

浙江大学

课程名称：数字视音频

项目名称：乐纹检索的实现

学 院：计算机学院

专 业：数字媒体

组 别：第五组

指导教师：邵健

2015 年 12 月 30 日

乐纹检索实验报告

一． 实验目的和要求

1. 熟悉和掌握音乐的取样压缩与特征提取技术。
2. 实现乐纹检索。

二． 开发环境

- Python3.5.0
- mysql
- Windows10

核心 python 库：

spicy numpy：用于数据科学计算，如施加短时傅里叶变换等。

Pydub：python 音频处理操作接口，用于音乐信号处理

Pymysql：支持 mysql 数据库

Matplotlib：数据绘图包（生成频谱图）

Hashlib：生成哈希码

Tkinter：python 界面处理 用于生成界面

三， 实验原理

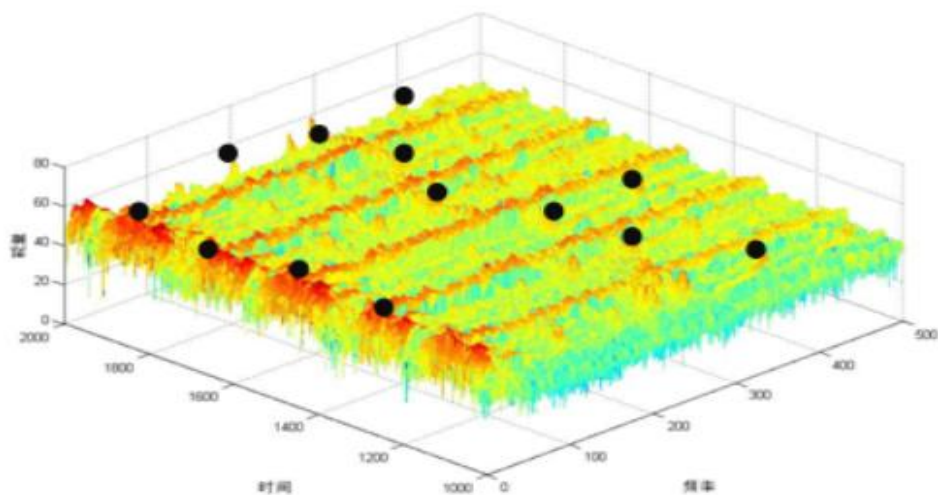
乐纹，就是可以代表一段音乐重要声学特征的基于内容的数字签名。具有鲁棒性(指未知音频能在经受比较严重的音频信号处理后仍然能够被识别出来)和区分性(即不同歌曲之间的指纹应该具有较大的差异，而同一音乐不同版本之间的指纹应该具有很小的差异)，基于乐纹的语音检索是基于内容的音乐检索的一种，可以利用用户提交的音乐片段，从存有大量音乐的数据库中找到匹配的音乐与哼唱检索相比，它适用范围更广，使用也更加方便。

