基于时域分析技术的语音识别

任泽华	自动化 71	学号:2171411498
姬文虎	自动化 71	学号:2176413411
程敏敏	自动化 71	学号:2171421548
刘雨芊	自动化 74	学号:2173724059
秦祯	自动化 75	学号:2175021232

2019.10.7

基于时域分析技术的语音识别问题

摘要

语音是一种典型的、易于获取的一维时序信号。语音信号处理及识别技术是数字信号处理课程的重要应用板块。时间序列分析、快速傅里叶变换、滤波器设计等多项数字信号处理的教学内容在语音识别核心技术中均占有重要地位。

本次实验面向孤立字语音识别基本任务，由时域角度进行探讨、分析、解决。本文将通过能识别数字 0～9 的语音识别系统的实现过程详细阐述基于 KNN、随机森林等算法的特定孤立词识别的相关原理和关键技术.

在信号采集部分，首先通过编程方式运用 audiorecorder 函数进行了语音信号的采集并建立了班级语音信号库。在预处理部分，我们对语音信号进行了批量处理：运用汉明窗函数进行分帧加窗处理，短时时域特性分析、基于双门限法的端点检测。基于时域分析提取的特征向量，本文选择了 KNN、随机森林等分类器进行了孤立字语音识别。最后进行了实验对比及量化分析。

运用 KNN，最后的识别准确度为 55%左右；运用随机森林，最后的识别准确度为 20%左右；运用隐马尔可夫模型，最后的识别准确度为 20%左右。综上。KNN 算法有更好的识别效果。

【关键词】语音识别 时域分析法 MATLAB KNN 分类器

一．问题分析

1.1 问题背景

语音信号处理是一个新兴的交叉学科，是语音和数字信号处理两个学科的结合产物。语音信号处理技术的发展依赖于这些学科的发展，语音信号处理技术的进步也将促进这些领域的进展。语音信号处理目的是得到一些语音特征参数，以便高效的传输或存储，或通过某种处理以达到特定目的，如语音合成，辨识出讲话者、识别出讲话的内容等。随着现代科学技术和计算机技术的发展，除了人与人的自然语言的沟通，人机对话和智能机领域也开始使用语言。这些人造的语言拥有词汇，语法，语法结构和语义内容等。自动语音识别技术起源于 20 世纪 50 年代，最早的商用系统是 IBM 在 90 年代推出的 ViaVoice。经过半个多世纪的发展，语音识别技术目前已日趋成熟并成功应用到人们的日常生活之中，如苹果手机的 Siri 体验、科大讯飞的迅速崛起等。根据识别的对象不同，语音识别任务大体可分为 3 类，即孤立词识别（isolated word recognition），关键词识别（keyword spotting）和连续语音识别。其中，孤立词识别 的任务是识别事先已知的孤立的词，如“开机”、“关机”等。本题即属于孤立词识别。

语音是一种典型的、易于获取的一维时序信号，语音信号处理及识别技术也是数字信号处理课程绝佳的实践途径。时间序列分析、快速傅里叶变换、滤波器设计等多项数字信号处理的教学内容在语音识别核心技术中均占有重要地位。本次实验即面向语音识别基本任务，需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。

1.2 问题提出

1. 采集“0”-“9”语音信号
2. 进行语音信号预处理：语音信号的分帧与加窗、语音信号短时时域分析、基于双门限法的端点检测等
3. 编程实现基于时域分析技术的孤立字语音识别

二．问题解决

2.1 采集“0”-“9”语音信号

本文在 MATLAB 环境中使用 audiorecorder 函数录制语音信号，采集“0”、“1”…“9”这 10 个语音的 wav 文件，每个类别应采集 10 组以上的样本。为获得更多数据集，本班建立了班级语音库。

2.2 语音信号预处理

2.2.1 语音信号的分帧与加窗

贯穿于语音分析全过程的是“短时分析技术”。因为，语音信号从整体来看其特性及表征其本质特征的参数均是随时间而变化的，所以它是一个非平稳态过程，不能用处理平稳信号的数字处理技术对其进行处理。但是，由于不同的语音是由人的口腔肌肉运动构成声道某种形状而产生的响应，而这种口腔肌肉运动相对于语音频率来说是非常缓慢的，所以从另一个方面看，虽然语音信号具有时变特性，但是在一个短时间范围内（一般认为在 10-30 ms 的时间内），其特性基本保持不变即相对稳定，因而可以将其看做是一个准稳态过程，即语音信号具有短时平稳性。所以任何语音信号的分析和处理必须建立在“短时”的基础上，即进行“短时分析”，将语音信号分为一段一段来分析其特征参数，其中每一段称为一“帧”，帧长一般取 10-30 ms。这样，对于整体的语音信号来讲，分析出的是由每一帧特征参数组成的特征参数时间序列。 分帧是用可移动的有限长度窗口进行加权的方法来实现的，这就是用一定的窗函数 w(n)来乘 s(n)，从而形成加窗语音信号 $S_W = \mathrm{s(n)} * \mathrm{w(n)}$ 。窗函数 w(n) 的选择（形状和长度），对于短时分析参数的特性影响很大。为此应选择合适的窗口，使其短时参数很好地反映语音信号的特性变化。在语音信号数字处理中常用的窗函数有三种：

$$(1)\textit{矩形窗}\, w(n) = \begin{cases} 1, & 0 \le n \le N-1 \\ 0, & \textit{其他} \end{cases}$$

$$(2)\text{汉明窗 } \mathrm{w(n)} = \begin{cases} 0.54 - 0.46cos\left[\dfrac{2\pi n}{N-1}\right], & 0 \le n \le N-1 \\ 0, & \textit{其他} \end{cases}$$

$$(3)\text{海宁窗 } \mathrm{w(n)} = \begin{cases} 0.5\left(1 - cos\left[\dfrac{2\pi n}{N-1}\right]\right), & 0 \le n \le N-1 \\ 0, & \textit{其他} \end{cases}$$

本次实验，我们选取汉明窗作为加窗函数。

2.2.2 语音信号短时时域分析

语音信号的时域分析就是分析和提取语音信号的时域参数。时域分析通常用于最基本的参数分析及应用，如语音的分割、预处理、分类等。语音信号的时域参数有多种，本实验着重掌握短时能量、短时平均幅度及短时过零率。

[1]、 短时能量与短时平均幅度

设第 n 帧语音信号 $x_n(m)$ 的短时能量用E_n表示，则其计算公式如下：

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

E_n是一个度量语音信号幅度值变化的函数，但它有一个缺陷，即它对高电平非常敏感（因为它计算时用的是信号的平方）。为此，可采用另一个度量语音信号幅度值变化的函数，即短时平均幅度函数 M_n ，它的定义为：

$$M_n = \sum_{m=0}^{N-1} |x(m)|$$

M_n也是一帧语音信号能量大小的表征，它与 E_n 的区别在于计算时小取样值和大取样值不会因取平方而造成较大差异，在某些应用领域中会带来一些好处。所以，本文选择 M_n作为语音信号能量大小的表征。

[2]、 短时过零率

短时过零率表示一帧语音中语音信号波形穿过横轴（零电平）的次数。对于连续语音信号，过零即意味着时域波形通过时间轴；而对于离散信号，如果相邻的取样值改变符号则称为过零。因此，过零率就是样本改变符号的次数。过零率实质上是信号频谱分布在时域的一种最简单的体现，即高频分量丰富的信号其过零率也一般较高。

