



建立一个音乐推荐系统



灰灰

公众号：磐创AI

已关注

作者|Amol Mavuduru

编译|VK

来源|Towards Data Science

原文链接: towardsdatascience.com/...

你有没有想过Spotify是如何根据你的收听历史来推荐歌曲和播放列表的？你想知道Spotify是如何找到听起来与你已经听过的歌曲相似的歌曲的吗？

有趣的是，Spotify有一个web api，开发者可以使用它来检索歌曲的音频特性和元数据，比如歌曲的流行度、节奏以及发布年份。我们可以利用这些数据建立音乐推荐系统，根据用户所听歌曲的音频特征和元数据向用户推荐歌曲。

在本文中，我将演示如何使用Spotify歌曲数据集和Spotify的Python客户端Spotipy构建基于内容的音乐推荐系统。

▲ 赞同 ▼ ● 添加评论 ➦ 分享 ❤ 喜欢 ★ 收藏 📄 申请转载 ...

Spotipy是spotify webapi的Python客户端，它使开发人员能够轻松地获取数据和查询Spotify的歌曲目录。

在这个项目中，我使用Spotipy来获取从Kaggle的原始Spotify歌曲数据集。你可以使用下面的命令使用pip安装Spotipy。

```
pip install spotipy
```

安装Spotipy后，你需要在Spotify开发者页面上创建一个应用程序，并保存你的客户端ID和密钥。

导入库

在下面的代码中，我导入了Spotipy和其他一些用于数据操作和可视化的基本库。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import spotipy
import os
%matplotlib inline
```

读取数据

为了建立一个音乐推荐系统，我使用了Spotify数据集，这个数据集在Kaggle上是公开的，包含了超过170000首不同歌曲的元数据和音频特性。我使用了这个数据集中的三个数据文件。第一个文件包含单个歌曲的数据，而接下来的两个文件包含按歌曲发行的类型和年份分组的数据。

```
spotify_data = pd.read_csv('./data/data.csv.zip')
genre_data = pd.read_csv('./data/data_by_genres.csv')
data_by_year = pd.read_csv('./data/data_by_year.csv')
```

我在下面包含了通过为每个数据帧调用Pandas info函数生成列元数据。

spotify_data

```
---
0  valence      170653 non-null float64
1  year         170653 non-null int64
2  acousticness 170653 non-null float64
3  artists      170653 non-null object
4  danceability 170653 non-null float64
5  duration_ms  170653 non-null int64
6  energy        170653 non-null float64
7  explicit      170653 non-null int64
8  id            170653 non-null object
9  instrumentalness 170653 non-null float64
10 key          170653 non-null int64
11 liveness      170653 non-null float64
12 loudness      170653 non-null float64
13 mode          170653 non-null int64
14 name          170653 non-null object
15 popularity    170653 non-null int64
16 release_date  170653 non-null object
17 speechiness   170653 non-null float64
18 tempo         170653 non-null float64
dtypes: float64(9), int64(6), object(4)
memory usage: 24.7+ MB
```

genre_data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2973 entries, 0 to 2972
Data columns (total 14 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   mode            2973 non-null  int64
1   genres          2973 non-null  object
2   acousticness    2973 non-null  float64
3   danceability    2973 non-null  float64
4   duration_ms     2973 non-null  float64
5   energy          2973 non-null  float64
6   instrumentalness 2973 non-null  float64
7   liveness        2973 non-null  float64
8   loudness        2973 non-null  float64
9   speechiness     2973 non-null  float64
```

```
dtypes: float64(11), int64(2), object(1)
memory usage: 325.3+ KB
```

