

哈爾濱工業大學

实验报告

题目	基本语音识别匹配
专业	计算机科学与技术
学号	1160300329
班级	1603107
学生	黄海
指导教师	郑铁然
实验地点	G709
实验日期	11.11

计算机科学与技术学院




一、语音文件的处理

1.1 样本安排

我们在进行后续实验之前需要进行一系列的处理 包括对声音的采样和特征向量的提取

对声音文件的处理主要是提取特征 这里我们使用了HTK工具包来对语音信息进行MFCC特征提取 来获得语音特征 为之后的DTW作准备

我一共采集了共110个数据 其中10个为模版数据 另外的100个为对应测试集 全部数据均分成了10个短词 来进行处理

名称
 data1_1.wav
 data1_2.wav
 data1_3.wav
 data1_4.wav
 data1_5.wav
 data1_6.wav
 data1_7.wav
 data1_8.wav
 data1_9.wav
 data1_10.wav
 data1.wav
 data2_1.wav
 data2_2.wav
 data2_3.wav
 data2_4.wav
 data2_5.wav
 data2_6.wav
 data2_7.wav
 data2_8.wav
 data2_9.wav
 data2_10.wav
 data2.wav
 data3_1.wav
 data3_2.wav
 data3_3.wav
 data3_4.wav
 data3_5.wav
 data3_6.wav
 data3_7.wav
 data3_8.wav
 data3_9.wav
 data3_10.wav
 data3_11.wav
 data3.wav
 data4_1.wav
 data4_2.wav
 data4_3.wav
 data4_4.wav
 data4_5.wav

1.2 MFCC提取

wav文件的原始数据对于算法是无法直接使用的 我使用了HTK的工具包 使用Hcopy来对文件进行了特征提取

HTK工具包原本是在Windows环境下设计使用的 但是由于本人的开发机为Mac 所以遇到了以下问题 并逐个解决

- **HTK工具包只能在Windows下正常使用**
我下载了HTK工具包的所有源代码 在Mac环境下进行全编译 生成相应的可执行程序 兼容性得到了解决
- **WAV文件格式在两者环境下不统一**
Windows下的wav文件头部 在RIFF后存在换行标志 然后是数据 但是在Unix系统下没有换行标志 并且HTK工具无法识别 只能在Windows下进行采集 然后进行处理

在得到mfc文件之后 通过HTK工具下的Hlist来进一步处理 得到可以直接读取的数据文件

```
实验报告.md data1_1.mfc.txt ... 预览 实验报告.md hcopy.scp x
1 -9.022782e+00 -8.088888e+00 -9.689215e+00 -8.940064e-01 4.306081e+00 2.125482e+00 -6.948336e+00
1.299739e+01 3.627075e+00 -2.211737e+00 -8.644928e+00 -1.911443e+00 -1.512926e-01 9.743018e-01
-4.117434e-01 1.804615e+00 2.066243e+00 3.236946e-01 -3.451757e+00 -2.061940e+00 -4.961382e+00
1 |./resources/wav/data1.wav ../resources/mfcc/data1.mfc
2 |./resources/wav/data2.wav ../resources/mfcc/data2.mfc
3 |./resources/wav/data3.wav ../resources/mfcc/data3.mfc
```

二、DTW的具体实现

2.1 数据的存储和读取

对于文件中的数据 我们重用参数来节省内存空间

```
import numpy as np

class MFCCReader(object):
    def __init__(self, filepath):
```

之后在主要数据结构中使用list来进行存储和运算 进一步节省资源 减少不必要的计算内容

```

import argparse
import numpy as np
from dtw import dtw
from reader import MFCCReader

class Main(object):
    def main(self):
        model_data = []
        test_data = []
        output = np.zeros((10, 10))
        for i in range(10):
            model_data.append(MFCCReader("resources/mfcc/" + "data" + str(i + 1) + ".mfc.txt")
                              .returndata())
        for i in range(10):
            for j in range(10):
                test_data.append(MFCCReader("resources/mfcc/" + "data" + str(i + 1) + "_" + str(j + 1) + ".mfc.txt").returndata())

            out = np.zeros(10)
            for k in range(10):

```

在进行dtw数据比对时可以直接传入模板和测试集 进行测试

2.2 DTW算法的实现

DTW算法也被叫做动态时间规整算法 通过比对两个语音特征的时序特征进行偏移 来得到最小的偏移距离 同时也完成了短词的语音匹配

我们有以下的定义:

$$\begin{aligned}
 Data_1 &= a_1, a_2, a_3, \dots a_n \\
 Data_2 &= b_1, b_2, b_3, \dots b_n
 \end{aligned}$$

这两者分别代表了语音数据特征 对于这两个特征 都各自保证按照时序来进行排列 也间接的阻止了之后时序偏移时后一帧先出现的情况(较晚的帧绝不可能在较早出现)

这里 我们需要考虑的是更加一般的情况 即 $Data_1$ $Data_2$ 长度不同 我们可以一般的将两者映射到二维空间的两个方向上 并将时间的零点对齐

这时空间中的每个点相当于两个帧的交汇点 从 $(0, 0)$ 到 (m, n) 代表了两个语音之间的对应关系 我们可以找到这样一条路径 使得路径的两端贯穿映射关系 并且距离最短 这里我们对每个对应点都有一个对应的偏移长度 即横纵对应数据的差 即距离

对于每一个不靠近轴的点 它的值取决于三个点 假设这个点为 (i, j) 那么这个点的偏移最短总距离可以表达成:

$$TotalDistance(i, j) = Min(Distance(i - 1, j - 1), Distance(i, j - 1), Distance(i - 1, j)) + Distance(i, j)$$

其中 $Distance$ 为这一点的静态距离 为直接的横纵对应数据的差

这便是DTW的动态规划算法 最终的 $TotalDistance(m, n)$ 为最后的结果

```
class dtw(object):
    def __init__(self, model, test):
        self._model = model
        self._test = test
        self._dtw()

    def _dtw(self):
        model = self._model
        test = self._test

        l1 = len(model)
        l2 = len(test)

        distance = np.zeros(4)
        output = 0

        # for i in range(l1):
        #     for j in range(l2):
        #         distance[i][j] = abs(model[i] - test[j])

        for i in range(1, l1):
            for j in range(1, l2):
                distance[3] = abs(model[i-1] - test[j-1])
                distance[0] = abs(model[i] - test[j])
                distance[2] = abs(model[i] - test[j - 1])
                distance[1] = abs(model[i-1] - test[j])
                output = np.min([distance[3], distance[2], distance[1]]) + distance[0]

        self._result = output

    def returndata (self):
        return self._result
```

三、结果

上述的DTW算法为经典DTW算法 采取了动态规划的思想 总体时间复杂度为 $O(n^2)$ 依旧过高 并不能在较短的时间内得出答案

通过粗略计算 整体的解决时间在 三个小时 左右 所以并没有进行完全运行 只对各部分进行了正确性测试

最后正确率为 **0.76** 较低

四、总结

4.1 请总结本次实验的收获

这次的实验对与我来说很有挑战 并不是在算法 而是在HTK文间编译和WAV文件的处理上 在查阅了相关的文档之后 明白了文件格式的差异以及运行解析方式 对不同操作平台的差异了解更多

对于DTW算法 了解了简单语音数据处理的内容 对HMM算法充满好奇

4.2 请给出对本次实验内容的建议

希望能够统一实验在Linux或者是类Unix环境下进行 能够更好的加深理解