Abstract

The objective is to predict song popularity as part of a regression task using song characteristics data collected by Spotify such as loudness and tempo.

Similarly to linear regression, Principal Linear Regression (PCR) seeks to model the relationship between a target variable (song popularity - 'track_pop') and predictor variables. The key distinction is that principal components will serve as the predictor variables instead of the original song features from the dataset.

Introduction

To perform this regression analysis, we will:

Apply PCA to generate principal components from the predictor variables, with the number of principal components matching the number of original features p.

A. Note: Of the 544 original features, 520 are dummy variables comprised of the different genre categories that a song may be classified under.

B. Of the remaining 24 original features, 12 are dummy variables comprised of the different pitches describing a song's psychoacoustic attributes, based on standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on.

C. Of the remaining 12 original features, 2 are dummy variables comprised of the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.

D. This leaves 10 original features (i.e. danceability energy loudness speechiness acousticness instrumentalness liveness valence tempo artist_pop) more uniquely salient to a given track.

Based on preliminary research and domain knowledge, we feel these features will have a more significant impact as predictor variables than the aforementioned dummy variables. For example, genres such as ska and opera-metal are considered very niche and will probably not have much of an effect on the popularity of a song, since popular songs are not known to fall into those genres.

Keep the first k principal components that explain most of the variance (where k < p), where k is determined by cross-validation

Fit a linear regression model (using ordinary least squares) on these k principal components

Put simply, we want a number of principal components that represent most of the variability in the data and, therefore by theoretic extension, most of the relationship with the target variable.

For purposes of this project, we used the Spotify Data to create a principal component regression model.The dataset used for this analysis will be explained later in this report, but some helpful packages that were used to ensure the imported data would be clean for the algorithm to learn from a standardized data set. The following is not an exhaustive list, but some of the most useful:

1. sklearn.preprocessing

- The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

1. sklearn.model_selection import KFold, cross_val_score, train_test_split

- Split arrays or matrices into random train and test subsets

1. sklearn.decomposition import PCA

- Solves a dictionary learning matrix factorization problem. Finds the best dictionary and the corresponding sparse code for approximating the data matrix X

1. sklearn.linear_model import LinearRegression, Ridge, RidgeCV, Lasso, LassoCV
2. sklearn.metrics import mean_squared_error

- **Calculated the mean squared error regression loss**

(source: https://docs.w3cub.com/scikit_learn/)

In [ ]:

```
pip install spotipy
```

```
Requirement already satisfied: spotipy in /Users/jinmin/opt/anaconda3/lib/python3.9/site-
packages (2.20.0)
Requirement already satisfied: six>=1.15.0 in /Users/jinmin/opt/anaconda3/lib/python3.9/s
ite-packages (from spotipy) (1.16.0)
Requirement already satisfied: requests>=2.25.0 in /Users/jinmin/opt/anaconda3/lib/python
3.9/site-packages (from spotipy) (2.27.1)
Requirement already satisfied: urllib3>=1.26.0 in /Users/jinmin/opt/anaconda3/lib/python3
.9/site-packages (from spotipy) (1.26.9)
Requirement already satisfied: redis>=3.5.3 in /Users/jinmin/opt/anaconda3/lib/python3.9/
site-packages (from spotipy) (4.3.4)
Requirement already satisfied: packaging>=20.4 in /Users/jinmin/opt/anaconda3/lib/python3
.9/site-packages (from redis>=3.5.3->spotipy) (21.3)
Requirement already satisfied: async-timeout>=4.0.2 in /Users/jinmin/opt/anaconda3/lib/py
thon3.9/site-packages (from redis>=3.5.3->spotipy) (4.0.2)
Requirement already satisfied: deprecated>=1.2.3 in /Users/jinmin/opt/anaconda3/lib/pytho
n3.9/site-packages (from redis>=3.5.3->spotipy) (1.2.13)
Requirement already satisfied: wrapt<2,>=1.10 in /Users/jinmin/opt/anaconda3/lib/python3.
9/site-packages (from deprecated>=1.2.3->redis>=3.5.3->spotipy) (1.12.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /Users/jinmin/opt/anaconda3/li
b/python3.9/site-packages (from packaging>=20.4->redis>=3.5.3->spotipy) (3.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/jinmin/opt/anaconda3/lib/python3.9/
site-packages (from requests>=2.25.0->spotipy) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in /Users/jinmin/opt/anaconda3/lib/pyth
on3.9/site-packages (from requests>=2.25.0->spotipy) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in /Users/jinmin/opt/anaconda3/l
ib/python3.9/site-packages (from requests>=2.25.0->spotipy) (2.0.4)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import re
```

