

**实验报告**

**实 验（三）**

题 目 命令词识别实验报告

专 业 计算机科学与技术

学　　 号 1160300426

班　　 级 1603107

学 生 李国建

指 导 教 师 郑铁然

实 验 地 点 G709

实 验 日 期 2018.11.12

**计算机科学与技术学院**

# 一、 设计命令词识别任务

## 1.1 描述你所设计的命令词识别任务

一共录制了十组命令词，每组 5 个，一共 50 个。每组的第一个取出，作为模板。剩余的 40 个语音命令作为测试样例，测试 DTW 算法。

1. 首先录制 50 个语音；存储在 RecordedVoice 文件夹下
2. 然后利用实验一的代码对录制的语音进行端点检测；存储在 RecordedVoice-EndPointed 文件夹下
3. 对端点检测后的语音命令进行操作，利用 HTK 中的 HCopy 工具提取出 MFCC 特征，存放在 xxx.mfc 文件中，放在 MFCC-EndPointedVoice 文件夹下。
4. 在代码中取出所有语音命令的 MFCC 特征，区分开模板和测试样例。根据 DTW 算法，对 40 个测试样例进行静态语音匹配，检测正确率。
5. 在代码中录音，然后保存录音的文件，存放在 RecordedVoice-RealTime 中。利用端点检测的代码对其进行端点检测，保存在 RecordedVoice-RealTime 中。
6. 在代码中动态调用 HCopy 程序，对实时录的语音进行 MFCC 特征提取，然后进行语音匹配，检测是否与对应的命令匹配上了。

## 1.2 列出你的词表

* 1. 开门
  2. 关门
  3. 播放音乐
  4. 关闭音乐
  5. 打开空调
  6. 关闭空调
  7. 打开电视
  8. 关闭电视
  9. 开始扫地
  10. 停止扫地

## 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。

规模：

一共采集 50 个语音。每个大约 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

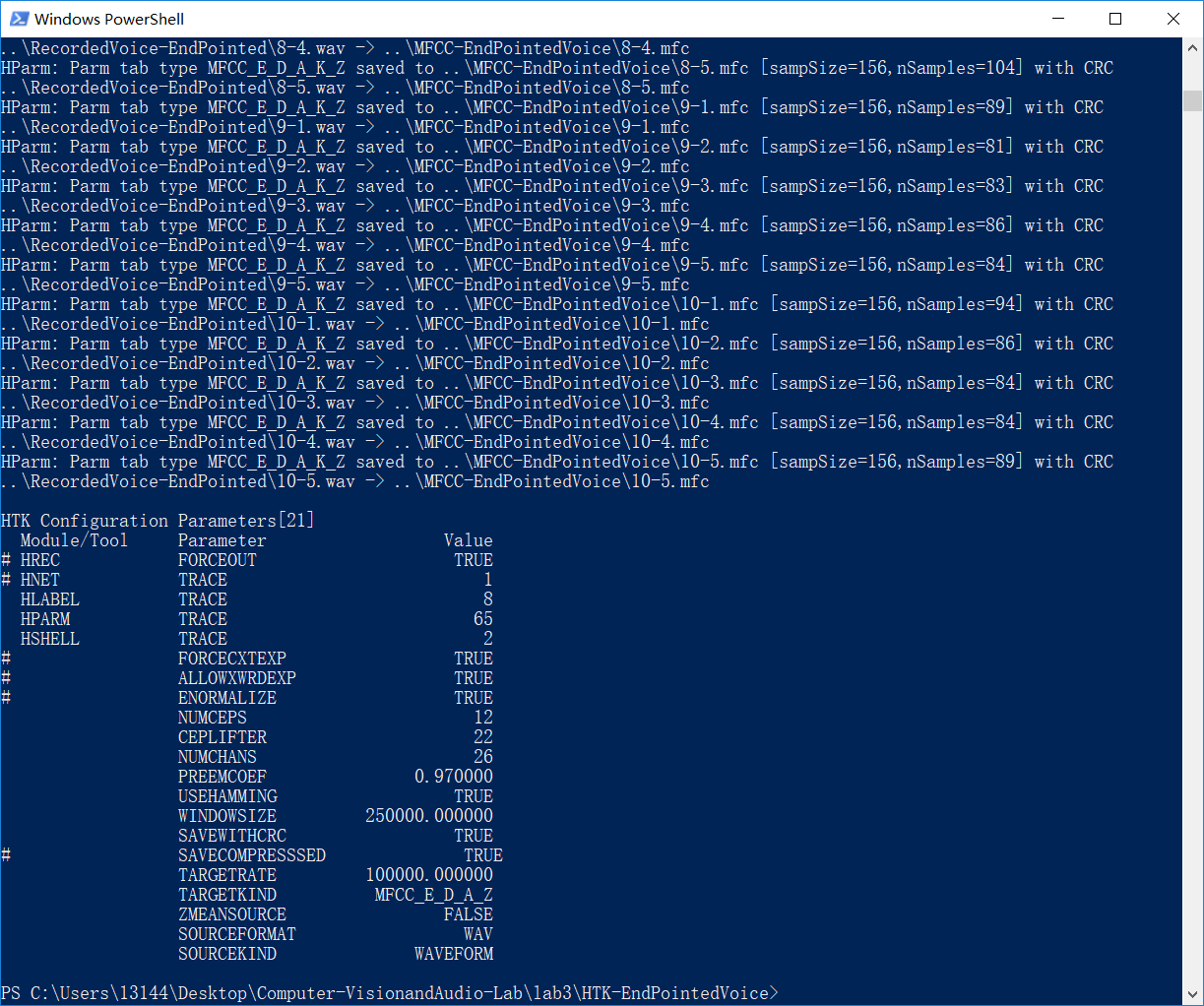
采用的特征：

采用的是利用 Hcopy 计算出的 39 维 MFCC 特征。包括12维 dct 系数，能量特征，一阶差分和二阶差分等。

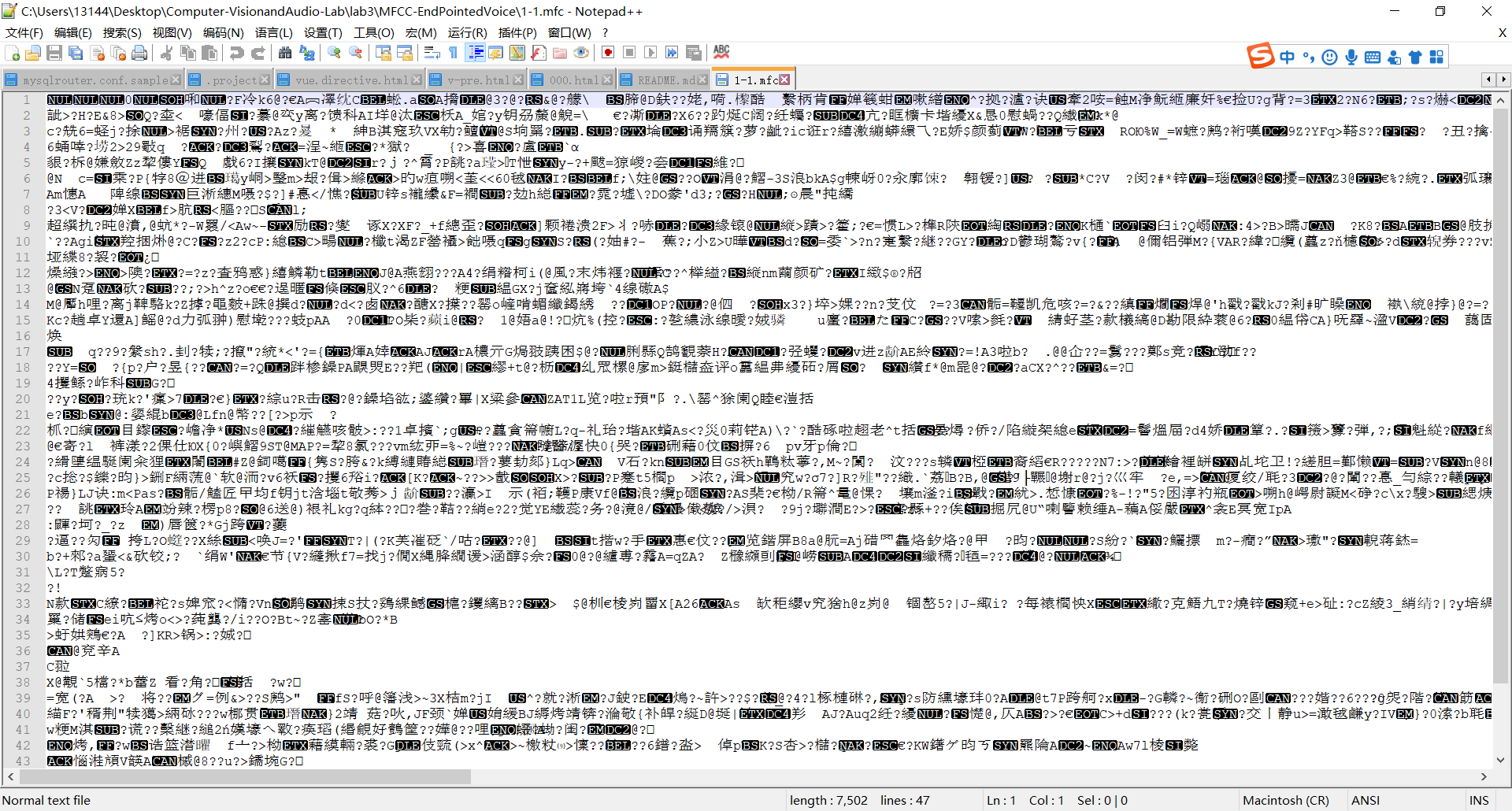
提取算法：

利用 HTK 中的 Hcopy 进行提取，存储在.mfc文件中。提取算法：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



2.3 给出特征文件内容的截图



# 三、 DTW计算

## 3.1 给出DTW算法，标明所采用的开发工具

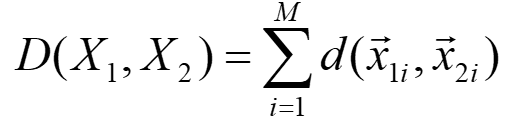
DTW算法：

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。

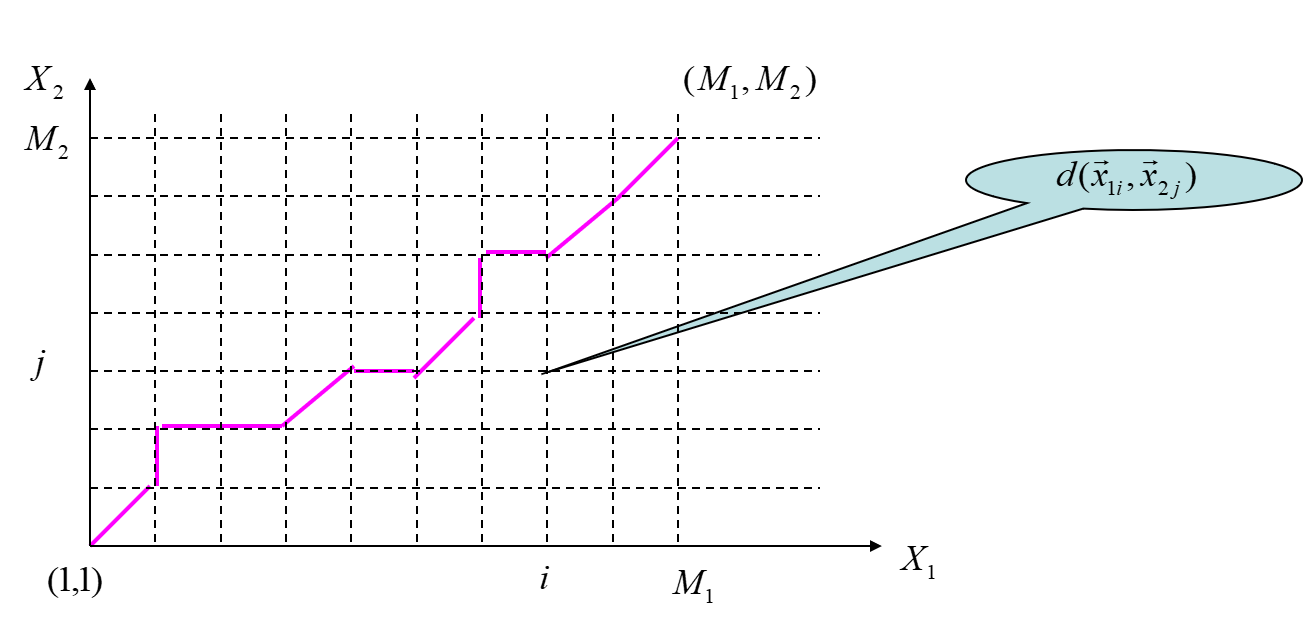
DTW 本质上是一个简单的动态规划算法，是用来计算两个维数不同的向量之间的相似度的问题，即计算向量 M1 和 M2 的最短距离。是一种非常常用的语音匹配算法，算法如下。

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：



那么，就可以这样进行匹配：



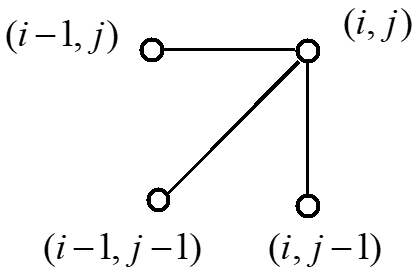
1.每一条从（1，1）倒（M1,M2）路径都有一个累计距离称为路径的代价.

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。

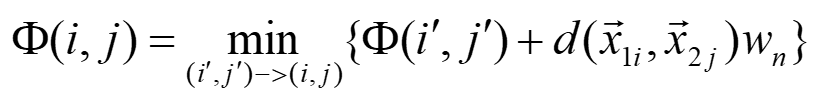
这样就可以将对准问题，或者说将求两个语音段的相似度问题，转化成了搜索代价最小的最优路径问题。

在搜索过程中，往往要进行路径的限制：

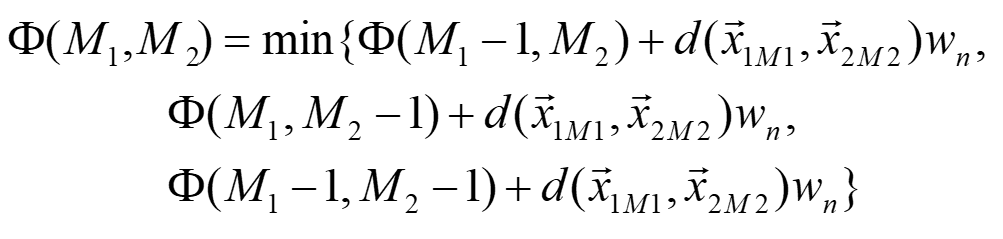


在此限制条件下，可以将全局最优化问题转化为许多局部最优化问题一步一步地来求解，这就动态规划(Dynamic Programming，简称DP )的思想。

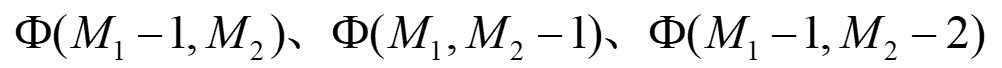
1.定义一个代价函数 ， 表示从起始点(1,1)出发，到达（*i*，*j*）点最小代价路径的累计距离。有：



2.这样的话，要计算两个向量之间的最短距离，可以表示为：



依次类推，可以得到：



可由更低一层的代价函数计算得到

3.这样就可以从  开始计算，定义加权系数，加权系数的取值与局部路径有关：



4.定义回溯函数：

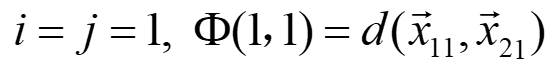


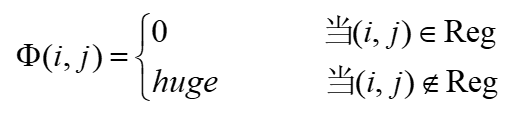
5.如图，为了减少计算量，可以定义平行四边形区域约束：



算法步骤：

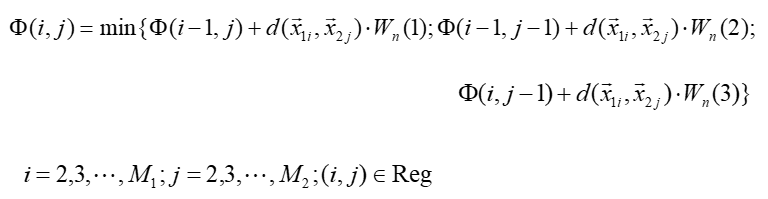
1. 初始化：





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(M1*,M*2)，相邻两条边的斜率分别为2和1/2。

1. 递推求累计距离 并记录回溯信息:



一般取距离加权值为:



3.计算出的值就是 m1 和 m2 之间的距离。（有时还需要进行归正，不过这里并没有写）

代码如下：

# DTW 算法...

def dtw(M1, M2) :

# 初始化数组 大小为 M1 \* M2

M1\_len = len(M1)

M2\_len = len(M2)

cost = [[0 for i in range(M2\_len)] for i in range(M1\_len)]

# 初始化 dis 数组

dis = []

for i in range(M1\_len) :

dis\_row = []

for j in range(M2\_len) :

dis\_row.append(distance(M1[i], M2[j]))

dis.append(dis\_row)

# 初始化 cost 的第 0 行和第 0 列

cost[0][0] = dis[0][0]

for i in range(1, M1\_len) :

cost[i][0] = cost[i - 1][0] + dis[i][0]

for j in range(1, M2\_len) :

cost[0][j] = cost[0][j - 1] + dis[0][j]

# 开始动态规划

for i in range(1, M1\_len) :

for j in range(1, M2\_len) :

cost[i][j] = min(cost[i - 1][j] + dis[i][j] \* 1, \

cost[i- 1][j - 1] + dis[i][j] \* 2, \

cost[i][j - 1] + dis[i][j] \* 1)

return cost[M1\_len - 1][M2\_len - 1]

# 两个维数相等的向量之间的距离

def distance(x1, x2) :

sum = 0

for i in range(len(x1)) :

sum = sum + abs(x1[i] - x2[i])

return sum

# 四、 计算正确率

## 4.1 正确率

正确检出的语料文件的个数：40

正确率= 100 %

# 五、 总结

## 5.1 请总结本次实验的收获

本次实验，让我对语音识别有了一个很好的认识。对语音的特征有了进一步的了解，对Python语言也更熟悉了

## 5.2 请给出对本次实验内容的建议

实验内容很好，很满意，指导书写的很明确，一点也不模糊，老师指导的也很到位，非常感谢。

## 5.3 介绍一下你开展了哪些扩展尝试，效果如何？

我开展了语音实时录取的尝试，在代码中动态对语音进行实时录取并进行匹配。实时录取2.5s的语音。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。