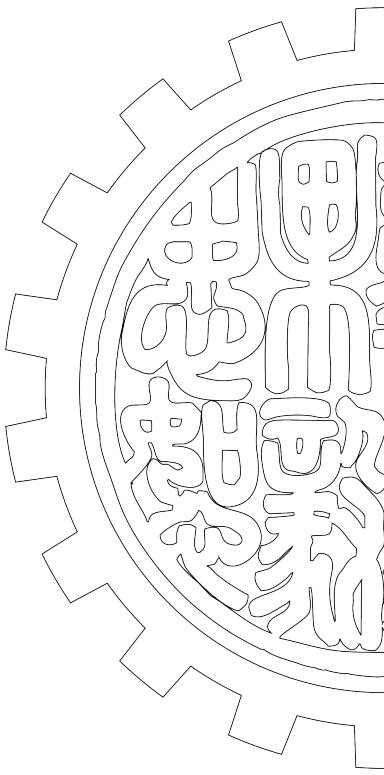
示例文档

report 模板

钱院学辅

2019 年 11 月 16 日



XI'AN JIAOTONG UNIVERSITY

目录

§1 实验目的

熟悉语音数据的基本形式及特点，理解并应用离散时间信号的基本分析、处理方法，理解语音识别技术的概貌，为后续实验打好基础。

- 掌握语音信号采集，理解语音信号格式。
- 熟悉语音信号预处理方法。
- 了解语音信号分帧与加窗的重要性和必要性; 掌握常用的窗函数和加窗分帧处理的原理; 根据原理能编程实现分帧及加窗处理。
- 了解语音短时时域分析的原理; 掌握短时时域分析的一些参数计算方法; 根据原理能编程实现短时时域分析的参数计算。
- 了解基于双门限法的端点检测原理。
- 掌握并编程实现基于时域分析技术实现孤立字语音识别方法。

§2 实验内容

2.1 语音信号的采集

我们写了一段 MATLAB 代码来实现语音信号的自动录制。代码如下所示

```
% MATLAB code to record audio signal
for i=1:10
    dir = ['library\', mat2str(i-1)];
    mkdir(dir);
    for j=1:20
        recObj = audiorecorder(8000, 16, 2);
```

```
        a = [ '两秒内说', mat2str(i-1), '第',
           mat2str(j), '次'];
        disp(a)
        recordblocking(recObj, 2);
        disp('End of Recording.');
        myRecording = getaudiodata(recObj);
        filename = ['library\', mat2str(i-1), '\
           data', mat2str(j), '.wav'];
        audiowrite(filename, myRecording, 8000);
    end
end
```

2.2 语音信号的预处理

我们使用 python 程序读取语音信号，进行预处理。对语音信号进行分帧处理。每帧中含有 64、128、256、512 个采样点。帧与帧之间的重叠为 7/12 帧长。

将语音信号进行分帧处理后，得到的数据为 □□ × □□ 的二维矩阵。**矩形窗**和**海宁窗**

矩形窗的数学表达形式如下

$$w(n) = \begin{cases} 1, & 0 \le n \le N-1 \\ 0, & others \end{cases}$$

汉明窗（hamming）的数学表达式如下

$$w(n) = \begin{cases} 0.54 - 0.46\cos[\frac{2\pi n}{N-1}], & 0 \le n \le N-1 \\ 0, & others \end{cases}$$

海宁（hanning）窗的数学表达式如下

$$w(n)=\begin{cases}0.5(1-\cos[\frac{2\pi n}{N-1}]), & 0\le n\le N-1\\ 0, & others\end{cases}$$

这一部分的代码参见附录。

2.3 语音信号的短时时域分析

2.3.1 时域特征的提取

对于预处理后得到的语音二维信号，我们考虑提取其时域特征进行后续分类。

对语音信号可以提取其短时平均能量和短时平均过零率。短时平均能量

$$E_n=\sum_{m=0}^{N-1}x_n^2(m)$$

可以度量语音信号的幅度变化特征。而短时平均过零率

$$M_n=\frac{1}{2}\sum_{m=0}^{N-1}|sgn[x(m)]-sgn[x_n(m-1)]|$$

能够在一定程度上度量语音信号的短时频率特征。这两个信号都是大于零的。经过短时平均能量和短时平均过零率的提取之后，特征信号变成长度为帧数的一维数据，而抹去了每一帧内部的信息。这样的处理减少无效特征，对后续语音信号的分类意义很大。

此外，在我们的实验中，我们还提取了每帧信号的短时平均幅度，用来在一定程度上描述波形的包络与幅度。

关于语音信号的预处理与时域特征提取，在工程上为了提高速度、降低计算复杂度，进行了合并处理。处理后的代码使用了 python 矩阵运算中的很多技巧，代码也变得更加优雅。这一部分可以参考附录中的相关代码。