data_by_year

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2973 entries, 0 to 2972
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   mode                  2973 non-null   int64
1   genres                2973 non-null   object
2   acousticness          2973 non-null   float64
3   danceability          2973 non-null   float64
4   duration_ms           2973 non-null   float64
5   energy                2973 non-null   float64
6   instrumentalness       2973 non-null   float64
7   liveness              2973 non-null   float64
8   loudness              2973 non-null   float64
9   speechiness           2973 non-null   float64
10  tempo                 2973 non-null   float64
11  valence               2973 non-null   float64
12  popularity            2973 non-null   float64
13  key                   2973 non-null   int64
dtypes: float64(11), int64(2), object(1)
memory usage: 325.3+ KBExploratory Data Analysis
```

根据上面的列描述，我们可以看到每个数据帧都有关于音频特征的信息，例如不同歌曲的可舞性（danceability）和响度（loudness），这些信息也在不同的流派和特定年份进行了汇总。

探索性数据分析

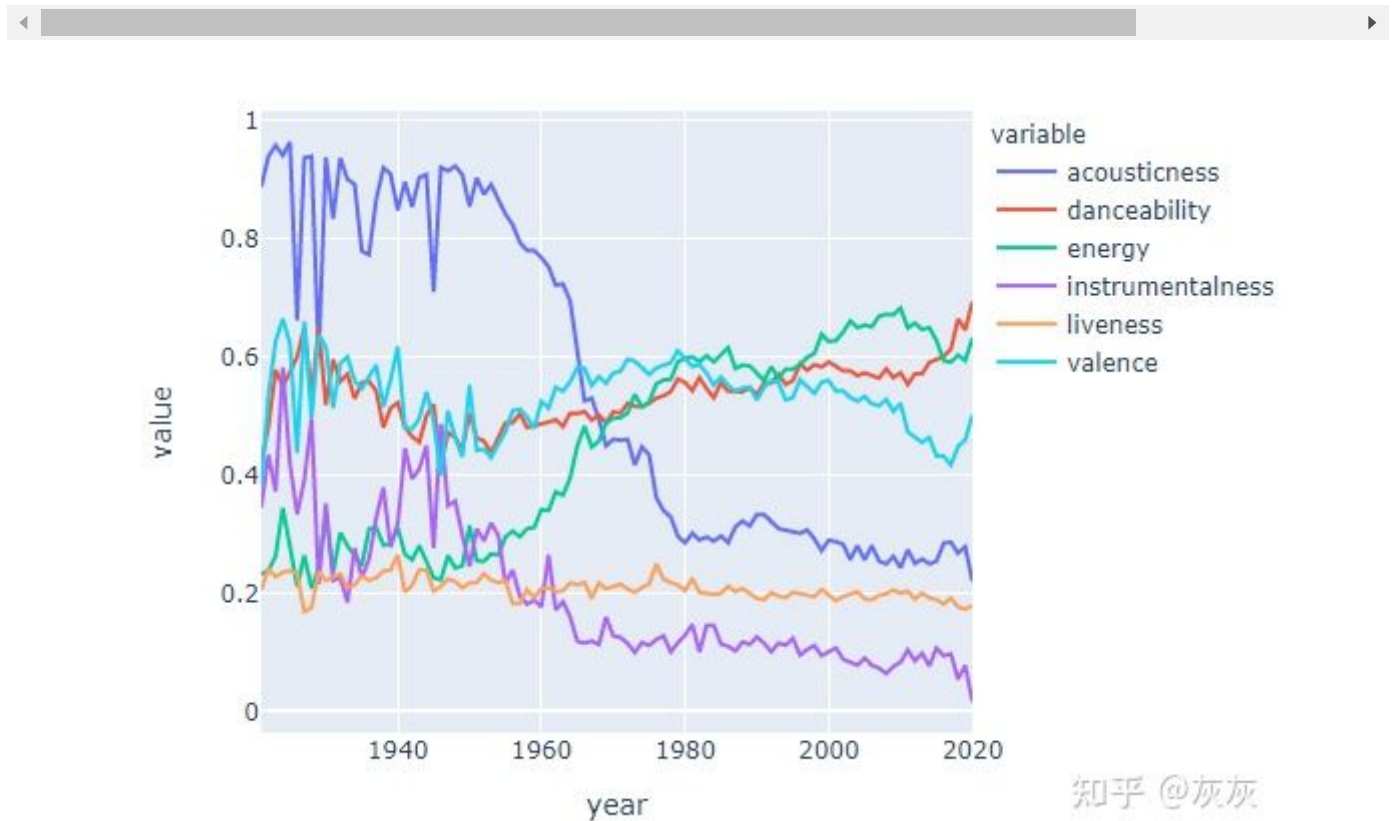
这个数据集非常有用，可以用于广泛的任務。在建立一个推荐系统之前，我决定创建一些可视化的东西，以便更好地了解过去100年来音乐的数据和趋势。

随时间推移的音乐

利用按年份分组的数据，我们可以了解从1921年到2020年，音乐的整体声音是如何变化的。在下

