HTK（V3.1）基础指南

（原文：HTK (v.3.1): Basic Tutorial）

Nicolas Moreau / 02.02.2002

陶阳　译

taoyangxp@163.com

江西.南昌

2009.6.1

目　录

0 HTK 简介

HTK 是指隐马尔可夫模型工具箱（Hidden Markov Model Toolkit），由剑桥大学工程系（CUED）研发而成。该工具箱的目的是搭建使用隐马尔可夫模型（HMMs）。HTK 主要用于语音识别研究（但是 HMMs 应用范围很广，还有很多其它可能的应用…）

HTK 由一系列库模块构成，包括 C 语言形式的可用工具，可自由下载，包括一个完整的文档说明（大约 300 页），见 http://htk.eng.cam.ac.uk/。

1 Yes/No 识别系统

本指南中，我们将基于 HTK 工具集建立一个 2-单词识别系统，词汇集是{ Yes ， No }。这是可以设计出来的最基本的自动语音识别（ASR）系统。

1.1 搭建步骤

构建语音识别系统的主要步骤如下：

（1）训练库的创建：词汇集中的每个元素进行多次录制，且与相应词汇做好标签；

（2）声学分析：训练波形数据转换为一系列系数向量；

（3）模型定义：为总词汇集中的每个元素定义一个 HMM 原型；

（4）模型训练：使用训练数据对每个 HMM 模型进行初始化、训练；

（5）任务定义：识别系统的语法（什么可被识别）的定义；

（6）未知输入信号识别；

（7）评估：识别系统的性能可通过测试数据进行评估。

1.2 工作环境构建

建议创建如下的目录结构：

（1）data/ ：存储训练和测试数据（语音信号、标签等等），包括 2 个子目录，data/train/ 和 data/test/，用来区分识别系统的训练数据和评估数据；

（2）analysis/：存储声学分析步骤的文件；

（3）training/：存储初始化和训练步骤的相关文件；

（4）model/：存储识别系统的模型（HMMs）的相关文件；

（5）def/：存储任务定义的相关文件；

（6）test/：存储测试相关文件。

1.3 标准 HTK工具选项

一些标准选项对每个 HTK 工具都是通用的。我们将使用以下一些选项：

（1）-A ：显示命令行参数；

（2）-D ：显示配置设置；

（3）-T 1：显示算法动作的相关信息。

完整的选项列表请参见：HTK 文档，第 50 页（第四章 操作环境）。

2 创建训练集

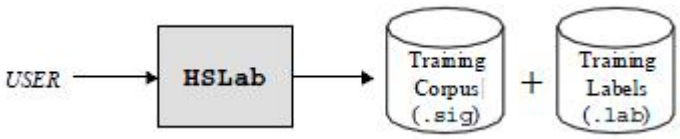

图 1 录制标签训练数据

首先，我们录制 Yes 和 No 两个语音信号，作为要训练的单词模型（训练集）。然后为每个语音信号打上标签，也就是说，关联一个文本来描述语音内容。录制和打标签，

可以使用 HSLab HTK 工具来做（也可使用其它工具）。

创建和标注语音文件，使用下面的工具命令：

HSLab any_name.sig

回车之后，该工具的图形界面呈现在面前。

2.1 录音

按下 Rec 按钮，开始录制语音信号， Stop 停止录音。

缓冲文件 any_name_0.sig 自动创建在当前目录下(如果录制一个新的文件，第二个缓冲文件命名为 any_name_1.sig)。

说明：

-这里的信号文件（.sig）以一种特定的 HTK 格式存在。当然，也可能使用其它音频格式（如.wav 等），参见 HTK 文档第 68 页（第五章 音频输入/输出）。

-缺省采样率是 16kHz。

2.2 标注信号

要为语音波形打标签，首先按下 Mark 按钮，然后选择你要打标签的区域。当区域标注之后，按下 Labelas ，键入标签名称，然后按下回车 Enter。

本指南中，我们仅录制孤立词（ Yes 或者 No ），通过短暂停顿隔开。对于每个信号，我们标注三个连续的区域：开始停顿（标记为 sil）、录音单词（标记为 yes 或 no）、结束停顿（标记为 sil）。这三个区域不能重叠（即使它们之间间隙很小）。这三个标注完成之后，按下 Save 按钮：标签文件 any_name_0.lab 被创建完成。这个时候，你就可以按下 Quit 退出按钮了。