设第 n 帧语音信号 $x_n(m)$ 的短时过零率用 Z_n 表示，则其计算公式如下：

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

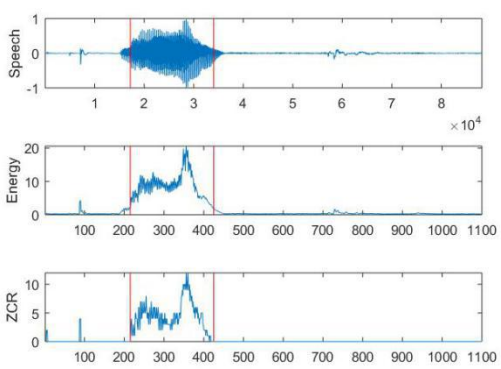
式中，sgn[.]是符号函数

其具体计算步骤如下：

a）首先对信号进行去直流化；

b）然后按照时间顺序统计采样点数值符号变号的次数；

c）将上述计数出的次数针对序列时长进行归一化操作，即得到过零率。



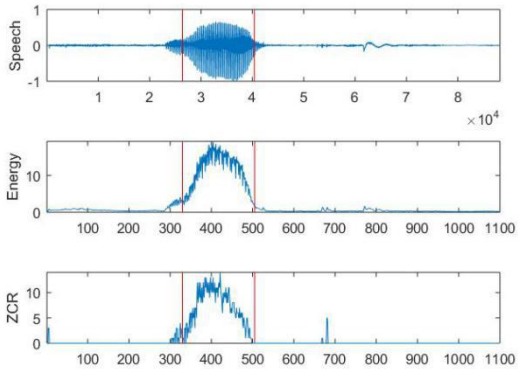


图 1（左）“0”的短时时域分析结果

图 1（右）“1”的短时时域分析结果

2.2.3 基于双门限法的端点检测

根据语音的统计特性，可以把语音段分为清音、浊音以及静音（包括背景噪声）三种。语音端点检测本质上是根据语音和噪声的相同参数所表现出的不同特征来进行区分。在双门限法中，短时能量可以较好地区分出浊音和静音。对于清音，由于其能量较小，在短时能量检测中会因为低于能量门限而被误判为静音；短时过零率则可以从语音中区分出静音和清音。将这两种检测结合起来，就可以检测出语音段（清音和浊音）及静音段。在基于短时能量和过零率的双门限端点检测算法中首先为短时能量和过零率分别确定两个门限，一个为较低的门限，对信号的变化比较敏感，另一个是较高的门限。当低门限被超过时，很有可能是由于很小的噪声所引起的，未必是语音的开始，到高门限被超过并且在接下来的时间段内一直超过低门限时，则意味着语音信号的开始。

2.3 基于时域分析技术实现孤立字语音识别

针对提取完成的语音特征向量，选取合适的分类器算法来实现自动语音判别。选择的分类器包括随机森林、KNN 等。具体实验情况如下：

2.3.1 选取 KNN 作为分类器

KNN 是通过测量不同特征值之间的距离进行分类。它的思路是：如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别，其中 K 通常是不大于 20 的整数。KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。KNN 算法的结果很大程度取决于 K 的选择。在 KNN 中，通过计算对象间距离来作为各个对象之间的非相似性指标，避免了对象之间的匹配问题，在这里距离一般使用欧氏距离或曼哈顿距离：

$$d(x,y) = \sqrt{\sum_{k=1}^{n} |x_k - y_k|}$$

同时，KNN 通过依据 k 个对象中占优的类别进行决策，而不是单一的对象类别决策。这两点就是 KNN 算法的优势。KNN 的算法描述为计算测试数据与各个训练数据之间的距离；按照距离的递增关系进行排序；选取距离最小的 K 个点；确定前 K 个点所在类别的出现频率；返回前 K 个点中出现频率最高的类别作为测试数据的预测分类。

KNN 的应用步骤为：

（1） 导入特征数据；

（2） 归一化数据；

（3） KNN 算法；

（4） 测试数据集；

（5） 预测函数。

2.3.2 选取随机森林作为分类器

随机森林学习模型具有随机性且由多棵决策树构成。随机性主要体现在随机抽样（行随机）和随机选择特征（列随机）两方面，可以很好地防止过拟合现象发生；多棵决策树可以防止模型泛化能力低现象发生，这些现象在研究中很常见。

随机森林的应用步骤为：

（1）选取语音信号的五个特征，将提取出的特征向量导入 excel 表格

（2）导入特征信息，调用 matlab 中的 TreeBagger 函数，作为随机森林分类器

（3）前 20 个信息作为训练集，最后一个作为测试集

（4）将分类结果与原本标签作比较，得出语音信号识别的准确率

（5）结果运行 100 次求均值，得出平均准确率

2.3.3 选取隐马尔可夫作为分类器

隐马尔可夫模型（Hidden Markov Model，HMM）是统计模型，它用来描述一个含有隐含未知参数的马尔可夫过程。其难点是从可观察的参数中确定该过程的隐含参数。然后利用这些参数来作进一步的分析，例如模式识别。是在被建模的系统被认为是一个马尔可夫过程与未观测到的（隐藏的）的状态的统计马尔可夫模型。

隐马尔可夫模型是马尔可夫链的一种，它的状态不能直接观察到，但能通过观测向量序列观察到，每个观测向量都是通过某些概率密度分布表现为各种状态，每一个观测向量是由一个具有相应概率密度分布的状态序列产生。所以，隐马尔可夫模型是一个双重随机过程----具有一定状态数的隐马尔可夫链和显示随机函数集。自 20 世纪 80 年代以来，HMM 被应用于语音识别，取得重大成功。到了 90 年代，HMM 还被引入计算机文字识别和移动通信核心技术“多用户的检测”。HMM 在生物信息科学、故障诊断等领域也开始得到应用。

1.评估问题。

给定观测序列 O=O1O2O3…Ot 和模型参数 λ=(A,B,π)，怎样有效计算某一观测序列的概率，进而可对该 HMM 做出相关评估。例如，已有一些模型参数各异的 HMM，给定观测序列 O=O1O2O3…Ot，我们想知道哪个 HMM 模型最可能生成该观测序列。通常我们利用 forward 算法分别计算每个 HMM 产生给定观测序列 O 的概率，然后从中选出最优的 HMM 模型。

这类评估的问题的一个经典例子是语音识别。在描述语言识别的隐马尔科夫模型中，每个单词生成一个对应的 HMM，每个观测序列由一个单词的语音构成，单词的识别是通过评估进而选出最有可能产生观测序列所代表的读音的 HMM 而实现的。

2.解码问题

给定观测序列 O=O1O2O3…Ot 和模型参数 λ=(A,B,π)，怎样寻找某种意义上最优的隐状态序列。在这类问题中，我们感兴趣的是马尔科夫模型中隐含状态，这些状态不能直接观测但却更具有价值，通常利用 Viterbi 算法来寻找。

3. 学习问题。

即 HMM 的模型参数 λ=(A,B,π)未知，如何调整这些参数以使观测序列 O=O1O2O3…Ot 的概率尽可能的大。通常使用 Baum-Welch 算法以及 Reversed Viterbi 算法解决。

2.3.4 选取 SVM 作为分类器

支持向量机(Support Vector Machine)是 Cortes 和 Vapnik 于 1995 年首先提出的，它在解决小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中。

支持向量机方法是建立在统计学习理论的 VC 维理论和结构风险最小原理基础上的，根据有限的样本信息在模型的复杂性（即对特定训练样本的学习精度，Accuracy）和学习能力（即无错误地识别任意样本的能力）之间寻求最佳折衷，以期获得最好的推广能力（或称泛化能力）。

