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 and title would be usefule

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 "like" 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 the `title`, 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`. Thus the data set `albumData` below:

```r
albumCovers <- read_csv('data/album-cover-data.csv')

glimpse(albumCovers)
```

```
## Observations: 865
## Variables: 11
## $ artist          <chr> "Bauhaus", "Bonobo", "Delorean", "Haxan Cloak,...
## $ year            <dbl> 2005, 2013, 2010, 2013, 1978, 1997, 2017, 2009...
## $ title           <chr> "New York City, NY 11.12.05", "The North Borde...
## $ genre           <chr> "Electronic", "Electronic", "Rock", "Electroni...
## $ faces           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0...
## $ primary_color   <chr> "White", "Gray", "White", "Black", "Black", "O...
## $ primary_score   <dbl> 0.79221308, 0.52558565, 0.53635633, 0.40122879...
## $ secondary_color <chr> "Silver", "Teal", "Silver", "Gray", "White", "...
## $ secondary_score <dbl> 0.08269589, 0.24687445, 0.27873254, 0.29843235...
## $ tertiary_color  <chr> "Black", "Gray", "Silver", "Black", "Silver", ...
## $ tertiary_score  <dbl> 0.05008276, 0.13973045, 0.13341449, 0.29201144...
```

`albumData` was scraped from several sources using a custom Node.js API that saved raw data responses into a MongoDB for gathering into well-formatted CSV:

- *Artist Names* come from my personal music collection and is based on the folder names in a Music folder.

- From *Artist Names*, a search was performed via the Disccogs.com search API: https://api.discogs.com/database/search?artist=. This yielded the first 50 albums in an artist discography, that each contained: `year`, `title`, `genre`, `format`, and the all important `cover_image` url.

- Once all the album information from my artist collection was scraped, 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 these image attributes were aggregated to create the album data from the Google Vision API image search results:

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 data Colors returned by `imageProperties` are RGB values, 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 in `albumData` which should be enough to capture variance in album colors for the size of the data set available.

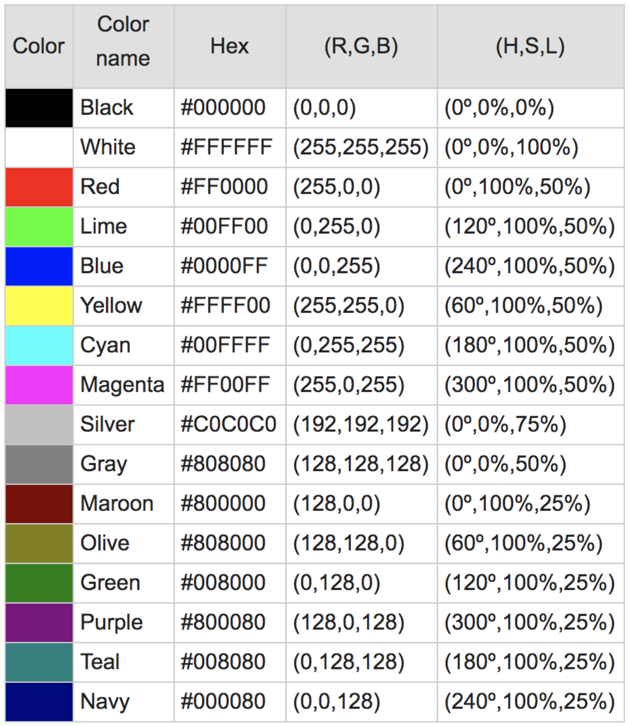
The *3 colors with the highest scores* are saved in `albumData` as: `primary_color`, `secondary_color`, and `tertiary_color`. Their scores are also saved in the similarly named: `primary_score`, `secondary_score`, and `tertiary_score` for the potential of building interactive terms in linear and logistical models if need be.

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



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

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

EDA

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.

Analysis