**抖音热门音乐特征分析**

**项目地址：** [**https://github.com/Cynyard999/NJUSEDouyinDataAnalysis**](https://github.com/Cynyard999/NJUSEDouyinDataAnalysis)

**成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | GitHub账户 | 主要分工 |
| 张卓楠 | 181830249 | github.com/sunflower-zzn | 数据处理 |
| 巫夷 | 181250153 | github.com/RickyWu9 | 数据提取 |
| 邱星曜 | 181830154 | github.com/Cynyard999 | 数据建模 |

目录

[**1. 摘要** 3](#_Toc47027829)

[**2. 引言** 3](#_Toc47027830)

[**3. 研究方法** 3](#_Toc47027831)

[**3.1 数据获取** 4](#_Toc47027832)

[**3.1.1 获取当下最热门的400首音乐** 4](#_Toc47027833)

[**3.2 音频处理** 5](#_Toc47027834)

[**3.2.1 音乐格式转换与时长处理** 6](#_Toc47027835)

[**3.2.2 midi生成** 7](#_Toc47027836)

[**3.3 音频特征** 7](#_Toc47027837)

[**3.3.1 基础音频信息** 8](#_Toc47027838)

[**3.3.2 统计特征提取** 12](#_Toc47027839)

[**3.4 数据降维及归一化** 17](#_Toc47027840)

[**3.4.1 PCA主成分分析** 17](#_Toc47027841)

[**3.4.2 数据归一化** 17](#_Toc47027842)

[**3.5 聚类算法** 18](#_Toc47027843)

[**3.5.1 K-Means聚类** 18](#_Toc47027844)

[**3.5.2 MiniBatch k-Means** 20](#_Toc47027845)

[**3.5.3 评价聚类模型** 20](#_Toc47027846)

[**3.6 分类算法** 22](#_Toc47027847)

[**3.6.1 HMM算法(隐马尔可夫模型)** 22](#_Toc47027848)

[**4. 案例实践** 25](#_Toc47027849)

[**4.1 数据爬取以及处理** 26](#_Toc47027850)

[**4.2 波形分析** 28](#_Toc47027851)

[**4.2.1 音乐波形图** 28](#_Toc47027852)

[**4.2.2 音乐语谱图** 31](#_Toc47027853)

[**4.2.3 音乐音调变化图** 34](#_Toc47027854)

[**4.2.4 音乐自相似矩阵图** 37](#_Toc47027855)

[**4.2.5 光谱过零率以及质心变化** 39](#_Toc47027856)

[**4.3 音乐特征分析聚类** 41](#_Toc47027857)

[**4.3.1 统计特征处理** 41](#_Toc47027858)

[**4.3.2 特征筛选** 42](#_Toc47027859)

[**4.3.3 归一化和降维处理** 44](#_Toc47027860)

[**4.3.4 K-Means聚类** 45](#_Toc47027861)

[**4.4 音乐风格分类** 50](#_Toc47027862)

[**4.4.1 HMM模型训练** 51](#_Toc47027863)

[**4.4.2 音乐分类预测** 53](#_Toc47027864)

[**4.5 midi文件生成** 55](#_Toc47027865)

[**5. 结论** 56](#_Toc47027866)

[**6. 反思与不足** 57](#_Toc47027867)

[**7. 参考文献** 57](#_Toc47027868)

**1. 摘要**

本次大作业选取了抖音当下最热门的400首音乐，通过一系列方法提取每首歌的波形特征，再经过降维以及机器学习等手段，进行无监督学习对音乐数据进行聚类的同时训练并使用监督学习分类器进行音乐流派分类，并通过可视化方法呈现分类聚类效果。

**关键词**：特征提取，PCA主成分分析，Normalization归一化，sklearn机器学习，pytorch神经网络，k-means聚类，Librosa音频处理，HMM隐马尔可夫模型，midi音序

**2. 引言**

“短视频红了之后，所有人听歌都只是听那一段。老一辈人说老歌好听，新人听歌只有15秒。”

随着移动网络与数字多媒体技术的飞速发展，基于快餐文化而快速崛起的短视频平台已经充斥在人们生活的各个角落，而随着人们的生活进行的“越来越快”，人们的时间貌似也越来越值钱，原本十几分钟才能讲完的事情，被浓缩到几分钟，甚至是十几秒就要讲完。

文本变为视频，无疑是满足了人们对于外界认知的获取速率的要求，但短视频平台产生的海量，庞大的视频数据确实大大超出了受众的需求和接收能力，因此，在一个月活五亿的平台上脱颖而出，抓住观众的感官，让视频观看量达到成千上万甚至达到百万级，千万级是所有短视频创作者的第一要务。

正如抖音名字所呈现的，音乐是抖音短视频的灵魂，背景音乐的选用是否恰当直接关系到作品的人们程度，因此，究竟什么样的音乐才能成为爆款，推动视频的传播，值得深入研究。

**3. 研究方法**

**3.1 数据获取**

**3.1.1 获取当下最热门的400首音乐**

由于数据量要求过大，仅仅通过人工获取数据明显是一个不现实的手段，我们选择利用爬虫工具对数据进行大批量的获取。但由于抖音短视频官网做了相当严密的反爬虫机制，很难从抖音官网获取视频信息，以及得到其背景音乐，所以我们选择使用第三方抖音数据分析网站来获取我们需要的热门音乐数据。

我们调查研究了*新榜、抖查查、66榜、飞瓜数据、卡思数据、蝉妈妈*等近十个分析网站，最终选取了蝉妈妈作为音乐数据来源。

注册会员后，通过点击音乐榜单，在f12控制台的network窗口监听到浏览器发出的http请求以及服务器返回的数据，经过筛选得到获取音乐榜单的请求：

search?keyword=&page=1&size=50&orderby=user\_count&incr\_type=7d&order=desc

通过分析header信息，利用python的*request*库模拟header对服务器发出请求，并且筛选得到音乐的名称，播放量等信息，写入csv:

try:

response = requests.get(url, headers=headers)

if response.status\_code == 200:

list = response.json().get('data').get('list')

for item in list:

del item['cover\_image']

del item['use\_trend']

del item['hot\_awemes']

del item['is\_fav']

return list

except requests.ConnectionError as e:

print('Error', e.args)

...

with open(csvPath, "w") as csvfile:

writer = csv.writer(csvfile)

writer.writerow(["音乐ID", "音乐名", "创作者", "歌曲时间", "使用人数", "使用人数增量"])

for item in music\_info:

writer.writerow([item.get('music\_id'), item.get('title'), item.get('author'), item.get('audition\_duration'),

item.get('user\_count'), item.get('user\_incr')])

print("csv写入成功！")

在得到音乐信息后，通过音乐的id，再次向服务器请求，获取音乐的mp3地址，下载后存储在本地:

try:

response = requests.get(url, headers=headers)

if response.status\_code == 200:

src = response.json().get('data')

print(name, end=": ")

if src != '' and src != 'None':

music = requests.get(src)

# 下面填写本地存储的路径，记得后缀添加mp3

open(downLoadPath + name + '.mp3', 'wb').write(music.content)

print("成功")

else:

print("歌曲不存在")

except requests.ConnectionError as e:

print('Error', e.args) # 输出异常信息

**3.2 音频处理**

**3.2.1 音乐格式转换与时长处理**

由于下载的音乐格式均为*mp3*，而后续所需的所有格式均为*wav*，所以先进行进一步的*mp3*到*wav*的转化:

for file in files: #遍历文件夹

if not os.path.isdir(file): #判断是否是文件夹，不是文件夹才打开

# 读取mp3的波形数据

sound = AudioSegment.from\_file(path+"/"+file, format='MP3')

# 将读取的波形数据转化为wav

f = wave.open(file+".wav", 'wb')

f.setnchannels(1) # 频道数

f.setsampwidth(2) # 量化位数

f.setframerate(16000) # 取样频率

f.setnframes(len(sound.\_data)) # 取样点数，波形数据的长度

f.writeframes(sound.\_data) # 写入波形数据

f.close()

由于大多数短视频的时长只有15s到20s，所以我们可以进一步将wav缩短至15s，因为我们认为，对于观看者而言，前15秒不能吸引到注意力，那么观看者将会毫不犹豫的切换到下一个视频。缩短音乐时长也同时降低了后期处理的复杂性:

def split\_music(begin, end, filepath, filename):

# 导入音乐

song = AudioSegment.from\_mp3(filepath)

# 取begin秒到end秒间的片段

song = song[begin \* 1000: end \* 1000]

# 存储为临时文件做备份

temp\_path = '../音乐/backup/' + filename+'.wav'

song.export(temp\_path)

return temp\_path

**3.2.2 midi生成**

由于我们要获取尽可能更多的纯音乐信息并设法生成我们的wav音乐，所以我们要从wav文件得到其midi文件的映射，从而来研究音乐的音调变化

import os

import subprocess

import time

dir='..\\wav\\'

filename= os.listdir(dir)

exe='wav2midi.exe'

print(filename)

for file in filename:

p = subprocess.Popen(exe+' '+dir+file)

time.sleep(10)

p.kill()

利用wave to midi映射程序，得到我们想要的midi文件（该文件在降噪处理后，利用傅里叶变换获取频率信息，再根据频率对应的音高得到音调）

**3.3 音频特征**

音频信号是（Audio）带有语音、音乐和音效的有规律的声波的频率、幅度变化信息载体。 根据声波的特征，可把音频信息分类为规则音频和不规则声音。其中规则音频又能够分为语音、音乐和音效。规则音频是一种连续变化的模拟信号，可用一条连续的曲线来表示，称为声波。声音的三个要素是音调、音强和音色。声波或正弦波有三个重要参数：频率 、幅度和相位 ，这也就决定了音频信号的特征。即**不一样频率和相位的正弦波的一个叠加**

**3.3.1 基础音频信息**

这里主要介绍波形结构和音频基础信息，即直接读取wav文件就可以获取的数据

**3.3.1.1 波形提取**

提取文件中所有的帧的信息。若文件为单通道，则直接将所有帧形成一维矩阵，若为双通道，则提取左声道的帧形成一维矩阵。最后将一维矩阵归一化，再将离散的点连线作图。

f=wave.open(filepath+file,'rb')#打开文件

params = f.getparams()

nchannels, sampwidth, framerate, nframes = params[:4]

strData = f.readframes(nframes)#读取音频，字符串格式

waveData = np.fromstring(strData,dtype=np.int16)#将字符串转化为int

waveData = waveData\*1.0/(max(abs(waveData)))#wave幅值归一化

waveData = np.reshape(waveData,[nframes,nchannels])

# plot the wave

time = np.arange(0,nframes)\*(1.0 / framerate)

plt.subplot(3,1,1)

plt.plot(time,waveData[:,0])

plt.xlabel("Time(s)")

plt.ylabel("Amplitude")

plt.title("Double channel wavedata")

plt.grid('on')#标尺，on：有，off:无。

plt.subplot(3,1,3)

plt.plot(time,waveData[:,1])

plt.xlabel("Time(s)")

plt.ylabel("Amplitude")

plt.title("Double channel wavedata")

plt.grid('on')#标尺，on：有，off:无。

plt.savefig('..\\波形图\\' + file + '.png')

plt.close()

print(file, "波形图已保存")

**3.3.1.2 音调提取**

利用傅里叶变换，将波形拆分成多个正弦曲线。不同的正弦曲线，代表的不同音调的波形，所以我们无法获取绝对应高，只能比较时间维度上，相对的音调变化。对音乐帧进行分段，8000个帧为一小段，进行快速傅里叶变换，并对分解的曲线进行对应其响度上的加权，最终获得该小段上的相对音高。最终将这些音高做成曲线。

def pitch(list):

ans=0

for i in range(len(list)):

ans+=(i+1)\*abs(list[i])

ans=ans/len(list)

return ans

f = wave.open(filepath+file, 'rb')#读取文件

params = f.getparams()

nchannels, sampwidth, framerate, nframes = params[:4]

time = nframes / framerate

strData = f.readframes(nframes) # 读取音频，字符串格式

waveData = np.fromstring(strData, dtype=np.int16) # 将字符串转化为int

data = waveData[0::2]

interval = 8000

extra = len(data) % interval

data = data[0:len(data) - extra]

anslist = []

for i in range(0, len(data), interval):

list = data[i:i + interval]

list = np.fft.rfft(list)

anslist.append(int(pitch(list)))

**3.3.1.3 过零率**

过零率（Zero Crossing Rate，ZCR）是指在每帧中，语音信号通过零点（从正变为负或从负变为正）的次数。 这个特征已在语音识别和音乐信息检索领域得到广泛使用，是对敲击的声音的分类的关键特征。对于一些卡点的bgm，其过零率会呈现更高的水平，即节奏性越强

代码如下

def crossingRate(filePath):

x, sr = librosa.load(filePath)

n0 = 9000

n1 = 9100

plt.figure(figsize=(14, 5))

plt.plot(x[n0:n1])

plt.grid()

zero\_crossings = librosa.zero\_crossings(x[n0:n1], pad=False)

return sum(zero\_crossings)

**3.3.1.4 频谱中心变化**

它指示声音的“质心”位于何处，并计算为声音中存在的频率的加权平均值。如果有两首歌曲，一首来自布鲁斯类型，另一首属于金属类型。从音乐风格来看，金属歌曲在接近尾声的频率会更高。因此，金属歌曲的光谱质心，相比于布鲁斯歌曲，将会更加朝向它的末端。我们通过提取音乐中的质心变化，就能知道这首歌在歌曲的那个部分达到高潮水平

代码如下

def centroid(filePath,fileName):

x, sr = librosa.load(filePath)

spectral\_centroids = librosa.feature.spectral\_centroid(x, sr=sr)[0]

print(spectral\_centroids)

# spectral\_centroids.shape(775,)

frames = range(len(spectral\_centroids))

t = librosa.frames\_to\_time(frames)

plt.figure(figsize=(20, 5))

librosa.display.waveplot(x, sr=sr, alpha=0.4)

plt.plot(t, normalize(spectral\_centroids), color='r')

# plt.savefig(source\_path+'/'+fileName.split('.')[0])

plt.show()

**3.3.1.5 梅尔频率倒谱系数**

信号的梅尔频率倒谱系数(MFCC)是一个通常由10-20个特征构成的集合，可简明地描述频谱包络的总体形状，这些形状决定了发出的声音是怎样的，如果能准确辨别出这些形状，就可以得到一种准确的**音位**