说明：

.lab 文件是一个简单的文本文件。它包括每个标注的类型信息，每行对应每个标签：

4171250 9229375 sil

9229375 15043750 yes

15043750 20430625 sil

其中数字代表每个标签的开始和结束采样点。这样的文件可以进行手工修改（比如调整标签的开始或结束点），甚至创建（并不一定使用 HSLab 工具）。

2.3 文件重命名

录音/标签之后，你可以依据自己的方便，对.sig 和.lab 文件进行重命名（如 yes01.sig 和 yes01.lab）。

本指南中，两个词的每个进行 10 个录音应该足够了。

信号文件存储在 data/train/sig/目录（训练集）里面，标签存储在 data/train/lab/目录（训练标签集）。

关于 HSLab 图形界面的更多详细信息参见 HTK 文档，第 237 页（参考小节，HSLab）。

3 声学分析

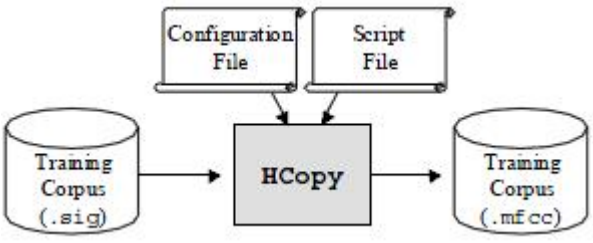

图 2 训练数据转化

语音识别工具不能直接处理波形语音，需要通过更简洁有效的方法来表示波形语音。这一步就是 声学分析 ：

-信号分成连续的帧，一般每帧长度介于 20ms 和 40ms 之间，段与段交织在一起；

-每帧与窗口函数相乘（如 Hamming 函数）；

-从每个窗口帧中提取声学系数向量（给出一种该帧的频谱属性的简洁表示）。

HCopy HTK 工具可用来转换原始波形文件，生成一系列声学向量，命令为：

HCopy –A –D –C analysis.conf –S targetlist.txt

analysis.conf 是配置文件，设置声学系数提取参数。

targetlist.txt 用于指定用于处理的每个波形文件的名称和存放位置，以及目标系数文件的名称和存放位置。

3.1 配置参数

配置文件是一个文本文件（ # 用来注释）。本指南中，使用的配置文件如下：

```
#
# Example of an acoustical analysis configuration file
#
SOURCEFORMAT = HTK              # Gives the format of the speech files
TARGETKIND = MFCC_0_D_A         # Identifier of the coefficients to use
# Unit = 0.1 micro-second :
WINDOWSIZE = 250000.0           # = 25 ms = length of a time frame
TARGETRATE = 100000.0           # = 10 ms = frame periodicity
NUMCEPS = 12                    # Number of MFCC coeffs (here from c1 to c12)
USEHAMMING = T                  # Use of Hamming function for windowing frames
PREEMCOEF = 0.97                # Pre-emphasis coefficient
NUMCHANS = 26                   # Number of filterbank channels
CEPLIFTER = 22                  # Length of cepstral liftering
# The End
```

列表 1 分析配置文件

从该配置文件可看出，使用了 MFCC（(Mel Frequency Cepstral Coefficient)分析，在 TARGETKIND 标识符的值的前缀 “MFCC”表明了这一点。对每个信号帧，可提取以下系数：

-12 个首 MFCC 系数[C1,…,C12]（因为 NUMCEPS = 12）

- null MFCC 系数 c0，与帧的总能量成正比（在 TARGETKIND 中，后缀 _0 ）

-13 个 Delta coefficients ，估计首次序[c0, c1,…, c12]派生（在 TARGETKIND 中，后缀 _D ）

-13 个 Acceleration coefficients ，估计第二次序[c0, c1,…, c12]派生（在 TARGETKIND 中，后缀 _A ）

总之，每个信号帧提取 39 个系数向量。

关于声学分析配置更详细内容参加 HTK 文档第 58-66 页（第五章 语音输入/输出）

3.2 源 /目标规范

一个或多个 源/目标文件 对（如，原始波形文件/系数文件），可在 HCopy 命令行中直接指定。如果要处理大量数据，可使用-S 选项。一般允许指定脚本文件格式为：

```
data/train/sig/yes01.sig data/train/mfcc/yes01.mfcc
```

```
data/train/sig/yes02.sig data/train/mfcc/yes02.mfcc
etc...
data/train/sig/no01.sig data/train/mfcc/no01.mfcc
data/train/sig/no02.sig data/train/mfcc/no02.mfcc
etc...
```

