**Project report**

**On**

**Music Recommendation from the Million Song Dataset**

**By,**

**Bernard Bollinger (305822288)**

**1.INTRODUCTION AND PROJECT GOAL**

Spotify has hundreds if not thousands of new releases daily that are not instantaneously rated or played by the user base, so recommendations are tough to deliver until people start interacting with the music. This dilemma is known as a 'cold start'. To solve this issue I propose that an analysis can be made of the unrated song using "The Echo Nest" API, and the features extracted from this API call can help to suggest songs with similar features.

This could be scaled to wider use on the Spotify system with access to a database containing analyses of all currently owned tracks on Spotify. However, for this project I can only work with the Million Song Dataset (MSD), a free dataset that has audio features and analysis for a million songs. Thus all recommendations will be based on the songs already analyzed in that dataset, as I have limited API access with Spotify's analysis service

**2.DATASETS AND APIs**

**The Million Song Database (MSD):**

The MSD is available as an entire package or as a slice of the entire dataset only containing 10,000 songs. Due to the extraneous and time consuming process of working with a million data sets it was decided best to make a prototype of the recommendation system only based off of the 10,000 song slice, with intentions to scale up to the full database at a later date.

The 10,000 song slice of the dataset is downloadable as a zip file. The zip file contains folder ‘A’ and folder ‘B’, which each contains folders A-Z, of which each folder contains another set of folders labeled A-Z, and each of those folders contains the audio data for roughly 8-10 songs per folder in HDF5 format. To better understand the folder structure, look below at (*Code snippet #0*) used to create the filenames. Despite the headache of creating all of those filenames I also had to figure out how to extract data from an HDF5 format so that I could make a pandas dataframe.

Also included in the downloadable file are sqlite database files with the majority of the song features. Unfortunately these database files did not include any audio analysis features, and were thus unusable for this project.

**MSD Audio Features:**

The following link includes a description of each of the features included in the dataset…

<https://labrosa.ee.columbia.edu/millionsong/pages/example-track-description>

(Code snippet #0)

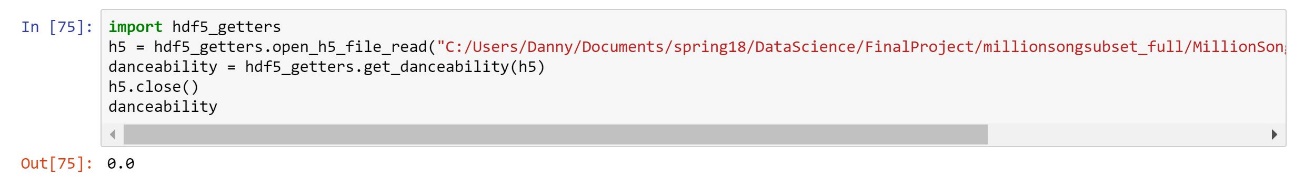


**HDF5 to CSV:**

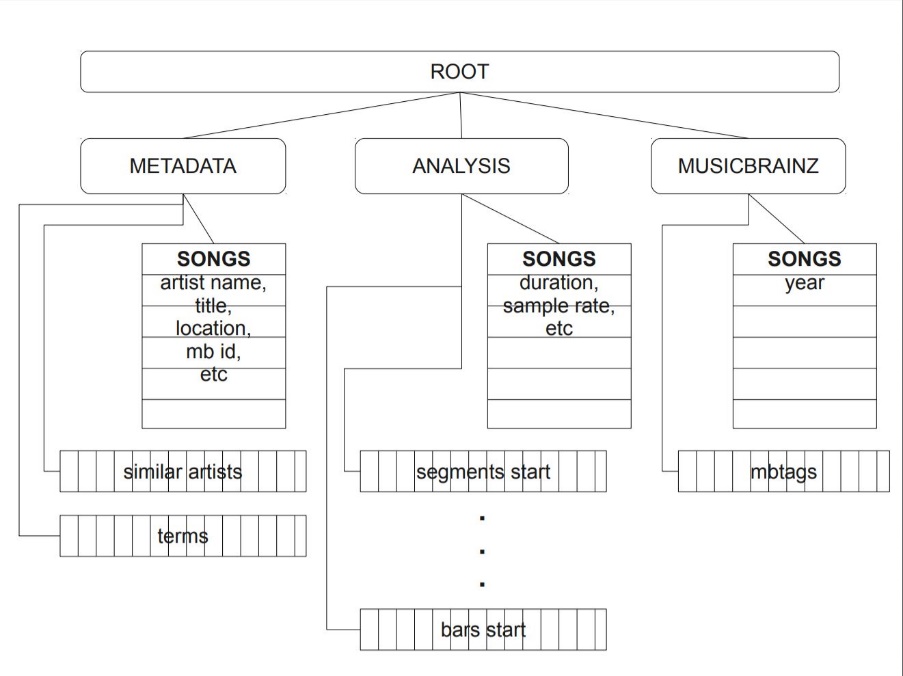
This was the most difficult and time consuming part of the project. HDF5 does not seem to be easily read in python as far as my research had touched. The most recommended method of doing so was by using HDFStore in the pandas library. After using HDFStore to parse through the filenames (Code snippet #1 below) and extract all the information I could I was running into trouble with the data. In order for HDFStore to work properly the schema of the HDF5 file had to be of a specific type. The schema for the MSD files is below (Image #0). I was able to extract all data that was not a list type. No matter how I tried to access them, the lists were unable to be extracted with HDFStore. This includes *segments start*, *bars start*, and all features that are list types in the link under the “MSD Audio Features” section above. Also, all of the rows of data for *danceability* and *energy* were listed with a 0. So I wanted to try another way to extract this data.

The MSD has many examples and helper files that are downloadable on their website. One of them is hdf5getters.py, a helper class to get hdf5 data from an hdf5 file. Despite only being from 2016, this hdf5getters class had many depreciated features in python, a problem that was faced often when working with this dataset. After fixing the depreciated code and running it on my computer I received the same issue as before. The lists were not being properly read, and all energy and danceability scores came out as 0. (Code snippet #2 below)

(*Code snippet #2*)



(*Image #0*)



(*Code snippet #1*)



**Exporting a final CSV dataset:**

After exploring 2 possible methods for extracting the data from hdf5 files I settled on using HDFStore because it was more recently written than the hdf5getters.py code. (Code snippet #1 above) shows the process for creating the final dataframe, which was then exported as a csv file. The following features are what was able to be extracted for each song through this process…

'artist.hotttnesss',

'artist.id',

'artist.name',

'artist\_mbtags',

'artist\_mbtags\_count',

'bars\_confidence',

'bars\_start',

'beats\_confidence',

'beats\_start',

'duration',

'end\_of\_fade\_in',

'familiarity',

'key',

'key\_confidence',

'latitude',

'location',

'longitude',

'loudness',

'mode',

'mode\_confidence',

'release.id',

'release.name',

'song.hotttnesss',

'song.id',

'start\_of\_fade\_out',

'tatums\_confidence',

'tatums\_start',

'tempo',

'terms',

'terms\_freq',

'time\_signature',

'time\_signature\_confidence',

'title',

'year'

Again, the following link describes the meaning of each of these features…

<https://labrosa.ee.columbia.edu/millionsong/pages/example-track-description>

**3.METHODS AND TECHNIQUES**

**Overall Plan:**

The first step of the plan is to cluster the data using k-means and assign the # of the cluster that each data point is assigned to as the label for that particular song. Once the labels are assigned back to the original data all of this is stored in a new csv file. That csv file can then be imported as a dataframe with labels in order to train a predictive model. The predictive model I chose was an Artificial Neural Network. Once the ANN is trained with the same features as what can be extracted from Spotify’s song analysis API, it can make predictions on single rows with less features than what was originally used to cluster. These predictions are cluster assignments. From the cluster that a single prediction is assigned to we can pull the most similar tracks, and these will be our recommendations.

**Clustering and Labelling:**

The previous list of data extracted for each song from the MSD files contains data that is irrelevant to assessing the similarity between tracks, such as track.id and song title. Other features were eliminated because of their lack of data, such as artist\_mbtags, which was only present in 3711/10000 samples. Below is the finalized list of features that were used to cluster the songs in the MSD.

'artist.hotttnesss',

'bars\_confidence',

'bars\_start',

'beats\_confidence',

'beats\_start',

'duration',

'end\_of\_fade\_in',

'familiarity',

'key',

'key\_confidence',

'latitude',

'longitude',

'loudness',

'mode',

'mode\_confidence',

'song.hotttnesss',

'start\_of\_fade\_out',

'tatums\_confidence',

'tatums\_start',

'tempo',

'terms',

'terms\_freq',

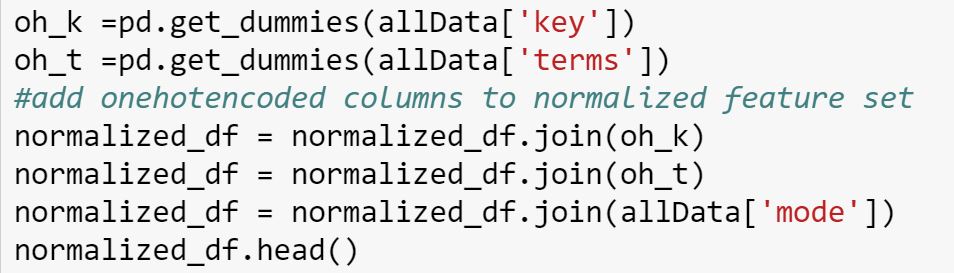
'time\_signature',

'time\_signature\_confidence',

'year'