梅尔频率分析基于人类听觉感知实验。实验观测发现人耳就像一个滤波器组一样，它只关注某些特定的频率分量（人的听觉对频率是有选择性的）。而梅尔频率倒谱系数（Mel Frequency Cepstrum Coefficient, MFCC）考虑到了人类的听觉特征，先将线性频谱映射到基于听觉感知的Mel非线性频谱中，然后转换到倒谱上。

从普通频率转换到Mel非线性频率的公式为: $mel(f) = 2595\*lg(1+f/700)$

它可以将不统一的频率转化为统一的频率，也就是统一的滤波器组。

我们将频谱通过一组Mel滤波器就得到Mel频谱。公式表述就是:$lg X[k] = lg(Mel-Spectrum)$

1）取对数：$lg X[k] = lg H[k] + lg E[k]。$

2）进行逆变换：$x[k] = h[k] + e[k]$。

在Mel频谱上面获得的倒谱系数h[k]就称为Mel频率倒谱系数，简称MFCC。总的过程如下图：



**3.3.2 统计特征提取**

**3.3.2.1 时域特征（waveform）**

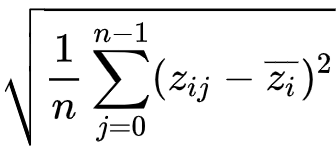
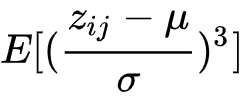
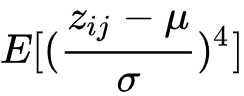
**含量纲的时域特征**

音频信号中含量纲的时域特征，常用的有十个，其中包括最大值(maximum)、最小值(minimum)、极差(range)、均值(mean)、中位数(media)、众数(mode)、标准差(standard deviation)、均方根值(root mean square/rms)、均方值(mean square/ms)、k阶中心/原点矩。

**无量纲的时域特征**

音频信号中无量纲的时域特征，分别为偏度(skewness)，峰度(kurtosis)，峰度因子(kurtosis factor)、波形因子(waveform factor)、脉冲因子(pulse factor)、裕度因子(margin factor)。

本次作业中我们选用其中的*均值*、*标准差*、*偏度*和*峰度*四项时域特征：

* 均值：
* 均值描述的是样本集合的中间点
* 
* 标准差：
* 标准差描述的是样本集合的各个样本点到均值的距离之平均
* 
* 偏度：
* 偏度是三阶标准矩
* 
* 峰度：
* 峰度是四阶标准矩
* 

# 提取时域特征

def TFeatures(x):

# 均值

mean = x.mean()

# 标准差

std = x.var() \*\* 0.5

# 偏态

skewness = ((x - mean) \*\* 3).mean() / std \*\* 3

# 峰态

kurtosis = ((x - mean) \*\* 4).mean() / std \*\* 4

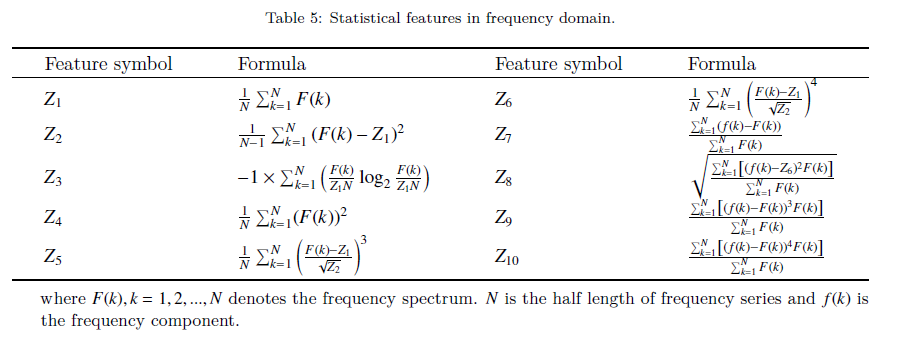
return [mean, std, skewness, kurtosis]

**3.3.2.2 频域特征（spectrogram）**

频域（频率域）——自变量是频率,即横轴是频率,纵轴是该频率信号的幅度,也就是通常说的频谱图。频谱图描述了信号的频率结构及频率与该频率信号幅度的关系。

对信号进行时域分析时，有时一些信号的时域参数相同，但并不能说明信号就完全相同。因为信号不仅随时间变化，还与频率、相位等信息有关，这就需要进一步分析信号的频率结构，并在频率域中对信号进行描述。动态信号从时间域变换到频率域主要通过傅立叶级数和傅立叶变换实现。周期信号靠傅立叶级数，非周期信号靠傅立叶变换。

下图为我们选用的10个频域特征的公式详情：



# 提取频域特征

def FFeatures(x):

x = np.array(x)

x = x.reshape(-1, 5).mean(1)

fft = np.fft.fft(x)

fft = fft[2: (fft.size // 2 + 1)]

fft = abs(fft)

power = fft.sum()

fft = np.array\_split(fft, 10)

return [i.sum() / power for i in fft]

**3.3.2.3 差分**

通常情况下，我们接收到的所有信号都是非平稳信号，比如我们的语音信号就是典型的非平稳信号。而平稳信号指在不同时间得到的采样值的统计特性(比如期望、方差等)是相同的，非平稳信号则与之相反，其特性会随时间变化。在信号处理中，这个特性通常指频率。对于非平稳信号，由于频率特性会随时间变化，为了捕获这一短时时变特性，我们需要对信号进行时频分析。

我们可以使用差分的方式来减轻数据之间的不规律波动，使其波动曲线更平稳，消除线性的趋势因素，得到平稳序列。通俗来说当间距相等时，用下一个数值，减去上一个数值 ，就叫“一阶差分”，做两次相同的动作，即再在一阶差分的基础上用后一个数值再减上一个数值一次，就叫“二阶差分"。

data = np.array(x)

diff = data[1:] - data[:-1]

**3.3.2.4 快速傅里叶变换（FFT）**

傅立叶原理表明：任何连续测量的时序或信号，都可以表示为不同频率的正弦波信号的无限叠加。而根据该原理创立的傅立叶变换算法利用直接测量到的原始信号，以累加方式来计算该信号中不同正弦波信号的频率、振幅和相位。

对于一个输入信号，假如我们不能确定该输入信号的频率组成，我们对其进行FFT处理之后，便可以很轻松的看出其频率分量，并且可以通过简单的计算来获知该信号的幅值信息等。在信号分析过程中，傅里叶变换的作用就是将组成这个回波信号的所有输入源在频域中按照频率的大小来表示出来。傅里叶变换之后，信号的幅度谱可表示对应频率的能量，而相位谱可表示对应频率的相位特征。经过傅立叶变换可以在频率中很容易的找出杂乱信号中各频率分量的幅度谱和相位谱，然后根据需求，进行高通或者低通滤波处理，最终得到所需要频率域的回波。

FFT（Fast Fourier Transformation），中文名快速傅里叶变换，是离散傅氏变换的快速算法，它是根据离散傅氏变换的奇、偶、虚、实等特性，对离散傅立叶变换的算法进行改进获得的。

fft = np.fft.fft(x)

fft = fft[2: (fft.size // 2 + 1)]

fft = abs(fft)

power = fft.sum()

fft = np.array\_split(fft, 10)

**3.3.2.5 平滑滤波（加窗）**

FFT提供了观察信号的新视角，但是FFT也有各种限制，可通过加窗增加信号的清晰度。使用FFT分析信号的频率成分时，分析的是有限的数据集合。 FFT认为波形是一组有限数据的集合，一个连续的波形是由若干段小波形组成的。 对于FFT而言，时域和频域都是环形的拓扑结构。时间上，波形的前后两个端点是相连的。 如测量的信号是周期信号，采集时间内刚好有整数个周期，那么FFT的上述假设合理。

平滑，也可叫滤波，或者合在一起叫平滑滤波，平滑滤波是低频增强的空间域滤波技术。它的目的有两类：一类是模糊；另一类是消除噪音。空间域的平滑滤波一般采用简单平均法进行，就是求邻近像元点的平均亮度值。邻域的大小与平滑的效果直接相关，邻域越大平滑的效果越好，但邻域过大，平滑会使边缘信息损失的越大，从而使输出的图像变得模糊，因此需合理选择邻域的大小。

我们主要通过平滑窗的选用来降低音频信号的噪声，并且显示音频信号在不同时间尺度上的特征值表现。

# 长度为1

data = x

# 长度为10

data = x.reshape(-1, 10).mean(1)

# 长度为100

data = x.reshape(-1, 100).mean(1)

# 长度为1000

data = x.reshape(-1, 1000).mean(1)

**3.4 数据降维及归一化**

**3.4.1 PCA主成分分析**

在多元统计分析中，主成分分析（英语：Principal components analysis，PCA）是一种统计分析、简化数据集的方法。它利用正交变换来对一系列可能相关的变量的观测值进行线性变换，从而投影为一系列线性不相关变量的值，这些不相关变量称为主成分（Principal Components）。具体地，主成分可以看做一个线性方程，其包含一系列线性系数来指示投影方向。PCA对原始数据的正则化或预处理敏感（相对缩放）。

主成分分析经常用于减少数据集的维数，同时保留数据集当中对方差贡献最大的特征。这是通过保留低维主成分，忽略高维主成分做到的。这样低维成分往往能够保留住数据的最重要部分。

from sklearn.decomposition import PCA

model\_pca = PCA(n\_components=2).fit(fvs)

fvs = model\_pca.transform(fvs)

**3.4.2 数据归一化**

数据的标准化（normalization）是将数据按比例缩放，使之落入一个小的特定区间。在某些比较和评价的指标处理中经常会用到，去除数据的单位限制，将其转化为无量纲的纯数值，便于不同单位或量级的指标能够进行比较和加权。

目前数据标准化方法有多种，归结起来可以分为直线型方法(如极值法、标准差法)、折线型方法(如三折线法)、曲线型方法(如半正态性分布)。不同的标准化方法，对系统的评价结果会产生不同的影响，然而不幸的是，在数据标准化方法的选择上，还没有通用的法则可以遵循。其中最典型的就是数据的归一化处理，即将数据统一映射到[0,1]区间上。

而我们为了特征数据的可视化效果，选用了StandardScaler均值-标准差归一化，保证均值为零标准差为1，会有大于0的情况。(注：所以处理后的数据区间可能会大于一)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(fvs)

fvs = scaler.transform(fvs)

**3.5 聚类算法**

聚类(Clustering)是按照某个特定标准(如距离)把一个数据集分割成不同的类或簇，使得同一个簇内的数据对象的相似性尽可能大，同时不在同一个簇中的数据对象的差异性也尽可能地大。也即聚类后同一类的数据尽可能聚集到一起，不同类数据尽量分离。

聚类属于无监督学习，目的是将具有相关性的数据聚合在一起而并不关心这些数据的标签。

**3.5.1 K-Means聚类**

K-Means聚类算法属于最基本的划分式聚类方法，需要事先指定簇类的数目或者聚类中心，通过反复迭代，直至最后达到"簇内的点足够近，簇间的点足够远"的目标。基于原型的、划分的距离技术，它试图发现用户指定个数(K)的簇。

算法原理：

选择K个点作为初始质心

repeat

将每个点指派到最近的质心，形成K个簇

重新计算每个簇的质心

until 簇不发生变化或达到最大迭代次数

特点：

* 需要提前确定k值
* 对初始质心点敏感
* 对异常数据敏感

from sklearn.cluster import KMeans

# 实例化k-means分类器，设置不同的k值，并计算评估系数

clf = KMeans(n\_clusters=8, init='k-means++', n\_init=10, max\_iter=300, tol=0.0001,

precompute\_distances='auto', verbose=0, random\_state=None,

copy\_x=True, n\_jobs=None, algorithm='auto')

y\_pred = clf.fit\_predict(X)

K-Means算法非常简单且使用广泛，但是其也有一些缺陷：

1. k值需要预先给定，属于预先知识，很多情况下k值的估计是非常困难的，对于像计算全部微信用户的交往圈这样的场景就完全的没办法用K-Means进行。对于可以确定K值不会太大但不明确精确的K值的场景，可以进行迭代运算，然后找出Cost Function最小时所对应的K值，这个值往往能较好的描述有多少个簇类。
2. K-Means算法对初始选取的聚类中心点是敏感的，不同的随机种子点得到的聚类结果完全不同
3. K-Means算法并不是很所有的数据类型。它不能处理非球形簇、不同尺寸和不同密度的簇，银冠指定足够大的簇的个数是他通常可以发现纯子簇。
4. 对离群点的数据进行聚类时，K-Means也有问题，这种情况下，离群点检测和删除有很大的帮助。

**3.5.2 MiniBatch k-Means**

MiniBatchKMeans是KMeans算法的变体，该算法使用小型批处理来缩短计算时间，同时仍在尝试优化相同的目标函数。小型批处理是输入数据的子集，在每个训练迭代中随机采样。这些小型批处理大大减少了收敛到本地解决方案所需的计算量。与其他减少 k-means 收敛时间的算法相比，小批 k-means 产生的结果通常只比标准算法稍差。

该算法在两个主要步骤之间进行检测，类似于 k-means 。在第一步中，从数据集中随机抽取样本，以形成小型批处理。然后，这些被分配给最近的质心。在第二步中，重心更新。与 k-means 不同，这是基于每个样本完成的。对于小批量中的每个样本，通过采集样本的流平均值和分配给该质心的所有先前样本来更新分配的质心。这具有随着时间的推移降低质心变化速率的效果。执行这些步骤，直到达到收敛或预定的迭代次数。

from sklearn.cluster import MiniBatchKMeans

# 实例化k-means分类器，设置k值，随机状态

clf = MiniBatchKMeans(n\_clusters=k, batch\_size = 400, precompute\_distances='auto', verbose=0, random\_state=None,

copy\_x=True, n\_jobs=None, algorithm='auto')

y\_pred = clf.fit\_predict(X)

**3.5.3 评价聚类模型**

**3.5.3.1 轮廓系数**

当文本类别未知时，可以选择轮廓系数作为聚类性能的评估指标。轮廓系数取值范围为[-1,1]，取值越接近1则说明聚类性能越好，相反，取值越接近-1则说明聚类性能越差。

from sklearn.metrics import silhouette\_score

import matplotlib.pyplot as plt

silhouettescore=[]

for i in range(2,15):

kmeans=KMeans(n\_clusters=i,random\_state=123).fit(iris\_data)

score=silhouette\_score(iris\_data,kmeans.labels\_)

silhouettescore.append(score)

plt.figure(figsize=(10,6))

plt.plot(range(2,15),silhouettescore,linewidth=1.5,linestyle='-')

plt.show()