提取指纹的算法很多，主要有三大类：echoprint, chromaprint 和 landmark 等，本文采用的是最为流行的 landmark 算法。

乐纹检索的主要步骤如下：

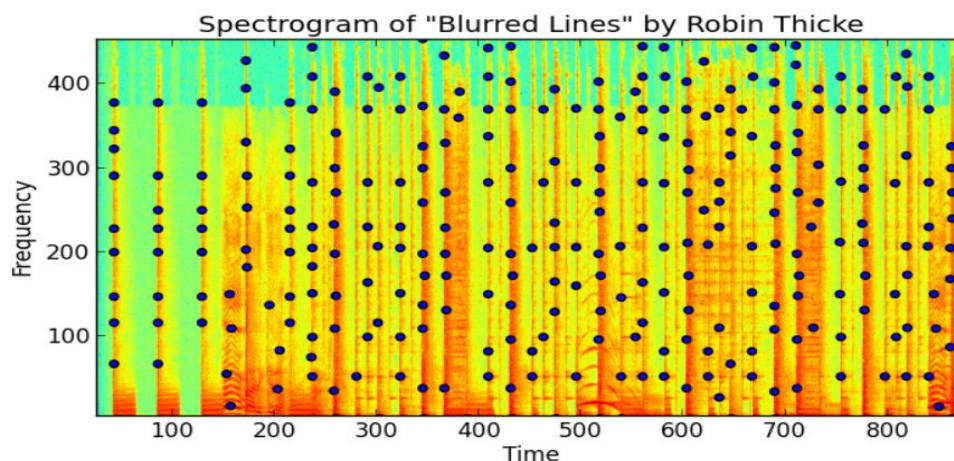
1，短时傅里叶变换

我们将输入的音乐片段进行归一化处理，转化成单通道，8k 采样率的 wav 格式，分帧后进行短时傅里叶变换，短时傅里叶变换是很多音频算法中的第一步，目的是将原始波形音乐由时域变换到频域。变换之后会得到一个频谱图，如图二所示。频谱图是一个三维图，X 坐标是时间，Y 坐标是频率，Z 坐标是能量。



2，提取特征点

特征点被定义为是频谱图中的一系列能量极大值点。能量极大值点的抗噪能力很强。求法多种多样，目的就是在二维平面中寻找峰值。通过调节参数可以控制每秒选取的 landmark 个数。一般情况下每秒保留 20~30 个点即可。



3，生成指纹

基本的思路是，对于 2 维频谱上的每一个峰值点，在固定大小的矩形范围内寻找其他峰值点，配成峰值点对，这些峰值点对的序列成为该片段的乐纹。

但是因为我们数据做了压缩，不同的歌曲和有可能有同样的峰值，因此我们结合时域值，将第一个峰值点的频率、第二个的频率和两点之间的时间差组成一个字符串并生成哈希，这样，指纹就由三部分构成，两个 landmark 的频率和时间差。同时每个指纹都有一个对应的时间，也即 landmark 的时间，表示这个指纹出现的时刻。

4，存入数据库

如果我们是原始音乐库构造指纹库，提取的指纹就放入指纹库。指纹库可以用散列表实现，每个表项表示相同指纹对应的音乐 id 和 time。如果是检索音乐，则利用提取的指纹访问指纹库。每个指纹都是一个整数，根据指纹的构造不同，可能需要 24 到 30 位不等。

5，检索

有了指纹库，当从用户传递的音乐片段到达后端之后，首先对该片段提取指纹，然后将所有的指纹与散列表中的指纹进行匹配，再依据相似度排序。

当找到匹配的指纹后，获得该指纹对应的音乐的 id 和该指纹在该音乐中出现的时刻 time。然后将提取的指纹对应的 time 减去从指纹库中获得的 time 得到一个时间差。最后将

这些 id 和 时间差进行排序，我们得到针对每个 id 及其对应的一系列的时间差。要检索的片段肯定来自于某一首完整音乐从某个时刻开始的片段，不能慢速或者快速播放，这样，它们的生成的指纹应该相同，则对应的时间差也应该相同。所以，我们在排序之后只需寻找含有最多相同时间差的音乐 id 即可。

四，实验步骤

1，数据库创建

Tables_in_fingerprint
fingerprints
songs

Field	Type	Null	Key	Default	Extra
song_id	mediumint(8) unsigned	NO	PRI	NULL	auto_increment
song_name	varchar(250)	NO		NULL	
file_sha1	binary(20)	NO		NULL	

Field	Type	Null	Key	Default	Extra
hash	char(10)	NO	PRI	NULL	
song_id	mediumint(8) unsigned	NO	PRI	NULL	
offset	int(10) unsigned	NO	PRI	NULL	

创建 fingerprint 指纹数据库。其中有两个表 songs 和 fingerprints。结构如图所示。

在 SONG_TABLE 中，我们创建 10 位字符串用于哈希来减少储存空间，因为我们获得的哈希值本身有 40 位，但是在识别过程中用不到这么多位也能达到相同的效果。OFFSET 则是用来记录特征提取的起始点的时间，一般不会对在储存音乐的第一秒直接开始提取特征。

2，读取 mp3 文件，返回通道，取样数，文件名哈希字符串

#读取 mp3 文件，返回通道，取样数，文件名哈希字符串

```
def read(filename):  
  
    audiofile = AudioSegment.from_mp3(filename)  
  
    data = np.fromstring(audiofile._data, np.int16)
```

```

channels = []

for chn in range(audiofile.channels):

    channels.append(data[chn::audiofile.channels])

fs = audiofile.frame_rate

return channels, audiofile.frame_rate, unique_hash(filename)

