End-to-End Distributed ML with Million Songs Dataset

Team-7

Team members

Dhanyalaxmi Panickar

AMENP2ARI20012

Lekshmy H O

AMENP2ARI20020

Jakesh P Kuriakose

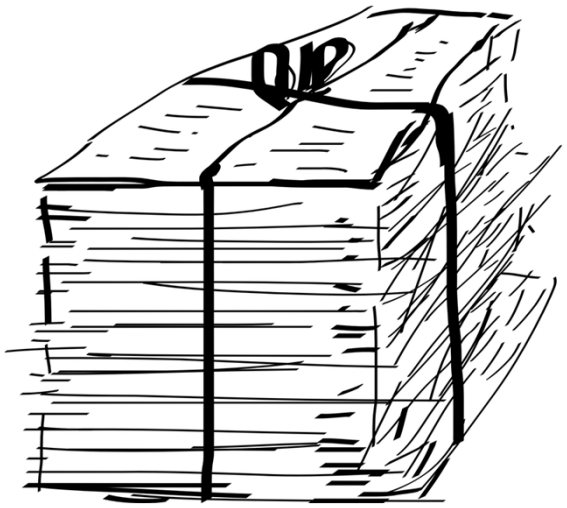
AMENP2ARI20013

AMRITA VISHWA VIDYAPEETHAM

AGENDA

- ✓ About the paper
- ✓ Problem formulation
- ✓ Motivation
- ✓ Solution overview
- ✓ Million song dataset
- ✓ Components
- ✓ MongoDB
- ✓ Methodology
- ✓ Challenges faced
- ✓ Take away

About the Paper

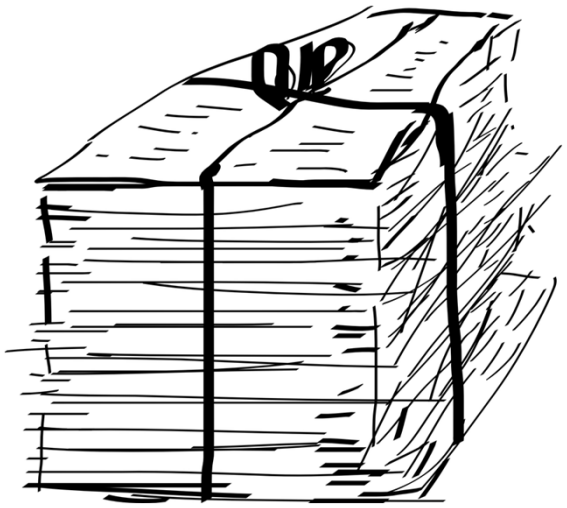


MILLION SONG DATASET

Thierry Bertin-Mahieux, Daniel P.W. Ellis
Columbia University
LabROSA, EE Dept

Brian Whitman, Paul Lamere
The Echo Nest
Somerville, MA, USA

Additional References



Song Year Prediction Using Apache Spark

2016 Intl. Conference on Advances in Computing, Communications and Informatics (ICACCI)

Prakhar Mishra, Ratika Garg, Akshat Kumar,

Arpan Gupta and Praveen Kumar

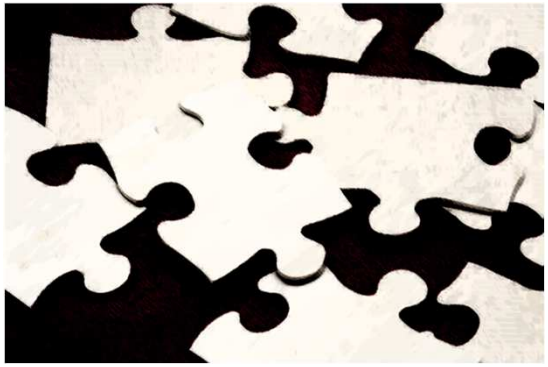
The Million Song Dataset Challenge

2012 Intl. International World Wide Web Conference Committee (IW3C2)

Brian McFee, Thierry Bertin-Mahieux,

Daniel P.W. Ellis, Gert R.G. Lanckriet

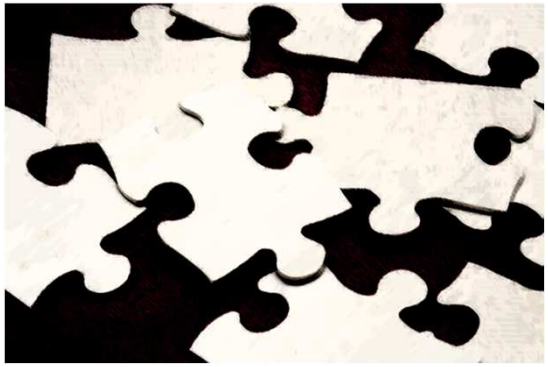
Motivation



Certain users often have particular affection towards the songs of a particular period, which we can easily predicted using this big data.