轮廓系数的优点

轮廓系数为-1时表示聚类结果不好，为+1时表示簇内实例之间紧凑，为0时表示有簇重叠。轮廓系数越大，表示簇内实例之间紧凑，簇间距离大，这正是聚类的标准概念。

轮廓系数的缺点

对于簇结构为凸的数据轮廓系数值高，而对于簇结构非凸需要使用DBSCAN进行聚类的数据，轮廓系数值低，因此，轮廓系数不应该用来评估不同聚类算法之间的优劣，比如Kmeans聚类结果与DBSCAN聚类结果之间的比较。

**3.5.3.2 Calinski-Harabasz指数评价**

CH（Calinski-Harabasz）指标通过计算总体的相似度，簇间平均相似度或簇内平均相似度来评价聚类质量。评价聚类效果的高低通常使用聚类的有效性指标，所以目前的检验聚类的有效性指标主要是通过簇间距离和簇内距离来衡量。

from sklearn.metrics import calinski\_harabaz\_score

for i in range(2,7):

kmeans=KMeans(n\_clusters=i,random\_state=123).fit(iris\_data)

score=calinski\_harabaz\_score(iris\_data,kmeans.labels\_)

print("聚类%d簇的calinski\_harabaz分数为：%f" %(i,score))

CH指数的优点

当簇类密集且簇间分离较好时，Caliniski-Harabaz分数越高，聚类性能越好。

计算速度快。

CH指数的缺点

凸簇的Caliniski-Harabaz指数比其他类型的簇高，比如通过DBSCAN获得的基于密度的簇。

**3.6 分类算法**

分类是一个有监督的学习过程，目标数据库中有哪些类别是已知的，分类过程需要做的就是把每一条记录归到对应的类别之中。

**3.6.1 HMM算法(隐马尔可夫模型)**

隐马尔可夫模型非常擅长对时间序列数据进行建模。由于音乐音频文件是时间序列信号，我们希望HMM能够满足我们的需求，给我们一个准确的分类。隐马尔可夫模型是表示观察序列的概率分布的模型。我们假设输出是由隐藏状态生成的。

通过包装hmmlearn库提供的模型，我们构建了一个处理HMM训练和预测的Python类。

from hmmlearn import hmm

class HMMTrainer(object):

def \_\_init\_\_(self, model\_name='GaussianHMM', n\_components=4, cov\_type='diag', n\_iter=1000):

self.model\_name = model\_name

self.n\_components = n\_components

self.cov\_type = cov\_type

self.n\_iter = n\_iter

self.models = []

if self.model\_name == 'GaussianHMM':

self.model = hmm.GaussianHMM(n\_components=self.n\_components, covariance\_type=self.cov\_type,n\_iter=self.n\_iter)

else:

raise TypeError('Invalid model type')

def train(self, X):

np.seterr(all='ignore')

self.models.append(self.model.fit(X))

# Run the model on input data

def get\_score(self, input\_data):

return self.model.score(input\_data)

为了训练隐马尔可夫模型，我们迭代子文件夹中的歌曲以提取特征并将其附加到变量。

def trainHHM():

hmm\_models = []

input\_folder = samplesPath

for dirname in os.listdir(input\_folder):

# 得到音乐类型文件夹

subfolder = os.path.join(input\_folder, dirname)

if not os.path.isdir(subfolder):

continue

# 提取音乐类型

label = subfolder[subfolder.rfind('/') + 1:]

X = np.array([])

y\_words = []

# 遍历该音乐类型文件夹下的所有音乐

print("读取" + label + "类型样本数据...")

for filename in [x for x in os.listdir(subfolder) if x.endswith('.wav')][:-1]:

filepath = os.path.join(subfolder, filename)

sampling\_freq, audio = wavfile.read(filepath)

# 得到mfcc特征

mfcc\_features = mfcc(audio, sampling\_freq)

if len(X) == 0:

X = mfcc\_features

else:

X = np.append(X, mfcc\_features, axis=0)

# 标记该音乐的类型

y\_words.append(label)

hmm\_trainer = HMMTrainer(n\_components=4)

print(label + "类型数据读取完成，" + "开始训练...")

hmm\_trainer.train(X)

print("训练结束!")

hmm\_models.append((hmm\_trainer, label))

hmm\_trainer = None

return hmm\_models

隐马尔可夫模型训练结束后，将其训练的模型作为参数，传入预测未知音乐的类型的方法中，将每一首未知音乐放入十种音乐类型的模型当中，取得相似得分最高的一种类型，我们就可以粗略认为该未知音乐属于该类型。

def predict(hmm\_models):

print("开始音乐分类...")

input\_folder = wavesPath

pred\_labels = []

for dirname in os.listdir(input\_folder):

subfolder = os.path.join(input\_folder, dirname)

if not os.path.isdir(subfolder):

continue

for filename in [x for x in os.listdir(subfolder) if x.endswith('.wav')][:-1]:

filepath = os.path.join(subfolder, filename)

sampling\_freq, audio = wavfile.read(filepath)

mfcc\_features = mfcc(audio, sampling\_freq)

max\_score = -99999999999999

output\_label = None

for item in hmm\_models:

hmm\_model, label = item

score = hmm\_model.get\_score(mfcc\_features)

if score > max\_score:

max\_score = score

output\_label = label

pred\_labels.append(output\_label)

# print(filename + ": " + output\_label)

result.get(output\_label).append(filename)

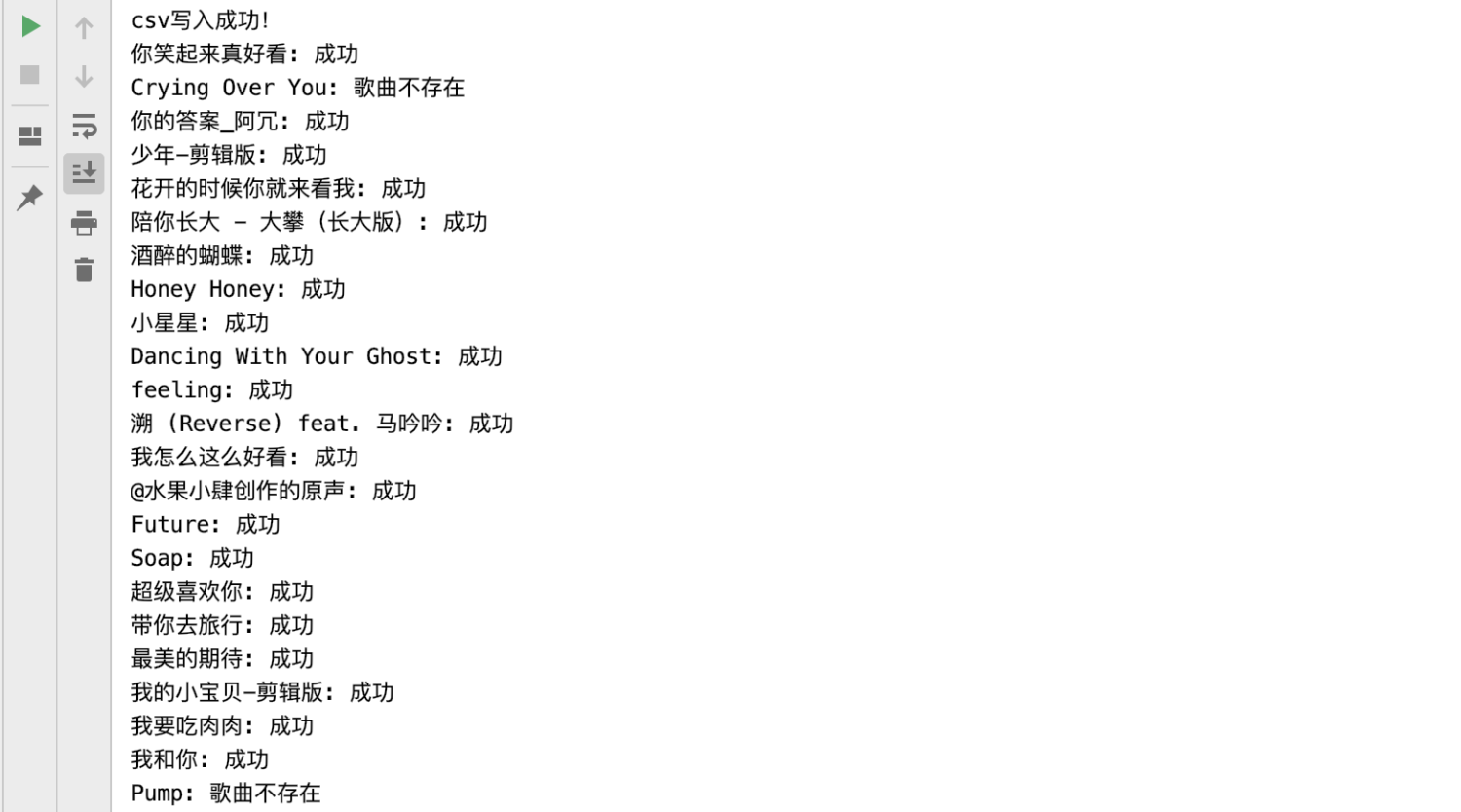
隐马尔可夫模型的性能相对平均，显然，还有很大的改进空间。实际上，隐马尔可夫模型对于时间序列数据是可解释的和强大的，但是，它们需要大量的微调（隐藏状态的数量，输入特征等）。也可以尝试其他方法来对音乐类型进行分类，比如循环神经网络(LSTM)。