```

读取的文件有两个声道，但是此处只需保留一个，因为大多数乐曲的两个声道都是相同的，而且提取指纹也只需要一个声道中的信息即可。

3，根据频谱图生成乐纹

对音乐进行预处理，利用 `mlab.specgram` 生成频谱图，提取峰值对，再用峰值对中的两个点的频率和时间差合成的字符串生成哈希。

#生成指纹的函数

```

def fingerprint(channel_samples, Fs=DEFAULT_FS,

                wsize=DEFAULT_WINDOW_SIZE,

                wratio=DEFAULT_OVERLAP_RATIO,

                fan_value=DEFAULT_FAN_VALUE,

                amp_min=DEFAULT_AMP_MIN):

```

此处的参数与提取峰值点的限定条件有关，会影响峰值点的个数

```

arr2D = mlab.specgram(

    channel_samples,

    NFFT=wsize,

    Fs=Fs,

    window=mlab.window_hanning,

    noverlap=int(wsize * wratio))[0]

```

#将线性变换转换成对数函数

```

arr2D = 10 * np.log10(arr2D)

arr2D[arr2D == -np.inf] = 0

```

`#寻找局部最大值点并画出图像`

```
hash_list= get_2D_peaks(arr2D, plot=True, amp_min=amp_min)
```

plot 参数代表是否绘制峰值点图，amp_min 代表筛选峰值点的条件，振幅需要大于这个值

```
return hash_list
```

其中峰值点提取的部分代码如下：

`#寻找局部最大值点函数`

```
def get_2D_peaks(arr2D, plot=False, amp_min=DEFAULT_AMP_MIN):
```

```
    struct = generate_binary_structure(2, 1)
```

```
    neighborhood = iterate_structure(struct, PEAK_NEIGHBORHOOD_SIZE)
```

`# 寻找局部峰值点`

```
    local_max = maximum_filter(arr2D, footprint=neighborhood) == arr2D
```

```
    background = (arr2D == 0)
```

```
    eroded_background = binary_erosion(background, structure=neighborhood,
```

```
border_value=1)
```

```
    detected_peaks = local_max - eroded_background
```

`# 提取峰值点`

```
    amps = arr2D[detected_peaks]
```

```
    j, i = np.where(detected_peaks)
```

`# 筛选峰值点`

```
    amps = amps.flatten()
```

```
    peaks = list(zip(i, j, amps))
```

```
    peaks_filtered = [x for x in peaks if x[2] > amp_min]
```

4，识别与匹配

主要的过程就是读取音乐信息，获得该片段的指纹，从数据库中寻找匹配的哈希码。返回匹配的歌曲 id,并按照重复值排序。

我们这里主要解决了指纹对其的问题，即利用对于用正常速度播放的同一首音乐来说，

所有相关联的 offset 之间都有着相同的距离这一点，计算不同的 offset,将 offset 和指纹的元组插入 match 列表返回，以实现对于从不同地方开始取样的音乐的处理。

整个过程将 fingerprints 表遍历一遍，统计与样本中提取出的指纹的重合度，相同 offset 的重合指纹最多的歌曲即为样本片段所属歌曲。获得 id 后从 songs 表中读取相应的歌曲名。

#从数据库中获取匹配的歌曲

```
def match(printhead):  
    database.cur.execute("SELECT * FROM fingerprints;")  
    match = []  
    for i in database.cur:  
        for j in printhead:  
            if i[0] == j[0][-10:]:#检验指纹匹配  
                offset_diff = i[2] - j[1]  
                match.append((offset_diff, i[1]))#将时间差和指纹的元祖插入 match 列表  
    sid = getid(match)  
    database.cur.execute("SELECT song_name FROM songs WHERE song_id = %d;" %(sid))  
    songname = database.cur.fetchone()  
    print(songname)
```

