UNIVERSITY OF TRENTO

The Million Song Dataset (MSD): ETL, data exploration and Gaussian Mixture clustering in *Databricks Community*

Big Data Technologies, A.A 2019-2020

Giacomo Lazzerini, Marco Bronzini
(Group 11)

Abstract

The MSDS is a database of popular songs spanning decades of music. It has been released to the public for research by The Echo Nest, a company that specializes in music intelligence services. The dataset has become more and more famed after its analysis competition released in Kaggle.com in 2012. The dataset is about 300 GB large and it contains a collection of audio features (loudness, tempo, etc.) and metadata (artist, year, etc.) for a million contemporary popular music tracks.

In this project, we suggest a simple Big Data system to process and analyse the Million Song Dataset (MSD) using Spark through the handy Databricks platform in its community Edition which is available for free. After displaying some statistics regarding the dataset, we have focused on creating different types of clusters based on different song attributes.

1 Choosing the platform

The initial challenge was to decide which tools would be more suitable for our purpose. We started off from the fact that we had to handle a static database and its processing would be through batch operations. In the latter aspect, we have seen two main tools across our course: Apache Hadoop and Apache Spark. We have chosen Spark because it seemed more suitable to our aims for data exploration and clustering. Since Apache Sparks provides us with some advantages concerning interactive queries (data exploration) and interactive data processing (e.g. machine learning models).
After choosing Spark, we had to decide how and where to run this software. Since the Million Song Dataset is hosted on AWS Public Dataset, we spent a while trying exploring this platform. Nevertheless, we have faced some limitations within the free tier (for more details, look at the chapter below). Discarded this option, we have decided to pick the handy platform of Databricks in its "Community Edition" which has been shown to us during the course. Thanks to this platform, we have been able to access a virtual machine with Spark already installed. We have chosen it also for its versatility; the interaction and co-operation are allowed through notebooks which can be used with different programming languages such as Python, Scala, R and SQL. Despite this plenty of choices, we managed the whole project through the Python language and some libraries such as PySpark.

2 ETL: Problems and Solutions:

The full dataset was freely provided by The Echo Nest and LabROSA, to encourage researchers to get started in the MIR (Music Information Retrieval). At first, the dataset could be accessed from an S3 bucket in AWS. Unfortunately, now is no longer the case, given that from 2016 the dataset can be accessed only via Amazon Web Services Public Dataset snapshot and the processing can be done only attaching to an Amazon EC2 virtual machine. Although there is a free tier for EC2 processors, this data snapshot must be contained inside a data partition (EBS) of at least 300 GB and it is not included in the free plan. Amazon charges for EBS usage; for instance, a 500G partition costs approximately \$10/week for the time it is in existence.
Despite this, we found two alternative ways to explore and analyse the dataset: a subset provided by the official website of the Million Song Dataset and a partial, pre-processed version hosted on the Databricks platform.

2.1 The official subset

A free downloadable subset of the original MSD is available for download in the official site: http://millionsongdataset.com/pages/getting-dataset. The subset contains 10.000 songs and is about 2.7 GB large. This subset is provided by LabROSA to do simple data exploration and to prepare scalable computations on the original dataset. As the original dataset, both the subset and the full dataset are in the form of compressed HDF5 files. Each file represents one track with all the related information organised hierarchically. There are three main directories (analysis, metadata and musicbrainz) in which there are some tables, numeric and character arrays and some matrices.

The first challenge was to try to read an HDF5 file, even though it has turned out one of the most difficult aspects of our project since HDF5 is a complex format and is not natively readable by Spark. HDF5 stands for "Hierarchical Data Format" and it is an open-source file format that supports large, complex and heterogeneous data. To overcome this issue, we used the h5py library in combination with Pandas to read raw data from a single HDF5 file and select only the columns in which we were interested in. Next, we used that function as "mapper" in the Spark - RDD for reading all files and consequently generating the whole Spark dataset.

```
rdd = sc.parallelize(files_list)
.flatMap(lambda path:read_hdf5(path))

df = spark.createDataFrame(rdd, schema)
```

After creating the whole dataset, we applied some transformations (adding, removing, etc.) to some columns according to our initial purposes. For instance, we computed the duration of song fade through subtracting three numeric columns as well as mapped the numeric column of the "key" feature into string values (0:'DO',..., 2:'RE").

One problem with the dataset, is that it's quite difficult yet not very useful to handle genre fields. The problem with genre tagging is that each song contains an array of genre tags provided by users but the range of attributes is very high and impossible to use it for classification methods and they refer to the artist and not to the song itself. One solution we adopted to better handle and reduce the range of genre tags was to left join "Tagtraum Genre Annotations" dataset provided by www.tagtraum.com, in which each track ID is paired with a more accurate genre tag.

```
f = spark.sql("SELECT SONGS.*, GENRE.SEED_GENRE AS GENRE FROM
SONGS LEFT JOIN genre ON SONGS.track_id=genre.track_id")
```

Lastly, we saved the whole dataset into the Databricks platform which exposes a version of Hive (Databricks Hive metastore) accessible via Spark.

```
df.write.format("parquet").saveAsTable("songs")
```

Thanks to using this Databricks functionality, the interaction with the dataset has been fairly straightforward.

2.2 The pre-processed version hosted on DataBricks

Databricks provides a "pre-processed" version of the full dataset that can be read from DBFS (DataBricks FileSystem) at dbfs:/databricks-datasets/songs/. Several features have been dropped from the original dataset in order to have a lighter version and use it for teaching purpose. In the folder, the dataset is splitted in more than 100 parts, and the first part is the list of headers. Each line of data consists of multiple fields separated by "/t". Retrieving information from header file, we set out a function to parse the data. To better query the large dataset, the table is then cached in memory. After this ETL process is still very handy to do some data exploration.

```
header = sc.textFile(
"/databricks-datasets/songs/data-001/header.txt")
.map(lambda line:  line.split(":")).collect()
```

2.3 Our choice

To sum up, we were able to work with all the MSD features but limited in the number of songs (1The foremost issue with the dataset on the Databricks platform was to have a limited number of song feature columns, which had been uploaded according to another user's choices. This can cut down valuable analysis, exploration or data manipulation that could be done only with the full-attributes dataset. Thereby, we have preferred to use a smaller number of songs (the official subset) but with all the features available rather than using the partial pre-processed version of Databricks. Although we actually had to work with 9.900 songs (100 fewer) due to the limitations of Databricks Community (up to 10.000 files).

ATTRIBUTE	TYPE	ATTRIBUTE	TYPE
analysis sample rate	float	**loudness**	float
artist 7digitalid	int	**mode**	int
artist familiarity	float	**mode confidence**	float
artist hotttnesss	float	**release**	string
artist id	string	**release 7digitalid**	int
artist latitude	float	**sections confidence**	array float
artist location	string	**sections start**	array float
artist longitude	float	**segments confidence**	array float
artist mbid	string	**segments loudness max**	array float
artist mbtags	array string	**segments loudness max time**	array float
artist mbtags count	array int	**segments loudness max start**	array float
artist name	string	**segments pitches**	2D array float
artist playmeid	int	**segments start**	array float
artist terms	array string	**segments timbre**	2D array float
artist terms freq	array float	**similar artists**	array string
artist terms weight	array float	**song hotttnesss**	float
audio md5	string	**song id**	string
bars confidence	array float	**start of fade out**	float
bars start	array float	**tatums confidence**	array float
beats confidence	array float	**tatums start**	array float
beats start	array float	**tempo**	float
danceability	float	**time signature**	int
duration	float	**time signature confidence**	float
end of fade in	float	**title**	string
energy	float	**track id**	string
key	int	**track 7digitalid**	int
key confidence	float	**year**	int

