Predicting Jay Chou Data 數據預測周杰倫

by Brian Cheng

This is a Python Jupyter notebook where I use Spotify API data and data I compiled myself to predict what a Jay Chou new song's Spotify Popularity will be.

1. Import packages

In [1]:
```
#!pip3 install spotipy
import spotipy #Spotify API
from spotipy.oauth2 import SpotifyClientCredentials #Spotify client credent
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
```

2. Compiling the data

2.1 Accessing the data from Spotify API

2.1.1 Accessing all the studio album tracks from Spotify

In [2]:
```
#Spotify API authentication token
#CHANGE IT TO YOUR OWN if you're running my code
CLIENT_ID = 'INPUT_YOUR_OWN'
CLIENT_SECRET = 'INPUT_YOUR_OWN'
#access the Spotify API
sp = spotipy.Spotify(client_credentials_manager = SpotifyClientCredentials(
    client_id = CLIENT_ID, client_secret = CLIENT_SECRET)
                     )
```

In [3]:
```
#access the playlist I made containing all Jay Chou's studio album tracks (
playlist = sp.user_playlist_tracks('b5jfei3kyhlyuiadni4zvzl84', 'spotify:pl
```

In [4]:
```
#get all the tracks from the playlist (while loop because Spotify only allo
tracks = playlist['items']
while playlist['next']:
    playlist = sp.next(playlist)
    tracks.extend(playlist['items'])
tracks_df = pd.DataFrame()
```

2.1.2 Obtaining the basic info for every track

In [5]:
```
#obtain basic info for the tracks
for track in tracks:
    track_df = pd.DataFrame(pd.DataFrame(track)['track']).transpose()
    tracks_df = tracks_df.append(track_df)
tracks_df = tracks_df.reset_index(drop = True)
tracks_df.head()
```

Out[5]:

	external_urls	href	
0	{'spotify': 'https://open.spotify.com/track/4R...	https://api.spotify.com/v1/tracks/4RbjlqzGc5pi...	4RbjlqzGc5pi
1	{'spotify': 'https://open.spotify.com/track/66...	https://api.spotify.com/v1/tracks/669LQQmGgcpQ...	669LQQmGgcp
2	{'spotify': 'https://open.spotify.com/track/4V...	https://api.spotify.com/v1/tracks/4VWn7L2kONeM...	4VWn7L2kONeMI
3	{'spotify': 'https://open.spotify.com/track/0E...	https://api.spotify.com/v1/tracks/0EEDSjk382WM...	0EEDSjk382WMF
4	{'spotify': 'https://open.spotify.com/track/0M...	https://api.spotify.com/v1/tracks/0MXi5biqp5KJ...	0MXi5biqp5KJw

2.1.3 Editing the basic info for every track

In [6]:
```
#extract release date as a column
tracks_df['release_date'] = tracks_df.apply(lambda x: x['album']['release_d
#extract album name as a column
tracks_df['album'] = tracks_df.apply(lambda x: x['album']['name'], axis = 1
```

In [7]:
```
#correct the release dates (some release date data on Spotify are incorrect
def correctReleaseDates(x):
    if x == '2002-07-30':
        return '2002-07-18'
    if x == '2003-11-01':
        return '2003-11-12'
    if x == '2006-01-16':
        return '2006-01-20'
    if x == '2007-08-10':
        return '2007-08-13'
    if x == '2008-01-29':
        return '2008-01-30'
    if x == '2008-03-10':
        return '2007-11-02'
    if x == '2008-11-09':
        return '2008-10-14'
    if x == '2010-05-20':
        return '2010-05-18'
    if x == '2011-11-16':
        return '2011-11-11'
    else:
        return x
tracks_df['release_date'] = tracks_df['release_date'].apply(correctReleaseD
```

In [8]:
```
#change available_markets to number of markets
tracks_df['available_markets'] = tracks_df['available_markets'].apply(lambd
```

In [9]:
```
#change release_date to days since
tracks_df['release_date'] = tracks_df['release_date'].apply(lambda x: (date
```

In [10]:
```
#correct the names (some names were inputted wrong on Spotify)
def correctNames(x):
    if x == '蛇 舞':
        return '蛇舞'
    if x == '花 海':
        return '花海'
    if x == '爸我回來了':
        return '爸 我回來了'
    if x == '我落淚 情緒零碎':
        return '我落淚 情緒零碎'
    else:
        return x
tracks_df['name'] = tracks_df['name'].apply(correctNames)
```

In [11]:
```
#drop unnecessary features from our basic info
tracks_df = tracks_df[['name', 'id', 'track_number', 'popularity', 'release
tracks_df['track_number'] = pd.to_numeric(tracks_df['track_number'])
tracks_df['popularity'] = pd.to_numeric(tracks_df['popularity'])
tracks_df.head()
```

Out[11]:

	name	id	track_number	popularity	release_date
0	可愛女人	4RbjlqzGc5piUTTr32XMlv	1	50	7473
1	完美主義	669LQQmGgcpQqfbliioj5qR	2	33	7473
2	星晴	4VWn7L2kONeMEQ6cAIfzXC	3	52	7473
3	娘子	0EEDSjk382WMP9PKsYPKIZ	4	33	7473
4	鬥牛	0MXi5biqp5KJw2wHh89o03	5	32	7473

2.1.4 Obtaining the audio features for every track

In [12]:
```
#audio features for the tracks
audio_features_df = pd.DataFrame()
for index, row in tracks_df.iterrows():
    audio_feature = pd.DataFrame(sp.audio_features('spotify:track:' + row['
    audio_features_df = audio_features_df.append(audio_feature)
audio_features_df = audio_features_df.reset_index(drop = True)
audio_features_df.head()
```

Out[12]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	livenes
0	0.767	0.671	5	-9.995	1	0.0317	0.552	0.000032	0.20
1	0.658	0.605	2	-9.036	0	0.1740	0.627	0.000000	0.34
2	0.637	0.329	7	-10.366	1	0.0349	0.725	0.000009	0.24
3	0.874	0.620	5	-8.700	0	0.0555	0.306	0.000000	0.11
4	0.844	0.645	6	-9.175	0	0.0770	0.437	0.000027	0.37

2.1.5 Editing the audio features for every track

In [13]:
```
#drop unnecessary audio features
audio_features_df = audio_features_df.drop(columns = ['analysis_url', 'live
audio_features_df.head()
```

Out[13]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	valenc
0	0.767	0.671	5	-9.995	1	0.0317	0.552	0.000032	0.92
1	0.658	0.605	2	-9.036	0	0.1740	0.627	0.000000	0.64
2	0.637	0.329	7	-10.366	1	0.0349	0.725	0.000009	0.25
3	0.874	0.620	5	-8.700	0	0.0555	0.306	0.000000	0.83
4	0.844	0.645	6	-9.175	0	0.0770	0.437	0.000027	0.73

2.2 Import the music credits data

In [14]:
```
#read in the credits data I compiled
credits_df = pd.read_csv('Jay Chou - Main Albums.csv').drop(columns = ['yea
credits_df
```

Out[14]:

	name	lyricist
0	可愛女人	徐若瑄
1	完美主義	方文山
2	星晴	周杰倫
3	娘子	方文山
4	鬥牛	方文山
...	...	...
145	不該	方文山
146	土耳其冰淇淋	周杰倫
147	告白氣球	方文山
148	Now You See Me	方文山
149	愛情廢柴	周杰倫

150 rows × 2 columns

2.3 Combine the Spotify API data with my compiled data

In [15]:
```
#combine basic data of tracks and their audio features together
df = credits_df.merge(tracks_df, on = 'name')
df = df.merge(audio_features_df, on = 'id').drop(columns = 'id')
df
```

Out[15]:

	name	lyricist	track_number	popularity	release_date	danceability	energy	key	loudness	m
0	可愛女人	徐若瑄	1	50	7473	0.767	0.671	5	-9.995	
1	完美主義	方文山	2	33	7473	0.658	0.605	2	-9.036	
2	星晴	周杰倫	3	52	7473	0.637	0.329	7	-10.366	
3	娘子	方文山	4	33	7473	0.874	0.620	5	-8.700	
4	鬥牛	方文山	5	32	7473	0.844	0.645	6	-9.175	
...	...	...	...	...	...	...	...	...	...	
145	不該	方文山	6	57	1764	0.441	0.619	4	-6.609	
146	土耳其冰淇淋	周杰倫	7	31	1764	0.662	0.924	7	-6.473	
147	告白氣球	方文山	8	63	1764	0.590	0.572	11	-7.658	
148	Now You See Me	方文山	9	34	1764	0.566	0.851	5	-5.549	
149	愛情廢柴	周杰倫	10	48	1764	0.411	0.671	2	-6.207	

150 rows × 17 columns

In [16]:
```
#one-hot encode lyricists
one_hot_lyricist = pd.get_dummies(df['lyricist'])
df = df.drop('lyricist', axis = 1)
df = df.join(one_hot_lyricist)
df
```

Out[16]:

	name	track_number	popularity	release_date	danceability	energy	key	loudness	mode	spe
0	可愛女人	1	50	7473	0.767	0.671	5	-9.995	1	
1	完美主義	2	33	7473	0.658	0.605	2	-9.036	0	
2	星晴	3	52	7473	0.637	0.329	7	-10.366	1	
3	娘子	4	33	7473	0.874	0.620	5	-8.700	0	
4	鬥牛	5	32	7473	0.844	0.645	6	-9.175	0	
...	...	...	...	...	...	...	...	...	...	...
145	不該	6	57	1764	0.441	0.619	4	-6.609	1	
146	土耳其冰淇淋	7	31	1764	0.662	0.924	7	-6.473	1	
147	告白氣球	8	63	1764	0.590	0.572	11	-7.658	1	
148	Now You See Me	9	34	1764	0.566	0.851	5	-5.549	0	
149	愛情廢柴	10	48	1764	0.411	0.671	2	-6.207	1	

150 rows × 30 columns

3. Testing out the model

Test and train a model on available data first to see if what I'm doing is doable.

In [17]:
```
#split 60% of data as training and 40% of data as testing
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns = ['nam
```

In [18]:
```
#random forest
rfr = RandomForestRegressor()
rfr.fit(StandardScaler().fit_transform(X_train), y_train)
```

Out[18]:
```
RandomForestRegressor()
```

In [19]:
```
#gradient boosting
gbr = GradientBoostingRegressor()
gbr.fit(StandardScaler().fit_transform(X_train), y_train)
```

Out[19]:
```
GradientBoostingRegressor()
```

In [20]:
```
#predictions from random forest
predictions1 = rfr.predict(StandardScaler().fit_transform(X_test))
```

In [21]:
```
#predictions from gradient boosting
predictions2 = gbr.predict(StandardScaler().fit_transform(X_test))
```

In [22]:
```
#r-squared for random forest
score1 = rfr.score(StandardScaler().fit_transform(X_test), y_test)
score1
```

Out[22]:
```
0.4110572845868248
```

In [23]:
```
#r-squared for gradient boosting
score2 = gbr.score(StandardScaler().fit_transform(X_test), y_test)
score2
```

Out[23]:
```
0.30231981313076206
```

In [24]:
```
#average the predictions
predictions = (predictions1 + predictions2) / 2
```

In [25]:
```
#how off the model is on average
mean_squared_error(y_test, predictions) ** 0.5
```

Out[25]:
```
7.817591704284327
```

4. Prepare the new song we want to predict

4.1 Get the song

In [26]:
```
#Spotify URI for the song we are predicting for
songURL = 'spotify:track:7BYMjh3vWnHU4IFyfZdIDO'
```

4.2 Get the track info

In [27]:
```
#get track info of the song
track_df = pd.Series(sp.track(songURL)).to_frame().T
track_df
```

Out[27]:

	album	artists	available_markets	disc_number	duration_ms	explicit	externa
0	{'album_type': 'single', 'artists': [{'externa...	[{'external_urls': {'spotify': 'https://open.s...	[AD, AE, AG, AL, AM, AO, AR, AT, AU, AZ, BA, B...	1	185008	False	{ 'TWK972001

4.3 Edit the track info

In [28]:
```
#extract release date as a column
track_df['release_date'] = track_df.apply(lambda x: x['album']['release_dat
#change release_date to days since
track_df['release_date'] = track_df['release_date'].apply(lambda x: (dateti
track_df = track_df[['name', 'id', 'track_number', 'popularity', 'release_d
track_df
```

Out[28]:

	name	id	track_number	popularity	release_date
0	Mojito	7BYMjh3vWnHU4IFyfZdIDO	1	58	315

4.4 Get the audio features

In [29]:
```
#get the audio features of the song
audio_features_track_df = pd.DataFrame(sp.audio_features(songURL)).drop(col
audio_features_track_df
```

Out[29]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	valenc
0	0.814	0.595	0	-7.726	1	0.0277	0.476	0	0.