#获取匹配歌曲的 id


```
def getid(match):  
    repeat = {}  
    for i in match:  
        repeat[i] = match.count(i)  
  
    sid = 0  
    most = 0  
    for i in repeat:  
        if repeat[i] > most:#排序获取重复值最多的 id  
            most = repeat[i]
```

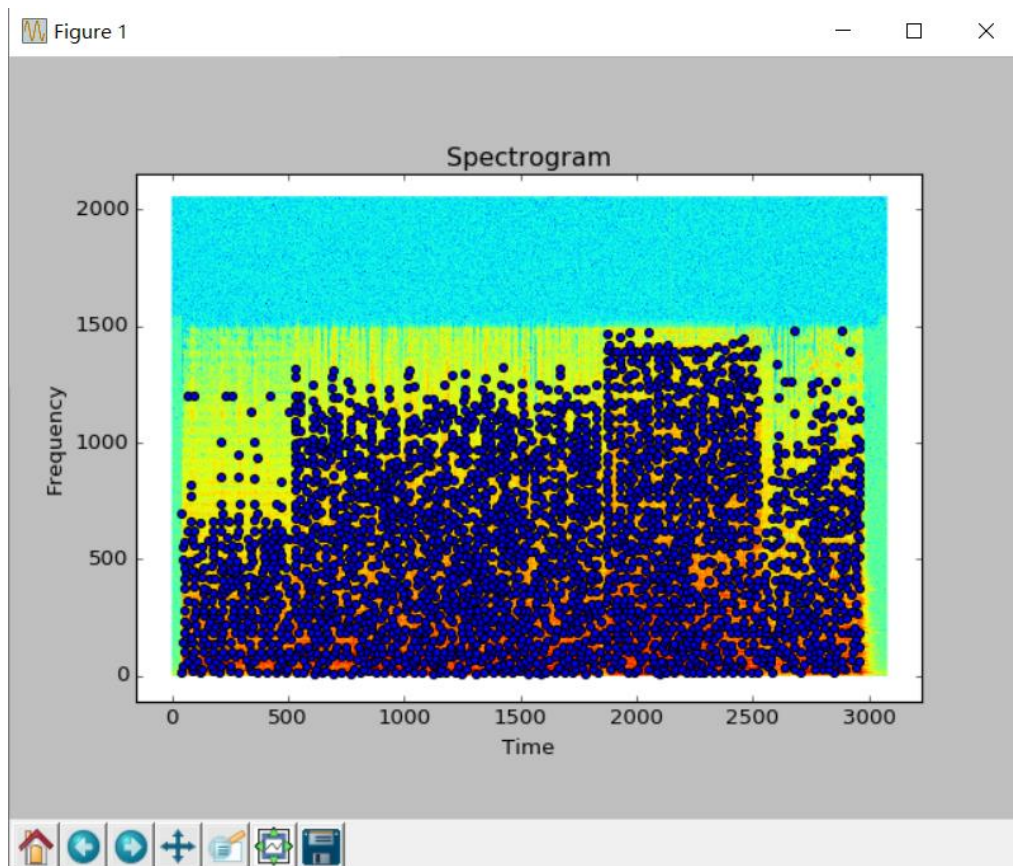


```
sid = i[1]

return sid
```

五，实验结果等

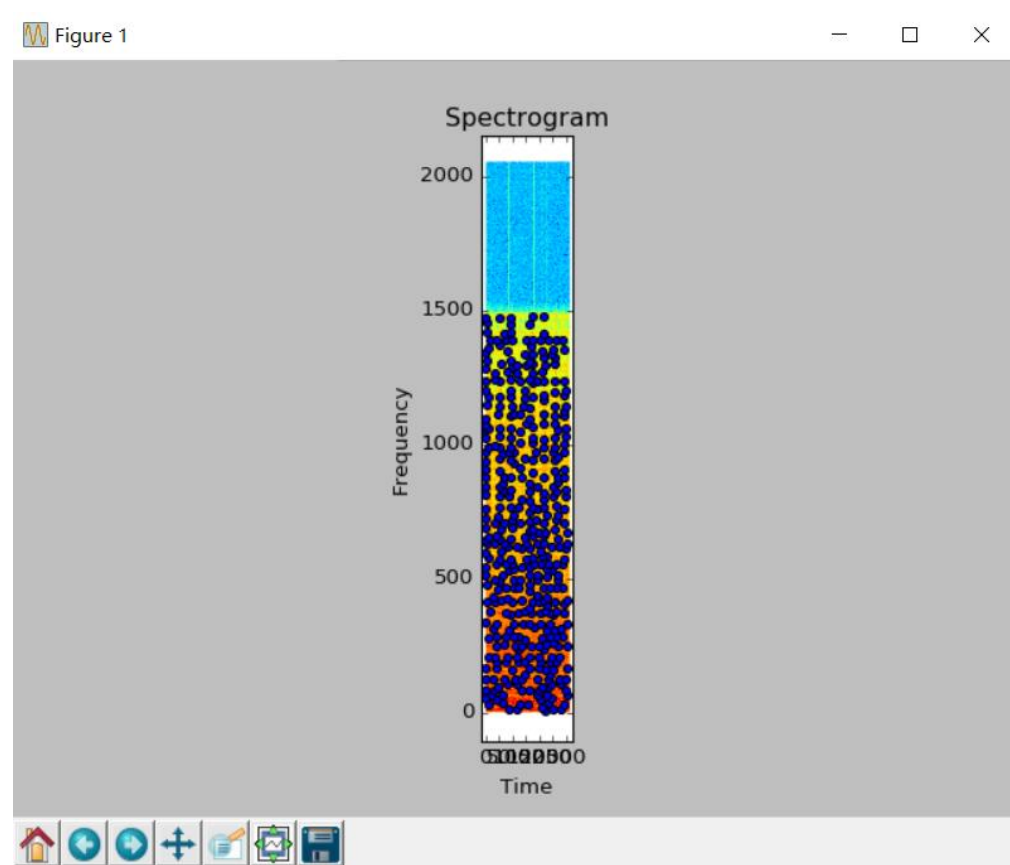
本次实验所用的样本为歌曲  Sairai.mp3 中提取的音乐片段，该歌曲的峰值点图如下



数据库中共有 10 首歌曲，且其中的指纹均已录入 fingerprints 表

song_id	song_name
1	Strawberry Swing.mp3
2	Dream.mp3
3	Fool's Day.mp3
4	Hard Candy.mp3
5	Hua Sui Yue.mp3
6	Oh My Love.mp3
7	Opening Version.mp3
8	Saika.mp3
9	Sairai.mp3
10	Sanctity.mp3

在命令行中运行 match.py，测试片段的峰值点图如下：



