A Million Songs.

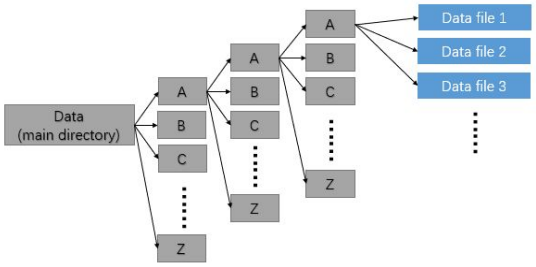
Ziyao Ding, Subha Karanam, Luwei Lei, Lu Sun

Introduction

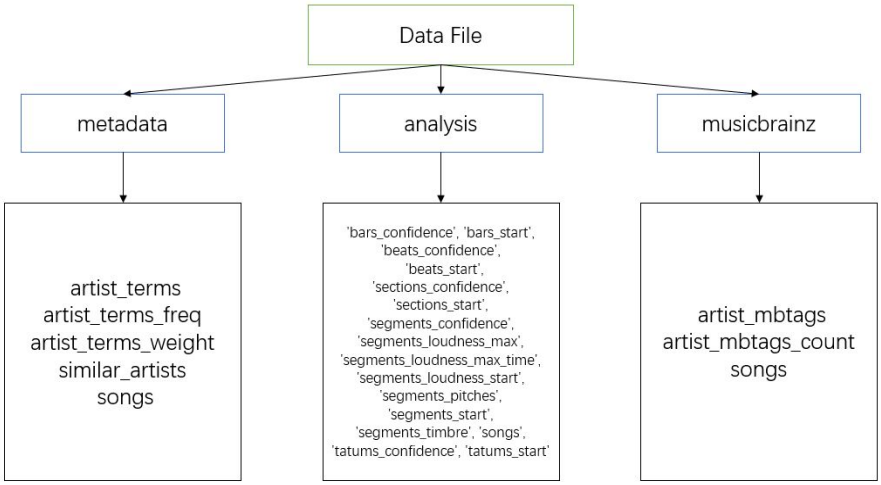
The Million Song Dataset is a collection of audio features and metadata for a million contemporary popular music tracks. This project focuses on extracting features from the large dataset and take a glance at the trending of the popular music by statistical analysis and several predictive models.

Dataset

This dataset, like the title suggests, contains a million songs, which is around 230GB. But for the purpose of this project, we have limited the amount of data that is processed to 45GB, which amounts to 170218 songs. The dataset is stored in hdf5 format. In this dataset, the data files are stored in three layers of directories that are sorted in alphabetical order.



Each data file in this dataset contains information for one song. Structure of the data in the files is shown below



Code files

Github repository Cleaned data CSV

Sourcing and Cleaning

|-bootstrapping.sh : special bootstrapping script for the cluster.
|-clean_artist_terms.ipynb : changes the formatting of artist_terms columns in the csv
|-cleaned-subset.csv : csv of a small subset(1.8GB) for initial experimentation
|-compile_subset.py : python script to compile the small subset(1.8GB) into csv.
|-convert_h5.ipynb : takes in all the hdf5 files from the data folder and compile it into CSV
|-data cleaning.ipynb : basic data cleaning for the small subset csv.
|-display_song.py : Taken from the creator of the dataset to learn how to read the hdf5 files
|-h5py_getter.py : getters written for all the different features in the hdf5 files, which can be used with Spark
|-hdf5_getters.py : getters written by the creator of the dataset to fetch all the features in hdf5 files.
|-make_smaller_subset.py : non-spark compatible code to process all the data.
|-utils.py : contains the glob code to fetch all the file paths ending with .hd5

Exploratory Analysis

|-genre_analysis.ipynb : uses the description data in artist_terms column and visualization of a wordcloud for the genres.
|-stat_analysis.ipynb : Basic statistical analysis of the data in the csv file.
|-year.html : visualization of the year-song counts.

Modelling

|-model_test.ipynb : script written as a part of the year-prediction model
|-model_year_recognition.ipynb : script written as a part of the year-prediction model
|-regression.ipynb : script written as a part of the model to predict the song_hotness

Methods

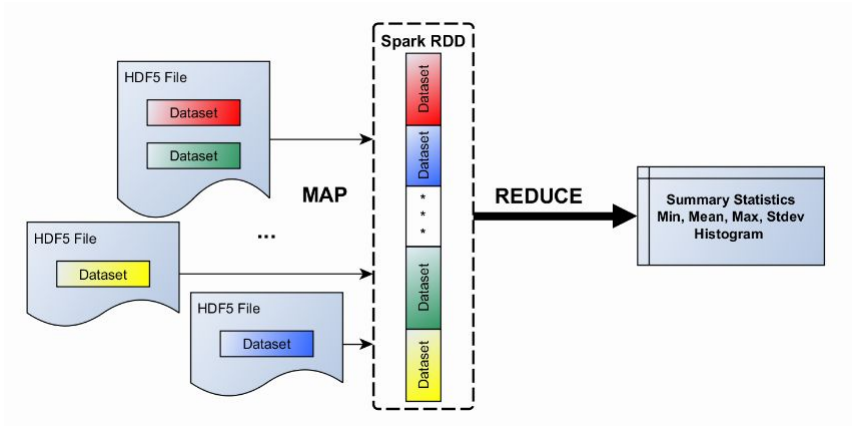
Challenges

- The dataset is stored on an aws snapshot, so it would be extremely not efficient to directly load or download it.
- The dataset is stored in the data format of hdf5. This format is designed for scientific storage. The data files in the dataset are deeply nested in structure, and hard to read by existing tools.
- Many attributes in the original dataset are very lengthy. For example, the ‘song’ attribute in all the three parts contains much information such as artists similarity, artists popularity, song duration and so on. These attributes should be reassigned on analyzing.

- Some of the data in the dataset is stored as the Byte type. However, byte is written differently in Java and Python. This would cause problems when we are using pyspark as our main tool to analyze data. Since we use python to communicate with it while it is actually running using Java.

Sourcing, Cleaning and Preprocessing

As mentioned before, the smaller subset(1.8GB) was available for direct download but the entire download was available through an Amazon EBS Snapshot. As we work as IAM users in AWS, one of our accounts was written a role so that EC2 could access the S3 buckets of that account. And then it is a simple matter of mounting the EBS volume to an EC2 machine with enough memory to contain one of the 26 folders. Copying from the EBS volume to S3 takes around 15 hours for the entire 230 GB. After being immigrated to the s3 bucket, the data need to be converted to a csv file for further analysis.



[1]

We iterate through the directories to get all the hdf5 addresses in a spark RDD and randomize them so that we can limit the number of files that are being processed without no impact to the results. We limit the number of files processed to 200000 and convert each hdf5 file into separate attribute lists. Then we combine all the attributes into a dataframe. We then do some rudimentary cleaning, including decoding the byte values and then write to a CSV file.
However, as mentioned above, the original features in the dataset are lengthy arrays and they would be very complex to analyze. We have to extract useful information from the arrays and reconstruct the dataframe. The authors of this dataset provide a recommended package called 'hdf5_getter'. It extracts new features from the original arrays and explains them finely. The only problem is that it is based on pytables, which does not support reading data from aws s3 bucket. Hence, we have to reimplement it with h5py library for the getters to work with the data in our storage. Even after extracting new features, many attributes still maintain array type. It caused two problems.

- Many information in the arrays are repeated and make little sense to our current objective. For example, the segments_start contains arrays with hundreds of elements in

each of them. We decided to apply the mean method to each of the attributes that are constructed with floats.

- The arrays are written with line breaks and random punctuation which makes it difficult to read with spark directly. Each line break causes Spark to start a new row in the dataframe which leads to a lot more rows than were written in. Hence, we had to clean the CSV separately, using pandas and regex.

Models and Hypotheses

Hypothesis - I

Songs of different ages seem to have different features and we may predict the corresponding decade of a song due to its features like loudness and danceability.

Model: we use Random Forest for the first hypothesis and we set the number of trees as 20.

Results: we implement the model and train it on two datasets. One is our whole Million Song Dataset and the other is a subset(1.8GB). As for the subset, the model achieves 78% on accuracy. However, as for the whole dataset, the model can only achieve 55% on accuracy.

Analysis and Conclusion: the performance of the model is not good on the whole dataset. And this means the connection between the decade of a song and the features we select is not strong enough. As for the error analysis, we think that the difference between subset and whole dataset may result from the imbalance distribution of data. In fact, there are lots of records that have null value for the column “year”. During training, the model tries to fit “year” with features of songs. And the null value seems to mislead the model into a bad performance.

Hypothesis - II

We suspected one song's popularity is associated with many factors such as artist, song structures. We want to build machine learning models to predict the popularity level and gain some insights on what features contribute the most to popularity

Model: Logistic Regression and Random Forest Model

Results: After removing the meaningless features such as artist ID, we converted the datatype of columns to float. Then, the song hotness rate was binned into two categories. The songs with greater than 0.5 hotness were marked as popular, otherwise non-popular. Then, the dataset was divided into 70% training data and 30% testing data. The logistic regression model reached 69.85% accuracy. It only improved a little compared to performance on the subset, which was 67.6%. Therefore, we fitted a more complexed random forest model. The accuracy reached 76.61%.

Analysis and Conclusion: The following feature importance table was extracted from the random forest model. It’s not surprising to see that the artist familiarity and artist hotness contributes most to the random forest model, followed by year and artist terms frequency. The song feature measurements were less important compared to artist information as their scores were lower.

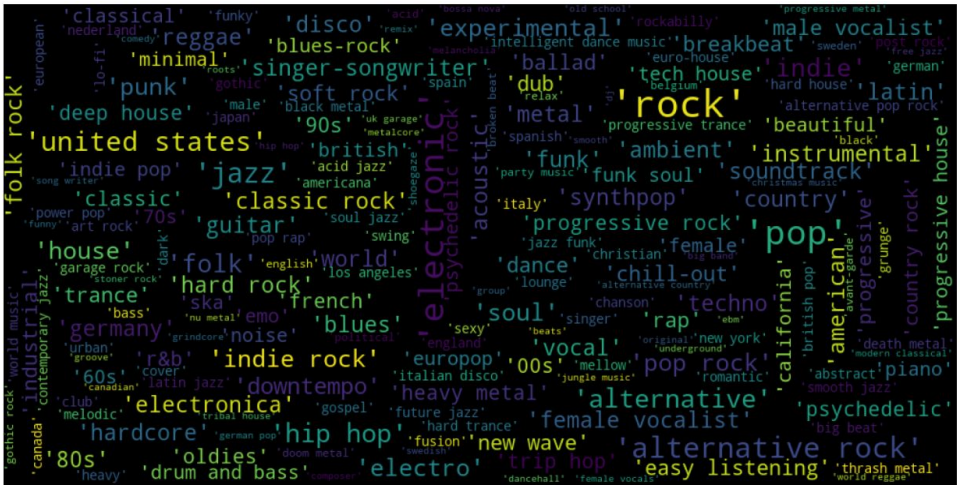
This validated our hypothesis that the artist information of a song was more important to the actual quality of the song.

idx	name	score
0	artist_familiarity	0.370370
1	artist_hotttnesss	0.237743
35	year	0.152502
5	artist_terms_freq	0.097953
16	loudness	0.027439
24	segments_loudness_start	0.026197
22	segments_loudness_max	0.022770
21	segments_confidence	0.021577
4	artist_terms	0.012151
2	artist_latitude	0.008826
36	segments_pitches_1	0.004216
3	artist_longitude	0.004195
37	segments_timbre_1	0.002680
23	segments_loudness_max_time	0.001888

Fig. Feature Importance of Random Forest

Visualization

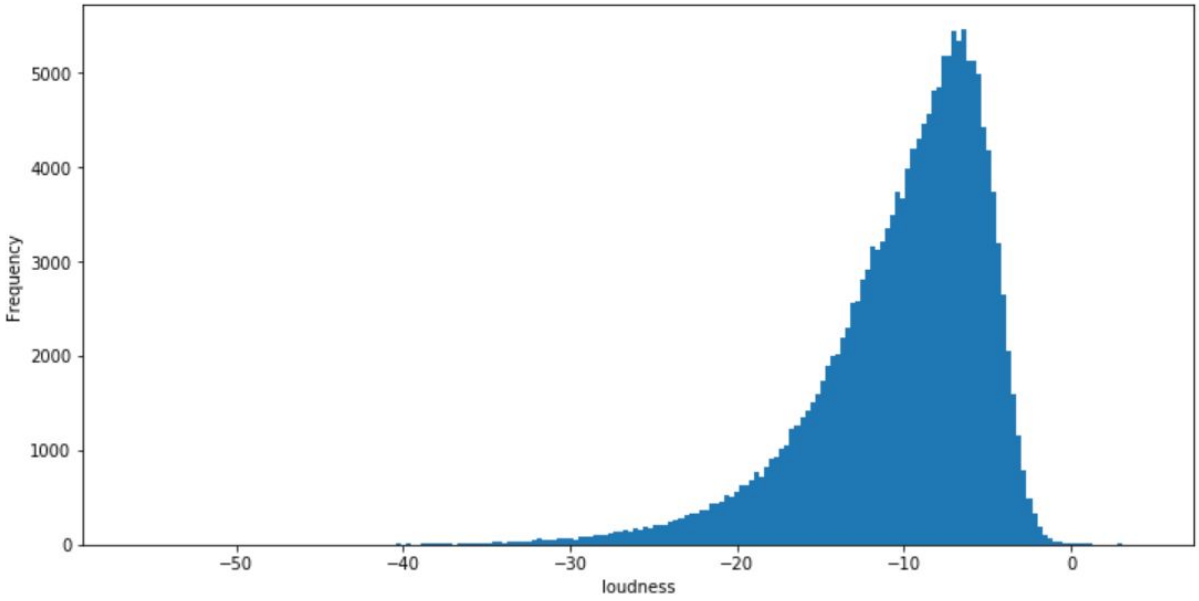
1. There is no direct column for a genre for each song, It is somewhat understandable as there is no universal measure for a thing like music genres. But there was a column which provided upto 25 terms for each artist of a song, describing the sort of music they produced. As we understand it, the descriptions are crowdsourced. So, we thought it would be interesting to explore these tags. But as stated before, they needed a lot of cleaning and we found that there were 9163 unique descriptions. After some exploration with the Arraytype operations of Spark, we were able to make a wordcloud visualization of the top 200 of these descriptions based on the frequency, seen below.

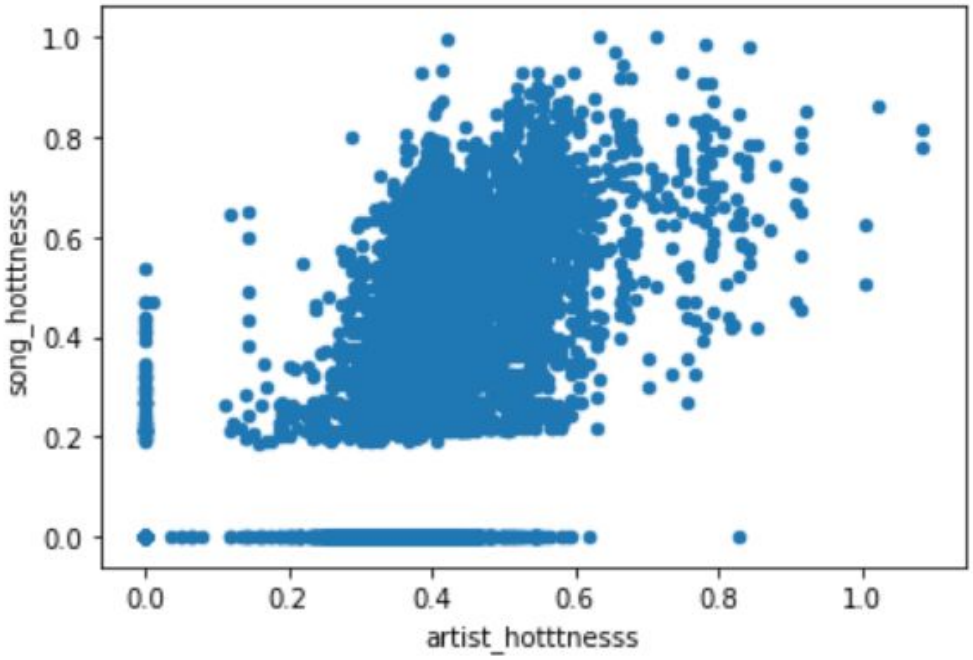


please look at the jupyter notebook ‘genre_analysis.ipynb’ for full resolution

2. We made a plotly visualization for the number of songs counted in each decade. As we mentioned before, plenty of the songs did not have any value for year, so the visualization was made ignoring those.
   Please look at the file “year.html” to see the visualization.

3. We also made a few rudimentary visualizations while making some statistical explorations.





4. Map of artists locations

Conclusion

In this project, we were able to perform many experimental processes and use the ones that worked on a small subset to the entire subset. This methodology works well for any big data project. We were also able to understand a reliable way to manage hdf5 files, which are usually made for storage, so they can easily be used for analysis.As a part of our exploratory analysis, we looked into the crowdsourced musical descriptions and we have seen that many of the expected genres(musical descriptions) appear often, as seen in the word cloud. Eg: rock, electronic, pop etc. As a part of our modelling exploration, we found that the artist information is the one that contributes the most to one song's popularity in our model. We were also able to reasonably predict the year of a song according to its features like loudness and danceability on the subset. As for the whole dataset, it doesn't work well due to the large amount of null values in the year column leading to imbalanced representation.

Future Work

1. Prediction of Genres: involves analysis of the artist_terms column and making a label before any prediction model can be built.
2. Similarity Index: Coming up with a similarity measure using all the musical features, like beats, fade_ins etc, which can be used to make playlists of similar music.
3. In the period of this project, we excluded any analysis of part of the information from the dataset. For most of the audio features, we took a mean out of the whole matrix. More detailed analysis based on those features can be done in the future.

Bibliography

[1] From HDF5 Datasets to Apache Spark RDDs, Gerd Heber, The HDF Group
https://www.hdfgroup.org/2015/03/from-hdf5-datasets-to-apache-spark-rdds/
[2] Million Song Dataset, official website by Thierry Bertin-Mahieux, available at: http://millionsongdataset.com/
[3] tbertinmahieux original repository to use the Million Song Dataset.
https://github.com/tbertinmahieux/MSongsDB

Division of Labor

Subha Karanam: Data Sourcing, Data Cleaning, Visualizations, Genre Analysis
Luwei Lei: Data Cleaning, Modeling, Visualizations
Ziyao Ding: Data converting, Dataframe reconstructing
Lu Sun: Modeling