```
sound_features = ['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness',  
fig = px.line(data_by_year, x='year', y=sound_features)
```

```
fig.show()
```



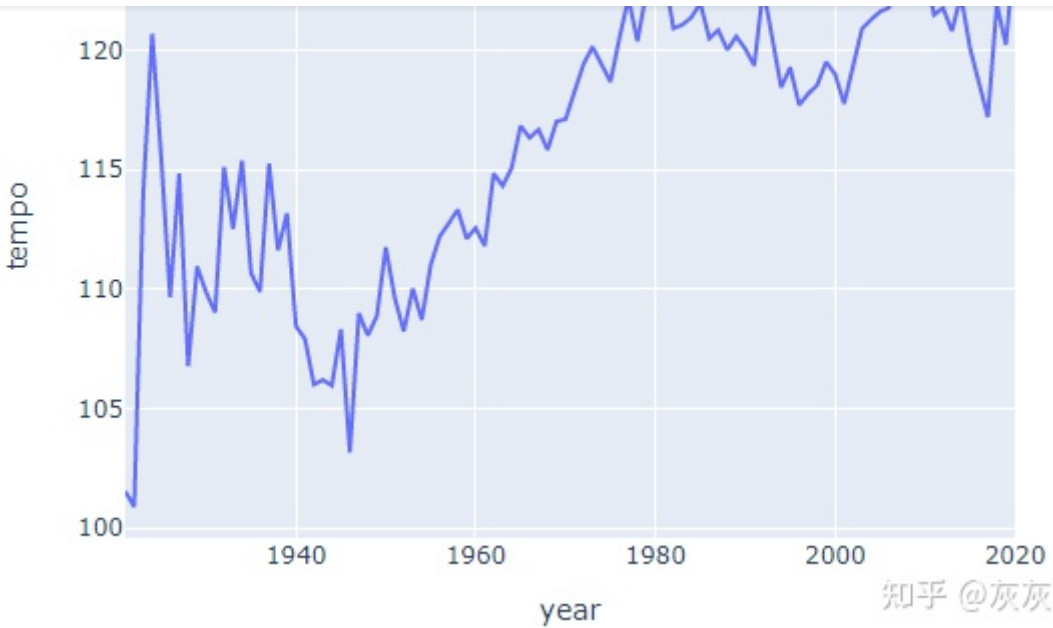
根据上面的图标，我们可以看到音乐已经从20世纪初更具声学 and 器乐性的声音过渡到21世纪更具舞蹈性和活力的声音。

20世纪20年代的大部分曲目很可能是古典和爵士乐流派的器乐作品。由于计算机和先进的音频工程技术的出现，21世纪的音乐听起来非常不同，这些技术使我们能够创造出具有广泛效果和节拍的电子音乐。

我们还可以看看这些年来音乐的平均节奏或速度是如何变化的。下面的代码所生成的图表也支持声音向电子音乐的巨大转变。

