哈爾濱工業大學

实验报告

实 验（三）

题　　目 命令词识别实验报告

专　　业 人工智能（视听觉信息处理）

学　　号 7203610121

班　　级 20W0362

学　　生 刘天瑞

指 导 教 师 郑铁然

实 验 地 点 线上

实 验 日 期 2022.12.11

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

1. 录制 5 组命令词，每组 10 个，所以一共有 50 个.wav 文件。存储在“语料”文件夹中；
2. 首先利用实验一的代码对录制语音进行端点检测，然后去除 50 个音频的静音部分。替换原始的音频文件；
3. 利用老师提供的 HTK 中 HCopy 工具来提取出.wav 文件的 MFCC 特征，并将其存放在.mfc 文件中，然后同样存储在“语料”文件夹中；
4. 编写代码提取出 MFCC 特征值，其中 MFCC 为帧数×维度的矩阵，共有 50 个 MFCC 矩阵，从而构成一个 5×10 的矩阵；
5. 对 50 个 MFCC 分别使用 DTW 模型和 HMM 模型进行匹配。每组首个取出作为训练模板，剩余 45 个语音命令作为测试样例来测试 DTW 算法，即共有 5 个训练样本，45 个测试样本；HMM 模型取每个音频的前 3 个为训练模板，后 7 个为测试样例，即共 15 个训练样本，35 个测试样本；
6. 最后分别对上述两种模型的测试样本进行匹配，再输出每个测试样例用类别标签表示的识别结果，计算两种模型的正确率。

1.2 列出词表

1.规格严格；
2.功夫到家；
3.人才辈出；
4.八百壮士；
5.人工智能。

1.3 介绍语料采集方法和规模

采集方法：
利用 Cool Edit Pro 2.1 进行语料采集录制。
规模：
5 个命令词，每个录制 10 遍，共采集 50 段语音。每个时长大约 3 s。

二、特征提取

2.1详细描述你所采用的特征和提取算法

1.采用的特征：

利用 HCopy，设置参数：帧长 25ms，帧移 10ms。计算出 39 维的 MFCC 特征。每个特征的值为 4Bytes，存放在帧数 nframes*特征数 ndim 的数组中。39 维特征包括 13 维 MFCC 的 dct 系数，13 维的一阶差分以及 13 维的二阶差分。

2.提取算法：

利用 HTK 中的 HCopy 进行提取，存储在.mfc 文件中。提取算法是首先进行预加重，然后进行分帧，加窗，然后进行快速傅里叶变换，将它转换为频域上的能量分布来观察；将能量谱通过一组 Mel 尺度的三角形滤波器组，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；计算每个滤波器输出的对数能量，经离散余弦变换（DCT）得到 MFCC 系数；然后计算对数能量；最后提取动态差分参数（包括一阶差分和二阶差分）。

2.2 给出特征提取部分运行结果的截图

终端运行指令：.\HCopy.exe -A -D -T 1 -C tr_wav.cfg -S .\list.scp；

其中 wav.cfg 为配置文件名。list.scp 为存放 wav 文件和提取对应特征的文件名。

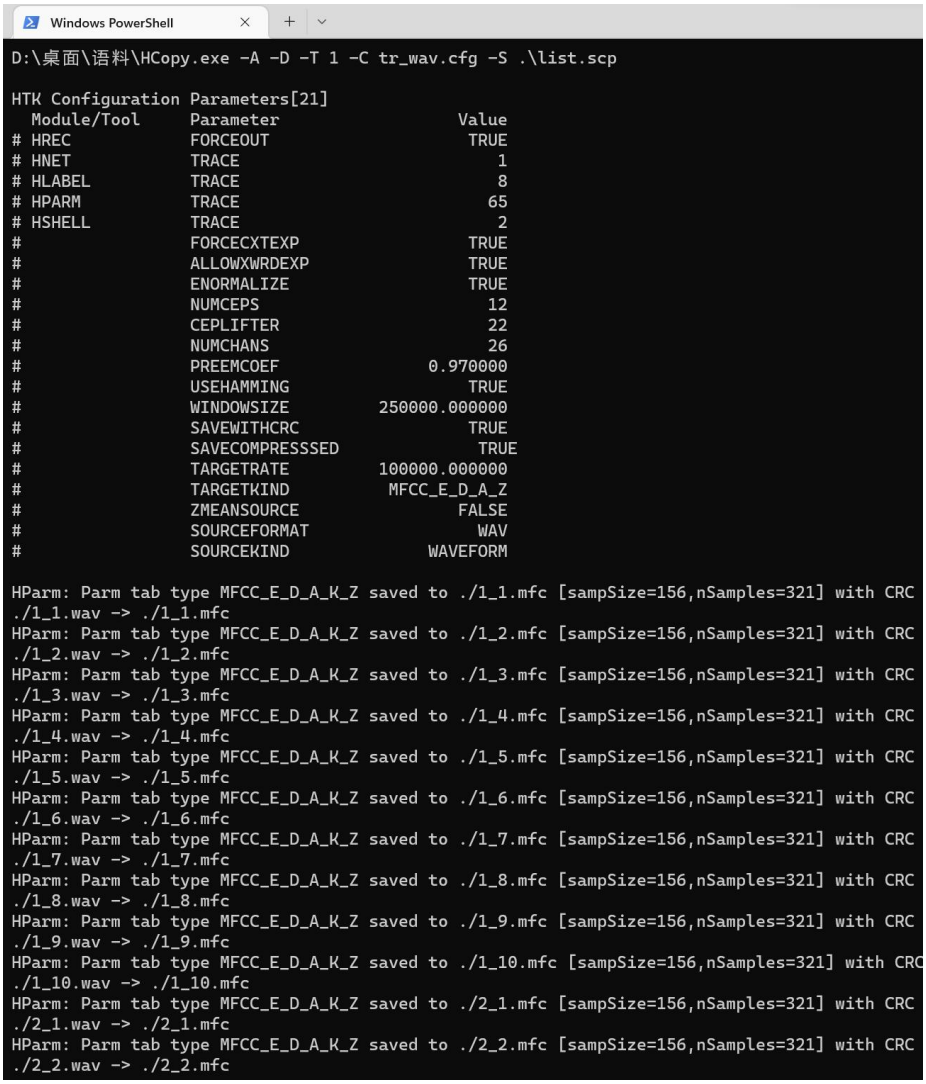

图 1 特征提取部分运行结果截图

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）

以 1_1.mfc 文件写入文本文件为例：

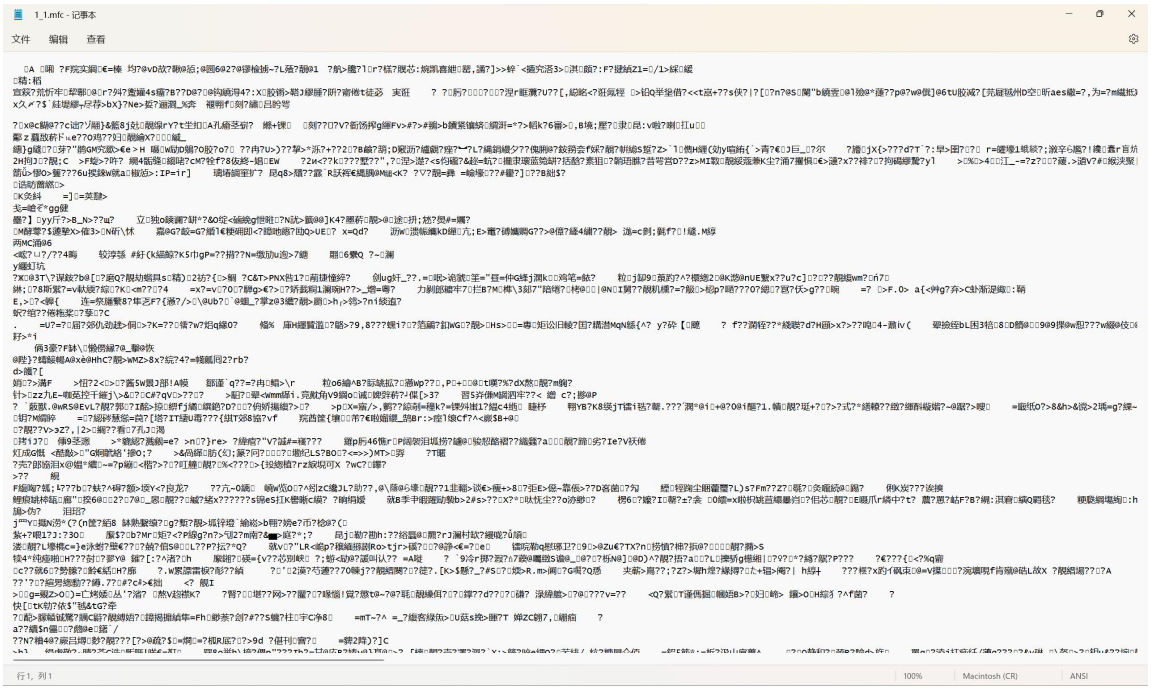

图 2 特征文件内容截图

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

因为即使同一个人不同时间发出同一个声音，也不可能完全具有相同的长度，因此就需要用到动态时间归正（DTW）算法。我认为 DTW 算法是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，也就是计算测试样本和训练样本哪个更相似，而相似程度则是通过计算两个 MFCC 特征距离来衡量的。同时 MFCC 特征是一个矩阵，计算特征距离可以通过计算每一帧的欧式距离之和来表示。除此之外每两个 MFCC 特征的帧数不一定完全相同，所以必定会有某些帧为空无法计算，从而引入常用的语音匹配算法——DTW 算法。

DTW 算法本质上是一个较为简单的动态规划算法。算法思想是对两个维数不同的语音矩阵 model 和 test 进行匹配。model 和 test 矩阵的每一维都是一个向量，是语音一帧的 MFCC 特征值。两个矩阵 model，test 的帧数分别设置为 M1，M2，则欧式距离可以表示为如下所示：

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

然后进行如下所示的匹配：

1. 每一条从坐标（1，1）到（M1，M2）的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；

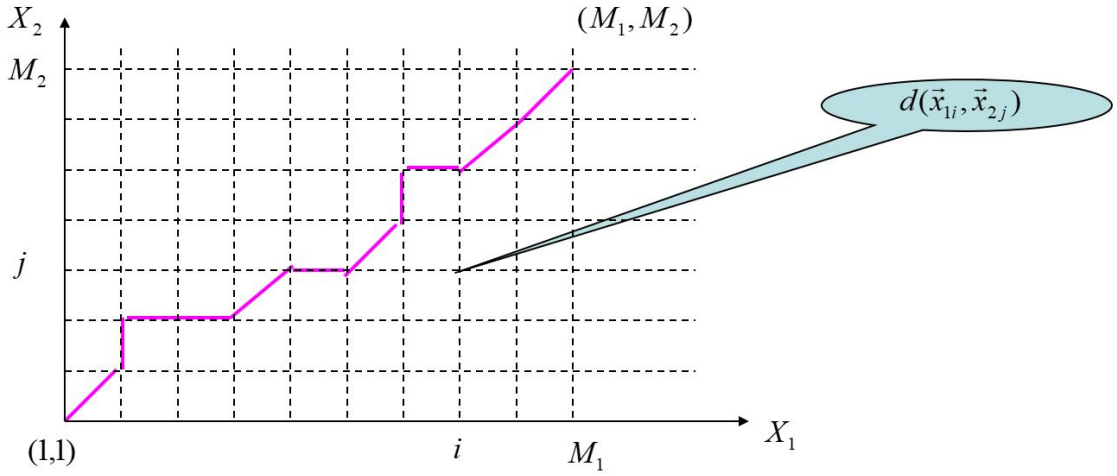

图 3 代价最小路径匹配对齐示意图

如此一来就可以将对准问题，即求两个语音段的相似度问题，转化为搜索代价最小的最优路径问题。而如何找到最小路径如下所示：

1.在搜索过程中首先往往需要对搜索路径进行如下限制：即（i，j）仅能通过（i-1，j）、（i-1，j-1）或者（i，j-1）到达下一坐标点。

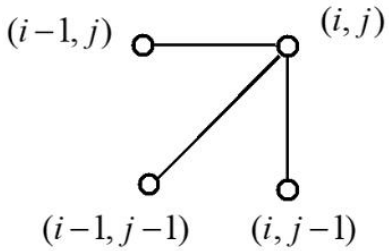

图 4 搜索路径限制示意图

2.在此限制条件下，可以将全局最优问题转换为各个局部最优问题（i，j），代价可以通过更低一层的代价系数相乘再累加一步步求解，即动态规划(Dynamic Programming，简称 DP)的思想。

3.定义一个代价函数为 $\Phi(i, j)$，其表示从起始点(1,1)出发，到达点（i，j）最小代价路径的累计距离。有如下式：

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

4.要计算两个向量之间的最短距离，可以表示为如下式：

$$\Phi(M_1,M_2)=\min\{\Phi(M_1-1,M_2)+d(\vec{x}_{1M1},\vec{x}_{2M2})w_n,$$
$$\Phi(M_1,M_2-1)+d(\vec{x}_{1M1},\vec{x}_{2M2})w_n,$$
$$\Phi(M_1-1,M_2-1)+d(\vec{x}_{1M1},\vec{x}_{2M2})w_n\}$$

依次类推可以得到如下：

$\Phi(M_1-1,M_2)$、$\Phi(M_1,M_2-1)$、$\Phi(M_1-1,M_2-2)$ 可由更低一层代价函数求解。

5.从 $\phi^{(1,1)}$ 开始计算，定义加权系数，加权系数取值与局部路径有关：

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

6.最后定义回溯函数来记录回溯信息，一般取距离加权值，以及为了减少计算量，定义约束区域。

我所采用的开发工具为：Windows 11；Visual Studio 2022；Python 3.9.7，调用的库为 numpy 和 matplotlib.pylab。

具体的运行代码以及必要且适当的注释说明如下图所示：

```
153  def DTW(x_1, x_2):
154      x_1 = np.array(x_1)
155      x_2 = np.array(x_2)
156      row = len(x_1)
157      col = len(x_2)
158      ndim = len(x_1[0])
159      D = np.zeros((row, col))
160      for i in range(row):# 初始化到点（i，j）的路径代价
161          for j in range(col):
162              D[i][j] = 0
163              D[i][j] = np.linalg.norm(x_1[i] - x_2[j])# 计算欧式距离
164              # for k in range(ndim):
165              #     D[i][j] += abs(x_1[i][k] - x_2[j][k])# 计算每一个差值的绝对值，最后求和
166      for i in range(1, row):# 计算每一列代价路径的累计距离
167          D[i][0] += D[i - 1][0]
168      for j in range(1, col):# 计算每一行代价路径的累计距离
169          D[0][j] += D[0][j - 1]
170      for i in range(1, row):# 开始搜索最短代价的路径
171          for j in range(1, col):
172              D[i][j] = min(D[i - 1][j] + D[i][j], D[i - 1][j - 1] + 2 * D[i][j], D[i][j - 1] + D[i][j])
173      return D[row - 1][col - 1] / (row + col - 2)# 计算结果用系数和来进行归正
174      # return D[row - 1][col - 1]
```

图 5 设计 DTW 算法具体运行代码

3.2 正确率：

当测试语料的类别标签与识别结果相同时，则说明该语料的识别结果是正确的。

刚开始向量距离使用欧式距离计算时正确检测出的语料文件个数只有 3 个，

正确率只有 6.67%，非常低，所以我又重新检查计算欧式距离的代码，发现某些欧式距离计算出来是负数而并没有取绝对值进行累加，并且我忘记了进行端点检测处理来消除静音，修改后正确率结果提升恢复至 100%，正确检测出的语料文件个数为 45 个。如下图所示：

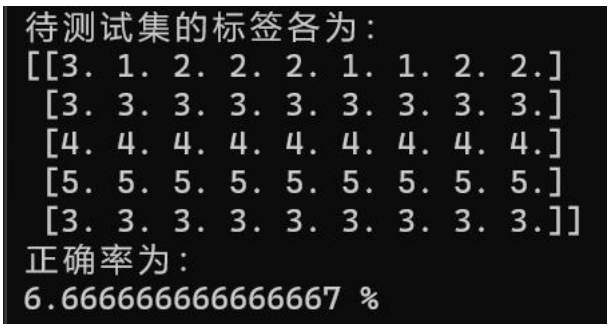

图 6 没取绝对值之前的正确率结果

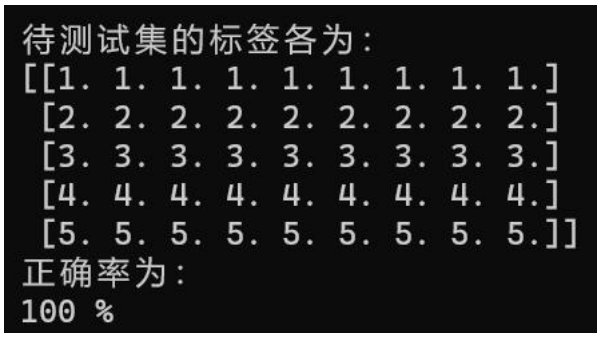

图 7 取绝对值修正之后的正确率结果

计算测试结果的具体运行代码如下所示：

```
def voice_Recognition(model, test):
    row = len(test)
    col = len(test[0])
    cnt = 0#  正确的语料文件个数
    flag_r = np.zeros((5, 9))#  待测试集的标签矩阵
    for i in range(row):
        flag_l = i + 1
        for j in range(col):
            print("正在处理第" + str(i * col + 1 + j) + "个音频")
            flag_r[i][j] = 1
            min_dis = DTW(test[i][j], model[0])
            for k in range(len(model)):
                dis = DTW(test[i][j], model[k])
                if dis < min_dis:
                    flag_r[i][j] = k + 1
                    min_dis = dis
            if flag_r[i][j] == flag_l:
                cnt += 1
    print("待测试集的标签各为：")
```

```
print(flag_r)
print("正确率为：")
      print(cnt / (row * col) * 100,"%")
```

四、 扩展内容

4.1 介绍你所设想出的方法及其算法实现

除了利用 DTW 算法，我还尝试设想开展语音实时录取，在代码中动态对语音进行实时录取并进行匹配。将经过实验验证的算法，转化为能实时采集，在线检测的命令词识别系统。实时录取了 2.5s（相比较之下更短）的语音。

4.2 正确率：

效果没有达到静态匹配的 100%准确率，但也有很高的准确率，对输入标准的命令词都可以识别出来。

五、 总结

5.1请总结本次实验的收获

答：1.本次实验使我对语音识别有了更为深刻的认识，同时也学会了如何使用 HCopy 工具进行提取每个语料音频的 MFCC 特征文件；2.我还对语音特征有了较为完整的了解，除此之外还掌握了 MFCC 特征的具体内容定义；3.最后我学会了用 Python 代码编写设计 DTW 算法，以及将其熟悉地灵活应用于语音识别领域。

5.2 请给出对本次实验内容的建议

答：我个人认为实验内容比较完备，非常满意，实验指导书也写的很明确，老师的实验指导也很到位，十分清晰明确，一点也不含糊。但是我才疏学浅，认为 HMM 算法设计实现较为困难，因此未能完成。总体还是进行得较为顺利，非常感谢！