Capstone Project

Machine Learning Nanodegree

Kushal Naidu

31st March, 2017

Billboard Rating Predictor

Overview:

Although millions of songs are made, very few of these songs get to the Billboard's Hot 100 chart, and stay on it. This project was aimed at determining whether or not a song would make it to this chart, given its metadata.

Being an avid listener to mostly top hits, I have always been curious as to why some songs, which seem to be completely different from each other are liked by millions of the same people, because of which these songs top the charts.

Statement:

The purpose of this project is to develop a classifier which could successfully identify whether or not a song is capable of making it to the Billboard's Hot 100 Chart, given its metadata.

Datasets & Inputs:

The metadata for ten thousand songs were obtained from the The Laboratory for the Recognition and Organization of Speech and Audio (**LabROSA**)'s website, here.

Each of the song's billboard rating was obtained by a web scraper which searched for each of the ten thousand song's rating on http://umdmusic.com/, and saved it in a csv file.

Combining the data from these 2 sources, a complete dataset would be obtained. By using some, or maybe all the features provided by the million song dataset's subset, along with the data from the ultimate music database as labels, a machine learning algorithm would train the model to try to predict whether or not a given song will make it to Billboard's Hot 100 chart, when given its metadata.

Metrics:

To evaluate this predictor, Accuracy Score metric will be used. Accuracy Score finds the fraction of correct predictions, with respect to the total number of predictions that has to be made by the model. In this model, the primary function of the metric is to show how accurately a trained model is able to predict if a song will make to the Billboard's Hot 100 chart or not. Since accuracy score is a metric which says that, I have decided to use it in this project.

Analysis:

Data exploration:

The Million song Dataset's subset has the meta-data for ten thousand different songs of various genres, various composers. This dataset contains songs of 4411 different artists. This dataset provides various types of metadata for each of the songs in the dataset This metadata has multiple ID tags from different music websites, and various details about a song such as mode, bars, confidence, energy of a song, and so on.

Features present in each of the ten thousand HDF5 files:

1. **Artist ID's:** ID's of the song on different websites like musicbrainz, playme, etc.
2. **bars_confidence:** confidence value (between 0 and 1) associated with each bar by The Echo Nest
3. **bars_start:** start time of each bar according to The Echo Nest, this song has 99 bars
4. **beats_confidence:** confidence value (between 0 and 1) associated with each beat by The Echo Nest
5. **beats_start:** start time of each beat according to The Echo Nest, this song has 397 beats
6. **danceability:** danceability measure of this song according to The Echo Nest (between 0 and 1, 0 => not analyzed)
7. **familiarity:** Get artist familiarity from a HDF5 song file
8. **Artist hotness:** value calibrated by EchoNest (Between 0 and 1)
9. **duration:** duration of the track in seconds
10. **end_of_fade_in:** time of the end of the fade in, at the beginning of the song, according to The Echo Nest
11. **energy:** energy measure according to The Echo Nest (between 0 and 1, 0 => not analyzed)
12. **key:** estimation of the key the song is in by The Echo Nest
13. **key_confidence:** confidence of the key estimation
14. **loudness:** general loudness of the track
15. **mode:** estimation of the mode the song is in by The Echo Nest
16. **mode_confidence:** confidence of the mode estimation
17. **release:** album name from which the track was taken
18. **sections_confidence:** confidence value (between 0 and 1) associated with each section by The Echo Nest
19. **sections_start:** start time of each section according to The Echo Nest
20. **segments_confidence:** confidence value (between 0 and 1) associated with each segment by The Echo Nest
21. **segments_loudness_max:** max loudness during each segment
22. **segments_loudness_max_time:** time of the max loudness during each segment
23. **segments_loudness_start:** loudness at the beginning of each segment
24. **segments_pitches:** chroma features for each segment (normalized so max is 1.)
25. **segments_start:** start time of each segment (~ musical event, or onset) according to The Echo Nest, this song has 935 segments
26. **segments_timbre:** MFCC-like features for each segment
27. **similar_artists:** a list of 100 artists (their Echo Nest ID) similar to Rick Astley according to The Echo Nest
28. **song_hotttnesss:** value calibrated by EchoNest (Between 0 and 1)