```
fig = px.line(data_by_year, x='year', y='tempo')
```

```
fig.show()
```

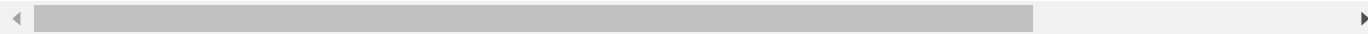


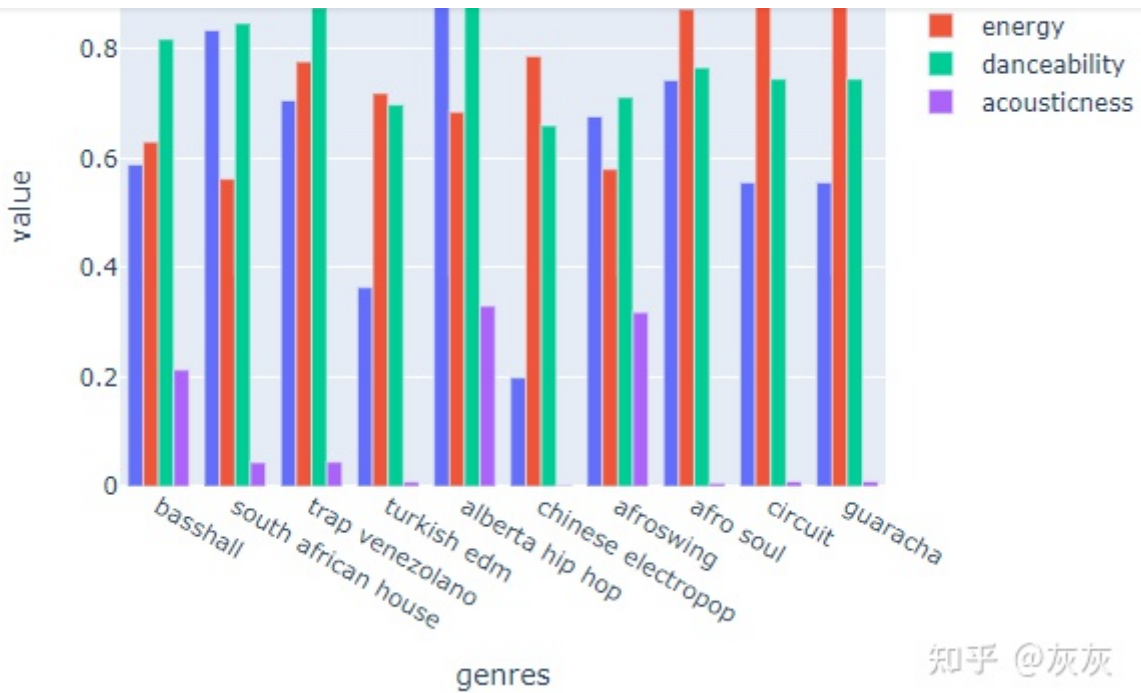
根据上面的图表，我们可以清楚地看到，音乐在上世纪的发展速度明显加快。这一趋势不仅是20世纪60年代迷幻摇滚等新流派的产物，也是音频工程技术进步的结果。

不同流派的特点

此数据集包含不同歌曲的音频功能以及不同流派的音频功能。我们可以利用这些信息来比较不同的流派，了解它们在声音上的差异。在下面的代码中，我从数据集中选择了十种最流行的流派，并为每种流派提供了可视化的音频特性。

```
top10_genres = genre_data.nlargest(10, 'popularity')
fig = px.bar(top10_genres, x='genres', y=['valence', 'energy', 'danceability', 'acoust
fig.show()
```





知乎 @灰灰

上面提到的许多流派，比如中国的电子流行乐，很可能属于一个或多个广泛的流派。我们可以利用这些高度特定的题材，根据它们的音频特征将它们进行分类，从而了解它们与其他题材的相似程度。

用K-均值聚类流派

在下面的代码中，我使用了著名而简单的K-means聚类算法，根据每种类型的数字音频特征，将这个数据集中的2900多个类型划分为10个聚类。

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_cluster

X = genre_data.select_dtypes(np.number)
cluster_pipeline.fit(X)
genre_data['cluster'] = cluster_pipeline.predict(X)
```