Downloading the data

To initially access data from Spotify API (Application Programming Interface), client ID and client secret authentication was required to access Spotify data. We created Spotify for developers account (no Spotify premium membership needed). Link -> https://developer.spotify.com/

In [ ]:

```
cid = '6f2f83e3a35040a2a5d312d897b1be1f'
secret = '3c763c21e7394e549a3c372d4b0903ad'
client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secret
)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

Accessing API

From API we wanted to access below column details from audio_featrues, artist and track databases to create a customized dataframe for our analysis: Music artist (artist id)

- **Music artist (artist id)**
- **Artist popularity (popularity from artist database)**
- **Artist genre (genres)**
- **Track popularity (popularity from track database)**

In [ ]:

```
### Script to extract Spotify API data here: https://github.com/enjuichang/PracticalDataS
cience-ENCA.git

def ari_to_features(ari):

    #Audio features
    features = sp.audio_features(ari)[0]

    #Artist of the track, for genres and popularity
    artist = sp.track(ari)["artists"][0]["id"]
    artist_pop = sp.artist(artist)["popularity"]
    artist_genres = sp.artist(artist)["genres"]

    #Track popularity
    track_pop = sp.track(ari)["popularity"]

    #Add in extra features
    features["artist_pop"] = artist_pop
    if artist_genres:
        features["genres"] = " ".join([re.sub(' ','_',i) for i in artist_genres])
    else:
        features["genres"] = "unknown"
    features["track_pop"] = track_pop

    return features

#if __name__ == "__main__":
    # Debug
    #result = ari_to_features("1o0nAjgZwMDK9TI4TTUSNn")
    #print(result)
```

In [ ]:

```
import os
from pprint import pprint
import json
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_extraction.text import TfidfVectorizer
#from textblob import TextBlob
#import re
from tqdm import tqdm
```

We then used the below script to transform raw json files from the dataset into a single panda dataframe and exported into excel, by using pandas json_normalize() function.

In [ ]:

```
#Establish path to files
directory = os.fsencode('/Users/jinmin/Desktop/MSDS/MA 544/Final Project/Sample_3_playlis
ts_test data')

#Import json files
for file in os.listdir(directory):
    filename = os.fsdecode(file)
    df = pd.DataFrame(columns=[u'pos',u'artist_name',u'track_uri',u'artist_uri',u'track_
name',u'album_uri',u'duration_ms',u'album_name',u'name'], dtype='object')
    if filename.endswith(".json"):
        raw_json = json.loads(open(filename).read())
        playlists = raw_json["playlists"]
        #transform data, add onto existing panda dataframe
        df = pd.concat([df,pd.json_normalize(playlists, record_path='tracks', meta=['nam
e'])],axis = 0)
        # Output
        df.to_excel("raw_data.xlsx")
        continue
    else:
        continue
```

Created initial dataFrame of songs from different playlists with below columns

Created initial dataFrame of songs from different playlists with below columns:

- Unnamed (later dropped for our analysis)
- Song position (pos)
- Artist name
- Track uri
- Artist uri
- Track name
- Album uri
- Duration of song (in milliseconds)
- Album name
- Playlist name

Examples of features picked up from audio_features database are shown in third cell below.

In [ ]:

```python
#Load the raw_data from the repo
dataPath = 'raw_data.xlsx'
df = pd.read_excel(dataPath)
df.head()
```

Out[ ]:

	Unnamed: 0	pos	artist_name	track_uri	artist_uri	track_na
0	0	0	Missy Elliott	spotify:track:0UaMYEvWZi0ZqiDOoHU3YI	spotify:artist:2wIVse2owClT7go1WT98tk	Lo Cont (feat. Ci & Fat M Sco
1	1	1	Britney Spears	spotify:track:6I9VzXrHxO9rA9A5euc8Ak	spotify:artist:26dSoYclwsYLMAKD3tpOr4	To
2	2	2	Beyoncé	spotify:track:0WqIKmW4BTrj3eJFmnCKMv	spotify:artist:6vWDO969PvNqNYHIOW5v0m	Crazy L
3	3	3	Justin Timberlake	spotify:track:1AWQoqb9bSvzTjaLralEkT	spotify:artist:31TPClRtHm23RisEBtV3X7	Rock Y Bo
4	4	4	Shaggy	spotify:track:1lzr43nnXAijIGYnCT8M8H	spotify:artist:5EvFsr3kj42KNv97ZEnqij	It Was

In [ ]:

```python
#Edit the track-uris to a more usable format
df["track_uri"] = df["track_uri"].apply(lambda x: re.findall(r'\w+$', x)[0])
df["track_uri"]
```

Out[ ]:

```
0        0UaMYEvWZi0ZqiDOoHU3YI
1        6I9VzXrHxO9rA9A5euc8Ak
2        0WqIKmW4BTrj3eJFmnCKMv
3        1AWQoqb9bSvzTjaLralEkT
4        1lzr43nnXAijIGYnCT8M8H
                  ...
67498    5uCax9HTNlzGybIStD3vDh
67499    0P1oO2gREMYUCoOkzYAyFu
67500    2oM4BuruDnEvk59IvIXCwn
67501    4Ri5TTUgjM96tbQZd5Ua7V
67502    5RVuBrXVLptAEbGJdSDzL5
Name: track_uri, Length: 67503, dtype: object
```

In [ ]:

```python
#Test the feature extraction script, and display features
```

```
ari_to_features(df["track_uri"][0])
```

Out[ ]:

```
{'danceability': 0.904,
 'energy': 0.813,
 'key': 4,
 'loudness': -7.105,
 'mode': 0,
 'speechiness': 0.121,
 'acousticness': 0.0311,
 'instrumentalness': 0.00697,
 'liveness': 0.0471,
 'valence': 0.81,
 'tempo': 125.461,
 'type': 'audio_features',
 'id': '0UaMYEvWZi0ZqiDOoHU3YI',
 'uri': 'spotify:track:0UaMYEvWZi0ZqiDOoHU3YI',
 'track_href': 'https://api.spotify.com/v1/tracks/0UaMYEvWZi0ZqiDOoHU3YI',
 'analysis_url': 'https://api.spotify.com/v1/audio-analysis/0UaMYEvWZi0ZqiDOoHU3YI',
 'duration_ms': 226864,
 'time_signature': 4,
 'artist_pop': 69,
 'genres': 'dance_pop hip_hop hip_pop pop pop_rap r&b rap urban_contemporary virginia_hip
_hop',
 'track_pop': 67}
```

In [ ]:

```
#Below here, we extract features from each track using the Spotify API and the associated
URI
dataLIST = df["track_uri"].unique()
print(dataLIST)
```

```
['0UaMYEvWZi0ZqiDOoHU3YI' '6I9VzXrHxO9rA9A5euc8Ak'
 '0WqIKmW4BTrj3eJFmnCKMv' ... '2oM4BuruDnEvk59IvIXCwn'
 '4Ri5TTUgjM96tbQZd5Ua7V' '5RVuBrXVLptAEbGJdSDzL5']
```

The below process allows us to access millions of songs from playlists in Spotify API. For reduced scope of the project, we randomly pulled 1000 songs from the dataframe to conduct analysis.

In [ ]:

```
featureLIST = []

for i in tqdm([uri for uri in dataLIST[0:1000]]):
    try:
        featureLIST.append(ari_to_features(i))
    except:
        continue
```

```
100%|██████████████████████████████████████████| 1000/1000 [10:12<00:00,  1.63it/s]
```

In [ ]:

```
#Preview the DataFrame
featureDF = pd.DataFrame(featureLIST)
tqdm(featureDF)
```

```
  0%|                                                          | 0/1000 [00:00<?, ?it/s]
```

Out[ ]:

```
<tqdm.std.tqdm at 0x7fbabf9f2c70>
```

In [ ]:

```
playlistDF = pd.merge(df,featureDF, left_on = "track_uri", right_on= "id")
#To export to excel if needed: playlistDF.to_excel('../data/processed_data.xlsx')
```

In [ ]:

```
playlistDF.drop(columns=["Unnamed: 0"], inplace = True)
playlistDF.head()
```

Out[ ]:

	pos	artist_name	track_uri	artist_uri	track_name	
0	0	Missy Elliott	0UaMYEvWZi0ZqiDOoHU3YI	spotify:artist:2wIVse2owClT7go1WT98tk	Lose Control (feat. Ciara & Fat Man Scoop)	spotify:album:6vV5UrXcf
1	73	Missy Elliott	0UaMYEvWZi0ZqiDOoHU3YI	spotify:artist:2wIVse2owClT7go1WT98tk	Lose Control (feat. Ciara & Fat Man Scoop)	spotify:album:6vV5UrXcf
2	14	Missy Elliott	0UaMYEvWZi0ZqiDOoHU3YI	spotify:artist:2wIVse2owClT7go1WT98tk	Lose Control (feat. Ciara & Fat Man Scoop)	spotify:album:6vV5UrXcf
3	42	Missy Elliott	0UaMYEvWZi0ZqiDOoHU3YI	spotify:artist:2wIVse2owClT7go1WT98tk	Lose Control (feat. Ciara & Fat Man Scoop)	spotify:album:6vV5UrXcf
4	1	Missy Elliott	0UaMYEvWZi0ZqiDOoHU3YI	spotify:artist:2wIVse2owClT7go1WT98tk	Lose Control (feat. Ciara & Fat Man Scoop)	spotify:album:6vV5UrXcf

5 rows × 30 columns

The following cells conduct further preprocessing from imported dataset to cater the data specifically for the content-based filtering.

Here is the general pipeline:

- **Useful data Selection**
- **List concatenation**

Due to the nature of playlist, there will be duplicates in songs across multiple playlists. Therefore, to scale down essential columns for base dataframe, song and artist column features were extracted and we used the drop_duplicates() function in pandas to remove duplicate songs when building the base dataframe with all unique songs.

In [ ]:

```
# Drop song duplicates
def drop_duplicates(df):
    '''
    Drop duplicate songs
    '''
    df['artists_song'] = df.apply(lambda row: row['artist_name']+row['track_name'],axis
= 1)
    return df.drop_duplicates('artists_song')

songDF = drop_duplicates(playlistDF)
print("Are all songs unique: ",len(pd.unique(songDF.artists_song))==len(songDF))
```

```
Are all songs unique:  True
```

In [ ]:

```
# Select useful columns
def select_cols(df):
        '''
        Select useful columns
        '''
        return df[['artist_name','id','track_name','danceability', 'energy', 'key', 'loud
ness', 'mode',
        'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo'
, "artist_pop", "genres", "track_pop"]]
songDF = select_cols(songDF)
songDF.head()
```

Out[ ]:

	artist_name	id	track_name	danceability	energy	key	loudness	mode	speechiness	acoustic
0	Missy Elliott	0UaMYEvWZi0ZqiDOoHU3YI	Lose Control (feat. Ciara & Fat Man Scoop)	0.904	0.813	4	-7.105	0	0.1210	0.0
6	Britney Spears	6I9VzXrHxO9rA9A5euc8Ak	Toxic	0.774	0.838	5	-3.914	0	0.1140	0.0
19	Beyoncé	0WqIKmW4BTrj3eJFmnCKMv	Crazy In Love	0.664	0.758	2	-6.583	0	0.2100	0.0
46	Justin Timberlake	1AWQoqb9bSvzTjaLralEkT	Rock Your Body	0.892	0.714	4	-6.055	0	0.1410	0.2
55	Shaggy	1lzr43nnXAijIGYnCT8M8H	It Wasn't Me	0.853	0.606	0	-4.596	1	0.0713	0.0

In [ ]:

```
def genre_preprocess(df):
    '''
    Preprocess the genre data
    '''
    df['genres_list'] = df['genres'].apply(lambda x: x.split(" "))
    return df
songDF = genre_preprocess(songDF)
songDF['genres_list'].head()
```

Out[ ]:

```
0     [dance_pop, hip_hop, hip_pop, pop, pop_rap, r&...
6                        [dance_pop, pop, post-teen_pop]
19                                 [dance_pop, pop, r&b]
46                                      [dance_pop, pop]
55                               [pop_rap, reggae_fusion]
Name: genres_list, dtype: object
```

In [ ]:

```
# pipeline for preprocessing any new playlist as below:

def playlist_preprocess(df):
    '''
    Preprocess imported playlist
    '''
    df = drop_duplicates(df)
    df = select_cols(df)
    df = genre_preprocess(df)

    return df
```

Feature Generation

Feature Generation

1. One-hot encoding One-hot encoding is a method to transform categorical variables into a machine-understandable langauge. This is done by converting each category into a column so that each category can be represented as either True (1) or False (0).
2. TF-IDF TF-IDF, also known as Term Frequency-Inverse Document Frequency, is a tool to quantify words in a set of documents. The goal of TF-IDF is to show the importance of a word in the documents and the corpus. The general formula for calculating TF-IDF is:

Term Frequency (TF): The number of times a term appears in each document divided by the total word count in the document. Inverse Document Frequency (IDF): The log value of the document frequency. Document frequency is the total number of documents where one term is present. The motivation is to find words that are not only important in each document but also accounting for the entire corpus. The log value was taken to decrease the impact of a large N, which would lead a very large IDF compared to TF. TF is focused on importance of a word in a document, while IDF is focused on the importance of a word across documents.

In this project, the documents ~ songs. Calculating the most prominent genre in each song and their prevalence across songs allows us to determine the weight of the genre. This is to prevent overweighting on uncommon genres.

1. Normalization Popularity variables are not normalized to 0 to 1, which would be problematic in the consine similarity function later on. In addition, the audio features are also not normalized. The MinMaxScaler() function from scikit learn allows us to scale all values from the min and max into a range of 0 to 1.

In [ ]:

```
def ohe_prep(df, column, new_name):
    '''
    Create One Hot Encoded features of a specific column
    ---
    Input:
    df (pandas dataframe): Spotify Dataframe
    column (str): Column to be processed
    new_name (str): new column name to be used

    Output:
    tf_df: One-hot encoded features
    '''
    tf_df = pd.get_dummies(df[column])
    feature_names = tf_df.columns
    tf_df.columns = [new_name + "|" + str(i) for i in feature_names]
    tf_df.reset_index(drop = True, inplace = True)
    return tf_df
```

In [ ]:

```
# TF-IDF implementation
tfidf = TfidfVectorizer()
tfidf_matrix =  tfidf.fit_transform(songDF['genres_list'].apply(lambda x: " ".join(x)))
genre_df = pd.DataFrame(tfidf_matrix.toarray())
genre_df.columns = ['genre' + "|" + i for i in tfidf.get_feature_names()]
genre_df.drop(columns='genre|unknown')
genre_df.reset_index(drop = True, inplace=True)
genre_df.iloc[0]
```

```
/Users/jinmin/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated
in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[ ]:

```
genre|_hip_hop            0.0
genre|abstract_hip_hop    0.0
genre|acoustic_pop        0.0
genre|adult_standards     0.0
genre|aesthetic_rap       0.0
                         ...
genre|wonky               0.0
genre|world_devotional    0.0
```

```
genre|world_devotional    0.0
genre|worship             0.0
genre|yacht_rock          0.0
genre|zolo                0.0
Name: 0, Length: 520, dtype: float64
```

In [ ]:

```
# artist_pop distribution descriptive stats
print(songDF['artist_pop'].describe())
# Normalization
pop = songDF[["artist_pop"]].reset_index(drop = True)
scaler = MinMaxScaler()
pop_scaled = pd.DataFrame(scaler.fit_transform(pop), columns = pop.columns)
pop_scaled.head()
```

```
count    1000.000000
mean       60.713000
std        17.089959
min         0.000000
25%        51.000000
50%        62.000000
75%        72.000000
max        95.000000
Name: artist_pop, dtype: float64
```

Out[ ]:

	artist_pop
0	0.726316
1	0.821053
2	0.936842
3	0.831579
4	0.757895

In [ ]:

```
def create_feature_set(df, float_cols):
    '''
    Process spotify df to create a final set of features that will be used to generate re
commendations
    ---
    Input:
    df (pandas dataframe): Spotify Dataframe
    float_cols (list(str)): List of float columns that will be scaled

    Output:
    final (pandas dataframe): Final set of features
    '''

    # Tfidf genre lists
    tfidf = TfidfVectorizer()
    tfidf_matrix =  tfidf.fit_transform(df['genres_list'].apply(lambda x: " ".join(x)))
    genre_df = pd.DataFrame(tfidf_matrix.toarray())
    genre_df.columns = ['genre' + "|" + i for i in tfidf.get_feature_names()]
    genre_df.drop(columns='genre|unknown') # drop unknown genre
    genre_df.reset_index(drop = True, inplace=True)

    # One-hot Encoding
    key_ohe = ohe_prep(df, 'key','key') * 0.5
    mode_ohe = ohe_prep(df, 'mode','mode') * 0.5

    # Normalization
    # Scale popularity columns
    pop = df[["artist_pop","track_pop"]].reset_index(drop = True)
    scaler = MinMaxScaler()
    pop_scaled = pd.DataFrame(scaler.fit_transform(pop), columns = pop.columns) * 0.2
```

```
    # Scale audio columns
    floats = df[float_cols].reset_index(drop = True)
    scaler = MinMaxScaler()
    floats_scaled = pd.DataFrame(scaler.fit_transform(floats), columns = floats.columns)
* 0.2

    # Concanenate all features
    final = pd.concat([genre_df, floats_scaled, pop_scaled, key_ohe, mode_ohe], axis = 1
)

    # Add song id
    final['id']=df['id'].values

    return final
```

In [ ]:

```
# Save the data and generate the features
float_cols = songDF.dtypes[songDF.dtypes == 'float64'].index.values
songDF.to_excel("allsong_data.xlsx", index = False)

# Generate features
complete_feature_set = create_feature_set(songDF, float_cols=float_cols)
complete_feature_set.to_excel("complete_feature.xlsx", index = False)
complete_feature_set.head()
```

```
/Users/jinmin/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated
in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[ ]:

	genrel_hip_hop	genrelabstract_hip_hop	genrelacoustic_pop	genreladult_standards	genrelaesthetic_rap	genrelafrofuturism
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 546 columns

In [ ]: