实验报告 8

语音识别

范皓年 1900012739　信息科学技术学院

2020 年 12 月 10 日

目录

1 实验目的

1. 了解语音识别基本算法
2. 掌握树莓派音频接口的使用方法

2 实验原理

2.1 硬件基础

树莓派提供了音频输入和输出接口，其中音频输出使用 3.5mm 耳机接口，音频输入可以使用 USB 2.0 接口接入 USB 麦克风。实验采用免驱 USB 麦克风，直接插入树莓派上的 USB 2.0 即可，不需要再安装驱动。

2.2 设备检验

为验证麦克风是否正常工作，可以录制一段音频。

首先，使用

```
arecord -l
```

可以列出所有录音设备，一般输出如下：[1]

```
**** List of CAPTURE Hardware Devices ****
card 2: Device [USB PnP Sound Device], device 0: USB Audio [USB Audio]
Subdevices:1/1
Subdevice #0: subdevice #0
```

同样地，aplay -l 可以列出所有播放设备。

可执行 Linux 自带的录音/播放命令，测试硬件是否正常：

```
1 sudo arecord -D “plughw:1,0” -d 5 test.wav
2 aplay -D hw:1,0 test.wav
```

其中 hw 即为我们在1中的声卡 card 和设备 subdevice 信息。

树莓派中的 CPU 性能较差，而 GPU 较强大，omxplayer 是专门针对树莓派的 GPU 的播放器，支持硬件解码。这里也可以用 omxplayer 进行播放：

```
omxplayer -o local test.wav
```

在 Python 中执行录音命令需要 pyaudio 模块，要使用 PyAudio，首先使用 pyaudio.PyAudio() 实例化 PyAudio；要录制或播放音频，使用 pyaudio.PyAudio.open() 在设备上打开所需音频参数的流，即设置了 pyaudio.Stream：

```
1 CHUNK = 1024
2 FORMAT = pyaudio.paInt16
```

[1]注意其中的声卡和设备信息。在后续调用过程中，要根据自己的机器更改示例中接口。

```
CHANNELS = 2
RATE = 44100
p = pyaudio.PyAudio()
stream = p.open(format=FORMAT,
                channels=CHANNELS,
                rate=RATE,
                input=True,
                frames_per_buffer=CHUNK)
```

特别注意，如果循环进行读入输出，stream 不能放在循环之外。否则随着循环进行，流对象的内容就发生了溢出。

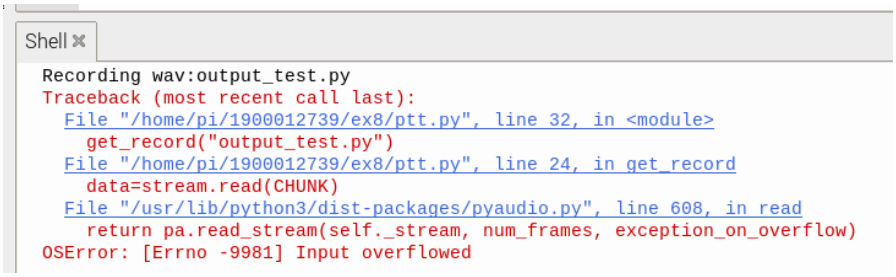

图 1: *Stream* 构造位置不合理导致的溢出

使用 pyaudio.Stream.write() 或 pyaudio.Stream.read() 录制或播放音频。需要注意的是，在”阻塞模式”，每个 pyaudio.Stream.write() 或 pyaudio.Stream.read() 阻塞直到操作完成；要动态生成音频数据或立即处理正在录制的音频数据，需要使用“回调”模式。使用 pyaudio.Stream.stop_stream() 暂停播放/录制，并 pyaudio.Stream.close() 终止流。最后，使用 pyaudio.PyAudio.terminate() 终止 portaudio 会话。录制的数据以.wav 格式保存，需要调用 wave 库：

```
wf = wave.open(filename, 'wb')
wf.setnchannels(CHANNELS)
wf.setsampwidth(p.get_sample_size(FORMAT))
wf.setframerate(RATE)
wf.writeframes(data)
```