既然这些类型已经被分配到了簇，我们可以通过在二维空间中可视化簇来进一步进行分析。

图1. GNF可视化流派聚类

```
from sklearn.manifold import TSNE

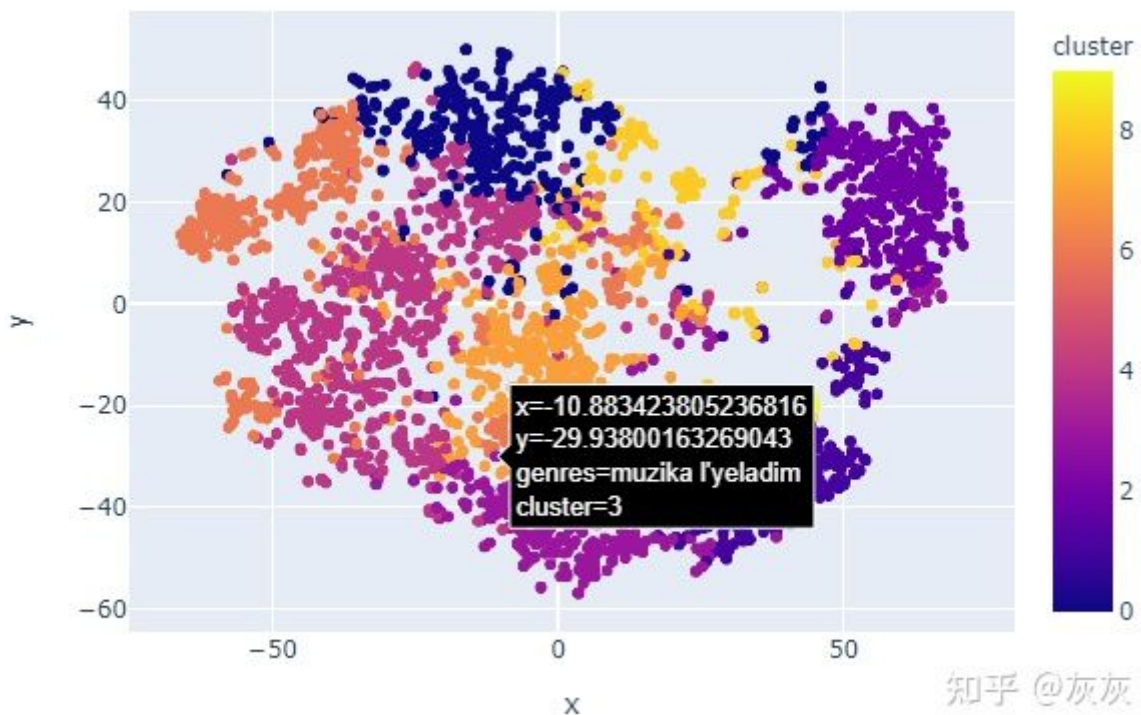
tsne_pipeline = Pipeline([('scaler', StandardScaler()), ('tsne', TSNE(n_components=2,
genre_embedding = tsne_pipeline.fit_transform(X)

projection = pd.DataFrame(columns=['x', 'y'], data=genre_embedding)
projection['genres'] = genre_data['genres']
projection['cluster'] = genre_data['cluster']
```

现在，我们可以很容易地在二维坐标平面上使用Plotly的scatter函数来可视化流派簇。

```
import plotly.express as px

fig = px.scatter(
    projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'genres'])
fig.show()
```



用K-均值聚类歌曲

我们还可以使用K-means对歌曲进行聚类，如下所示，以了解如何构建更好的推荐系统。

▲ 赞同 ▼ ● 添加评论 ➦ 分享 ❤ 喜欢 ★ 收藏 📄 申请转载 ...


```
verbose=2, n_jobs=4))], verbose=True)
```

```
X = spotify_data.select_dtypes(np.number)
number_cols = list(X.columns)
song_cluster_pipeline.fit(X)
song_cluster_labels = song_cluster_pipeline.predict(X)
spotify_data['cluster_label'] = song_cluster_labels
```

