

实训项目说明书

基于GMM的说话人识别

|  |  |
| --- | --- |
| 院（系）名称 | 工学部 |
| 专业名称 | 智能科学与技术 |
| 学生姓名 | 吴冠峰、游塞雅  付孝慈、曲帅凱  李澳 |
| 指导教师 | 程步云 |

2020年6月7日

#### 基于GMM的说话人识别

#### 摘 要

说话人识别是在通过说话人的语音进行说话人的识别，是将测试需要识别的语音与对应库中的说话人语音模型进行匹配的一个过程。此次设计基于GMM模型进行说话人识别即声纹识别，首先对说话人的声音进行MFCC提取作为观察向量，利用GMM算法进行说话人的模型训练和测试识别。项目提供友好的人机交互GUI界面，更好的观测感受到识别的效率。经过测试该项目说话人识别准确率较高。

关键词：说话人识别；GMM；MFCC；GUI

Speech Recognition based on GMM

Abstract

Speaker recognition is to recognize the speaker through the speaker's voice. It is a process of matching the voice to be recognized by the test with the speaker's voice model in the corresponding database. This design is based on GMM model for speaker recognition, that is, voiceprint recognition. Firstly, MFCC is used to extract speaker's voice as observation vector, and GMM algorithm is used for speaker model training and test recognition. The project provides a friendly GUI interface for human-computer interaction to better observe and feel the recognition efficiency. After testing, the accuracy of speaker recognition is high.

Key words: Speaker recognition;GMM; MFCC; GUI

**目 录**

[1 绪论 1](#_Toc42590519)

[1.1 说话人识别概述及背景应用 1](#_Toc42590520)

[1.2 国内外研究现状 2](#_Toc42590521)

[2 项目介绍 4](#_Toc42590522)

[2.1 项目功能 4](#_Toc42590523)

[2.2 主要技术 4](#_Toc42590524)

[2.2.1 特征提取 4](#_Toc42590525)

[2.2.2 GMM模型的建立 6](#_Toc42590526)

[2.2.3 说话人识别测试 7](#_Toc42590527)

[3 详细设计与实现 9](#_Toc42590528)

[3.1 实现方式 9](#_Toc42590529)

[3.2 功能实现 9](#_Toc42590530)

[3.2.1 特征提取模块 9](#_Toc42590531)

[3.2.2 模型训练及测试模块 10](#_Toc42590532)

[3.2.3 GUI界面 12](#_Toc42590533)

[结论 13](#_Toc42590534)

[致谢 14](#_Toc42590535)

[参考文献 15](#_Toc42590536)

[附录A 代码 16](#_Toc42590537)

# 1 绪论

在日常生活中，形态语言视觉感觉等的协同合作，使得每一个人作为一个独立的个体，拥有独立的思想。每个人自出生以来的所见所闻所感形成了一个规模庞大的训练集，在不断的训练与学习中，我们隔很远听到有人喊自己的名字，便能知道是谁在说话，而机器想要模拟人的行为，做到‘听音辨人’的程度 ，却需要一系列复杂的学习与训练过程，这个过程就叫做‘说话人识别’技术。

## 说话人识别概述及背景应用

语音识别、图像处理、自然语音处理是近年来机器模仿人类生物体特征所取得的重大突破性进展，帮助人们解决了许多繁重或者重复性的工作，也给人们的生活提供许多便利。语音处理技术也在近三十年来得到了快速发展，在日常生活中取得广泛的应用。如手机语音输入法，只需要动动嘴巴，就可以写出一篇文档，节约了打字带来的时间成本；如声纹密码锁，通过识别说话人的音频音色，授予某一权限，既加强了密码的安全性，又避免了重复性设立各种密码而引起的遗忘等问题。

说话人识别和语音识别一样,都是通过对采集到的语音信号进行分析和处理,提取相应的特征或建立相应的模型,然后据此做出判断;但它与语音识别又有区别,其目的不是识别语音的内容,而是识别说话人的身份。早在上世纪40年代末期就有相关研究者开始进行相关技术的探索，主要应用于军事情报领域。其理论基础就是“每个人的说话特性都具有其独特的特征”

说话人识别从任务上来说，主要分为两部分。一是声纹确认技术，判断两句话是不是同一个人说的，是1比1的。二是声纹识别技术，判断某句话属于给定样本集里面谁说的，是一比N的。说话人识别通用原理为采集数据集、噪声抑制音频处理、特征提取、声音建模、识别匹配五部分。

随着大数据技术的发展，智能领域也迎来了井喷式发展，大量的数据支撑以及软硬件水平的提高，使得说话人识别在刑侦、隐私安全、智能机器人领域等都得到了广泛的应用。

刑侦安全----建立声纹识别库，反电信网络诈骗案

前几年电信诈骗案甚是猖獗，电信诈骗团伙凭借其流动性及隐蔽性使得破案变得极其困难。受骗家庭往往因被骗半生积蓄导致妻离子散，家破人亡。公安机关以技术对抗犯罪，利用大数据分析技术，监测网络环境，发现正在被诈骗的受害群体，及时拦截了一批诈骗电话 ，阻止了一批受害人上当受骗。同时以大数据、声纹技术应用为牵引，通过数据建模，部署全省公安机关对电信诈骗重点人员采集声纹，建立声纹信息库，对犯罪嫌疑人形成强大震慑。数据显示，去年四川省电信网络诈骗案件发案同比下降20.61%,群众损失同比下降68.77%。

隐私安全----使用声纹密码，方便快捷更安全

传统的密码形式是由数字字母字符构成，难以记忆又很容易通过技术手段破译，再加上随着个人账号的增多，密码遗忘的问题，极其的不方便。采用说话人识别录制的声纹密码很好的解决了这一问题，由于每个人的声音特征及说话习惯的不同，就可以无需记忆密码，凭借语音获取某一通过授权，便捷性得到很大的提升。

3）机器人识别----最强大脑 某期人机对战‘不能说的秘密’

复杂的声音环境下，人类能很轻松的分辨杂音，并能够同时接收到不同的人物传递的信息，机器做到这一点却很困难。然而在百度机器人与人类选手的比拼中，机器人能在嘈杂环境下，根据不稳定的只言片语，在音色相近，年龄相仿又刻意改变发音唱大合唱的人群中找到目标人物，与人类打成平手，可见实力的强劲。

## 1.2 国内外研究现状

从1970年代末到1980年代末，说话人识别的研究重点已经转移到声学特征参数的处理和新的模式匹配方法上。研究人员提出了LPC频谱系数，LSP频谱系数，感知线性预测系数，梅尔倒谱系数和其他说话人识别参数。此时，动态时间规整，矢量量化，隐马尔可夫模型，人工神经网络等技术已在语音识别领域得到广泛应用，并成为说话人识别的核心技术。自1990年代以来，GMM以其简单性，灵活性，有效性和更好的鲁棒性已成为独立于文本的说话人识别的主流技术，这使说话人识别的研究进入了一个新的阶段。 2000年，提出了通用背景模型和高斯混合模型GMM-UBM，为从实验室到实践的说话人识别做出了巨大贡献。在21世纪，采用传统的GMM-UBM方法，P.Kenny、n.dehak等人先后提出了联合因子分析和I向量模型，将说话人模型映射到低维子空间，克服了GMM-UBM系统中高斯分量独立性的局限性，提高了系统性能。为了进一步提高模型的分化能力，出现了相关的分化训练方法。另外，近年来随着深度机器学习在语音识别，图像处理等领域的快速发展和成功应用，基于深度学习的相关方法逐渐应用于说话人识别，取得了良好的效果。目前，基于时下声纹识别技术研究的前沿观点及商业化应用，有如下三大发展趋势。  
1、声纹识别研究朝着深度学习和端到端方向发展。近年来，声纹识别的研究趋势正在快速朝着深度学习和端到端方向发展，其中最典型的就是基于句子层面的做法。在网络结构设计、数据增强、损失函数设计等方面还有很多工作去做，还有很大的提升空间。  
2、提升声纹识别系统的短时语音情况。在实际应用中，由于对基于语音的访问控制需求的不断增长，提升声纹识别系统在短时语音情况下的性能变得尤为迫切。短时语音中说话人信息不足以及注册和测试语音的文本内容不匹配，对于主流的基于统计建模的声纹识别系统是一个严峻的挑战。  
3、改进现有的深度说话人学习方法。目前采用的深度说话人识别方法首先利用神经网络提取前端的帧级特征，然后通过池化映射获得可以表示说话人特性的段级向量，最后采用 LDA/PLDA 等后端建模方法进行度量计算。相对于传统的 i-vector 生成过程，基于深度学习的说话人识别方法优势主要体现在区分性训练和利用多层网络结构对局部多帧声学特征的有效表示上。如何进一步改进现有的深度说话人学习方法是现阶段的一个研究热点。

# 2 项目介绍

## 2.1 项目功能

本项目主要实现的功能是说话人的识别，即声纹识别，对说话人的声音进行特征提取，进行GMM模型的训练，得出说话人的GMM模型，之后进行测试录音，GMM进行识别，最终识别出说话人。

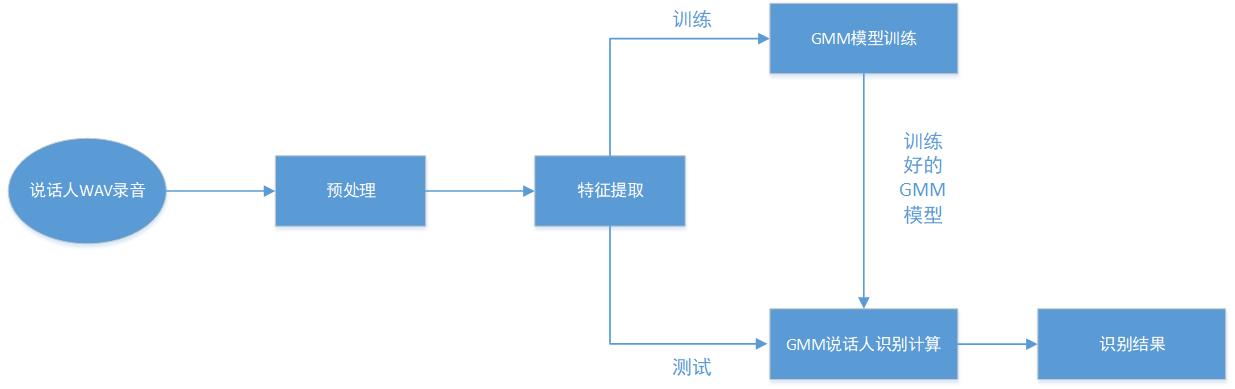


图 1 设计流程图

## 2.2 主要技术

本项目主要的技术语音的特征提取，GMM模型的训练，以及说话人的预测识别。

### 2.2.1 特征提取

项目主要提取的特征是MFCC（Mel倒谱系数），MFCC的提取过程如图2所示。

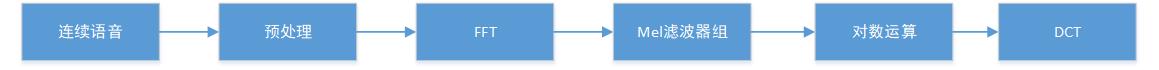


图 2 MFCC提取过程

预处理的过程主要包括有预加重，分帧和加窗。预加重主要的一个处理是将连续的语音通过一个滤波器：

(1)

式中的值一般介于0.9-1.0之间，通常会取0.97，通过预加重可以提升高频部分，使语音的频谱变得平坦，也可以除去声带和嘴唇的效应，从而补充语音受到发音系统而压制的高频率一部分，也是为了突出高频率的共振峰。将语音的频率保持在低频率到高频率的频率带中，能用同样的信噪比求频谱。

其次是分帧操作，主要应用的方法是连续分段，帧移主要取160，帧长一般在10-30ms之间，本项目主要取512。

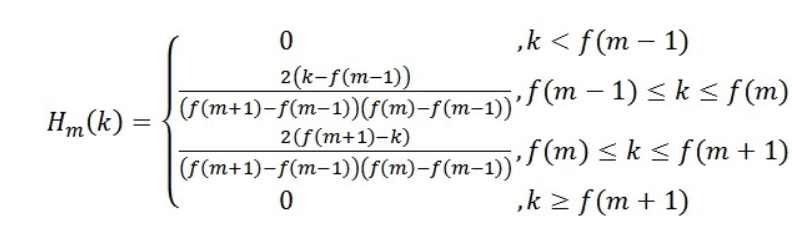
分帧后的操作是加窗，本项目选取汉明窗，将每一帧乘以汉明窗，以增加帧左右端的连续性。如式(2)所示：

, (2)

在经过预处理后，处理的语音会变成离散型的数据帧。

接下来进行快速傅里叶变换，因为语音在时域上的变换一般不会得出语音的特点，因此将语音转换为频域上的能量分布来看，不一样的能量分布特点就是不同语音的特性。因此在加窗后，每一帧还要经过FFT来得到在频谱上的能量分布。对分帧加窗后的各帧信号进行FFT得到各帧的频谱。并对语音的频谱通过计算平方得到语音的能量谱。

在得到能量谱之后通过三角滤波器组进行滤波，三角滤波器的频率响应定义如下列公式所示，其中f（m）是三角滤波器的中心频率。

 （3）

利用三角通滤波器的主要作用是对频谱进行平滑化以及除去谐波的影响，并且可以降低计算量。

之后开始运算滤波器组输出的对数能量。具体公式如下，其中Hm(k)为三角滤波器的频率响应。

 (4)

经过DCT（离散弦变换）得到MFCC，具体公式如下

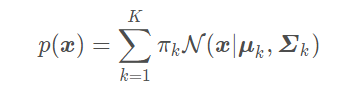
(5)

整体一个提取MFCC的过程可以总结为预加重，分帧和加窗，FFT，Mel滤波器组运算，对数运算后进行DCT。

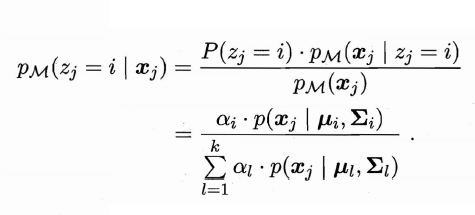
### 2.2.2 GMM模型的建立

对于特征向量的建模有很多模型，比如Kmeans，神经网络模型，GMM模型，GMM-UBM模型，本项目主要运用GMM模型进行训练。高斯混合模型（Gaussian mixture model，GMM）是单个高斯密度函数的发展，因为高斯混合模型可以靠近各种形状的概率密度分布，因此GMM模型被应用到各种领域比如语音识别领域别等并取得了理想的效果。

模型的建立过程主要有以下步骤，首先输入样本集D={x1,x2,….,xm},定义高斯混合成分的个数k。设有随机变量X，则混合高斯模型用下式表示：

 (6)

开始初始化高斯混合模型的分布的模型参数{(,,}，为了得到GMM模型的参数，一般有两种方法，一种是最大似然估计，一种是EM算法估计。本项目采用EM算法，E步根据下列式子计算出来Xj由各混合成分生成的后验概率，

 (6)

计算后得到：

*()* (7)

M步，估计单个高斯概率分布的参数，在E步中得到的后验概率是数据由第j个高斯概率分布生成的，每单个高斯概率分布都是平均的高斯分布，得到更新参数：

计算新均值向量：

(8)

计算新协方差矩阵：

(9)

计算新混合系数：

= (10)

将模型参数{(,,}进行更新，记为Yj，按照上述方法，如果有N个人，则会有Y1,Y2..Yn。

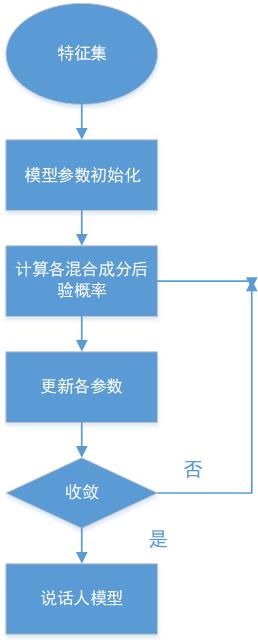


图 3 GMM训练流程

### 2.2.3 说话人识别测试

每个人所提取出来的MFCC不同，在说话人识别中，每个人的GMM模型都包含自己的语音特征，所以说话人识别就是测试人的语音和已经建立GMM模型的库中进行对比，并根据似然函数从里面选出最接近的说话人模型，具体过程如下：

对于N个说话人，我们以GMM模型Y1,Y2..Yn来代表。对于一段测试语音X，对比已有的高斯混合模型试图找到一个最大后验概率值的模型：

(11)

根据最大后验概率与贝叶斯定理上式可以改写为：

(12)

此时最大后验概率就变为了最大似然估计，将上式进行取对数得到：

(13)

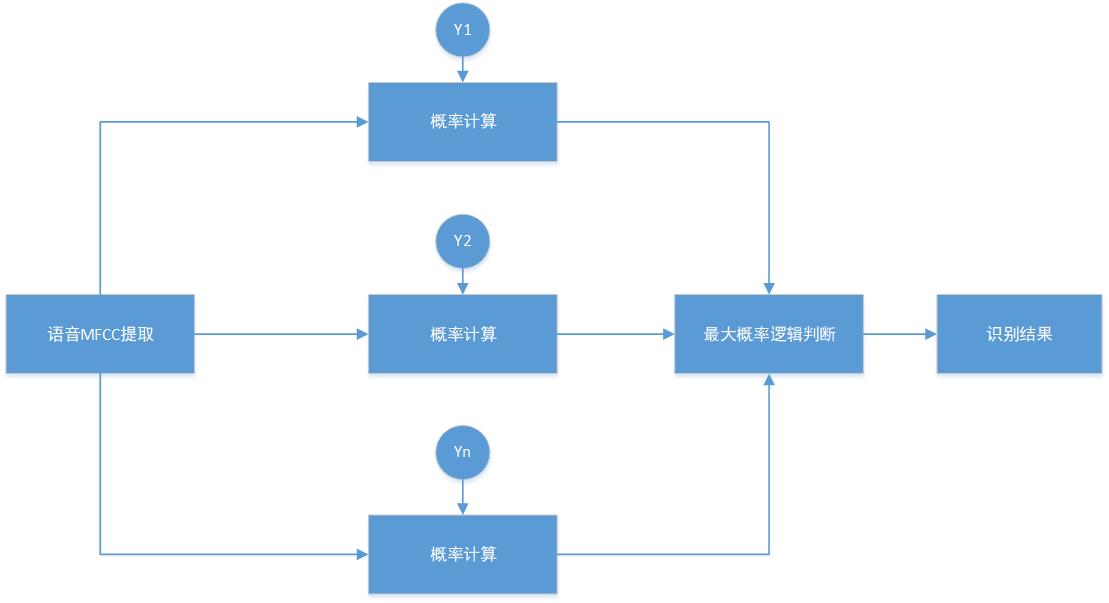


图 4 说话人测试流程

# 3 详细设计与实现

## 3.1 实现方式

主要实现方式是利用Python技术栈来实现说话人识别，中间借助Python强大的第三方拓展环境，以及友好的开发工具Pycharm，来实现快速开发。对于wav的录音利用的是Pyaduio进行录制，整体模块分为4个，分为MFCC提取模块，模型训练模块，测试模块以及GUI界面设置模块。四个模块进行类的封装，形成接口，从而实现整个项目的完成。

## 3.2 功能实现

### 3.2.1 特征提取模块

对于特征MFCC提取的设计，主要借助的是第三方库python\_speech\_features

来快速实现。

首先读取wav文件，qr, sig = wav.read(file)，qr是wav文件的采样率，sig是wav文件的内容，file是要读取的音频文件的路径。之后需要从python\_speech\_features调入mfcc和delta两个方法，具体实现方式如下：

mfcc\_feature = mfcc(  
 audio,  
 rate,  
 winlen = 0.025,  
 winstep = 0.01,  
 numcep = 20,  
 nfilt = 30,  
 nfft = 512,  
 appendEnergy = **True**)

其中aduio需要用来计算特征的音频信号，是一个数组，rate是我们用来工作的信号的采样率，winlen为分析窗口的长度，按秒计，默认0.025s(25ms)，winstep为连续窗口之间的步长，按秒计，默认0.01s（10ms），numcep为倒频谱返回的数量，默认13，本项目设置为20，nfilt - 滤波器组的滤波器数量，默认26，本项目设置为30， FFT的大小设置为默认512，appendEnergy设置为true，则将第0个倒谱系数替换为总帧能量的对数。之后利用deltas= delta(mfcc\_feature, 2)，返回一个大小为特征数量的numpy数组，包含有delta特征，每一行都有一个delta向量。

### 3.2.2 模型训练及测试模块

GMM模型训练这里的具体实现，我们用了两种方法，一种是调用sklearn第三方库中的GaussianMixture，另外一种是我们自己写了一个GMM模型。两种方法的实现形成一种对比，促进模型的改进。

首先说一下自己实现的GMM模型步骤：

1.对MFCC提取的特征进行预处理，将数据极差进行归一化处理

max\_ = x[:, i].max()  
min\_ = x[:, i].min()  
x[:, i] = (x[:, i] - min\_) / (max\_ - min\_)

2.之后初始化模型的参数，其中 shape为样本矩阵x的维数（样本数，特征数），k为模型的个数， mu, cov, alpha分别为模型的均值、协方差以及混合系数

n,d= shape  
self.mu = np.random.rand(k, d)  
self.cov = np.array([np.eye(d)] \* k) \* 0.1  
self.alpha = np.array([1.0 / k] \* k)

3. 然后对数据进行E步，计算每个模型对样本的响应度此时的，x 为样本矩阵，行数等于样本数，列数等于特征数，mu为均值矩阵， cov为协方差矩阵，alpha为各模型混合系数组成的一维矩阵，然后计算各模型下所有样本出现的概率矩阵prob，行对应第i个样本，列对应第K个模型。

prob = []  
 for j in range(n):  
 prob\_j = []

for k in range(self.K):  
 prob\_j.append(self.alpha[k] \* self.phi(x[j], self.mu[k], self.cov[k]))  
 s = sum(prob\_j)+1e-3  
 prob\_j = [item/s for item in prob\_j]  
 prob.append(prob\_j)  
 prob = np.array(prob)

4. 然后进行m步根据e步获得的验概率进行迭代更新

# 更新权重

self.alpha[i] = SUM / n   
 #更新均值向量  
 new\_mu = sum([gamma \* y for gamma, y in zip(gamma\_k, x)]) / SUM   
 self.mu[i] = new\_mu  
 #更新协方差阵  
 delta\_ = x - new\_mu self.cov[i] = sum([gamma \* (np.outer(np.transpose([delta]), delta)) for gamma, delta in zip(gamma\_k, delta\_)]) / SUM

5. 把e步m步整合起来构成em算法然后进行迭代

**for** i **in** range(times):  
 prob = self.expectation(x)  
 mu, cov, alpha= self.maximization(x, prob)

相比较再说一下使用利用Skearn库实现的效果，设置好相应的参数，进行模型的训练，对比两者得出的效果来说，Skearn库中的模型训练时间较快，准确率高，而我们自己写的训练的模型训练时间较长，准确率有待提高。

接下来是对说话人的测试，录制一小段wav，对这个说话人的语音特征进行提取，提取后进行与训练好的模型对比打分，与哪个模型吻合度高则为说话人的模型。具体实现利用的是GaussianMixture中score函数对MFCC进行打分，得分高的即为对应的说话人的模型。

### 3.2.3 GUI界面

视图界面所实现的功能有，首先点击进行声音的wav格式录制按钮，然后点击测试按钮，界面随即显示说话人是谁。具体界面如下：



图 5 GUI界面

界面利用的是Tkinter，Tkinter 是 Python 的标准 GUI 库。Python 使用 Tkinter 可以快速的创建 GUI 应用程序。由于 Tkinter 是内置到 python 的安装包中、只要安装好 Python 之后就能 import Tkinter 库、而且 IDLE 也是用 Tkinter 编写而成、对于简单的图形界面 Tkinter 还是能应付自如。Tk 和 Tkinter 可以在大多数的 Unix 平台下使用,同样可以应用在 Windows 和 Macintosh 系统里。

WAV格式录音利用的是PyAduio，PyAudio是Python开源工具包，由名思义，是提供对语音操作的工具包。提供录音播放处理等功能，可以视作语音领域的OpenCv。 PyAudio为跨平台音频I / O库 PortAudio 提供 Python 绑定。使用PyAudio，您可以轻松地使用Python在各种平台上播放和录制音频，例如GNU / Linux，Microsoft Windows和Apple Mac OS X / macOS。

# 结论

最终的实验测试表明该项目能够实现对说话人的识别，其精确度较高。通过实验发现对于自己写的模型还是有待改进，比如准确率的提高，训练时间的缩短，模型参数的调整等方面，以及对于数据的预处理方面也有待提高，对噪音的去除，运算矩阵中出现的Nan值和Infs值的消除有待改进。相比较来说Sklearn中的训练较好，这是促进我们向这个库学习改进的动力，下一步会多去学习库中的源码，来对我们的模型进行改进。

其次是GUI界面的改进，下一步我们会把它做成网页的形式，通过Django技术栈，Ajax和JSON前后端交互，实现线上上传数据，训练模型，测试，让更多人体验和激发学习的动力。

# 致谢

此次的实验项目得以完成，首先要衷心的感谢导师程步云对我们的教导和实验期间的对于项目的指导。没有程步云老师的启发和这两年的教导，是没有我们今天的努力成果，再次要感谢我们的导师程步云先生。同时，也要感谢本论文所引用的各位学者的期刊。

# 参考文献

[1]周志华.机器学习[M].清华大学出版社:北京,2016-1-1:206-211.

[2]吴慧玲,杜成东,毛鹤.基于GMM 的说话人识别算法的研究与应用[J].现代计算机,2014,14:31-35.

[3]刘士.基于 GMM 的声纹识别技术的研究[D].成都:电子科技大学,2012.

# 附录A 代码

**import** numpy **as** np  
**from** sklearn **import** preprocessing  
**from** python\_speech\_features **import** mfcc  
**from** python\_speech\_features **import** delta  
**class** FeaturesExtractor:  
 **def** \_\_init\_\_(self):  
 **pass  
 def** extract\_features(self, audio, rate):  
 mfcc\_feature = mfcc(  
 audio,  
 rate,  
 winlen = 0.025,  
 winstep = 0.01,  
 numcep = 20,  
 nfilt = 30,  
 nfft = 512,  
 appendEnergy = **True**)  
  
 mfcc\_feature = preprocessing.scale(mfcc\_feature)  
 deltas = delta(mfcc\_feature, 2)  
 double\_deltas = delta(deltas, 2)  
 combined = np.hstack((mfcc\_feature, deltas, double\_deltas))  
 **return** combined  
 **def** accelerated\_get\_features\_vector(self, input\_wave\_file, audio, sound\_rate):  
 **try** :  
 **return** self.extract\_features(audio, sound\_rate)  
 **except**:  
 print(**"不能找到文件"**, input\_wave\_file.split(**'.'**)[0])  
 **return** np.array([])

**import** os  
**import** pickle  
**import** warnings  
**import** numpy **as** np  
**from** sklearn.mixture **import** GaussianMixture  
**from** FeaturesExtractor **import** FeaturesExtractor  
**from** scipy.io.wavfile **import** read  
**class** trainer():  
 **def** mt(self,inputwav):  
 warnings.filterwarnings(**"ignore"**)  
 features = np.asarray(())  
 features\_extractor = FeaturesExtractor()  
 sr, audio = read(inputwav)  
 vector = features\_extractor.accelerated\_get\_features\_vector(sr, audio, 8000)  
 **if** features.size == 0:  
 features = vector  
 **else**:  
 **try**:  
 features = np.vstack((features, vector))  
 **except**:  
 print(**"错误: Shape mismatch"**)  
 gmm = GaussianMixture(n\_components = 16, max\_iter = 200, covariance\_type=**'diag'**, n\_init = 3)  
 gmm.fit(features)  
 picklefile = inputwav.split(**"."**)[0] + **".gmm"  
 with** open(picklefile, **'wb'**) **as** gmm\_file:  
 pickle.dump(gmm, gmm\_file)  
qn =trainer()  
qn.mt(**"李澳.wav"**)  
**import** pickle  
**import** warnings  
**import** numpy **as** np  
**from** FeaturesExtractor **import** FeaturesExtractor  
**from** scipy.io.wavfile **import** read  
**from** GMM\_EM **import** Gmmly  
**class** trainer():  
 **def** mt(self,inputwav):  
 warnings.filterwarnings(**"ignore"**)  
 features = np.asarray(())  
 features\_extractor = FeaturesExtractor()  
 GMM\_two = Gmmly()  
 sr, audio = read(inputwav)  
 vector = features\_extractor.accelerated\_get\_features\_vector(sr, audio, 8000)  
 **if** features.size == 0:  
 features = vector  
 **else**:  
 **try**:  
 features = np.vstack((features, vector))  
 **except**:  
 print(**"错误: Shape mismatch"**)  
 gmm = GMM\_two.gmm\_em(features,2,10)  
  
 picklefile = inputwav.split(**"."**)[0] + **".gmm"  
 with** open(picklefile, **'wb'**) **as** gmm\_file:  
 pickle.dump(gmm, gmm\_file)  
qn =trainer()  
qn.mt(**"李澳.wav"**)  
**import** copy  
**import** numpy **as** np  
**from** scipy.stats **import** multivariate\_normal  
**class** Gmmly:  
 **def** \_\_init\_\_(self):  
 **pass****def** scale\_data(self,x):  
 **for** i **in** range(x.shape[1]):  
 max\_ = x[:, i].max()  
 min\_ = x[:, i].min()  
 x[:, i] = (x[:, i] - min\_) / (max\_ - min\_)  
 **return** x**def** init\_params(self,shape, k):  
 n,d= shape  
 self.K=k  
 self.mu = np.random.rand(k, d)  
 self.cov = np.array([np.eye(d)] \* k) \* 0.1  
 self.alpha = np.array([1.0 / k] \* k)  
 **return** self.mu, self.cov, self.alpha**def** phi(self,Y, mu\_k, cov\_k):  
 norm = multivariate\_normal(mean=mu\_k, cov=cov\_k)  
 **return** norm.pdf(Y)  
  
**def** expectation(self,x,):  
 *# 样本数，模型数* n, k = x.shape[0], self.alpha.shape[0]  
 a=self.mu  
 b=self.cov  
 *# 计算各模型下所有样本出现的概率矩阵prob，行对应第i个样本，列对应第K个模型* prob = []  
 **for** j **in** range(n):  
 prob\_j = [] *# 依次求每个样本对K个分模型的响应度* **for** k **in** range(self.K):  
 aa=self.alpha[k] \* self.phi(x[j], self.mu[k], self.cov[k])  
 prob\_j.append(self.alpha[k] \* self.phi(x[j], self.mu[k], self.cov[k]))  
 s = sum(prob\_j)+1e-3  
 prob\_j = [item/s **for** item **in** prob\_j]  
 prob.append(prob\_j)  
 prob = np.array(prob)  
 **return** prob**def** maximization(self,x, prob):  
 *# 样本数，特征数* n, d = x.shape  
 *# 模型数* old\_alpha = copy.copy(self.alpha)  
 k = prob.shape[1]  
  
 *# 初始化模型参数* mu = np.zeros((k, d))  
 cov = []  
 alpha = np.zeros(k)  
  
 *# 更新每个模型的参数* **for** i **in** range(k):  
 gamma\_k = prob[:, i]  
 SUM = np.sum(gamma\_k)+1e-4  
 *# 更新权重* self.alpha[i] = SUM / n *# 更新权重  
 # 更新均值向量* new\_mu = sum([gamma \* y **for** gamma, y **in** zip(gamma\_k, x)]) / SUM *# 1\*d* self.mu[i] = new\_mu  
 *# 更新协方差阵* delta\_ = x - new\_mu *# n\*d* self.cov[i] = sum([gamma \* (np.outer(np.transpose([delta]), delta)) **for** gamma, delta **in** zip(gamma\_k, delta\_)]) / SUM *# d\*d* alpha\_delta = self.alpha - old\_alpha  
 gamma\_all\_final = prob  
 **return** self.mu, self.cov, self.alpha, gamma\_all\_final**def** gmm\_em(self,dataset,k, times):  
 *# 数据归一化处理* x = self.scale\_data(dataset)  
 *# 初始化模型参数* mu, cov, alpha = self.init\_params(x.shape, k)  
 *# 迭代模型参数* **for** i **in** range(times):  
 print(**"这是第{0}次"**,i)  
 prob = self.expectation(x)  
 mu, cov, alpha, gamma = self.maximization(x, prob)  
 **return** gamma  
  
**import** os  
**import** pickle  
**import** warnings  
**import** numpy **as** np  
**from** FeaturesExtractor **import** FeaturesExtractor  
**from** scipy.io.wavfile **import** read  
  
  
**class** spk\_id():  
 **def** person\_identifer(self):  
 warnings.filterwarnings(**"ignore"**)  
  
  
 modelpath = **"SpeakerModels/"** db = {}  
 **for** fname **in** [fname **for** fname **in** os.listdir(modelpath) **if** fname.endswith(**'.gmm'**)]:  
 speaker = fname.split(**'.'**)[0]  
 model = pickle.load( open(os.path.join(modelpath, fname), **'rb'**))  
 db[speaker] = model  
  
  
 error, total\_sample = 0, 0  
  
  
 features\_extractor = FeaturesExtractor()  
 sr, audio = read(**"output.wav"**)  
 vector = features\_extractor.accelerated\_get\_features\_vector(sr, audio, 8000)  
  
  
 **if** vector.shape != (0,):  
 print(vector.shape)  
 total\_sample += 1  
 log\_likelihood = {}  
 m = {}  
 **for** speaker, model **in** db.items():  
 gmm = model  
 scores = np.array(gmm.score(vector))  
 log\_likelihood[speaker] = round(scores.sum(), 3)  
 m[speaker] = scores  
  
 max\_log\_likelihood = max(log\_likelihood.values())  
 keys, values = list(log\_likelihood.keys()), list(log\_likelihood.values())  
 winner = keys[values.index(max\_log\_likelihood)]  
  
 **return** winner  
  
  
**import** pyaudio  
**import** wave  
**from** tqdm **import** tqdm  
  
**class** py\_wav():  
  
 **def** record\_audio(self,wave\_out\_path,record\_second):  
 CHUNK = 1024  
 FORMAT = pyaudio.paInt16  
 CHANNELS = 2  
 RATE = 44100  
 p = pyaudio.PyAudio()  
 stream = p.open(format=FORMAT,  
 channels=CHANNELS,  
 rate=RATE,  
 input=**True**,  
 frames\_per\_buffer=CHUNK)  
 wf = wave.open(wave\_out\_path, **'wb'**)  
 wf.setnchannels(CHANNELS)  
 wf.setsampwidth(p.get\_sample\_size(FORMAT))  
 wf.setframerate(RATE)  
 print(**"\* recording"**)  
 **for** i **in** tqdm(range(0, int(RATE / CHUNK \* record\_second))):  
 data = stream.read(CHUNK)  
 wf.writeframes(data)  
 print(**"\* done recording"**)  
 stream.stop\_stream()  
 stream.close()  
 p.terminate()  
 wf.close()

**from** tkinter **import** \*  
**from** pywav **import** py\_wav  
**from** SpeakerIdentifier1 **import** spk\_id  
  
  
spk\_per = spk\_id()  
luzhi = py\_wav()  
  
**def** va():  
 luzhi.record\_audio(**"output.wav"**,record\_second=9)  
**def** da():  
 **global** a  
 a = spk\_per.person\_identifer()  
 b = Label(root, text=a)  
 b.pack()  
  
root = Tk()  
root.title(**'说话人识别'**)  
root.geometry(**'200x300'**)  
Button(root, text=**"录音"**, command=va,height=3,width=3).place(x=75, y=130)  
Button(root, text=**"识别"**, command=da).place(x=75, y=90)  
w = Label(root, text=**"说话人为："** )  
w.pack()  
root.mainloop()