

哈尔滨工业大学

实验报告

实 验（三）

题 目 命令词识别实验报告

专 业 _____

学 号 _____

班 级 _____

学 生 _____

指 导 教 师 _____

实 验 地 点 _____

实 验 日 期 _____

计算机科学与技术学院

一、设计命令词识别任务

1.1 描述所设计的命令词识别任务

基于命令词识别的暗语式自动报警系统：

在人们的日常生活当中，独居或独行的人群可能潜在会受到歹徒的劫持和不法侵害。当危险情况发生时，报警是得到救助的重要途径，但贸然直接报警可能会激怒歹徒或受到歹徒的阻拦。因此，可以通过命令词识别技术，在歹徒不知情的情况下，假意配合歹徒，实则激活了报警系统，对现场进行录音，并将受害人位置发送给警方等。

实际应用场景设想：

歹徒：（人身控制并胁迫）

受害人：我一定配合，我保证不报警，你要什么我都给你（命令词）。

歹徒：把你的银行卡和密码告诉我。

受害人：这是我的卡，里面有 X 万，密码是 **325876**（命令词）。

歹徒：那支付密码是多少？

受害人：**742421**（命令词），这个不对那就是 123456（真密码）。

歹徒：你没骗我，钱我已经收到了。

受害人：**求求你了，放了我吧**（命令词）。

歹徒：……

在这个场景中，命令词应当保证基本不会在平时日常生活场景中出现，避免储出现报假警的情况。因此一串不存在的数字密码作为命令词稳健性会比较强。一个可能的判断规则是：5 个命令词出现 1 个时即进行现场录音，5 个命令词出现 3 个时确定不法侵害发生，触发报警，向警方发送受害人定位和当前场景录音，供警方判断现场情况。

该报警系统仅为被保护人服务，被保护人会事先录制好模板、确定好命令词并记忆，是一个特定人命令词识别任务。

1.2 列出词表

- ① 求求你了
- ② 放了我吧
- ③ 要什么我都给你
- ④ 325876
- ⑤ 742421

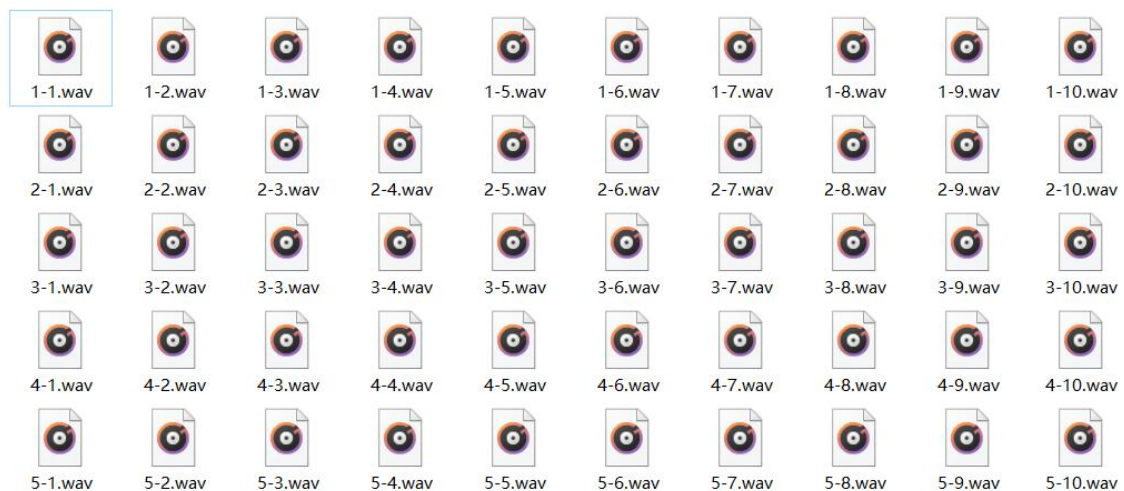
1.3 介绍语料采集方法和规模

采集方法：

利用 CoolEdit 进行语料采集，存储为 wav 格式。

规模：

每个命令词采集 10 遍，共 50 个语音。每份语料在 2s 左右。



二、特征提取

2.1 详细描述你所采用的特征和提取算法

对语音信号提取 mfcc 特征，特征提取算法和工具如下：

(1) mfcc 特征提取的流程和算法

① 预加重

预加重处理其实是将语音信号通过一个高通滤波器： $H(z) = 1 - \alpha z^{-1}$ 。

式中的值介于 0.9-1.0 之间，我们通常取 0.97。预加重的目的是提升高频部分，使信号的频谱变得平坦，保持在低频到高频的整个频带中，能用同样的信噪比求频谱。同时，也是为了消除发生过程中声带和嘴唇的效应，来补偿语音信号受到发音系统所抑制的高频部分，也为了突出高频的共振峰。

② 分帧

先将 N 个采样点集成一个观测单位，称为帧。通常情况下 N 的值为 256 或 512，涵盖的时间约为 20~30ms 左右。为了避免相邻两帧的变化过大，因此会让两相邻帧之间有一段重叠区域，此重叠区域包含了 M 个取样点，通常 M 的值约为 N 的 1/2 或 1/3。通常语音识别所采用语音信号的采样频率为 8KHz 或 16KHz，以 8KHz 来说，若帧长度为 256 个采样点，则对应的时间长度是 $256/8\text{KHz} = 32\text{ms}$ 。

③ 加窗（Hamming Window）

将每一帧乘以汉明窗，以增加帧左端和右端的连续性。假设分帧后的信号为 $S(n)$, $n=0,1,\dots,N-1$, N 为帧的大小，那么乘上汉明窗后为 $S'(n) = S(n) \times W(n)$ ，其中 $W(n)$ 形式如下：

$$W(n,a) = (1-a) - a \times \cos \frac{2\pi n}{N-1}, \quad 0 \leq n \leq N-1$$

不同的 a 值会产生不同的汉明窗，一般情况下 a 取 0.46。

④ 快速傅里叶变换

由于信号在时域上的变换通常很难看出信号的特性，所以通常将它转换为频域上的能量分布来观察，不同的能量分布，就能代表不同语音的特性。所以在乘上汉明窗后，每帧还必须再经过快速傅里叶变换以得到在频谱上的能量分布。对分帧加窗后的各帧信号进行快速傅里叶变换得到各帧的频谱。并对语音信号的频谱取模平方得到语音信号的功率谱。设语音信号的 DFT 为：

$$x_a(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}, \quad 0 \leq k \leq N$$

式中 $x(n)$ 为输入的语音信号， N 表示傅里叶变换的点数。

⑤ 三角带通滤波器

将能量谱通过一组 Mel 尺度的三角形滤波器组，定义一个有 M 个滤波器的滤波器组（滤波器的个数和临界带的个数相近），采用的滤波器为三角滤波器，中心频率为 $f(m)$ 。通常取 22-26。各 $f(m)$ 之间的间隔随着 m 值的减小而缩小，随着 m 值的增大而增宽。

三角滤波器的频率响应定义为：

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{2(k-f(m-1))}{(f(m+1)-f(m-1))(f(m)-f(m-1))} & f(m-1) \leq k \leq f(m) \\ \frac{2(f(m+1)-k)}{(f(m+1)-f(m-1))(f(m)-f(m-1))} & f(m) \leq k \leq f(m+1) \\ 0 & k \geq f(m+1) \end{cases}$$

$$\sum_{m=0}^{M-1} H_m(k) = 1$$

三角带通滤波器有两个主要目的：

对频谱进行平滑化，并消除谐波的作用，突显原先语音的共振峰。因此一段语音的音调或音高，是不会呈现在 MFCC 参数内，换句话说，以 MFCC 为特征的语音辨识系统，并不会受到输入语音的音调不同而有所影响。

此外，还可以降低运算量。

⑥ 计算每个滤波器组输出的对数能量为

$$s(m) = \ln\left(\sum_{k=0}^{N-1} |X_a(k)|^2 H_m(k)\right), \quad 0 \leq m \leq M$$

⑦ 经离散余弦变换（DCT）得到 MFCC 系数

$$C(n) = \sum_{m=0}^{M-1} s(m) \cos \frac{\pi n(m-0.5)}{M}, \quad n=1, 2, \dots, L$$

将对数能量带入离散余弦变换，求出 L 阶的 Mel-scale Cepstrum 参数。 L 阶指 MFCC 系数阶数，通常取 12-16。这里 M 是三角滤波器个数。

⑧ 对数能量

此外，一帧的音量（即能量），也是语音的重要特征，而且非常容易计算。因此，通常再加上一帧的对数能量（定义：一帧内信号的平方和，再取以 10 为底的对数值，再乘以 10）使得每一帧基本的语音特征就多了一维，包括一个对数能量和剩下的倒频谱参数。

注：若要加入其它语音特征以测试识别率，也可以在此阶段加入，这些常用的其它语音特征包含音高、过零率以及共振峰等。

⑨ 动态查分参数的提取（包括一阶差分和二阶差分）

标准的倒谱参数 MFCC 只反映了语音参数的静态特性，语音的动态特性可以用这些静态特征的差分谱来描述。实验证明：把动、静态特征结合起来才能有效提高系统的识别性能。差分参数的计算可以采用下面的公式：

$$d_t = \begin{cases} C_{t+1} - C_t & t > K \\ \frac{\sum_{k=1}^K k(C_{t+k} - C_{t-k})}{\sqrt{2 \sum_{k=1}^K k^2}} & \text{其他} \\ C_t - C_{t-1} & t \geq Q - K \end{cases}$$

式中， d_t 表示第 t 个一阶差分； C_t 表示第 t 个倒谱系数； Q 表示倒谱系数的阶数； K 表示一阶导数的时间差，可取 1 或 2。将上式中结果再代入就可以得到二阶差分的参数。

总结：因此，MFCC 的全部组成其实是由 N 维 MFCC 参数（ $N/3$ MFCC 系数 + $N/3$ 一阶差分参数 + $N/3$ 二阶差分参数）和帧能量（此项可根据需求替换）组成。

由此可知，39 维的 MFCC 参数即由 13 维 MFCC 系数、13 维一阶差分参数和 13 维二阶差分参数组成。

(2) mfcc 特征提取的方法

用 python 配置路径文件：

将待提取 mfcc 特征的 wav 文件路径和提取后的 mfc 文件路径存入 list.scf.

```
def main(train_path, wave_path, mfcc_path):
    filepath = train_path + '/' + wave_path
    savepath = train_path + '/' + mfcc_path
    folder = os.path.exists(savepath)
    if not folder:
        os.makedirs(savepath)
    filenames = os.listdir(filepath)
    with open(train_path + '/' + 'list.scf', 'w') as f:
        for filename in filenames:
            f.write(str(filepath) + '/' + str(filename) + ' '
                    + str(savepath) + '/' + str(filename[:-4]) + '.mfc' + '\n')
    print('成功保存:list.scf')
    f.close()
```

路径文件结果如下：

视听觉信号处理实验报告

```
list.scf - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-1.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-1.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-10.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-10.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-2.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-2.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-3.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-3.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-4.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-4.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-5.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-5.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-6.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-6.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-7.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-7.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-8.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-8.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/1-9.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/1-9.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-1.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-1.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-10.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-10.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-2.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-2.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-3.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-3.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-4.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-4.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-5.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-5.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-6.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-6.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-7.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-7.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-8.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-8.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/2-9.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/2-9.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-1.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-1.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-10.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-10.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-2.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-2.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-3.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-3.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-4.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-4.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-5.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-5.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-6.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-6.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-7.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-7.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-8.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-8.mfc
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/3-9.wav C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/3-9.mfc
第 19 行, 第 87 列 100% Windows (CRLF) ANSI
```

HTK 工具包中的 hcopy 工具提取 mfcc 特征:

配置好 list_v1.scf 文件后, 运行 “hcopy -A -D -T 1 -C tr_wav.cfg -S list.scf” 命令即可。特征提取参数 tr_wav.cfg 如下:

```
#[MODULE]    PARAMETER    = VALUE

SOURCEKIND    = WAVEFORM
SOURCEFORMAT  = WAV
ZMEANSOURCE   = F        #

TARGETKIND    = MFCC_F_D_A_Z
TARGETRATE    = 100000.0  # frame period = 10msec

SAVECOMPRESSED = T
SAVWITHCRC    = T

WINDOWSIZE    = 250000.0  # window size = 25msec
USEHAMMING    = T
PREEMCOEF     = 0.97      # 1st order preemphasis, coefficient = 0.97

NUMCHANS      = 26        # num. of filterbank channel = 26
CEPLIFTER     = 22        # num. of cepstra = 22
NUMCEPS       = 12        # num. of MFCC coefficient = 12

ENORMALIZE    = T        # energy normalization (live: F, otherwise: T)
ALLOWXMRDEXP  = T        # Needed for cross word systems
FORCECXTXP    = T        # Needed for cross word systems

#HSHELL Parameters
HSHELL: TRACE = 0002      # cotal

#HPARM Parameters
HPARM:  TRACE = 0101

#HLABEL Parameters
HLABEL: TRACE = 0010

#HNET Parameters
HNET:   TRACE = 0001      # Needed for recognition

#HREC Parameters
HREC:   FORCEOUT = T
```

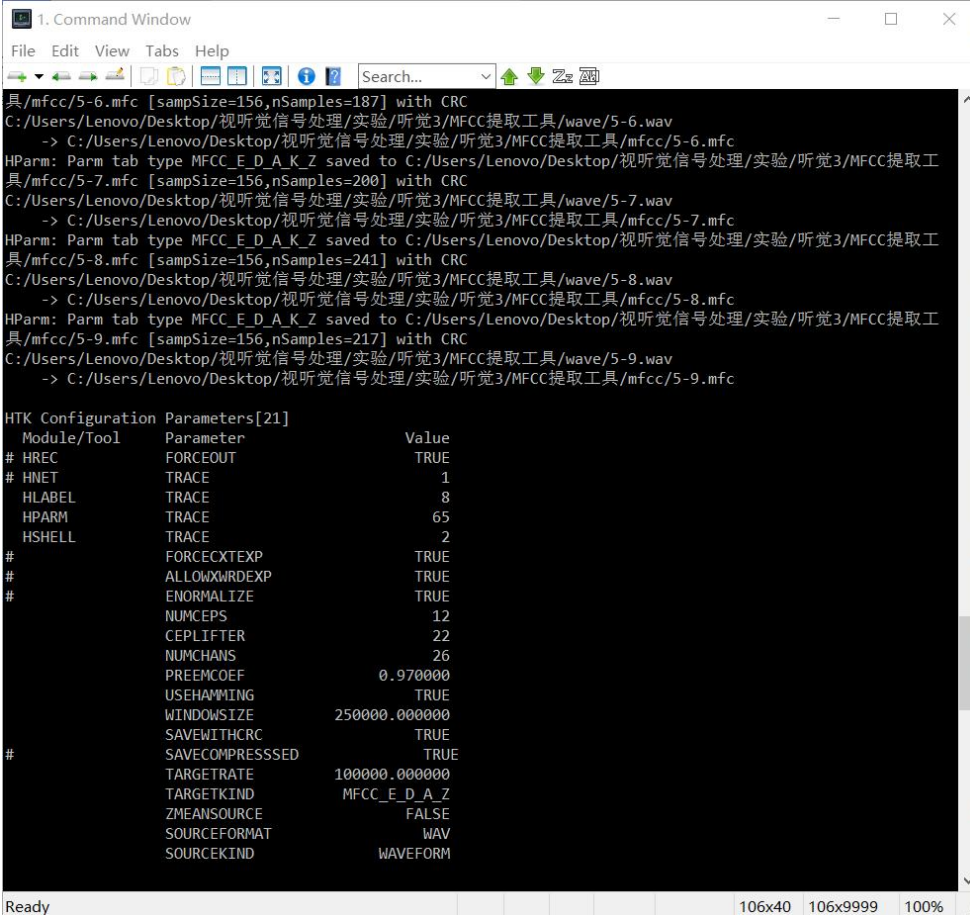

用 python 读取 mfc 文件:

通过 Python 读取 mfc 文件, 并将特征矢量序列转换为 numpy.array 形式储存。

```
def read_mfc(filename):
    f = open(filename, 'rb')
    '''
    nframes: number of frames 采样点数
    frate: frame rate in 100 nano-seconds unit
    nbytes: number of bytes per feature value
    feakind: 9 is USER
    '''
    nframes, frate, nbytes, feakind = struct.unpack('>IIHH', f.read(12))
    ndim = nbytes // 4 # feature dimension(4 bytes per value) 39 维度

    mfcc = np.zeros((nframes, ndim))
    for i in range(nframes):
        for j in range(ndim):
            mf = f.read(4)
            c = struct.unpack('>f', mf)
            mfcc[i, j] = c[0]
    f.close()
    #print (mfcc)
    return mfcc
```

2.2 给出特征提取部分运行结果的截图



```
1. Command Window
File Edit View Tabs Help
具/mfcc/5-6.mfc [sampSize=156,nSamples=187] with CRC
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/5-6.wav
-> C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/5-6.mfc
HParam: Parm tab type MFCC_E_D_A_K_Z saved to C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/5-7.mfc [sampSize=156,nSamples=200] with CRC
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/5-7.wav
-> C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/5-7.mfc
HParam: Parm tab type MFCC_E_D_A_K_Z saved to C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/5-8.mfc [sampSize=156,nSamples=241] with CRC
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/5-8.wav
-> C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/5-8.mfc
HParam: Parm tab type MFCC_E_D_A_K_Z saved to C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/5-9.mfc [sampSize=156,nSamples=217] with CRC
C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/wave/5-9.wav
-> C:/Users/Lenovo/Desktop/视听觉信号处理/实验/听觉3/MFCC提取工具/mfcc/5-9.mfc

HTK Configuration Parameters[21]
Module/Tool Parameter Value
# HREC FORCEOUT TRUE
# HNET TRACE 1
# HLABEL TRACE 8
# HPARM TRACE 65
# HSHELL TRACE 2
# FORCECXTXP TRUE
# ALLOWXWRDEXP TRUE
# ENORMALIZE TRUE
# NUMCEPS 12
# CEPLIFTER 22
# NUMCHANS 26
# PREEMCOEF 0.970000
# USEHAMMING TRUE
# WINDOWSIZE 250000.000000
# SAVETHCRC TRUE
# SAVECOMPRESSED TRUE
# TARGETRATE 100000.000000
# TARGETKIND MFCC_E_D_A_Z
# ZMEANSOURCE FALSE
# SOURCEFORMAT WAV
# SOURCEKIND WAVEFORM

Ready 106x40 106x9999 100%
```


2.3 给出特征文件内容的截图（将其写入到文本文件后展示）

```
[ 2.53523827e+00 -4.39902782e+00 -1.16047897e+01 ... 1.24498248e+00
-9.16726828e-01 1.46178547e-02]
[ 2.57184052e+00 -4.83891249e+00 -8.67219639e+00 ... 1.06726599e+00
-4.01484519e-01 -9.66004282e-03]]
[[-2.02777843e+01 5.01955843e+00 -2.41582985e+01 ... -1.24291766e+00
-8.11919749e-01 8.86904448e-03]
[-2.21830559e+01 -3.01916599e-01 -3.00562210e+01 ... -1.61042833e+00
-1.13247895e+00 1.75701233e-03]
[-2.66199799e+01 7.22254097e-01 -2.18450165e+01 ... -9.93406653e-01
-8.73671949e-01 -8.10719095e-03]
...
[ 8.78985977e+00 -5.34077168e+00 -9.32533073e+00 ... -2.08094731e-01
-4.50051010e-01 -1.69655606e-02]
[ 7.34272623e+00 -3.77006960e+00 -9.62185097e+00 ... -3.99493068e-01
-1.80229515e-01 -7.86344800e-03]
[ 3.05386496e+00 -5.36164093e+00 -1.71461849e+01 ... -3.68080705e-01
3.64850014e-02 2.00271001e-03]]
[[-2.98539906e+01 -5.92559624e+00 -5.66962528e+00 ... -1.78047687e-01
-1.15380667e-01 7.07826298e-03]
[-2.61033459e+01 4.22648096e+00 -2.36575794e+00 ... 1.05556987e-01
-7.90966630e-01 -1.20849926e-02]
[-3.74821510e+01 6.98279190e+00 -7.63497972e+00 ... 3.88148457e-01
-1.39890563e+00 -3.28706801e-02]
```

三、 基于 DTW 的命令词识别

3.1 介绍你所设计 DTW 算法，标明所采用的开发工具

开发环境：

Windows10; python 3.9.12; Visual Studio Code; Anaconda.

(1) DTW 算法：

由于即使同一个人不同时间发出同一个声音，也不可能具有相同的长度，因此就需要用到动态时间归正（DTW）算法。把时间归正和距离测度计算结合起来的一种非线性归正技术。

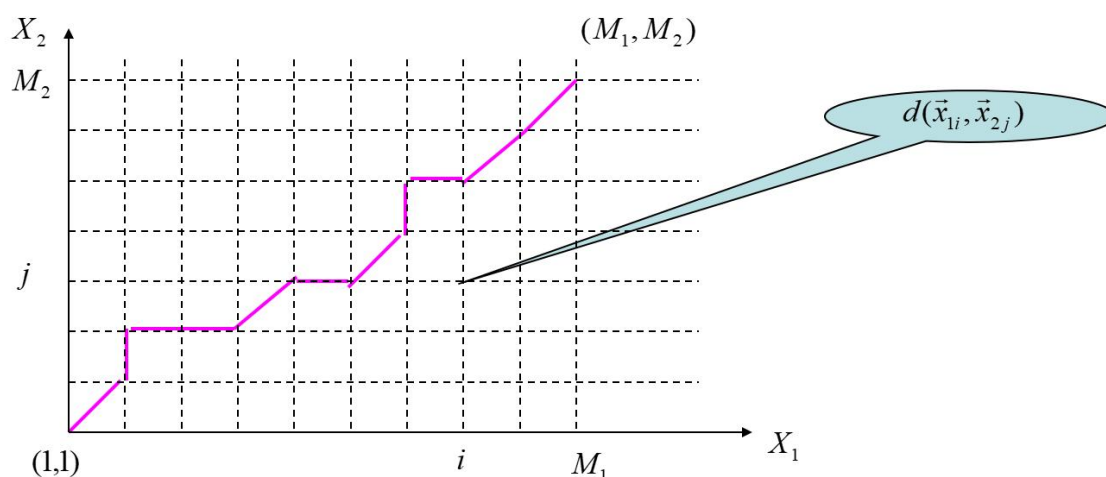
DTW 本质上是一个简单的动态规划算法，是用来计算两个维数不同的向量之间的相似度的问题，即计算向量 $M1$ 和 $M2$ 的最短距离。是一种非常常用的语音匹配算法，算法如下。

算法思想：

对两个不同维数的语音向量 $m1$ 和 $m2$ 进行匹配（ $m1$ 和 $m2$ 的每一维也是一个向量，是语音每一帧的特征值，这里利用的是 MFCC 特征）。设两个向量的长度为 $M1$ 和 $M2$ ，则距离可以表示为：

$$D(X_1, X_2) = \sum_{i=1}^M d(\vec{x}_{1i}, \vec{x}_{2i})$$

那么，就可以这样进行匹配：

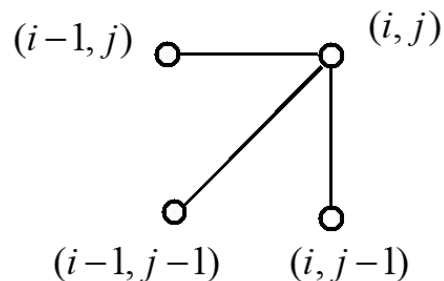


1. 每一条从 $(1, 1)$ 到 (M_1, M_2) 路径都有一个累计距离称为路径的代价。
2. 每一条路径都代表一种对齐情况
3. 代价最小的路径就是所求的对准路径。

这样就可以将对准问题，或者说将求两个语音段的相似度问题，转化成了搜索代

价最小的最优路径问题。

在搜索过程中，往往要进行路径的限制：



在此限制条件下，可以将全局最优化问题转化为许多局部最优化问题一步一步地来求解，这就动态规划(Dynamic Programming，简称 DP)的思想。

1. 定义一个代价函数 $\Phi(i, j)$ ，表示从起始点(1,1)出发，到达 (i, j) 点最小代价路径的累计距离。有：

$$\Phi(i, j) = \min_{(i', j') \rightarrow (i, j)} \{ \Phi(i', j') + d(\vec{x}_{1i}, \vec{x}_{2j}) w_n \}$$

2. 这样的话，要计算两个向量之间的最短距离，可以表示为：

$$\begin{aligned} \Phi(M_1, M_2) = \min \{ & \Phi(M_1 - 1, M_2) + d(\vec{x}_{1M_1}, \vec{x}_{2M_2}) w_n, \\ & \Phi(M_1, M_2 - 1) + d(\vec{x}_{1M_1}, \vec{x}_{2M_2}) w_n, \\ & \Phi(M_1 - 1, M_2 - 1) + d(\vec{x}_{1M_1}, \vec{x}_{2M_2}) w_n \} \end{aligned}$$

依次类推，可以得到：

$$\Phi(M_1 - 1, M_2), \Phi(M_1, M_2 - 1), \Phi(M_1 - 1, M_2 - 2)$$

可由更低一层的代价函数计算得到

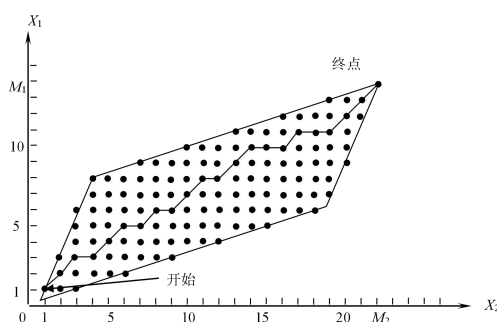
3. 这样就可以从 $\Phi(1,1)$ 开始计算，定义加权系数，加权系数的取值与局部路径有关：

$$w_n = \begin{cases} 2 & (i-1, j-1) \rightarrow (i, j) \\ 1 & \text{其它} \end{cases}$$

4. 定义回溯函数：

$$P(i, j)$$

5. 如图，为了减少计算量，可以定义平行四边形区域约束：



算法步骤:

1. 初始化:

$$i = j = 1, \Phi(1, 1) = d(\vec{x}_{11}, \vec{x}_{21})$$

$$\Phi(i, j) = \begin{cases} 0 & \text{当}(i, j) \in \text{Reg} \\ \text{huge} & \text{当}(i, j) \notin \text{Reg} \end{cases}$$

其中约束区域 Reg 可以假定是这样一个平行四边形, 它有两个顶点位于(1,1)和 (M_1, M_2) , 相邻两条边的斜率分别为 2 和 1/2。在本实验中, 计算规模不大, 没有平行四边形约束。

2. 递推求累计距离 并记录回溯信息:

$$\Phi(i, j) = \min \{ \Phi(i-1, j) + d(\vec{x}_{1i}, \vec{x}_{2j}) \cdot W_n(1); \Phi(i-1, j-1) + d(\vec{x}_{1i}, \vec{x}_{2j}) \cdot W_n(2);$$

$$\Phi(i, j-1) + d(\vec{x}_{1i}, \vec{x}_{2j}) \cdot W_n(3) \}$$

$$i = 2, 3, \dots, M_1; j = 2, 3, \dots, M_2; (i, j) \in \text{Reg}$$

一般取距离加权值为:

$$W_n(1) = W_n(3) = 1 \quad W_n(2) = 2$$

3. 计算出的值就是 m1 和 m2 之间的距离。

代码如下:

```
def dtw(mfc_data1, mfc_data2):
    len1 = mfc_data1.shape[0]
    len2 = mfc_data2.shape[0]
    # 记录代价矩阵
    Cost = np.zeros((len1 + 1, len2 + 1))
    # 记录路径
    Path = np.zeros((len1 + 1, len2 + 1, 2))

    # 左下增加一圈无穷大的数, 达到限制边界的目的
    for i in range(1, len1 + 1):
        Cost[i][0] = sys.maxsize
    for j in range(1, len2 + 1):
        Cost[0][j] = sys.maxsize
```

```

for i in range(1, len1 + 1):
    for j in range(1, len2 + 1):
        # 两 39 维向量的欧氏距离
        d = np.linalg.norm(mfc_data1[i-1] - mfc_data2[j-1])
        a1 = 2 * d + Cost[i-1, j-1] if (i != 1 or j != 1) else d
        a2 = d + Cost[i-1, j]
        a3 = d + Cost[i, j-1]
        minFai = min(a1, a2, a3)
        Cost[i, j] = minFai
        if minFai == a1:
            Path[i, j] = [i-1, j-1]
        elif minFai == a2:
            Path[i, j] = [i-1, j]
        else:
            Path[i, j] = [i, j-1]
# 返回归一化最终 Cost
return Cost[len1][len2]/(len1 + len2 - 2), Path

```

(2) 模板提取方法

模板提取方法：首先选取语音的平均长度，然后选取距离平均长度最近的语音作为临时模板，把其他语音向这个临时模板进行规整后，进行平均。代码如下：

```

def main(template_path, template_num, repeat_num):
    filename = os.listdir(template_path)
    # 存储得到的接近平均长度的平均 mfc_array(即返回值)
    final_template = []

    for i in range(template_num):
        # 存储同一命令词的 10 个 mfc-array
        mfc_of_1template = []
        # 存储同一命令词的 10 个 mfc-array 的长度
        len_array = np.zeros(repeat_num, dtype=int)
        for j in range(repeat_num):
            total_filename = template_path + '/' + filename[i*repeat_num+j]
            mfc_of_1template.append(AT.read_mfc(total_filename))
            len_array[j] = mfc_of_1template[j].shape[0]

        ave_len = np.mean(len_array)
        # 选取距离平均长度最近的语音作为临时模板
        ave_index = np.argmin(np.abs(len_array - ave_len))
        # 存储同一命令词的 10 个 mfc-array 映射到相同长度 ave_len 后，存入 same_len_mfc
        same_len_mfc = np.zeros((repeat_num, mfc_of_1template[ave_index].shape[0],
                                   mfc_of_1template[ave_index].shape[1]))
        for j in range(repeat_num):
            if j == ave_index:
                same_len_mfc[j] = np.array(mfc_of_1template[j])
            else:
                cost, path = AT.dtw(mfc_of_1template[ave_index], mfc_of_1template[j])
                same_len_mfc[j] = AT.get_same_len(mfc_of_1template[j], path)

        final_template.append(np.mean(same_len_mfc, axis=0))

    print('平均模板成功生成! ')
    return final_template

```

其中，计算平均模板的过程需要对路径进行回溯，对原有语音文件局部进行压缩或者放大，规整成相同长度。

```
def get_same_len(mfc_array, Path):
    len1 = Path.shape[0] - 1
    len2 = Path.shape[1] - 1
    new_array = np.zeros((len1, mfc_array.shape[1]))
    Path[0, 0] = np.array([-1, -1])

    m = np.array([len1, len2])
    for i in range(len1 - 1, -1, -1):
        count = 0
        sum_array = np.zeros(mfc_array.shape[1])
        while True:
            sum_array = sum_array + mfc_array[int(m[1]-1)]
            count += 1
            pm = Path[int(m[0]), int(m[1])]
            if pm[1] != i:
                break
            m = pm
        new_array[i] = sum_array / count
        m = pm
    return new_array
```

使用该模板选取方法，对标准模板进行平均后，每个单元音的识别率均有不同程度的提高，说明将模板平均化之后模板内部的距离变小了，从图上我们也可以看出模板间的距离也是减小的，但没有模板间距离减小的程度大，所以提高了识别率。

参考文献：W. H. Abdulla, D. Chow and G. Sin, "Cross-words reference template for DTW-based speech recognition systems," TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region, 2003, pp. 1576-1579 Vol.4, doi: 10.1109/TENCON.2003.1273186.

3.2 正确率

在测试集中，每个命令词有 3 段录音，其中还有 6 段录音是非命令词。共 21 个录音特征文件。

不包括非命令词：

正确检出的语料文件的个数：15

正确率：100%

包括非命令词：

正确检出的语料文件的个数：15

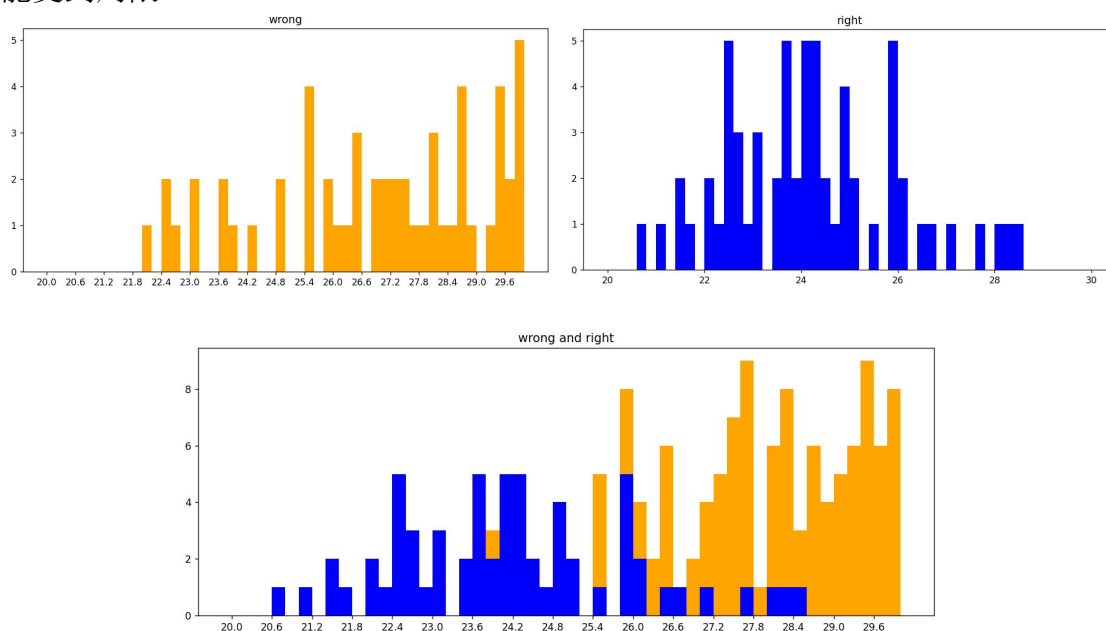
正确率：71.43%

从中可以看出，错误率主要来自对非命令词的识别不应当归入命令词当中，可以进一步改善。

四、 扩展内容

4.1 介绍你所设想出的方法及其算法实现

之前的算法没有考虑到非命令词的排除，因此对 dtw 算法得到的 cost 值设置一个阈值，如果模板匹配后最小的 cost 值还是大于阈值，则说明待识别语音不属于任何一个命令词。为了设置合理的阈值，分别观察正确识别和错误识别对应的的 cost 值的大小。不过由于训练集和测试集规模都很小，观察的阈值泛化能力可能受到局限。



从中可以看出，当阈值 threshold 取 27 左右时，基本能保证非命令词不被误分类，命令词能够正确分类。有阈值的匹配代码如下，不属于任何类别时返回 0：

```
def match(template, test_mfc_filename, threshold):
    test_mfc_array = AT.read_mfc(test_mfc_filename)
    cost_lst = np.zeros(len(template))

    for i in range(len(template)):
        cost, path = AT.dtw(template[i], test_mfc_array)
        cost_lst[i] = cost

    if np.min(cost_lst) <= threshold:
        return np.argmin(cost_lst) + 1, cost_lst
    else:
        return 0, cost_lst
```


4.2 正确率

包括非命令词：

正确检出的语料文件的个数：20

正确率：95.24%

其中，唯一的错误标签，是把某非命令词识别成了命令词。这可能与阈值的取值变化有关。

五、 总结

5.1 请总结本次实验的收获

在本次实验中，我使用 python 软件和 hcopy 工具，体会了命令词识别的 DTW 算法和 mfcc 特征提取，理解了序列对齐的思想。同时，在实验中通过观测 cost 值特征并建立数学模型，自己对阈值进行尝试，拓宽了思路，增强了理论运用能力和理解的深度，收获很大。

5.2 请给出对本次实验内容的建议

本实验难度适中，设计任务合理，自定义使用场景和录制语料很有意思，感谢老师的实验设计和答疑解惑。