29. **start_of_fade_out:** start time of the fade out, in seconds, at the end of the song, according to The Echo Nest
30. **tatums_confidence:** confidence value (between 0 and 1) associated with each tatum by The Echo Nest
31. **tatums_start: shape** start time of each tatum according to The Echo Nest, this song has 794 tatums
32. **tempo:** tempo in BPM according to The Echo Nest
33. **time_signature:** time signature of the song according to The Echo Nest, i.e. usual number of beats per bar
34. **time_signature_confidence:** confidence of the time signature estimation
35. **title:** song title
36. **year:** year when this song was released

Note: In the above list, the features marked **orange** are features that have not been considered, and the features marked **green** are considered while designing the predictor.

Along with this dataset, a web scraper, was written to obtain the billboard data for each of the ten thousand songs that were in the Million song Database's Subset. This web scraper firstly found out whether a song made it to the billboard's Hot 100 chart, and if it did make it, it finds out its peak position and the number of weeks it spent on the billboard.
This web scraper uses BeautifulSoup and Regular Expression libraries to retrieve data from the website. This web scraper performs 2 basic steps.

i. Beautiful Soup obtains the raw code, which is the source code of the web page, and saves it. Since the website does not have named tables, beautiful soup cannot be used to simplify the source code, and therefore 'regular expression'(regex) was used to find the part of the code which contained the billboard details of the song.
ii. Regex then searches for the part of the source code which contains the table with the billboard information. From this table, it retrieves the names and ranks of all the search results, and saves it in 2 arrays. The web scraper then compares the search result details with the required song's name. If they are the same, it returns 3 different parameters.
    a. Whether or not the song was a part of Billboard's Hot100
    b. The song's peak position on the billboard
    c. The number of weeks the song was on the chart.
iii. If there are no songs matching this details, regex will not find the table containing the matched songs, and it will not save anything in the ranks and names arrays. The web scraper will then return the value 0 for the 3 parameters listed above.

Data Exploration:

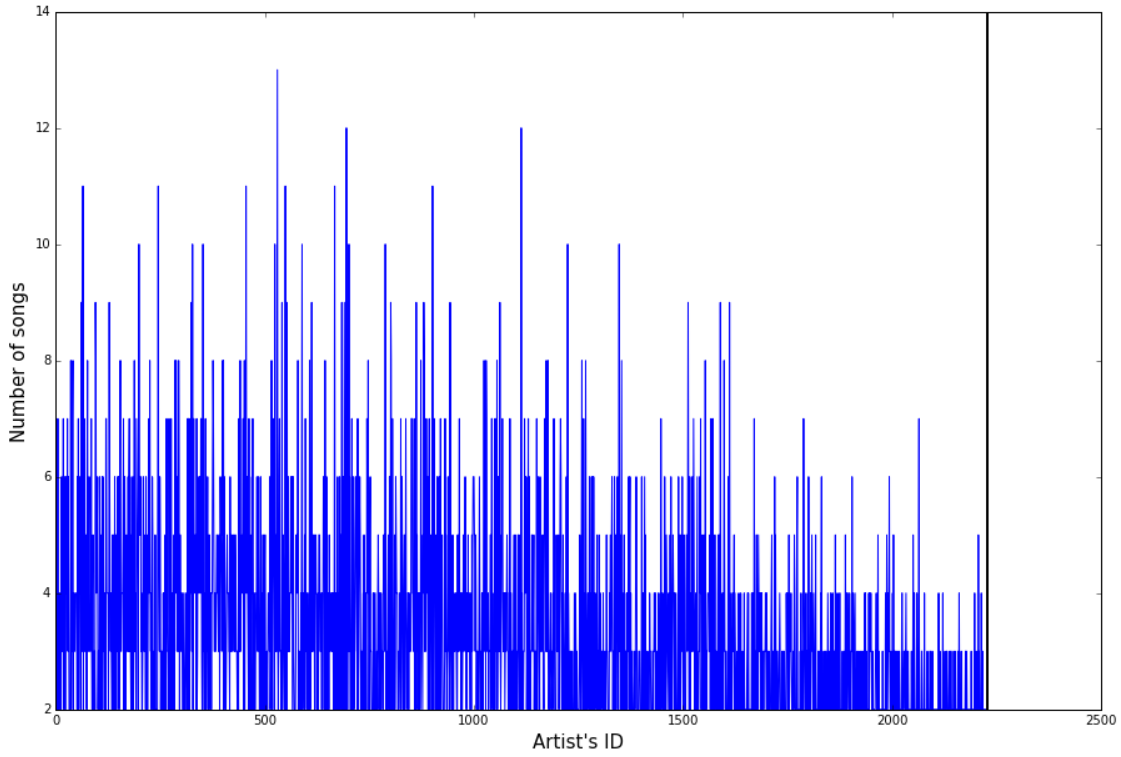
The Millionsongs database's subset is available in the form of HDF5 files for each of the 10,000 songs here. Since these HDF5 files contain a lot of data, it takes very long for a program to read the data from each of these files and obtain what is required. Since a lot of data is being ignored from these files, I realized that it would be faster if I would extract only the necessary data and save it in another csv file, which can be quickly accessed by pandas. Also, Since Sci-Kit only accepts 2D arrays for machine learning, some of a data had to be reduced from an array of numbers to either the length or the mean of the array.
Some song names, which contained special characters like '?', '*', and other such characters were yielding false results when the web scraper passed it to the ultimate music database, and hence have been excluded from the dataset.
A few examples from the dataset has been displayed below:

```
1.
Title                     Away in a Manger
Artist_Name                   Loretta Lynn
Familiarity                       0.658731
Hotness                           0.410541
Song_hotness                           NaN
Danceability                             0
energy                                   0
loudness                           -10.694
tempo                               83.976
mode_confidence                      0.722
time_sig_confidence                   0.77
no_segments                            486
avg_segment_confidence            0.572856
avg_segment_pitches               0.279582
no_sections                             10
avg_sections_confidence              0.4795
no_beats_start                          266
avg_beats_confidence              0.516744
no_bars                                  87
avg_bar_confidence                0.181057
no_tatums_start                         533
avg_tatums_start                  0.239949
key                                      4
Mode                                     1
duration                            197.59
Presence                                 0
Name: 3420, dtype: object
2.
Title                          Ai Shi...
Artist_Name                  Sammi Cheng
Familiarity                     0.490611
Hotness                         0.391574
Song_hotness                         NaN
Danceability                           0
energy                                 0
loudness                          -9.059
tempo                             104.25
mode_confidence                    0.679
time_sig_confidence                    1
no_segments                          503
avg_segment_confidence          0.522978
avg_segment_pitches             0.404468
no_sections                            4
avg_sections_confidence          0.61375
no_beats_start                       225
avg_beats_confidence            0.523236
no_bars                                55
avg_bar_confidence              0.100455
no_tatums_start                      449
avg_tatums_start                0.141272
key                                    4
Mode                                   1
duration                         126.824
Presence                               0
Name: 5260, dtype: object
3.
Title                     Show a Sign of Life
Artist_Name                  High Strung_ The
Familiarity                         0.599945
Hotness                               0.3968
Song_hotness                             NaN
Danceability                               0
energy                                     0
loudness                                -4.399
tempo                                  155.524
mode_confidence                          0.632
time_sig_confidence                      0.411
no_segments                                221
avg_segment_confidence                0.239629
avg_segment_pitches                   0.429306
no_sections                                  4
avg_sections_confidence                0.77375
no_beats_start                             316
avg_beats_confidence                  0.138006
no_bars                                      78
avg_bar_confidence                    0.146923
no_tatums_start                            633
avg_tatums_start                      0.065237
key                                          0
Mode                                         1
duration                               123.872
Presence                                     0
Name: 8102, dtype: object
```

Identifying the required features:

From the 35 or so features present for each song in the million songs database's subset, I have chosen only 18 of the features, and have ignored the rest of the features.
Features such as ID tags, titles, year of creation, title, Album's name, year of release definitely do not play a role in whether or not a song makes it to the billboard top 100.
Some of the features, such as Song_hotness, Danceability, Energy have null values for 4434 of the ten thousand songs. Due to the presence of too many null values, these 3 features were dropped.
Apart from these features, the rest of the metadata was used to compute the song's presence on the billboard.
Data Visualization:

One of the most interesting features being considered is the artist's name. (In the form of uniquely encoded integers to each of the artists)
The number of songs by each artist, who have at least 2 songs present in the dataset has been graphically represented below.



There were a total of 2226 different artists, who composed at least 2 songs in the dataset. The most number of songs composed by a single artist was 13. I think feature will play an important role in determining whether or not a song makes it to the billboard, as songs sung by famous artists seem to trend more commonly than new/lesser known artists.

Benchmark:

Since there is no similar working model related to this project to test against, an Ideal benchmark would be running the code in a default parameterized, non-boosted dummy classifier, and comparing the results between both the models to find out how much of a difference a well optimized classifier makes.

Algorithms and techniques:

The classifier is a Naïve Bayes classifier. It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

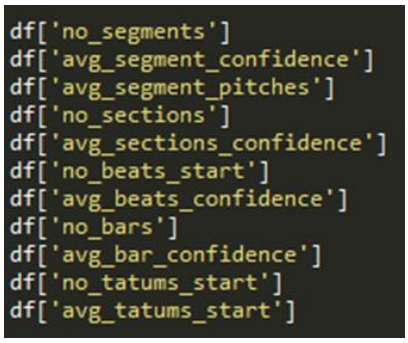
The Naïve Bayes classifier was boosted with AdaBoostClassifier and GridSearchCV optimized the parameters for the boosting classifier.

Methodology:

Creating the required dataset: Since the hdf5 files have a lot of data, the I/O read time for each of the files is too large, and the algorithm takes very long to process the data each time the program is run.
Also, extracting the billboard data from the billboard database present on the internet is time consuming, and sometimes can stop running the web scraper due to 503, 502 bad gateway errors, internet connection problems, the algorithm could stop processing and stop midway.
So, I decided to save the data required into 2 different csv files, data.csv and billboard.csv. These files are created by running Features_data_creator.py and Billboard_ Data_creator.py respectively.
These 2 files will help in easy access for the machine learning algorithm and will help effectively reduce the runtime by a lot by preparing the data beforehand.

Data Preprocessing:

a. Artist names: Since artist's names are in the form of strings, they had to be converted into values. LabelEncoder gives each different string a unique value, which thus makes it compatible for Scikit-Learn to process.
b. The following features are two dimensional arrays, and these features cannot be processed by scikitlearn, and thus I reduced those into single dimensional arrays, by either finding the mean of all the values in the array, or just counting the number of values in the single dimensional array.
c. Song names containing characters like '*', '?', '(' weren't compatible characters on the ultimate



music's database and were not returning accurate values, and have thus have been ignored while creating the dataframe.

Running the Algorithm:

The 23 remaining features are saved in a pandas dataframe, with the billboard data stored in a pandas series, which is used to train and then try to predict.
The data is split as 80 – 20, i.e. 80% of the data is used to train the model, and 20% of the data was used to test the algorithm, to find out how well the algorithm can predict the outcome.

Implementation:

When I first started out to develop the dataset, I found a few features were in the form of a 1D/2D arrays. Since SciKit accepts only a single value per cell, I had to reduce these arrays into singular values, by either find the number of elements, or the mean of all the elements; whichever seemed more appropriate. An algorithm, firstly I found a few cells devoid of any value (null values) in the dataset, which was removed. Once I removed these columns, the size of the dataset dropped by over 50%, to about 4,300 from 10,000. After exploring a little, I found 3 features which had a very high number of null values, and dropped those columns. As some of the indexes of the dataframe were dropped, StratifiedKfold was not able to split the data correctly, as it was not considering index numbers to perform the split. Thus, I had to use the Test Train Split function in an iterating loop to run the algorithm a few times to show results on different splits. As there were only a few songs from the dataset which were present on the billboard, I had to split it a few times before finding a good split between the data to have enough number of billboard's songs on both the training and the testing set.

Training the model:

To train the model, I have used the Gaussian Naïve Bayes classifier. To improve the performance of this classifier, I also added in an Adaptive boosting classifier, which works in conjunction with the Naïve Bayes classifier. After trying different combinations of parameters of the AdaBoost classifier, the parameters which made a difference while trying to find a more accurate algorithm were: **number of estimators** and the **learning rate**. These values for the boosted model was found using GridSearchCV.
Since the ratio of the songs present on the billboard with respect to the total songs present in the data set was very small, the accuracy of the algorithm seemed to improve just a little when compared to the dummy classifier which was set as the benchmark. The optimal values of the parameters seemed to be ~ 13 for the number of estimators, and 0.1 as the learning rate. Adaptive boosting with optimal parameters only slightly improved the algorithm's performance over a normal Naïve Bayes classifier, and did not have any significantly large improvements over the algorithm's accuracy.

Justification:

As previously stated, due to a bad ratio of the number of songs present on the billboard when compared to the entire dataset, lead to a large number of values in the label have the value 0. (About 9600 of the 9900 values.) Because of this, both the dummy classifier and the boosted Naïve Bayes classifier seemed to have high accuracy scores, but the boosted classifier was clearly able to make more accurate predictions when it comes to predicting correctly about songs that were on the billboard's chart.

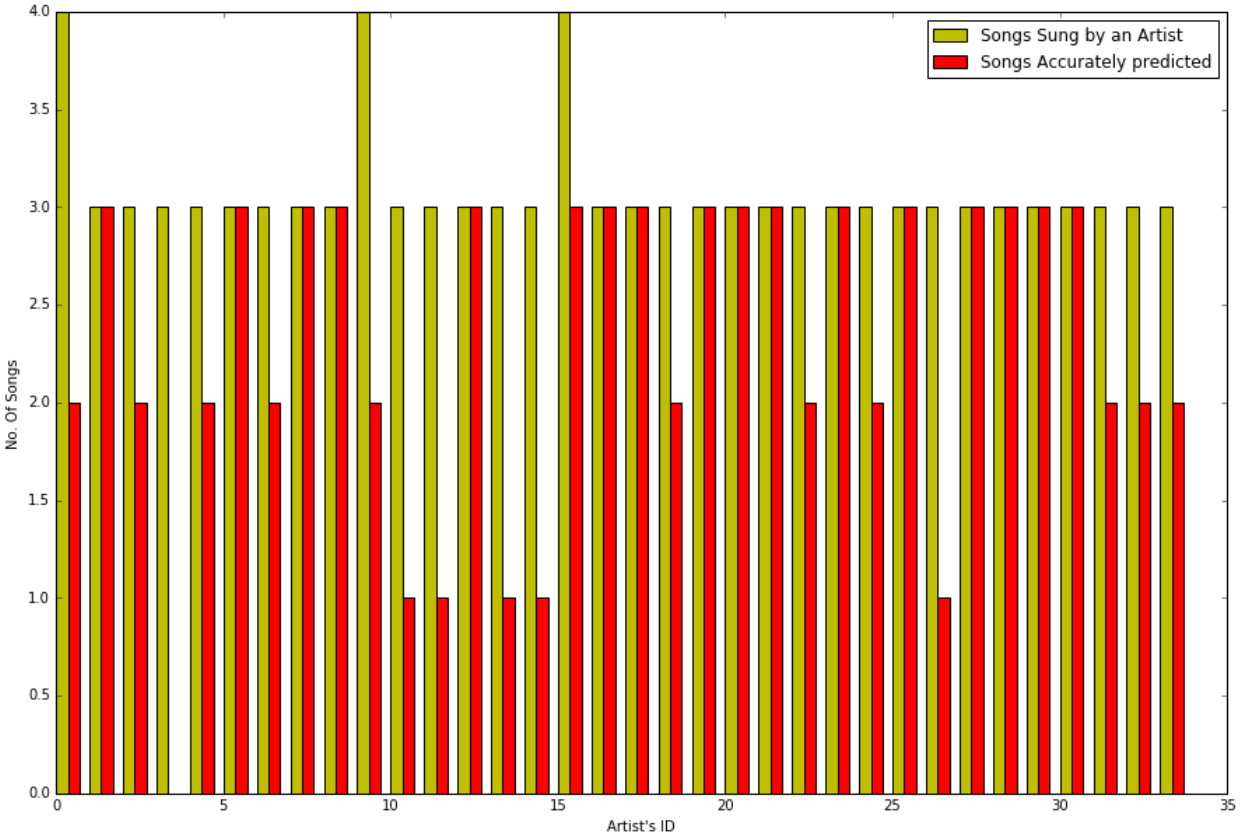
Results and conclusion:

The trained model had an accuracy of ~0.9, which means it could predict most values in the testing set accurately, and predicted ~20% of the songs which were on the billboard's chart correctly.

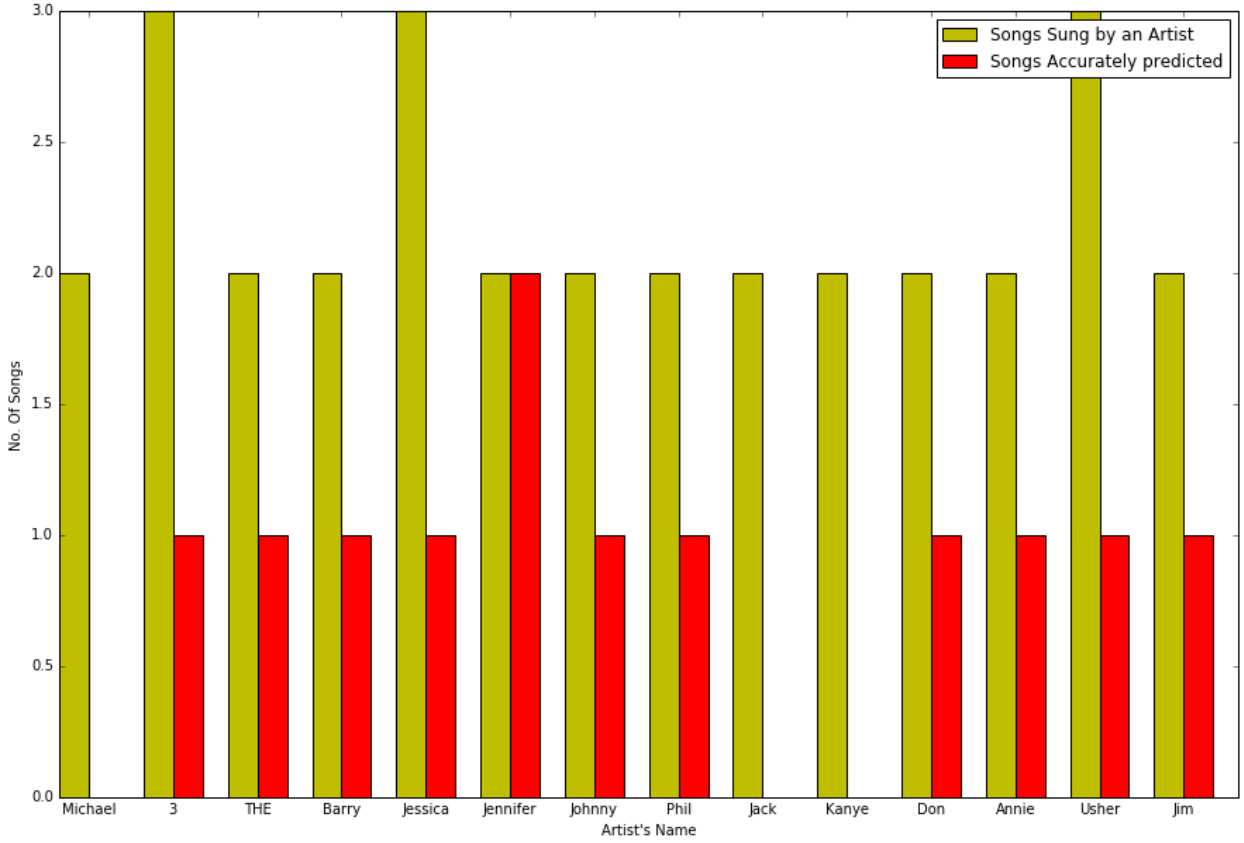
I shall be diving deeper into the results provided by the algorithm when the 'random_state' was set as 2.

Again, I'll be comparing the results against the labels with respect to the artist's name in this dataset:

Firstly, comparing the results of the songs in the testing dataset wherein there were at least 3 songs that were sung by the same artist:



In the above graph, in majority of the cases, the model predicted accurately for more than 50% of the songs per artist, and in some of the cases, it predicted it all correctly, showing the importance of this feature.
Also, in the graph below, I have plotted the same accuracy comparison histogram, wherein each of the artists have at least 1 song which is present in the billboard, and the artist has at least 2 songs present in the test dataset.



Even in this graph, it could predict at least 50% correctly for majority of the artists.
From the above 2 graphs, we can conclude that the results have a lot of dependency on the artist's name, and this helps the algorithm predict more accurately.

Reflection:

- While doing this project, I learnt a lot about I/O with respect to HDF files, and how it is used to store large amounts of data in a structured format.
- This was the first time I built and used my very own web scraper, which was also a nice learning experience.
- Once I got the data, I noticed a few columns in the data consisted of 1/2D arrays, and after analyzing, I figured I could find its length or mean to convert it into single values, which will thus make it compatible with sklearn.
- I got to learn about and use boosting classifiers, which was something I had never used before.

- I also got to explore a lot in matplotlib, and learnt to do things like plotting comparison histograms, which I never knew was possible.

Drawbacks of this model:

1. Since I used scikit learn to run this model, I was restricted to single dimensional arrays for each song, and had to compress the data into a compatible form for scikit learn, whenever I had a 2 dimensional array, or string values. Also, I had to ignore either the song, or the whole feature when some values were absent (NaN), since it was throwing syntax errors.
2. The ratio of the number of songs which were a part of the billboard, with respect to the number of songs which were present in the millionsongdatabase's subset was very low, (171 of the 9810 songs were present on the billboard) which also lead to a lot of inaccurate predictions when it comes to predicting the label of songs which were on the billboard.

Because of these 2 drawbacks, the results while considering the entire dataset will still be low, unless I find a way around these 2 drawbacks, and will not try this algorithm on the entire dataset present on Amazon's EC2 machines for now.

Room for improvement:

Using a different machine learning libraries, such as deep learning libraries, such as tensorflow, to use the entire data, even those present in the form of arrays, without having to compress the data into compatible forms should yield better results. Also, finding a way to increase the % of songs present on the billboard, with respect to the dataset would help the algorithm perform better.