2.3 音频识别算法 HMM-GMM 简介

语音识别任务通常需要将一段音频转化成相应的文字，从数学角度上看，也就是求一段音频属于哪段文字的概率最大。传统语音识别框架中，所谓声学模型就是把语音的声学特征分类对应到（解码）音素或字词这样的单元；语言模型接着把字词解码成一个完整的句子。定义声学模型输入为 $O = (o_1, o_2, ..., o_T)$，对应的句子 $W = (w_1, w_2, ..., w_N)$，语音识别的任务就是求概率 $P(W|O)$ 最大时对应的字序列 $W*$。要计算 $P(W|O)$，可以利用贝叶斯公式：

$$P(W|O) = \frac{P(O|W)P(W)}{P(O)} \tag{1}$$

其中 $P(O)$ 在计算时可以忽略，这样就得到了两部分 $P(O|W)P(W)$，分别对应于传统语音识别框架中的声学模型和语言模型，前者的任务是计算给定文字之后发出这段语音的概率；后者表示某一字词序列发生的概率。

在命令词识别任务中，只需要使用到声学模型，本实验采用传统的 HMM-GMM (hidden Markov model and Gauss mixture model 隐马尔可夫与高斯混合) 模型进行识别。一个 HMM-GMM 模型对应一个孤立词，首先对输入的语音进行分帧，对每帧计算 MFCC (Mel Frequency Cepstral Coefficents 梅尔频率倒谱系数) 特征，得到一组描述语言信号能量在不同频率范围的分布的特征向量，之后对 HMM-GMM 模型进行训练。解码过程一般采用 Viterbi 算法（用于解决多步骤每步多选择模型的最优选择问题或篱笆型图的最短路径问题），输入一组特征向量，对每帧用 GMM 计算隐藏状态的概率值，结合 HMM 的转移概率，用 Viterbi 算法进行路径搜索，得到最大概率值，这样就获得了最后的识别结果。

本实验只涉及最简单的案例——10 个独立词的识别。程序上，以测试集为例，音频的类别为文件名下划线后面的数字，例如 1_1.wav。

首先，我们要对采集到的音频数据进行数据编号预处理。输入为训练集路径或者测试集路径，预处理程序 gen_wavlist 分别得到音频文件字典 wavdict 及相应的标注字典 labeldict：

```
wavdict:
{'1_1': 'test_data\\1_1.wav', '1_10': 'test_data\\1_10.wav',
'1_2': 'test_data\\1_2.wav','1_3': 'test_data\\1_3.wav',
'1_4': 'test_data\\1_4.wav', '1_5': 'test_data\\1_5.wav',
'1_6': 'test_data\\1_6.wav', '1_7': 'test_data\\1_7.wav',
'1_8': 'test_data\\1_8.wav','1_9': 'test_data\\1_9.wav'}
labeldict:
{'1_1': '1', '1_10': '10', '1_2': '2',
 '1_3': '3', '1_4': '4', '1_5': '5', '1_6': '6',
'1_7': '7', '1_8': '8', '1_9': '9'}
```

之后的特征提取直接调用 python_speech_features 实现 MFCC 算法：

```
from python_speech_features import mfcc
# 特征提取，feat = compute_mfcc(wadict[wavid])
def compute_mfcc(file):
    fs, audio = wavfile.read(file)
    mfcc_feat = mfcc(audio)
    return mfcc_feat
```

我们利用 hmmlearn 工具包搭建 HMM-GMM，因为需要识别 10 个独立词，需要初始化 10 个独立的 HMM-GMM 模型，即一个 HMM-GMM 模型的集合 self.models：

```
class Model():
    def __init__(self, CATEGORY=None, n_comp=4,
            n_mix = 3, cov_type='diag', n_iter=1000):
        super(Model, self).__init__()
        self.CATEGORY = CATEGORY
        self.category = len(CATEGORY)
```

```
        self.n_comp = n_comp
        self.n_mix = n_mix
        self.cov_type = cov_type
        self.n_iter = n_iter
        # 关键步骤，初始化 models，返回特定参数的模型的列表
        self.models = []
        for k in range(self.category):
            model = hmm.GMMHMM(n_components=self.n_comp, n_mix = self.n_mix,
            covariance_type=self.cov_type, n_iter=self.n_iter)
            self.models.append(model)
```

这里我们简单地对几个参数进行阐述：

- CATEGORY: 所有标签的列表
- n_comp: 每个孤立词中的状态数
- n_mix: 每个状态包含的混合高斯数量
- cov_type: 协方差矩阵的类型
- n_iter: 训练迭代次数

在语音处理中，每个 HMM 对应于一个词 (word)，一个词由若干音素 (phoneme) 组成。一个词 (word) 表示成若干状态 (states)，每个状态 (state) 表示为一个音素；汉语的词一般由 5 个状态组成，英语的为 3 个。用混合高斯密度函数去表示每个状态的出现概率，只需要求出其均值和协方差就可以了。在 HMMGMM 模型中，如果观测序列是一维的，则观测状态的概率密度函数是一维的普通高斯分布。如果观测序列是 N 维的，则隐藏状态对应的观测状态的概率密度函数是 N 维高斯分布。高斯分布的概率密度函数参数可以用 表示高斯分布的期望向量，Σ 表示高斯分布的协方差矩阵。在 GaussianHMM 类中，“means”用来表示各个隐藏状态对应的高斯分布期望向量形成的矩阵，而“covars”用来表示各个隐藏状态对应的高斯分布协方差矩阵 Σ 形成的三维张量。参数 covariance_type，取值为”full”意味所有的 ,Σ 都需要指定。取值为“spherical”则 Σ 的非对角线元素为 0，对角线元素相同。取值为“diag”则 Σ 的非对角线元素为 0，对角线元素可以不同，“tied”指所有的隐藏状态对应的观测状态分布使用相同的协方差矩阵 Σ。

然后，用同一种类的数据训练特定的模型：

```
# 提取声学特征
mfcc_feat = compute_mfcc(wavdict[x])
# hmm-gmm 模型训练
model.fit(mfcc_feat)
```

模型训练完毕后，对待测试的数据分别用十个模型打分，选出得分最高的为识别结果。

```
# 提取声学特征
mfcc_feat = compute_mfcc(wavdict[x])
# 用模型打分
re = model.score(mfcc_feat)
```

3 实验内容

3.1 麦克风测试

在2.2中我们给出了使用树莓派中内置 linux 进行录音和播放的命令。我们借用这个命令组合进行麦克风的测试，如果在播放后能够得到录入的音频材料，则说明麦克风可以正常工作。[2]

经过测试我们发现这个过程是正常的。故而可以进行接下来的实验。

3.2 Press To Talk

注意这里的 GPIO 不能简单地使用上下跳沿进行触发，否则会出现断续。这里可以直接对输出的电平进行高低的判断。

如果发现按钮按下，就进行触发调用：

```
1 if GPIO.input(channel) == 0:
```

而后的调用是连续的：

```
1 while GPIO.input(channel) == 0:
2     data=stream.read(CHUNK)
3     wf.writeframes(data)
```

要特别注意，我们要把音频流相关的对象建立和函数调用过程放在录音的函数中，否则外部的流对象一直工作，将使得流对象中的内容发生溢出。

```
1  def record_test():
2      stream=p.open(format=FORMAT,channels=CHANNELS,
3                    rate=RATE,input=True,frames_per_buffer=CHUNK)
4      filename = 'test.wav'
5      wf = wave.open(filename, 'wb')
6      wf.setnchannels(CHANNELS)
7      wf.setsampwidth(p.get_sample_size(FORMAT))
8      wf.setframerate(RATE)
9      print("Recording wav:"+filename)
10     while GPIO.input(channel) == 0:
11         data=stream.read(CHUNK)
12         wf.writeframes(data)
13     print("Finished :b")
14     wf.close()
15     stream.stop_stream()
16     stream.close()
```

完整代码见A.1。

[2] 一个小问题：麦克风的物理滑动开关要打开

3.3 语音命令的训练与识别

利用3.2中给出的 press to talk 方法，我们连续录制 20 段命令音频，每 5 个一组，利用 trainDataRecord 进行名称编制：

```
record_list = []
def trainDataRecord():
    global record_list
    for i in range(20):
        filename = 'train_data/' + 'train_' + str(i//5+1) + '_' + str(i%5+1)
        record_list.append(filename)
```

从而可以得到一 4 个基本命令的训练集。

正式训练开始之前，要重新读取先前的音频资料。利用如下的 gen_wavlist 的函数进行文件读取，得到两个列表，列表的键相同，值分别为音频文件名，另一个为标签值：

```
def gen_wavlist(wavpath, mode):
    wavdict = {}
    labeldict = {}
    for (dirpath, _, filenames) in os.walk(wavpath):
        for filename in filenames:
            if filename.endswith('.wav'):
                filepath = os.sep.join([dirpath, filename])
                fileid = filename.strip('.wav')
                if mode in fileid:
                    wavdict[fileid] = filepath
                    # 获取文件的类别
                    label = (fileid.split(mode)[1]).split('_')[1]
                    labeldict[fileid] = label
    return wavdict, labeldict
```

