# 摘要

目前说话人识别研究主要是基于专业录音设备和处理机，来提取用户的语音中的特征并训练模型和识别确认处理。主要适用于大型安全领域。整套解决方案的可移动性不强，且不利于大规模普及和部署。因此本课题旨在研究开发出一个面向智能手机的说话人识别系统，在智能手机上获取用户语音，并在手机上提取特征并训练模型和识别确认处理。从而实现说话人识别方案的大规模普及和快速部署。本课题开发的说话人识别系统主要面向Android平台手机用户，使用基于HMM隐马尔科夫模型的HTK工具箱，通过文本相关的方式，要求用户先录制文本相关的语音以进行特征提取和训练，供之后用户再次录音并进行特征提取以完成说话人识别操作。

关键字：智能手机，说话人识别，特征提取，训练模型，Android

# Abstract

Speaker verification research at present is mainly based on the professional recording equipment and processor, to extract the features in the user's voice and training model and processing verification. It is mainly suitable for large areas of security. The mobility of the whole solution is not strong, and not conducive to large-scale popularization and deployment. Therefore, the purpose of this subject is to develop a speaker verification system for the smartphone, get the user voice and extract the features and training model and processing verification in smart phone. So it can realize the mass popularity and rapid deployment of the speaker verification system. The speaker verification system developed in this project is for the Android platform mobile phone users.

Keywords：Smart phone, speaker verification, feature extraction, training model, Android

目录

[1 背景 1](#_Toc421395430)

[1.1 课题背景 1](#_Toc421395431)

[1.2 开发背景 1](#_Toc421395432)

[1.3 研究目的和意义 2](#_Toc421395433)

[1.4 论文组织结构 2](#_Toc421395434)

[2 文献综述 3](#_Toc421395435)

[2.1 识别系统结构 3](#_Toc421395438)

[2.2 隐马尔科夫模型(HMM) 5](#_Toc421395439)

[2.3 HTK工具箱 5](#_Toc421395440)

[2.4 Android开发 7](#_Toc421395441)

[2.5 SQLite数据库以及ORM框架 9](#_Toc421395442)

[2.6 Apache-commons-io包 10](#_Toc421395443)

[2.7 JNI 11](#_Toc421395444)

[3 系统需求分析 12](#_Toc421395445)

[3.1 系统概述 12](#_Toc421395449)

[3.2 模块功能描述 12](#_Toc421395450)

[4 详细设计 14](#_Toc421395451)

[4.1 总体设计 14](#_Toc421395456)

[4.2 体系结构设计 14](#_Toc421395457)

[4.3 数据结构设计 15](#_Toc421395458)

[4.4 HTK封装设计 17](#_Toc421395459)

[4.5 开发平台及开发工具 18](#_Toc421395460)

[5 功能和实现 20](#_Toc421395461)

[5.1 数据模型 20](#_Toc421395467)

[5.2 HTK封装实现 24](#_Toc421395468)

[5.3 用户登陆 25](#_Toc421395469)

[5.4 主界面 28](#_Toc421395470)

[5.5 用户管理 29](#_Toc421395471)

[5.6 阈值设置 32](#_Toc421395472)

[5.7 修改密码 33](#_Toc421395473)

[5.8 说话人训练 35](#_Toc421395474)

[5.9 说话人测试 38](#_Toc421395475)

[6 实验和测试 43](#_Toc421395476)

[6.1 实验遇到的问题和解决方法 43](#_Toc421395483)

[6.1 43](#_Toc421395489)

[7 展望与总结 45](#_Toc421395490)

[7.1 总结 45](#_Toc421395498)

[7.2 展望 45](#_Toc421395499)

[参考文献 46](#_Toc421395500)

[致谢 47](#_Toc421395501)

## 背景

### 课题背景

说话人识别最初被广泛应用是在司法领域，用于帮助对嫌疑人的查证或判定罪犯，经过不断发展后进入安保和军事领域，用于一些机密场所出入控制，机要设备的使用控制和战场监听等等。

出此之外，说话人识别还有着广阔的市场应用前景。例如在通信和互联网尤其是新兴的移动互联网领域，说话人识别技术可以应用于语音拨号，电话银行，信息服务，安全控制，账户登录，E-mail，即时通信等。

说话人识别是一种生物认证技术。通过语音信号中的波形变化反映说话人生理和应为上的特征，并根据特征识别说话人。这些特征涉及到说话人的年龄，性别，感情，种族等等。

说话人识别的大致过程是首先录入说话人的语音样本，提取其中的语音特征并保存以待应用。在识别时将待测试的语音的特征与保存的语音特征做对比，从而确定说话人身份。

### 开发背景

#### 开发平台的选择

本课题开发的说话人识别系统是面向Android平台手机用户。

Android是一种基于Linux的开源的自由的操作系统，主要使用于智能手机，平板电脑，相机，游戏机，电视机等移动设备。最初由由Andy Rubin开发，后在 2005年8月被Google收购，随后Google以Apache开源许可证和GPL的授权方式，发布了Android的源代码。

从Android刚刚发布时的1.0版本到目前的5.0版本，Android逐渐从一个崭新的系统发展为最成熟的移动端操作系统。

Android作为市场份额最大的手机端操作系统，得益于其的平台开放性和开发开放性。Google允许任何手机终端厂商加入Android平台联盟，也正因为众多的手机厂商加盟，使得Android平台拥有了众多的开发者，开发的Android应用更加丰富，用户数越来越多。

Google提供给Android一个相当自由的开发环境，有着不停更新的Development Toolkit，和较少约束的系统API，使得开发者可以自由地进行开发。

Android 是运行于Linux kernel之上，但并不是GNU/Linux。因为在一般GNU/Linux 里支持的功能，Android 大都没有支持，包括Cairo、X11、Alsa、FFmpeg、GTK、Pango及Glibc等都被移除掉了。Android又以Bionic 取代Glibc、以Skia 取代Cairo、再以opencore取代FFmpeg等等。

Android 为了达到商业应用，必须移除被GNU GPL授权证所约束的部份，例如Android将驱动程序移到 Userspace，使得Linux driver 与 Linux kernel彻底分开。

选择Android平台开发，主要是看重它丰富的硬件资源和开发上的开放性。开放的平台允许任何移动终端厂商加入到Android联盟中来。显著的开放性又可以使其拥有更多的开发者。

#### 认证方式的选择

根据识别系统对待识别语音内容的不同，又可以分为文本相关和文本无关两种方式。

文本相关的说话人识别方式要求说话人发音的关键词或关键句子与训练文本相同，且在识别时也要按照相同内容发音。

文本无关的说话人识别方式在训练和识别时都不对说话人的的发音内容做要求，其识别对象是任意的语音信号。

本课题所研究开发的系统采用文本相关的说话人识别方式。

### 研究目的和意义

通过本课题的面向Android平台手机用户的说话人识别系统，可以对用户进行生物身份识别确认以替代原本的手机上的数字密码验证，避免了数字密码遭非法用户窃取的风险。

除此之外本课题还使得生物身份识别系统更简单易用，在安保、司法等以外领域的地方部署，可以极大地促进说话人识别技术的发展。

最后，目前移动互联网行业发展迅速，但是安全性还是原地踏步，本系统还可以与Android平台上的其他应用进行绑定，以极大地增强一些移动互联网应用的安全性，例如手机支付钱包等功能。

### 论文组织结构

本文组织如下，第一章，背景，主要介绍课题背景，开发平台和认证方法的选取，课题的目的和意义等；第二章，文献综述，主要介绍系统开发所需要基本知识；第三章，系统的需求分析；第四章，系统的详细设计；第五章，系统的主要功能及实现；第六章，实验中所遇到的问题及解决方法；第七章，对未来的展望和当前系统的总结。

## 文献综述



### 识别系统结构

说话人识别系统主要分为三个模块，分别是特征提取模块，模型训练模块，模型识别模块。

#### 特征提取模块

语音特征提取是指采用数字技术与模拟技术相结合，选择和提取语音信号中的特征，得到说话人的模型，而不是直接从语音信号中得到。无论是训练时的语音还是测试时的语音，都要通过语音特征提取才能进行后续建模处理。

对于说话人识别，语音信号中提取到的特征需要满足能较好地区别不同用户的能力，又能对相同用户具有同一性。

人们能对不同人的语音做出区别是因为人们的耳蜗类似于一个滤波器组，可以对语音信号进行过滤后找出不同说话人的特征。因此想要在计算机上对语音进行特征提取就要模拟出一个类似人耳的滤波器组。

当频率在1000HZ以下时，人们的耳朵的感知能力与频率成线性关系；但当频率在1000HZ以上时，人们的耳朵的感知能力与频率不再构成线性关系，而更偏向于对数关系，这就使得人们的耳朵相比于对比高频信号来说对低频信号更敏感，故采用Mel 频率倒谱系数(即 MFCC)。

特征提取模块首先对要处理的语音信号进行预加重、分帧、加窗等处理。预处理后得到大量的数据量。提取特征这一过程就是为了去除原来语音中的冗余信息，减少数据量。

Mel 频率倒谱系数提取过程如图2-1所示：

1. 对语音信号s(n)进行预加重，分帧，加窗等处理，得到每个帧的时域信号x(n)。
2. 将上述对时域信号x(n)后补0，直至形成长度为N的序列(N一般为256)，然后进行快速傅立叶变换(FFT)后得到线性频谱x(k)：

|  |  |
| --- | --- |
|  | ——公式2-1 |

1. 将上一步的线性频谱x(k)通过Mel频谱滤波器，并取每个三角形的滤波器频率带宽内的所有信号幅度加权和作为某个带通滤波器的输出，从而得到Mel频谱，并通过对数能量的处理，得到对数频谱s(m)：

|  |  |
| --- | --- |
|  | ——公式2-2 |

为各个带通滤波器的传递函数。

1. 将上一步的对数频谱s(m)经过离散余弦变换(DCT)得到倒谱频率，即可得到Mel频率倒谱系数c(n)：

|  |  |
| --- | --- |
|  | ——公式2-3 |

图2-1 Mel 频率倒谱系数提取过程

#### 模型训练和识别模块

在特征被提取出来后，需要用模型为说话人建模，并根据特征以标记其是属于哪一个说话人。一般常用的建模模型有参数模型、非参数模型、人工神经网模型和支持向量机模型。

典型的参数模型包括高斯混合模型(GMM)，隐马尔科夫模型(HMM)，也是在语音识别领域，最常用到的两个模型。高斯混合模型常用于文本无关的语音识别，而隐马尔科夫模型常用于文本相关的语音识别。

对于一个时间长度为T的语音序列X=()，它的GMM似然函数可以写作：

|  |  |
| --- | --- |
|  | (公式2-1) |
|  | (公式2-2) |

识别时运用贝叶斯定理，在N个未知话者的模型中，得到的最大似然概率的模型对应的说话人即为识别结果。

|  |  |
| --- | --- |
|  | (公式2-3) |

### 隐马尔科夫模型(HMM)

隐马尔科夫模型（Hidden Markov Model，HMM）是一个统计模型，用来描述一个含有隐含未知参数的状态跳转过程。其要点在于从可观察的参数中找出这一系列状态跳转过程中的隐含参数。然后利用这些参数来做进一步的分析，例如语音识别。

在正常的马尔科夫模型中，状态参数对于观察者来说是完全直接可见的。而在隐马尔科夫模型中，状态参数并不是直接可见的，但模型中受状态影响的某些参数变量则是可见的。

例如在某地的某人在雨天喜欢宅在家中，在晴天时喜欢出去踢球，那么当你得知这个人在某一天出去踢球(或宅在家中)，就可以推断出这个人所在的地方的天气是晴天(或雨天)。这个人告诉了你的他的活动，也就是一个受天气影响的参数变量，而我们对天气并不是可知的，这样一个系统就是隐马尔科夫型模型。

通过给定测试序列(测试用户的语音特征序列) 和模型参数(事先训练好的语音特征序列集合)，再利用Viterbi算法寻找某种意义上最优的状态序列(测试用户的身份)。这就是在文本相关语音识别领域使用隐马尔科夫模型的过程。

HMM隐马尔科夫模型方法现已成为语音识别的主流技术，目前大多数大词汇量、连续语音的非特定人语音识别系统都是基于HMM隐马尔科夫模型的。HMM是对语音信号的时间序列结构建立统计模型，将之看作一个数学上的双重随机过程：一个是用具有有限状态数的马尔可夫链来模拟语音信号统计特性变化的隐含的随机过程，另一个是与马尔可夫链的每一个状态相关联的观测序列的随机过程。前者通过后者表现出来，但前者的具体参数是不可测的。

人的言语过程实际上就是一个双重随机过程，语音信号本身是一个可观测的时变序列，是由大脑里的(不可观察的)、根据言语需要和语法知识(状态选择)所发出的音素(词、句)的参数流。可见HMM合理地模仿了这一过程，很好地描述了语音信号的整体非平稳性和局部平稳性，是较为理想的一种统计语音模型。

通常一个HMM隐马尔科夫模型由以下参数来描述：

1）N，隐马尔科夫模型的状态数，在计算时各个状态为{1,2,3…,N}，在t时刻所处的状态为。

2）M，每个状态中可以观测到的符号数，标记各个观察符号为V={,,…,}，观察序列为O={,,…,}，其中为集合V中的一种观察符号，T为观察序列的长度。

3)A=[]，状态转移概率矩阵，其中 。

4)B=[ (k)]，某个状态下观察符号的概率分布，其中 。

5)π=[]，初始状态概率分布，其中 。

综上所述，一个HMM隐马尔科夫模型完全可以由两个模型N，M和3个概率分布函数A，B，π来确定。因此常常将HMM隐马尔科夫模型定义为λ=(A,B,π)。

总的看来，HMM隐马尔科夫模型可以分为两个部分，一个是Markov链，由π，A描述，其产生的输出为状态序列；另一个是随机过程B，产生的输出为观测序列。

要在语音识别领域使用HMM隐马尔科夫模型就是要解决在已知观察序列O和模型λ=(A,B,π)时，如何确定一个合理的状态序列，使之能最佳地产生O，即如何选择最佳的状态序列q={,,…,}这一问题。

要解决这一问题，需要用到Viterbi维特比算法。这个算法解决了给定一个观察序列O={,,…,}和模型λ=(A,B,π)，在最佳意义上确定一个状态序列q={,,…,}。

### HTK工具箱

#### HTK工具箱概述

HTK工具箱是一个用于构建和操作隐马尔科夫模型的便携式工具箱。主要用于语音识别的研究。

HTK工具箱的多数功能已被建立成库模块。这些模块确保了每个工具对外部的接口使用完全相同的方法，并提供了通用功能的中心资源。图2-2说明了HTK工具箱的架构和输入输出接口。

* HShell控制用户的输入输出以及与操作系统的交互。
* HMem控制所有的内存管理。
* HMath提供数学支持。
* HSigP完成语音分析所需的信号量处理操作。
* HLabel提供标签文件的接口。
* HLM用于语言模型文件。
* HNet用于网络和网格。
* HDict用于字典。
* HVQ用于矢量量化码本。
* HModel用于隐马尔科夫模型（HMM）的定义。
* HWave完成所有在波形阶段的语音的输入输出。
* HParm完成语音的参数化阶段。
* HWave和HLabel支持多文件格式并允许数据从其他系统输入。
* HAudio支持音频的直接输入。
* HGraf提供简单的交互式图形。
* HUtil提供大量用于计算HMM的通用程序。
* HTrain和HFB支持多种HTK训练工具。
* HAdapt提供多种适应工具的支持。
* HRec包含了识别过程中使用的主要函数。



图2-2 HTK工具箱的架构和输入输出接口

#### HTK工具箱使用流程

如图2-3所示，HTK工具箱的使用有四个主要的阶段：数据准备，训练，测试，分析。

* 数据准备

对于语音信号，可以使用HCopy参数化波形，得到MFCC特征文件。

* 训练

由HInit读入所有的引导训练数据，并迭代计算一组初始参数值。通过HInit计算的初始参数值由HRest进一步重估，直至收敛。

* 测试

HVite使用令牌传递算法来执行基于维特比的语音识别。HVite把输入看作是一个网络，该网络描述了所容许的单词序列，并定义了每个单词如何发声的字典和一组HMM。它将单词网络转换成音素网络，之后在每个音素实例附上合适的HMM定义，以实现运转。接着就可在一系列存储的语音文件上或者直接的音频输入上进行识别。

* 分析

基于HMM的识别系统的性能评估由HResults完成。



图2-3 HTK工具箱使用的四个阶段

### Android开发

如图2.3所示，Android开发有4个层次，自下而上分别是Linux内核层开发，类库层开发，应用框架层开发，应用层开发。

第一层Linux内核层开发由C实现，主要负责管理Android硬件的驱动。

第二层类库层开发依然由C实现，主要负责Android上一些常用类库和组件的开发，例如数据库SQLite，浏览器内核WebKit，常用的C函数库libc，运行应用的虚拟机Dalvik。

第三层是Android系统应用框架，由Java实现，包含了Activity Manager界面管理，Notification Manager通知管理，Resource Manager资源管理等等，实现了对Linux内核层和类库层的封装，让Android不过度依赖与Linux内核，达到内核无关的特性，让Android框架开发和Android应用开发能在不考虑Linux内核和驱动下顺利完成，以吸引更多的开发者。

第四层是Android应用程序的开发，由Java实现，可以利用Android提供的API接口实现各种功能的应用，例如信息服务，拍照，录音，定位等等，以及各种界面控件完成用户交互，例如文本框EditText，按钮Button等等。

图2.3 Android开发层次图

Android应用程序由4大组件构成，分别是Activity、Service、Broadcast Receiver、Content Provider。

Activity是Android应用与用户交互的界面，可以显示一些控件也可以监听并处理用户的事件做出响应，是Android 应用程序的基本功能单元。Activity 本身是没有界面的。所以Activity类创建了一个接口，开发人员可以通过setContentView(View v)接口把用户界面的xml布局文件放到Activity创建的窗口上。Android 中通过 Intent 对象来表示一条消息，一个 Intent 对象不仅包含有这个消息的目的地，还可以包含内容，用以传递信息，或执行跳转操作。

Service是一个没有用户界面的在后台运行执行耗时操作的应用组件。其他应用组件能够启动Service，并且当用户切换到另外的应用场景，Service将持续在后台运行。另外，一个组件能够绑定到另一个Service与之交互。例如，一个Service可能会处理网络操作，播放音乐，操作文件I/O或者与内容提供者（Content Provider）交互，所有这些活动都是在后台进行。

Broadcast Receiver用来对外部事件进行过滤，从而只获得自己感兴趣的内容。广播接收器是一个专注于接收广播通知信息，并做出对应处理的组件。很多广播是源自于系统代码的──比如，通知时区改变、电池电量低、拍摄了一张照片或者用户改变了语言选项。应用程序也可以进行广播──比如说，通知其它应用程序一些数据下载完成并处于可用状态。

应用程序可以拥有任意数量的广播接收器以对所有它感兴趣的通知信息予以响应。所有的接收器均继承自BroadcastReceiver基类。

广播接收器没有用户界面。然而，它们可以启动一个Activity来响应它们收到的信息，或者用NotificationManager来通知用户。通知可以用很多种方式来吸引用户的注意力──闪动背灯、震动、播放声音等等。一般来说是在状态栏上放一个持久的图标，用户可以打开它并获取消息。

Content Provider用于给应用提供各种各样的数据，例如SQLite数据库，SharedPreferences数据存储文件甚至比如音频、视频、图片和私人通讯录等。

Content Provider还可以支持在多个应用中存储和读取数据。这也是跨应用共享数据的唯一方式。在Android系统中，没有一个供多个应用共享存储数据的公共的内存区域。

本课题的开发的说话人识别系统主要在第二层和第四层完成。先在第二层中移植好HTK语音工具箱类库，再在第四层中开发一个应用调用这个类库，完成语音特征提取，训练建模，测试识别等任务，同时丰富好用户界面和业务逻辑。

### SQLite数据库以及ORM框架

SQLite是Android应用中最常用的数据库，不同于MySql、MS SQL Server等客户端服务器数据库，SQLite是一个单文件数据库。使用C构建，并集成在Android系统的第二层——类库层中，每个Android应用都可以获得一个专属的SQLite数据库，并不需要依赖第三方软件，也不需要所谓的安装。Android应用在开发时可以使用SQLiteOpenHelper类中的方法，将应用的Context作为参数传入，得到这个应用的SQLiteDatabase，之后就可以通过execSQL方法将SQL语句传入并执行。

然而使用SQLiteOpenHelper提供的方法操作数据库并不符合面向对象的软件开发思想。因此本课题在操作数据库的过程中使用了ORM对象关系映射(Object Relational Mapping)框架。

ORM框架即对象关系映射，Object Relational Mapping，是一种编程技术，用于实现在面向对象语言中通过面向对象的思维操纵数据库。对象和关系数据是业务实体的两种表现形式，业务实体在内存中表现为对象，在数据库中则表现为关系数据。内存中的对象之间存在关联和继承关系，而在数据库中，关系数据一般通过外键表示关联关系。因此，ORM框架一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射。

目前ORM框架可谓汗牛充栋，但为了能扎实掌握这一门技术以及更好地适应于本课题的在该方面的需求，在借鉴了一些ORM框架后，改编了其中之一xUtils，使其成为更小更适合本课题的xutil\_orm框架。通过使用ORM框架，不仅可以省去撰写SQL语句的困扰，还可以更好地对一些异常和事务进行处理。

DbUtils db = DbUtils.create(Context context)是xUtils中数据库模块的入口，在使用时往其中传入Android上下文参数context，xUtils就会通过单例模式返回一个通过SQLiteDatabase.openOrCreateDatabase(Context context)创建的SQLiteDatabase对象，增加了面向对象的增删改差等功能，通过db.save(Entity e)、db.update(Entity e)、db.find(int id)、db.delete(Entity e)等实现，还提供了事务和异常处理等机制，并且依然可以使用传统SQL增删改差数据。

### Apache-commons-io包

本课题中涉及到不少文件的操作，例如WAV语音文件的新建、复制与删除，MFCC特征文件的新建、复制与删除，HMM模型文件的新建、复制与删除，以及文件路径的获取和传递。为了使得本系统有更好的健壮性，以及遵守软件工程中“不重复造轮子”的思想，本系统中使用了Apache-commons-io包。

Apache-commons-io包含有Utility、Input、Output、Filters、Comparators、Monitor等工具包。

Utility工具包主要用来处理文件、文件夹的操作。Input、Output工具包实现了对Java环境中原有的输入输出流对象的封装。Filters工具包实现了对目录文件过滤的功能。Comparators工具包实现了文件比较的一些方法。Monitor工具包实现了对文本和目录变化的监视。

通过其中的FileUtils类中的不同的方法可以清晰简单地对文件、文件夹进行新建、修改、删除、复制、获取路径等操作。使用Apache-commons-io类库可以提升开发效率，并且更好地处理一些文件异常，增强应用的健壮性。

例如执行FileUtils.copyInputStreamToFile(InputStream inputStream, File file)，传入输入流和文件参数即可实现从流到文件的复制；执行FileUtils.forceMkdir(File file)即可实现建立文件夹；执行FileUtils.write(File file, String string)即可实现写入字符串到文件中；执行FileUtils.deleteQuietly(File file)即可实现删除功能；执行FileUtils.openInputStream(File file)即可实现从文件读取流的功能；执行FileUtils.readLines(File file)即可实现读取文件到字符串列表中。

### JNI

JNI即Java Native Interface，是Java环境中提供的可以和其他语音(C&C++)通信的API。是Java平台的一部分，它允许Java代码和其他语言写的代码进行交互。

编写JNI接口时首先要编写带有native声明的方法的Java类。其次使用javac命令编译所编写的java类。然后使用javah生成扩展名为h的头文件。接着引入头文件，并根据头文件编写C/C++程序以实现本地方法。最后将C/C++编写的文件生成动态连接库。并在相关Java类中通过System.loadLibrary方法加载。

在使用JNI时，只需在方法前声明native关键字，JVM即会自动调用(invoke)这个方法对应的C/C++的本地实现方法。

Android应用层的类都是以Java编写，这些Java类编译为Dex型式的Bytecode之后，必须靠Dalvik虚拟机运行。执行Java类的过程中，如果Java类需要与C组件沟通时，Dalvik虚拟机就会去载入C/C++组件，然后让Java的函数顺利地调用到C组件的函数。此时，Dalvik扮演着桥梁的角色，让Java与C/C++组件能通过标准的JNI介面而相互沟通。

由于Java和C/C++是两种不同的语音，因此在交互时涉及到数据格式的转换，分别是基本数据类型的转换和引用数据类型的转换，分别如表2.1和表2.2所示。

表2.1 JNI基础数据类型转换图

|  |  |  |
| --- | --- | --- |
| Java基础数据类型 | C/C++数据类型 | 空间 |
| boolean | jboolean | 无标志8位 |
| byte | jbyte | 有标志8位 |
| char | jchar | 无标志16位 |
| short | jshort | 有标志16位 |
| int | jint | 有标志32位 |
| long | jlong | 有标志64位 |
| float | jfloat | 32位 |
| double | jdouble | 64位 |

表2.2 JNI引用数据类型转换图

|  |  |
| --- | --- |
| Java引用数据类型 | C/C++数据类型 |
| Object | Jobject |
| String | jstring |
| Object[] | jobjectarray |
| byte[] | jbytearray |

## 系统需求分析



### 系统概述

经过简单的需求分析，如图3-1所示，本课题开发的说话人识别系统主要有一下几个模块：



图3-1功能模块

### 模块功能描述

#### 登录模块

系统的登录界面，要求用户输入密码，防止非法用户登录本系统进行操作，若密码正确，用户通过此界面进入主界面，否则退出。

#### 主界面模块

系统的主界面，是其他功能模块的入口所在，用户可以在此界面跳转到测试模块，训练模块，管理模块等等。

#### 用户管理模块

系统的用户管理界面，是用户训练界面的上一级入口。在该界面上可以添加删除说话人或是查看用户详细信息。

#### 训练模块

系统的训练界面，由用户管理界面进入。通过AudioRecord类实现对用户的语音的记录，并保存为WAV格式文件。再利用HTKTools中的HCopy进行梅尔频率倒谱系数MFCC特征提取处理得到MFCC模型。得到特征模型后，先用HTKTools中的HInit对模型初始化确定状态数和特征维数，在用HRest进行两次迭代，得到建模训练后的HMM模型。

#### 阈值设置模块

阈值设置模块，允许用户修改阈值，并将阈值保存到系统数据库中，供每次测试时读取。

#### 测试模块

系统的测试界面，通过AudioRecord类实现对用户的语音的记录，再利用HTKTools中的HCopy进行梅尔频率倒谱系数MFCC特征提取处理得到MFCC模型。最后利用HTKTools中的HVite对MFCC模型和原本的HMM模型进行维特比算法解码识别，得到测试结果。

#### 锁屏界面模块

通过Android系统的API对Android手机进行锁定，通过AudioRecord类实现对用户的语音的记录，再利用HTKTools中的HCopy进行梅尔频率倒谱系数MFCC特征提取处理得到MFCC模型，以进行特征提取和解码，当通过系统确认时锁定解除。

## 详细设计



### 总体设计

Android上应用开发普遍遵循MVC设计模式，MVC的全称是Model模型，View视图和Controller控制器。

MVC是Xerox PARC在八十年代为编程语言Smalltalk－80发明的一种软件设计模式，至今已被广泛使用。被推荐为Oracle旗下Sun公司Java EE平台的设计模式。MVC是用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

它强制性的使应用程序的输入、处理和输出分开。使用MVC应用程序被分成三个核心部件：模型、视图、控制器。它们各自处理自己的任务。

视图是用户看到并与之交互的界面。模型表示企业数据和业务规则。控制器接受用户的输入并调用模型和视图去完成用户的需求。

使用MVC设计模式最典型的就是JSP + servlet + javabean框架。JSP负责View视图层，servlet负责Controller控制器层，javabean负责Model模型层。然而不同的情况下MVC框架也是有不同的。

MVC设计模式在Android应用上的体现就是：Model模型对应用户数据和文件，View视图对应XML界面布局和Activity中对界面的渲染和数据绑定，Controller控制器对应Activity中的Intent跳转和业务逻辑。

通过MVC设计模式，可以将数据，业务逻辑和界面解耦，将这三部分分别都集中部署，在定制数据结构、改进业务逻辑或修改界面时可以不需要重新编写全部代码，只需修改相应部分的代码。使用MVC设计模式可以提高代码质量，提升开发效率。

### 体系结构设计

#### 模型层Model

模型层主要包含有用户、KeyValue参数、文件等结构。分别用于表示用户信息（包含编号、姓名、训练状态、训练时间、测试时间等信息），系统参数（包含系统密码、状态参数等信息），和训练后生成的相关文件。

用户和KeyValue参数储存于数据库中，并通过AccessObject封装后访问，可以进行添加删除修改用户，修改密码，修改阈值，修改训练状态，修改系统参数等操作。

#### 视图层View

视图层包含有初始注册界面、登陆界面、主界面、用户管理界面、训练界面、测试界面、阈值设置界面、修改密码界面。

初始注册界面用以在安装系统过后展示初始化登陆密码界面。

登陆界面用以展示输入登陆密码防止非法用户登陆的界面。

主界面上展示有其他界面的入口，例如用户训练管理、锁屏服务状态、测试界面，是整个系统的中心界面。

用户管理界面展示有所有说话人账户，包含有添加删除说话人账户功能按钮，同时还是训练界面的入口。

训练界面展示有语音文本提示的选取框，同时还包含有录音功能按钮，指导用户录音的文本，以进行特征提取和模型训练。

阈值界面用以查看和设置阈值。供测试时读取使用。

测试界面包含训练过的所有用户，同时包含有录音功能按钮，帮助用户录音，以确认该录音是否来自用户声明的说话人。

修改密码界面展示有用户修改密码的文本框，供用户修改密码并保存。

#### 控制器层Controller

控制器层包含有登陆模块、主界面模块、用户管理模块、阈值设置、训练模块、测试模块。

登陆模块用以控制初始界面和登陆界面的用户交互。在安装后第一次登陆时获取界面上用户输入的密码并加密存储到数据库中。之后使用时再将获取界面上用户输入的密码加密后与数据库中的数据校验。若正确则跳转到主界面，不正确则继续输入直至密码正确。

主界面模块上含有不同界面的入口。当用户选取主界面上不同的控件时，则根据用户需要执行相应的操作。

用户管理模块用以控制用户管理界面的用户交互。根据用户的响应执行添加删除说话人，或进入该说话人训练模块的操作，同时通过AccessObject将添加或删除的操作保存到数据库。

阈值设置模块用以查看和修改阈值，并将结果保存至数据库中，供测试时使用，以判断用户是否合法，防止非法用户冒充。

训练模块用以控制训练界面的用户交互。在用户确认训练后开始录音，并对录音进行特征提取，再将特征进行建模并储存。

测试模块用以控制测试界面的用户交互。在用户声明了一个说话人后开始录音，并对录音进行特征提取，再将特征进行建模，最后再将建模的结果与之前训练的说话人对比，确认是否是合法的说话人。

### 数据结构设计

本系统数据结构主要包含三个部分，分别是用户信息、系统参数信息、文件。其中用户信息和系统参数信息存放于数据库中，文件存放于手机存储卡中。

USER用户信息表设计表如表4.1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 允许null | 备注 |
| Id | Int | 不允许 | 编号，作为主键可自增长 |
| NAME | String | 不允许 | 姓名 |
| IS\_TRAINED | Boolean | 允许 | 训练状态 |
| TRAIN\_TIME | Date | 允许 | 训练时间 |
| LAST\_VERIFY\_TIME | Date | 允许 | 最后一次识别时间 |
| QUESTION | String | 允许 | 问题提示 |
| THRESHOLD | String | 允许 | 阈值 |

表4.1 USER用户信息表设计表

KEYVALUE系统参数表设计表如表4.2所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 允许null | 备注 |
| KEY | String | 不允许 | 参数名，作为主键 |
| VALUE | String | 允许 | 参数值内容 |

表4.2 KEYVALUE系统参数表设计表

文件结构树形结构

│ all.mmf // HVite解码时所需的所有hmm模型

│ config //存放HTK的配置信息

│ dict.txt //依据当前训练的用户生成的词典文件

│ gram.txt //语法格式文件

│ hmmlist.txt // HVite解码时所需的所有hmm模型的名称的列表

│ net.slf //HParse解析后生成的网格文件

│ reco.mlf //HVite解码识别结果

│ trainlist.txt //需要训练hmm模型的特征文件的名称的列表

│ wavlist.txt //需要生成特征文件的wav音频文件的名称的列表

│

├─hmm0 //存放执行HInit后的hmm模型结果的文件夹

│ hmm\_id4 //执行HInit后的hmm模型结果

│ hmm\_id5 //执行HInit后的hmm模型结果

│

├─hmm1 //存放第一次执行HRest后的hmm模型结果的文件夹

│ hmm\_id4 //第一次执行HRest后的hmm模型结果

│ hmm\_id5 //第一次执行HRest后的hmm模型结果

│

├─hmm2 //存放第二次执行HRest后的hmm模型结果的文件夹

│ hmm\_id4 //第二次执行HRest后的hmm模型结果

│ hmm\_id5 //第二次执行HRest后的hmm模型结果

│

├─lab //存放hmm模型的标签文件的文件夹

│ ├─id4//存放具体用户的hmm模型的标签文件的文件夹

│ │ id4\_1.lab //第一段音频的lab标签文件

│ │ id4\_2.lab //第二段音频的lab标签文件

│ │ id4\_3.lab //第三段音频的lab标签文件

│ │

│ └─id7

│ │ id7\_1.lab //第一段音频的lab标签文件

│ │ id7\_2.lab //第二段音频的lab标签文件

│

├─mfcc //存放提取了的MFCC特征文件

│ id4-540968289.mfc //用户id4的音频的MFCC特征提取文件

│ id4-541404317.mfc //用户id4的音频的MFCC特征提取文件

│

├─proto //存放最初始的hmm模型文件，以供训练使用

│ hmm\_id4 //初始化的hmm模型文件，供HInit训练使用

│ hmm\_id5 //初始化的hmm模型文件，供HInit训练使用

│

├─testwav //存放测试时录制的声音文件

│ id4-540968289.wav //用户id4的测试录音音频文件

│ id4-541404317.wav //用户id4的测试录音音频文件

│

└─trainwav //存放训练时时录制的声音文件

id4\_1.wav //用户id4的第一次训练录音音频文件

id4\_2.wav //用户id4的第二次训练录音音频文件

id4\_3.wav //用户id4的第三次训练录音音频文件

### HTK封装设计

经过需求分析后，本系统将使用到HTK工具箱中的HCopy特征提取模块，HInit初始化建模模块，HRest模型迭代模块，HParse语法生成模块，HVite解码识别模块。

HCopy中将特征格式设置为MFCC\_0\_D\_A，输入音频格式设置为WAV，设置好帧长度，滤波器类型等等。系统将设置参数存放于文件中以供使用。

特征提取时使用-C [config] -S [wavlist.txt]命令，config表示设置文件路径，wavlist.txt文件包含有需要进行特征的音频文件路径列表。

HInit初始化时使用39度10状态数的HMM模型和3次训练音频提取的MFCC特征进行初始化。系统首先将初始化一个39度10状态数的HMM模型原型proto，再配合提取的MFCC特征和音频所对应的时间标签文件即可进行训练。

训练时使用-S [aaatrainlist] -M [hmm0] -H [proto/hmm\_aaa] -l [aaa] -L [lab/aaa] [aaa]命令，aaatrainlist文件包含有需要进行训练的所有MFCC特征路径，hmm0表示训练后结果输出的文件夹，proto/hmm\_aaa表示HMM模型原型proto，aaa表示这段训练语音的名称，lab/aaa表示语音标签所在的文件夹。

HRest只需迭代计算HInit的结果，无需设置。

HParse依据当前用户列表生成的词典和语法文件进行解析，生成语法网格文件，供HVite使用。

生成时使用[gram.txt] [net.slf]命令，gram.txt为语法文件，内容为“$WORD=aaa|bbb;([ $WORD])”，net.slf为生成的网格文件。

HVite依据测试时的音频的MFCC特征文件和所有已训练的HMM模型，从所有用户的HMM模型中识别出最接近特征的模型，给出模型名和似然概率。

识别时使用-H [all.mmf] -i [reco.mlf] -w [net.slf] [dict.txt] [hmmlist.txt] [mfcc/lock5.mfc]命令，all.mmf表示所有训练后的HMM模型文件，reco.mlf表示输出的结果文件，net.slf表示HParse生成的网格文件，dict.txt表示为词典文件内容为“aaa [aaa] aaa \n bbb [bbb] bbb”，hmmlist.txt文件包含有将用到的HMM模型名，mfcc/lock5.mfc则是需要识别的MFCC特征文件。

### 开发平台及开发工具

开发平台：Android（系统运行平台）。

开发工具：Eclipse with Android Development Tool（集成开发环境）、Android SDK（Android应用编译工具）、Android NDK（交叉编译工具）、MS Visual Studio（x86环境下编译工具）。

