Predicting Song popularity using pseudo random sampling of spotify catalog

prj-pcbarko-schen176–mettler3-yc62-gianghl2

Contents

List of Figures

List of Tables

0.1 Project Abstract

Spotify is an audio streaming service used by hundreds of millions of people.The popularity of each song is quantified using a numeric popularity index. Each song is associated with metadata including genre, artist, several acoustic attributes:

- acousticness
- danceability
- durationms
- energy
- instrumentalness
- key
- liveness
- loudness
- mode
- speechiness
- tempo
- timesignature
- valence
- genre

We propose to model the popularity index based on the acoustic attributes and genre. There are several published Spotify datasets, but these are several years old. Our *fist objective* is to create a new, updated dataset of Spotify songs. We will attempt to accomplish this by generating random song IDs and using these to search the Spotify API. We anticipate this will be the most challenging aspect of the project, as Spotify does not enable bulk, random queries. Our *second objective* is to model the use song popularity (dependent variable) from acoustic attributes. Others have attempted to model popularity from the acoustic attributes and genre, but most used linear models that did not perform well. We plan to use alternative approaches to modeling/predicting song popualrity from acoustic attributes and compare them.

0.2 Sampling Techniques

0.2.1 Issues generating random sample

Randomly sampling the spotify catalog is a difficult task that is beyond the timeframe and computing power available to our cohort. Spotify randomly assigns each music track a twenty one character id. The first character is always numeric but the remaining characters are case sensitive alphanumeric characters with repitition. Thus, we have a sample space of $(62^{20})(10)$ but the actual sample space of spotify tracks is a much smaller subset of the total sample space (approximately 1.41^{-28} of the sample space). Below we implemented the stringi package to generate random id's.Unfortunately, due to the time it would take to generate a random sample this way and that the spotify API limits the number of inquiries a developer can make we would not be able to generate a sample this way.

```
spot_id_track_check <- function(x) get_tracks(x)

spot_ids <- function(Length) {

    st_int <- stri_rand_strings(1, 1, "[0-9]")
    st_char <- stri_rand_strings(as.integer(Length), 21, pattern = "[A-Za-z0-9]")
    spot_id <- seq(st_int)
    for (i in spot_id) {
        spot_id[i] <- paste(st_int[i], st_char[i], sep = "")
```

```
    }

    xt <- do.call(c, replicate(n = Length, mclapply(spot_id, function(x) {
        spot_id_track_check(x)
    }, mc.cores = 48)))
    return(xt)
}
songs <- spot_ids("1000")
songs
```

We decided to use the function *search_spotify* to return tracks from randomly generated strings.The function allows developers to search Spotify tracks by matching strings. Below are several examples of pseudo random samples using shell script, R, and Python.

0.2.2 Bash sampling script

```
# Bash script to search for random ids for Spotify tracks using the web API.
# STAT 447
# Giang Le
# Usage: bash spotify_scraping.sh $CLIENT_ID $CLIENT_SECRET
# bash spotify_scraping.sh 7c83827741ee4cbc9bd117a9cf608a39 9c2a651fdabd472892b873644f774b09

#CLIENT_ID=$1
#CLIENT_SECRET=$2

#creds=$CLIENT_ID:$CLIENT_SECRET
#encoded_creds=$(echo -n $creds | base64)

access_token=$(curl -s -X "POST" -H "Authorization: Basic $encoded_creds" -d grant_type=client_credentia
echo $access_token
rm result.json extracted_ids.txt
for x in {a..z}
do
curl --request GET --url "https://api.spotify.com/v1/search?q=%25"$x"%25&type=track&limit=50&market=US"
done

cat result.json | grep -o "\"id\" :.*" | sed 's/"id" : "//g' | sed 's/",$//g' >> extracted_ids.txt
```

0.2.3 R sampling script

```
# R script to sample tracks using spotifyr package


# function ran inside mylist function to get audio features of sampled tracks
spot_id_track_check <- function(x) get_track_audio_features(x)

mylist <- function(x) {
    xL <- c()
    st_char <- stri_rand_strings(2, x, pattern = "[A-Za-z0-9]")  ## generate random strings to sample f
    xL <- foreach(i = st_char) %do% {
        # search track returns tracks from spotify API
        search_spotify(i, type = "track", market = NULL, limit = 50, offset = 0,
            include_external = NULL, include_meta_info = FALSE)

    }
    xt <- xL
    # bind
    sp_s <- bind_rows(xt, .id = "column_label")
    x_id <- sp_s$id

    ## get audio track features
    sp_f <- spot_id_track_check(x_id)

    ## return two dataframes,track information and audio features
    return(list(sp_s, sp_f))
```

```
}
sp_feat <- mylist(x)

### Replicate function to get large dataset of spotify tracks
x <- replicate(n = 200, mclapply(3, function(x) {
    mylist(x)
}, mc.cores = 48))

# remove NA
sp_f2 <- x[!(is.na(x))]

## Beind and seperate dataframes to then reincorporate into one dataframe
sp_s1 <- bind_rows(x[1:200], .id = "column_label")
sp_s2 <- sp_s1[!(is.na(sp_s1$explicit)), ]
sp_s2 <- sp_s2[, 1:30]


sp_f2 <- sp_s1[(is.na(sp_s1$explicit)), ]
sp_f2 <- sp_f2[, 31:44]

sp_all <- cbind(sp_s2, sp_f2)

sp_s2[3, ]
sp_f2[3, ]
SP_ALL <- sp_all

######################################### Check if all track id's are unique,
######################################### if not second line removes duplicates
length(unique(SP_ALL$id))
sp_all_unique <- SP_ALL[!duplicated(SP_ALL[8]), ]

#########################################
```

0.2.4 Python sampling script

```
# Python script to sample tracks

import csv
import itertools
import string

import pandas as pd
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials


class CrawlerSpotify:
    def __init__(self, client_id, client_secret):
        # authorization information
        self.id = client_id
        self.secret = client_secret
        self.spotify = spotipy.Spotify(client_credentials_manager=
```

```
                                    SpotifyClientCredentials(client_id=self.id, client_secret=self.s

    def save_tracks(self, output_path: str = None):
        search_words = self._generate_search_words()  # generate search words

        results = []
        for sq in search_words:  # iterate search words
            search_results = self.spotify.search(q=sq, type='track', limit=50)
            total = search_results['tracks']['total']
            print(f'===== A total of {total} results from search word: {sq} =====')

            for offset in range(0, total if total < 1000 else 1000, 50):  # iterate pages.
                # for each search word, the API returns 1000 results at most,
                # so we can get maximum 20 pages for each search word (50 tracks each page)
                # see https://developer.spotify.com/documentation/web-api/reference/#/operations/search
                print(f'results from page {int(offset / 50 + 1)}')
                tracks = self._parse_tracks(self.spotify.search(q=sq, type='track', limit=50, offset=of
                audio_features = self.spotify.audio_features([track['track_id'] for track in tracks])
                result = [{**track, **self._parse_audio_features(audio_feature)} if audio_feature else
                          track, audio_feature in
                          zip(tracks, audio_features)]  # combine track details and audio features
                self._save_csv(result, output_path)  # save to csv
                results.append(result)

        return pd.DataFrame(results)

    @staticmethod
    def _save_csv(result, output_path: str):
        csv_header = ['track_id', 'track_name', 'track_artist', 'track_popularity', 'track_album_id',
                      'track_album_name', 'track_album_release_date', 'danceability', 'energy', 'key',
                      'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness'
                      'valence', 'tempo', 'duration_ms']

        with open(output_path, 'a', newline='', encoding='utf-8-sig') as fp:
            csv_writer = csv.DictWriter(fp, csv_header)
            if fp.tell() == 0:
                csv_writer.writeheader()
            csv_writer.writerows(result)

    @staticmethod
    def _parse_tracks(search_results):
        """
        extract useful data from search results
        """
        details = []
        for track in search_results['tracks']['items']:
            details.append({
                'track_id': track['id'],
                'track_name': track['name'],
                'track_artist': track['artists'][0]['name'],
                'track_popularity': track['popularity'],
                'track_album_id': track['album']['id'],
                'track_album_name': track['album']['name'],
```

```
                'track_album_release_date': track['album']['release_date']
            })
        return details

    @staticmethod
    def _parse_audio_features(audio_feature_results):
        """
        delete unnecessary data from audio feature results
        """
        del_key = ['type', 'uri', 'track_href', 'analysis_url', 'time_signature', 'id']
        return {key: audio_feature_results[key] for key in audio_feature_results if key not in del_key}

    @staticmethod
    def _generate_search_words():
        """
        generate search words. from letters a-z and numbers 0-9, with years 1985-2022
        """
        search_words = []
        for word, year in itertools.product(list(string.ascii_lowercase) + list(range(0, 10)), list(ran
            search_words.append(f'{word} year:{year}')
        return search_words


if __name__ == '__main__':
    my_client_id = 'Your Client ID'
    my_client_secret = 'Your Client Secret'
    my_output_path = r'result_spotify.csv'

    crawler = CrawlerSpotify(client_id=my_client_id, client_secret=my_client_secret)
    data_tracks = crawler.save_tracks(output_path=my_output_path)  # 1,368,207 tracks

    # delete duplicates
    data = pd.read_csv(my_output_path)
    data_final = data.drop_duplicates().reset_index(drop=True)  # 573,131 tracks
    data_final.to_csv(r'result_spotify_noduplicates.csv', index=False, encoding='utf-8-sig')
    # data_final['loudness'].isna().sum()  # 615 tracks can not extract audio features
```

0.3 Sample Summary Statistics and Exploratory Analysis

0.4 Modeling Analysis

0.4.1 Regression models

0.4.1.1 1) Random Forest

0.4.1.2 2) Gradient Boosting Machine

0.4.1.3 3) Ridge Regression

0.4.1.4 4) Lasso Regression

0.4.1.5 5) Support Vector Regression

0.4.2 Classification Models

0.4.2.1 1) Random Forest

0.4.2.2 2) Gradient Boosting Machine

0.4.2.3 3) K-Nearest Neighbors

0.4.2.4 4) Linear Discriminant Analysis

0.4.2.5 5) Support Vector Regression

0.5 Results

0.6 Shiny app?

0.7 Conclusion

0.8 References