Predict Album Genre by Title and Cover Image Attributes

John Dawes

November 1, 2019

Business Case

There are two scenarios that predicting an album genre by it's cover image attributes would be useful:

Scenario A: The Music Label

In the music industry, it is not unusual for labels to expect that a newly signed artist/band to have their all their branding ready to go for relase at contract signing. This includes the music being mixed and mastered, along with cover art for their first release on that label.

Scenario B: The Music Artist Looking to Get Signed on a Label

It would be useful for the artist to have a tool that would give them an idea of how much their album cover communicates the genre of their music before they shop lables. For labels, to release their artists' sophomore (and later) albums, it would be a handy tool to input the album cover to get an idea of how much it represents the genre by album cover attributes.

Data Gathering

Some useful attributes of albums to predict genre would be how many `faces` are on the cover, the most prevelant `colors`, identification of various `objects` and how abtract the cover image is - i.e. is it an `illustration` or a `photograph`. Since there isn't a data set like this available, it was necessary to create a data scraper to aggregate these ablum features using 3rd-party API's and data sets.

To gather these dissparate data sources, a custom Node.js API platform was built that saved raw data responses from APIs into a MongoDB, and a simple script for converting and gathering them into a well-formatted CSV file. Additional criteria for scraping artist albums were that they must be:

- The initial studio release (no singles, EPs, Live/Tour, Greatest Hits/Best of, Soundtracks, Remixes, Remasters, etc.)
- A CD or LP (to avoid duplicate entries from Tapes, file donwloads, etc.)
- Released in the United States (to avoid duplicate entries)

The scraping process was performed as follows:

- *Artist Names* initially come from my personal music collection, based on the folder names on my NAS' shared **Music** folder. However the was a strong bias towards Rock and Electronic genres. So the scraper was modified to randomly pick artist names via the Discogs API. However, this second attempt was yielding about 10% hits on actual music artits, as the data in Discogs is structured to include mixers, engineers, producers, execs, etc. as "artists". After some more searching I eventually found the "10,000 MTV Artists" list. The list ended up only containing approximately 7400 artists with their genres. I picked the top 10 genres, since the 11th fell to from several hundred artists to 53.
- Next, to obtain album information, each artist name was searched on via the Disccogs.com search API. This yielded up to the first 50 albums in an artist discography, that each contained: `year`, `title`, `genre`, `format`, and the all important `cover_image` url. This initial scrape of albums via artist name and release criteria yielded over 20,000 albums. However, I elected to sample 500 random albums from the entire album collection as resources and time where limited.

- With this randomized sample of album data eadch `cover_image` was sent through three Google Vision APIs to: find how many faces, what color content, if there were any musical objects, and whether or not the cover is an abstract illustration or photograph.

The following three sections describe how image attributes were aggregated from the Google Vision API image search results gather more variables for the album image covers:

Faces

The `faceDetection` function of Google Vision returns an object for each face found in an image, where those faces are (via coordinates for the eyebrows, eyes, and nose), and the probability of the emotion detected. Many album covers contain portraits of either a solo artist, the entiere band, or a human model. While being able to differentiate what kind of portraits/human subjects are in the photo would be useful, the album data only includes the number of faces detected in the image, for the sake of simplifing the use of `faces` as a predictor in `albumData`.

Colors

The `imageProperties` function of Google Vision returns an object that provides a percentage of each color found in the image. Each color is also scored as a percentage of the amount found in the image, and this is the value that is used to determined the most recurrent colors found in an album cover.

The colors values returned by `imageProperties` are RGB, and coverted to six-digit Hexidecimal via the custom Node.js API. However, 16 Million colors can be represented with the RGB and Hexidecimal values. It made more sense to convert this to a more managable list of factored values by converting the numerical color values into Color Names - i.e. from #FF0000 to "Red."