Music services can derive financial gain by improving their recommendations to potential customers of the songs they like.

Problem Formulation



Predicting the year of a songs release given a set of audio features such as the tempo, duration, etc.

From the available 1M songs we would like to recommend the most appropriate songs for a user.

Solution Overview

Implemented machine learning algorithms for the customized recommendation and prediction problems using Apache Spark in Databricks platform.

Improved data accessibility by incorporating MongoDB Atlas as a cloud data storage.

Million Song Dataset

- 280 GB of data
- 1,000,000 songs/files
- 44,745 unique artists
- 7,643 unique terms (Echo Nest tags)
- 2,321 unique music brainz tags
- 43,943 artists with at least one term
- 2,201,916 asymmetric similarity relationships
- 515,576 dated tracks starting from 1922

Million Song Dataset

Fields : 55 Fields

1) analysis sample rate
2) artist 7digitalid
3) artist familiarity
4) artist hotttnesss
5) artist id
6) artist latitude
7) artist location
8) artist longitude
9) artist mbid
10) artist mbtags
11) artist mbtags count
12) artist name
13) artist playmeid
14) artist terms
15) artist terms freq
16) artist terms weight
17) audio md5
18) bars confidence
19) bars start
20) beats confidence
21) beats start
22) danceability
23) duration
24) end of fade in
25) energy
26) key
27) key confidence
28) loudness
29) mode
30) mode confidence
31) num songs
32) release
33) release 7digitalid
34) sections confidence
35) sections start
36) segments confidence
37) segments loudness max
38) segments loudness max time
39) segments loudness start
40) segments pitches
41) segments start
42) segments timbre
43) similar artists
44) song hotttnesss
45) song id
46) start of fade out
47) tatums confidence
48) tatums start
49) tempo
50) time signature
51) time signature confidence
52) title
53) track 7digitalid
54) track id
55) year

Million Song Dataset

The original data is stored using HDF5 format to efficiently handle the heterogeneous types of information such as audio features in variable array lengths, names as strings, longitude/ latitude, similar artists, etc.

Millionsongsubset › A › A › A

Name	Date modified	Type	Size
TRAAAAW128F429D538.h5	26-04-2021 07:14 PM	H5 File	278 KB
TRAAABD128F429CF47.h5	26-04-2021 07:42 PM	H5 File	201 KB
TRAAADZ128F9348C2E.h5	26-04-2021 07:42 PM	H5 File	201 KB
TRAAAEF128F4273421.h5	26-04-2021 07:42 PM	H5 File	259 KB
TRAAAFD128F92F423A.h5	26-04-2021 07:42 PM	H5 File	226 KB
TRAAAMO128F1481E7F.h5	26-04-2021 07:42 PM	H5 File	265 KB
TRAAAMQ128F1460CD3.h5	26-04-2021 07:42 PM	H5 File	181 KB
TRAAAPK128E0786D96.h5	26-04-2021 07:42 PM	H5 File	204 KB
TRAAARJ128F9320760.h5	26-04-2021 07:42 PM	H5 File	278 KB
TRAAAVG12903CFA543.h5	26-04-2021 07:16 PM	H5 File	362 KB
TRAAAVO128F93133D4.h5	26-04-2021 07:42 PM	H5 File	239 KB

Data conversion and processing:

These **HDF5 files** are converted in to **csv** and **txt** file using hdf5_getters.py and h5py modules.

Million Song Dataset Applications

Artist recognition: Recognizing the artist from the audio is a straightforward task that provides a nice showcase of both audio features and machine learning.

Automatic music tagging: The correlation between tags and metadata could be of great interest in a commercial system. There are also correlations between artist names and genres; you can probably guess the kind of music the band is making.

Recommendation: Music recommendation and music similarity.

Cover song recognition: One motivation behind this task is the belief that finding covers relies on understanding something deeper about the structure of a piece.