列表 2 转换脚本文件

可自动生成这样的文本脚本文件(比如使用 Perl 脚本编辑工具)。新生成的训练集(.mfcc 文件)存储在 data/trail/mfcc/目录下。

更多关于 HCopy 工具的详细信息参见 HTK 文档第 195 页(参考小节 HCopy)

4 HMM 定义

本指南中,3个声学事件需要使用HMM隐马尔可夫模型建模,即 Yes 、No 、Silence 。每个事件设计一个 HMM 模型。

第一步,为每个 HMM 模型选择一个 priori 结构:

-状态数

-观察函数的形式(对应每个状态)

-状态转换排列

这种定义不是一成不变的。其实并没有固定的规则。

在这里,我们为 3 个 HMM 模型选择同一个结构(如图 3 所示)

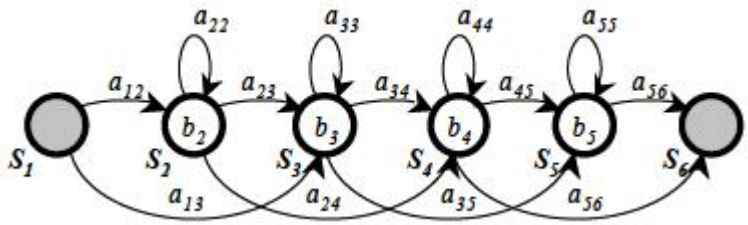

图 3 基本结构

该模型包含 4 个 活动 状态{S_2,S_3,S_4,S_5},开始和结束状态(这里是 S_1、S_6),是 非发散 状态(无观察函数),仅供 HTK 用于一些功能的实现。观察函数 b_i 是带对角矩阵的高斯分布。状态的可能转换由 a_{ij} 表示。

在 HTK 中,HMM 模型是通过文本文件来描述的。如图 3 一样的 HMM 模型一般描述如下:

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "yes"
<BeginHMM>
<NumStates> 6
<State> 2
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

```
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<State> 3
<Mean> 39
0.0 0.0 (...) 0.0
<Variance> 39
1.0 1.0 (...) 1.0
<State> 4
<Mean> 39
0.0 0.0 (...) 0.0
<Variance> 39
1.0 1.0 (...) 1.0
<State> 5
<Mean> 39
0.0 0.0 (...) 0.0
<Variance> 39
1.0 1.0 (...) 1.0
<TransP> 6
0.0 0.5 0.5 0.0 0.0 0.0
0.0 0.4 0.3 0.3 0.0 0.0
0.0 0.0 0.4 0.3 0.3 0.0
0.0 0.0 0.0 0.4 0.3 0.3
0.0 0.0 0.0 0.0 0.5 0.5
0.0 0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

列表 3 HMM 描述文件（原型）

~o <VecSize> 39 <MFCC_0_D_A>

文件头，给出系数向量大小（这里是 39 个系数）、系数类型（MFCC_0_D_A）。

~h "yes" <BeginHMM> (...)<EndHMM>

封装对所谓的 yes 的 HMM 模型的描述。

<NumStates> 6

给出 HMM 模型的状态总数，包括 2 个非发散状态 1 和 6。

<State> 2

表示对状态 2 的观察函数的描述。这里我们使用单一高斯观察函数，带有对角矩阵。这样的函数完全由一个平均向量和一个变化向量（自相关矩阵的对角元素）。状态 1 和 6 没有描述，他们没有观察函数。

<Mean> 39

0.0 0.0 (...) 0.0 (x 39)

给出当前观察函数的平均向量（在 39 维的观察空间中）。每个元素是强制初始化为 0，该文件仅给出 HMM 模型（它的全局结构）的原型。这些系数后面将用来训练。

<Variance> 39

1.0 1.0 (...) 1.0 (x 39)

给出当前观察函数的变化向量。每个元素强制初始化为 1.

<TransP> 6

给出 HMM 模型的 6x6 转换矩阵，即：

$$a_{11}\ a_{12}\ a_{13}\ a_{14}\ a_{15}\ a_{16}$$

$$a_{21}\ a_{22}\ a_{23}\ a_{24}\ a_{25}\ a_{26}$$

$$a_{31}\ a_{32}\ a_{33}\ a_{34}\ a_{35}\ a_{36}$$

$$a_{41}\ a_{42}\ a_{43}\ a_{44}\ a_{45}\ a_{46}$$

$$a_{51}\ a_{52}\ a_{53}\ a_{54}\ a_{55}\ a_{56}$$

$$a_{61}\ a_{62}\ a_{63}\ a_{64}\ a_{65}\ a_{66}$$

其中 aij 表示状态 i 到 j 转换的可能性。Null 值说明相应的转换不允许。其它值进行强制初始化（但是矩阵的每行之和为 1）：在训练过程中将被修改。

我们必须为每个模型生成一个这样的原型。

在我们的例子中，我们要写 3 个 HMM 模型原型，即 yes 、 no 、 sil （3 个描述文件的头部分别是~h "yes"、 ~h "no" 、 ~h "sil"）。

这 3 个文件可以命名为 hmm_yes、hmm_no、hmm_sil，存储在 model/proto 目录下。

关于 HMM 描述文件更多详细信息参加 HTK 文档第 94 页（第七章 HMM 定义文件）。

5 HMM 训练

训练过程描述如图 4 所示。

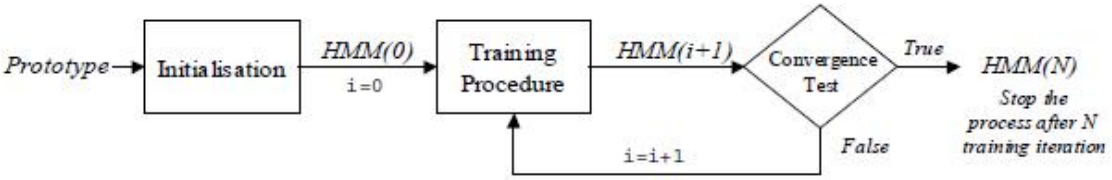

图 4 完整的训练过程

5.1 初始化

在训练过程开始之前，为了使得训练算法快速精准收敛，HMM 模型参数必须根据训练数据正确初始化。

HTK 提供了 2 个不同的初始化工具：Hinit 和 HCompv。

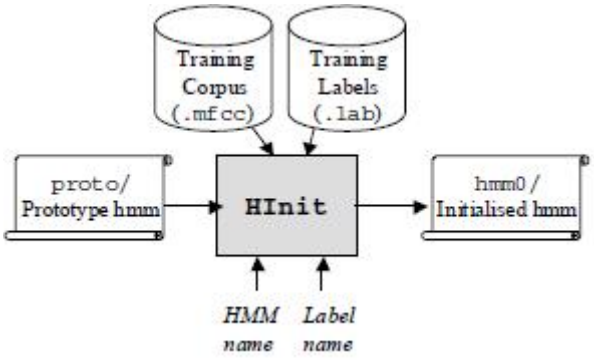

图 5 原型初始化

HInit

下面的命令行是使用 Viterbi 算法通过训练数据的时间轴对 HMM 模型进行初始化：

```
HInit -A -D –T 1 -S trainlist.txt -M model/hmm0 \
-H model/proto/hmmfile -l label -L label_dir nameofhmm
```

nameofhmm 是要进行初始化的 HMM 模型的名称（这里指 yes、 no、sil）

hmmfile 是一个描述文件，包含被称作 nameofhmm 的 HMM 模型的原型（这里指

proto/hmm_yes，proto/hmm_no， proto/hmm_sil)。

trainlist.txt 给出完整的 .mfcc 文件列表，建立训练集(存储在目录 data/train/mfcc/)。

label_dir 是存储相关训练集的标签文件（.lab）的目录（这里是 data/train/lab/）。

label 用来标示哪个标签段用于训练集中（这里指 yes、no 或 sil，因为它们对于标签和 HMM 模型使用了相同的名字，当然这不是强制的。

model/hmm0 是初始化 HMM 模型描述结果输出的目录名称（必须提前创建好）。

这个过程对每个模型(hmm_yes, hmm_no, hmm_sil)重复执行。

说明：

由 HInit 输出的 HMM 模型文件和输入原型具有相同的名字。

HCompv

HCompv 工具用来对模型进行 平坦 初始化。HMM 模型的每个状态给定相同的平均向量和变化向量：在整个训练集上全局计算而得。一般初始化命令行形式如下：

**HCompv -A -D –T 1 -S trainlist.txt -M model/hmm0flat **

-H model/proto/hmmfile -f 0.01 nameofhmm

nameofhmm , **hmmfile** , **trainlist.txt**： 见 HInit.中的解释。

model/hmm0flat：输出目录，与 HInit 必须保持不同，避免覆盖。

说明：

label 选项也可以使用。这种情况下，对全局平均和变化向量的估值就仅基于训练集的相应的标签部分。

我们不使用 HCompv 来初始化我们的模型（已经使用了 HInit）。

但是 HCompv 也与初始化模型一起输出一个名称为 vFloors 的有趣的文件，它包含乘以因子的全局变化向量(见列表 4)。这个因子可通过-f 选项来设置(这里是 0.01)。

```
~v varFloor1
<Variance> 39
5.196781e-001 2.138549e-001 (...) 3.203219e-003
```

列表 4 可变基底宏文件，vFloors

存储在 varFloor1 中的值（称作 可变基底宏 ）在后面训练过程中用作估计的变化向量的基底。

在训练迭代过程中，与特定 HMM 模型状态相关联的训练帧数可能很低。该状态的估计变化值会很小（如果只有一个可用的训练帧的话，变化甚至为空 null）。这种情况下，可用基底来替代，避免变化值趋于极小（甚至产生计算错误）。

这里，我们将对 HMM 原型之一使用 HCompv 一次，来计算上述 varFloor1 宏。相应的 vFloors 文件输出到目录 model/hmm0flat/。

5.2 训练

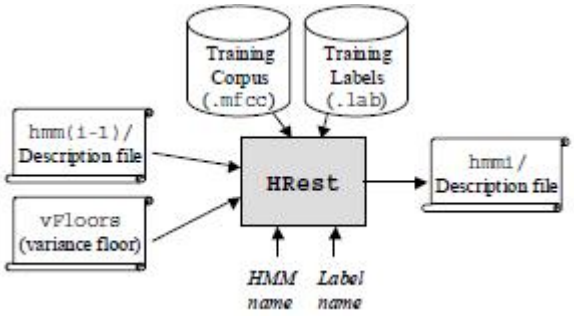

图 6 一次再估计迭代

下面的命令行可实现 HTK 工具 HRest 的一次再估计迭代，估计 HMM 模型参数（每个观察函数的变换可能性、加平均、变化向量）的最佳值：

**HRest -A -D -T 1 -S trainlist.txt -M model/hmm*i* -H vFloors **

-H model/hmm*i-1*/hmmfile -l label -L label_dir nameofhmm

nameofhmm 是要训练的 HMM 模型的名称（这里是 yes,no,sil）。

hmmfile 是名为 nameofhmm 的 HMM 模型的描述文件，存储在表示最近一次迭代的索引为名字的目录下（如 model/hmmi-1/）

vFloors 是一个文件，包含由 HCompv 生成的变化基宏。

trainlist.txt 给出构成训练集的.mfcc 文件的完整列表（存储在 data/train/mfcc/）。

lab_dir 是存放对应训练集的标签文件（.lab）的目录（这里是 data/train/lab/）。

label 指示训练数据中使用的标签（yes,no,sil）

model/hmmi，输出目录，标示当前迭代 i 的索引。

对于每个要训练的 HMM 模型，这个过程要重复许多次。

每次，HRest 迭代（即当前再估计迭代中的迭代）显示在屏幕上，通过 change 量度标示收敛性。一旦这个量度值不再从一个 HRest 迭代到下个迭代减少（绝对值），过程就该停止了。在我们的例子中，2 或 3 次再估计迭代就足够了。

最终单词 HMM 模型是 hmm3/hmm_yes, hmm3/hmm_no, and hmm3/hmm_sil.

6 任务定义

与任务相关的每个文件都应该存储在专用的 def/目录。

6.1语法和字典

在使用我们的单词模型之前，要定义识别器的基本结构（任务语法）。我们首先定义最简单的语法：开始停顿、接着简单单词（这里指 Yes，No）、结束停顿。

在 HTK 中，根据一些句子造句法规则，任务语法写成一个文本文件。在我们的这个例子中，造句语法很简单：

```
/*
* Task grammar
*/
$WORD = YES | NO;
( { START_SIL } [ $WORD ] { END_SIL } )
```

列表 5 基本的任务语法

Word 变量可以被 yes 或 no 替换。

用括号{}括住 START_SIL 和 END_SIL 表示其可不存在或者重复多次（允许在单词之前或之后长时间的停顿，或者根本没有停顿）。

括号[]括住$WORD 表示零个或一次出现（如果没有单词，可能只是识别停顿）。

关于 HTK 构造语法的更多详细信息参见 HTK 文档第 163 页（第 12 章 网络、字典和语言模型）

当然，系统要知道 HMM 模型与语法变量 YES、NO、START_SIL 和 END_SIL 的对应关系。这种信息存储在文本文件中，命名为任务字典。在这样的简单任务中，对应关系是直截了当的，一个任务字典一般包括 4 个入口：