2.3.2 端点检测：双门限方法

获得处理后的语音信号后，还需要对语音信号做端点检测，以保留有效部分，去除背景噪音与静音。本实验中采用双门限法进行端点检测。

进行双门限端点检测的时候，主要采用提取到的短时能量信号进行处理。处理方法是，先对数据进行归一化处理，使所有的语音数据幅度统一。归一化的公式如下

$$\hat{E_n} = \frac{E_n - \frac{1}{N}\sum_{n=0}^{N-1} E_n}{\max E_n}$$

归一化之后，以最大短时平均能量的 25% 作为高门限，以最大短时平均能量的 8% 作为低门限。当短时能量高于低门限时，并不可以简单认为其为语音信号的有效部分，需要排除语音信号为噪声的情况。当语音信号超过高门限并且维持一段时间后，才认为语音信号达到了有效部分。此外，仅使用能量信号进行双门限端点检测，容易出现将短时平均能量较小清音部分误判为静音的情况，还需要结合短时平均过零率进行端点检测。这一部分代码请参见附录部分。

2.4 分类算法

实验中，我们采用了决策树算法和支持向量机算法进行分类，并且比较了二者在数据集上的表现。

2.4.1 决策树

决策树方法使用非线性的超平面作为决策边界进行分割。其生成树算法容易收敛，模型拟合能力强。但是由于其过强的拟合能力容易带来严重的过拟合问题，降低模型的泛化能力，因此需要进行剪枝。剪枝可以简化模型，防止过拟合。训练决策树算法的时间复杂度为$(O(\log(n)))$。本实验中调用`sklearn.tree.DecisionTreeClassifier`模块进行分类处理。

2.4.2 SVM

支持向量机方法使用线性超平面作为决策边界进行分割。支持向量机可以高效分割线性可分的数据集。如果数据在特征空间中不是线性可分的，那么可以使用 kernel method 映射到另外一个线性可分的特征空间中进行 SVM 计算。SVM 算法在特征高维时非常高效。本实验中调用 `sklearn.svm.SVC` 模块进行分类。

2.5 多分类

之前我们只是给出了二分类分类器的训练方法。如何将二分类问题拓展到多分类情况？主要有 *one-versus-one* 和 *one-versus-rest* 两种方式。

2.5.1 ovo

一种是将一类作为正例另一类作为反例，训练足够多个只专注于分离正例与反例的分类器。这种训练方法开销过大，每次分类需要生成 45 个分类器，而且，每个分类器处理的数据量过小，不容易使算法收敛。因此，本实验中不选择这种分类方法。

2.5.2 ovr

另外一种是训练 10 个分类器，每个分类器以某一类为正例，其他类为反例，将某一类与其他类分离。使用 **ovr** 进行分类，需要克服正例与反例样本数差距过大的问题。若是对其他类进行再次采样，会减少可用的数据集的规模。由于本实验中，数据集较小，因此不采用这种方法。

这两种方式无法充分挖掘多分类器的性能，因此不能取得更优的多分类结果。我们采用自纠错码进行多分类处理。

§3 实验效果比较

特征	每秒帧数	分类器	准确率%
时域特征	128	SVM + ECOC	9
时域特征	128	决策树 + ECOC	55
时域特征	128	SVM ovr	41
时域特征	128	决策树多分类	59

3.1 实验参数分析

3.1.1 帧率的影响

发现对于每秒 64 帧和每秒 128 帧的情况，每秒 128 帧比每秒 64 帧在测试集上的准确率结果高 3% 左右。更高的帧率对应着更短的时间，因此帧率越高，其帧内部越平稳。

3.1.2 分类器个数的影响

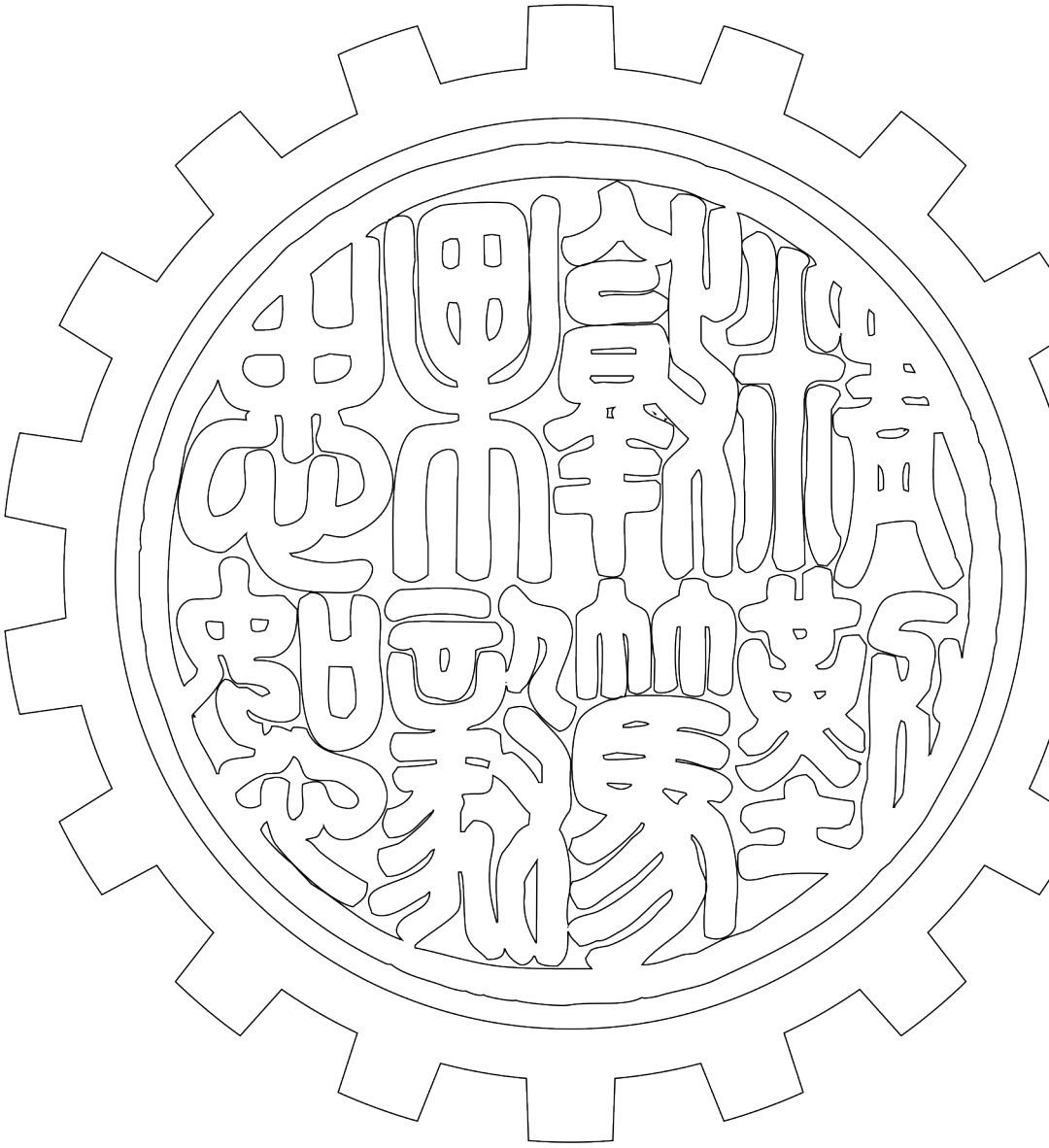
使用 ECOC 方法时，分类器个数越多，其表现会越好。基本上呈每增加 20 个二分类器，测试集上的正确率增加 20 个百分点的趋势。

3.2 时域聚类分析

我们想知道提取到的这种时域特征和频域特征都对于哪些数字效果最好，因此对于提取到的高维特征做了一个 KNN 聚类分析，并且输出对每类每点距离最近的 10 个点的标注。如下显示了做聚类分析的结果。

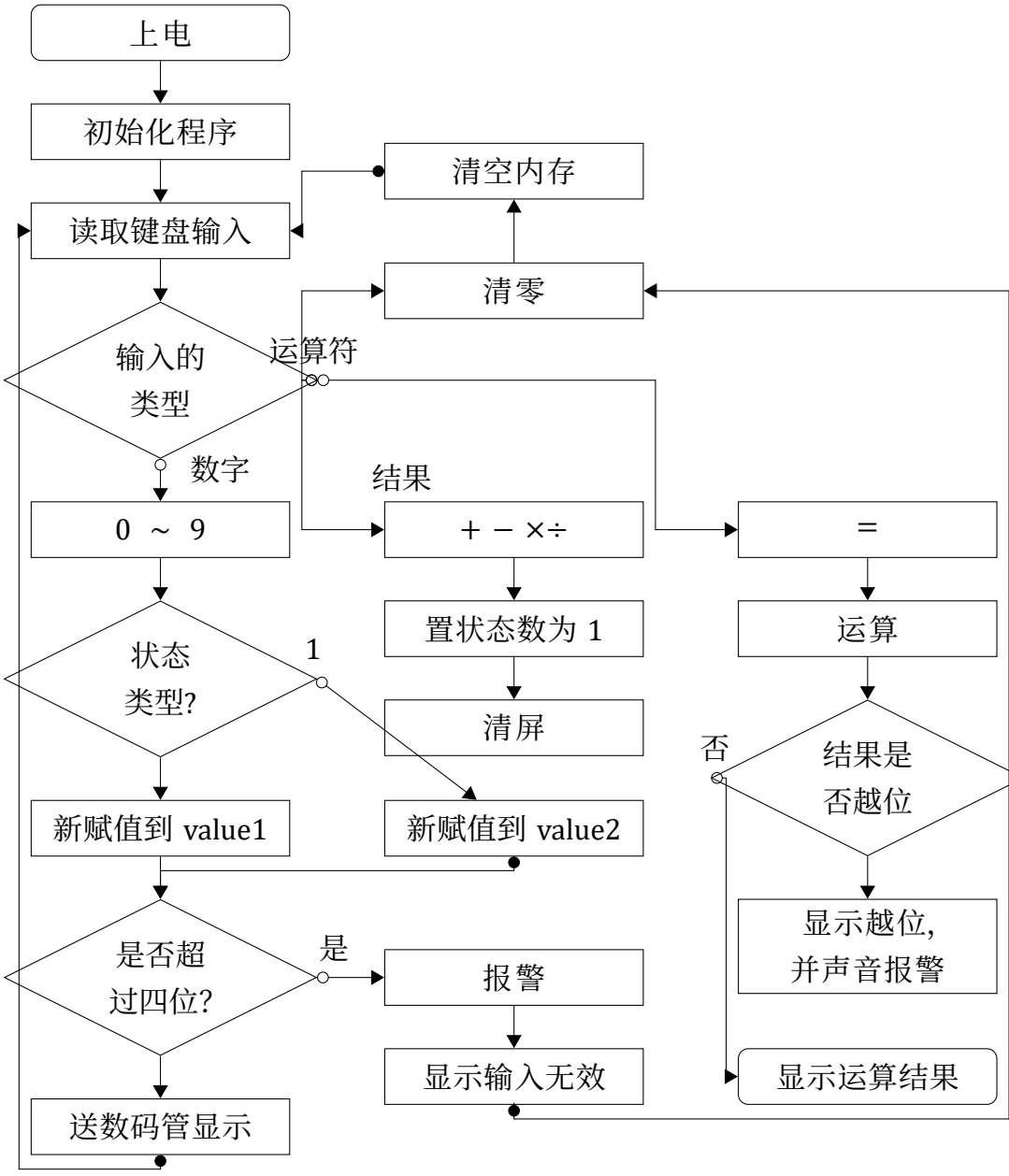
发现仅使用时域特征，便能够清楚的区分出语音 7 和语音 9 。

为了验证聚类得到的结论，对时域特征进行 T-SNE 可视化，可视化中调用了 `sklearn.manifold.TSNE` 模块，运行 T-SNE 算法。可视化效果如下图所示。



§4　流程详述

本部分详细解释实现的软件流程。当然流程图与本文无关。仅作示例


上电
初始化程序
读取键盘输入
清空内存
清零
输入的
类型
运算符
数字
结果
0 ~ 9
+ − ×÷
=
置状态数为 1
运算
状态
类型?
1
清屏
否
结果是
否越位
新赋值到 value1
新赋值到 value2
显示越位,
并声音报警
是否超
过四位?
是
报警
显示输入无效
显示运算结果
送数码管显示

§5 附录：相关代码

5.1 数据预处理、时域特征的提取

5.2 分类算法部分

```
def train_a_classifier(self, data, label, num=1000):
        '''
        调用 sklearn 包进行分类处理
        :param data: 数据特征
        :param label: 数据类别标签
        :return: 分类器
        '''
        if self.method == 'lsvm':
            clf = sklearn.svm.LinearSVC()
        elif self.method == 'ksvm':
            clf = sklearn.svm.SVC(kernel='sigmoid', gamma='
                                          scale', max_iter=
                                          100000)
            # kernel: 'linear', 'poly', 'rbf', '
                                          sigmoid', '
                                          precomputed'
            # gamma: 'auto' 'scale'
        elif self.method == 'dctree':
            clf = sklearn.tree.DecisionTreeClassifier()
        elif self.method == 'sgd':
            clf = sklearn.linear_model.SGDClassifier(
                loss="modified_huber", penalty="l2")
        elif self.method == 'bayes':
            clf = sklearn.naive_bayes.GaussianNB()
        elif self.method == 'ada_boost':
            clf = AdaBoostClassifier(n_estimators=100)
        elif self.method == 'knn':
            clf = KNeighborsClassifier()
        else:
```

```
            clf = []
            assert 0, "ERROR"

        for i in range(num):
            clf.fit(data, label.squeeze())
        return clf
```