主要步骤：

（1）根据给定的数据，选定训练集和测试集，并选定标签集；

（2）利用训练集进行训练分类器得到 model；

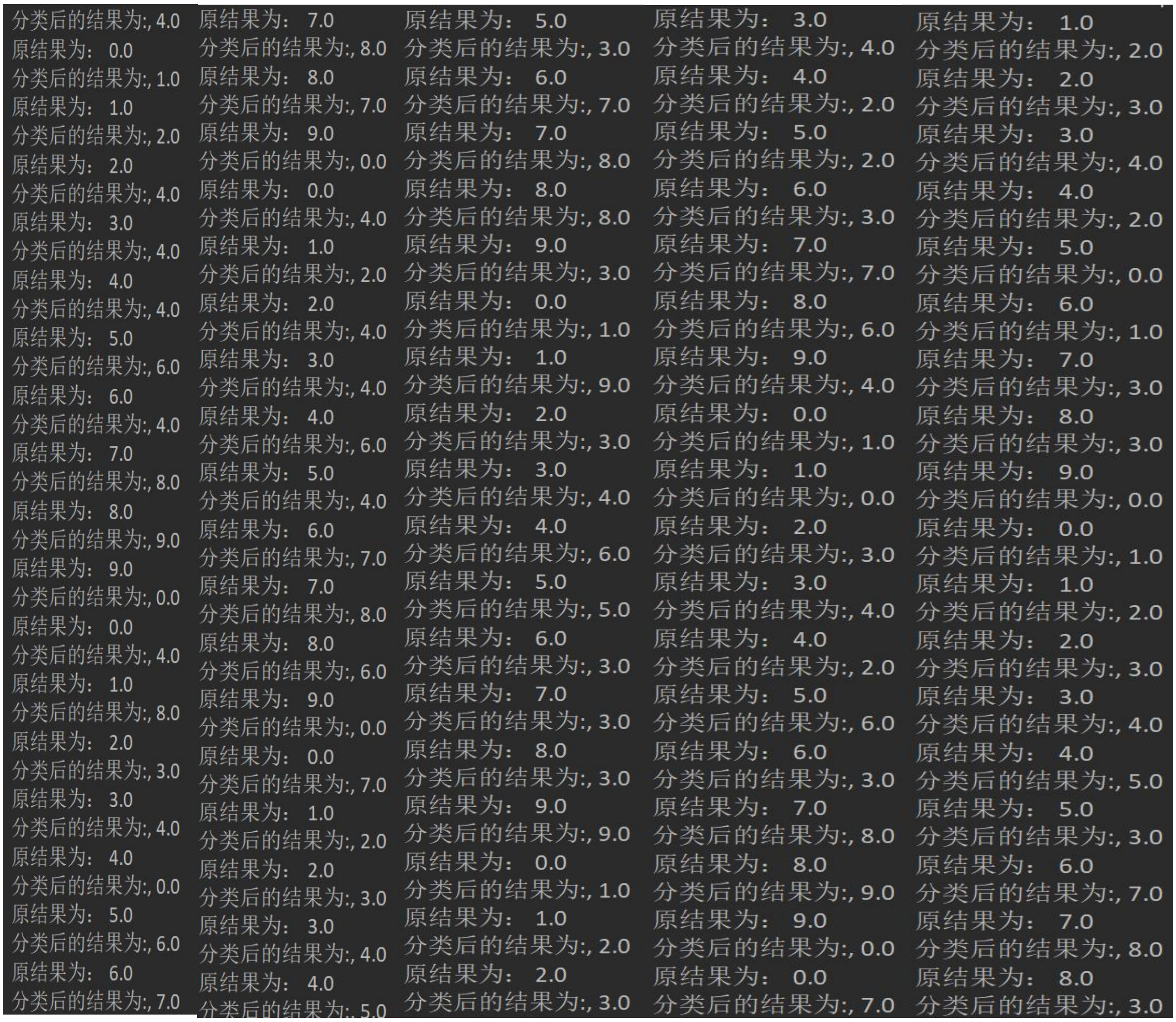
（3）根据 model，对测试集进行测试得到准确率。

三．实验对比与量化分析

3.1 选取 KNN 作为分类器

3.1.1 预测结果(部分)

运行部分结果如下图所示：



3.1.2 平均准确率

运行 100 次 KNN 结果的正确分类正确率仅有 55%,可见 KNN 算法针对此问题的分类效果比较可观。

3.2 选取随机森林作为分类器

3.2.1 预测结果(部分)

原始标签	0	1	2	3	4	5	6	7	8	9
预测 1	1	0	2	1	9	0	2	1	2	2
预测 2	1	0	2	6	1	1	0	0	8	2
预测 3	1	7	2	1	1	0	3	0	2	2
…	……									

3.2.2 平均准确率

results =	results =	results =	results =	results =
0.1870	0.1800	0.2030	0.2130	0.2100

3.3 选取隐马尔可夫模型作为分类器

3.3.1 预测结果(部分)

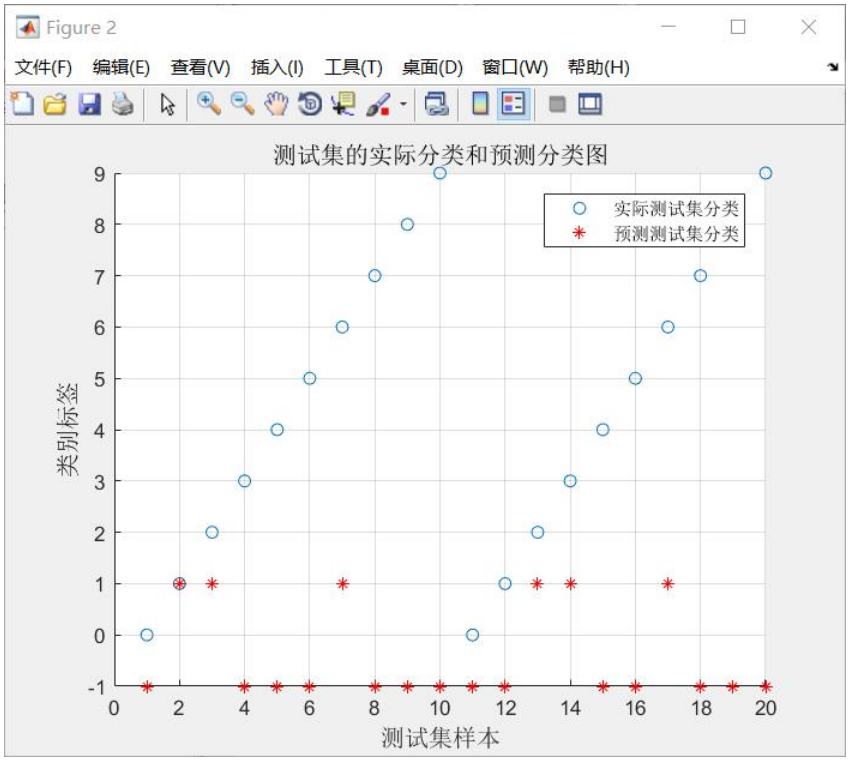
```
命令行窗口
    3,1 3,2 3,3
    4,1 4,2 4,3
    总和输出概率(log)=-6.136901e+03
    收敛!
    >> recog
    第1个词, 识别为5
    第2个词, 识别为9
    第3个词, 识别为2
    第4个词, 识别为3
    第5个词, 识别为6
    第6个词, 识别为5
    第7个词, 识别为5
    第8个词, 识别为5
    第9个词, 识别为5
    第10个词, 识别为5
    第11个词, 识别为5
    第12个词, 识别为9
    第13个词, 识别为5
    第14个词, 识别为5
    第15个词, 识别为6
    第16个词, 识别为5
    第17个词, 识别为5
    第18个词, 识别为2
    第19个词, 识别为5
    第20个词, 识别为5
fx >>
```

3.3.2 平均准确率

运行隐马尔可夫模型结果的正确分类正确率约有 20%.

3.4 选取支持向量机作为分类器

3.4.1 预测结果




```
*
optimization finished, #iter = 30
obj = 1718.300533, rho = 57.837007
nSV = 61, nBSV = 59
Accuracy = 5% (1/20) (classification)
```

3.4.2 平均准确率

经过 120 个数据的训练，对 20 个样本进行测试，识别的准确率约为 5%。

4.3 分类器对比

在 KNN 中，通过计算对象间距离来作为各个对象之间的非相似性指标，避免了对象之间的匹配问题，在这里距离一般使用欧氏距离或曼哈顿距离。同时，KNN 通过依据 k 个对象中占优的类别进行决策，而不是单一的对象类别决策。这两点就是 KNN 算法的优势。

随机森林模型的优势在于，基模型之间的独立性很强，所以该算法可以并行处理， 也就是基于随机森林和大数据平台可以很好地应对大数据时代的到来。随机抽样和随机抽取特征可以很好地防止过拟合现象；随机抽取特征也可以克服

特征维度过高问题；模型结构相对简单。

隐马尔可夫模型是马尔可夫链的一种，它的状态不能直接观察到，但能通过观测向量序列观察到，每个观测向量都是通过某些概率密度分布表现为各种状态，每一个观测向量是由一个具有相应概率密度分布的状态序列产生。所以，隐马尔可夫模型是一个双重随机过程----具有一定状态数的隐马尔可夫链和显示随机函数集。

SVM 支持向量机是 Cortes 和 Vapnik 于 1995 年首先提出的，它在解决小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中