用PCA实现歌曲簇的可视化

歌曲数据帧比流派数据帧要大得多，所以我决定使用PCA进行降维，而不是t-SNE，因为它的运行速度要快得多。

```
from sklearn.decomposition import PCA

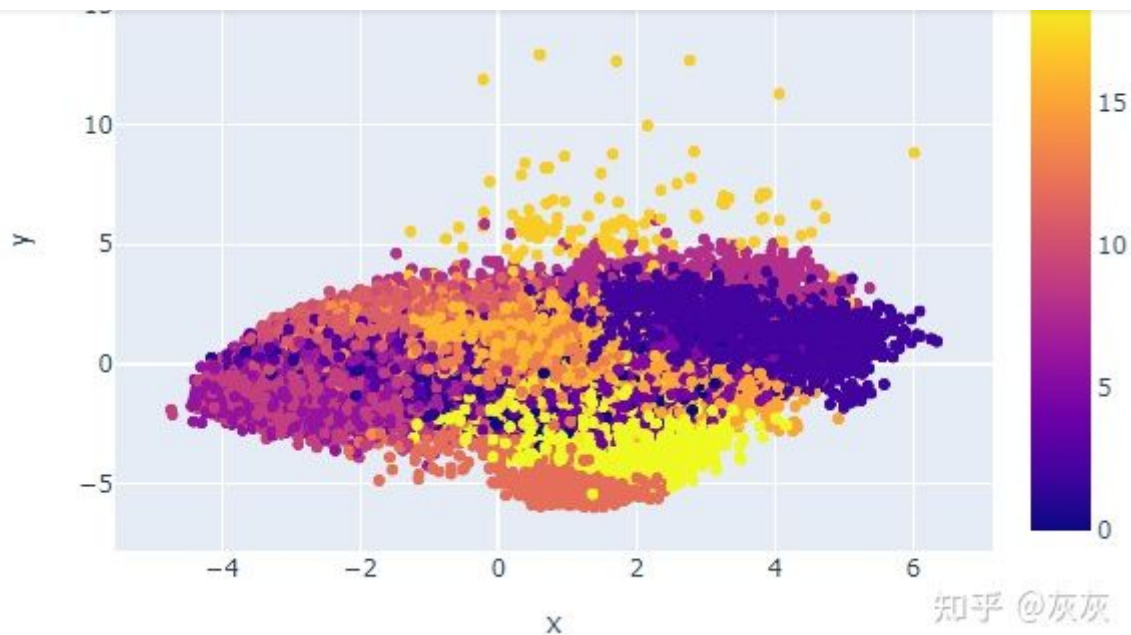
pca_pipeline = Pipeline([('scaler', StandardScaler()), ('PCA', PCA(n_components=2))])
song_embedding = pca_pipeline.fit_transform(X)

projection = pd.DataFrame(columns=['x', 'y'], data=song_embedding)
projection['title'] = spotify_data['name']
projection['cluster'] = spotify_data['cluster_label']
```

现在，我们可以使用下面的代码在二维空间中可视化歌曲簇。

```
import plotly.express as px

fig = px.scatter(projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'tit
fig.show()
```



如果你花些时间探索生成的图，你会发现相似的歌曲往往彼此靠近，而簇中的歌曲往往至少有些相似。这是我在下一节中创建的基于内容的推荐系统背后的关键思想。

基于内容的推荐系统的构建

基于上一节的分析和可视化，很明显，相似类型的歌曲往往有彼此靠近的数据点，而相似类型的歌曲也聚集在一起。

在实践层面上，这种观察是非常有意义的。相似的流派听起来相似，而这些流派中的歌曲也是如此。利用这一思想，我们可以通过提取用户所听歌曲的数据点，并推荐与附近数据点对应的歌曲来构建推荐系统。

查找不在数据集中的歌曲

在构建这个推荐系统之前，我们需要能够容纳原始Spotify歌曲数据集中不存在的歌曲。我在下面定义的find_song函数从Spotify的目录中获取给定歌曲名称和发行年份的歌曲的数据。结果以Pandas数据帧的形式返回。

```
from spotipy.oauth2 import SpotifyClientCredentials
from collections import defaultdict

sp = spotipy.Spotify(auth_manager=SpotifyClientCredentials(client_id=os.environ["SPOTI
client_secret=os.environ["S
```

这个函数返回一个数据帧，其中包含给定歌曲名称和发行年份的数据。
该函数使用Spotipy获取指定歌曲的音频特性和元数据。

```
"""  
  
song_data = defaultdict()  
results = sp.search(q= 'track: {} year: {}'.format(name,  
                                                    year), limit=1)  
  
if results['tracks']['items'] == []:  
    return None  
  
results = results['tracks']['items'][0]  
  
track_id = results['id']  
audio_features = sp.audio_features(track_id)[0]  
  
song_data['name'] = [name]  
song_data['year'] = [year]  
song_data['explicit'] = [int(results['explicit'])]  
song_data['duration_ms'] = [results['duration_ms']]  
song_data['popularity'] = [results['popularity']]  
  
for key, value in audio_features.items():  
    song_data[key] = value  
  
return pd.DataFrame(song_data)
```