```
YES [yes] yes
NO [no] no
START_SIL [sil] sil
END_SIL [sil] sil
```

列表 6 任务字典

左边的元素指任务语法变量的名字，右边的元素指 HMM 模型（在 HMM 模型定义文件中通过-h 选项引入的）的名字。中间带括号的元素是可选的，它们表示识别器要输出的符号：这里使用了标签的名称（缺省情况下，使用的是语法变量名称）。

说明：不要忘记在文件的末尾添加新行(如果不添加,那么最后一个入口将被忽略)。

6.2 网络

任务语法（在 gram.txt 中描述）必须使用 HParse 工具进行编译，生成任务网络（写入 net.slf）：

```
HParse -A -D -T 1 gram.txt net.slf
```

至此，我们的语音识别任务（如图 7）完成了，其完全由其网络、它的字典、它的 HMM 模型集（存储在 model/hmm3/）来定义。

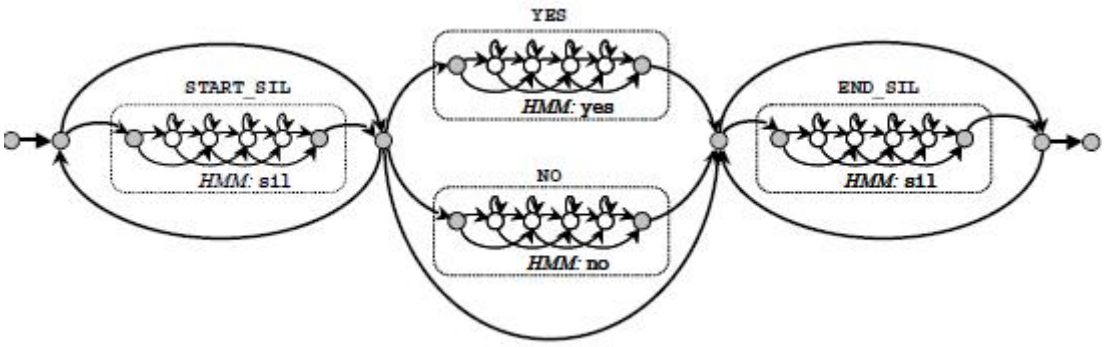

图 7 识别器=网络+字典+HMM 模型

说明：在写语法时要确保没有错误，可使用 HSGen 工具来测试

```
HSGen -A -D -n 10 -s net.slf dict.txt
```

其中 dict.txt是任务字典，选项 -n表示 10个符合语法的句子（即 10个可能的识别结果）将被随机生成且显示。

当然，在这里这没什么重要用途，但是当语法很复杂的时候，就会很有用了。

7 识别

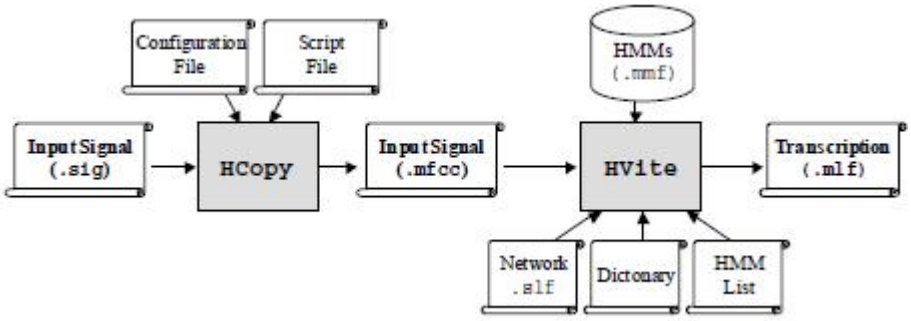

图 8 未知输入信号的识别过程

来看看识别过程本身：

-语音输入信号 input.sig首先使用 HCopy工具转换成一系列声学向量（MFCC），如同转换训练数据一样（见声学分析步骤）。转换结果存储在 input.mfcc文件中（常称作声学观察）。

-然后使用 Viterbi算法处理输入观察，该算法用来测试输入观察是否与识别器的马尔可夫模型相一致。可以使用 HVite工具达到这个目的：

**HVite -A -D -T 1 -H hmmsdef.mmf -i reco.mlf -w net.slf **

dict.txt hmmlist.txt input.mfcc

input.mfcc是要被识别的输入数据。

hmmlist.txt列出要使用的模型的名称（yes,no,sil）。每个元素通过新行字符隔开。不要忘记在最后一个元素后面插入新行。

dict.txt是任务字典。

net.slf是任务网络。

reco.mlf是输出识别副本文件。

hmmsdef.mmf包含 HMM模型的定义。在我们这个例子中可能要重复 -H选项，列出不同的 HMM模型定义文件：