I found the `coloraze` npm module, which converts a Hexidecimal value to the closest color in it's pallette of 2196 color names. While this library is a massive reduction from the 16 Million colors that can be represented in Hex, I reduced the palete further to *16 web-safe colors* (See Figure 1: 16-Color Palette.) This was effective in distilling the essence of an album image's colors into a simplified color scheme and coverted the color values into factored predictors which should be enough to capture variance in album colors.

Figure 1: 16-Color Palette:

The list of colors used to reduce the range of 6-digit Hexidecimal colors from 16 Million continous values to 16 unique factors.

Captions

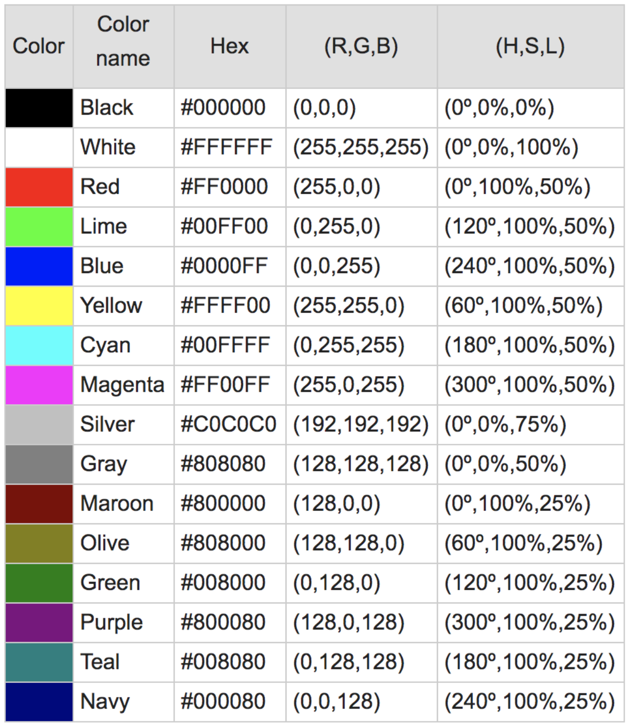
TBD - `labelDetection`

EDA

With the data scraping and aggregation completed, let's look at the raw data collected:

Machine Learning

Supervised learning is an excellent canditate for training a machine on album attributes, as we already know the `genre`, we'll train a machine with linear and logistical regressions to see which is most effective in predicting the genre of an album. Interactive terms will also be used to emphasize the color related data variables to tease out more variance.



Color	Color name	Hex	(R,G,B)	(H,S,L)
⬛	Black	#000000	(0,0,0)	(0º,0%,0%)
⬜	White	#FFFFFF	(255,255,255)	(0º,0%,100%)
🟥	Red	#FF0000	(255,0,0)	(0º,100%,50%)
🟩	Lime	#00FF00	(0,255,0)	(120º,100%,50%)
🟦	Blue	#0000FF	(0,0,255)	(240º,100%,50%)
🟨	Yellow	#FFFF00	(255,255,0)	(60º,100%,50%)
🟦	Cyan	#00FFFF	(0,255,255)	(180º,100%,50%)
🟪	Magenta	#FF00FF	(255,0,255)	(300º,100%,50%)
⬜	Silver	#C0C0C0	(192,192,192)	(0º,0%,75%)
⬛	Gray	#808080	(128,128,128)	(0º,0%,50%)
🟫	Maroon	#800000	(128,0,0)	(0º,100%,25%)
🟫	Olive	#808000	(128,128,0)	(60º,100%,25%)
🟩	Green	#008000	(0,128,0)	(120º,100%,25%)
🟪	Purple	#800080	(128,0,128)	(300º,100%,25%)
🟦	Teal	#008080	(0,128,128)	(180º,100%,25%)
🟦	Navy	#000080	(0,0,128)	(240º,100%,25%)

Figure 1: Source: https://www.rapidtables.com/web/color/color-wheel.html

Analysis