Barad Balázs  
I0ZQRD  
Intelligent Systems Homework

My goal is to create an application based on a neural network to predict if a song reaches the Billboard Hot 100 chart, using the properties from the audio feature detection of Spotify’s WEB API.

Musical trends are changing over time, so it would be pointless to use older charts as a baseline to measure the likelihood of a new song reaching a top 100 position. Because of that I used the annual charts from 2015, and the 2019 chart at the moment of the creation of this project (2019.04.25). And I only used songs with the release date of 2014 or newer for the training set.

The Billboard Hot 100 (<https://www.billboard.com/charts/hot-100>) chart is one of the most popular top list in the world. It represents well the modern popular music taste in the western culture, especially the younger generations listening preferences. The chart is based on the combination of the United States record sales, radio plays, and online music streaming.

Spotify is the most popular audio streaming service. Nearly all the published songs and podcasts are available to listen to in the application. It also has a comprehensive API, with numerous features. I use one of these, the audio feature detection to obtain properties about the songs, and then I feed these properties into a machine learning application.

This option of the Spotify API returns the following features of the audio tracks about their musical properties useful for my application:

* Duration: The total duration of the track in milliseconds.
* Key: Musical root note of the track
* Mode: Major or Minor is the musical modality of the track
* Time signature: Specifies how many beats are in each bar (usually 4/4 in modern western music)
* Acousticness: A confidence measure of whether the track is acoustic.
* Danceability: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity.
* Energy: Energy represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale.
* Instrumentalness: Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context.
* Liveness: Detects the presence of an audience in the recording.
* Loudness: The overall loudness of a track in decibels (dB).
* Speechiness: Speechiness detects the presence of spoken words in a track.
* Valence: Describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
* Tempo: The overall estimated tempo of a track in beats per minute (BPM).

For further information about this feature you can read the official documentation:  
<https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/>

I used Matlabs pattern classifier neural network (nprtool) to decide whether a track possesses the properties to reach top 100 position in the Billboard Hot 100 chart. Nprtool is a two-layer, feed forward network, which classifies inputs into categories. I have only two, whether a track is a top song or not. The network uses conjugate gradient backpropagation as a training method.

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

The first obstacle was to fetch enough data for a machine learning method to perform well. I manually selected numerous playlists in Spotify and used its API to collect the songs unique IDs, artists, release dates. Most of the selected playlists were popular and recent. But I tried to collect a diverse dataset, which includes many different genres. Including heavy electronic, metal, ambient, ethnic, hip-hop, rock, soul, jazz, etc.

Requestsongs.m is the script to obtain this information. It uses HTTP request to get the songs from a given playlist. For the requests I needed an authentication token, which is available at <https://developer.spotify.com/console/get-playlist-tracks/> website, after logging in with a Spotify account. Matlab converts the JSON response to a struct, in which each track is contained in a field. These are stored in the playlists folder with the same name as the playlist.

Importfile.m script is used to merge the above described playlists into two structs. The script iterates through each file with .mat extension in the two folders containing the playlists, then merges them into billboard\_combined2 and combined\_not\_billboard\_not\_filtered structs.

It is possible that regular, non-billboard playlists also contain top songs, so I have to filter those out, in order to create a valid training dataset. billboard\_filter.m executes this operation, and also simplifies the structs, so they only contain the tracks at their roots, instead of the playlists. The not\_billboard\_filtered\_combined, and billboard\_combined3 variables contain these structs.

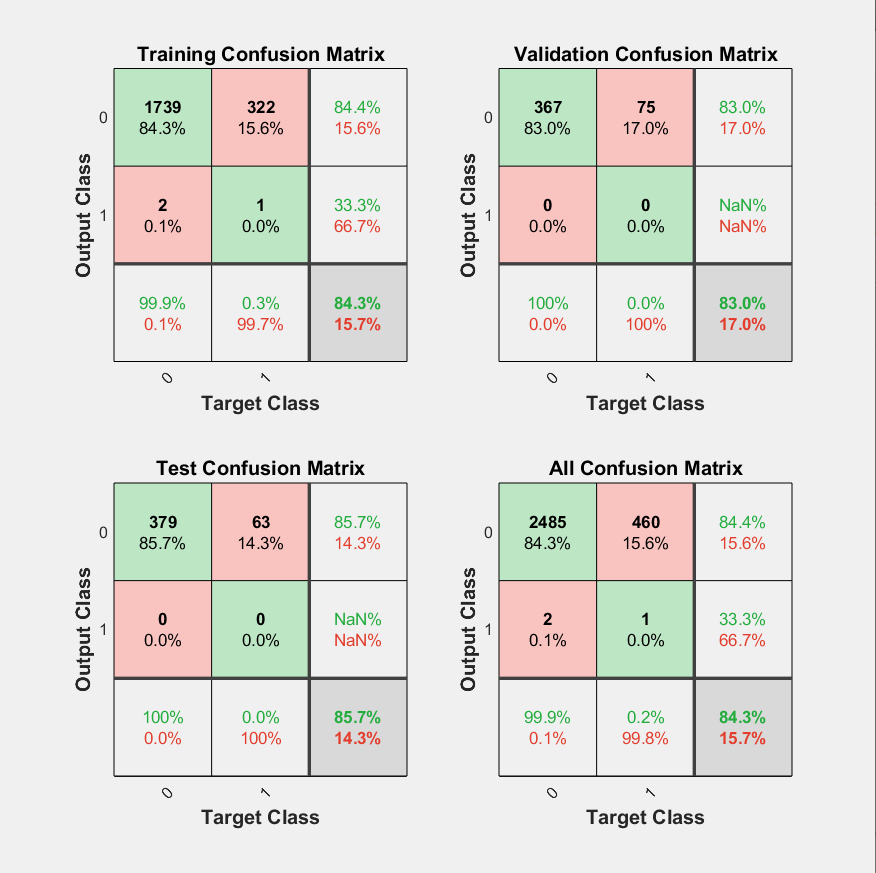
The next operation uses the previously created not\_billboard\_filtered\_combined.mat and billboard\_combined3.mat as input. The two sets are now separated, but these can still contain repeated elements. It ‘s important to remove those in order to create a valid training dataset. Distinct.m removes theses repetitions, and saves the new structs into billboard\_filtered\_distinct, and notbillboard\_filtered\_distinct variables.

Now there is two separated sets available, one, for top list songs, and one for songs that have not reached top 100 position. But these still not contain any features about the songs. It is time to collect these properties required to classify the songs.

I use again the Spotify WEB API to request the features in the requestfeatures.m script. It is similar to the requestsongs.m script, it also uses HTTP request/response, gets a JSON data, and Bearer authentication is required to use the API. A new problem was introduced in this script, which is the API has a maximum number of songs to handle in a single request. I had to fragment the dataset to 100 songs per request, and combine the responses to a single variable. These structs are saved as: features\_billboard.mat and features\_notbillboard.mat

Convert.m executes the final steps to process the input data. In order to create a reliable training data set, it is recommended to normalize the features, so each of them will affect the learning algorithm nearly the same amount. And the script also randomizes the order of the rows. The final output is stored in the x and y variables. This is a ready to use dataset, x stores the features, y is the target data, it only stores the label required for the supervised learning: one (top song), or zero (not top song).

I used the nprtool with its default values, because I tried different network sizes, but there was not any significant change in the performance of the neural network. You can see the confusion map:



I assume the reason why there is considerably larger amount of false positives, than false negatives is because today the pop songs are too similar to each other. And the top 100 songs are not primarily on the chart due to their exceptionality, but those have better marketing, more famous artist, than the other not top list pop songs.