```
-H model/hmm3/hmm_yes -H model/hmm3/hmm_no -H model/hmm3/hmm_sil
```

但还是把每个定义集中到一个文件（称为主宏文件，扩展名.mmf）比较方便一些（尤其当存在 3 个以上模型时）。简单地，可通过复制每个定义到一个文件中得到这样的文件，不用重复复制文件的头部信息（见列表 7）

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "yes"
<BeginHMM>
        (definition...)
<EndHMM>
~h "no"
<BeginHMM>
        (definition...)
<EndHMM>
~h "sil"
<BeginHMM>
        (definition...)
```

```
<EndHMM>
```

列表 7 主宏文件（在一个文件中有许多 HMM 模型描述）

输出结果存储在名称为 reco.mlf 的文件中，包含对输入信息的副本。比如，如果我们使用文件 data/train/mfcc/yes01.mfcc 作为输入数据，我们得到的输出文件 reco.mlf 如：

```
#!MLF!#
  "../data/train/mfcc/yes01.rec"
  0 4900000 SIL -2394.001465
  4900000 12000000 YES -5159.434570
  12000000 18300000 SIL -3289.197021
.
```

列表 8 识别结果输出（识别副本）

在这个例子中，从输入信号中识别出 3 个连续的 单词假设 。目标单词 Yes 被正确识别出来。每个假设的开始和结束点也给了出来，同时还有它们的声谱，这可由 Viterbi 解码算法得出结果（列表上右列）。

测试识别系统的更方便交互的方式是使用 Hvite 的 直接输入 选项：

**HVite -A -D -T 1 -C directin.conf -g -H hmmsdef.mmf **
-w net.slf dict.txt hmmlist.txt

这种情况下，不需要输入信号参数，也无输出文件。提示符 READY[1]>出现在屏幕上，表示开始录音第一个输入信号，按下任何键可停止录音。然后显示识别结果，接着显示 READY[2]>提示符等待下一个输入信号。

-g 选项允许每个输入信号回放一次。

directin.conf 是 Hvite 的配置文件，允许使用直接音频输入：

```
  #
  # HVite Configuration Variables for DIRECT AUDIO INPUT
  #
  # Parameters of the input signal
  SOURCERATE = 625.0 # = 16 kHz
  SOURCEKIND = HAUDIO
    SOURCEFORMAT = HTK
  # Conversion parameters of the input signal
  TARGETKIND = MFCC_0_D_A # Identifier of the coefficients to use
  WINDOWSIZE = 250000.0 # = 25 ms = length of a time frame
  TARGETRATE = 100000.0 # = 10 ms = frame periodicity
  NUMCEPS = 12 # Number of MFCC coeffs (here from c1 to c12)
  USEHAMMING = T # Use of Hamming function for windowing frames
  PREEMCOEF = 0.97 # Pre-emphasis coefficient
  NUMCHANS = 26 # Number of filterbank channels
  CEPLIFTER = 22 # Length of cepstral liftering
  # Defines the signal to be used for remote control
  AUDIOSIG = -1 # Negative value = key-press control
# The End
```

列表 9 直接语音输入识别的配置文件

为了允许从输入信号中直接提取声学系数，这个文件必须包含前面训练数据使用的

声学分析配置参数。

8 性能测试

ASR 系统的识别性能评估必须使用不同于训练数据集的数据进行测试。单独的测试集包括新的 Yes 和 No 的录音数据，可创建在目录 data/test/下面，正如前面处理训练集一样。同样，这些数据（存储在子目录 test/sig）需要打上标签（存储在 test/lab），进行变换（存储在 test/mfcc）。