四种方法各有优势，但由本次实验结果可得：对于孤立字语音识别，四种方法的准确率分别为 55%，21%，20%，5%，所以 KNN 的准确率更高。

参考文献

[1]郑南宁.数字信号处理简明教程.西安.西安交通大学出版社,2015
[2]奥本海姆.信号与系统.电子工业出版社,2012

附录

```
%vad 函数
function Character = vad(x)

%幅度归一化到[-1,1]
x = double(x);
x = x / max(abs(x));

%常数设置
FrameLen = 240;
FrameInc = 80;

amp1 = 10;
amp2 = 2;
zcr1 = 10;
zcr2 = 5;

maxsilence = 8;  % 8*10ms  = 80ms
minlen  = 15;    % 15*10ms = 150ms
status  = 0;
count   = 0;
silence = 0;

%计算过零率
% tmp1  = enframe(x(1:end-1), FrameLen, FrameInc);
tmp1  = enframe(x(1:end-1), hanning(FrameLen), FrameInc);%变汉宁窗
% tmp2  = enframe(x(2:end)  , FrameLen, FrameInc);
tmp2  = enframe(x(2:end), hanning(FrameLen), FrameInc);%变汉宁窗
signs = (tmp1.*tmp2)<0;
diffs = (tmp1 -tmp2)>0.02;
zcr   = sum(signs.*diffs, 2);

%计算短时能量
% amp = sum(abs(enframe(filter([1 -0.9375], 1, x), FrameLen, FrameInc)),
2);
amp = sum(abs(enframe(filter([1 -0.9375], 1, x), hanning(FrameLen),
FrameInc)), 2);%变汉宁窗
%调整能量门限
amp1 = min(amp1, max(amp)/4);
amp2 = min(amp2, max(amp)/8);

%开始端点检测
x1 = 0;
```

```
% x2 = 0;
for n=1:length(zcr)
%    goto = 0;
   switch status
   case {0,1}                   % 0 = 静音, 1 = 可能开始
      if amp(n) > amp1          % 确信进入语音段
         x1 = max(n-count-1,1);
         status  = 2;
         silence = 0;
         count   = count + 1;
      elseif amp(n) > amp2 || ... % 可能处于语音段
             zcr(n) > zcr2
         status = 1;
         count  = count + 1;
      else                       % 静音状态
         status  = 0;
         count   = 0;
      end
   case 2                       % 2 = 语音段
      if amp(n) > amp2 || ...     % 保持在语音段
         zcr(n) > zcr2
         count = count + 1;
      else                       % 语音将结束
         silence = silence+1;
         if silence < maxsilence % 静音还不够长，尚未结束
            count  = count + 1;
         elseif count < minlen   % 语音长度太短，认为是噪声
            status  = 0;
            silence = 0;
            count   = 0;
         else                    % 语音结束
            status  = 3;
         end
      end
   case 3
      break;
   end
end

count = count-silence/2;
x2 = x1 + count -1;

%画图图
% subplot(311)
```

```
% plot(x)
% axis([1 length(x) -1 1])
% ylabel('Speech');
% line([x1*FrameInc x1*FrameInc], [-1 1], 'Color', 'red');
% line([x2*FrameInc x2*FrameInc], [-1 1], 'Color', 'red');
%
% subplot(312)
% plot(amp);
% axis([1 length(amp) 0 max(amp)])
% ylabel('Energy');
% line([x1 x1], [min(amp),max(amp)], 'Color', 'red');
% line([x2 x2], [min(amp),max(amp)], 'Color', 'red');

%
% subplot(313)
% plot(zcr);
% axis([1 length(zcr) 0 max(zcr)])
% ylabel('ZCR');
% line([x1 x1], [min(zcr),max(zcr)], 'Color', 'red');
% line([x2 x2], [min(zcr),max(zcr)], 'Color', 'red');

ampMax=max(amp(x1:x2));
ampMin=min(amp(x1:x2));
ampAverage=sum(amp(x1:x2))/(x2-x1);
ZCRMax=max(zcr(x1:x2));
ZCRMin=min(zcr(x1:x2));
ZCRaverage=sum(zcr(x1:x2))/(x2-x1);
Character=[ampMax,ampMin,ampAverage,ZCRMax,ZCRMin,ZCRaverage];

end
```

```
%批量处理
path='F:\实验\dsp\时域语音信号处理\语音库\';      %语音库存储路径,根据自己的修改即可
samplenum=14;      %实验样本数（每个样本包括 0~9 10 个录音）

Character=zeros(samplenum*10,7);
%样本参数矩阵，7 列若干行，其中每列代表的参数为：
%[
%短时能量最大值，短时能量最小值，短时能量平均值，
%短时过零点数最大值，短时过零点数最小值，短时过零点数平均值，样本标签（0~9）
%]
```

```
for j=1:samplenum
    filename=dir([path,int2str(j),'\*.wav']);
    for i=1:size(filename)
        x=audioread([path,int2str(j),'\',filename(i).name]);
        Character((j-1)*10+i,:)=[vad(x),i-1];
    end
end
Character


%KNN
function Character = vad(x)

%•ù¶È¹éÒ»»¯µ½[-1,1]
x = double(x);
x = x / max(abs(x));

%³£ÊýÉèÖÃ
FrameLen = 240;
FrameInc = 80;

amp1 = 10;
amp2 = 2;
zcr1 = 10;
zcr2 = 5;

maxsilence = 8;  % 8*10ms  = 80ms
minlen  = 15;    % 15*10ms = 150ms
status  = 0;
count   = 0;
silence = 0;

%¼ÆËã¹ýÁãÂÊ
% tmp1  = enframe(x(1:end-1), FrameLen, FrameInc);
tmp1  = enframe(x(1:end-1), hanning(FrameLen), FrameInc);%±äººÄþ´°
% tmp2  = enframe(x(2:end)  , FrameLen, FrameInc);
tmp2  = enframe(x(2:end), hanning(FrameLen), FrameInc);%±äººÄþ´°
signs = (tmp1.*tmp2)<0;
diffs = (tmp1 -tmp2)>0.02;
zcr   = sum(signs.*diffs, 2);

%¼ÆËã¶ÌÊ±ÄÜÁ¿
% amp = sum(abs(enframe(filter([1 -0.9375], 1, x), FrameLen, FrameInc)),
```

```
2);
amp = sum(abs(enframe(filter([1 -0.9375], 1, x), hanning(FrameLen),
FrameInc)), 2);%±äººÄþ´°
%µ÷ÕûÄÜÁ¿ÃÅÏÞ
amp1 = min(amp1, max(amp)/4);
amp2 = min(amp2, max(amp)/8);

%¿ªÊ¼¶ËµãÌ¼ì²â
x1 = 0;
% x2 = 0;
for n=1:length(zcr)
%    goto = 0;
   switch status
   case {0,1}                   % 0 = ¾²Òô, 1 = ¿ÉÄÜ¿ªÊ¼
      if amp(n) > amp1          % È·ÐÅ½øÈëÓïÒô¶Î
         x1 = max(n-count-1,1);
         status  = 2;
         silence = 0;
         count   = count + 1;
      elseif amp(n) > amp2 || ... % ¿ÉÄÜ´¦ÓÚÓïÒô¶Î
             zcr(n) > zcr2
         status = 1;
         count  = count + 1;
      else                         % ¾²Òô×´Ì¬
         status  = 0;
         count   = 0;
      end
   case 2                       % 2 = ÓïÒô¶Î
      if amp(n) > amp2 || ...     % ±£³ÖÔÚÓïÒô¶Î
         zcr(n) > zcr2
         count = count + 1;
      else                         % ÓïÒô½«½áÊø
         silence = silence+1;
         if silence < maxsilence % ¾²Òô»¹»¹³¬£¬ÉÐÎ´½áÊø
            count  = count + 1;
         elseif count < minlen   % ÓïÒô³¬¶ÌÈÌ«¶Ì£¬ÈÏÎªÊÇÔëÉù
            status  = 0;
            silence = 0;
            count   = 0;
         else                    % ÓïÒô½áÊø
            status  = 3;
         end
      end
   case 3
```

```
      break;
   end
end

count = count-silence/2;
x2 = x1 + count -1;

%»-Í¼Í¼
% subplot(311)
% plot(x)
% axis([1 length(x) -1 1])
% ylabel('Speech');
% line([x1*FrameInc x1*FrameInc], [-1 1], 'Color', 'red');
% line([x2*FrameInc x2*FrameInc], [-1 1], 'Color', 'red');
%
% subplot(312)
% plot(amp);
% axis([1 length(amp) 0 max(amp)])
% ylabel('Energy');
% line([x1 x1], [min(amp),max(amp)], 'Color', 'red');
% line([x2 x2], [min(amp),max(amp)], 'Color', 'red');

%
% subplot(313)
% plot(zcr);
% axis([1 length(zcr) 0 max(zcr)])
% ylabel('ZCR');
% line([x1 x1], [min(zcr),max(zcr)], 'Color', 'red');
% line([x2 x2], [min(zcr),max(zcr)], 'Color', 'red');

ampMax=max(amp(x1:x2));
ampMin=min(amp(x1:x2));
ampAverage=sum(amp(x1:x2))/(x2-x1);
ZCRMax=max(zcr(x1:x2));
ZCRMin=min(zcr(x1:x2));
ZCRaverage=sum(zcr(x1:x2))/(x2-x1);
Character=[ampMax,ampMin,ampAverage,ZCRMax,ZCRMin,ZCRaverage];

end
```

```
path='C:\Users\fdft\Desktop\大三上实验\数字信号处理实验\语音库
```

```
';      %语音库存储路径，根据自己的修改即可
samplenum=14;      %实验样本数（每个样本包括 0~9 10 个录音）

Character=zeros(samplenum*10,7);
%样本参数矩阵，7 列若干行，其中每列代表的参数为：
%[
%短时能量最大值，短时能量最小值，短时能量平均值，
%短时过零点数最大值，短时过零点数最小值，短时过零点数平均值，样本标签
（0~9）
%]

for j=1:samplenum
    filename=dir([path,int2str(j),'\*.wav']);
    for i=1:size(filename)
        x=audioread([path,int2str(j),'\',filename(i).name]);
        Character((j-1)*10+i,:)=[vad(x),i-1];
    end
end
Character




train_data = xlsread('Character.xls','sheet1','A1:E200') ;
train_label = xlsread('Character.xls','sheet1','F1:F200') ;
test_data = xlsread('Character.xls','sheet1','A201:E210') ;
test_label = xlsread('Character.xls','sheet1','F201:F210') ;

% 随机森林分类器（Random Forest）

nTree = 5
result=zeros(100,1);
for j=1:100
    B = TreeBagger(nTree,train_data,train_label);
    predict_label = predict(B,test_data);
    for i=1:10
        if ischar(predict_label{i,1})
                predict_label{i,1}=str2num(predict_label{i,1});
        end
    end
    m=0;
    for i=1:10
        if predict_label{i,1}==test_label(i,1)
            m=m+1;
```

```
        end
    end
    result(j)=m/10;
end
results=mean(result)
```

```
%KNN
# coding:utf-8

from numpy import *
import operator
from collections import Counter
import xlrd


# 导入特征数据
def file2matrix():
    workbook                                                    =
xlrd.open_workbook(r'C:\Users\Administrator\Desktop\shuju.xls')
    # 改成自己的文件路径和文件名
    # print(workbook.sheet_names())
    sheet1 = workbook.sheet_by_index(0)  # 按 excel 的表格的顺序,从 0
开始
    # print(sheet1.name,sheet1.nrows,sheet1.ncols)#名字,行,列
    # returnMat =sheet1.row_values(0:6) 目标就是把 140*6 的矩阵整进去
    returnMat = []
    for i in range(sheet1.nrows - 10):
        returnMat = sheet1.row_values(i, 0, -1) + returnMat
    returnMat = array(returnMat).reshape(130, 5)
    # print(returnMat)
    classLabel = sheet1.col_values(5)
    # print(classLabel)
    return returnMat, classLabel  # 将文本中的数据导入到列表


# 归一化数据,保证特征等权重
def autoNorm(dataset):
    minVals = dataset.min(0)
    maxVals = dataset.max(0)
```

```
    ranges = maxVals - minVals
    normDataSet = zeros(shape(dataset))  # 建立与 dataSet 结构一样的矩
阵
    # print(normDataSet)
    m = dataset.shape[0]
    for i in range(1, m):
            normDataSet[i, :] = (dataset[i, :] - minVals) / ranges
    return normDataSet, ranges, minVals


# KNN 算法
def classify(input, dataSet, label, k):
    dataSize = dataSet.shape[0]
    # 计算欧式距离
    diff = tile(input, (dataSize, 1)) - dataSet
    sqdiff = diff ** 2
    squareDist = sum(sqdiff, axis=1)  # 行向量分别相加，从而得到新的一
个行向量
    dist = squareDist ** 0.5

    # 对距离进行排序
    sortedDistIndex = dist.argsort()
    # argsort()根据元素的值从大到小对元素进行排序，返回下标

    classCount = {}
    for i in range(k):
        voteLabel = label[sortedDistIndex[i]]
        # 对选取的 K 个样本所属的类别个数进行统计
        classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
    # 选取出现的类别次数最多的类别
    maxCount = 0
    for key, value in classCount.items():
        if value > maxCount:
            maxCount = value
            classes = key
    return classes


# 测试
def datingTest():
    testdata                                                    =
xlrd.open_workbook(r'C:\Users\Administrator\Desktop\shuju.xls')
    # 改成自己的文件路径和文件名
    # print(testdata.sheet_names())
```

```
    sheet1 = testdata.sheet_by_index(0)  # 按 excel 的表格的顺序,从 0
开始
    # print(sheet1.name,sheet1.nrows,sheet1.ncols)#名字,行,列
    datingDataMat = []
    for i in range(100):
        datingDataMat = sheet1.row_values(i, 0, -1) + datingDataMat
    datingDataMat = array(datingDataMat).reshape(100, 5)
    print(datingDataMat)
    datingLabels = sheet1.col_values(5, 0, 100)
    print(datingLabels)
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    testNum = 10
    # int(m * rate)
    errorCount = 0.0
    for i in range(100):  # testNum
        classifyResult = classify(normMat[i, :], normMat[0:m, :],
datingLabels[0:m], 10)
        print("分类后的结果为:,", classifyResult)
        print("原结果为: ", datingLabels[i])
        if (classifyResult != datingLabels[i]):
            errorCount += 1.0
    print("误分率为:", (errorCount / float(testNum)))


# 预测函数
def classifyPerson():
    resultList = ['0','1', '2', '3', '4', '5', '6', '7', '8', '9']
    DataMat, Labels = file2matrix()
    normMat, ranges, minVals = autoNorm(DataMat)
    inArr = array([6.401899671, 0.362470814, 3.133093082, 3,
1.441860465])  # Label 8
    classifierResult = int(classify((inArr - minVals) / ranges, normMat,
Labels, 11))
    print("分类类别为", resultList[classifierResult - 1])


classifyPerson()
datingTest()
```

%隐马尔可夫模型

Part1：

```
function f=enframe(x,win,inc)
nx=length(x(:));              % 取数据长度
nwin=length(win);             % 取窗长
if (nwin == 1)                % 判断窗长是否为 1，若为 1，即表示没有设窗
函数
   len = win;                 % 是，帧长=win
else
   len = nwin;                % 否，帧长=窗长
end
if (nargin < 3)               % 如果只有两个参数，设帧 inc=帧长
   inc = len;
end
nf = fix((nx-len+inc)/inc); % 计算帧数
f=zeros(nf,len);              % 初始化
indf= inc*(0:(nf-1)).';       % 设置每帧在 x 中的位移量位置
inds = (1:len);               % 每帧数据对应 1:len
f(:) = x(indf(:,ones(1,len))+inds(ones(nf,1),:));   % 对数据分帧
if (nwin > 1)                 % 若参数中包括窗函数，把每帧乘以窗函数
    w = win(:)';              % 把 win 转成行数据
    f = f .* w(ones(nf,1),:);  % 乘窗函数
end
```

Part2:

```
function [x1,x2] = vad(x)

%幅度归一化到[-1,1]
x = double(x);
x = x / max(abs(x));

%常数设置
FrameLen = 240;
FrameInc = 80;

amp1 = 10;
amp2 = 2;
zcr1 = 10;
zcr2 = 5;

maxsilence = 8;  % 6*10ms  = 30ms
minlen  = 15;    % 15*10ms = 150ms
status  = 0;
count   = 0;
silence = 0;
```

```
%计算过零率
tmp1  = enframe(x(1:end-1), FrameLen, FrameInc);
tmp2  = enframe(x(2:end)  , FrameLen, FrameInc);
signs = (tmp1.*tmp2)<0;
diffs = (tmp1 -tmp2)>0.02;
zcr   = sum(signs.*diffs, 2);

%计算短时能量
amp = sum(abs(enframe(filter([1 -0.9375], 1, x), FrameLen, FrameInc)),
2);

%调整能量门限
amp1 = min(amp1, max(amp)/4);
amp2 = min(amp2, max(amp)/8);

%开始端点检测
x1 = 0;
x2 = 0;
for n=1:length(zcr)
   goto = 0;
   switch status
   case {0,1}                   % 0 = 静音, 1 = 可能开始
      if amp(n) > amp1          % 确信进入语音段
         x1 = max(n-count-1,1);
         status  = 2;
         silence = 0;
         count   = count + 1;
      elseif amp(n) > amp2 | ... % 可能处于语音段
             zcr(n) > zcr2
         status = 1;
         count  = count + 1;
      else                      % 静音状态
         status  = 0;
         count   = 0;
      end
   case 2,                      % 2 = 语音段
      if amp(n) > amp2 | ...    % 保持在语音段
         zcr(n) > zcr2
         count = count + 1;
      else                      % 语音将结束
         silence = silence+1;
         if silence < maxsilence % 静音还不够长，尚未结束
            count  = count + 1;
         elseif count < minlen   % 语音长度太短，认为是噪声
```

```
            status  = 0;
            silence = 0;
            count   = 0;
         else                    % 语音结束
            status  = 3;
         end
      end
   case 3,
      break;
   end
end

count = count-silence/2;
x2 = x1 + count -1;
```

Part3：

```
function ccc = mfcc(x)
% 归一化 mel 滤波器组系数
bank=melbankm(24,256,8000,0,0.5,'m');
bank=full(bank);
bank=bank/max(bank(:));

% DCT 系数,12*24
for k=1:12
  n=0:23;
  dctcoef(k,:)=cos((2*n+1)*k*pi/(2*24));
end

% 归一化倒谱提升窗口
w = 1 + 6 * sin(pi * [1:12] ./ 12);
w = w/max(w);

% 预加重滤波器
xx=double(x);
xx=filter([1 -0.9375],1,xx);

% 语音信号分帧
xx=enframe(xx,256,80);

% 计算每帧的 MFCC 参数
for i=1:size(xx,1)
  y = xx(i,:);
  s = y' .* hamming(256);
  t = abs(fft(s));
```

```
  t = t.^2;
  c1=dctcoef * log(bank * t(1:129));
  c2 = c1.*w';
  m(i,:)=c2';
end

%差分系数
dtm = zeros(size(m));
for i=3:size(m,1)-2
  dtm(i,:) = -2*m(i-2,:) - m(i-1,:) + m(i+1,:) + 2*m(i+2,:);
end
dtm = dtm / 3;

%合并 mfcc 参数和一阶差分 mfcc 参数
ccc = [m dtm];
%去除首尾两帧，因为这两帧的一阶差分参数为 0
ccc = ccc(3:size(m,1)-2,:);
```

Part4：

```
function prob = mixture(mix, x)
%计算输出概率
%输入:
%  mix  -- 混合高斯结构
%  x    -- 输入向量, SIZE*1
%输出:
%  prob -- 输出概率

prob = 0;
for j = 1:mix.M
    m = mix.mean(j,:);
    v = mix.var (j,:);
    w = mix.weight(j);
    prob = prob + w * pdf(m, v, x);
end

% 加上 realmin, 以防止 viterbi.m 中计算 log(prob)时溢出
if prob==0, prob=realmin; end
```