**4. 案例实践**

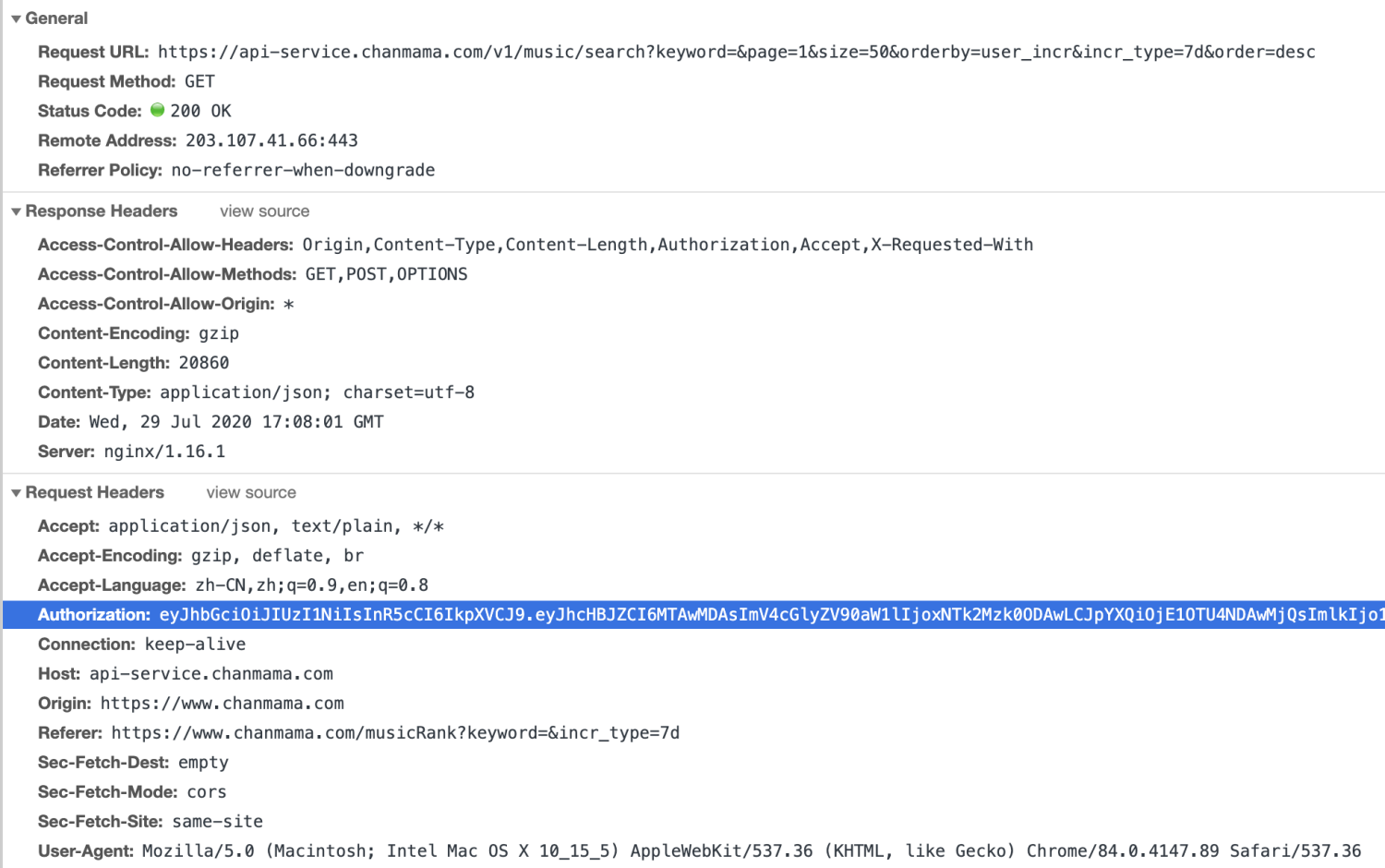
**4.1 数据爬取以及处理**

以蝉妈妈数据榜单前50首为例，通过对header的解析，模拟浏览器向服务器发出请求，最终保存音乐以及音乐信息至本地。

header信息如下：



爬取过程控制台输出如下



最终本地文件效果如下

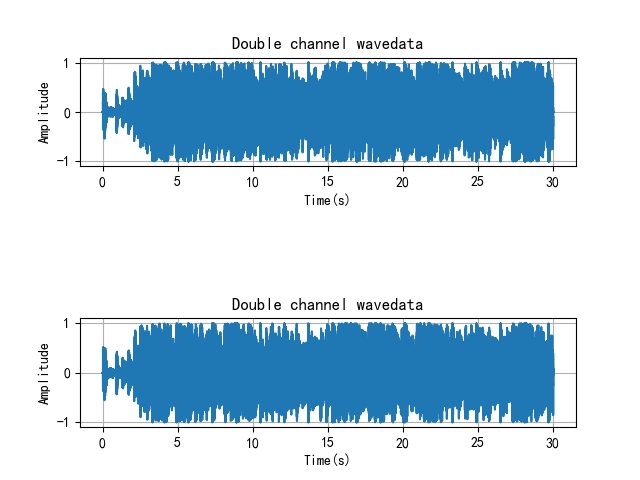


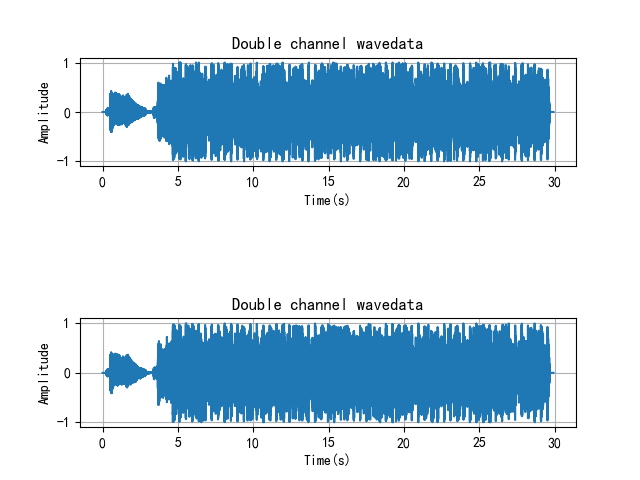
**4.2 波形分析**

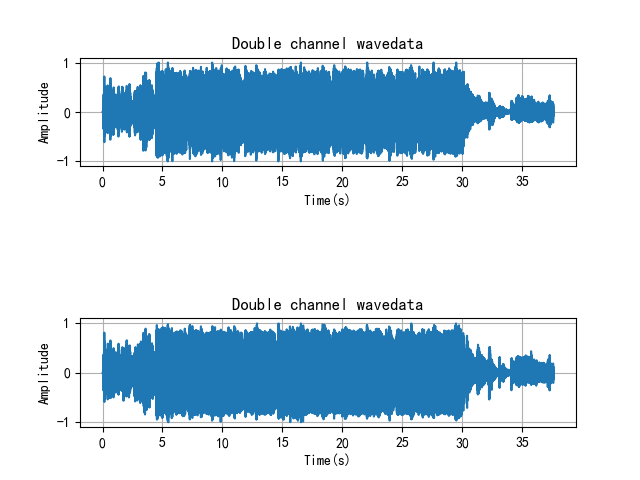
**4.2.1 音乐波形图**

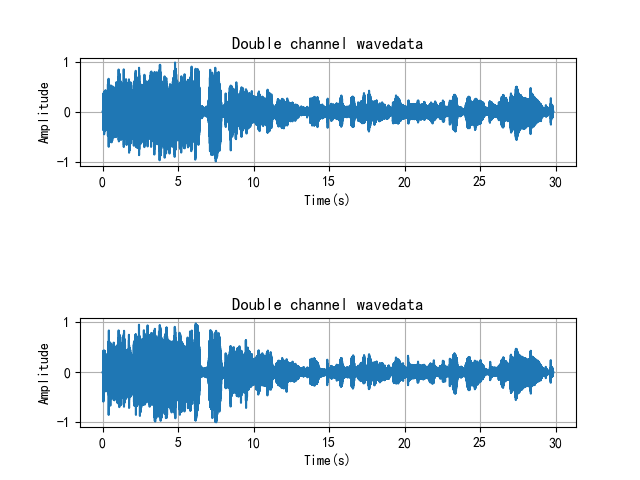
根据歌曲生成的波形曲线，在经过坐标归一化后，我们可以看到下面四张案例，振幅曲线在一开始进入歌曲时，有明显的变化，而再完全进入歌曲副歌部分后，波形变化较小。这使我们认识到，传统所认为的抑扬顿挫的优美感，更适用于音调的变化，而不是音量。而歌曲一开始的音量变化，则是为进入副歌做铺垫，使得歌曲的层次分明，让人留下更深的印象。

**图例**

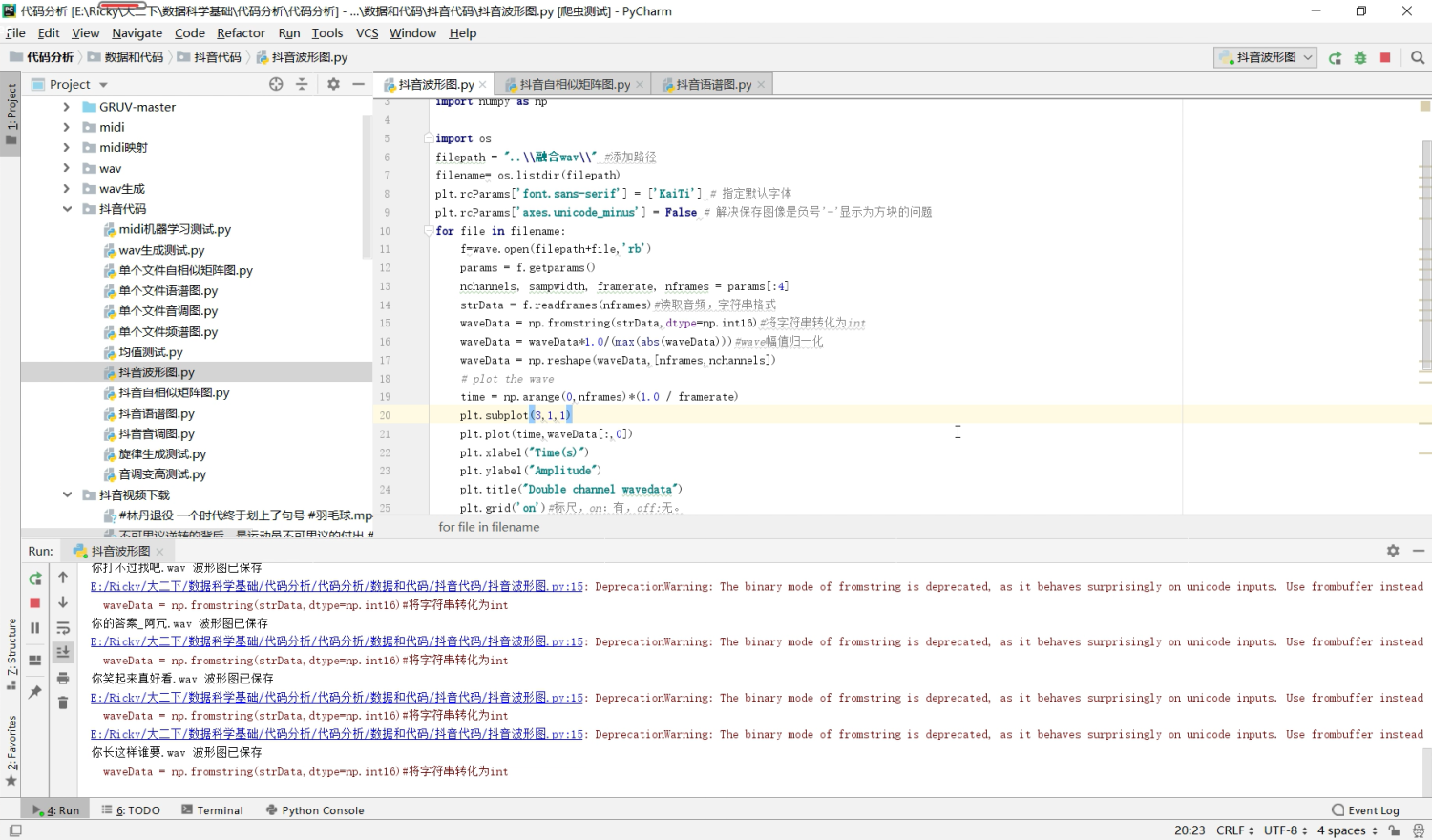








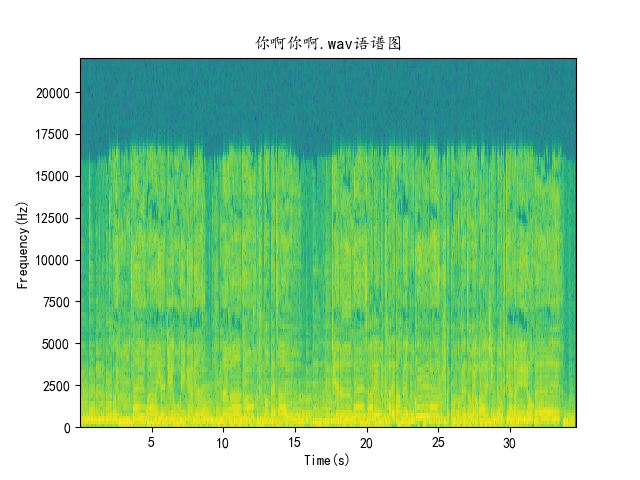
运行时截图：

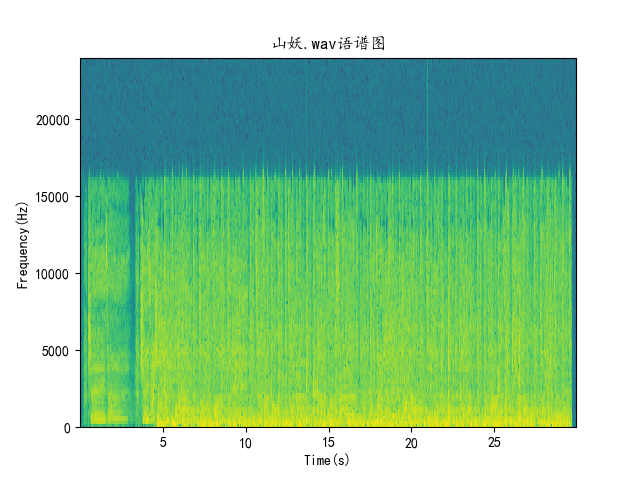


**4.2.2 音乐语谱图**

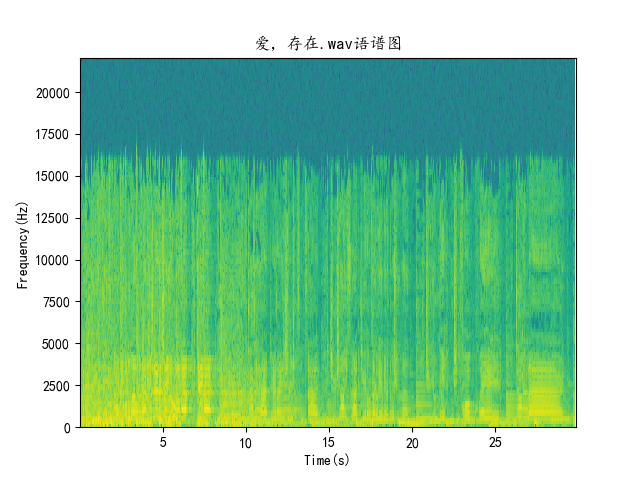
在语谱图中，我们根据亮度，可以看出不同时间上，不同声音频率的占据成分。下面四张图不同时间的的音频都集中在0到16000上，而不同频率的声波成分占比中，低频占比更高。**我们认为，这是和声在音乐中的衬托作用**，通过和声，音调差，给音乐营造一种意境感和距离感。

**图例**

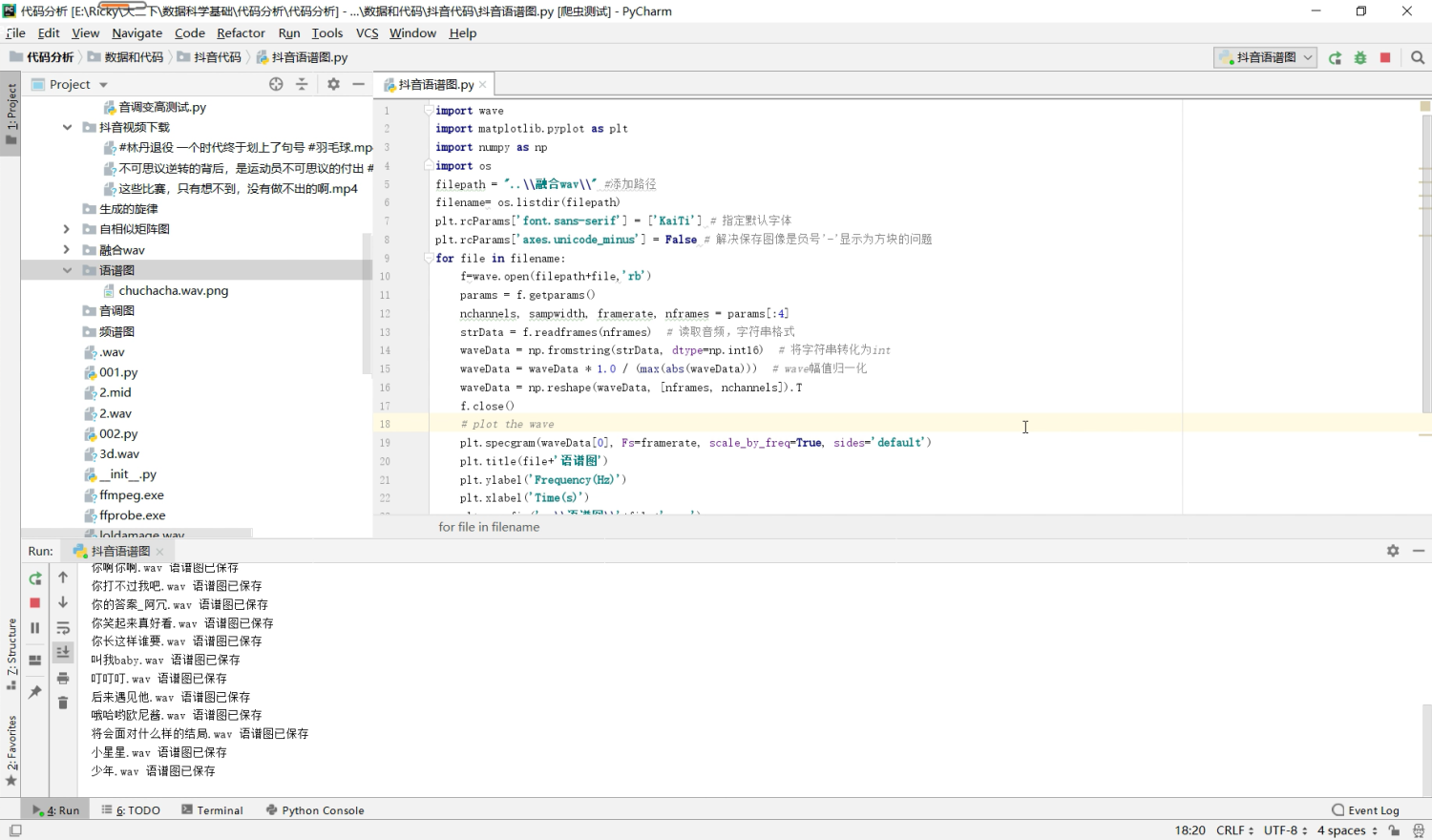








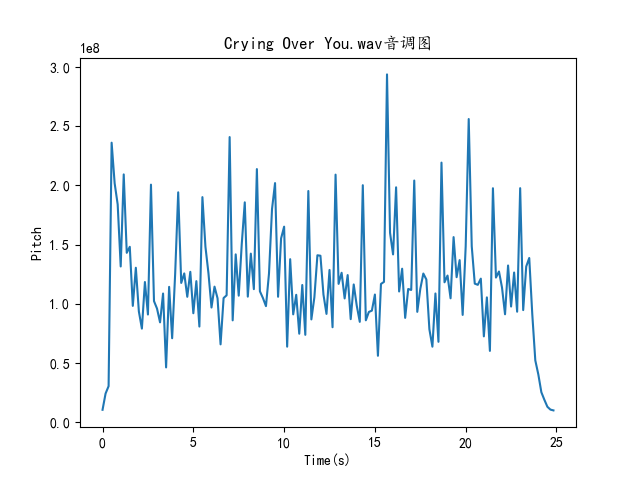
运行时截图：



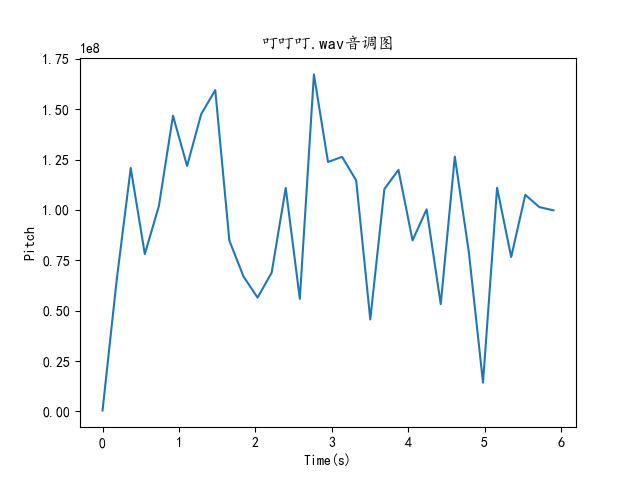
**4.2.3 音乐音调变化图**

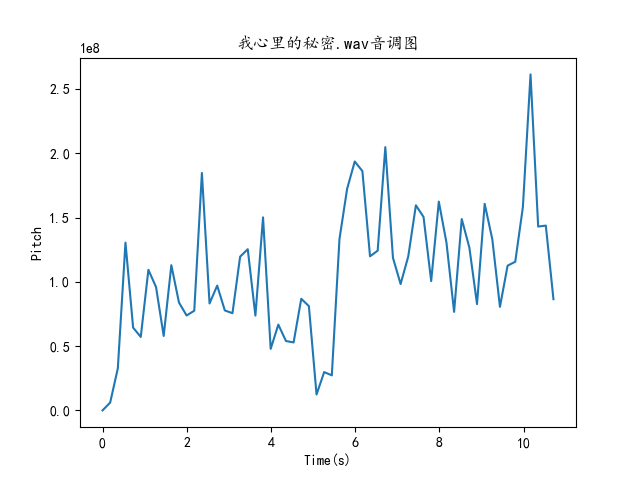
音调图上，我们可以明显看出音乐所具有的两个特性，变化性以及周期性。在副歌中，音调的变化程度很大，且呈现了一定的周期性。我们认为，一个好的歌曲，**在其副歌部分必须要有抑扬顿挫的音调变化**，以及变化的旋律性（周期性）。**用重复，但稍加变化的音调组合，是副歌好听的评判标准**。