利用如下的函数抽取训练集中命令音频的特征。

```
def train(self, wavdict=None, labeldict=None):
    for k in range(self.category):
        model = self.models[k]
        for x in wavdict:
            if labeldict[x] == self.CATEGORY[k]:
                mfcc_feat = compute_mfcc(wavdict[x])
                model.fit(mfcc_feat)
```

然后保存下来，

```
def save(self, path="models.pkl"):
    joblib.dump(self.models, path)
```

```
Result:  ['3', '4', '4', '4', '3', '1', '2', '2', '3', '1', '4', '2', '2', '1', '3', '1', '2', '1', '4', '3']

Label : ['3', '4', '4', '4', '3', '1', '2', '2', '3', '1', '4', '2', '2', '1', '3', '1', '2', '1', '4', '3']

Correct Number 1.0
>>>
```

图 2

使用 value 函数进行模型检验，很容易得到识别准确率达到一个很高的水平。

完成模型训练之后，我们进行实况检测，每次检测过程先进行一次录音，而后调用 controller 反馈模型识别结果：

```
elif mode == 'test':
    CATEGORY = ['1', '2', '3', '4']
    models = Model(CATEGORY=CATEGORY)
    try:
        models.load()
        while True:
            if GPIO.input(channel) == 0:
                record_test()
                models.controller()
    except KeyboardInterrupt:
        p.terminate()
        GPIO.cleanup()
        pass
```

其中 controller 为：

```
def controller(self):
    result = 0
    big_re = -10000000
    for k in range(self.category):
        subre = []
        label = []
        model = self.models[k]
        mfcc_feat = compute_mfcc('test.wav')
        # 生成每个数据在当前模型下的得分情况
        re = model.score(mfcc_feat)
        print(re)
        if(re > big_re):
            big_re = re
            result = k
            print(result)
    print('result',self.CATEGORY[result])
```

完整代码附于A.2。

3.4 语音控制系统初步

在上一个部分中，我们已经可以实现了命令的对应识别。接下来这部分想要进行语音控制是不困难的。

LED 灯的亮度变化利用 PWM 的占空比调节实现。将新输入的语音转换成命令集中对应的数字，分支控制 PWM 的占空比即可。

```
while True:
    if GPIO.input(channel) == 0:
        record_test()
        mode = models.controller()
        print(mode)
        if mode == 1:
            dc = 100
        elif mode == 2:
            dc = 0
        elif mode == 3:
            dc = dc * 2
            if dc > 100:
                dc = 100
        elif mode == 4:
            dc = dc // 2
        print(dc)
        pwm.ChangeDutyCycle(dc)
```

其余完整代码附于A.3

A 附录代码

A.1 Press To Talk

```
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
channel = 20
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel,GPIO.IN, GPIO.PUD_UP)

import pyaudio
import wave
CHUNK = 512
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
```

```
p = pyaudio.PyAudio()

index = 0
def get_record(filename):
    global index
    # stream initialized here, else eaily overflow
    stream=p.open(format=FORMAT,channels=CHANNELS,
                  rate=RATE,input=True,frames_per_buffer=CHUNK)
    filename = filename + '.wav'
    wf = wave.open(filename, 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(p.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    print("Recording wav:"+filename)
    while GPIO.input(channel) == 0:
        data=stream.read(CHUNK)
        wf.writeframes(data)
    print("Finished :b")
    index = index + 1
    wf.close()
    stream.stop_stream()
    stream.close()

record_list = []
def trainDataRecord():
    global record_list
    for i in range(20):
        filename = 'train_data/' + 'train_' + str(i//5+1) + '_' + str(i%5+1)
        record_list.append(filename)

try:
    trainDataRecord()
    while True:
        if GPIO.input(channel) == 0:
            get_record(record_list[index])
        if index > 20:
            break
except KeyboardInterrupt:
    p.terminate()
    GPIO.cleanup()
```