有关如何使用Spotipy的详细示例，请参阅此处的文档页：[spotipy.readthedocs.io/...](https://spotipy.readthedocs.io/)

生成歌曲推荐

现在我们终于可以建立音乐推荐系统了！我使用的推荐算法非常简单，分为三个步骤：

1. 计算用户听过的每首歌的音频和元数据特征的平均向量。
2. 找到数据集中距离此平均向量最近的n个数据点。
3. 记下这n个点，推荐与之对应的歌曲。

该算法遵循一种在基于内容的推荐系统中使用的常见方法，并且是可泛化的，因为我们可以从数学

$$\text{distance}(u, v) = 1 - \frac{1}{\|u\| \|v\|} = 1 - \cos \theta$$

换句话说，余弦距离是1减去余弦相似性-两个向量之间夹角的余弦。余弦距离是推荐系统中常用的距离，即使使用的向量大小不同，也能很好地工作。

如果两首歌的向量是平行的，那么它们之间的夹角将为零，这意味着它们之间的余弦距离也将为零，因为0的余弦是1。

我在下面定义的函数借助于Scipy的cdist函数来实现这个简单的算法，用于查找两对点集合之间的距离。

```
from collections import defaultdict
from scipy.spatial.distance import cdist
import difflib

number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy',
               'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness']

def get_song_data(song, spotify_data):
    """
    获取特定歌曲的歌曲数据。song参数采用字典的形式，其中的键-值对表示歌曲的名称和发布年份。
    """

    try:
        song_data = spotify_data[(spotify_data['name'] == song['name'])
                                & (spotify_data['year'] == song['year'])].iloc[0]
        return song_data

    except IndexError:
        return find_song(song['name'], song['year'])

def get_mean_vector(song_list, spotify_data):
    """
    获取歌曲列表的平均向量。
    """
```

```
if song_data is None:
    print('Warning: {} does not exist in Spotify or in database'.format(song['
    continue
song_vector = song_data[number_cols].values
song_vectors.append(song_vector)

song_matrix = np.array(list(song_vectors))
return np.mean(song_matrix, axis=0)

def flatten_dict_list(dict_list):

    """
    用于扁平化字典列表的效用函数。
    """

    flattened_dict = defaultdict()
    for key in dict_list[0].keys():
        flattened_dict[key] = []

    for dictionary in dict_list:
        for key, value in dictionary.items():
            flattened_dict[key].append(value)

    return flattened_dict

def recommend_songs(song_list, spotify_data, n_songs=10):

    """
    根据用户之前听过的歌曲列表来推荐歌曲。
    """

    metadata_cols = ['name', 'year', 'artists']
    song_dict = flatten_dict_list(song_list)

    song_center = get_mean_vector(song_list, spotify_data)
    scaler = song_cluster_pipeline.steps[0][1]
    scaled_data = scaler.transform(spotify_data[number_cols])
    scaled_song_center = scaler.transform(song_center.reshape(1, -1))
    distances = cdist(scaled_song_center, scaled_data, 'cosine')
    index = list(np.argsort(distances)[: , :n_songs][0])
```

算法背后的逻辑听起来很有说服力，但这个推荐系统真的有效吗？找到答案的唯一方法是用实际的例子来检验它。

假设我们想推荐音乐给那些听90年代音乐的人，特别是涅槃的歌。我们可以使用推荐歌曲功能来指定他们的收听历史，并生成如下所示的推荐歌曲。

```
recommend_songs([{'name': 'Come As You Are', 'year': 1991},
                  {'name': 'Smells Like Teen Spirit', 'year': 1991},
                  {'name': 'Lithium', 'year': 1992},
                  {'name': 'All Apologies', 'year': 1993},
                  {'name': 'Stay Away', 'year': 1993}], spotify_data)
```