Mood prediction: Mood prediction from lyrics could be investigated with this data.

Components

- Apache Spark
- Databricks
- MongoDB Atlas
- ML-Library

MongoDB

- MongoDB is an open-source document database.
- Data objects are stored as separate document inside a collection.
- Each MongoDB instance can have multiple databases and each database can have multiple collections.
- Provides high performance.
- https://www.analyticsvidhya.com/blog/2021/04/how-to-connect-databricks-and-mongodb-atlas-using-python-api/

MongoDB Atlas

- MongoDB Atlas, a fully managed cloud database for modern applications.
- MongoDB belongs to "Databases" category of the tech stack, while MongoDB Atlas can be primarily classified under "MongoDB Hosting".
- No infrastructure on client side. Free of management.
- Easy to scale up and down.
- It has strong authentication and encryption features.

MongoDB Atlas With Spark

- ✓ Integration of Apache Spark with MongoDB extends analytics capabilities even further to perform real-time analytics and machine learning.
- ✓ With Spark and MongoDB, developers can build more functional applications faster using a single database technology.
- ✓ Integration of these two Big Data technology saves operations teams the hassle of shuttling data between separate operational and analytics infrastructure.
- ✓ Together MongoDB and Apache Spark are enabling success by turning analytics into real-time action.

Methodology

P

Prediction

- ❖ Data storage – MongoDB
- ❖ Connected MongoDB with Databricks

```
from pyspark.sql import SparkSession
database = "MSD"
collection = "MSD_C"
connectionString= 'mongodb+srv://admin:admin@dhanyatest.acrh4.mongodb.net/MSD?retryWrites=true&w=majority'
spark = SparkSession\
.builder\
    .config('spark.mongodb.input.uri',connectionString)\
    .config('spark.mongodb.input.uri', connectionString)\
    .config('spark.jars.packages', 'org.mongodb.spark:mongo-spark-connector_2.12:3.0.1')\
.getOrCreate()
```

```
Command took 0.04 seconds -- by amenp2ari20012@am.students.amrita.edu at 4/27/2021, 7:43:26 PM on TestMongo (clone) (clone)
```

Methodology

P

Prediction

- ❖ Reading data from MongoDB
- ❖ Save it as a dataframe in spark

```
# Reading from DB
df = spark.read\
    .format("com.mongodb.spark.sql.DefaultSource")\
    .option("uri", connectionString)\
    .option("database", database)\
    .option("collection", collection)\
    .option("inferSchema" , "true")\
    .load()
```

▸ (1) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [Column1: double, Column10: double ... 90 more fields]

Methodology

P

Prediction

```
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline

# Create a linear regression model
lr = LinearRegression(featuresCol="features", labelCol="Column1")
lrModel = lr.fit(vecTrainDF)
pipeline = Pipeline(stages=[vecAssembler, lr])
pipelineModel = pipeline.fit(trainDF)
```

▸ (4) Spark Jobs

```
1  trainingSummary2.rootMeanSquaredError
```

```
Out[88]: 9.69928313874379
```

- ❖ Used ML – Lib for Linear Regression
- ❖ Applied concept of pipeline and transformers

Methodology

R

Recommendation

1. Song recommendation using similarity and popularity
2. Song recommendation using cosine similarity
    1. Find top 5 similar users using cosine similarity
    2. Recommend unique songs

Data stored in Databricks directly

Methodology

R1

1.Song recommendation using similarity and popularity

1. user id, song id and rating are given as input.
2. Rating Triplet RDD: Contains details of user id, song id, play count and rating
3. Define a function, which checks for users who have liked the same given song id and given a rating higher than the input rating.
4. Users RDD: RDD of similar users who have liked the same song as input with higher rating
5. Join the users RDD with the Rating Triplet RDD to obtain the list of songs, listened to by these users. Use the filter function to filter out the input song, so that it does not show up in list of recommended other songs.