**图例**

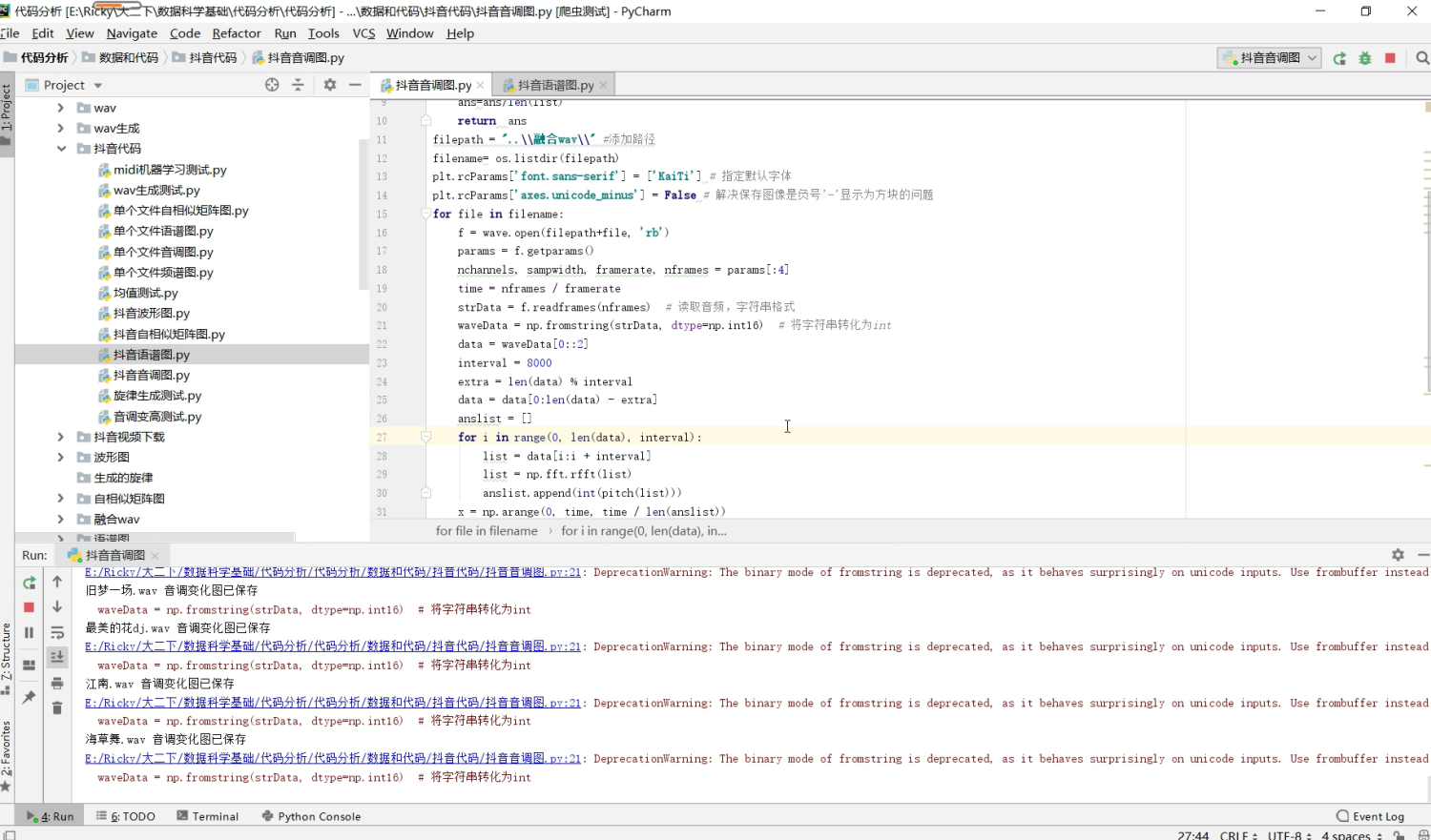








运行时截图：

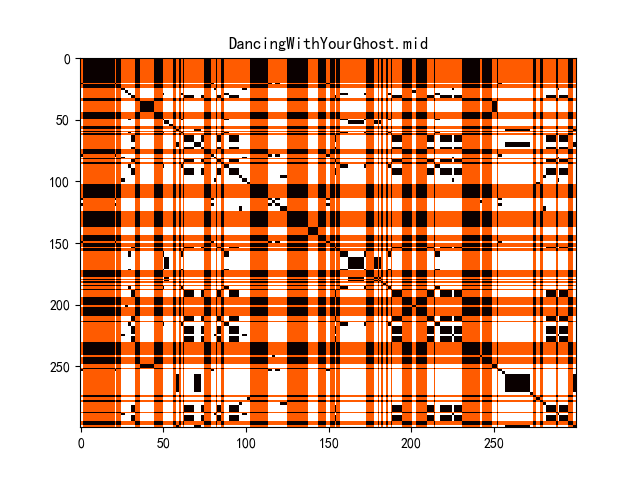


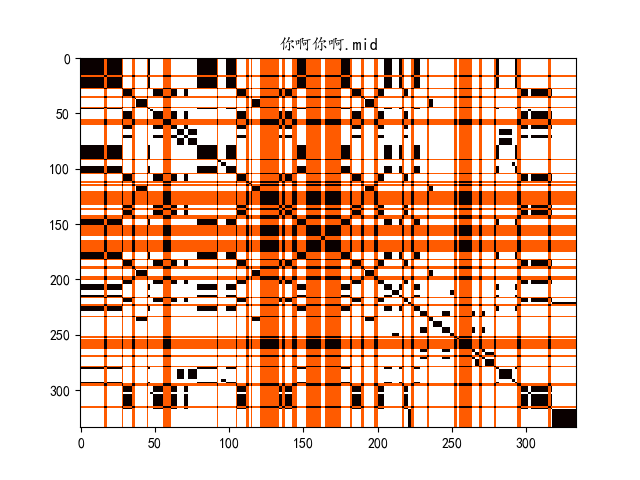
**4.2.4 音乐自相似矩阵图**

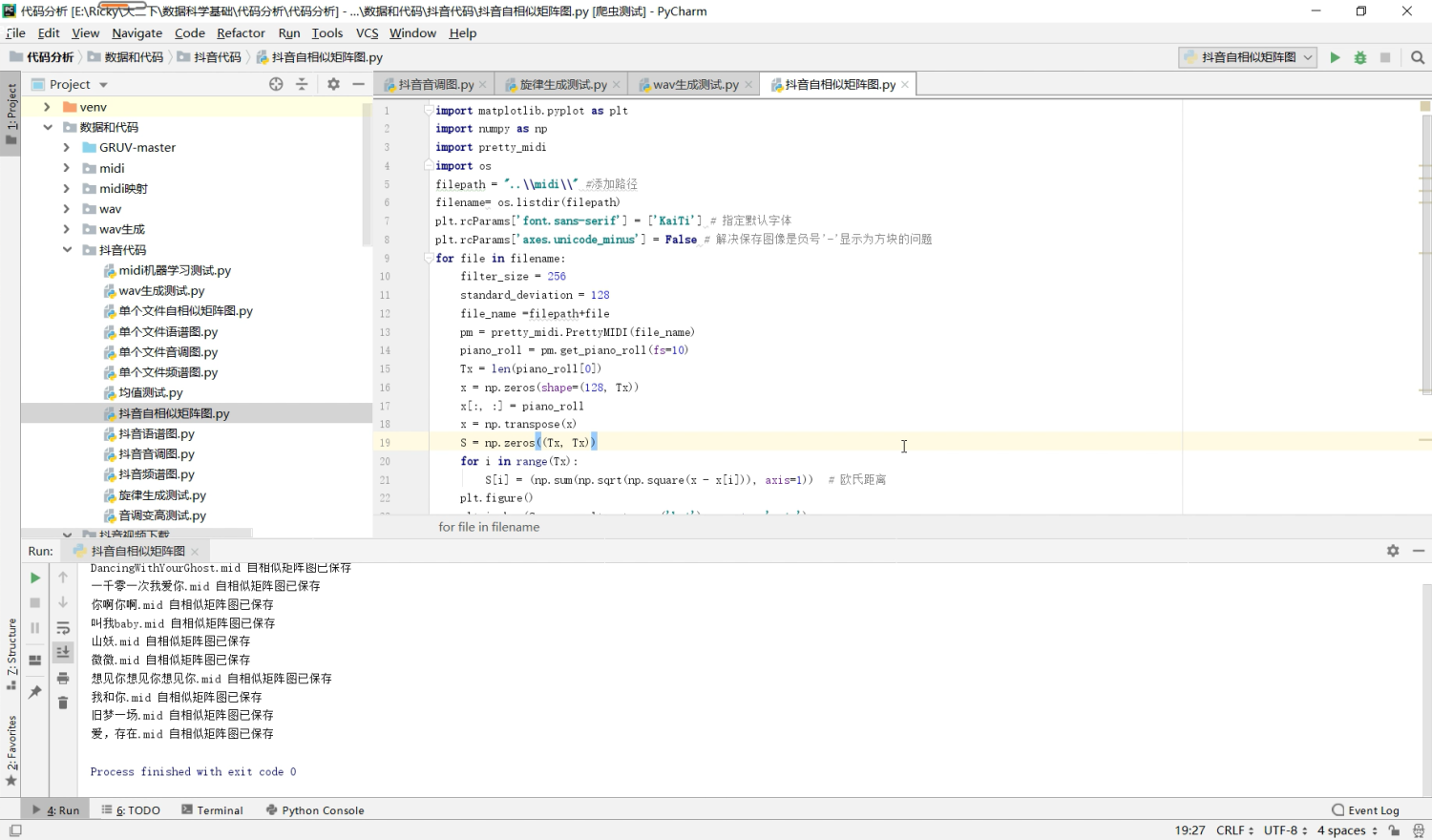
读取midi文件,根据其音调信息生成谱图

根据谱图对角线方块颜色的变化，我们可以看出音乐音调变化相似性，从自相似矩阵中，我们可以验证音乐本身所具有的节奏感，旋律感。

**图例**







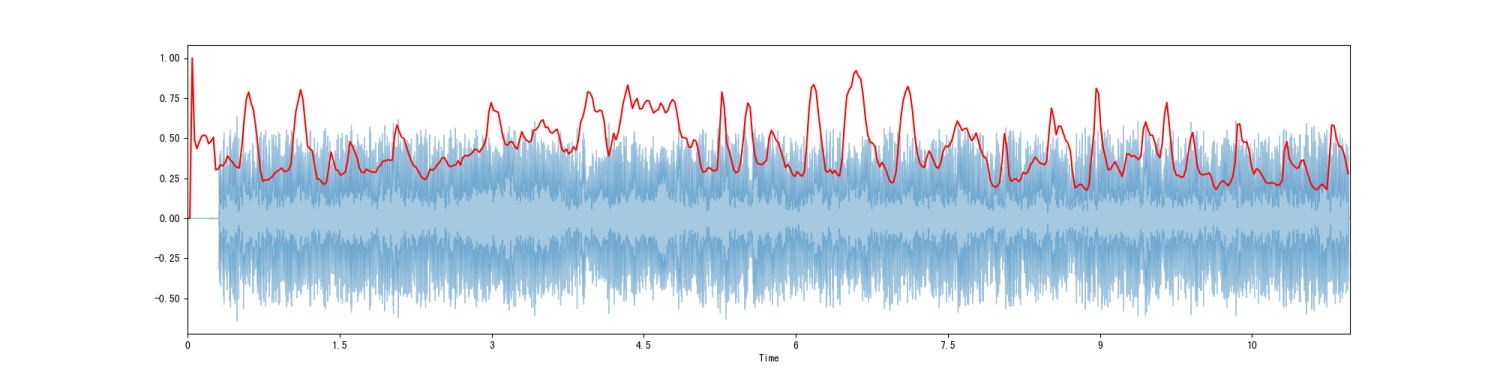
**4.2.5 光谱过零率以及质心变化**

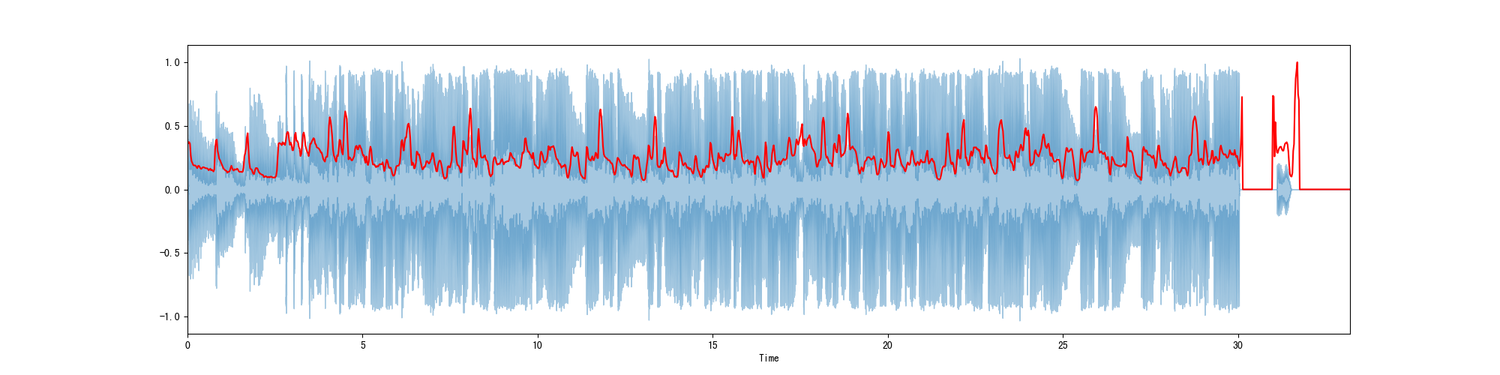
首先，任选九首歌得到其过零点的个数，如下图

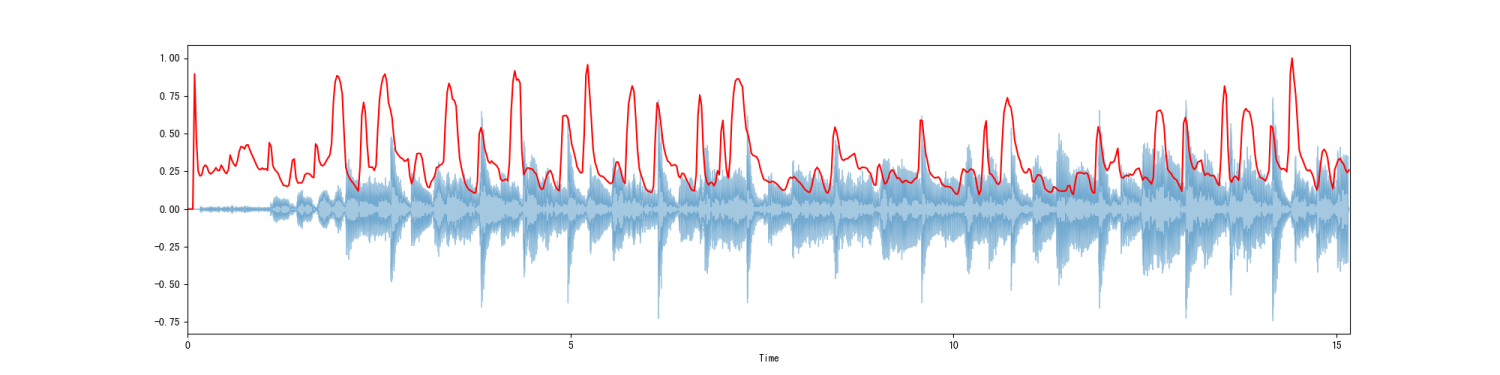


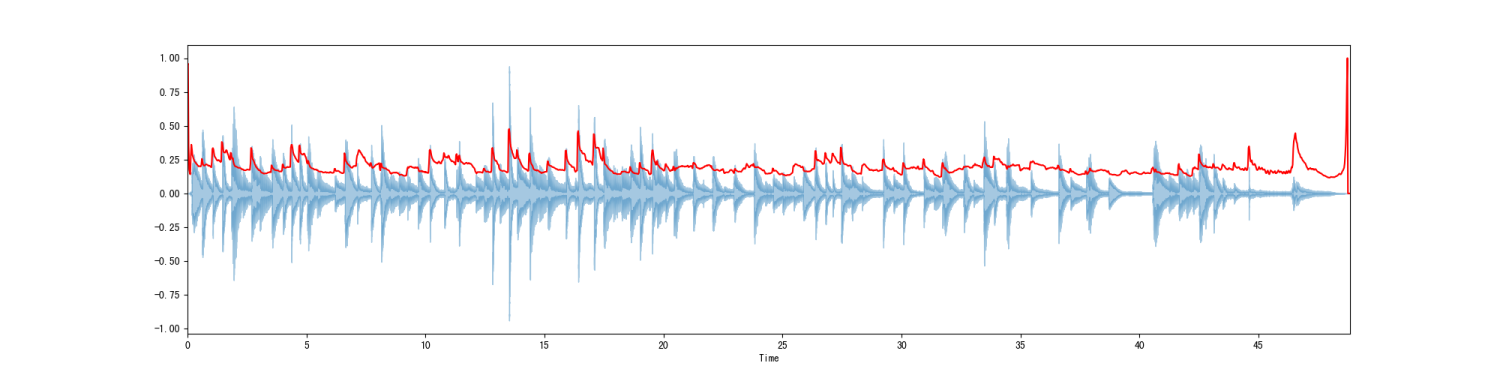
可以看出，不同的歌曲有着不同的过零点，即整体歌的节奏不同，通过对比过零点最高的音乐**我要吃肉肉**以及过零点最低的音乐**心愿便利贴**，可以分辨出后者明显节奏更为舒缓，打点节奏更为缓和。