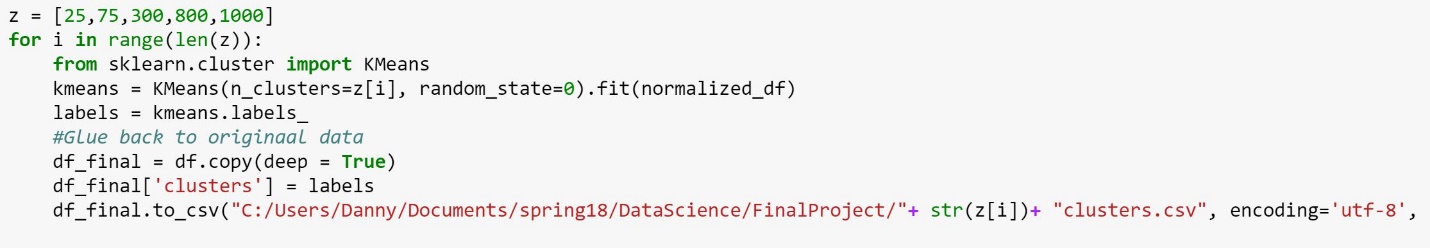
Of these features, all features were numerically ordered except for 3. ‘mode’ is either 0 or 1, standing for major key or minor key. ‘terms’ is the genre of the song as a string. ‘key’ is numeric, but a key of 3 is not “less than” a key of 5, thus they are not ordered. For all three of these features they had to be extracted from the dataframe, as they could not be normalized with the other data. After normalizing the other data on a column by column basis, ‘key’ and ‘terms’ were One Hot Encoded and added back into the normalized dataset. Because ‘mode’ was already binary it was simply added back into the normalized dataset as is. (Code snippet #3)

(*Code snippet #3*)



Once this was done I could finally cluster the songs and add the cluster numbers as labels. I did this with multiple cluster sizes in order to see which division of songs would work best. (Code snippet #4)

(*Code snippet #4*)



**Training an Artificial Neural Network:**

Once the data is in csv format and has labels it is something that I am used to working with in class. The data is used to train an Artificial Neural Network. Once trained this ANN can output predicted labels for songs that were not initially clustered. The training stage is shown below. (Code snippet #5)

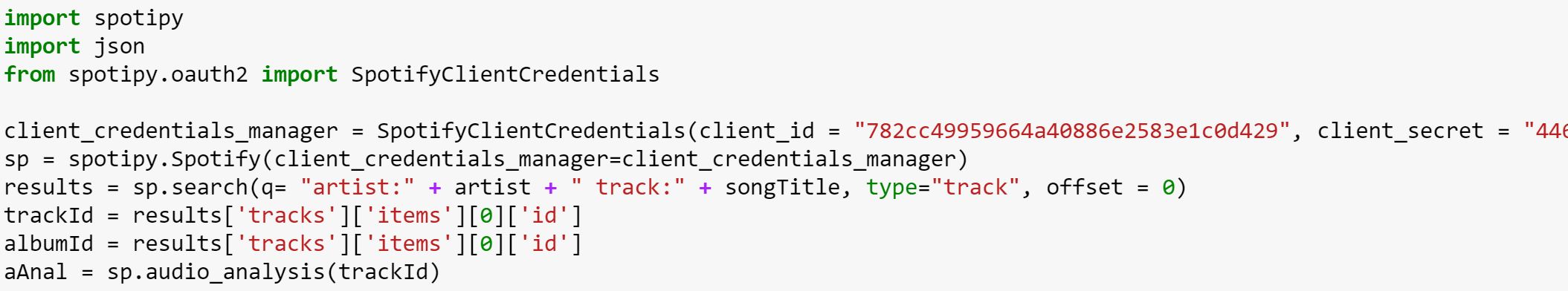
*(Code snippet #5)*



**Assigning a User Submitted Song to a Cluster:**

Finally it is time to prompt the user for a song. This is done using simple input statements in python. Once the song title and artist is taken from the user it is put into a query string and sent to the Spotify API. This is best done with spotipy, a library for using the spotify api more easily in python. This library also contained many depreciated features and was difficult to use. After making the call to the The Spotify API with a query string it responds with song objects in JSON that relate to the query string, in order of best match. From that response I extract the songid so that I can make a separate audio analysis request. (Code snippet # 6).