Methodology

R1

Song recommendation using similarity and popularity

6. To obtain top recommended songs, aggregate the total rating across songs
7. Run a lambda function (TakeOrdered( )) to obtain top 5 recommended songs to the user

```
rating_triplet.map(lambda x:(x[0],[x[1],x[2],x[3]]))\
              .join(users).filter(lambda x:x[1][0][0]!=song)\
              .map(lambda x:(x[1][0][0],x[1][0][2]))\
              .reduceByKey(lambda x,y:round(x+y,2))\
              .takeOrdered(5,key=lambda x:-x[1])
```

▸ (1) Spark Jobs

```
Out[33]: [('SOWSPUS12AC468BEE3', 0.81),
 ('SOLGPOU12A58A7EA20', 0.76),
 ('SOFAONV12A67020E43', 0.55),
 ('SONDDMN12A8C13B3E8', 0.53),
 ('SOSAFRM12AF72A9768', 0.53)]
```

Methodology

R2

Song recommendation using cosine similarity

1. To find cosine similarity we need the dot product of ratings for all songs listened to by a userset.
2. Cosine Similarity = DotProduct(A, B) / Norm(A) * Norm(B)
3. Top_users RDD - With a given user ID filterout all user sets which have this input userID and apply Cosine similarity to obtain top 5 similar users in the user sets.
4. Join the Top_users RDD with the original rating_triplet RDD to obtain the songs listened to by these similar users.
5. Use a distinct function to find unique songs and generate song recommendations for the given user.

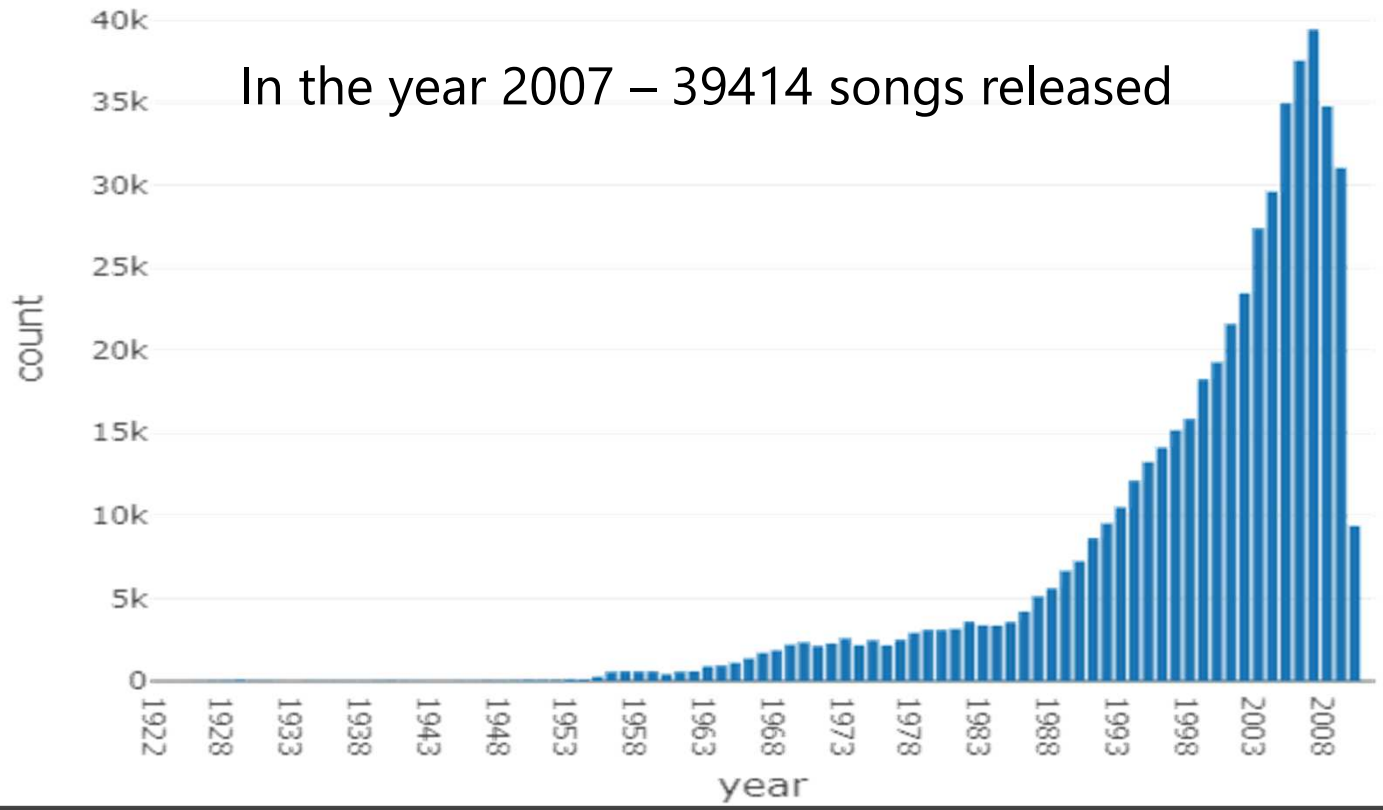
```
triplet_temp = rating_triplet.map(lambda x: (x[0], (x[1], x[2], x[3]))
user_temp = triplet_temp.join(top_users)
user_temp.map(lambda x: x[1][0][0]).distinct().collect()
```

▸ (1) Spark Jobs

```
Out[60]: ['SONTRSE12A8C13727C',
 'SOETFVO12AB018DFF3',
 'SOSELMV12A6D4FCF5A',
 'SOYAICE12AB01806E2',
 'SOFWKCI12A8C13A22A',
 'SOFOKPG12A58A7E768',
 'SOLODPO12AB017F217',
 'SOSDYJH12AF72A179D',
 'SONQCXC12A6D4F6A37']
```

Exploratory Analysis

Songs per year



Exploratory Analysis

Genre trend in each year

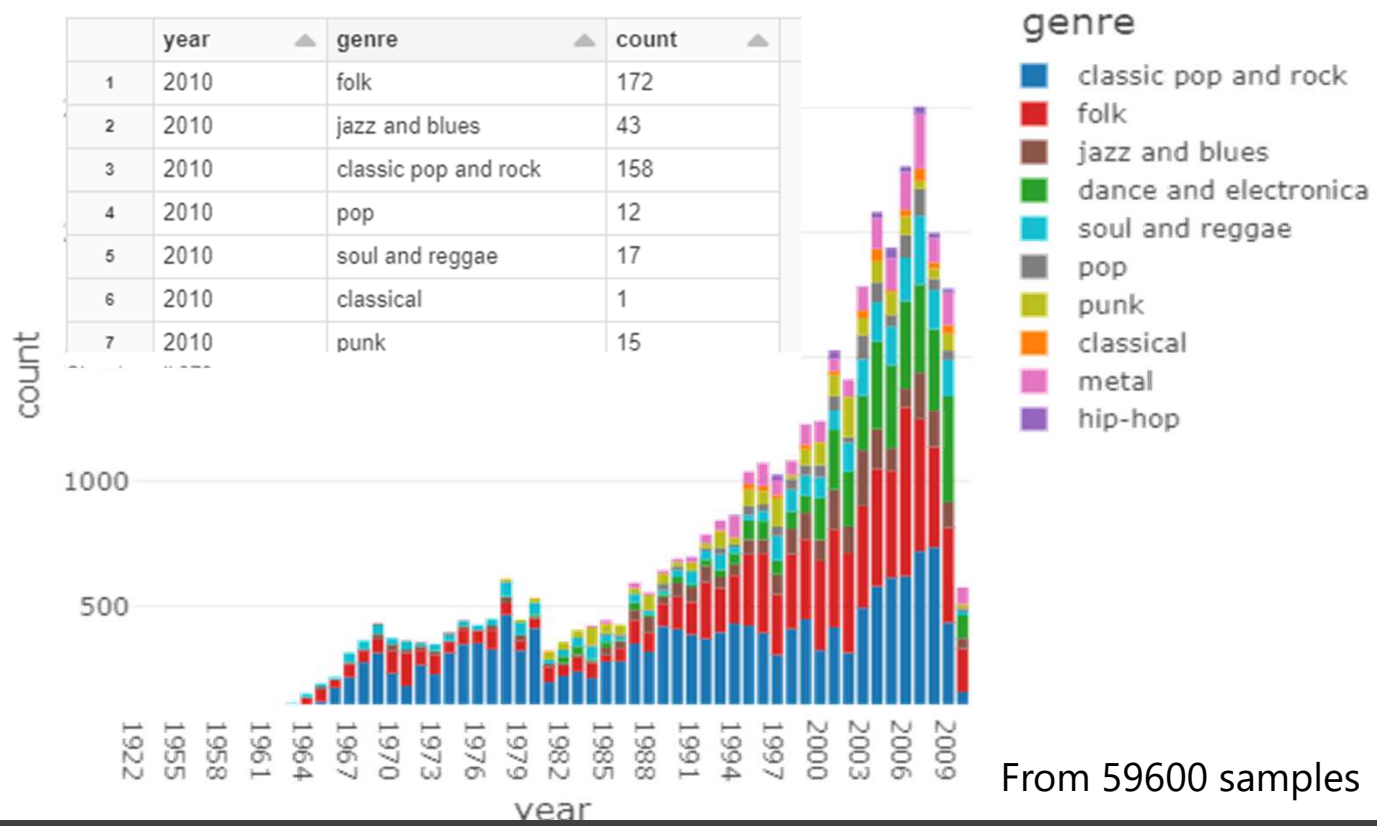
Most of the songs released in a year are of genre type