A.2 语音命令的训练与识别

```
from python_speech_features import mfcc
from hmmlearn import hmm
import os
import joblib
import numpy as np
from scipy.io import wavfile # used in compute_mfcc

import pyaudio
import wave
CHUNK = 512
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
p = pyaudio.PyAudio()

def compute_mfcc(file):
    fs, audio = wavfile.read(file)
    mfcc_feat = mfcc(audio)
    return mfcc_feat

class Model():
    def __init__(self, CATEGORY=None, n_comp=4, n_mix = 3, cov_type='diag', n_iter=10
        super(Model, self).__init__()
        self.CATEGORY = CATEGORY
        self.category = len(CATEGORY)
        self.n_comp = n_comp
        self.n_mix = n_mix
        self.cov_type = cov_type
        self.n_iter = n_iter
        # 关键步骤，初始化models，返回特定参数的模型的列表
        self.models = []
        for k in range(self.category):
            model = hmm.GMMHMM(n_components=self.n_comp, n_mix = self.n_mix,
            covariance_type=self.cov_type, n_iter=self.n_iter)
            self.models.append(model)

    def train(self, wavdict=None, labeldict=None):
        for k in range(self.category):
            model = self.models[k]
```

```
            for x in wavdict:
                if labeldict[x] == self.CATEGORY[k]:
                    mfcc_feat = compute_mfcc(wavdict[x])
                    model.fit(mfcc_feat)

    def value(self, wavdict=None, labeldict=None):
        result = []
        for k in range(self.category):
            subre = []
            label = []
            model = self.models[k]
            for x in wavdict:
                mfcc_feat = compute_mfcc(wavdict[x])
                # 生成每个数据在当前模型下的得分情况
                re = model.score(mfcc_feat)
                subre.append(re)
                label.append(labeldict[x])
            result.append(subre)
        # 选取得分最高的种类
        result = np.vstack(result).argmax(axis=0)
        # 返回种类的类别标签
        result = [self.CATEGORY[label] for label in result]
        print('Result: ',result, '\n')
        print('Label :', label,  '\n')

        totalnum = len(label)
        correctnum = 0
        for i in range(totalnum):
            if result[i] == label[i]:
                correctnum += 1
        print('Correct Number', correctnum/totalnum)

    def controller(self):
        result = 0
        big_re = -10000000
        for k in range(self.category):
            subre = []
            label = []
            model = self.models[k]
            mfcc_feat = compute_mfcc('test.wav')
            # 生成每个数据在当前模型下的得分情况
```

```
            re = model.score(mfcc_feat)
            print(re)
            if(re > big_re):
                big_re = re
                result = k
                print(result)
        print('result',self.CATEGORY[result])

    def save(self, path="models.pkl"):
        joblib.dump(self.models, path)

    def load(self, path="models.pkl"):
        self.models = joblib.load(path)

def gen_wavlist(wavpath, mode):
    wavdict = {}
    labeldict = {}
    for (dirpath, _, filenames) in os.walk(wavpath):
        for filename in filenames:
            if filename.endswith('.wav'):
                filepath = os.sep.join([dirpath, filename])
                fileid = filename.strip('.wav')
                if mode in fileid:
                    wavdict[fileid] = filepath
                    # 获取文件的类别
                    label = (fileid.split(mode)[1]).split('_')[1]
                    labeldict[fileid] = label
    return wavdict, labeldict

import RPi.GPIO as GPIO
GPIO.setwarnings(False)
channel = 20
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel,GPIO.IN, GPIO.PUD_UP)

def record_test():
    stream=p.open(format=FORMAT,channels=CHANNELS,
                  rate=RATE,input=True,frames_per_buffer=CHUNK)
    filename = 'test.wav'
    wf = wave.open(filename, 'wb')
    wf.setnchannels(CHANNELS)
```

```
    wf.setsampwidth(p.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    print("Recording wav:"+filename)
    while GPIO.input(channel) == 0:
        data=stream.read(CHUNK)
        wf.writeframes(data)
    print("Finished :b")
    wf.close()
    stream.stop_stream()
    stream.close()

if __name__ == "__main__":
    mode = 'test'
    CATEGORY = ['1', '2', '3', '4']
    if mode == 'train':
        wavdict, labeldict = gen_wavlist('train_data', 'train')
        models = Model(CATEGORY=CATEGORY)
        models.train(wavdict=wavdict, labeldict=labeldict)
        models.save()
    elif mode == 'test':
        CATEGORY = ['1', '2', '3', '4']
        models = Model(CATEGORY=CATEGORY)
        try:
            models.load()
            while True:
                if GPIO.input(channel) == 0:
                    record_test()
                    models.controller()
        except KeyboardInterrupt:
            p.terminate()
            GPIO.cleanup()
            pass
```