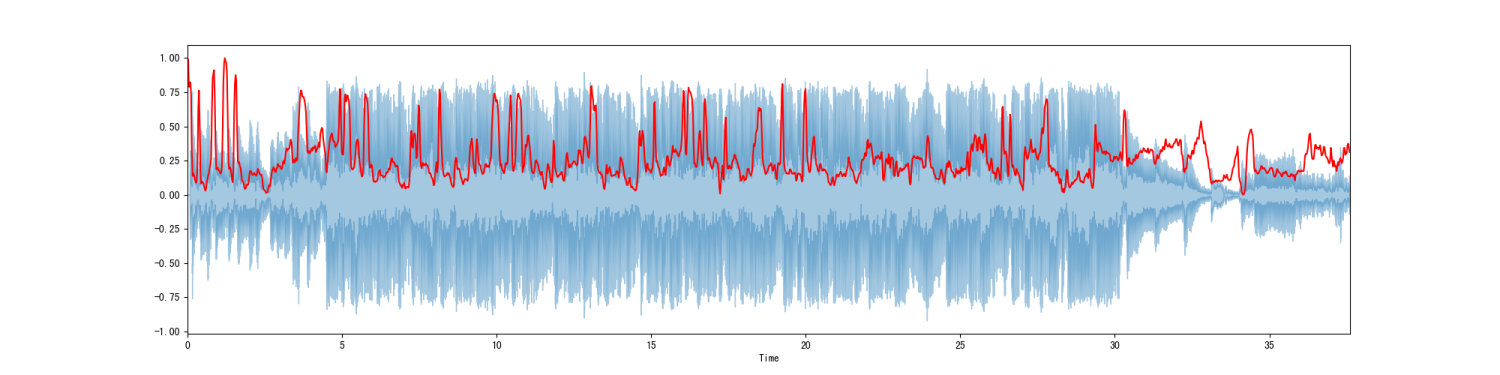
之后，我们对这九首歌做质心变化的统计，结果如下图：

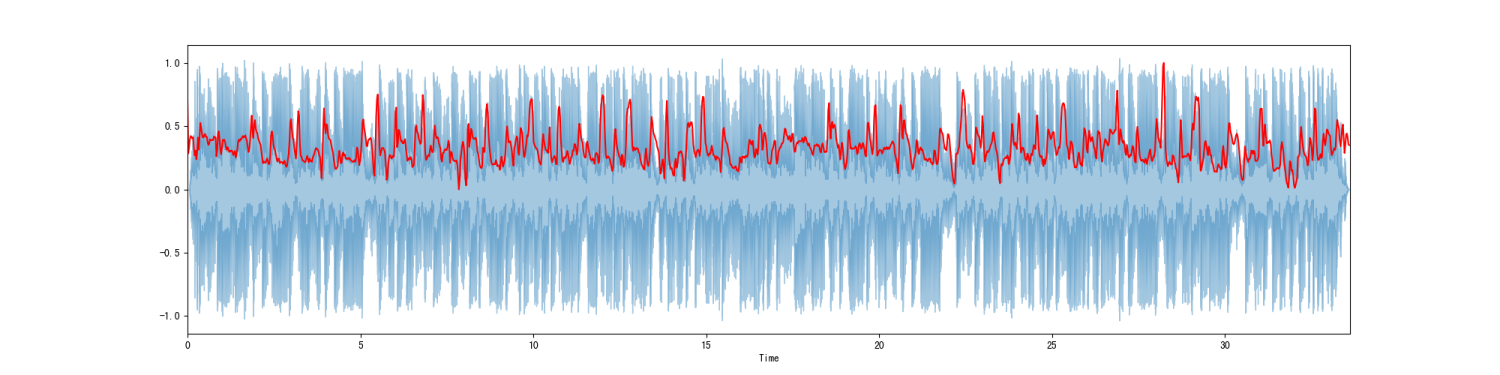


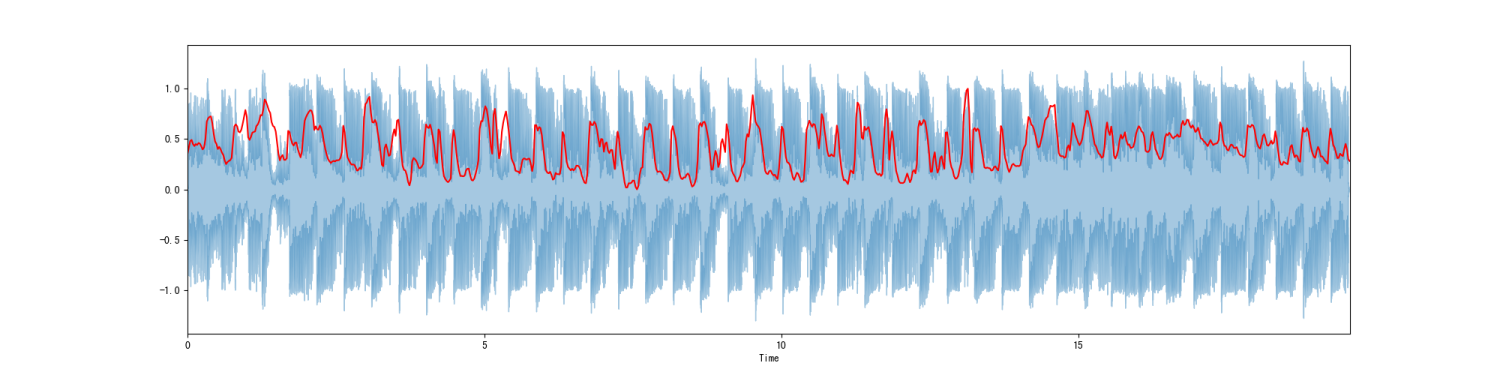


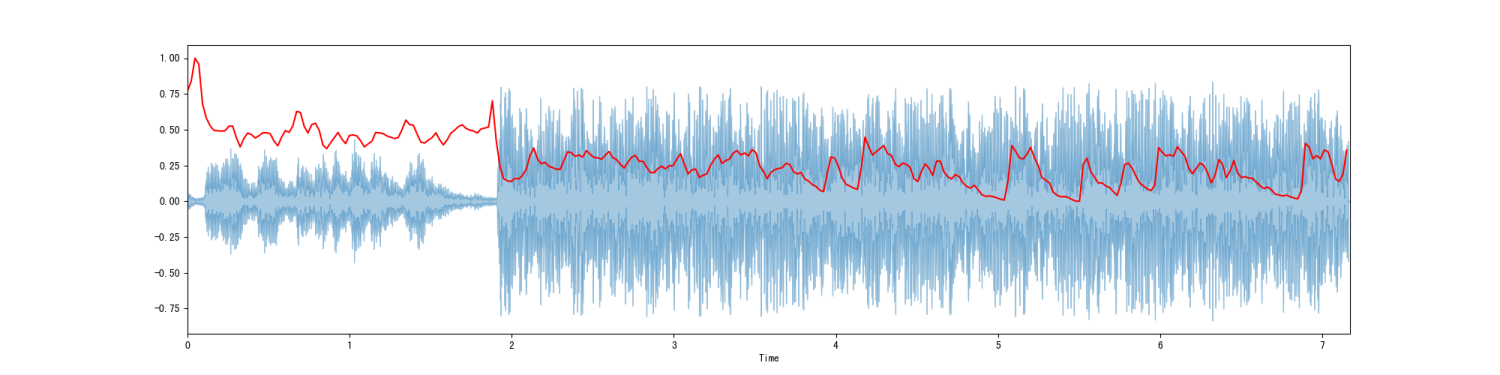


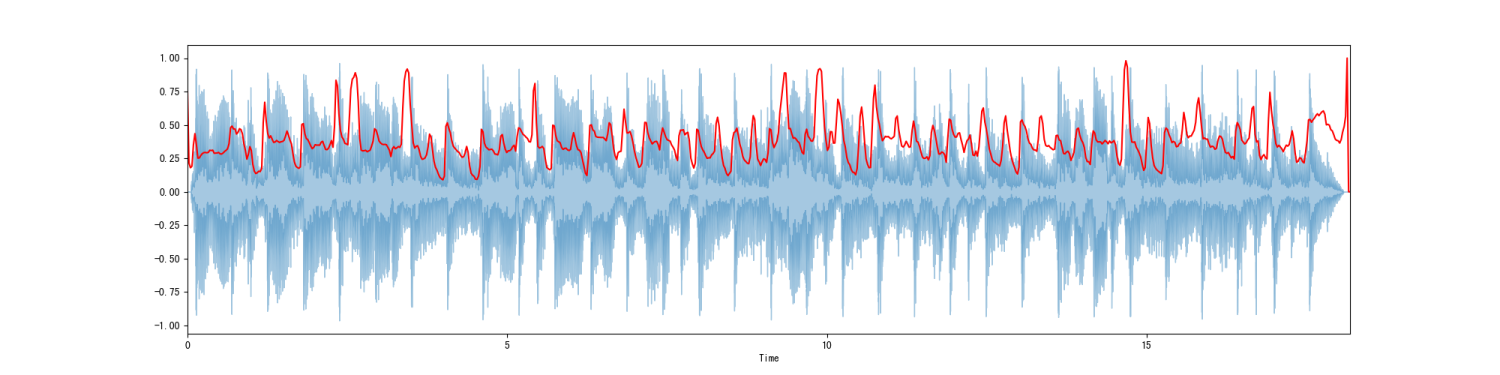












可以很明显的看出，每首歌曲的质心在歌曲开始的时候会达到一个相对较高的水平，之后随着时间不断高低跳动，可以看出这些热门歌曲的共同特征都旨在开头吸引人的注意力，之后随着自身的节奏变化来吸引人们注意

**4.3 音乐特征分析聚类**

在之前的波形特征提取部分，我们研究了不同音乐间的歌的**波形结构**，但我们发现即使同一名歌手演唱的同一首歌曲的不同段落音频波形结构之间的差异也很大，而我们想要探讨这些抖音热歌之间的共同点以及与非热门歌曲之间的区别，为此我们想到了提取音乐的统计特征并进行**聚类**来探讨他们之间的**关联性**。

**4.3.1 统计特征处理**

根据我们之前进行的音频特征提取，我们可以得到音频的波形结构的时域频域特征，通过相应的通过一阶差分和平滑窗(递增平滑，长度分别为1,10,100,1000)来获取这些特征在不同时间尺度上的表现，以及对歌曲在不同评断的能量占比进行快速傅里叶变化，最终得出如下42个特征值：

4\*4+4\*4+10=42

# 在四个平滑窗下的16个特征值

amp1mean amp1std amp1skew amp1kurt

amp10mean amp10std amp10skew amp10kurt

amp100mean amp100std amp100skew amp100kurt

amp1000mean amp1000std amp1000skew amp1000kurt

# 一阶差分幅度下在四个平滑窗下的16个特征值

amp1dmean amp1dstd amp1dskew amp1dkurt

amp10dmean amp10dstd amp10dskew amp10dkurt

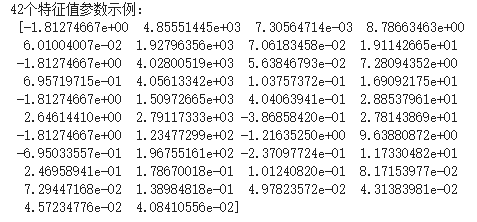
amp100dmean amp100dstd amp100dskew amp100dkurt

amp1000dmean amp1000dstd amp1000dskew amp1000dkurt

# 十个频段的频域特征

power1 power2 power3 power4 power5

power6 power7 power8 power9 power10

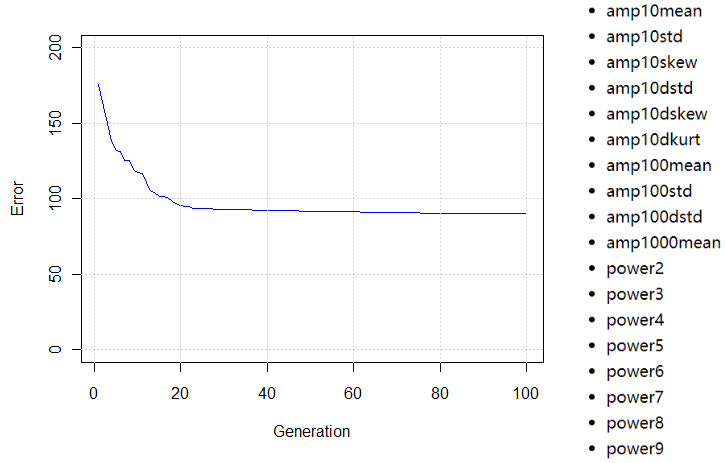


**4.3.2 特征筛选**

然而在使用这些特征值的过程中我们发现，由于这些特征值是人为提取的，所以并不能很好地表现出歌曲特征，并且这些特征之间的相关系数是不为0的，也就是说存在冗余特征，因此我们需要对特征值进行筛选。

我们从每首歌曲中抽取两个15s 的样本，并尝试找到一种算法，该算法可以最好地将每首歌曲中的两个样本匹配在一起。

为了找到在所有歌曲中提供最佳平均匹配特征值子集，我们使用**遗传算法（R 中的 genalg 包）**来控制42 个特征值中的每个特征值。下面的图显示了遗传算法超过 100 代的目标函数（即歌曲的两个样本由最近的邻域分类器匹配在一起）的结果。



如果我们选用全部42个特征值，那么目标函数的错误率会达到275，而通过筛选上图中右侧的18个特征值，我们能够将目标函数的错误率在100次以上迭代中降低到90，这是一个重大的优化。

通过对42个特征值数组的提取重组，我们可以得到音乐的18个特征值向量(上图右侧)：



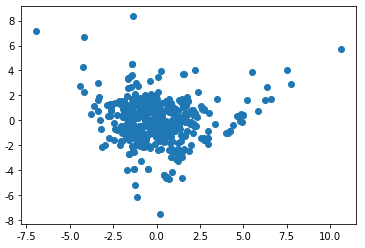
**4.3.3 归一化和降维处理**

为了最后服务于我们的歌曲聚类目标以及最后结果的可视化，我们需要将这些数据进行归一化处理和降维处理。

使用StandardScaler进行去均值和方差归一化，调用sklearn的preprocessing库进行数据的归一化。

使用PCA主成分分析方法进行数据降维，将18维数据降为2维数据，调用sklearn的decomposition进行主成分分析降维。

最终处理结束后绘制数据分布图：



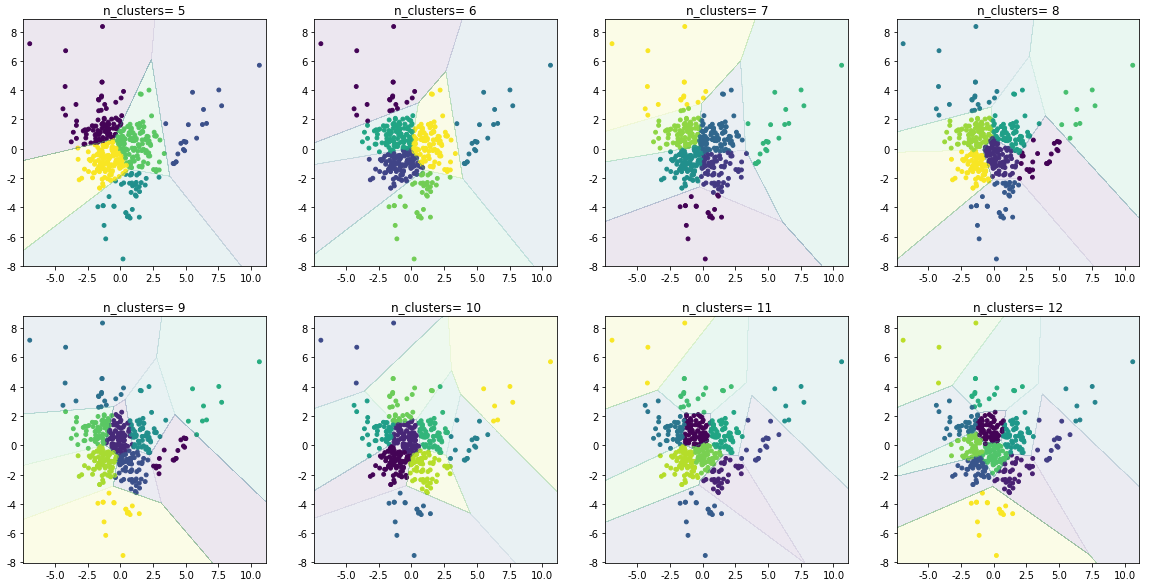
**4.3.4 K-Means聚类**

为了分析这些音乐数据之间的关联关系，我们使用了无监督分类的方式试图找到他们之间的关联关系。

**4.3.4.1 传统K-Means算法**

KMeans算法通过尝试在等方差组中分离样本来对数据进行聚类，从而最小化称为惯性或聚类内平方和的标准。此算法要求指定群集数。它可很好地扩展到大量的样品中，并已被广泛应用于许多不同的领域。k-means算法将一组样本划分为不相交的聚类，每个样本都用群集中样本的均值描述。这些手段通常称为聚类"中心"。

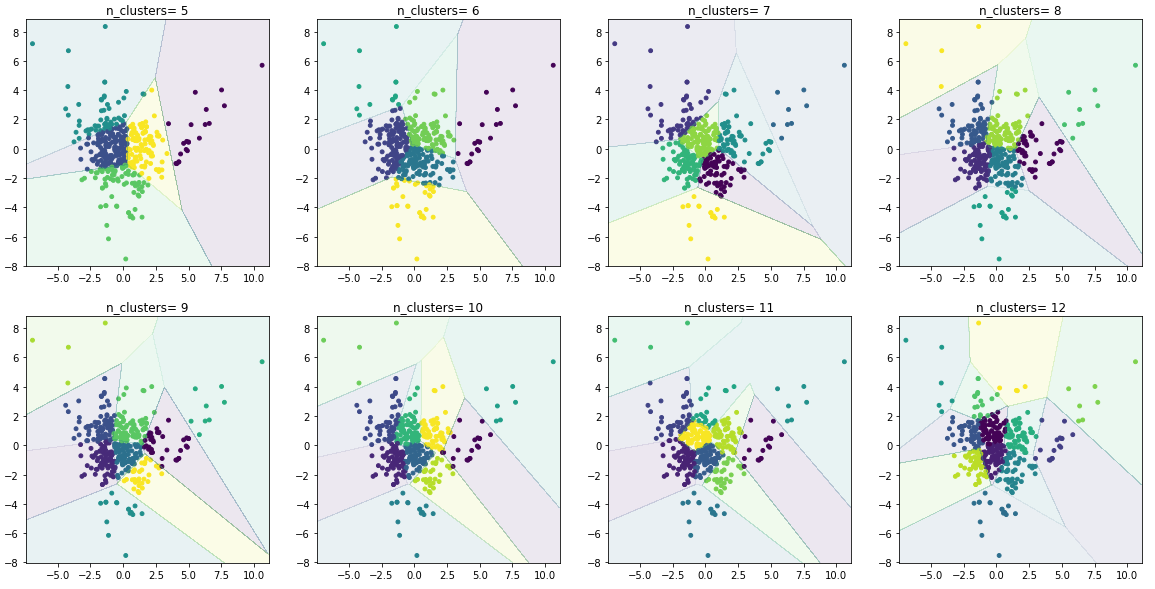
我们绘制了k=5-k=13时的聚类结果：



**4.3.4.2 MiniBatchKMeans算法**

MiniBatchKMeans是KMeans算法的变体，该算法使用小型批处理来缩短计算时间，同时仍在尝试优化相同的目标函数。小型批处理是输入数据的子集，在每个训练迭代中随机采样。这些小型批处理大大减少了收敛到本地解决方案所需的计算量。与其他减少 k-means 收敛时间的算法相比，小批 k-means 产生的结果通常只比标准算法稍差。

我们绘制了k=5-13时的聚类结果：



**4.3.4.3 选择K值**

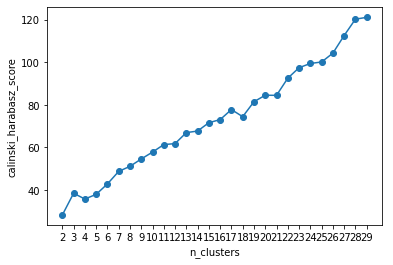
对于K-Means聚类算法来说，K值的选择是至关重要的，很多情况下K值的估计是非常困难的，对于像计算全部微信用户的交往圈这样的场景就完全的没办法用K-Means进行。对于可以确定K值不会太大但不明确精确的K值的场景，可以进行迭代运算，然后找出Cost Function最小时所对应的K值，这个值往往能较好的描述有多少个簇类。

对于我们本次作业中分析的项目，音乐的特征分类，我们主要采用了两个评价指标，分别是：calinski\_harabasz\_score评估系数和轮廓系数 silhouette\_score。

**calinski\_harabasz\_score评估系数**

CH指标通过计算类中各点与类中心的距离平方和来度量类内的紧密度，通过计算各类中心点与数据集中心点距离平方和来度量数据集的分离度，CH指标由分离度与紧密度的比值得到。从而，CH越大代表着类自身越紧密，类与类之间越分散，即更优的聚类结果。