- Classic Pop and Rock
- Folk

The least released are

- Hip Hop
- Classical

	year	genre	count
1	2010	folk	172
2	2010	jazz and blues	43
3	2010	classic pop and rock	158
4	2010	pop	12
5	2010	soul and reggae	17
6	2010	classical	1
7	2010	punk	15



From 59600 samples

Exploratory Analysis

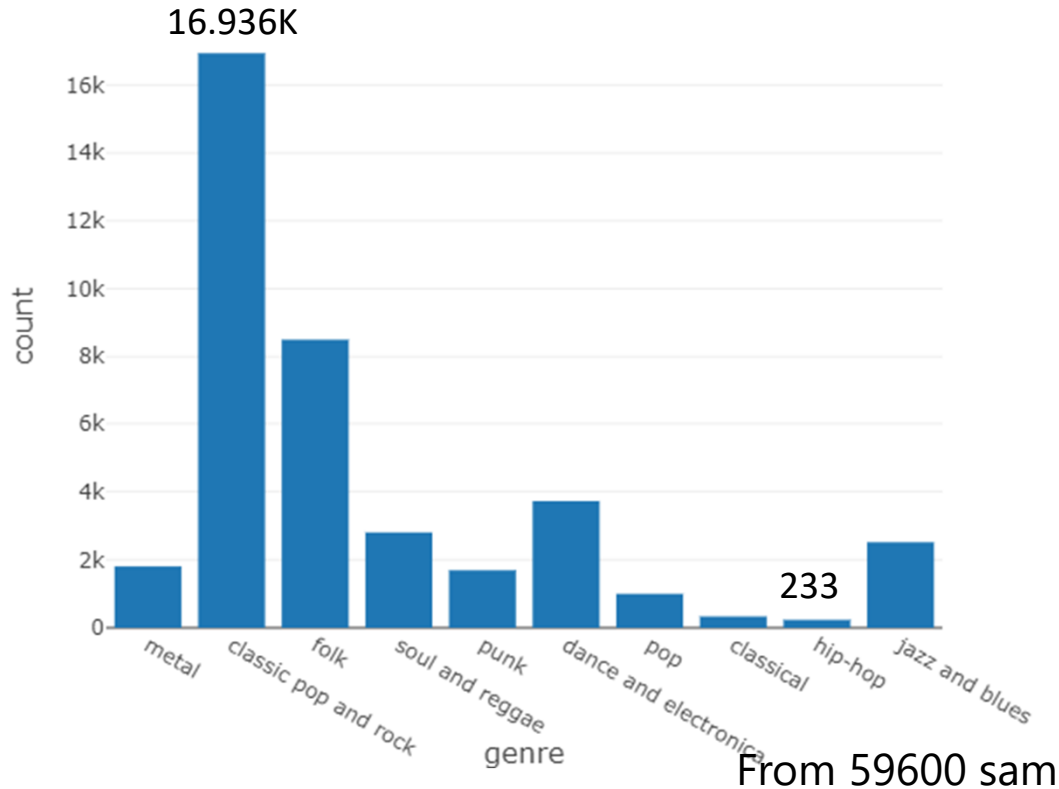
Genre total count so far

Most of the songs released so far are of genre type

- Classic Pop and Rock

The least released are

- Hip Hop



From 59600 samples

Challenges Faced

Lack of documentations on how to connect MongoDB Atlas with python API.

We are forced to use subset of Millionsong dataset because of its large size.

The limitations on services provided by community edition platforms.

Take Away

Million songs analysis opens the possibility of a customized song recommendation and prediction based on audio features with a clean ground truth.

Familiarized big data analysis from HDF5 files.

Familiarized with different Spark libraries.

Studied various about databases especially MongoDB and its remote access.

Thankyou…