媒体与认知课程实验

基于 Kaldi 的连续语音识别

组长：刘　前　学号：2014011216
组员：杨泽龙　学号：2014011208
组员：刘　宇　学号：2014011198

目录

1 实验目标

学习语音识别的基本原理和相关知识，掌握语音识别的整体框架，理解关键的隐式马尔科夫模型 (Hidden Markov Model, HMM) 和高斯混合模型 (Gaussian Mixture Model, GMM) 技术。熟悉常用的开源语音识别工具箱 Kaldi、HTK，解析工具箱中关键脚本文件的代码，并能够利用工具箱分别基于英文朗读数据库 TIMIT 和中文朗读数据库 THCHS-30，运行得到相应的语音识别模型，同时对系统参数、系统性能进行调试，对所得实验结果进行讨论和评估。最后利用训练得到的模型搭建离线系统及 demo，实现用户体验良好、识别准确率合格的中文连续语音识别系统。

2 实验内容

- 熟悉语音识别原理，利用 Kaldi 相关脚本基于 TIMIT 数据训练得到英文语音识别模型；
- 熟悉中文语音识别框架，利用 Kaldi 脚本在 THCHS-30 数据库上训练模型；
- 基于训练得到的模型搭建中文连续语音识别系统：

  (1) 实现离线系统：能够实现对既有语音文件的识别功能，同时可以获取在测试集上的识别准确率等评价参数；

  (2) 搭建 demo：设计用户交互式界面，包含离线识别和在线识别两部分功能。通过用户点击相应的功能按钮，离线识别输入一段保存好的音频文件，输出相应的文本；在线识别先实时采集语音，在利用训练好的模型进行识别并最终将文字显示在界面上。

- 完成实验报告：详细描述实验原理简介、实验系统搭建、实验结果、结果分析、demo 搭建方法简介等。

3 实验原理

语音识别中关键方面是对自然语言的识别和理解：能够将连续语音切分成离散的、较短的单位，能够识别语音对应的含义即语义。因此必须首先将连续的语音分解为词、音素、状态等单位，其次还需要建立一个理解语义的规则 [1]。

上述过程是语音识别中最重要的两个模型，声学模型和语言模型，分别对应于语音到音节概率的计算和音节到字概率的计算。此外，语音识别需要考虑环境噪声的干扰，噪声的混入将影响识别效果；识别过程中还需要考虑不同人说话方式随时间变化的因素，尤其是中文识别中语音的模糊性对识别效果影响较大：字词的发音可能受到上下文的影响改变了重音、音调等。

基于上述语音识别中的关键因素，语音识别可基本分为前期处理、声学特征选择和提取、建立声学模型、建立语音模型四个部分。

3.1 前期处理

在进行对语音分割和特征提取之前，需要先对原始语音进行处理，消除部分噪声和不同人说话带来的影响，避免影响后续的关键特征提取。同时希望能够对语音信号进行增强，提高其信噪比，一般采用各种滤波方法实现噪声的消除与减弱。

3.2 声学特征选择和提取

语音的声学特征是实现连续语音分割的依据，选择不同的声学特征如梅尔频率倒谱系数 (Mel Frequency Cepstral Coefficents，MFCC)、倒谱均值和方差归一化 (Cepstral Mean and Variance Normalization，CMVN) 等直接决定了后面的声学模型建立和分割 [2]。更具体地说，输入语音“今天天气真好”，则在语音波形中，“今天” 两个音的波形由于共同表示语义则非常接近，与后面音的波形有一定距离，这是最简单的声学特征。

3.3 建立声学模型

一般采用隐马尔科夫模型（HMM）进行声学建模，先将连续语音分为音素和状态，然后得到相应状态的观察概率和转移概率，构建状态网络。根据考虑的音素数量不同可以建立单音素模型（Monophone HMMs）、三音素模型（Triphones HMMs），三因素模型考虑前一音素和后一音素的影响，称为 Triphones。基于 HMM 的声学模型构建可参见图1。

3.4 建立语音模型

一般利用概率统计的方法来解释语言单位内在的统计规律，结合构建的音素词典，构建语言模型。简要来说，就是利用该模型从状态网络中寻找与声音（单词）最匹配的路径，实现语音识别（解码过程）[3]。