（如果你对这种新的要求高的录音和打标签内容的看法不以为然，也可以在这里使用训练数据作为测试数据集，那么本指南的目的等于只教你如何使用一些 HTK 工具了，并没有达到相应的性能测试的目的。）

8.1 主标签文件

在性能评测之前，我们需要创建两个文件，命名为主标签文件，扩展名是.mlf：

-第一个文件包含整个训练集的 正确 副本，即是通过手工标注的副本。把 ref.mlf 记作参考副本。

-第二个文件包含整个测试集的识别副本，即识别器产生的假设副本。把 rec.mlf 记为识别副本。

通过比较每项数据的参考副本和识别假设，进行性能评测。

主标签文件具有下面的结构：

```
#!MLF!#
"path/data01.ext"
Label1
Label2
.
"path/data02.ext"
Label1
Label2
Label3
Label4
.
(ETC...)
```

列表 10 主标签文件（一个文件中包含许多副本）

每个副本通过文件名称引入，以 . 号作为结束。副本由一系列标签组成，标签独立占用一行。每个标签可在其前面加上开始和结束时间索引，且/或在后面加上识别分值（见列表 8）。

没有 HTK 工具用来创建 ref.mlf 文件，必须手工来写，或者用脚本写（比如，参见我曾经写过的 Perl 脚本 MakeRefMLF.pl）。

每个标签文件（如 yes01.lab）的内容必须相继拷贝到文件 ref.mlf 中，每个标签文件介于文件名称（占一行，如 */yes01.lab ：这里的陆军可使用*号代替）和结束符号之间。

识别副本文件 rec.mlf 可通过 Hvite 工具直接得到。这次，Hvite 工具不只有一个参数（一个简单输入文件名称），还有整个测试集的文件名（.mfcc），这些名字在一个文本文件中列出：

```
HVite -A -D -T 1 -S testlist.txt -H hmmsdef.mmf -i rec.mlf \
-w net.slf dict.txt hmmlist.txt
```

hmmlist.txt, **dict.txt**, **net.slf**, **hmmsdef.mmf**: 同前面提到的。

rec.mlf是识别副本输出文件。

testlist.txt 测试文件名称列表(data/test/*.mfcc)。

命令执行之后，rec.mlf 就包含一系列副本，正如列表 8 中列出的那个。每个副本由相应的文件名称开始，该文件的扩展名不同（.rec 替代.lab）。

8.2 错误率

使用 HTK 性能评估工具 HResults 比较 ref.mlf 和 rec.mlf 副本：

```
HResults -A -D -T 1 -e ??? sil -I ref.mlf \
labellist.txt rec.mlf > results.txt
```

results.txt 包含性能统计结果输出 (如 List.11).

rec.mlf 包含测试数据的副本，作为识别器的输出。

labellist.txt 出现在副本文件中的标签列表(这里指： yes, no和sil).

ref.mlf 包含测试数据的参考副本(手工打标签获得的测试数据)。

-a ??? sil 这个选项表明，当计算性能统计结果的时候，sil标签可以被忽略，因为我们仅关注单词 Yes 和 No 的识别率。

```
====================== HTK Results Analysis
=======================
  Date: Tue Dec 03 19:12:58 2002
  Ref : .\ref.mlf
  Rec : .\rec.mlf
  ------------------------ Overall Results
--------------------------
  SENT: %Correct=80.00 [H=8, S=2, N=10]
  WORD: %Corr=80.00, Acc=80.00 [H=8, D=0, S=2, I=0, N=10]
```

列表11 性能评测结果

列表 11 给出的是一个由性能评测获得的结果的例子。第一行（SENT）给出句子的识别率（%Correct=80.00），第二行（WORD）给出的是单词的识别率（%Corr=80.00）。

在我们的例子中，这两个比率是相同的，这是因为我们的任务语法仅使用一个单词（除了停顿之外）作为句子。这是孤立词识别任务。这里只要考虑第一行（SENT）就够了。H=8 给出的是测试数据被正确识别的数量，S=2，表示识别相反的数量（Yes 识别成了 No，或者 No 识别成了 Yes），N=0 表示测试数据总数。

第二行（WORD）给出的统计数据只有对更复杂的识别系统才考虑（如连续语音识别任务）。更详细信息参见 HTK 文档第 232 页（参考小节，Hresults）。

最后，译者的话：原文请参见 HTK (v.3.1): Basic Tutorial。由于译者水平有限，有不当之处请批评指正！欢迎来信指教！Email：taoyangxp@163.com QQ：381333215