Table 1: Field Data Types

	MIN	MAX	MEAN	MEDIAN	STD	COUNT
analysis_sample_rate	22050.0	22050.0	22050.0	22050.0	0	1000
artist_7digitalid	-1	796501.0	111122.37	38010.0	144303.99	1000
artist_familiarity	0	0.95	0.57	0.57	0.16	1000
artist_hotttnesss	0	1	0.39	0.38	0.13	1000
artist_latitude	-37.82	64.95	36.99	38.55	15.89	383
artist_longitude	-155.43	144.97	-65.61	-81.54	48.02	383
artist_playmeid	-1	215105	25738.57	2578.5	45448.23	1000
danceability	0	0	0	0	0	1000
duration	9.66	1030.19	236.63	225.11	98.69	1000
end_of_fade_in	0	28.17	0.73	0.19	1.86	1000
energy	0	0	0	0	0	1000
key	0	11	5.27	5.0	3.5	1000
key_confidence	0	1	0.46	0.49	0.27	1000
loudness	-38.52	-1.48	-10.21	-9.1	5.39	1000
mode	0	1	0.71	1	0.45	1000
mode_confidence	0	1	0.48	0.5	0.19	1000
release_7digitalid	130.0	819440.0	382500.77	368893.0	231486.93	1000
song_hotttnesss	0	1	0.35	0.36	0.24	562
start_of_fade_out	9.66	1027.48	228.2	216.56	96.63	1000
tempo	0	241.88	124.05	120.99	35.91	1000
time_signature	1	7	3.48	4.0	1.24	1000
time_signature_confidence	0	1	0.51	0.54	0.37	1000
track_7digitalid	1647.0	9050196.0	4241405.38	4120811.5	2574875.31	1000
year	0	2010.0	972.87	0	999.04	1000

Table 2: Summary Table of all numeric fields for the a 1000 song sample

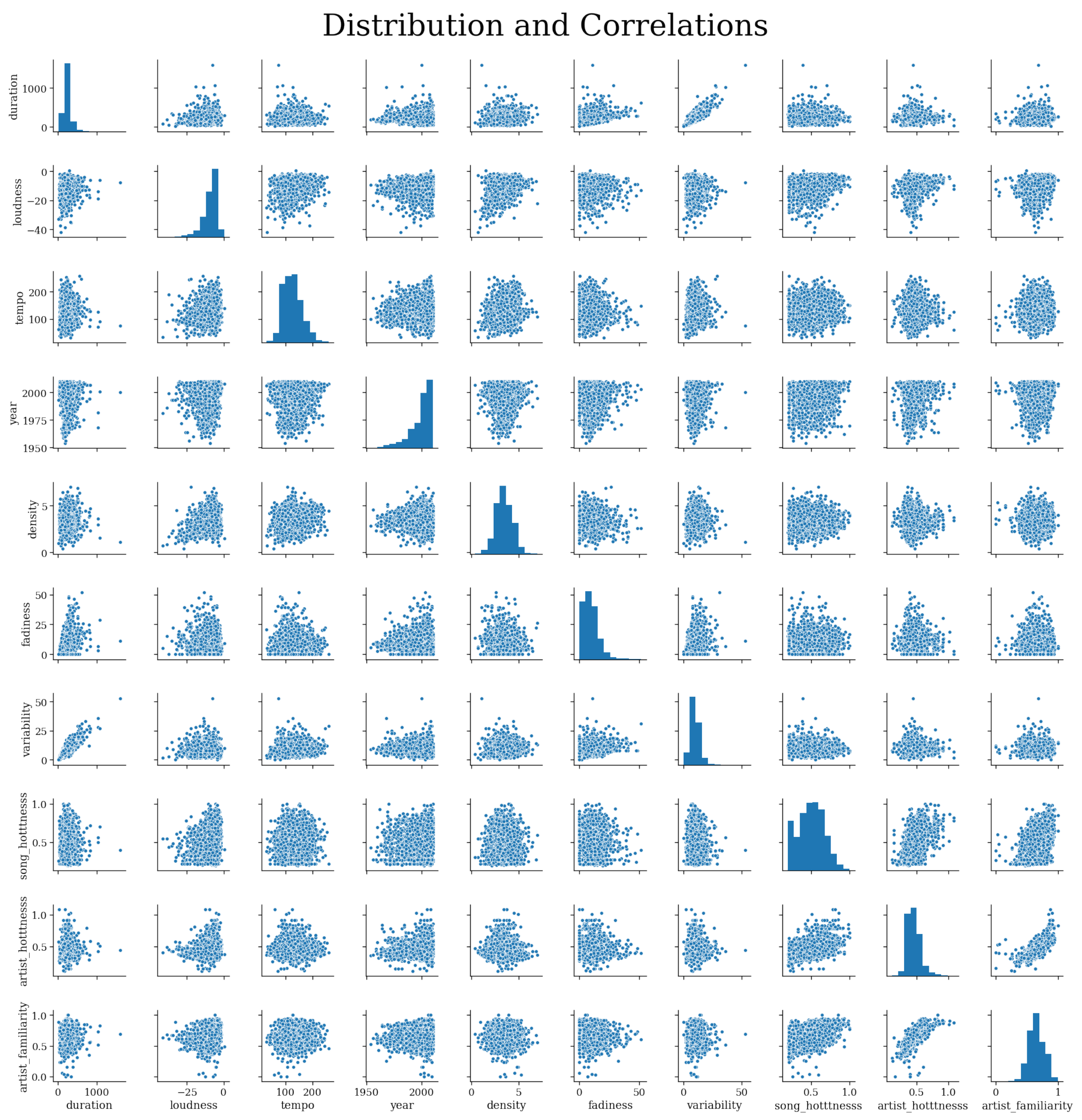

Figure 2: Scatter-matrix

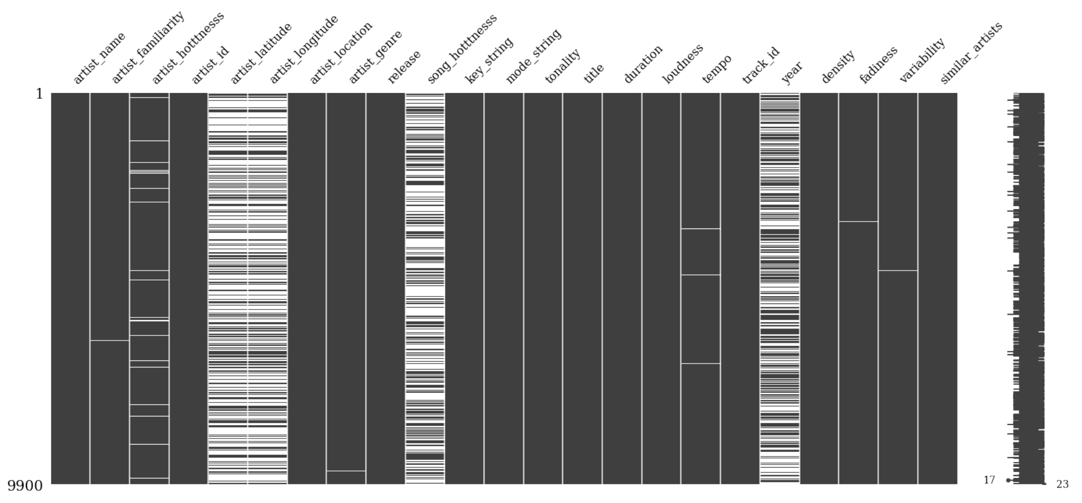

Figure 1: Missing Data

3 Data Exploration

Before data modeling is always a good idea to explore data for missing data, correlations and possible interactions between variables. Plotting the data is also very functional to summarize and get an accurate overview of different distributions.

3.1 Data Cleaning

Before importing the subset into databricks, we first look at all song fields contained in the hdf5 files. Hierarchical data formats are generally not human readable, but using software such as HDFView is useful to look for having a general idea of song features. Each song contains a total of 54 different fields. (*Table 1*)

Each field is the result of a massive audio-analysis and metadata for one million songs, provided by The Echo Nest. Data types are mainly numerical, strings or arrays. For our clustering model we needed numerical values, so we started the data exploration process by examining the dataset in aggregate form computing basic statistics (count, mean, standard deviation, min, max) for each of the integer/float valued fields. (*Table 2*)

The table is quite informative about the "integrity" of the dataset. The dataset seems not to be complete, given that there are several fields with missing values. In addition, some zero values are not valid for some specific fields (e.g. years, tempo, duration, danceability and energy). In order to avoid errors related to invalid values, we prepared the data by removing those ones to better fit models for clustering. (*Figure 1*)

3.2 Data Visualization

We plotted some graphs to get a sense of the distribution of different fields in our dataset. As can be seen from the scatterplot matrix (*Figure 2*), the fields that we are interested in, follow a gaussian distribution, which is important for distance metric-based models such as KNN algorithms. Considering that some correlation are quite obvious (e.g. song hotness and artist hotness) and do not add useful information about the interactions between different fields the correlation heatmap seems not too informative.

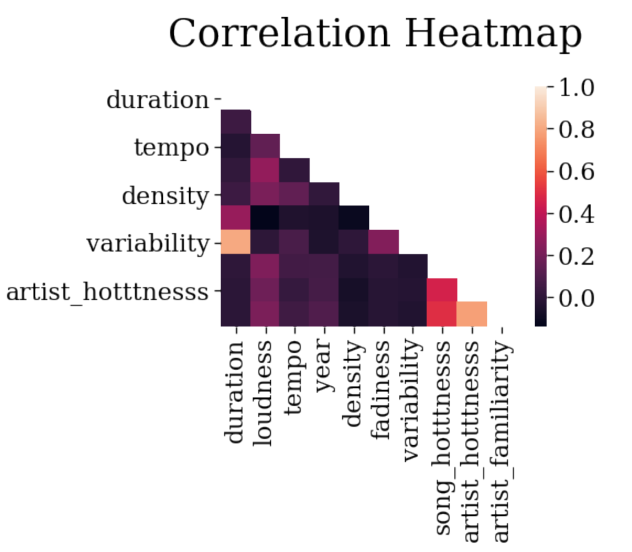

Figure 3: Correlation Heatmap

The only feature that does not follow a gaussian distribution is the year. This could be because of the advent of digital recording and musical market expansion, the dataset has more records from the late 2000s. Even if the years do not follow a normal distribution, it is quite interesting to look at how some parameters change over time.

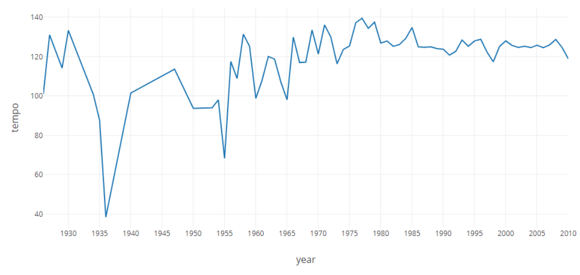

Figure 4: Tempo over years

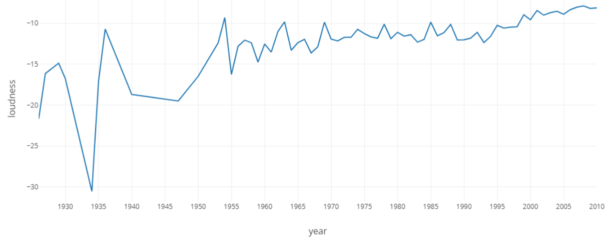

Figure 5: Loudness over years

One reason behind the loudness trend through years, could be depend on the shift from A.M. to F.M. for radio music broadcasts from late 50s.

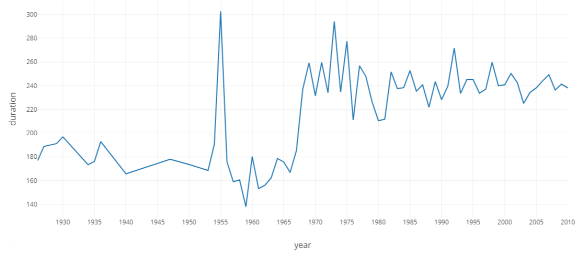

Figure 6: Duration over years

Looking at the duration, in the past, many songs were bound to a limited time in order to fit as much song as possibles in old record techniques; this could explain a growing trend.

With respect to the genre field, we obtain more accurate results by joining “tagtraum Genre Annotations”, but not sufficiently reduced for a classification analysis or logistic regression based on categorical features. To get a broader sense of how genre tags are distributed, we plot a word-cloud.

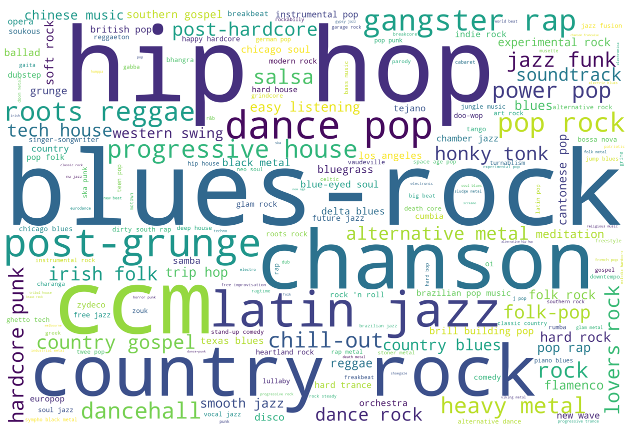

Figure 7: Genre Wordcloud

4 Clustering Modeling

In the last section of our project, we have focused on clustering songs by creating some models according to different types of clusters that we have come up with, according to the features available. In the figure below are displayed our cluster types and, for each, the cluster number. Those division numbers have been thought according to our empirical outcomes, we started off from four clusters for each cluster type and then changed to better fit with our sample data.

NAME	N. OF CLUSTERS
popularity_type	6
quality_of_song	4
type_of_running	4
quantity_of_fade	3
song_style	5
estimated_region	10

Table 3: Number of clusters

Spark, and thus its Python implementation (PySpark), provides a machine learning library (MLlib) in which there are also tools for clustering. We have used a “Gaussian Mixture” model for each cluster type. The latter is a soft-clustering model in which the cluster previsions consist of probabilities of being clustered in different classes. The model generates a column with all the probabilities and another in which it puts the most probable cluster for that song. Concerning the probability column, we decided to extract only the probability of the actual cluster selected and use it as a precision measurement of our clustering (confidence graph). By contrast, in case we would have picked a hard-clustering model (e.g. K-Means), we would have obtained a pure hard classification. Going deeper, a “Gaussian Mixture” model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing K-Means clustering to incorporate information about the covariance structure of the data as well as the centres of the latent Gaussians. The MLlib implementation expects input in a vector column. Fortunately, there are already utilities in the library that can help us convert existing columns in our table to a vector field and It is called VectorAssembler.

```
...
assembler_c1 =
VectorAssembler(inputCols=FEATURES[0],outputCol=’features_C1’)

assembler_c2 =
VectorAssembler(inputCols=FEATURES[1],outputCol=’features_C2’)
...
```

We used it for building the inputs for the clustering models, and then defined six different Gaussian mixture models and put all inside a Pipeline object.

```
...
gmm_c1 = GaussianMixture(k=NUM_CLUSTER[0] ,
featuresCol="features_C1",
predictionCol=TYPE_CLUSTER[0],probabilityCol=’probability_’+
TYPE_CLUSTER[0], seed=1)

gmm_c2 = GaussianMixture(k=NUM_CLUSTER[1] ,
featuresCol="features_C2",
predictionCol=TYPE_CLUSTER[1],probabilityCol=’probability_’+
TYPE_CLUSTER[1], seed=1)
...
```

Next, trained the models on our dataset and predicted the six different cluster types for each song.

```
...
model = pipeline.fit(songData)
song_labelled = model.transform(songData)
...
```

Then, after analysing the outcomes we decided the cluster names for each type. Lastly, we generated and saved an outcome table with the track_id, the six different cluster types and their confidences (below is displayed an example of a song after being clustered).

CLUSTERS	**VALUES**
track_ID	TRB...33F1
popularity_type	One-hit Wonder
quality_of_song	Popular
type_of_running	Cardio
quantity_of_fade	Short
song_style	Dub/Instrumental
estimated_region	South America
confidence_popularity_type	1386398.17.00
confidence_quality_of_song	165253.26.00
confidence_type_of_running	868312.55.00
confidence_quantity_of_fade	135810.12.00
confidence_song_style	109600.55.00
confidence_estimated_region	1658444.54.00

Table 4: Predicted clusters for a sampled song

To summarise some difficulties which we have encountered throughout this clustering phase: despite each stage has been challenging for us, from features selection to cluster visualization, the most difficult aspect has been to come up with possible cluster types and which features include in. As mentioned before, we have come up with six different cluster types nevertheless, after deeply analysing them, we would like to highlight some clusters which might not be so significant, even though we have preferred to not delete them. The first one is "quantity_of_fade" which although contains a value computed by us, it uses only one attribute and does not provide more insights compared to the simple numeric value (fade in seconds). The second is the "estimated_region" cluster which although is fairly precise on our sample data, its classes, the regions, have been identified through several empirical tests. Thus, we could suppose that its outcomes, and accordingly the cluster names, will be highly influenced according to the latitude and longitude values available.

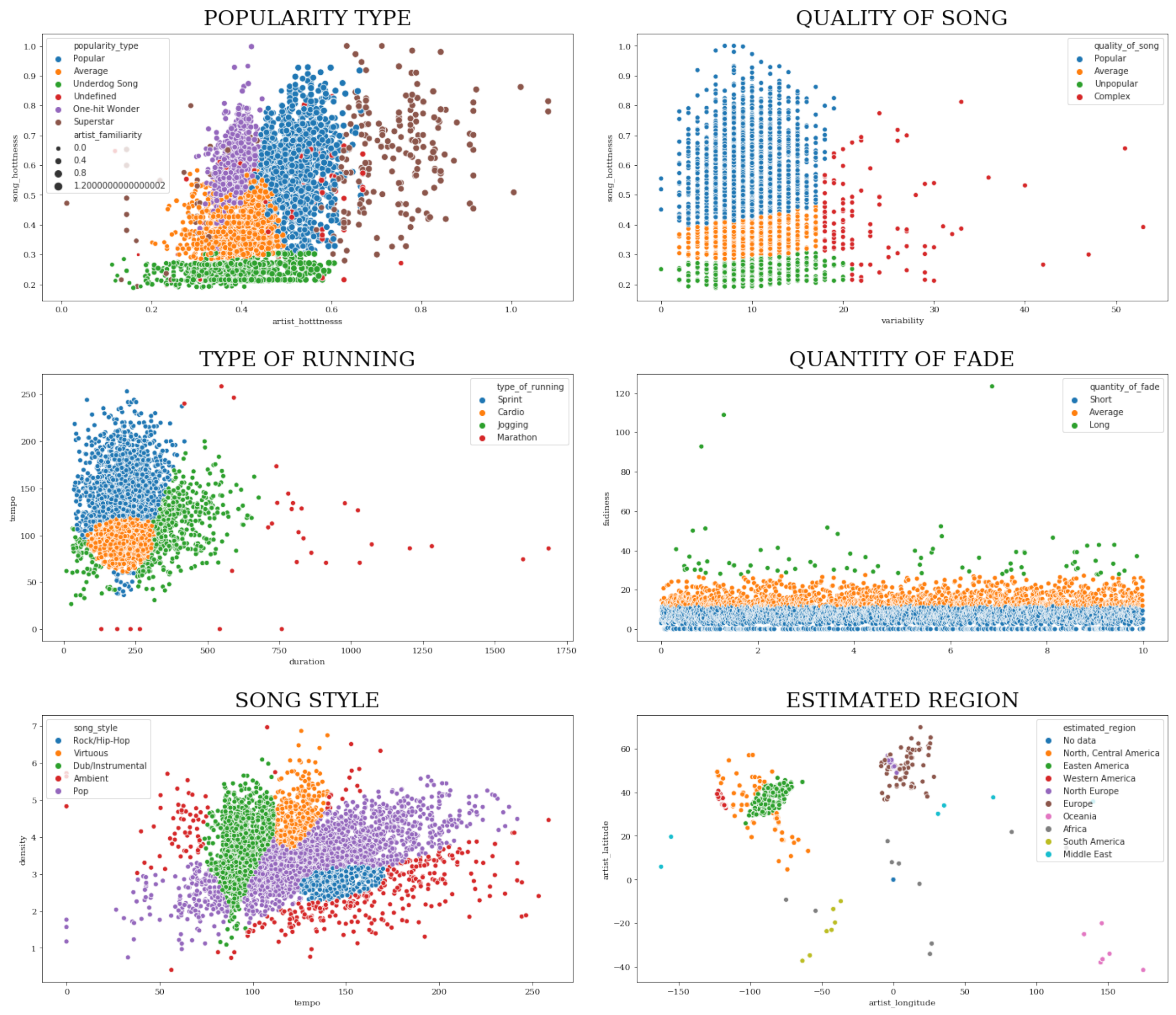

Figure 8: Scatterplots of the Gaussian Mixture

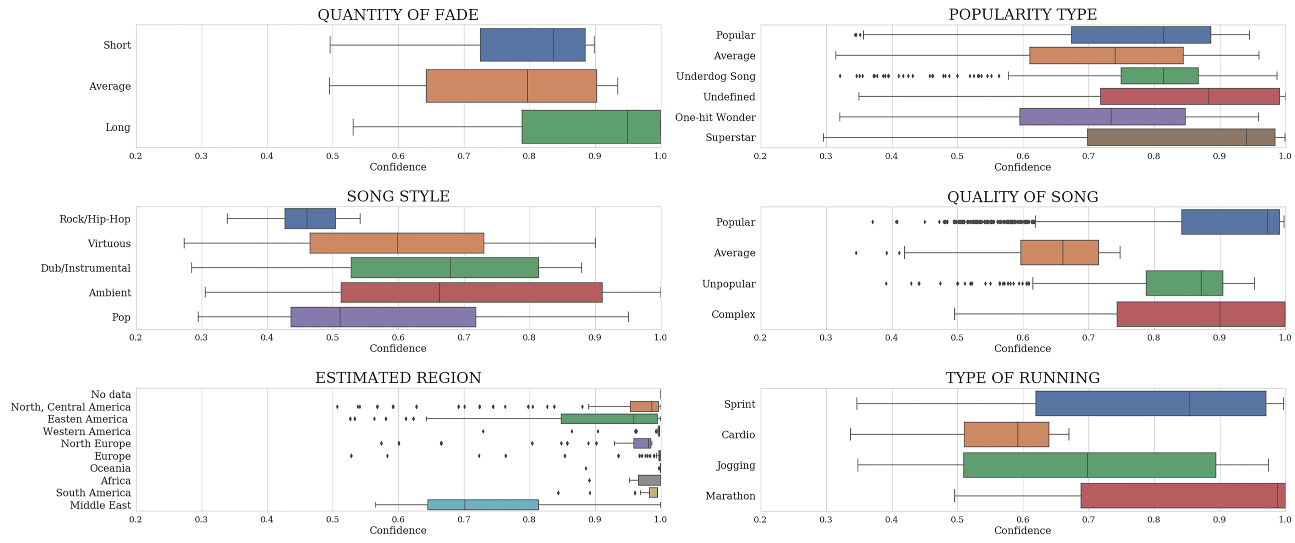

Figure 9: Confidence level boxplots

5 Conclusions

The main difficulty of this project relies on not having access to the full dataset. Even if some universities own a physical copy of the entire dataset, the only documented way to process it is via AWS EC2 through a public snapshot. Full data access could considerably improve data exploration and clustering. Storing the dataset into an S3 bucket and accessing it via Databricks could be the best solution to work with. In general, the project was very challenging, especially in the ETL process, and working with a very creative theme as a music-related database, let us enjoy the journey through this implementation. We suggested our personal solution to the project but there would be different implementations that should be considered for possible future works/upgrades.

5.1 Possible future work

One hypothetical scenario could also involve the use of Amazon Web Services (AWS) infrastructures. Regarding map-reduce, with a small investment, the results would be incredibly better. A general pipeline could be implemented by creating instances of an Elastic Compute Cloud (EC2), after setting up a key-pair, and then creating an Elastic Map Reduce (EMR) cluster. MRJob Python library could be used for writing and running map-reduce jobs in the clusters. These kinds of operations work well with simple Map Reduce jobs, however, clustering and machine learning algorithms would require additional AWS tools.

In this project, we used the Tagtraum Genre Annotations Dataset, which helped us to get a better (but not range-reduced) genre field. However, using a detailed parser we could reduce all genre items into small (max 8 genre fields) genre “macro-categories”. Since the genre attribute of a song is extremely important for classification this could bring a big improvement for further analyses. The Million Song Dataset official site provides additional datasets to work along with the original dataset. These datasets could better improve the data exploration by adding new information about audio features, lyrics and even user listening data. Some of these fields could not only expand data exploration and data classification, but also improve the clustering model.

Another possible method to process and cluster the song could have been implemented in Neo4j platform creating a graph database (also GraphX could be a good choice). The array field containing similar artists for each song could be a solid starting point to build a graph database based on transversal relationships. Several fields such as genre, tempo or duration could also be taken for complex graph representations of the dataset; this platform would certainly be optimal for graph-based clustering models and it could also be good for recommendation systems.