Part5：

```
function [x,mn,mx]=melbankm(p,n,fs,fl,fh,w)
%MELBANKM    determine    matrix    for    a    mel-spaced    filterbank
[X,MN,MX]=(P,N,FS,FL,FH,W)
%
% Inputs:  p   number of filters in filterbank
```

```
%       n   length of fft
%       fs  sample rate in Hz
%       fl  low end of the lowest filter as a fraction of fs (default =
0)
%       fh  high end of highest filter as a fraction of fs (default = 0.5)
%       w   any sensible combination of the following:
%             't'  triangular shaped filters in mel domain (default)
%             'n'  hanning shaped filters in mel domain
%             'm'  hamming shaped filters in mel domain
%
%             'z'  highest and lowest filters taper down to zero (default)
%             'y'  lowest filter remains at 1 down to 0 frequency and
%              highest filter remains at 1 up to nyquist freqency
%
%              If 'ty' or 'ny' is specified, the total power in the fft
is preserved.
%
% Outputs: x     a sparse matrix containing the filterbank amplitudes
%                     If  x  is  the  only  output  argument  then
size(x)=[p,1+floor(n/2)]
%             otherwise size(x)=[p,mx-mn+1]
%       mn    the lowest fft bin with a non-zero coefficient
%       mx    the highest fft bin with a non-zero coefficient
%
% Usage:   f=fft(s);          f=fft(s);
%       x=melbankm(p,n,fs);       [x,na,nb]=melbankm(p,n,fs);
%       n2=1+floor(n/2);      z=log(x*(f(na:nb)).*conj(f(na:nb)));
%       z=log(x*abs(f(1:n2)).^2);
%       c=dct(z); c(1)=[];
%
% To plot filterbanks e.g.   plot(melbankm(20,256,8000)')
%



%       Copyright (C) Mike Brookes 1997
%
%       Last modified Tue May 12 16:15:28 1998
%
%                                    VOICEBOX          home          page:
http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
%   This program is free software; you can redistribute it and/or modify
```

```
%   it under the terms of the GNU General Public License as published by
%   the Free Software Foundation; either version 2 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You can obtain a copy of the GNU General Public License from
%   ftp://prep.ai.mit.edu/pub/gnu/COPYING-2.0 or by writing to
%   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139,
USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%

if nargin < 6
  w='tz';
  if nargin < 5
    fh=0.5;
    if nargin < 4
      fl=0;
    end
  end
end
f0=700/fs;
fn2=floor(n/2);
lr=log((f0+fh)/(f0+fl))/(p+1);
% convert to fft bin numbers with 0 for DC term
bl=n*((f0+fl)*exp([0 1 p p+1]*lr)-f0);
b2=ceil(bl(2));
b3=floor(bl(3));
if any(w=='y')
  pf=log((f0+(b2:b3)/n)/(f0+fl))/lr;
  fp=floor(pf);
  r=[ones(1,b2) fp fp+1 p*ones(1,fn2-b3)];
  c=[1:b3+1 b2+1:fn2+1];
  v=2*[0.5 ones(1,b2-1) 1-pf+fp pf-fp ones(1,fn2-b3-1) 0.5];
  mn=1;
  mx=fn2+1;
else
  b1=floor(bl(1))+1;
  b4=min(fn2,ceil(bl(4)))-1;
  pf=log((f0+(b1:b4)/n)/(f0+fl))/lr;
```

```
  fp=floor(pf);
  pm=pf-fp;
  k2=b2-b1+1;
  k3=b3-b1+1;
  k4=b4-b1+1;
  r=[fp(k2:k4) 1+fp(1:k3)];
  c=[k2:k4 1:k3];
  v=2*[1-pm(k2:k4) pm(1:k3)];
  mn=b1+1;
  mx=b4+1;
end
if any(w=='n')
  v=1-cos(v*pi/2);
elseif any(w=='m')
  v=1-0.92/1.08*cos(v*pi/2);
end
if nargout > 1
  x=sparse(r,c,v);
else
  x=sparse(r,c+mn-1,v,p,1+fn2);
End
```

Part6：

```
function p = pdf(m, v, x)
%计算多元高斯密度函数
%输入:
%  m -- 均值向量, SIZE*1
%  v -- 方差向量, SIZE*1
%  x -- 输入向量, SIZE*1
%输出:
%  p -- 输出概率


% p = (2 * pi * prod(v)) ^ -0.5 * exp(-0.5 * (x-m) * (x-m)'./ prod(v) );
p = (2 * pi * prod(v)) ^ -0.5 * exp(-0.5 * (x-m) ./ v * (x-m)');
```

Part7：

```
function hmm = inithmm(samples, M)

K = length(samples); %语音样本数
N = length(M);          %状态数
hmm.N = N;
hmm.M = M;
```

```
% 初始概率矩阵
hmm.init    = zeros(N,1);
hmm.init(1) = 1;

% 转移概率矩阵
hmm.trans=zeros(N,N);
for i=1:N-1
    hmm.trans(i,i)   = 0.5;
    hmm.trans(i,i+1) = 0.5;
end
hmm.trans(N,N) = 1;

% 概率密度函数的初始聚类
% 平均分段
for k = 1:K
    T = size(samples(k).data,1);
    samples(k).segment=floor([1:T/N:T T+1]);
end

%对属于每个状态的向量进行 K 均值聚类,得到连续混合正态分布
for i = 1:N
    %把相同聚类和相同状态的向量组合到一个向量中
    vector = [];
    for k = 1:K
        seg1 = samples(k).segment(i);
        seg2 = samples(k).segment(i+1)-1;
        vector = [vector ; samples(k).data(seg1:seg2,:)];
    end
    mix(i) = getmix(vector, M(i));
end

hmm.mix = mix;

function mix = getmix(vector, M)

% kmeans : k-均值算法,求类中心,是通过求类中所有数据点的平均值求得的
%     nn : N*1 的向量，每个样本点所在的类别
%   mean : 所聚类别的中心点坐标位置 k*p，k 是所聚类别

[nn mean] = kmeans(vector,M);

% 计算每个聚类的标准差, 对角阵, 只保存对角线上的元素
for j = 1:M
    ind = find(j==nn);
```

```
    tmp = vector(ind,:);
    var(j,:) = std(tmp);
end

% 计算每个聚类中的元素数, 归一化为各 pdf 的权重
weight = zeros(M,1);
for j = 1:M
    weight(j) = size(find(j==nn),1);
end
weight = weight/sum(weight);

% 保存结果
mix.M      = M;
mix.mean   = mean;      % M*SIZE
mix.var    = var.^2; % M*SIZE
mix.weight = weight; % M*1
```