整个连续语音识别过程见图2。

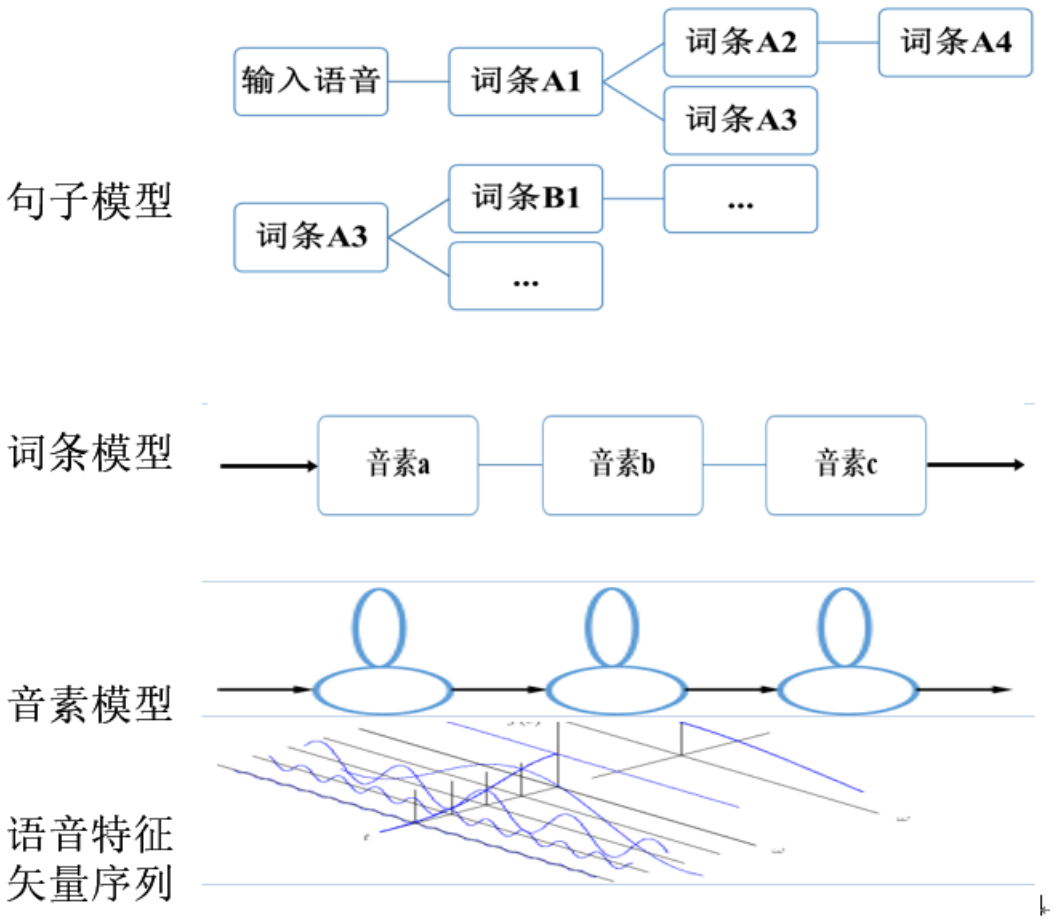

图 1: 基于 HMM 的声学模型

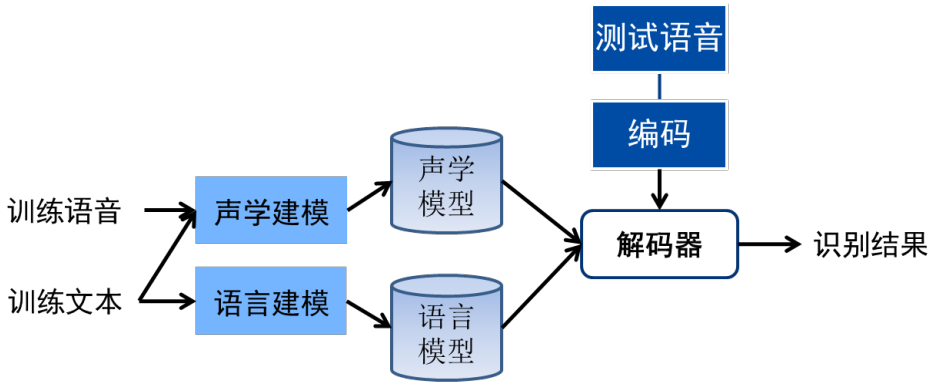

图 2: 连续语音识别基本原理

4 实验系统搭建

4.1 Kaldi 语音识别工具箱

4.1.1 基本介绍

Kaldi 是一个开源的语音识别工具箱，基于 C++ 编写，可在 Windows 和 Linux 平台下编译[1]。该工具箱含有丰富的声学模型，线性变换 LDA、HLDA 模型，自适应最大似然线性回归（MLLR）模型，支持 GMM，DNN 等机器学习训练模型。

4.1.2 Kaldi 语音识别系统具体流程

在使用 Kaldi 工具箱之前，需要对其环境进行配置，这在 Kaldi 的相关安装文件上有详细的说明[2]。配置好环境后[3]，即可利用 Kaldi 工具箱在指定的数据集上进行训练和测试。训练和测试模型对应的关键脚本为 run.sh，基本涵盖了全部的训练过程。基于 Kaldi 的连续语音识别整体流程如图3。

图 3: 基于 Kaldi 工具箱的连续语音识别过程

图3中除了简单的单音素模型和三音素模型，还可以训练 DNN 等模型，但是需要 GPU 等条件。根据图3的流程可以在 TIMIT 数据库和 THCHS-30 数据库上分别进行实验。

4.2 TIMIT 数据库

4.2.1 数据库基本介绍

TIMIT（The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus）是专门用于开发和评估自动语音识别的数据集之一，包含了 6300 句语音数据，共 630 个人，每个人说 10 个句子，这 630 人来自于美国的 8 个不同的地区。表1展示了各地区人数和性别占比等信息。

[1]本实验所有操作和结果均在 5G 内存、1 核 CPU 的 Ubuntu 虚拟机内运行得到。

[2]`https://github.com/kaldi-asr/kaldi/blob/master/tools/INSTALL`

[3]使用 Kaldi 中的 yesno 样例进行测试，判断是否安装编译成功。

方言地域 (dr)	男性人数 (比例)	女性人数 (比例)	占总体比例
dr1: New England	31 (63%)	18 (27%)	49 (8%)
dr2: Northern	71 (70%)	31 (30%)	102 (16%)
dr3: North Midland	79 (67%)	23 (23%)	102 (16%)
dr4: South Midland	69 (69%)	31 (31%)	100 (16%)
dr5: Southern	62 (63%)	36 (37%)	98 (16%)
dr6: New York City	30 (65%)	16 (35%)	46 (7%)
dr7: Western	74 (74%)	26 (26%)	100 (16%)
dr8: Army Brat	22 (67%)	11 (33%)	33 (5%)
总计	438 (70%)	192 (30%)	630 (100%)

表 1: TIMIT 数据库中语音的地域及性别分布

4.2.2 数据目录结构

TIMIT 语料库的具体数据主要分为训练 (train) 数据和测试 (test) 数据，其目录结构可以总结为表2[4]。

语料库	timit
用法	train | test
方言种类	dr1 | dr2 | dr3 | dr4 | dr5 | dr6 | dr7 | dr8
性别	m | f
说话者 ID	< 说话者缩写 > <0-9 任意数字 >
句子 ID	< 文本类型 > < 句子编号 >，其中，文本类型：sa | si | sx
文件类型	wav | txt | wrd | phn

表 2: TIMIT 数据库目录结构

4.2.3 TIMIT 模型训练流程

首先修改数据文件的路径 (cmd.sh)，并在脚本 run.sh 内对应的位置进行路径及参数的修改。模型训练的过程主要包括数据格式转换、特征提取、训练单音素模型、训练三音素模型、解码等步骤。在 TIMIT 数据库上进行模型训练的流程可以总结为图4。具体的识别结果和分析见**实验结果与分析**部分。

[4]具体各文件类型详见官方描述：`https://catalog.ldc.upenn.edu/docs/LDC93S1/timit.readme.html`

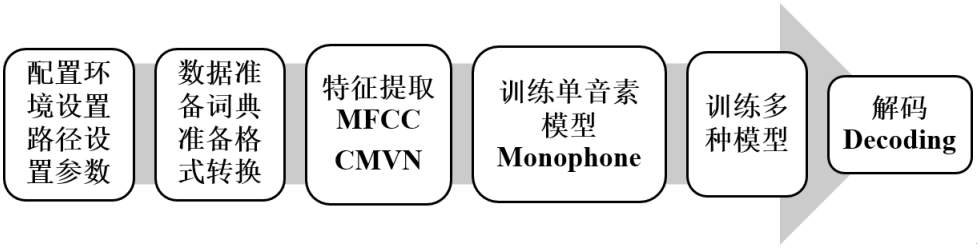

图 4: TIMIT 数据库模型训练流程

4.3 THCHS-30 数据库

4.3.1 数据库基本信息

THCHS-30(Tsinghua Chinese 30-hour database) 是一个免费中文语料库，由清华大学信息技术研究院语音和语言技术中心（CSLT）的王东教授开发，其前身为清华普通话连续语音语料库（TCMSD)。

THCHS-30 包含 30 小时的语音信号，在安静的办公室环境下，由 40 名志愿者录制而成。这些志愿者大多为清华大学在校学生，普通话标准，采样率为 16000Hz。整个语料库由四个子集组成，四个子集分别录制相同的 500 个句子。前三个子集用于训练，第四个子集用于测试。四个子集具体信息如表3所示。

Data Set	Speaker	Male	Female	Age	Utterance	Duration(hour)
Training	30	8	22	20-55	10893	27.23h
Test	10	1	9	19-50	2496	6.24h

表 3: THCHS-30 语料集数据

4.3.2 数据库目录结构

THCHS-30 数据目录结构[5]如表4所示。

4.3.3 系统脚本文件说明

基于 THCHS-30 数据训练中文连续语音识别模型时，主要涉及的脚本文件及相应的功能如表5所示。

[5] http://data.cslt.org/thchs30/standalone.html

|wav
| |-train
| |-test
| |-noise
| |-test-noise
|doc
|lm
wav : signals including the training/cv/test sets.
doc : transcripts, reference results, etc.
lm : language models including word based and morpheme based.

表 4: THCHS-30 数据目录结构

脚本名称	功能描述
cmd.sh	设置环境变量 (相当于全局)
run.sh	用于运行的主脚本，会调用很多其他的 shell 文件
path.sh	对程序中需要用到的文件路径进行设置

表 5: THCHS-30 脚本文件说明

5 Demo 搭建

5.1 设计思路

本实验要求使用训练的模型，实现离线语音识别和实时语音识别两部分功能，并集成在图形用户界面 (GUI) 上。离散语音识别比较容易实现，只需将录制好的音频作为训练模型的输入，使用模型得到识别结果，再输出到 GUI 相应的位置即可。对于在线语音识别系统，实现思路主要包括三个功能模块：语音的采集 (录音)、使用模型进行语音识别、将识别结果输出到界面，后文将主要对在线语音识别系统的实现方法进行详细说明。

5.2 软件及环境介绍

本实验设计的 GUI 是在 Ubuntu 16.04 操作系统下完成。由于 Kaldi 中模型的训练和语音识别需要运行 shell 脚本，而 Python 调用 shell 非常方便，因而 demo 的搭建也主要基于 Python 来实现。表6给出了各功能模块主要涉及的 Python 库，下文将在各功能模块内具体介绍各库函数的使用方法。

功能模块	Python 库函数
语音的采集 (输出)	Pyaudio, PyQt4, threading
使用模型识别语音	wav, pylab, os
将识别结果输出	sys, PyQt4

表 6: 图形用户界面各功能模块使用的 Python 库

5.3 语音的采集

对于在线语音识别系统，需要能够采集实时的语音，调用函数使用麦克风录制音频，并存成固定格式的 wav 文件，用于之后的语音识别，核心的 Python 库是 Pyaudio。

5.3.1 参数设置

由于 Kaldi 在识别语音文件时，对 wav 文件的格式有明确的限制。因而在使用 pyaudio 录制音频时，首先需要设置以下参数：

- 量化位数：用若干 bit 表达一次采样所采集的数据，通常有 8bit、16bit、24bit 和 32bit 等。本实验设置量化位数为 16bit。
- 声道数：可以是单声道或双声道。Kaldi 要求设置为单通道。
- 采样频率：一秒内对声音信号的采样次数。Kaldi 要求采样率为 16kHz。

5.3.2 单次录音的实现

在用户图形界面中，设计一个功能按键，用于控制录音的开始与结束。设置一个全局 BOOL 变量 RECORDING，指示当前是否需要进行录音，RECORDING 的初始值为 FALSE。第一次点击，RECORDING 变为 TRUE，此时开始录音，调用一个录音线程 (python 的 threading 库)，使用 pyaudio 库中的 writeframes() 函数将实时的语音加到声音流 (stream)。当使用者再次点击按钮时，指示录音结束，RECORDING 变为 FALSE，录音线程在 RECORDING 变量后立即停止录音，将声音流 (stream) 写入 wav 文件。

5.3.3 重复录音的实现

上述操作实现了单次录音，但是可想而知用户的体验并不好，因为这要求每次录音之后必须重启界面。为了改善用户体验，我们对录音按键函数进行了修改，将其设置为生成器 (generator)。Python 的生成器非常实用，简单概括来说，生成器能够使得函数在不同情况下调用时返回值或者执行的操作不同，每个不同的操作之间用 yield 分隔开。

基本以上原理，主要在单次录音下实现了以下改变：初始时 RECORDING 为 FALSE，第一次点击按钮变量值变为 TRUE，开始调用录音线程进行录音，第二次点击后变量值变为

FALSE，此时停止录音 (进行后续的识别)，两次点击操作之间用 yield 分隔开。由于生成器的存在，下次再点击按钮，迭代操作下 RECORDING 再次变为 TRUE，依次类推，从而实现了多次录音和语音识别，使用体验明显改善。

5.4 识别录制的语音

采集到实时的语音之后，需要使用训练好的 Kaldi 语音模型对 wav 文件进行识别。我们使用的是 Kaldi 的 voxforge 中自带的 online_demo 文件夹，根据需要进行了相应的改动，实现了对语音模型文件的调用。这里使用的语音模型由 THCHS-30 训练得到，训练得到的最好的模型是 tri4b。本实验中语音识别的核心功能是由训练得到的 tri4b 模型文件实现的。

在进行语音识别前，需要先将录制好的音频移动到指定的文件夹，再使用 Python 中 os 库中的 system 函数调用 shell 脚本文件 run.sh。经过一段时间后，会在命令行输出得到识别的中文结果。

5.5 输出识别结果

语音识别的大部分功能已经实现完毕，也有较好的识别结果。但是仍然需要将识别的结果在用户图形界面上也显示出来。

5.5.1 文件操作

由于识别结果输出到命令行是由 Kaldi 中的 C++ 库函数决定的，考虑到操作的简单性，我们决定将命令行输出的结果先输出到文件中 (“> file.txt”)，然后只需将文件中识别的中文结果输出到界面中即可。

5.5.2 中文输出

我们设计 GUI 使用的是 PyQt4 库 [4], PyQt4 的字符默认编码与 Python 不同，因而一开始在界面上输出中的识别结果全是乱码。因而需要修改默认的编码方式 (“sys.setdefaultencoding('utf8')”)，即可实现中文的正常输出。

5.6 离线语音识别系统

离线语音识别只涉及实现在线语音识别的一小部分功能，因而不予赘述。我们只需在 GUI 中添加一个按钮，点击按钮之后，将离线的已录制好的音频移动到相应文件夹，即可调用相关模型及函数实现离线语音识别。

5.7 最终图形用户界面

总结来说，在线语音识别经过语音采集、使用模型进行语音识别、将识别结果输出到界面等步骤，结合离线语音识别的功能，再加上一些对界面的美观优化，最终实现的 GUI 如5所示。在**附录**中的 **demo 演示**部分将对 GUI 的具体使用和实现效果进行展示。

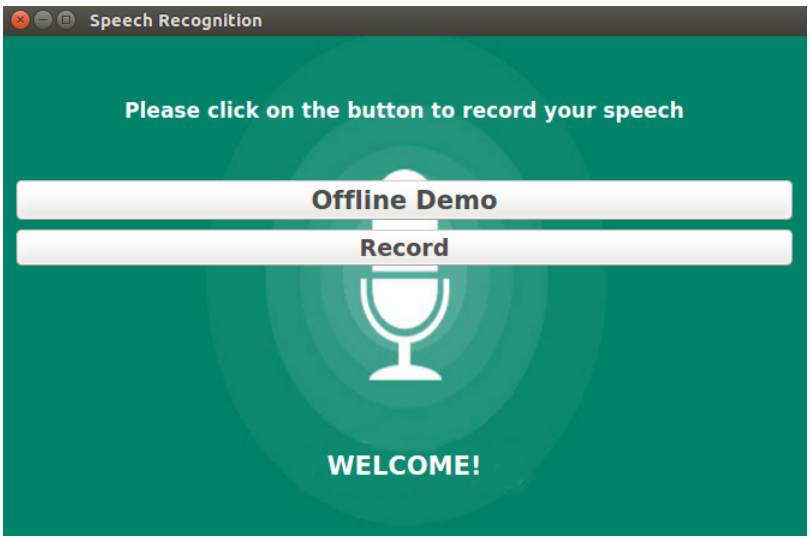

图 5: Demo：图形用户界面

6 实验结果与分析

6.1 TIMIT 结果与分析

对于 TIMIT 数据库，根据在测试集上的输出结果可以对不同模型进行评价，评价指标为词错误率 (Word Erorr, WER)。具体结果如表7。

模型	Word Error
tri1	26.0%
tri2	23.8%
tri3	21.6%
SGMM2	19.7%
MMI+SGMM2	19.6%

表 7: TIMIT 数据库目录结构

对比发现，基于三音素模型的 SGMM2 以及 SGMM2+MMI，相对于简单的三音素绑定状态系统在准确率上有 5% 左右的提升，识别效果有明显的改善。

6.2 THCHS-30 结果与分析

对于中文语音识别的性能，我们以测试组 (D 组) 第一个语音文件为例：D4_750.wav。该语音文件的正确内容为："苏北军的一些爱国将士马占山、李杜、唐聚武、苏炳艾、邓铁梅等也奋起抗战。"按照不同训练模型运行得出的测试结果如表8所示。

模型	D4_750.wav 的识别结果
mono	受北京理性爱国驾驶满载山你不长期武说弊案丹田每版扁奋起抗战
tri1	艘北京的一些爱国将是马占山泥杜盘据武苏比爱根甜美等也奋起抗战
tri2	苏北军礼写爱国将是马占山里杜唐据武苏并爱蹦甜美但也奋起抗战
tri3b	苏北军的一些爱国将士马占山里杜唐据武苏并爱奔甜美但也奋起抗战
tri4b	苏北京的一些爱国将士马占山里杜唐据吴素饼爱奔甜美但也奋起抗战

表 8: 不同训练模型的识别结果

由上面三种方法测试同一个语音文件的结果来看：mono 的结果很差，从 mono 的结果基本分辨不出原句的意思；tri1 相对于 mono 由很大改善，但细节上还是很不完善，没有看过原句也很难分辨出原句的意思；tri2 明显比 tri1 效果更好，对人名的翻译也比之前两种方法要好很多。tri3b 和 tri4b 则效果更好，其中 tri4b 的在线识别所需时间明显比 tri3b 要短，性能更好。以上三种方法都没法很好地识别并处理人名。对于 tri2 来说，尽管效果比较不错，但是两个地方的错误 ("的一些" 识别成 "礼写"，"等" 识别成 "但") 使得整体可读性大为下降。

查看运行所得的 RESULTS 文件，获得在测试集上的整体性能。词错误率如表9所示。

模型	Word Error
mono	50.76%
tri1	36.16%
tri2	32.55%
tri3b	29.65%

表 9: 不同训练模型的识别结果

图6是数据集自带的识别结果 (用于判断用户是否训练成功)。对比前四行部分，发现自己运行所得结果和数据库自带结果十分接近，说明 mono、tri1、tri2 和 tri3b 部分运行正确。

7 实验总结

本次试验，我们小组用时 8 周完成。首先通过阅读 HTK book 第三章内容，学习了 Kaldi 语音识别系统的理论知识，然后通过 Github 安装了 Kaldi，并先后通过运行 yesno 和 timit 两个数据库来验证 Kaldi 是否安装成功。为了实现中文连续语音识别，我们安装了 THCHS-30

```
#word task
%WER 50.88 [ 41280 / 81139, 506 ins, 2393 del, 38381 sub ] exp/mono/decode_test_word/wer_9_0.0
%WER 35.97 [ 29188 / 81139, 531 ins, 1041 del, 27616 sub ] exp/tri1/decode_test_word/wer_10_0.0
%WER 32.14 [ 26078 / 81139, 418 ins, 1057 del, 24603 sub ] exp/tri2b/decode_test_word/wer_10_0.0
%WER 29.47 [ 23913 / 81139, 396 ins, 864 del, 22653 sub ] exp/tri3b/decode_test_word/wer_10_0.0
%WER 33.65 [ 27300 / 81139, 483 ins, 1049 del, 25768 sub ] exp/tri3b/decode_test_word.si/wer_9_0.0
%WER 28.07 [ 22776 / 81139, 418 ins, 762 del, 21596 sub ] exp/tri4b/decode_test_word/wer_11_0.0
%WER 31.50 [ 25559 / 81139, 511 ins, 928 del, 24120 sub ] exp/tri4b/decode_test_word.si/wer_10_0.0
%WER 23.68 [ 19217 / 81139, 490 ins, 597 del, 18130 sub ] exp/tri4b_dnn/decode_test_word/wer_7_0.0
%WER 23.44 [ 19022 / 81139, 465 ins, 576 del, 17981 sub ] exp/tri4b_dnn_mpe/decode_test_word_it1/
wer_7_0.0
%WER 23.35 [ 18947 / 81139, 408 ins, 631 del, 17908 sub ] exp/tri4b_dnn_mpe/decode_test_word_it2/
wer_8_0.0
%WER 23.30 [ 18904 / 81139, 406 ins, 622 del, 17876 sub ] exp/tri4b_dnn_mpe/decode_test_word_it3/
wer_8_0.0
```

图 6: THCHS-30 自带的模型训练结果 (参考)

中文语音数据库，训练了 mono、tri1、tri2、tri3b 和 tri4b 五个模型，并最终选择了 tri4b 来搭建最终的 demo。

我们采用 online-demo 进行在线解码，完成了离线语音识别测试。并使用 PyQt 搭建 demo 界面，实现了在线语音识别功能。在最终的展示中，我们的在线语音识别测试在准确性和识别速度两个方面都达到了预期的效果。

在实验的进行的八周时间里，小组三名成员每天都在小组微信群中讨论实验相关内容，并经常聚在一起研讨和设计，共同推进实验进度。我们轮流完成每周实验进度报告，实现了工作量在时间和小组成员层面上的合理分配，并按计划顺利完成了各项工作。通过这次试验，我们小组三位成员的友谊变得更加深厚、合作能力也得到了明显提高。

本次试验不仅加深了我们对《媒体与认知》课程理论知识的理解，还使得我们的文献阅读、时间安排、团队合作以及问题解决等能力有了全面提升。我们在本次试验中投入了很多的时间精力，但这都是值得的，小组成员都感觉收获很大，受益匪浅。我们各个小组由于每周都提交了进度总结，并且按计划完成各项任务，因而当其他项目的同学由于前期投入时间不足导致后期工作量陡然升高的时候，我们的工作量基本没有什么太大的变化，并且也没有出现小组中只有一位同学包揽全部工作量的情况，进度安排十分合理。每周进度报告也为最终的实验报告分担了很多工作量，避免了重复劳动。刘艺助教在实验中给予了很大的指导和帮助，在此表示由衷的感谢!

参考文献

[1] Young S, Evermann G, Gales M, et al. The HTK book[J]. 2006.

[2] 俞栋, 邓力, 俞凯, 等. 解析深度学习语音识别实践 [M]. 电子工业出版社, 2016.

[3] 杜俊, 高天, 戴礼荣, 等. 连续语音识别方法及系统:, CN106157953A[P]. 2016.

[4] Summerfield M. Rapid gui programming with python and qt: the definitive guide to pyqt programming[M]. Prentice Hall, 2008:89-94.

8 附录一：文件清单及说明

本次实验主要涉及 timit 和 thchs30 两部分代码。后文将对各部分主要代码进行说明。

8.1 TIMIT 模型训练相关文件

脚本文件名	功能介绍
cmd.sh	设置训练、解码的命令 (本地运行)%
path.sh	路径的设置%
run.sh	主要执行的脚本，训练得到模型

表 10: timit 文件夹内的代码

8.2 THCHS-30 模型训练相关文件

THCHS-30 的全部代码文件在 thchs30 文件夹内（其中训练所用的数据、训练得到的模型文件 HCLG.fst 等因文件太大未上传)。thchs30 文件夹包含 s5 和 online-demo 两个文件夹，现对两个文件夹内的代码分别加以说明。

脚本文件名	功能介绍
cmd.sh	设置训练、解码的命令 (本地运行)%
path.sh	路径的设置%
run.sh	主要执行的脚本，训练得到模型

表 11: thchs30 文件夹下 s5 文件夹内的代码

8.3 demo 设计的相关代码

文件 (夹) 名	功能介绍
run.sh	主要执行的脚本，调用训练的模型进行语音识别
demo.py	课堂展示使用的 demo 界面，实现在线语音识别功能
demo2.py	在课堂展示的基础上添加了离线语音识别功能
practice 文件夹	本次实验尝试过程中编写的半成品
online-data 文件夹	demo 设计中需要的背景图片、测试语音等

表 12: thchs30 文件夹下 online-demo 文件夹内的代码

9 附录二：Demo 演示

9.1 初始界面

在 Ubuntu 命令行下输入命令“python demo2.py”运行 python 脚本，即可看到图7所示的界面。

9.2 离线语音识别

9.2.1 开始识别

点击按钮“Offline Demo”，进行离线语音识别 (仍以测试数据 D4_750 为例)，显示图8。

9.2.2 获得离线语音识别结果

等待一段时间 (约 10 至 20 秒)，界面显示识别得到的结果 (见图9)。D4_750.wav 的正确内容为：“苏北军的一些爱国将士马占山、李杜、唐聚武、苏炳艾、邓铁梅等也奋起抗战”。可以看出，识别效果还算不错。

9.3 在线语音识别

9.3.1 开始录音

点击“Record”按钮开始录音，见图10。

9.3.2 结束录音并识别

在图10中点击“Stop”按钮结束录音，系统立即进行语音识别，见图11。

9.3.3 将结果输出到界面

等待一段时间 (约 10 至 20 秒)，界面显示识别得到的结果 (见图12)

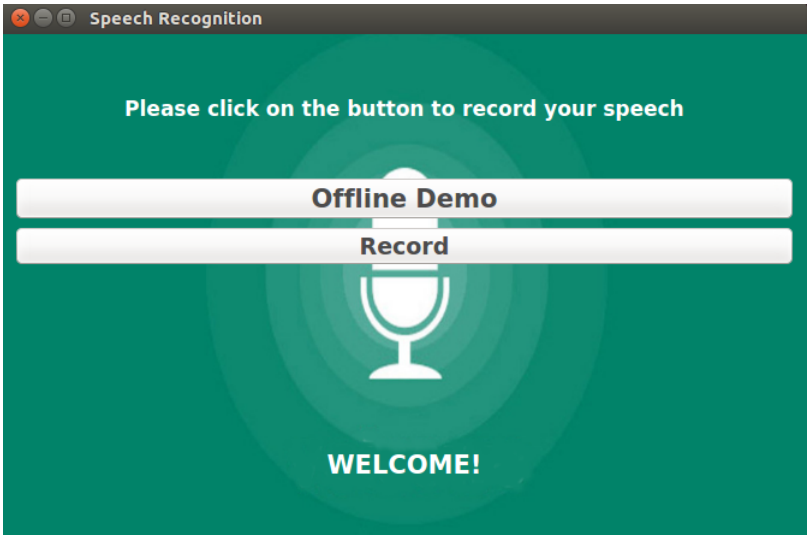

图 7: 起始界面

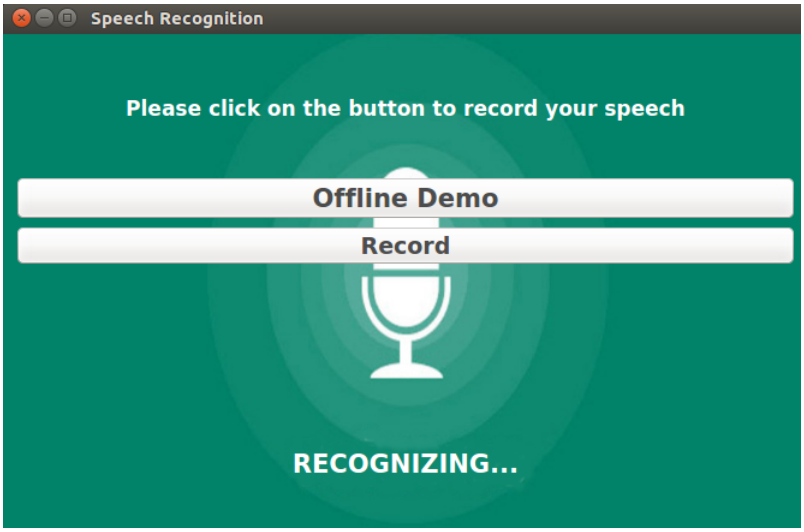

图 8: 离线语音识别

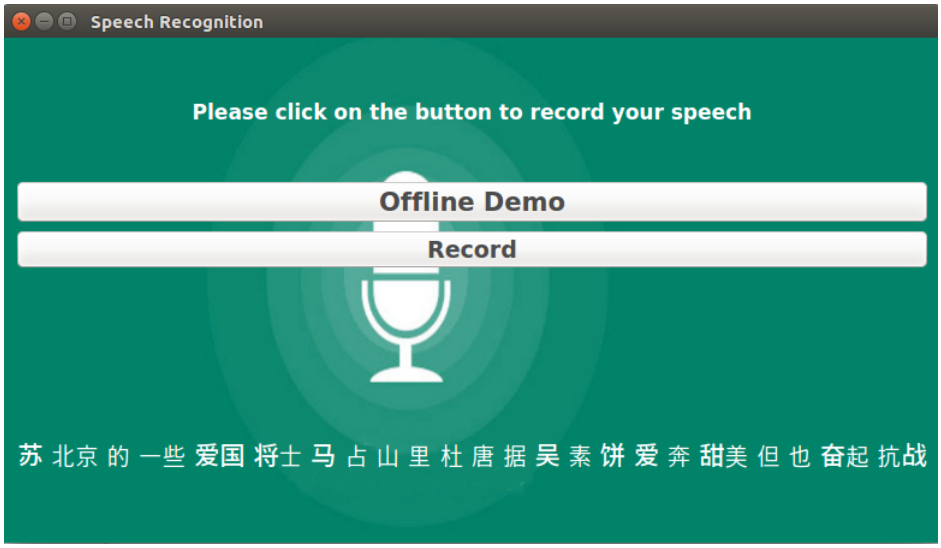

图 9: 离线语音识别结果

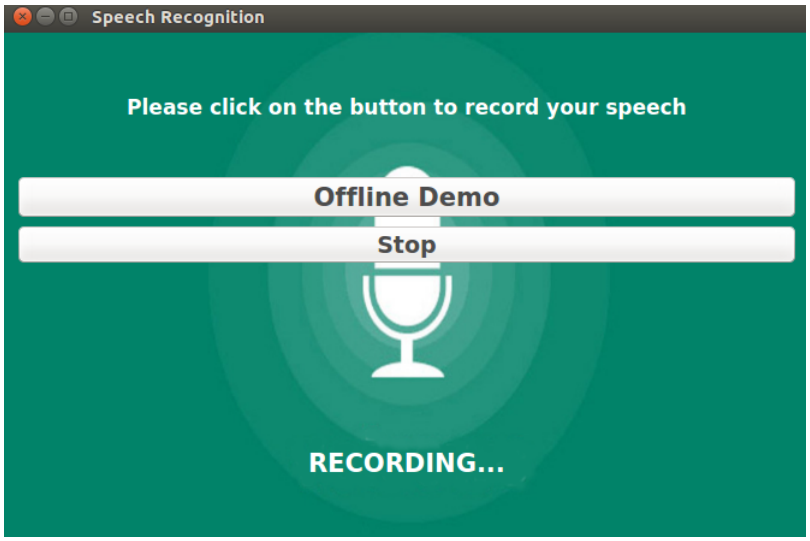

图 10: 在线语音识别：录音

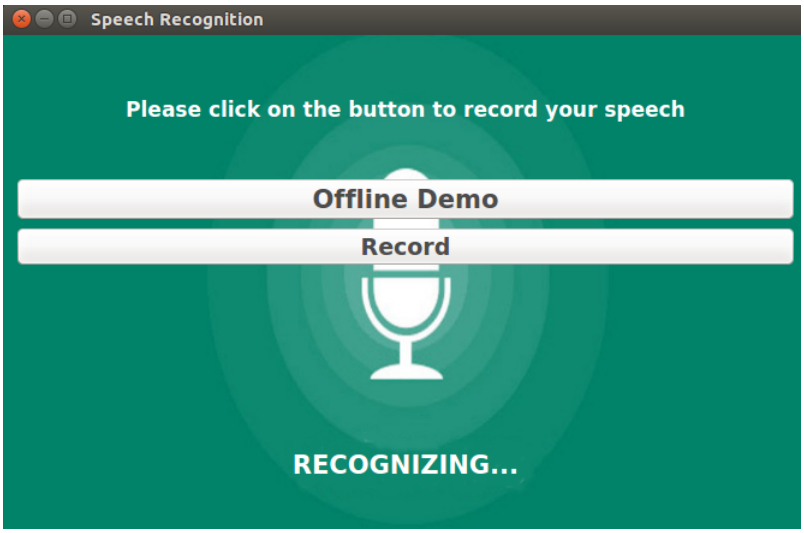

图 11: 在线语音识别：结束录音并识别

图 12: 在线语音识别: 输出结果