运行此函数将生成下面的歌曲列表。

```
[{'name': 'Life is a Highway - From "Cars"',
  'year': 2009,
  'artists': "['Rascal Flatts']"},
 {'name': 'Of Wolf And Man', 'year': 1991, 'artists': "['Metallica']"},
 {'name': 'Somebody Like You', 'year': 2002, 'artists': "['Keith Urban']"},
 {'name': 'Kayleigh', 'year': 1992, 'artists': "['Marillion']"},
 {'name': 'Little Secrets', 'year': 2009, 'artists': "['Passion Pit']"},
 {'name': 'No Excuses', 'year': 1994, 'artists': "['Alice In Chains']"},
 {'name': 'Corazón Mágico', 'year': 1995, 'artists': "['Los Fugitivos']"},
 {'name': 'If Today Was Your Last Day',
  'year': 2008,
  'artists': "['Nickelback']"},
 {'name': "Let's Get Rocked", 'year': 1992, 'artists': "['Def Leppard']"},
 {'name': "Breakfast At Tiffany's",
  'year': 1995,
  'artists': "['Deep Blue Something']"}]
```

从上面的列表中我们可以看到，推荐算法产生了上世纪90年代和21世纪的摇滚歌曲列表，列表中的乐队如Metallica、Alice in Chains和Nickelback与Nirvana相似。榜单上的第一首歌《Life is a Highway》并不是一首蹩脚的歌，但如果你仔细听的话，吉他即兴曲的节奏听起来其实和涅槃的《Smells Like Teen Spirit》很相似。

如果我们想对听过涅槃歌曲的人做同样的事情？


```
{'name': 'Thriller', 'year': 1982}], spotify_data)
```

推荐函数给出了下面的输出。

```
[{'name': 'Hot Legs', 'year': 1977, 'artists': "['Rod Stewart']"},  
{'name': 'Thriller - 2003 Edit',  
 'year': 2003,  
 'artists': "['Michael Jackson']"},  
{'name': 'I Didn't Mean To Turn You On',  
 'year': 1984,  
 'artists': "['Cherrelle']"},  
{'name': 'Stars On 45 - Original Single Version',  
 'year': 1981,  
 'artists': "['Stars On 45']"},  
{'name': 'Stars On '89 Remix - Radio Version',  
 'year': 1984,  
 'artists': "['Stars On 45']"},  
{'name': 'Take Me to the River - Live',  
 'year': 1984,  
 'artists': "['Talking Heads']"},  
{'name': 'Nothing Can Stop Us', 'year': 1992, 'artists': "['Saint Etienne']"}]
```

榜单上最受欢迎的歌曲是罗德·斯图尔特，他和迈克尔·杰克逊一样，在20世纪80年代成名。榜单上还包括2003年迈克尔·杰克逊惊悚片的剪辑，考虑到用户已经听过这首歌1982年的原版，这是有道理的。该榜单还包括80年代的流行和摇滚歌曲，比如45岁的明星。

我们可以使用更多的例子，但是这些例子应该足以说明推荐系统是如何产生歌曲推荐的。有关更完整的示例集，请查看此项目的GitHub存储库。请随意创建你自己的播放列表与代码！

github.com/AmolMavuduru...

总结

Spotify跟踪歌曲的元数据和音频功能，我们可以使用这些功能构建音乐推荐系统。在本文中，我演示了如何使用这些数据，使用余弦距离度量构建一个简单的基于内容的音乐推荐系统。像往常一样，你可以在GitHub上找到这个项目的完整代码：

github.com/AmolMavuduru...

[towardsdatascience.com/...](#)

来源

1. Y. E. Ay, Spotify Dataset 1921–2020, 160k+ Tracks, (2020), Kaggle.

2. L. van der Maaten and G. Hinton, Visualizing Data using t-SNE, (2008), Journal of Machine Learning Research.

3. P. Virtanen et. al, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, (2020), Nature Methods.

发布于 1 小时前

Spotify

数据集

推荐系统

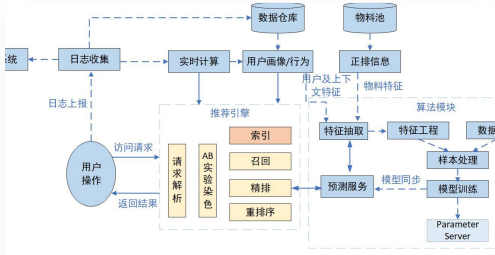
文章被以下专栏收录



磐创AI

欢迎投稿。

推荐阅读



从零搭建推荐系统——引擎篇

北冥渊

发表于智记