Part8：

```
function param = getparam(hmm, O)
%给定输出序列 O, 计算前向概率 alpha, 后向概率 beta, 标定系数 c, 及
ksai,gama
%输入:
%  hmm -- HMM 模型参数
%  O   -- n*d 观察序列
%输出:
%  param -- 包含各种参数的结构

T = size(O,1);    %序列的长度

init  = hmm.init;%初始概率
trans = hmm.trans;   %转移概率
mix   = hmm.mix; %高斯混合
N     = hmm.N;       %HMM 状态数

% 给定观察序列 O, 计算前向概率 alpha
alpha = zeros(T,N);

% t=1 的前向概率
x = O(1,:);
for i = 1:N
    alpha(1,i) = init(i) * mixture(mix(i),x);
end

% 标定 t=1 的前向概率
```

```
c    = zeros(T,1);
c(1) = 1/sum(alpha(1,:));
alpha(1,:) = c(1) * alpha(1,:);

% t=2:T 的前向概率和标定
for t = 2:T
    for i = 1:N
        temp = 0;
        for j = 1:N
            temp = temp + alpha(t-1,j) * trans(j,i);
        end
        alpha(t,i) = temp * mixture(mix(i),O(t,:));
    end
    c(t) = 1/sum(alpha(t,:));
    alpha(t,:) = c(t)*alpha(t,:);
end

% 给定观察序列O, 计算后向概率 beta
beta = zeros(T,N);

% t=T 的后向概率及标定
for l = 1:N
    beta(T,l) = c(T);
end

% t=T-1:1 的后向概率和标定
for t = T-1:-1:1
    x = O(t+1,:);
    for i = 1:N
    for j = 1:N
        beta(t,i) = beta(t,i) + beta(t+1,j) * mixture(mix(j),x) *
trans(i,j);
    end
    end
    beta(t,:) = c(t) * beta(t,:);
end

%过渡概率 ksai
ksai = zeros(T-1,N,N);
for t = 1:T-1
    denom = sum(alpha(t,:).*beta(t,:));
    for i = 1:N-1
    for j = i:i+1
        nom = alpha(t,i) * trans(i,j) * mixture(mix(j),O(t+1,:)) *
```

```
beta(t+1,j);
        ksai(t,i,j) = c(t) * nom/denom;
    end
    end
end

%混合输出概率:gama
gama = zeros(T,N,max(hmm.M));
for t = 1:T
    pab = zeros(N,1);
    for l = 1:N
        pab(l) = alpha(t,l) * beta(t,l);
    end
    x = O(t,:);
    for l = 1:N
        prob = zeros(mix(l).M,1);
        for j = 1:mix(l).M
            m = mix(l).mean(j,:);
            v = mix(l).var (j,:);
            prob(j) = mix(l).weight(j) * pdf(m, v, x);
        end
        tmp  = pab(l)/sum(pab);
        for j = 1:mix(l).M
            gama(t,l,j) = tmp * prob(j)/sum(prob);
        end
    end
end

param.c     = c;
param.alpha = alpha;
param.beta  = beta;
param.ksai  = ksai;
param.gama  = gama;

Part9:
function hmm = baum(hmm, samples)

mix  = hmm.mix;          %高斯混合
N    = length(mix);      %HMM 状态数
K    = length(samples); %语音样本数
SIZE = size(samples(1).data,2); %参数阶数

% 计算前向, 后向概率矩阵, 考虑多观察序列和下溢问题
disp('计算样本参数...');
```

```
for k = 1:K
    fprintf('%d ',k)
    param(k) = getparam(hmm, samples(k).data);
end
fprintf('\n')

% 重估转移概率矩阵 A: trans
disp('重估转移概率矩阵 A...')
for i = 1:N-1
    denom = 0;
    for k = 1:K
        tmp   = param(k).ksai(:,i,:);
        denom = denom + sum(tmp(:));
    end
    for j = i:i+1
        nom = 0;
        for k = 1:K
            tmp = param(k).ksai(:,i,j);
            nom = nom   + sum(tmp(:));
        end
        hmm.trans(i,j) = nom / denom;
    end
end

% 重估混合高斯的参数
disp('重估混合高斯的参数...')
for l = 1:N
for j = 1:hmm.M(l)
    fprintf('%d,%d ',l,j)
    % 计算各 pdf 的均值和方差
    nommean = zeros(1,SIZE);
    nomvar  = zeros(1,SIZE);
    denom   = 0;
    for k = 1:K
        T = size(samples(k).data,1);
        for t = 1:T
            x       = samples(k).data(t,:);
            nommean = nommean + param(k).gama(t,l,j) * x;
            nomvar       =    nomvar       +    param(k).gama(t,l,j)    *
(x-mix(l).mean(j,:)).^2;
            denom   = denom   + param(k).gama(t,l,j);
        end
    end
    hmm.mix(l).mean(j,:) = nommean / denom;
```

```
    hmm.mix(l).var (j,:) = nomvar / denom;

    % 计算各 pdf 的权
    nom   = 0;
    denom = 0;
    for k = 1:K
        tmp = param(k).gama(:,l,j);     nom   = nom   + sum(tmp(:));
        tmp = param(k).gama(:,l,:);     denom = denom + sum(tmp(:));
    end
    hmm.mix(l).weight(j) = nom/denom;
end
fprintf('\n')
end
```

Part10：

```
function [prob,q] = viterbi(hmm, O)
%Viterbi 算法
%输入:
%  hmm -- hmm 模型
%  O   -- 输入观察序列, N*D, N 为帧数,D 为向量维数
%输出:
%  prob -- 输出概率
%  q    -- 状态序列

init  = hmm.init;%初始概率
trans = hmm.trans;   %转移概率
mix   = hmm.mix; %高斯混合
N     = hmm.N;       %HMM 状态数
T     = size(O,1);   %语音帧数

% 计算 log(init);
ind1  = find(init>0);
ind0  = find(init<=0);
init(ind0) = -inf;
init(ind1) = log(init(ind1));

% 计算 log(trans);
ind1 = find(trans>0);
ind0 = find(trans<=0);
trans(ind0) = -inf;
trans(ind1) = log(trans(ind1));

% 初始化
delta = zeros(T,N);
```

```
fai   = zeros(T,N);
q     = zeros(T,1);

% t=1
x = O(1,:);
for i = 1:N
    delta(1,i) = init(i) + log(mixture(mix(i),x));
end

% t=2:T
for t = 2:T
for j = 1:N
    [delta(t,j) fai(t,j)] = max(delta(t-1,:) + trans(:,j)');
    x = O(t,:);
    delta(t,j) = delta(t,j) + log(mixture(mix(j),x));
end
end

% 最终概率和最后节点
[prob q(T)] = max(delta(T,:));

% 回溯最佳状态路径
for t=T-1:-1:1
    q(t) = fai(t+1,q(t+1));
end

Part11：
function [hmm, pout] = train(samples, M)
%输入:
%  samples -- 样本结构
%  M       -- 为每个状态指定 pdf 个数,如:[3 3 3 3]
%输出:
%  hmm      -- 训练完成后的 hmm

K   = length(samples);

% 计算语音参数
disp('正在计算语音参数');
for k = 1:K
    if isfield(samples(k),'data') & ~isempty(samples(k).data)
        continue;
    else
        samples(k).data = mfcc(samples(k).wave);
    end
```

```
end

hmm = inithmm(samples, M);

for loop = 1:40
    fprintf('\n 第%d 遍训练\n\n',loop)
    hmm = baum(hmm, samples);

    %计算总输出概率
    pout(loop)=0;
    for k = 1:K
        pout(loop) = pout(loop) + viterbi(hmm, samples(k).data);
    end

    fprintf('总和输出概率(log)=%d\n', pout(loop))

    %比较两个 HMM 的距离
    if loop>1
        if abs((pout(loop)-pout(loop-1))/pout(loop)) < 5e-6
            fprintf('收敛!\n');
            return
        end
    end
end

disp('迭代 40 次仍不收敛, 退出');
```

Part12：

```
% 导入训练样本
load('samples.mat')

for i=1:length(samples)
    sample=[];
    for k=1:length(samples{i})
        sample(k).wave=samples{i}{k};
        sample(k).data=[];
    end
    hmm{i}=train(sample,[3 3 3 3]);
end
```

Part13：

```
for i=1:20
    fname = sprintf('ch6\\%d.wav',i-1);
    x = audioread(fname);
```

```
    [x1 x2] = vad(x);
    m = mfcc(x);
    m = m(x1-2:x2-2,:);
    for j=1:10
        pout(j) = viterbi(hmm{j}, m);
    end
    [d,n] = max(pout);
    if n ==10
        n = n-10 ;
    end
    fprintf('第%d 个词, 识别为%d\n', i,n)
end


%SVM 分类器
clear all;
clc;
train_set =xlsread('character.xlsx','sheet1','A1:F120');
train_set_labels =xlsread('character.xlsx','sheet1','F1:F120');
test_set =xlsread('character.xlsx','sheet1','A121:F140');
test_set_labels =xlsread('character.xlsx','sheet1','F121:F140');
% 数据预处理,将训练集和测试集归一化到[0,1]区间
[mtrain,ntrain] = size(train_set);
[mtest,ntest] = size(test_set);
test_dataset = [train_set;test_set];
% mapminmax 为 MATLAB 自带的归一化函数
[dataset_scale,ps] = mapminmax(test_dataset',0,1);
dataset_scale = dataset_scale';
train_set = dataset_scale(1:mtrain,:);
test_set = dataset_scale( (mtrain+1):(mtrain+mtest),: );
 %% SVM 网络训练
model = svmtrain(train_set_labels, train_set, '-s 2 -c 1 -g 0.07');
%% SVM 网络预测
[predict_label] = svmpredict(test_set_labels, test_set, model);
%% 结果分析
% 测试集的实际分类和预测分类图
figure;
hold on;
plot(test_set_labels,'o');
plot(predict_label,'r*');
xlabel('测试集样本','FontSize',12);
ylabel('类别标签','FontSize',12);
legend('实际测试集分类','预测测试集分类');
title('测试集的实际分类和预测分类图','FontSize',12);
```

```
grid on;
```