A.3 语音控制系统

```
from python_speech_features import mfcc
from hmmlearn import hmm
import os
import joblib
import numpy as np
from scipy.io import wavfile # used in compute_mfcc
```

```
import pyaudio
import wave
CHUNK = 512
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100

import RPi.GPIO as GPIO
GPIO.setwarnings(False)
channel = 20
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel,GPIO.IN, GPIO.PUD_UP)

LED = 26
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)

pwm = GPIO.PWM(LED, 50)
pwm.start(0)

p = pyaudio.PyAudio()

def compute_mfcc(file):
    fs, audio = wavfile.read(file)
    mfcc_feat = mfcc(audio)
    return mfcc_feat

class Model():
    def __init__(self, CATEGORY=None, n_comp=4, n_mix = 3, cov_type='diag', n_iter=10
        super(Model, self).__init__()
        self.CATEGORY = CATEGORY
        self.category = len(CATEGORY)
        self.n_comp = n_comp
        self.n_mix = n_mix
        self.cov_type = cov_type
        self.n_iter = n_iter
        # 关键步骤，初始化models，返回特定参数的模型的列表
        self.models = []
        for k in range(self.category):
            model = hmm.GMMHMM(n_components=self.n_comp, n_mix = self.n_mix,
```

```
            covariance_type=self.cov_type, n_iter=self.n_iter)
            self.models.append(model)

    def train(self, wavdict=None, labeldict=None):
        for k in range(self.category):
            model = self.models[k]
            for x in wavdict:
                if labeldict[x] == self.CATEGORY[k]:
                    mfcc_feat = compute_mfcc(wavdict[x])
                    model.fit(mfcc_feat)

    def value(self, wavdict=None, labeldict=None):
        result = []
        for k in range(self.category):
            subre = []
            label = []
            model = self.models[k]
            for x in wavdict:
                mfcc_feat = compute_mfcc(wavdict[x])
                # 生成每个数据在当前模型下的得分情况
                re = model.score(mfcc_feat)
                subre.append(re)
                label.append(labeldict[x])
            result.append(subre)
        # 选取得分最高的种类
        result = np.vstack(result).argmax(axis=0)
        # 返回种类的类别标签
        result = [self.CATEGORY[label] for label in result]
        print('Result: ',result, '\n')
        print('Label :', label,  '\n')

        totalnum = len(label)
        correctnum = 0
        for i in range(totalnum):
            if result[i] == label[i]:
                correctnum += 1
        print('Correct Number', correctnum/totalnum)

    def controller(self):
        result = 0
        big_re = -10000000
```

```
        global mode
        for k in range(self.category):
            subre = []
            label = []
            model = self.models[k]
            mfcc_feat = compute_mfcc('test.wav')
            # 生成每个数据在当前模型下的得分情况
            re = model.score(mfcc_feat)
            print(re)
            if(re > big_re):
                big_re = re
                result = k
                print(result)
        print('result',self.CATEGORY[result])
        return result+1

    def save(self, path="models.pkl"):
        joblib.dump(self.models, path)

    def load(self, path="models.pkl"):
        self.models = joblib.load(path)

def gen_wavlist(wavpath, mode):
    wavdict = {}
    labeldict = {}
    for (dirpath, _, filenames) in os.walk(wavpath):
        for filename in filenames:
            if filename.endswith('.wav'):
                filepath = os.sep.join([dirpath, filename])
                fileid = filename.strip('.wav')
                if mode in fileid:
                    wavdict[fileid] = filepath
                    # 获取文件的类别
                    label = (fileid.split(mode)[1]).split('_')[1]
                    labeldict[fileid] = label
    return wavdict, labeldict



def record_test():
    stream=p.open(format=FORMAT,channels=CHANNELS,
```

```
                    rate=RATE,input=True,frames_per_buffer=CHUNK)
    filename = 'test.wav'
    wf = wave.open(filename, 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(p.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    print("Recording wav:"+filename)
    while GPIO.input(channel) == 0:
        data=stream.read(CHUNK)
        wf.writeframes(data)
    print("Finished :b")
    wf.close()
    stream.stop_stream()
    stream.close()

if __name__ == "__main__":
    CATEGORY = ['1', '2', '3', '4']
    models = Model(CATEGORY=CATEGORY)
    models.load()
    try:
        dc = 0
        while True:
            if GPIO.input(channel) == 0:
                record_test()
                mode = models.controller()
                print(mode)
                if mode == 1:
                    dc = 100
                elif mode == 2:
                    dc = 0
                elif mode == 3:
                    dc = dc * 2
                    if dc > 100:
                        dc = 100
                elif mode == 4:
                    dc = dc // 2
                print(dc)
                pwm.ChangeDutyCycle(dc)
    except KeyboardInterrupt:
        p.terminate()
        GPIO.cleanup()
```

```
        pass
```