由于寻找歌曲的算法是测试片段中的指纹和数据库中的指纹各遍历一遍，所以速度取决于数据库的大小，10 首歌时，寻找到歌曲的时间在三秒以内。最后的结果如下。

```
C:\C:\WINDOWS\system32\cmd.exe

D:\课件\数字视音频处理\音乐\DM-musicprint>python match.py
('Sairai.mp3',)
```

六，感想和总结：

这次实验中我们小组使用 Shazam 公司的 landmark 算法实现了基本的乐纹检索功能。有以下几点值得总结：

- 1，善于使用已有的库、包等。这次实验中使用了很多 python 扩展包，包括 spacy、Pydub、Matplotlib 等，不但降低了编程难度也减少了代码数量。很多大的公司和 IT 大牛编写了不少可以实现多种功能的包，我们应该学习更好地使用这些包，“站在前人肩膀上”不断前进。
- 2，注意小组的协作和配合。这是一次小组作业，组内进行了明确的分工，每个人各司其职，都完成好了自己的那一份工作。以后我们走向工作岗位后，大多都是集体协作开发，培养良

好的团体意识和集体责任感对我们助益良多。

3, 要编写一个成功的程序, 必须要对程序所用到的算法非常了解。另外, 程序编译运行成功之后, 应该再看看有没有可以改进的地方, 比如提高性能, 增强可用性等。

小组分工：

赵智韬(3130104197)：代码编写, 小组展示

陈鸿泽(3130000167)：代码完善, 前期开题

孙峙梁(3130000912)：代码完善

李睿琪(3130000547)：文档撰写

张小璐(3130100086)：文档撰写

高泊宁(3120101209)：资料收集, 测试

徐炜迪(3130102117)：测试