spark Jobs	
0.3 : 0.2	[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
70.52 %	[ 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ]
Coefficients: [0.536485463491,0.608977065214]	[ 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 ]
Intercept: -0.111631771341	[ 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 ]
0.3 : 0.3	[ 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 ]
70.54 %	[ 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ]
Coefficients: [0.398005067136,0.509544162572]	[ 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 ]
Intercept: 0.00932249108361	[ 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 ]
0.3 : 0.4	[ 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 ]
70.46 %	[ 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 ]
Coefficients: [0.250397787312,0.405271947005]	[ 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. ]
Intercept: 0.137190355235	
0.3 : 0.5	

3D values

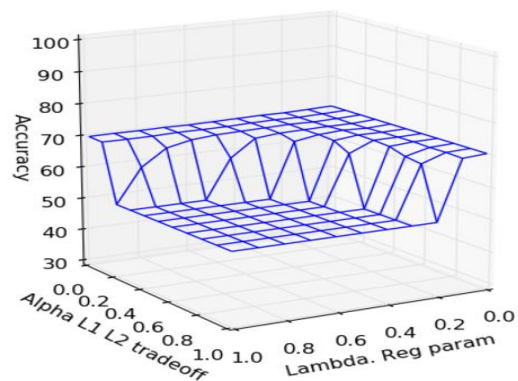
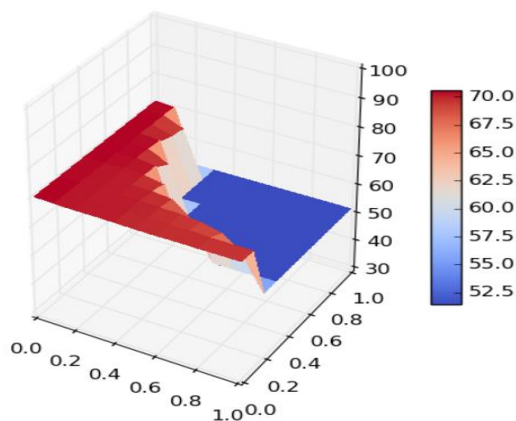
Values of Alpha or Lambda to plot

## 2 feature Only: Artist Hotness and Artist Familiarity

### Logistic regression

No norm Accuracy 70.66, Lambda 0.0, Alpha 0.0 , gives the maximum for validation set,

Test Accuracy 69.2 %



## Results from Random Forest

No of classes= 2, numTrees=3, featureSubsetStrategy="auto", impurity : gini, max Depth 4,

Accuracy: 69.52

## 2) Second Study

Taken Year as Label

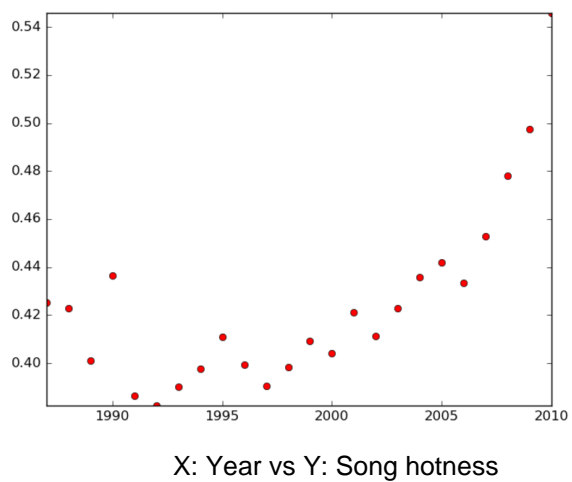
Plots of Group by Each Year and Mean for each column

Linear slope of every feature as compared to Year

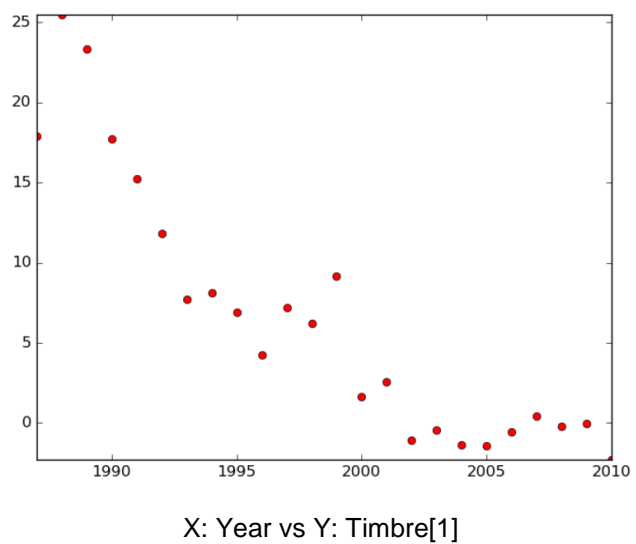
- #song hotness increasing average
- #artist hotness somewhat increasing
- #artish familiarity also increasing
- #Tempo stable
- #KeySignature stable slight decrease
- #KeySignatureConfidence decreasing
- #TimeSignature slight increase
- #TimeSignatureConfidence slight increase
- #timbre 0 increase
- #timbre 1 decrease
- #timbre 2 somedecrease
- #timbre 3 stable
- #timbre 4 slight increase
- #timbre 5 decrease
- #timbre 6 increase
- #timbre 7 stable zigzag
- #timbre 8 increase
- #timbre 9 stable zigzag
- #timbre 10 stable
- #timbre 11 stable

## Visualizations of features:

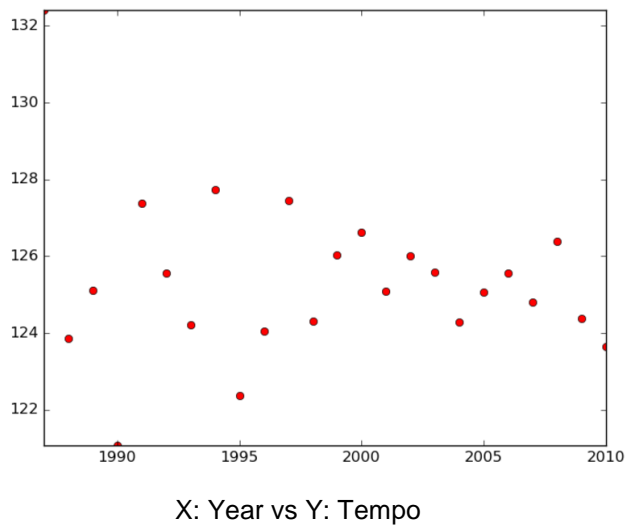
a) Year vs song hotness



b) Year vs Timbre[1]



### c) Year vs Tempo



## Results of ML models

RMSE before linear regression  
6.02561870749

Coefficients: 1.17101336026,-2.6942118484,3.68019076694,-0.00612548360007,-0.00643381855742,-  
0.694063408879,0.0140043179813,0.68739613501,0.492695999676,-0.0487422740476,-  
0.0297095842043,0.0646018604929,0.00989018175108,-0.0667447007473,-  
0.017989731082,0.000911212558484,-0.0496513253882,0.0366425852941,-  
0.236905590108,0.0586397020195,-0.000692933108853]  
Intercept: 1979.00194878

RMSE after linear regression  
5.29737161654

Random Forest with binary Classification

- > Normalized the features
- > 1990-2000 label 0
- > 2001-2010 label 1
- > Random sample with equal number of points

Random Forest

numClasses=2, numTrees=7, featureSubsetStrategy="auto",impurity='gini', maxDepth=4  
Accuracy 60%

Some failed attempts:

- a) Classification of years instead of decade.  
Random Forest  
Label each year from 1986 to 2010 for 21 features  
Accuracy 10 %
- b) Logistic regression. Binary classification: Expansion of features  
Label: Songs hotness  
 $x, x^2, x^3$  for artist hotness and artist familiarity  
No significant improvements. Still Accuracy close to 70%.

## CONFIGURATION, SET\_UP AND DEPLOYMENT USING GOOGLE CLOUD

Google cloud platform was used to store and perform data analysis.

### 1) Google Cloud Storage:

- Google cloud provides a simple UI or command line interface to upload our data.
- Google cloud stores data as a bucket
- Each bucket contains a storage class in which data is stored.
- The data stored is a single storage object and it is not possible to create directories.
- The million songs data is present as tar file and can be accessed in two ways.
- The data can be downloaded from an amazon snapshot.
- Unfortunately to transfer the data to other cloud technologies the data needs to be moved to an amazon S3 bucket and then to google cloud platform or microsoft azure if we wish to take the data from an amazon snapshot
- An alternative is to download the data is to use Open Science DataCloud. They provide a simple tutorial to download the data for free. The link is provided in references.
- The data was downloaded to a local system and then the tar files were

extracted and the extracted data was uploaded to google cloud storage. This was done to save cost as total data used would be over 500GB instead of 270GB and the tar files would not be used again.

- Below is a screenshot of the stored data.

Browser <a href="#">↑ UPLOAD FILES</a> <a href="#">↑ UPLOAD FOLDER</a> <a href="#">CREATE FOLDER</a> <a href="#">REFRESH</a> <a href="#">SHARE PUBLICLY</a> <a href="#">DELETE</a>						
Buckets / millionsongs / A / A / A						Filter by prefix...
<input type="checkbox"/> Name	Size	Type	Storage class	Last modified	Share publicly	
<input type="checkbox"/> TRAAAK128F9318786.h5	211.84 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAAV128F421A322.h5	168.97 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAAW128F429D538.h5	277.78 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAAY128F42A73F0.h5	194.82 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAAD128F429CF47.h5	200.17 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAACN128F9355673.h5	345.43 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAACV128F423E09E.h5	203.69 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAADJ128F4287B47.h5	383.8 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAADT12903CCC339.h5	182.21 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮
<input type="checkbox"/> TRAAADZ128F9348C2E.h5	200.32 KB	application/octet-stream	Regional	11/22/16, 5:08 PM	<input type="checkbox"/>	⋮

## 2) Google Cloud Dataproc:

- Dataproc is an easy to use tool to setup clusters in google cloud.
- Google cloud Dataproc provides an easy interface to setup a cluster, submit jobs and access the various master and worker nodes using ssh.
- The google cloud Dataproc allows us to add bash scripts during the initialization of the cluster. A customized script was used to setup the cluster.
- Using this script we were able to setup a jupyter notebook on our master node which used a python 3 kernel along with pyspark.
- Below is a screenshot of setting up clusters:

Name ?

cluster-1

Zone ?

us-east1-b

### Master node

Contains the YARN Resource Manager, HDFS NameNode, and all job drivers

Machine type ?

n1-standard-1 (1 vCPU, 3.75 GB ...

Cluster mode ?

Standard (1 master)

Primary disk size (minimum 10 GB) ?

10 GB

### Worker nodes

Each contains a YARN NodeManager and a HDFS DataNode.  
The HDFS replication factor is 2.

Machine type ?

n1-standard-1 (1 vCPU, 3.75 GB ...

Nodes (minimum 2) ?

2

Primary disk size (minimum 10 GB) ?

10 GB

Local SSDs (0-8) ?

0 x 375 GB

YARN cores ?

2

YARN memory ?

6.00 GB

### Preemptible worker nodes ?

Each contains a YARN NodeManager. HDFS does not run on preemptible nodes.  
Machine type is copied from the Worker section.

Nodes ?

0

Cloud Storage staging bucket (Optional) ?

gs://millionsongsoutput

Network ?

default

Image version ?

Initialization actions ?

gs://millionsongsoutput/jupyter.sh

gs://mybucket/action-xyz

Project access ?

☐ Allow API access to all Google Cloud services in the same project. [Learn more](#)

Labels (Optional) ?

+ Add item

⏏ Less


Create

Cancel

Equivalent [REST](#) or [command line](#)

### 3) Jupyter Notebook:

- Jupyter Notebook is a web application which provides an interactive web application to run python scripts.
- It has rich media representations in HTML,PDF and PNG
- It allows us to run a part of python script and provides the visualization results of the part of our script.
- Google provides documentation to setup jupyter notebook on google cloud Dataproc
- Below is a screenshot of jupyter notebook.

 **jupyter** pyspark\_notebook (3) Last Checkpoint: 15 hours ago (autosaved)

File Edit View Insert Cell Kernel Help

PySpark 3 

          Code   CellToolbar

```
In [1]: #import findspark
        #findspark.init()
```

```
In [2]: #import pyspark
```

```
In [3]: #sc=pyspark.SparkContext(appName='10k_explore')
```

```
In [4]: sc
```

```
Out[4]: <pyspark.context.SparkContext at 0x7f94401f1940>
```

```
In [5]: import pandas as pd
        import numpy as np
        from pyspark.sql import Row
        from pyspark.sql import DataFrame
        from pyspark.sql import SQLContext
        from pyspark.sql import SparkSession
        from pyspark.sql.types import *
```

```
In [6]: spark = SparkSession \
        .builder \
        .appName("10k_explore") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```



#### 4) Cost:

- The overall cost of storing the 270 GB of data on google cloud is around 5 dollars a month.
- To run create a cluster of lowest configuration(master node with 3,75 GB ram and similar work node) in dataproc it costs less one dollar an hour.
- The total cost was 15 dollars for storing the data, running our clusters.

## CONCLUSION

In this project we used the million songs dataset to predict the songs hotness. We used logistic regression and random forests to build our prediction model. Elastic net was used to extract the important features. Google Cloud platform was used to setup a spark cluster. The spark cluster provided an accuracy of around 70% for song hotness.

## REFERENCES

1. MSD official website : [labrosa.ee.columbia.edu/millionsong/](http://labrosa.ee.columbia.edu/millionsong/)
2. Paper by creator of MSD: [http://tbertinmahieux.com/Papers/tbm\\_thesis.pdf](http://tbertinmahieux.com/Papers/tbm_thesis.pdf)
3. BB unofficial API : <https://github.com/guoguo12/billboard-charts>
4. Open Science data cloud: <https://github.com/guoguo12/billboard-charts>
5. Amazon google cloud link: <https://aws.amazon.com/datasets/million-song-dataset/>
6. Dataproc with jupyter: [https://blog.sourced.tech/post/dataproc\\_jupyter/](https://blog.sourced.tech/post/dataproc_jupyter/)