代码管理工具：Git

代码托管网站：GitHub

Eclipse 是一个开放源代码的、基于Java的集成开发环境（IDE）。在刚面世时，主要是用来进行Java相关的开发，但是之后可以通过安装不同的插件，支持不同的计算机开发语言，比如C++和Python等。这使得Eclipse成为了一个框架平台，众多的插件使得Eclipse不同于MS Visual Studio等集成开发环境，具有较强的灵活性。

本系统的开发使用到了Eclipse集成开发环境，以及其中的Android Development Tools插件。Android Development Tools为Eclipse扩展了功能，使得开发者能够较为轻松地建立Android项目，以及绘制应用程序界面。除此之外，还能调用Android SDK和Android NDK完成编译和调试。

Android SDK是Android软件开发工具包。包含开发者用于开发应用的各种开发工具。例如emulator模拟器、adb系统调试工具、ddms Dalvik虚拟机调试工具、ant编译工具等等。

Android NDK是Android Native Development Kit即原生开发工具包，是编译C/C++本地JNI源码的工具，为开发人员将本地C/C++方法整合到Android应用中提供了方便。方便了Android平台上的JNI编程。原生开发工具包中包含了基于gcc的不同手机平台的toolchains交叉编译工具集。利用Android NDK可以将用C编写的HTK工具箱编译运行在Android应用中使用。

MS Visual Studio是由微软公司开发的目前最流行的Windows平台应用程序的集成开发环境，包含有MSBuild编辑工具、UML工具、代码管控工具、调试工具、团队协作工具等等，是一个基本完整的开发工具集。利用MS Visual Studio可以在Windows平台上编译HTK工具箱，做到对HTK工具箱能有基本的掌握，熟悉相应的输入输出结果。

Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。Git 最初是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。不同于Subversion，Git并没有中心库的概念，Git是一个分布式的版本管理控制系统，每一个克隆出的库都可以脱离中心库当作离线仓库使用。

使用Git可以在自己的机器上根据不同的开发目的，创建分支branch，修改代码提交commit到本地分支，回滚reset到旧的commit版本，合并merge到不同分支，推出push到别的仓库，从别的仓库同步拉取pull等操作。

在开发时，使用Git对源代码的版本进行管理。在添加新功能时创建新的分支branch，编号好代码后使用merge合并到原本的master分支上。同时还可以使用reset回滚版本或revert撤销操作。当主要功能添加完成后，还可以建立tag标记，紧密地发布迭代版本。本系统预计将进行3-5次迭代，分别在实现界面，完成功能，改进功能，修复bug等等之后。

GitHub是一个提供Git库的代码托管网站，通过Git可以将代码同步到GitHub空间中，方便在任何联网的地点查看和修改代码。同时还可以通过fork同步别人开源的项目，或是pull request提出合并自己修改的请求，或者是仅仅提出issue问题。

由于在系统设计和实现期间，仍有实习和面试等一系列活动，因此需要在不同的开发机上开发，因此使用了基于Git的代码托管网站GitHub。通过GitHub可以在任意联网的地方，通过Git进行pull或push等操作来下载或更新代码库。

## 功能和实现



### 数据模型

系统中存在于内存的数据模型有用户信息User的POJO类，参数值KeyValue的POJO类，User的AccessObject类，各类文件的AccessObject类。

* 用户信息的POJO类，允许获取或设置用户信息，同时相关字段带有注解，允许ORM框架通过反射获取方法和属性，进行数据库操作。

实现代码：

User.java

|  |
| --- |
| @Table(name = "USER")//绑定数据库表的注解  public class User {  @Id//绑定数据库主键字段的注解  private int id;//自增长编号  private String name; //用户姓名  private Boolean isTrained; //用户训练状态  private Date trainTime; //用户训练时间  private Date lastVerifyTime; //用户最后验证时间  private String tips//用户语音文本提示  public int getId() {  return id;  }//相关属性的get、set方法封装  public void setId(int id) {  this.id = id;  }  ……  } |

* KeyValue的POJO类，允许获取或设置系统参数，例如系统密码、是否第一次使用等信息。同时类中的相关字段带有注解，允许ORM框架通过反射获取方法和属性，进行数据库操作。

实现代码：

KeyValue.java

|  |
| --- |
| @Table(name = "KEYVALUE")//绑定数据库表的注解  public class KeyValue {  @Id//设置数据库主键的注解  @Column(column = "KEY")//设置数据库字段的注解  private String key; //KeyValue键名  private String value; //KeyValue值内容  public String getKey() {  return key;  }//相关属性的get、set方法封装  public void setKey(String key) {  this.key = key;  }  ……  } |

* User的AccessObject类，使用singleton单例模式以节约内存。用来操作数据库中的用户表，包含有获得用户表、获得已训练用户表、添加用户、删除用户、训练用户、校验用户等操作。

实现代码：

UserAccessObject.java

|  |
| --- |
| public class UserAccessObject {  Context context; //Android上下文参数  DbUtils db; //数据库对象  static UserAccessObject singleton=null;  public static UserAccessObject getInstance(Context context) {  ……  }//单例模式获取该类的实例  private UserAccessObject (Context context) {  ……  }//带有Android上下文参数的非空构造函数  public List<User> getUserList() {  ……  }//获取用户列表  public List<User> getTrainedUserList() {  ……  }获取已训练的用户列表  public boolean addUser(User user) {  ……  }//添加用户  public boolean deleteUser(User user) {  ……  }//删除用户  public void trainUser(int id) {  ……  }//训练用户  public void verifyUser(int id){  ……  }//验证用户  } |

* KeyValue的AccessAccessObject类，使用singleton单例模式以节约内存。用来操作数据库中的KeyValue表。包含有设置密码、检查是否第一次登陆、校验密码、获取密码等操作。

实现代码：

KeyValueAccessObject.java

|  |
| --- |
| public class KeyValueAccessObject {  DbUtils db; //数据库对象  static KeyValueAccessObject singleton=null;  public static KeyValueAccessObject getInstance(Context context) {  ……  }//单例模式获取该类的实例  private KeyValueAccessObject (Context context){  ……  }//带有Android上下文参数的非空构造函数  public boolean validate(String password) {  ……  }//校验密码是否正确  public String getOldPassword() {  ……  }//获取密码  public boolean isFirstLogin() {  ……  }//判断是否第一次登录  public boolean setPassword( String password) {  ……  }//设置密码  } |

* 各类文件的AccessObject类。使用singleton单例模式以节约内存。用来进行各种文件操作，包括获得目录应用根目录、获得存放模型目录、获得存放mfcc特征目录、创建lab文件、创建dict文件、创建netslf文件、创建模型集合文件等操作。

实现代码：

FileAccessObject.java

|  |
| --- |
| public class FileAccessObject {  String appRoot; //系统的SD卡存储根目录  String hmm0Path; //hmm0文件夹绝对路径  String hmm1Path; //hmm1文件夹绝对路径  String hmm2Path; //hmm2文件夹绝对路径  String protoPath; //proto文件夹绝对路径  String mfccPath; //mfcc文件夹绝对路径  String labPath; //lab文件夹绝对路径  String labUserPath; //lab用户文件夹绝对路径  String trainWavPath; //训练音频文件夹绝对路径  String testWavPath; //测试音频文件夹绝对路径  String resultFile;//结果文件绝对路径  Context context;//Android上下文参数  static FileService singleton=null;  public static FileAccessObject getInstance(Context context) {  ……  }//单例模式获取该类的实例  private FileAccessObject (Context context) {  ……  }//带有Android上下文参数的非空构造函数，初始化各个文件夹并获取路径  public String getAppRoot() {  ……  }//相关路径的get方法封装  public String getHmm0Path() {  ……  }  ……  } |

### HTK封装实现

要在Android平台上使用HTK工具箱，需要对HTK工具箱进行编译和封装，本系统使用到了HTK工具箱中的HCopy、HInit、HParse、HRest等。

1)Android NDK的编译描述文件，此处以HCopy为例，不详细赘述。

实现代码：

Android.mk

|  |
| --- |
| LOCAL\_PATH := $(call my-dir)  include $(CLEAR\_VARS)  LOCAL\_MODULE := libHTKCore  #编译后的静态库名称  LOCAL\_SRC\_FILES := esignal.c esig\_asc.c esig\_edr.c esig\_nat.c HAdapt.c HArc.c HAudio.c HDict.c HExactMPE.c HFB.c HFBLat.c HGraf.null.c HLabel.c HLat.c HLM.c HMap.c HMath.c HMem.c HModel.c HNet.c HParm.c HRec.c HShell.c HSigP.c HTrain.c HUtil.c HVQ.c HWave.c strarr.c Tool\_HCopy.c Tool\_HInit.c Tool\_HParse.c Tool\_HRest.c Tool\_HVite.c  #引用的C代码文件  include $(BUILD\_STATIC\_LIBRARY)  include $(CLEAR\_VARS)  LOCAL\_MODULE := HCopy  #编译后的动态库名称  LOCAL\_SRC\_FILES := com\_flo\_util\_HCopyFunc.c  #编译需要的JNI封装入口方法所在的文件  LOCAL\_WHOLE\_STATIC\_LIBRARIES := libHTKCore  #编译需要的静态库  include $(BUILD\_SHARED\_LIBRARY)  …… |

2)Java中利用Java Native Interface接口调用编译好的C动态共享库，此处以调用HTK中的HCopy为例

实现代码：

HCopyFunc.java

|  |
| --- |
| public class HCopyFunc {  static {  System.loadLibrary("HCopy");  }//静态块中加载HCopy动态库，实现程序在编译时即执行加载动作  public static void exec(String configFilePath, String wavlistPath) {  HCopy(configFilePath, wavlistPath);  } //调用本类中的native方法  private native static void HCopy(String configFilePath, String wavlistPath);  //声明native方法  } |

3)HTK中HCopy的JNIEXPORT封装。

实现代码：

com\_flo\_util\_HCopyFunc.c

|  |
| --- |
| JNIEXPORT void JNICALL Java\_com\_flo\_util\_HCopyFunc\_HCopy  (JNIEnv \* env, jclass j, jstring jconfigFilePath,jstring jwavlistPath) {  char\* configFilePath = (char\*) (\*env)->GetStringUTFChars(env,jconfigFilePath, 0);  //获取configFilePath字符串的数组的引用  char\* wavlistPath = (char\*) (\*env)->GetStringUTFChars(env,jwavlistPath, 0);  //获取wavlistPath字符串的数组的引用  int argc=5;  char \*argv[5];  argv[0]="HCopy";  argv[1]="-C";  argv[2]=configFilePath;  argv[3]="-S";  argv[4]=wavlistPath;  main\_HCopy(argc, argv);  } |

### 用户登陆

为了防止非法用户登陆本系统修改数据，本系统使用密码来管理使用权限。在第一次使用时需要设置一次密码，并储存到数据库中。若不是第一次登陆则要求输入密码并校验是否正确。登录界面是一个dialog，包含有文本框和相关按钮。第一次登录时显示两个密码文本框，供用户输入新密码和确认密码，第二次用户则只显示一个密码文本框。第一次登录时将新密码通过MD5加密后保存到数据库中，用户下次登录时则需要校验密码是否相同。登录界面如图5-1所示。



图5-1用户登录界面 左侧为第一次登录 右侧为非第一次登录

实现代码：

LoginActivity.java

|  |
| --- |
| if (loginAccessObject.isFirstLogin()) {  //如果是初次登录，则需要设置密码  ……  editText\_NewPassword = (EditText) view1  .findViewById(R.id.editText\_NewPassword);  //绑定文本框1视图  editText\_ConfirmPassword = (EditText) view1  .findViewById(R.id.editText\_ConfirmPassword);  //绑定文本框2视图  button\_Register.setOnClickListener(new OnClickListener() {  //按钮绑定单击事件  @Override  public void onClick(View arg0) {  String password1 = editText\_NewPassword.getText()  .toString();//获取文本框1内容  String password2 = editText\_ConfirmPassword.getText()  .toString();//获取文本框2内容  if (password2.equals(password1)) {  if (password1.equals("")) {  //若密码为空则弹出提示  ToastUtil.show(getApplicationContext(), R.string.password\_blank);  } else {  //密码输入无误，保存密码  loginAccessObject.setPassword(password1);  ToastUtil.show(getApplicationContext(),  R.string.register\_success);  Intent intent = new Intent(getApplicationContext(),  MainActivity.class);  startActivity(intent);  //声明主界面到意图中，并执行以跳转到主界面  }  } else {  //若两次密码不同则弹出提示  ToastUtil.show(getApplicationContext(),  R.string.passwords\_diff);  }  }  });  } else {  //如果不是初次登录，则需要输入密码  ……  editText\_Password = (EditText) view2  .findViewById(R.id.editText\_Password);  //绑定文本框视图  button\_Submit.setOnClickListener(new OnClickListener() {  @Override  public void onClick(View arg0) {  password1 = editText\_Password.getText().toString();  //获取密码文本框中的值  if (loginAccessObject.validateUser(password1)) {  //校验密码正确，跳转到主界面  Intent intent = new Intent(getApplicationContext(),  MainActivity.class);  startActivity(intent);  } else {  //校验密码失败，弹出提示  ToastUtil.show(getApplicationContext(),  R.string.password\_error);  }  }  });  } |

### 主界面

主界面上主要用来响应用户的操作，当用户点击不同的控件时在、跳转到响应的模块。主界面上有后台锁屏服务开关按钮，用户管理训练按钮，修改密码按钮，测试界面按钮。后台锁屏服务开关负责启动后台锁屏服务，并监视状态。用户管理训练按钮负责跳转到用户管理训练界面。修改密码负责弹出修改密码对话框。测试界面负责跳转到锁屏界面。主界面如图5-2所示。



图5-2主界面

实现代码：

MainActivity.java

此处以跳转到用户管理界面为例：

|  |
| --- |
| button\_UserManage.setOnClickListener(new OnClickListener() {  @Override  public void onClick(View arg0) {  Intent intent = new Intent(getApplicationContext(),  UserActivity.class);  //声明Intent意图，指定将要跳转的模块  startActivityForResult(intent, 100);  //执行Intent意图进行跳转  }  }); |

### 用户管理

用户管理界面显示有全部用户的信息，并允许添加删除用户，或进入某一用户的训练界面。用户管理界面有一添加按钮，单击后弹出添加用户对话框。用户管理界面中央是一个ListView控件，显示有所有用户，其中的每一项都可以单击，单击后弹出对话框，供用户选择跳转至训练界面或删除用户。用户管理界面如图5-3所示。



图5-3用户管理界面

实现代码：

UserActivity.java：

|  |
| --- |
| //绑定数据到ListView控件上  private void bindView() {  listView = (ListView) findViewById(R.id.listView);  //绑定用户列表视图  userMapList = list2Map(userAccessObject.getUserList());  //获取用户列表数据，并转换成HashMap格式  adapter = new SimpleAdapter(this, userMapList, R.layout.item\_user,  new String[] { "textView\_Username", "textView\_TrainState",  "textView\_TrainTime", "textView\_TestTime" }, new int[] {  R.id.textView\_Username, R.id.textView\_TrainState,  R.id.textView\_TrainTime, R.id.textView\_TestTime });  //初始化适配器，绑定数据和视图文件  listView.setAdapter(adapter);  //绑定适配器到列表上  }  //绑定ListView单击事件：  private void bindListener() {  listView.setOnItemClickListener(new OnItemClickListener() {  @Override  public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,  long arg3) {  //单击列表某一项时弹出选择训练或删除的对话框  button\_Train.setOnClickListener(new OnClickListener() {  @Override  public void onClick(View arg0) {  Intent intent = new Intent(getApplicationContext(),  TrainActivity.class);  intent.putExtra("USERNAME", map  .get("textView\_Username").toString());  intent.putExtra("USERID", map.get("USERID").toString());  dialog.cancel();  startActivityForResult(intent, 100);  }//训练按钮绑定跳转事件跳转到训练界面  });  button\_Delete.setOnClickListener(new OnClickListener() {  @Override  public void onClick(View arg0) {  ……  userAccessObject.deleteUser(user);  }//删除按钮绑定删除事件  });  }  //绑定右上角添加按钮单击事件：  button\_Register.setOnClickListener(new OnClickListener() {  @Override  public void onClick(View arg0) {  User user = new User();//声明新用户  user.setName(username);//设置新用户姓名  user.setIsTrained(false);//设置新用户训练状态  userAccessObject.addUser(user);//保存新用户到数据库  ……  }  } |

### 阈值设置

阈值设置模块允许用户查看当前阈值或调整阈值。阈值设置界面在载入时将先读取数据中保存的阈值，并显示到到SeekBar控件上，同时将SeekBar控件中数据停止改变事件onStopTrackingTouch绑定更新数据库阈值的方法。阈值设置界面如图5.4所示。

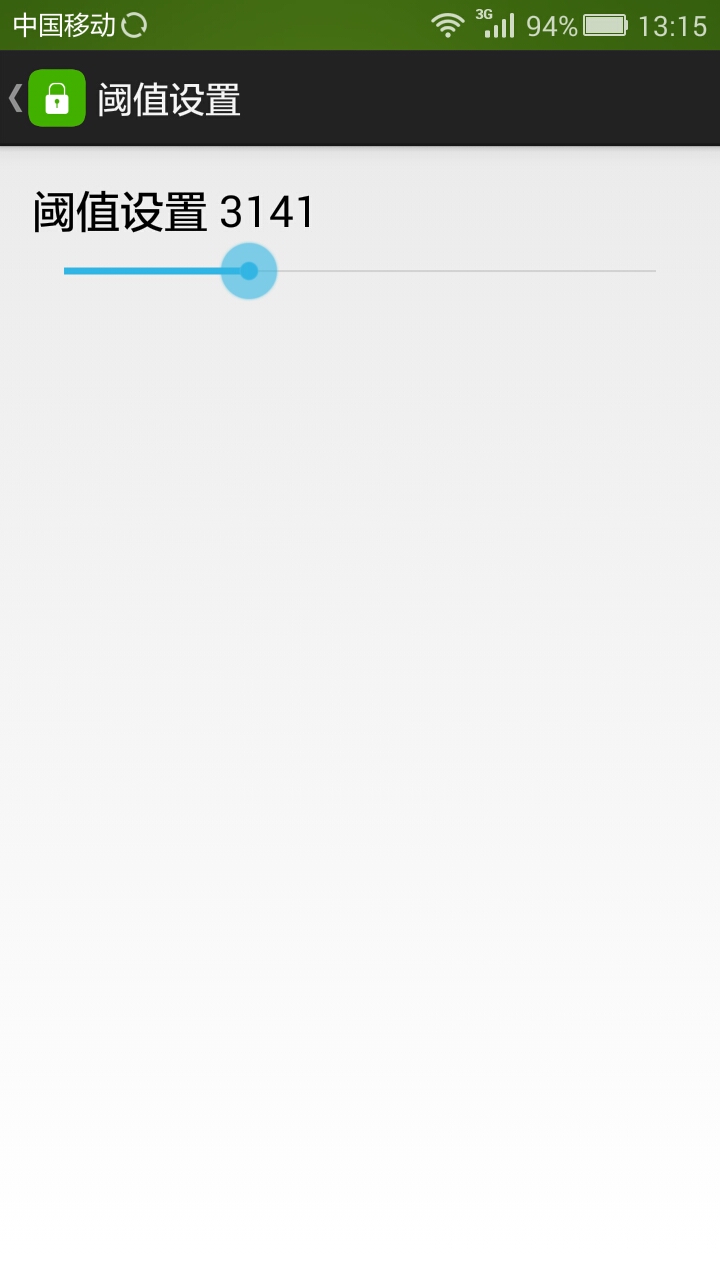


图5.4 阈值设置

实现代码：

ConfigActivity.java

|  |
| --- |
| private void bindListener() {  //绑定拖拉条变化事件  seekBar\_Threshold  .setOnSeekBarChangeListener(new OnSeekBarChangeListener() {  @Override  public void onStopTrackingTouch(SeekBar arg0) {  kVAccessObject.setThreshold(threshold+5000+ "");  }//停止改变时记录调整后的阈值，保存到数据库  @Override  public void onStartTrackingTouch(SeekBar arg0) {  }  @Override  public void onProgressChanged(SeekBar arg0, int arg1,  boolean arg2) {  threshold = arg1;  textView\_Threshold.setText(" " + threshold);  }//绑定空间中的数值到变量中  });  } |

### 修改密码

系统允许用户多次修改密码，以提高安全性。修改密码界面含有三个密码文本框和确认、清除按钮，分别用于输入旧密码，新密码，新密码确认，当用户按下确认后首先将校验旧密码，随后判断新密码和确认密码是否合法，若合法则保存至数据库中。修改密码界面如图5.5所示。



图5.5 修改密码

实现代码：

MainActivity.java

|  |
| --- |
| button\_ChangePassword.setOnClickListener(new OnClickListener() {  @Override  public void onClick(View arg0) {  ……  button\_Register.setOnClickListener(new OnClickListener() {  @Override  public void onClick(View arg0) {  if (loginAccessObject.validateUser(editText\_OldPassword  .getText().toString())) {  //校验旧密码是否正确  String password1 = editText\_NewPassword.getText()  .toString();  String password2 = editText\_ConfirmPassword  .getText().toString();  //获取文本框1和文本框2内容  if (password2.equals(password1)) {  if (password1.equals("")) {  //文本框或文本框2若为空，弹出提示  ToastUtil.show(getApplicationContext(),  R.string.password\_blank);  } else {  //更改成功，弹出提示  loginAccessObject.setPassword(password1);  ToastUtil.show(getApplicationContext(),  R.string.password\_change\_success);  dialog.cancel();  }  } else {  //文本框和文本框2若不同，弹出提示  ToastUtil.show(getApplicationContext(),  R.string.passwords\_diff);  }  } else {  ToastUtil.show(getApplicationContext(),  R.string.password\_error);  }  }  });  ……  dialog.show();//弹出当前修改密码对话框  }  }); |

### 说话人训练

说话人训练时，用户首先将设置语音提。用户可以选择系统内置的提示，也可以自定义提示。随后再进行三次相同的录音。界面上方有RadioButton和Spinner供用户选择提示类型和具体提示内容。界面下方有三次录音按钮，按下后开始录音，松开后录音结束。录音后得到3个wav音频文件，再将这3个wav音频文件通过HCopy进行批量特征提取，得到3个MFCC特征文件。最后在生成lab标签文件，并对这3个MFCC特征文件进行建模训练。训练界面如图5.6所示。



图5.6 说话人训练界面

实现代码：

TrainActivity.java

|  |
| --- |
| //开始录音并在3秒之后停止录音：  protected void startRecord(final int n) {  audioRecordFunc = AudioRecordFunc.getInstance();  //以单例模式调用录音实例  int result = audioRecordFunc.startRecordAndFile(wavPath, wavString,  rawString);//开始录音  ……  new Handler().postDelayed(new Runnable() {  public void run() {  alertDialog.cancel();  stopRecord(n);//延迟3秒后结束录音  }  }, 3000);  }  //停止录音进行特征提取并开始准备训练：  protected void stopRecord(int n) {  audioRecordFunc.stopRecordAndFile();  //结束录音并保存音频文件  switch (n) {  case 1:  ……  case 2:  ……  case 3:  button\_Record1.setEnabled(false);  button\_Record2.setEnabled(false);  button\_Record3.setEnabled(false);  NativeHTK.createMFCC(fileAccessObject, wavPath, userid,true);  //根据录音音频生成mfcc特征文件  final Context mContext=this;  new Handler().postDelayed(new Runnable() {  public void run() {  NativeHTK.train(fileService, userid);  //调用NativeHTK中的train静态方法开始训练  userAccessObject.trainUser(Integer.valueOf(userid.substring(2)));  ToastUtil.show(mContext, R.string.train\_end);  }  }, 1000);//延迟1秒后开始训练该用户  break;  default:  break;  }  } |

生成相关文件并开始训练。

实现代码：

NativeHTK.java

|  |
| --- |
| // MFCC特征提取：  public static void createMFCC(FileAccessObject fileAccessObject, String wavPath,  String userid, boolean isTrain) {  String wavlist = fileAccessObject.createWavList(wavPath, userid, isTrain);  //生成音频批量列表文件  HCopyFunc.exec(fileAccessObject.getConfigFilePath(), wavlist);  //生成mfcc特征文件  }  执行HTK工具箱中的HInit、HRest开始训练：  public static void train(FileAccessObject fileAccessObject, String userid) {  String labUserPath = fileAccessObject.createLab(userid);  String protoFile = fileAccessObject.createProto(userid);  //生成proto文件并获得路径  String trainlist = fileAccessObject.createTrainList(userid);  //生成trainlist批量训练文件并获得路径  HInitFunc.exec(trainlist, fileService.getHmm0Path(), protoFile, userid,  labUserPath);  //执行HInit方法初始化HMM模型  HRestFunc.exec(trainlist, fileService.getHmm1Path(),  fileService.getHmm0Path() + "/hmm\_" + userid, userid,  labUserPath);  //执行HRest方法第一次迭代HMM模型  HRest2Func.exec(trainlist, fileService.getHmm2Path(),  fileService.getHmm1Path() + "/hmm\_" + userid, userid,  labUserPath);  //执行HRest2方法第二次迭代HMM模型  } |

### 说话人测试

说话人测试界面有两种解锁方式，分别是数字密码方式和语音解锁方式。当选择数字密码方式时，用户输入密码即可解锁。当选择语音解锁状态时，需要说话人先声明身份，再进行录音。语音解锁界面中央有一个ListView控件，每列的左边显示有已训练的用户名，右边则有一个录音按钮。录音时按下按钮开始录音，放开后结束录音。语音解锁界面的下方可以切换解锁方式录音后得到wav音频文件，其通过HCopy进行批量特征提取，得到MFCC特征文件。最后利用HTK工具箱中的HVite进行识别，若结果与声明的相同且似然概率符合阈值设置要求则测试通过，否则测试不通过。测试界面和顺序如图5.7所示。



图5.7 说话人测试界面 左侧选取以声明说话人身份 右侧开始录音识别

实现代码：

AuthActivity.java

|  |
| --- |
| //绑定已训练的说话人列表：  private void bindView() {  ……  userMapList = list2Map(userAccessObject.getTrainedUserList());  //获取已训练用户列表数据，并转换成HashMap格式  if (userMapList.size() < 3) {//若训练用户小于3则无法使用语音解锁  isSoundMode=false;  }  adapter = new AuthListViewAdapter(this, userMapList,  R.layout.item\_auth, new String[] { "textView\_UserName",  "imageButton\_UnLock", "userId" }, new int[] {  R.id.textView\_UserName, R.id.imageButton\_UnLock });  //初始化适配器，绑定已训练用户列表数据和相关视图xml文件  listView\_User.setAdapter(adapter);//绑定适配器到listview列表  } |

AuthListViewAdapter.java

|  |
| --- |
| //声明说话人身份及开始录音按钮绑定单击事件：  class ButtonListener implements OnClickListener {  ……  @Override  public void onClick(View v) {  ……  startRecord(userId);//绑定列表视图中的某一项中的录音按钮的单击事件  }  }  //开始录音并在3秒之后停止：  protected void startRecord(final String userId) {  ……  audioRecordFunc = AudioRecordFunc.getInstance();  //以单例模式调用录音实例  int result = audioRecordFunc.startRecordAndFile(wavPath, wavString,  rawString);//开始录音  ……  new Handler().postDelayed(new Runnable() {  public void run() {  stopRecord(userId);  alertDialog.cancel();  }  }, 3000);//延迟3秒后结束录音  }  //停止录音进行特征提取并准备识别：  protected void stopRecord(String userId) {  audioRecordFunc.stopRecordAndFile();//终止录音并保存音频文件  NativeHTK.createMFCC(userAccessObject, wavPath, userId, false);  //根据录音音频生成特征文件  NativeHTK.test(fileAccessObject, userAccessObject, userId);  //调用NativeHTK中的test静态方法开始验证  verify(userId);  //开始验证  }  //验证录音结果是否符合声明，是则通过测试，否则报出提示：  protected void verify(String userId) {  String result = fileAccessObject.parseRecoMlf();  //获取解析HVite识别后的字符串  String [] r=result.split("-");  String r1=r[0];  Double r2=Double.parseDouble(r[1]);  KVAccessObject kVAccessObject = KVAccessObject  .getInstance(mContext);  //提取结果用户名和似然概率  Double threshold=Double.parseDouble(kVAccessObject.getThreshold());  //获取数据库中设置的阈值  if (r1.equalsIgnoreCase(userId)&&(r2<threshold)) {  //比较用户名和阈值，若符合要求则合法，解锁界面  int id = Integer.parseInt(userId.substring(2));  kVAccessObject.getThreshold();  userAccessObject.verifyUser(id);  ToastUtil.show(mContext, R.string.unlock\_success);  ToastUtil.show(mContext, userId+" "+r1+" "+ r2+" "+threshold);  AuthActivity authActivity = (AuthActivity) mContext;  authActivity.unLock();  authActivity.finish();  } else {  //否则仅弹出提示  ToastUtil.show(mContext, R.string.unlock\_failure);  ToastUtil.show(mContext, userId+" "+r1+" "+ r2+" "+threshold);  } } |

实现代码：

NativeHTK.java

利用HTK中的HParse生成词典网格文件，再用HVite进行识别测试：

|  |
| --- |
| public static void test(FileAccessObject fileAccessObject, UserAccessObject userAccessObject,  String userId) throws IOException, InterruptedException {  List<User> userList = userAccessObject.getTrainedUserList();  String gramFile = fileAccessObject.createGram(userList);  //生成语法文件，并返回路径  HParseFunc.exec(gramFile, fileService.getSlfFilePath());  //调用HParse生成语法网格文件  String dictFile = fileAccessObject.createDict(userList);  //生成词典文件，并返回路径  String allMmfFile = fileAccessObject.createAllMmf(userList);  //生成全体HMM文件，并返回路径  String netSlfFile = fileAccessObject.getSlfFilePath();  //获取语法网格文件路径  String hmmListFile = fileAccessObject.createHmmListFile(userList);  //生成HMM列表文件，并返回路径  String resultFile = fileAccessObject.getResultFilePath();  //获取结果输出文件路径  String mfcFile = fileAccessObject.getMfccPath() + "/" + userId + ".mfc";  //获取要验证的mfcc特征文件路径  String hViteE = fileAccessObject.getHviteE();  //获取要调用的的HVite二进制可执行文件路径  HViteFunc.exec(hViteE, allMmfFile, resultFile, netSlfFile, dictFile,  hmmListFile, mfcFile);  //调用HVite进行验证  } |

## 实验和测试



### 实验遇到的问题和解决方法

在进行系统的需求分析和代码编写以及测试的过程中遇到了一些问题，经过查询资料、仔细分析和不断的调试，问题也得到了解决，所遇到的主要的问题如下。



#### HTK工具箱使用问题

HTK工具箱分为两个部分，分别是HTKLib和HTKTool。HTKLib是HTK工具箱的核心类库，包含HShell命令处理、HDcit词典、HUtil通用计算程序等。而HTKTool中的每个HTK工具是分别需要完整的HTKLib。在一开始编译Andorid平台上的HTK动态库时，并没有意识到这一点，而是将所有的HTKTool共享使用一个HTKLib，因此在实验时，当执行了一个HTKTool工具后在次执行时会出现内存溢出或占用的情况，经过仔细分析后，将所有的HTKTool各自生成独立使用的HTKLib，或是将HTKTool编译成可在Android平台上执行的二进制文件，就可以解决内存溢出的问题。

#### 录音计时问题

系统在最初设计时，训练时间和测试时间是固定的，程序较为稳定，但是识别率较低，因为在一段音频中语音中时间位置并不固定。之后为了提高识别率，同时改善用户在语音时的用户体验，将录音和测试时的用户交互的时间更改为用户可控的。

改进后，用户在按下按键时开始录音，放开按键时，录音结束。

改进后，音频中的用户语音更为紧凑，大大地提升了识别率。

#### 语音模型问题

在系统需求分析时，所有的说话人的语音文本被设计成是相同的，这样就导致了在系统初步编码完毕之后发现有识别率较低的问题，无法准确地识别出说话人。之后在老师的指导下，将不同的说话人的语音文本被设计成不同的，较大地提高了识别率，同时内置了一个噪声模型作为噪声说话人，使得系统的鲁棒性更强。

尽管不同的说话人的文本被设计成不同的，但是当文本被非法的用户知晓时，通过系统验证的可能性却有可能存在，因此需要用户对自己的语音文本保密。

#### 阈值问题

修改了语音模型后，不同的说话人的文本被设计成不同的，这样就存在了当文本被非法的用户知晓时，通过系统验证的可能性却有可能存在。

因此设置了阈值判断，当HTK识别后的结果的似然概率大于某一个值时，即使结果可能是合法的，系统依旧会拒绝这个用户，使得系统的安全性得到保障。同时还将不同用户的阈值分开设置，使得不同的用户可以自定义各自的阈值，进一步提升识别率和安全性。

## 展望与总结



### 总结

本课题下的系统是面向Android平台手机用户的说话人识别系统的，可以丰富Android平台的安全性，解决传统的文本密码和图案认证的不安全性。

系统开发之后，之前在需求分析时定下的目标基本完成。

系统的界面美观大方，功能详实有效。用户管理模块界面简约清晰、功能完整，训练和测试模块界面操作便捷、提示丰富、功能有效、识别率较高。

本人自己在这次开发过程中也有不少体会和收获。技术方面，掌握了Android开发的基本知识，例如Activity、Service、Broadcast Receiver、Content Provider四大组件的生命周期和使用，以及组件之间的数据交互和数据存储。还掌握了Android NDK编译工具的使用，例如Makefile的编写和JNI接口的封装。识别环节上还了解了说话人识别的主要流程及所涉及到的算法，例如HTK工具箱的使用和HMM隐马尔科夫模型等等。

### 展望

综合实验中遇到的问题，本系统还是有许多方面可以进入深入研究。例如在用户交互上面做得更友好一些，引用一些外观资源；或是与文本无关的方案结合，当前使用的是基于HMM隐马尔科夫模型的HTK工具箱，对训练语音要求较高；亦或是在识别率方面能有更大的提升以避免在不同的说话人使用相同的语音文本的情况下识别率异常的情况。

## 参考文献

1. Cambridge University Engineering Department (CUED). HTKBook[EB/OL]. http://htk.eng.cam.ac.uk/docs/docs.shtml. 2006
2. Anthony Larcher, Kong Aik Lee, Bin Ma等. Text-dependent speaker verification: Classifiers, databases and RSR2015[J]. Speech Communication 2014，60：56–77
3. Zhizheng Wu,Nicholas Evans, Tomi Kinnunen等. Spoofing and countermeasures for speaker verification: A survey [J]. Speech Communication 2014，60：130–153
4. wikipedia.org. Android[EB/OL]. http://en.wikipedia.org/wiki/Android\_(operating\_system). 2015
5. apache.org. Commons IO Javadoc[EB/OL]. http://commons.apache.org/proper/commons-io/download\_io.cgi. 2014
6. The SQLite Development Team. Appropriate Uses For SQLite[EB/OL]. http://www.sqlite.org/whentouse.html. 2015
7. 李刚.疯狂Android讲义.第二版[M].北京：电子工业出版社，2013.1-343.
8. (美)Bruce Eckel 著 陈昊鹏 译.Java编程思想[M].北京：机械工业出版社，2007.1-649
9. 李建平 林劼.生物特征的安全计算理论与技术[M].成都：电子科技大学出版社，2011，1-270
10. android.com. Android NDK [EB/OL]. http://developer.android.com/tools/sdk/ndk/index.html. 2015
11. android.com. Android Developer Tools[EB/OL]. http://developer.android.com/tools/help/adt.html. 2015
12. 陈泉金.基于HTK的连续语音识别技术研究[D].南京：南京邮电大学，2010
13. 安徽大学计算机智能与信号处理教育部重点实验室.基于HTK的汉语语音售票系统的设计与实现[D].合肥：安徽大学，2010
14. M.E. Forsyth, A.M. Sutherland, J.A. Elliott, M.A. Jack. HMM speaker verification with sparse training data on telephone quality speech[J]. Speech Communication 1993，13：411–416
15. Sunita Chauhan , Ping Wang . A computer-aided MFCC-based HMM system for automatic auscultation[J]. Expert Systems with Applications. 39, 2, 1 2012, 2157–2165
16. 赵力.HMM 在说话人识别中的应用[J].电路与系统学报，2001-9，6（3）：51-57
17. jamesju.基于HTK语音工具包进行孤立词识别的使用教程[EB/OL]. http://my.oschina.net/jamesju/blog/116151. 2013

## 致谢

谨在本系统开发完成之际，感谢给予我帮助和鼓励的人们。在历时4个月的过程中，我遇到了无数的困难和障碍，都在同学、老师和家人的帮助下度过了。尤其要感谢我的论文指导老——李冬冬老师，在整个论文的选题、研究、设计实现和撰写过程中，都给我了我精心的指导、热忱的鼓励和支持。李老师曾发表过许多论文和专利，在说话人语音识别识别方面有着丰富的研究经验，多次为我批阅文章并对系统提出修改意见，她的精心点拨为我开拓了研究视野，修正了写作和设计思路，对论文的完善和质量的提高起到了关键性的作用。其次要感谢我的大创队友，张耀和谢南杰，他们在解决系统的设计和实现问题时，给予我的帮助和建议。最后感谢被我引用了文献的数位学者以及网上各种资料的佚名提供者们。