我们通过调用sklearn中的metrics函数，计算出不同k值下K-Means模型的CH指数，结果如下图：

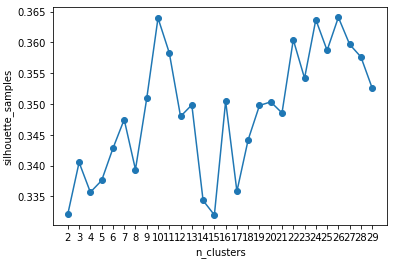


可以看到效果并不如我们预期的那样出现峰值，而是呈现随着k值增长，CH指数不断增加的趋势，这一方面说明CH指数评价指标并不适合我们的音乐特征数据，由之前的原始数据我们可以看到样本数据的特征分布并不呈现比较完美的分簇化趋势，而是中心集中，因此当k值接近样本数量时CH指数会上升至最高。另一方面也推动我们寻找其他的聚类评价指标，于是我们找到了另一种评价指标——轮廓系数。

**轮廓系数 silhouette\_score**

* 计算样本i到同簇其他样本的平均距离ai。ai 越小，说明样本i越应该被聚类到该簇。将ai 称为样本i的**簇内不相似度**。
* 簇C中所有样本的a i 均值称为簇C的簇不相似度。
* 计算样本i到其他某簇Cj 的所有样本的平均距离bij，称为样本i与簇Cj 的不相似度。定义为样本i的**簇间不相似度**：bi =min{bi1, bi2, ..., bik} bi越大，说明样本i越不属于其他簇。
* 根据样本i的簇内不相似度a i 和簇间不相似度b i ，定义样本i的轮廓系数

该值取值范围为[−1,1][-1, 1][−1,1]， 越接近1则说明分类越优秀。在sklearn中函数silhouette\_score()计算所有点的平均轮廓系数，而silhouette\_samples()返回每个点的轮廓系数。我们调用了sklearn中的metrics库函数计算不同k值下K-Means模型的轮廓系数，绘制图形如下：



可以看到，相较于CH指数，轮廓系数能够显著的显示某一K值的局部峰值，如图即K=10时K-Means算法聚类效果最好，我们可以设置K=10。

**4.3.4.4 聚类结果**

最后，我们可以根据聚类结果生成音乐信息分类情况，**为我们后续的热门音乐流派分析提供对比集**(以数量最多的一类为示例)：

class 6 [数量：76]：['来吧开整', 'waiting for love', '你长这样谁要', '叮叮叮', 'chuchacha', '将会面对什么样的结局', '花开的时候你就来看我', '酒醉的蝴蝶', '我怎么这么好看', '最美的期待', '学猫叫-完整剪辑版', '小可爱 （我的小可爱 今天有没有乖）', '123我爱你', '听我说谢谢你－李昕融', '就是不想长大！', '学猫叫-剪辑版', '火山用户创作的原声', '黄梅戏-慕容晓晓', '殇雪（剪辑版）', '三月里的小雨\_王恰恰', 'Make Some TikTok', '生日快乐', '@德华饰品创作的原声', '一曲红尘', '@我是苏朋友创作的原声', '宝贝宝贝我爱你（剪辑版）', '天在下雨我在想你', '一路向北-剪辑版', '捉泥鳅', '@✨🌟淡然一笑🌟✨创作的原声', '金久哲 - 干就完了（剪切版）', '你会爱我到什么时候-剪辑版', '敖包相会', '男人要有担当', '小小的太陽', '千年等一回', '站着等你三千年-高潮版', '醉千年', '女人是世界最美丽的花（剪辑版）', '@开心每一天💋创作的原声', 'Wrap Me In Plastic', '嘴巴嘟嘟-剪辑版', 'Delícia Tchu Tcha Tcha - Dj King Remix', '狂浪-花姐', '皮一下很开心(剪辑版)', '用户创作的原声~3', '@天启体验教育创作的原声', 'Getaway ', '多想抱抱你', '小星星~1', '小白兔遇上卡布奇诺', '爱情让我心痛DJ版-赵小南', '确认过眼神（撩妹撩汉版）', '你个小坏坏', '阳光彩虹小白马', '像个孩子', '小奶狗', '她扒拉我', 'MyLove一起去旅行', '我不会唱歌\_主歌版', '习惯你的好', '小棉袄儿（剪辑版）', '爱你三千遍（剪辑版）', '点歌的人-海来阿木', '歌曲来自柠檬啊', '@宾县＿＿仓少创作的原声', '忘情牛肉面', '皓皓最棒', '我带上你', '隔壁泰山(part.2)', '@Z小仙女创作的原声~1', '再见彩霞', '生日快乐~1', '就是这个范儿', '姐妹', '你打不过我吧']

**4.4 音乐风格分类**

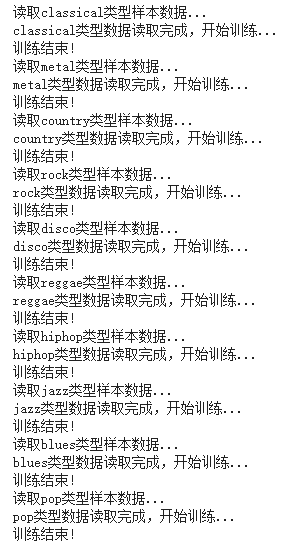
我们使用Marsyas提供的机器学习数据集，构建了一个利用隐马尔可夫模型的训练、预测音乐类型的分类器，并将测试数据导入进行风格分类。

**4.4.1 HMM模型训练**

**数据集**：我们将使用Marsyas提供的机器学习数据集，这是一个名为GTZAN的开源数据集。它是每30秒长的1000个音轨的集合。代表10种类型，每种包含100个音轨。所有音轨都是.wav格式的22050Hz单声道16位音频文件。

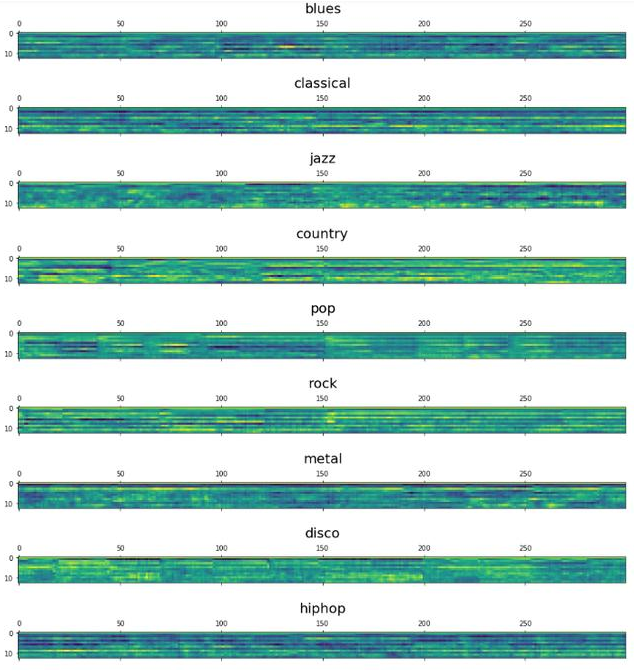
**训练特征**：我们需要找到一种简洁地表示歌曲波形的方法。Mel频率倒谱系数（MFCC）是一种很好的方法。MFCC获取信号的功率谱（power spectrum），然后使用Filter banks和离散余弦变换的组合来提取特征。

训练过程：



我们可以通过可视化每类歌曲的MFCC特征来以图片形式展示我们的分类器

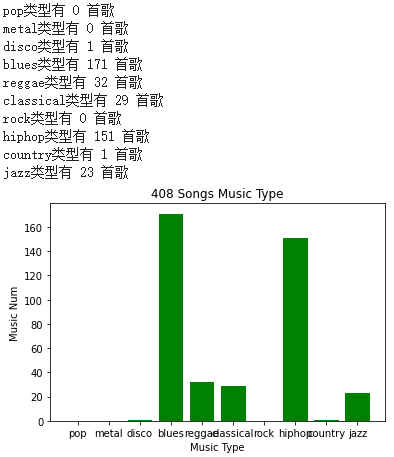
训练效果：



**4.4.2 音乐分类预测**

我们可以存储HMM模型的训练模型，这样我们就能对测试集中的音乐进行风格流派的分类预测。我们明确假设将为每个类构建单独的HMM模型。然后导入我们的408首音乐进行分类。

音乐分类结果：



可以看到分类的效果比较明显，对于抖音热门的400多首歌曲而言，他们的类型更加贴近blues和hiphop风格，这也比较符合我们的认知，这两类歌曲恰好以节奏明确，鼓点跳跃为主，我们绘制了类别最多的blues类歌曲词云，可以看到跟我们日常生活中的抖音热门音乐比较贴近：



**4.5 midi文件生成**

根据读取wav文件的帧，对其进行傅里叶变换获取音高信息，最终导入midi文件

**山妖原音乐**

[https://umlpicture.oss-cn-shanghai.aliyuncs.com/数据科学大作业/音乐/山妖.wav](https://umlpicture.oss-cn-shanghai.aliyuncs.com/%E6%95%B0%E6%8D%AE%E7%A7%91%E5%AD%A6%E5%A4%A7%E4%BD%9C%E4%B8%9A/%E9%9F%B3%E4%B9%90/%E5%B1%B1%E5%A6%96.wav)

**山妖mid文件音乐**

[https://umlpicture.oss-cn-shanghai.aliyuncs.com/数据科学大作业/音乐/山妖mp32mid.mp3](https://umlpicture.oss-cn-shanghai.aliyuncs.com/%E6%95%B0%E6%8D%AE%E7%A7%91%E5%AD%A6%E5%A4%A7%E4%BD%9C%E4%B8%9A/%E9%9F%B3%E4%B9%90/%E5%B1%B1%E5%A6%96mp32mid.mp3)

**wav融合**

根据不同的wav文件，我们综合其属性和特征，融合了属于自己的wav

[https://umlpicture.oss-cn-shanghai.aliyuncs.com/数据科学大作业/绝活终极.wav](https://umlpicture.oss-cn-shanghai.aliyuncs.com/%E6%95%B0%E6%8D%AE%E7%A7%91%E5%AD%A6%E5%A4%A7%E4%BD%9C%E4%B8%9A/%E7%BB%9D%E6%B4%BB%E7%BB%88%E6%9E%81.wav)

但是令人遗憾的是，这一曲子太过嘈杂，我们认为，这是因为在形成文件时，我们著重音乐波形和音调的变换，而忽略音乐本身的旋律性，缺少了节奏。因此我们通过midi映射，生成对应音调变化文件。

**最终音调**

[https://umlpicture.oss-cn-shanghai.aliyuncs.com/数据科学大作业/最终.mp3](https://umlpicture.oss-cn-shanghai.aliyuncs.com/%E6%95%B0%E6%8D%AE%E7%A7%91%E5%AD%A6%E5%A4%A7%E4%BD%9C%E4%B8%9A/%E6%9C%80%E7%BB%88.mp3)

这就是最终生成的音调变化，我们期望以该旋律为基础，加上适当旋律和和声，生成一首该项目的抖音BGM

**5. 结论**

经过前面所述的数据分析方法，我们得出以下结论：

1. 通过**4.2 波形分析**，我们可以发现，抖音BGM旨在用最低的时间成本，让听众迅速地记住这首歌，大部分视频的BGM都是直接采用副歌片段，在开头吸引人的注意力，之后采用重复，稍加变化的音调组合继续抓住人们的注意力，并通过歌曲的抑扬顿挫对应视频的内容，通过抖音自带的滑动机制，让人们沉迷于每十多秒就能带来的新鲜感之中。
2. 通过提取近400首的抖音热门BGM的众多特征，通过一系列降维的方法，我们可以发现这些歌曲从某方面来看是具有共通性的，在某种衡量水平上来看，呈现聚集的趋势，详见**4.3 音乐特征分析聚类**。从而进一步我们选取音乐风格为他们的分类标准，试图探究抖音热门BGM的音乐风格差异，详见**4.4 音乐风格分类**，我们可以很明显的看出，**blues风格**以及**hippop风格**的歌曲占有很大的比重，即这两种风格的音乐在抖音中更容易得到关注。
3. 通过对最热门的音乐**共同特征提取**和**聚类效果**，我们主观地给出一些对于热门音乐共通点的看法。随着信息技术的发展，短视频走上了时代的风口浪尖，对于一首歌来说，决定它是否好听很大程度就看他的副歌部分的“流行性”，这里的流行性指的是旋律有记忆点、易传唱，特别是副歌部分突出的特点。抖音中热曲也符合这个特点，它们中的大多数都有10s左右特别洗脑，特别抓耳的旋律，我们在听这些“**快餐式音乐**”的时候基本记不住主歌部分，歌名也不是用户的关注对象，这些音乐能够做到用最快的时间，以最洗脑的旋律在人耳朵中留下记忆点，这一现象也值得我们给予更多地关注和思考。
4. 通过wav文件读取特征值分析音高变化，学习抖音热门音乐音调变化，并通过midi文件融合生成音乐，通过类似的方法我们可以做到某种程度上**批量产出抖音热门音乐**，即使这些音乐在质量上参差不齐，不一定比得上音乐人制作的音乐，但是它胜在批量生产，简单便捷，可以**以低效的成本达到高回报的产出**。据我们所知，目前抖音内部还在使用人工推流的方式来筛选热门音乐，我们的工作可以从一些角度对他们有启发和帮助。

**6. 反思与不足**

1. 只研究了声学三要素（音调、音色、响度）相关的知识，没有研究音乐三要素（旋律、和声、节奏）相关的内容，并没有直接的midi文件，而由于水平受限，没有生成起初所期待的音乐。
2. 选取的聚类模型还有较大的评估优化空间，对于聚类算法以及参数的选用还可以更加细致地进行调参。
3. 准备的训练数据量较少，不太能够支撑完整模型的训练，训练后的模型召回率和精确率都比较低。

**7. 参考文献**

[pytorch神经网络——TORCHAUDIO TUTORIAL](https://pytorch.org/tutorials/beginner/audio_preprocessing_tutorial.html)

[MFCC音乐特征](https://blog.csdn.net/zouxy09/article/details/9156785/)

[R语言中的遗传算法](http://blog.fens.me/algorithm-ga-r/)

[sklearn聚类算法官网](https://scikit-learn.org/stable/modules/clustering.html)

[音乐收藏分析](https://www.christianpeccei.com/musicmap/)

[基于神经网络的音乐流派分类](https://medium.com/@navdeepsingh_2336/identifying-the-genre-of-a-song-with-neural-networks-851db89c42f0)

[使用隐马尔可夫模型进行音乐流派分类](https://www.toutiao.com/a6664179482661749256/)