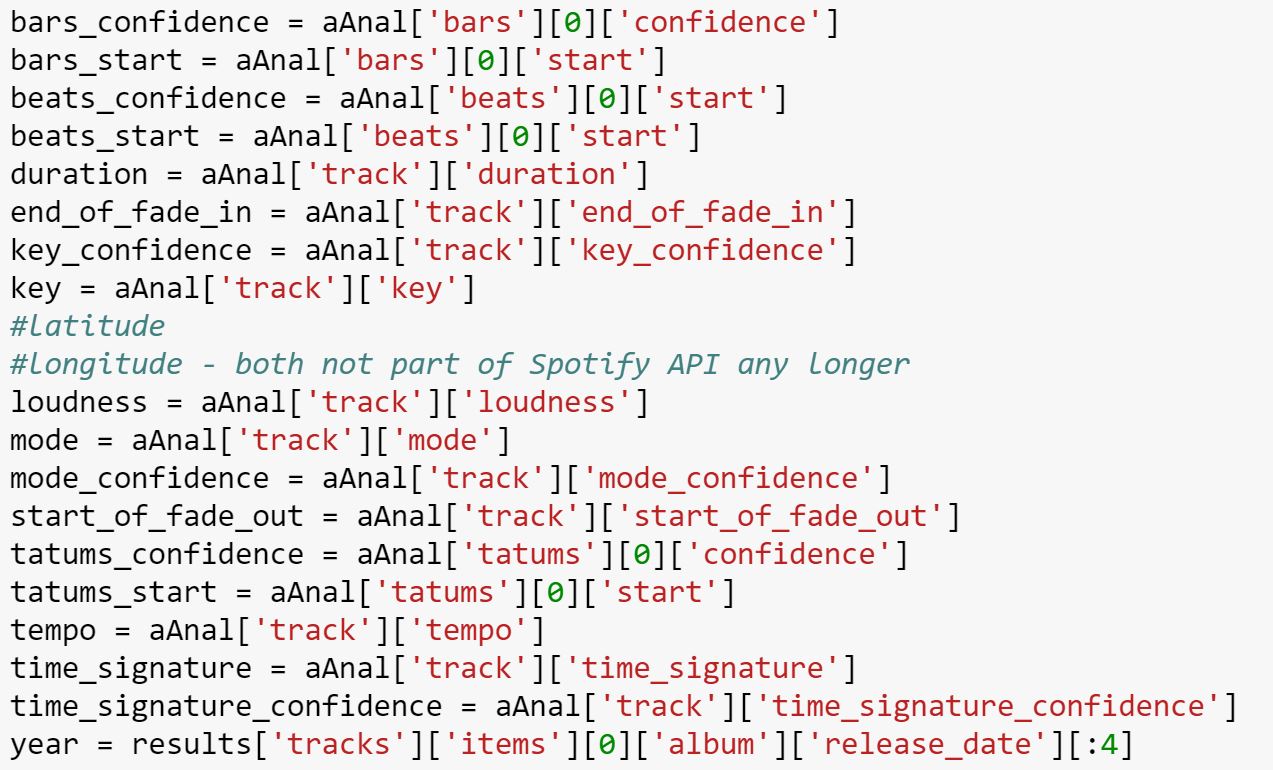
*(Code snippet #6)*



From the JSON result “aAnal” (short for audio analysis) there are many audio features to extract that relate to our MSD features. Below is a list of them and code showing how they were accessed.

(Code snippet #7)

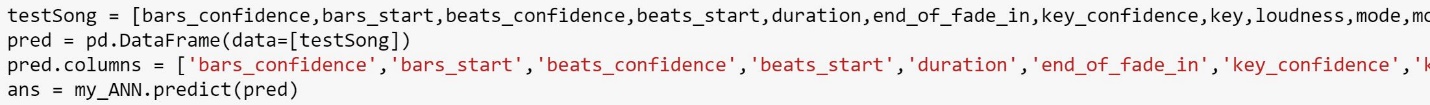
*(Code snippet #7)*

**

**Making recommendations from the cluster:**

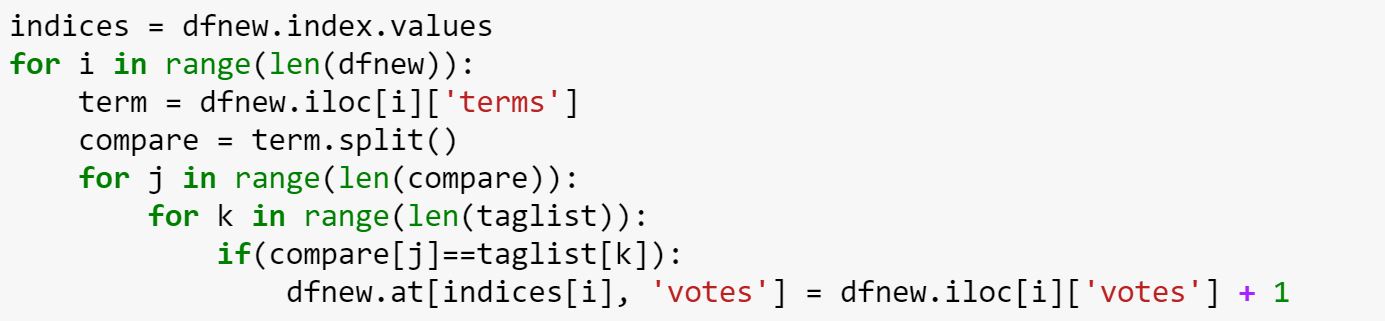
So now that the user has inputted a song title and artist name, and the Spotify API has returned audio analysis information we can then feed this audio analysis info as a properly formatted row into the ANN created above. The result of this process is the number of a cluster from the original clustered MSD data. Below is the code that shows this process. (Code snippet #8)

(Code snippet #8)



Once this cluster number is found, all that must be done is to go to the original cluster and print out the names of the songs and artists found in that cluster. When this was done, however, many irrelevant songs snuck into the larger cluster sizes, and the smaller cluster sizes were not always consistent with their recommendations. In order to better filter this finalized list of songs, a scoring system was put in place to match the genre of the song chosen by the user to the genre of songs in the chosen cluster (Code snippet #9). Once this was put in place the songs outputted as recommendations were more relevant to the song searched by the user.

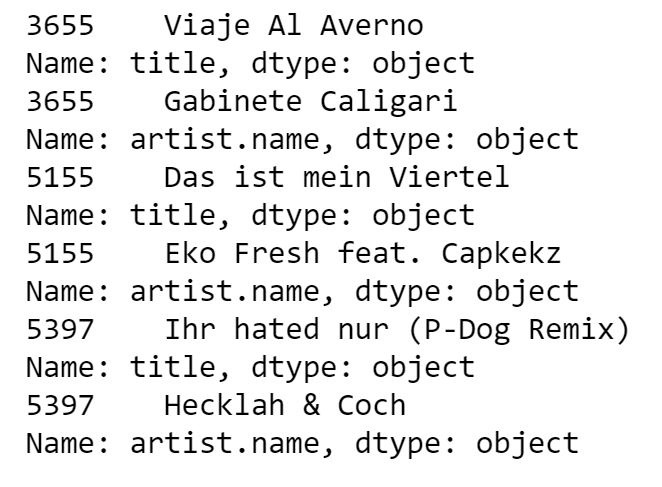
*(Code snippet #9)*

**

**5.RESULTS & CONCLUSION:**

****

Above is a snapshot of the user input. In this case the song searched for is “Bodak Yellow” by Cardi B. After running through the above steps, the following is our results…



From running through the program several times with different songs and different cluster sizes, the larger cluster sizes seem to work best. When dividing the data into 200 clusters as opposed to 1000 clusters, there are more songs in each cluster. Because the audio analysis information is not complete it is better to have more recommendations to filter through at the end. This technique however is assuming that recommendations should strictly be within a similarly named genre as the song the user chooses.

**6.POSSIBLE EXTENSIONS:**

If it were possible to extract more feature analysis information from the MSD files then that would help greatly in the analysis. The Spotify API analysis has hundreds of list values for each beat, bar, and tatum in each song, which could be used to further link the rhythmic structures.

Also it would be advantageous for someone with more time and larger computing power to run this process on the entire million song dataset, not just the 10,000 row slice.

Lastly, as mentioned in the *Introduction* section, this could be expanded to the entire Spotify database of songs if someone had access to their databases.

**DATA SET**

https://labrosa.ee.columbia.edu/millionsong/

**References**

* 1. <http://spotipy.readthedocs.io/en/latest/#indices-and-tables>
  2. <https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/io.html> https://github.com/tbertinmahieux/MSongsDB/blob/master/PythonSrc/hdf5\_getters.py https://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/FileSchema.pdf