4.5 Merge the track info and audio features

In [30]:
```
#merge track info and audio features
track_df = track_df.merge(audio_features_track_df, on = 'id').drop(columns
track_df
```

Out[30]:

	name	track_number	popularity	release_date	danceability	energy	key	loudness	mode	spee
0	Mojito	1	58	315	0.814	0.595	0	-7.726	1	

4.6 Manually add the lyricist

In [31]:
```
#manually input the lyricist for the new song to be predicted
TRACK_LYRICIST = '黃俊郎'
```

In [32]:
```
#add lyricist data to the track's data
for column in df.columns:
    if column not in track_df.columns:
        track_df[column] = 0
track_df[TRACK_LYRICIST] = 1
track_df
```

Out[32]:

	name	track_number	popularity	release_date	danceability	energy	key	loudness	mode	spee
0	Mojito	1	58	315	0.814	0.595	0	-7.726	1	

1 rows × 30 columns

5. Predicting the new song's popularity

I am using all the data available to build the model this time and using boosting with 2 random forest regressors and 2 gradient boosting regressors.

In [33]:
```
#random forest 1
rfr1 = RandomForestRegressor()
rfr1.fit(StandardScaler().fit_transform(df.drop(columns = ['name', 'popular
```

Out[33]: RandomForestRegressor()

In [34]:
```
#random forest 2
rfr2 = RandomForestRegressor()
rfr2.fit(StandardScaler().fit_transform(df.drop(columns = ['name', 'popular
```

Out[34]: RandomForestRegressor()

In [35]:
```
#gradient boosting 1
gbr1 = GradientBoostingRegressor()
gbr1.fit(StandardScaler().fit_transform(df.drop(columns = ['name', 'popular
```

Out[35]: GradientBoostingRegressor()

In [36]:
```
#gradient boosting 2
gbr2 = GradientBoostingRegressor()
gbr2.fit(StandardScaler().fit_transform(df.drop(columns = ['name', 'popular
```

Out[36]: GradientBoostingRegressor()

In [37]:
```
#predictions from random forest 1
predictions1 = rfr1.predict(StandardScaler().fit_transform(track_df.drop(co
```

In [38]:
```
#predictions from random forest 2
predictions2 = rfr2.predict(StandardScaler().fit_transform(track_df.drop(co
```

In [39]:
```
#predictions from gradient boosting 1
predictions3 = gbr1.predict(StandardScaler().fit_transform(track_df.drop(co
```

In [40]:
```
#predictions from gradient boosting 2
predictions4 = gbr2.predict(StandardScaler().fit_transform(track_df.drop(co
```

In [41]:
```
#average the predictions
PREDICTION_FINAL = ((predictions1 + predictions2 + predictions3 + predictio
PREDICTION_FINAL
```

Out[41]: 42.21946230104162

In [42]:
```
#check what songs are similar to this predicted value
df[df['popularity'].between(PREDICTION_FINAL - 2, PREDICTION_FINAL + 2)]
```

Out[42]:

	name	track_number	popularity	release_date	danceability	energy	key	loudness	mode	spe
6	伊斯坦堡	7	41	7473	0.633	0.479	2	-10.148	1	
11	爸 我回來了	2	44	7161	0.676	0.639	3	-7.008	0	
17	威廉古堡	8	42	7161	0.634	0.554	11	-6.572	0	
18	雙截棍	9	44	7161	0.701	0.876	7	-6.499	1	
25	分裂	6	44	6854	0.637	0.438	1	-8.504	1	
29	最後的戰役	10	44	6854	0.607	0.785	1	-3.810	1	
33	三年二班	4	42	6476	0.624	0.688	5	-8.857	0	
55	四面楚歌	5	44	5652	0.741	0.843	0	-3.260	0	
59	麥芽糖	9	43	5652	0.617	0.815	9	-4.553	1	
91	時光機	9	43	4574	0.450	0.626	0	-7.435	1	
98	好久不見	5	43	3993	0.759	0.537	0	-7.471	1	
101	我落淚 情緒零碎	8	41	3993	0.557	0.668	11	-8.510	1	
102	愛的飛行日記	9	44	3993	0.639	0.791	8	-5.968	1	
109	你好嗎	5	43	3451	0.281	0.284	8	-11.452	1	
120	傻笑	5	43	3038	0.642	0.689	5	-6.446	0	
131	天涯過客	4	44	2310	0.505	0.715	2	-6.932	1	

16 rows × 30 columns

In [ ]: