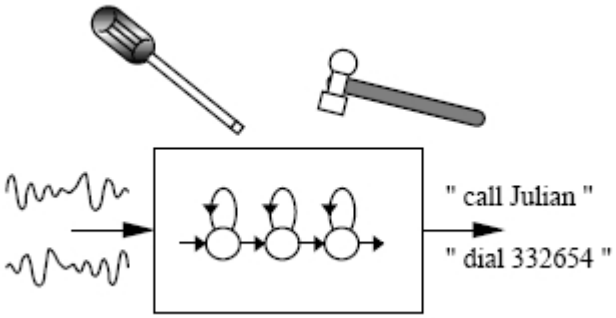
第三章　一个例子——使用 HTK 指南



本章作为指南部分的最后一章，将描述构造一个简单的用于语音拨号的识别器的过程。这个识别器可以识别连续的数字串和有限个名字。由于这是一个子词[译者注：并非为整个单词建立模型，而是词的一部分建立模型，可理解成对音素建模](sub-word)识别系统，所以向词典中加入一个新的名字只需要修改发音词典和语法网络（task grammar）。HMM模型使用的是连续密度混合高斯模型，且使用决策树聚类的状态绑定的triphone。虽然这个语音拨号系统本身相当简单，但是系统的设计具有代表性，对学习HTK的使用很有帮助。

我们将从无到有构建此系统，包括使用HTK工具HSLab录制训练数据和测试数据。为了简化，这个系统是说话人相关的，但是建造一个说话人无关的系统可以采用相同的设计方法。唯一的不同在于需要足够数量的不同说话人的数据，以及相应地增加模型复杂度。

构造一个“全新”的识别器包括很多相关的子任务，而且没有一个明确的实现这些任务的最佳顺序。这里就按时间顺序来介绍，构造类似系统时可以参考本书介绍的步骤。本书剩下的部分相当详细地描述了完整的过程，并清晰地给出HTK工具集所涉及的范围。

HTK的发行软件中也包含了一个用于ARPA海军资源管理任务的1000个词的识别系统的例子，放在HTK软件包的目录RMHTK下。在HTKDemo目录下有更多体现HTK性能和功能的实例。另外还可以在HTKTutorial目录下找到对学习指南部分有帮助的一些例子的脚本。

这一章的指南里介绍的每个步骤，假定使用者在执行命令之前完全了解了所有部分，并对于每个 HTK 工具都可以参考第 17 章[1]（参考章节），所以所有的命令行参数和选项都应该是容易理解的。

3.1 数据准备

任何构造识别器的工程的第一步都是数据准备。训练数据和测试数据都需要准备。在我们需要建造的系统里，所有的语音都需要由最原始的录音得到，而且需要对每句录音进行标注。测试数据的重要性在于，可以通过测试数据度量识别器的性能，选择测试语料的一种简单的办法是由目标语法随机生成。对训练数据来说，语料内容的标注和发音词典一起用于完成初始音素层的标注，而这个标注是开启HMM训练过程所必须的。因为实际可能要将任意名字加入识别器，所以训练数据应该尽量提高音素覆盖率和均衡性[译者注：训练数据各方面（比如音素覆盖、语法结构等）的均衡性对HTK训练极其重要，对识别来说，会提高识别器的鲁棒性和识别性能，对合成来说会提高合成语音的自然度]。这里为了方便，我们用于训练的使用的提示脚本是从TIMIT acoustic-phonetic数据库获取的。

根据上面的描述，在录数据之前，必须先定义音素集合、覆盖训练数据和测试数据的词典和目标语法。

---

[1]指南的最后一部分涉及说话人相关系统中加入新的说话人的自适应方法

3.1.1 第一步 —— 目标语法

为电话拨号创建一个语音操作接口，是我们系统的最终目标。因此，识别器必须要能处理数字串和人名列表。典型输入可能是这样的：

Dial three three two six five four

Dial nine zero four one oh nine

Phone Woodland

Call Steve Young

HTK 规定了一个语法定义语言，用于制定简单的目标语法。如下所示，它包括下面一组变量定义规则，描述需要识别的词。语音拨号应用中，一个可能的语法如下：

```
$digit = ONE | TWO | THREE | FOUR | FIVE |SIX | SEVEN | EIGHT | NINE | OH | ZERO;
$name = [ JOOP ] JANSEN |[ JULIAN ] ODELL |[ DAVE ] OLLASON |[ PHIL ] WOODLAND|
        [ STEVE ] YOUNG;
( SENT-START ( DIAL <$digit> | (PHONE|CALL) $name) SENT-END )
```

间隔符"|"表示几选一的，方括号"[]"表示可选项，尖括号"<>"表示一个或可多个重复。一个完整的语法可以被描述成一个如图3.1表示的网络。

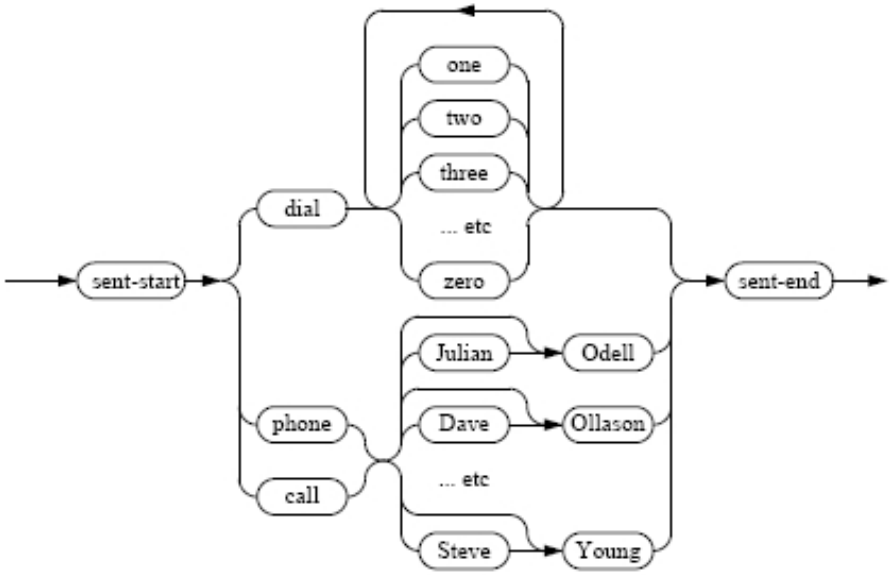


Fig. 3.1 Grammar for Voice Dialling

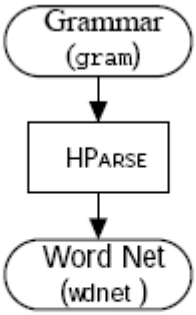


Fig. 3.2
Step 1

上图中上层的目标语法描述是为了方便使用者。HTK识别器实际上需要的是一个使用更低层的符号定义的词网络，称为HTK标准网格SLF格式（HTK Standard Lattice Format (SLF)），在SLF里，每个词的实例和词之间的转换都明确列出了。这种词网络可以使用HParse工具由上面的语法自动建立，比如，包含上面的语法的文件称为gram，执行下面的操作

HParse gram wdnet

将会建立一个等效的词网络并存储到wdnet文件中（如图3.2）。

3.1.2 第二步 —— 词典

建立一个词典的第一步是建立一个经排序的包含所有词的列表。在电话拨号任务中，建立一个所有词的列表可以轻松地手工完成。但是如果任务更加复杂，则应该根据训练数据中的例句建立词列表。此外，若建立一个鲁棒性声学相关模型，就需要使用一个包含很多词的并且音素更均衡的大的句子集合。因此，训练数据是由和电话拨号任务无关的英文句子组成。下面，给出一个由句子提示建立一个词列表的简单例子。这里训练用的句子都取自TIMIT数据库中使用的一些提示命令，并且为了便于处理，这些句子都重新编号。例如，开始一部分数据内容可能是下面这样的：

S0001 ONE VALIDATED ACTS OF SCHOOL DISTRICTS

S0002 TWO OTHER CASES ALSO WERE UNDER ADVISEMENT

S0003 BOTH FIGURES WOULD GO HIGHER IN LATER YEARS

S0004 THIS IS NOT A PROGRAM OF SOCIALIZED MEDICINE

等等

我们希望得到的训练词列表*wlist*可以通过下面介绍的方式自动生成。在使用HTK之前，首先需要以合适的方式编辑文本。例如，可能需要将所有的空格变成换行，然后使用UNIX命令sort和uniq对词进行排序，最终的结果是一行一词并严格按字母顺序排序。可以使用HTKTutorial目录中的脚本*prompts2wlist*完成此功能。

词典本身能由标准的输入源用HDMan建立。举个例子，可以用British English BEEP发音词典[2]。词典中的音标集合除重音记号之外都将被直接采用；并且在每个发音的后面都加上一个短暂的停顿（sp）。如果词典包含任何静音标记，则MP命令将把sil和sp合并成一个sil。这些操作都可以用HDMan和一个包含三行命令的编辑脚本来完成（保存在global.ded中）。

AS sp

RS cmu

[2] 可通过匿名访问FTP：svr-ftp.eng.cam.ac.uk/pub/comp.speech/dictionaries/beep.tar.gz获得。 需要注意的是应该移除词典开头的目录部分。

MP sil sil sp

其中，cmu表示采用了一种重音符号的风格，这种风格中，词汇的重读程度是通过直接写在音素名后面的单个数字表示的（例如，eh2 表示次重读（level 2 stress）的音素eh)。

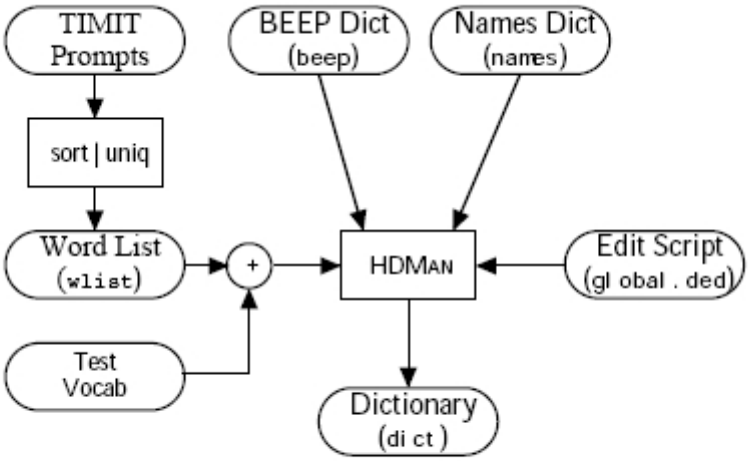


Fig. 3.3 Step 2

命令

HDMan -m -w wlist -n monophones1 -l dlog dict beep names

会建立一个名为*dict*的新词典，通过搜索源词典*beep*和*names*，找到*wlist*中每个词的发音（见图3.3）。这里的*wlist*可以仅仅是一个对目标语法中出现的所有词进行排序得到的列表。其中，*names*是一个人工生成的包含目标语法中出现的每个名字的发音的文件。HDMan的命令选项-l表示将构造词典相关的各种统计数据输出到一个log文件dlog中。特别地，如果有些词在发音词典中没有出现，它将给出提示。HDMan也能输出一个词典中出现的音素列表，这里称为monophones1。一旦训练和测试数据录好后，将根据每个音素重估出一个HMM。通常每个词典的词条格式如下：

WORD [outsym] p1 p2 p3 ....

方括号中的字符串指定词被识别出后的输出，缺损的输出是词本身。如果是空字符串，则表示没有任何输出。

让我们来看看词典是什么样子，下面列出了一部分词典内容：

A ah sp
A ax sp
A ey sp
CALL k ao l sp
DIAL d ay ax l sp
EIGHT ey t sp
PHONE f ow n sp
SENT-END [] sil
SENT-START [] sil
SEVEN s eh v n sp
TO t ax sp
TO t uw sp
ZERO z ia r ow sp

需要注意的是某些词比如 A 和 TO 有多种发音。SENT-START和 SENT-END 的实体有一个静音模型sil作为它们的发音，并且输出的是空符号。

3.1.3 第三步 —— 录数据

我们使用HTK工具HSLab录取训练和测试数据。这是一个结合了录音和标注功能的工具。在我们的例子里，仅使用HSLab来录音，因为标注已经存在了。如果没有预先准备好的训练语料（如TIMIT数据库中的数据），你可以使用HSLab通过文本（上面介绍的那样）建立它们，或者使用HSLab标注你的训练语料。 HSLab按如下的方式调用

HSLab noname

执行命令后，将显示这样的窗口，窗口上半部分是一个波形播放区域，下半部分是一排按钮，包括录音按钮等。如果一个文件名当作命令参数，HSLab将播放这个文件。这里，指定的文件名是*noname*，指明了将要录新的数据。HSLab并不给用户特别的提示，只要录音按钮被按下，它就将录得的数据交替地写到noname_0，noname_1,….这些文件里，因此很容易写一个shell脚本，当有noname_0之类的文件出现时，就输出提示信息，并按照事先约定的提示方式重命名文件（如图3.4所示）。

当训练语料句子的提示按上面的方法生成后，测试语料句子的提示在录音前也要生成。工具HSGen可以帮助我们完成测试句子提示的生成；HSGen能随机的遍历一个词网络并输出穿越过程中遇到的每个词。例如，下面的命令

HSGen -l -n 200 wdnet dict > testprompts

将创建一个包含200个词的测试语料，内容可能是下面这样的：

1. PHONE YOUNG
2. DIAL OH SIX SEVEN SEVEN OH ZERO
3. DIAL SEVEN NINE OH OH EIGHT SEVEN NINE NINE
4. DIAL SIX NINE SIX TWO NINE FOUR ZERO NINE EIGHT
5. CALL JULIAN ODELL

... etc

可以用它生成测试数据所需的提示文件*testprompts*。

3.1.4 第四步 —— 建立标注文件

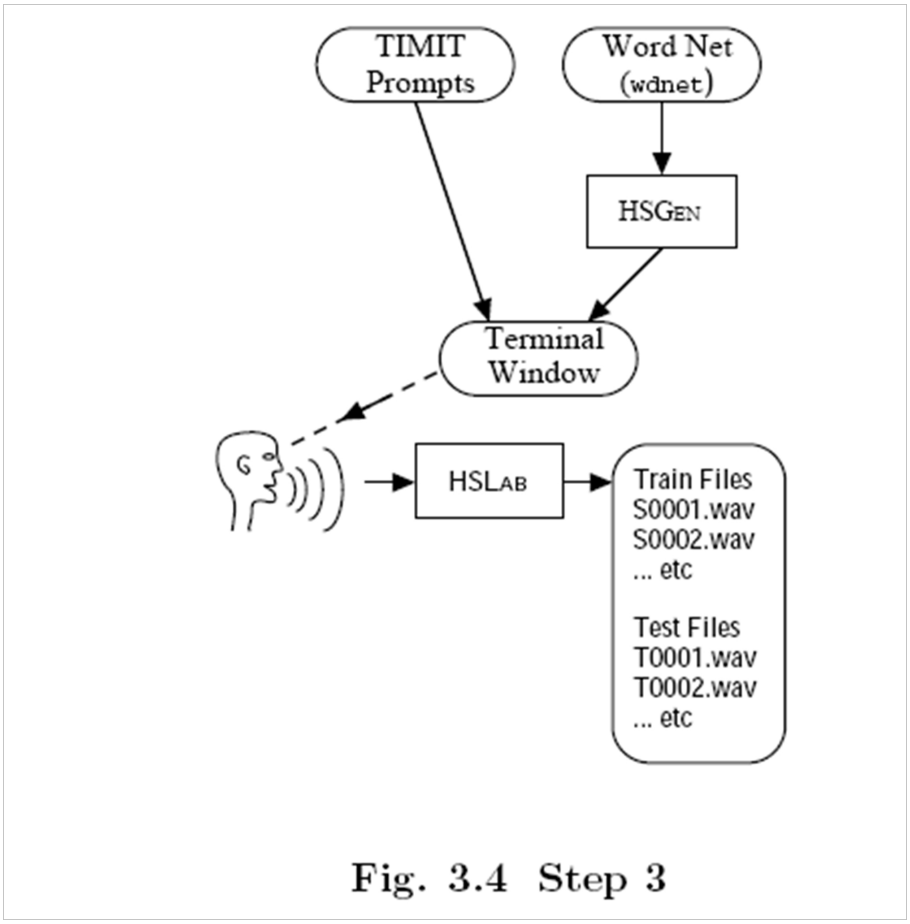


Fig. 3.4 Step 3

在训练一套HMM集合时，每个训练数据文件都必须对应一个音素级的标注。如果没有手工标注的数据用来开启初始（bootstrp）的模型训练，可以采用一种被称为flat-start的方法来代替。这种方法中，需要两套音素标注，开始使用的脚本中词之间不包含短暂停顿模型（sp），一旦一套合理的音素模型建立后，sp模型将被插入到可能有停顿的词之间。

两种音素标注的起始点都是一个按HTK标签格式规范的标注，可以很容易地用文本编辑器或脚本语言创建 。在RM Demo的point 0.4中能找到生成这种脚本的一个例子。另外，HTKTutorial目录下提供了脚本*prompts2mlf*，完成的功能是将上面例子的提示语料转化成下面的格式：

```
#!MLF!#
"*/S0001.lab"
ONE
VALIDATED
ACTS
OF
SCHOOL
DISTRICTS
.
"*/S0002.lab"
TWO
OTHER
CASES
ALSO
```

WERE
UNDER
ADVISEMENT
.
"*/S0003.lab"
BOTH
FIGURES
(etc.)

我们可以看到，提示标注要转换成“路径名”（path name），每个词单独作为一行，并且每个语料按照自己的周期终止。文件第一行指明了文件是*Master Label File* (MLF)。这个例子里，一个文件包含了全部的标注。HTK也允许每个标注拥有各自的MLF文件，但是全部存在于同一个MLF中更加有效率。

MLF中使用的“路径名”格式需要解释一下，因为实际上，它是一种模式（*pattern*）而不是名字（*name*）。HTK处理语音文件时，需要找和语音文件名字相同但是扩展名不同的标注（transcription）（或标注文件(*label file*)）。因此，如果要处理文件/root/sjy/data/S0001.wav，HTK要找一个名为/root/sjy/data/S0001.lab的标注文件。如果使用MLF文件，HTK 浏览此文件找到匹配标注文件文件名的相应模式。一个“*”可以匹配任何字符串，因此例子里使用的模式路径是没有约束的。所以，允许存储在不同位置的不同版本的语音使用同样的标注。

词一级的MLF建立后，可以使用标注编辑工具HLEd生成音素级的MLF。例如，使用上面的例子中的存储在文件words.mlf中的词一级MLF，执行下面的命令：

HLEd -l '*' -d dict -i phones0.mlf mkphones0.led words.mlf

将能生成一个音素级的标注，选项-l表示输出的模式中生成路径’*’，格式如下，

#!MLF!#
"*/S0001.lab"
sil
w
ah
n
v
ae
l
ih
d
.. etc

处理过程见图3.5。

HLEd编辑脚本mkphones0.led，包含以下命令：

EX
IS sil sil
DE sp

“扩展命令”EX用词典文件dict中相应的发音替换words.mlf中的每个词；
“插入命令”IS在每个语料的开头和结尾插入静音模型sil；
“删除命令”DE删除所有的sp标记，因为此时脚本里不需要该标注。

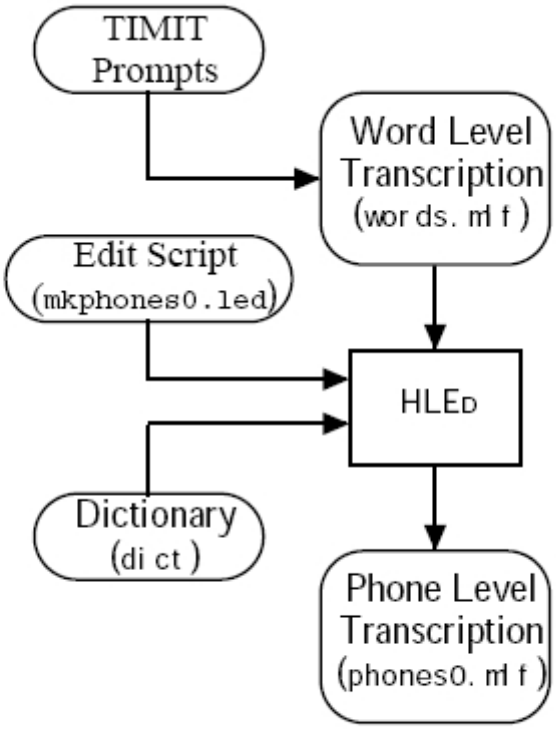


Fig. 3.5 Step 4

3.1.5 第五步 —— 数据编码 Coding the Data

数据准备的最后一步是对原始语音数据进行参数化运算，即将波形语音转化为特征向量序列。HTK支持基于FFT的和基于LPC的分析。这里使用的是Mel域倒谱系数（Mel Frequency Cepstral Coefficients）（MFCC）。编码工作可以通过工具HCopy，自动生成MFCC矢量。

为此，需要使用一个配置文件config，指定所有转化需要的参数，合理的配置文件格式如下：

```
# Coding parameters
TARGETKIND = MFCC_0
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F
```

其中有些设置是默认设置，为了完整性，这里都全部列出了。简要解释一下，此配置文件指定了目标参数是使用C0作为能量维的MFCC，帧长是10ms（HTK使用的单位是100ns），输出格式要经过压缩，并且加入了CRC校验。FFT之前需要经过加Hamming窗的处理，并且语音信号首先要经过预加重，预加重系数为0.97。filterbank使用26个三角滤波器，最后得到

12个MFCC系数。变量ENORMALISE默认值为true，表示对录音数据文件进行能量规一。在线识别中不能使用这种方法。由于我们要做的系统是在线的，因此这里这个变量被设成false。

并不需要预先创建特征编码文件，因为可由原始波形文件在处理过程中完成特征提取的工作，要实现这一过程，只需通过对相应HTK工具指定合适的配置文件（上面的格式）即可。但是，事先建立这些文件可以减少训练工作中的预处理的时间，本身这又是一个非常耗时的工作。

运行HCopy之前，要准备一个源文件的列表，这个列表包含源文件和相应的输出文件。例如，这个列表文件的前几行可能是这样的：

/root/sjy/waves/S0001.wav /root/sjy/train/S0001.mfc
/root/sjy/waves/S0002.wav /root/sjy/train/S0002.mfc
/root/sjy/waves/S0003.wav /root/sjy/train/S0003.mfc
/root/sjy/waves/S0004.wav /root/sjy/train/S0004.mfc
(etc.)

包含列表的文件可以当作脚本文件[3]，通常灌以后缀名scp （虽然HTK并不强制这样做）。脚本文件在命令中用“标准”选项 –S 指定，它们的内容只是作为命令行参数简单地读取。因此，避免了命令行有几千个参数这样的情况[4]。

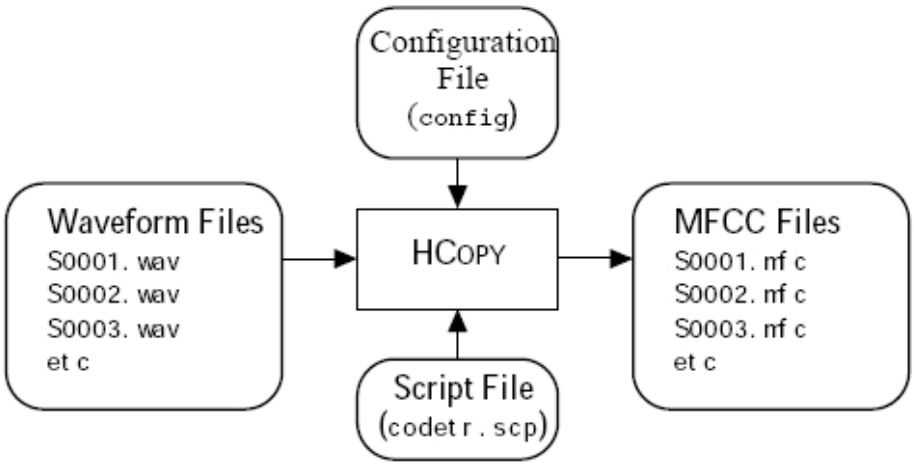


Fig. 3.6 Step 5

假设上面提到的脚本存储在名为codetr.scp的文件中，训练数据通过执行下面的命令进行编码：

HCopy -T 1 -C config -S codetr.scp

图3.6有详细的图示。所有的训练都已经开始后，编码测试数据（配置文件里使用TARGETKIND = MFCC_0_D_A）也采用类似的过程。

3.2建立单音素（Monophone）HMMs

这一节描述的是一组单高斯的单音素HMM模型的训练过程。训练开始前，应该有一组相同的HMM，每个模型的每个均值和方差都是相同的。然后重新训练这些原始模型，并加入sp模型，以及sil（静音）模型，然后再次训练这些单音素模型。

[3] 不用为文件包含编辑脚本感到困扰

[4] 大多数 UNIX shells，特别是 C shell 对命令行参数有限制，只能包含有限个且非常少的参数。

词典中的某些词条可能有多个发音。但是，使用HLEd将词一级的MLF扩展成音素级的MLF时，它总是选择遇到的第一个发音。一旦一套可用的单音素模型HMM集训练出来后，可以用识别工具HVite对训练数据进行强制校正。这就意味着，将根据声学特性选择发音，从而建议一个新的音素级MLF。这个新的MLF可被用于单音素模型的最终重估，即根据校正后的数据再一次重估。

3.2.1 第六步 —— 建立 Flat Start Monophones

HMM模型训练的第一步是定义一个原始模型。这个模型的参数并不重要，它的作用是定义模型的拓扑结构。对于基于音素的系统，一种较好的拓扑结构是使用3状态、无跳转的左-右模型，结构如下：

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "proto"
<BeginHMM>
<NumStates> 5
<State> 2
<Mean> 39
0.0 0.0 0.0 ...
<Variance> 39
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 ...
<TransP> 5
0.0 1.0 0.0 0.0 0.0
0.0 0.6 0.4 0.0 0.0
0.0 0.0 0.6 0.4 0.0
0.0 0.0 0.0 0.7 0.3
0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

每个特征矢量长度都是39。39是静态矢量（MFCC_0 = 13）加一阶系数（+13）和二阶系数（+13）得到的。

HTK工具HCompV会扫描一组数据文件，计算全局的均值和方差，并将给定的HMM中所有的高斯模型的均值方差都设为同样的值。因此，假设所有训练文件的列表被存储在train.scp中，下面的命令：

```
HCompV -C config -f 0.01 -m -S train.scp -M hmm0 proto
```

将在hmm0目录下建立一个新的原始模型，这个模型中，原模型的0均值和单位方差被替换成全局均值和方差。需要指出的是原始模型定义了特征矢量类型是MFCC_0_D_A (注意：’0’

而不是 'o')。这意味着一阶和二阶系数将在前面描述编码过程中计算并存储的静态MFCC的基础上被计算，并且添加在静态MFCC后，为确保在数据装载过程中先计算，应该修改配置文件config，其中的target kind要作修改，即配置文件中的TARGETKIND应该被改为：

TARGETKIND = MFCC_0_D_A

HCompV命令有一组选项。-f 选项将方差下限（称为vFloors）设为全局方差的0.01倍，这组向量值将被用于后面步骤中方差估计的下限值。-m选项指定均值和方差同时计算。定存储在hmm0目录下的新原始模型后，将会构造名为hmmdefs的*Master Macro File* (MMF)文件，hmmdefs中包含需要的每个单音素模型（包括"sil"）的副本，每个模型都复制了新的原始模型，且作了重新标记。MMF文件的格式和MLF文件的格式类似，而且也出于类似的目的，避免了单独定义模型导致数量过大的HMM定义文件（如图3.7）。

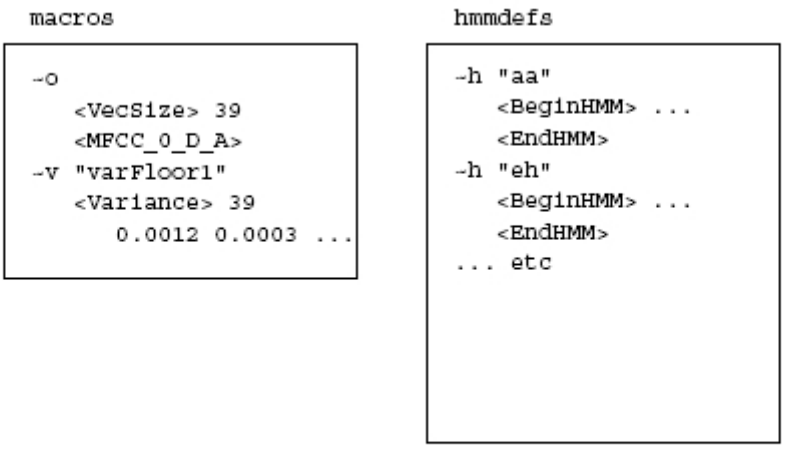


Fig. 3.7 Form of Master Macro Files

存储在hmm0目录下的flat start单音素模型可以用嵌入式重估工具HERest进行重估，命令如下：

```
HERest -C config -I phones0.mlf -t 250.0 150.0 1000.0 \
       -S train.scp -H hmm0/macros -H hmm0/hmmdefs -M hmm1 monophones0
```

完成的功能是，加载模型列表monophones0（monophones1 去掉sp模型）中的所有模型。这些模型将被重估，使用的数据是train.scp中列出的，新模型存储在目录hmm1下。这个命令中用到的大部分文件，除了文件macros以外都已经介绍了。macros文件需要包含一个称为*global options* 的macro和前面生成的方差下限macro，即vFloors。全局选项macro 简单地定义了模型的参数类型和特征向量大小，例如：

```
~o <MFCC_0_D_A> <VecSize> 39
```

见图3.7。这个选项可以和vFloors写在同一个文件中命名为macros。

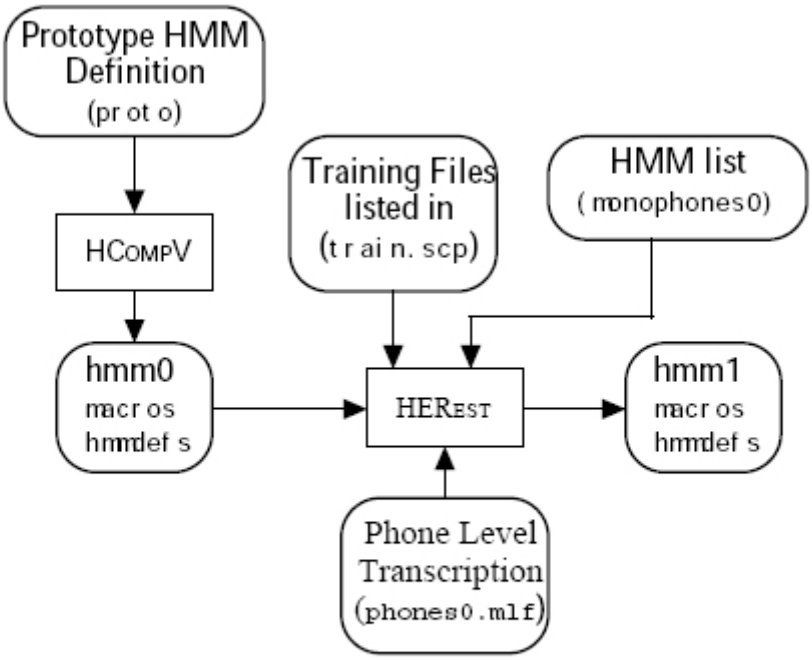


Fig. 3.8 Step 6

-t选项设置训练过程中的裁剪门限。裁剪限制了前-后向算法中状态队列的范围，可降低一个数量级的运算量。对大多数训练数据来说，都可以设置一个比较窄的裁剪门限，但是有些数据可能在声学特性上不能很好的匹配，因此有必要设置一个较宽的门限。HERest为处理这种情况，采用一种自动增益控制的裁剪门限。上面的例子中，一般的裁减门限是250.0，如果个别文件重估失败，裁剪门限增加150.0并对此文件重新处理。如果对文件的处理不成功，这种操作会一直循环，直到达到裁剪门限的上限1000.0。因此，我们可以假定，如果训练文件出现严重问题，则提示有错误应该予以修复（典型的错误是标注有错误），即应该丢弃相应的训练数据文件。目录hmm0中的初始单音素模型的训练过程见图3.8。

每次运行HERest，都是一次独立的重估过程。新的HMM集合被存储在新的目录下。HERest应该至少执行两次以上。每次改变输入和输出目录的名字（设置-H和-M选项），直到目录hmm3包含初始单音素模型的最终模型集。

3.2.2 第七步 —— 确定静音模型

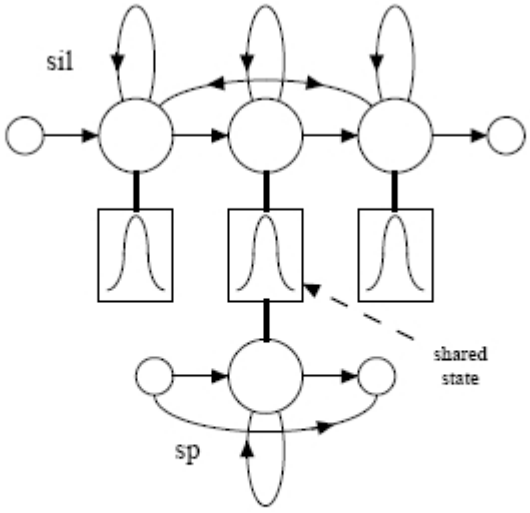


Fig. 3.9 Silence Models

经过前面的步骤，为每个音素和静音模型sil生成了各自的3状态左右无跳转HMM。下一步是在静音模型中加入第2个状态和第4个状态之间相互跳转的弧。其目的是，通过允许独立的状态吸收训练数据中的各种噪声，使模型更具鲁棒性。后向跳转的弧可以让状态吸收噪声，而不通过下一个词的模型。

同样出于这个目的，要建立只有单状态的sp模型。这种模型被称为*tee-model*，有一条出入节点之间直接跳转弧，sp的输出（emitting）状态绑定在sil模型的中间状态上。这两个静音模型（two silence models）的拓扑结构见图3.9。

这些静音模型可以按两步来完成：

- 使用一个文本编辑器，编辑文件hmm3/hmmdefs，拷贝sil模型的中间状态来建立一个新的sp模型，并存储到新的MMF，*hmmdefs*中，放到新目录hmm4下。
- 运行HMM编辑工具HHEd，加入额外的跳转弧，并将sp的状态绑定到sil的中心状态上。

HHEd的工作方式类似于HLEd。它允许用脚本定义一套命令去修改一组HMM集合。HHEd的运行方式如下：

```
HHEd -H hmm4/macros -H hmm4/hmmdefs -M hmm5 sil.hed monophones1
```

上面的sil.hed包含下面的命令：

```
AT 2 4 0.2 {sil.transP}
AT 4 2 0.2 {sil.transP}
AT 1 3 0.3 {sp.transP}
TI silst {sil.state[3],sp.state[2]}
```

AT命令向给定的转移概率矩阵中加入转移弧；最后一个TI命令建立一个绑定状态（tied-state）称为silst。绑定状态的参数影响到每个静音模型并存储在*hmmdefs*文件中，初始状态参数都被这个宏替换。关于Macros更详细的描述将在后面提到。现在只要理解这是HTK实现的一种参数共享的机制。注意这里使用的音素列表已经发生变化了，因为原来的列表monophones0已经加入新的sp模型而得以扩展。从上面的HHEd命令中可以看到，新文件被命名为monophones1。

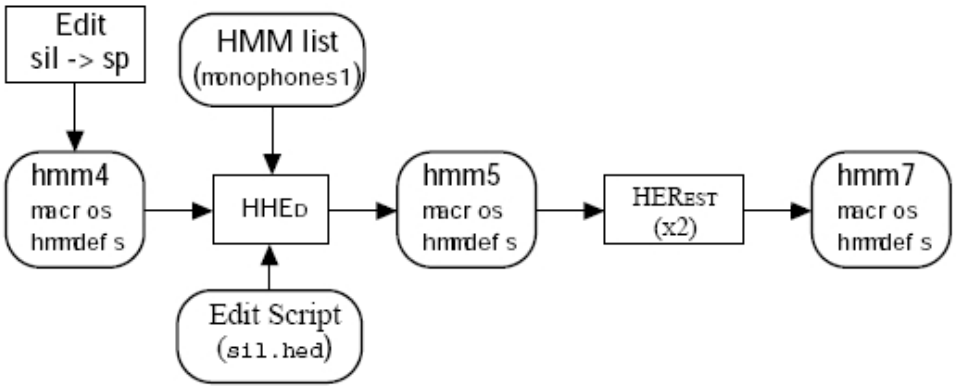


Fig. 3.10 Step 7

最后，使用词之间加入sp的音素标注，运行两遍HERest。最终会生成monphone的HMM模型集并存储在目录hmm7中，详细的步骤见图3.10。

3.2.3 第八步 —— 重新校正训练数据

前面已经提过，词典中可能有些词有多种发音，特别是功能词(指前置词, 连词等)。前面建立的音素模型可用于重新校正（realign）训练数据并建立新的标注。这个工作可以仅由一个工具完成，即HTK识别工具HVite。

```
HVite -l '*' -o SWT -b silence -C config -a -H hmm7/macros \
    -H hmm7/hmmdefs -i aligned.mlf -m -t 250.0 -y lab \
    -I words.mlf -S train.scp dict monophones1
```

这个命令使用存储在hmm7下的HMM集合，根据存储在词典dict中的发音，将词集标注words.mlf 转换为新的音素级标注aligned.mlf（见图3.11）。和第四步中完成最初的词到音素映射的HLEd相比，关键的不同在于，识别器考虑了每个词的所有发音，并输出于声学数据最匹配的发音。

上面的命令中，-b选项用于在每句语料的开始和结束点插入静音模型。假设词典中包含

```
silence sil
```

就可以在上述命令中使用名称silence。

注意首先应该对词典按大小写排序（大写在前），再按字母顺序。-t选项设置裁剪门限为250.0， -o选项用于禁止在MLF中输出得分、词内容和时间边界。

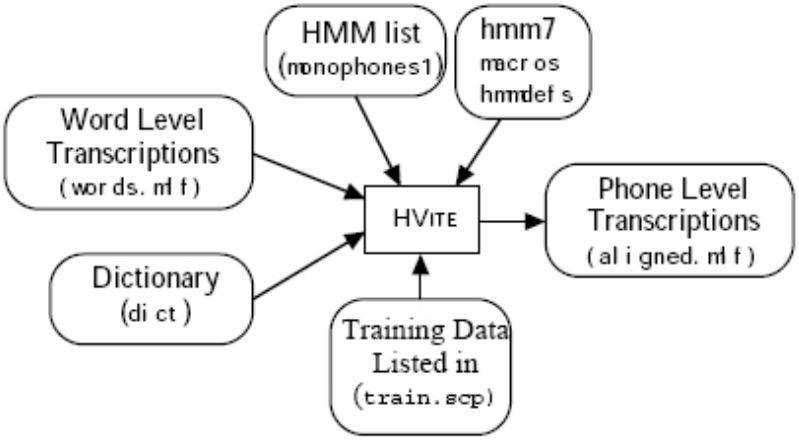


Fig. 3.11 Step 8

新的音素集合建立后，可再次执行两遍HERest，重新重估HMM集合的参数。如果这些工作都完成了，最终的单音素模型集合将被存储在目录hmm9下。

3.3 建立邦定状态 Triphones

现在已经得到一组单音素 HMM，模型训练的最后阶段是建立上下文相关的 triphone HMM。分两步来完成。首先，monophone 标注要被转化成 triphone 标注，通过拷贝 monophone 再重估来训练一组 triphone 模型。其次，这些 triphone 声学特性相似的状态将邦定在一起，防止因为数据不足造成一些状态无法重估，以确保所有的状态分布能够稳定的重估，从而并具有足够的鲁棒性。

3.3.1 第九步 —— 由 Monophones 建造 Triphones

上下文相关的triphone可通过简单地克隆monophone后使用triphone标注重估得到。后者必须要事先通过HLEd工具建立，它有一个附带产品，既生成了所有的（训练数据中至少有一个样本）triphone列表。执行：

HLEd -n triphones1 -l '*' -i wintri.mlf mktri.led aligned.mlf

将aligned.mlf中的monophone标注转换成triphone标注，并保存到wintri.mlf中，同时，将triphone列表写入到文件triphones1中。编辑脚本mktri.led中包含下面的命令：

WB sp
WB sil
TC

两个WB命令定义sp和sil作为词边界标记（*word boundary symbols*）。 这些标记将对TI命令执行的上下文扩展起间隔作用。下面的脚本可以看出，所有的音素（除词边界标记外）都被转化成triphone。例如，

sil th ih s sp m ae n sp ...

变成

sil th+ih th-ih+s ih-s sp m+ae m-ae+n ae-n sp ...

这种triphone标注的格式是包含词内扩展triphone。注意，一些biphone也是以词为边界，通过上下文生成的，不过它只包含两个音素。模型的克隆操作可用使用HMM的编辑器HHEd有效地完成。

```
HHEd -B -H hmm9/macros -H hmm9/hmmdefs -M hmm10 \
    mktri.hed monophones1
```

编辑脚本mktri.hed包含克隆命令CL，后面跟着TI命令来绑定所有转移概率矩阵：

```
CL triphones1
TI T_ah {(*-ah+*,ah+*,*-ah).transP}
TI T_ax {(*-ax+*,ax+*,*-ax).transP}
TI T_ey {(*-ey+*,ey+*,*-ey).transP}
TI T_b {(*-b+*,b+*,*-b).transP}
TI T_ay {(*-ay+*,ay+*,*-ay).transP}
...
```

文件mktri.hed可使用HTKTutorial目录中提供的*Perl* 脚本*maketrihed*生成。运行HHEd命令的时候，你会收到关于试图为sil和sp模型绑定状态转移概率的警告。因为这两个模型都不是上下文相关的，所以没有矩阵需要绑定。

克隆命令CL有一个参数，是包含上面生成的triphone（和biphone）列表的文件名。列表中每个模型的格式是a-b+c，它寻找单音素模型（monophone） b 并复制一份。每个TI命令的参数包括一个宏名称和一个HMM组成的列表。后者使用一个符号，试图模拟HMM参数组的分段结构，这种结构中，转移概率矩阵transP作为HMM的子结构。括号中的项目是一种匹配triphone的模式，每个phone都有右相关biphone和左相关biphone。

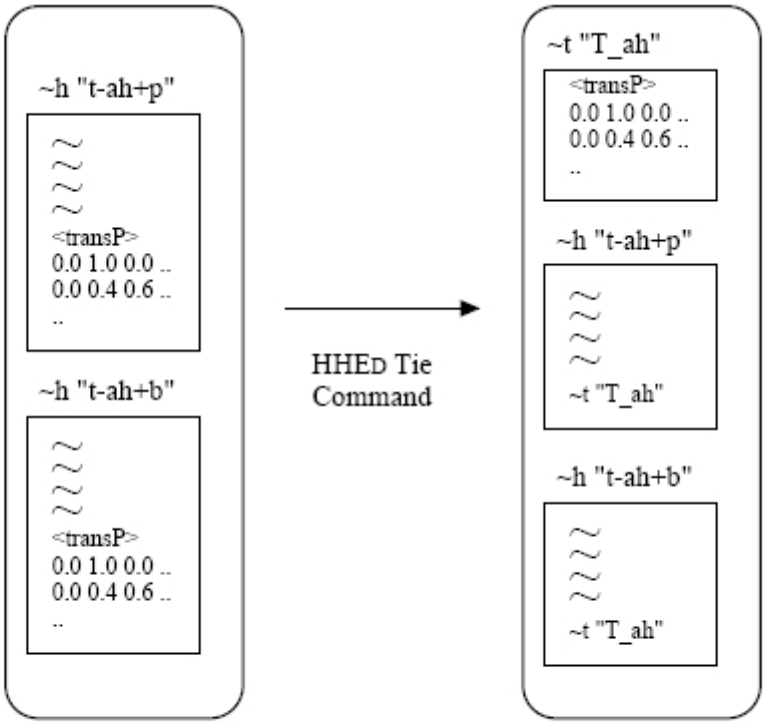


Fig. 3.12 Tying Transition Matrices

到目前为止，宏和绑定仅仅是顺带提到的，在第7章中有更详细的解释，这里只是简单介绍一下。绑定意味着一个或多个HMM共享相同参数。在图3.12的左边，可以看到两个HMM的定义。每个模型都自己独立的转移概率矩阵；再看右边，上面的脚本mktri.hed中第一个TI命令产生的效果是，独立的转移概率矩阵以一个名为T_ah的宏（macro）取代，这个宏包含了一个被两个模型共用的矩阵。重估绑定的参数时，原来用于非绑定参数的数据都汇集起来，

这样将获得更可靠的重估。

当然，不加选择的绑定可能会影响性能。因此，仅仅绑定对识别率有很小影响的参数是很重要的。 比如转移概率参数，对声学上下文来说并不是非常重要，但仍然需要精确评估。如果不进行绑定，某些triphone可能只会出现一次或者两次，所以只能得到非常不准确的评估。数据不足的问题也会影响输出，但是这个问题将在下一步中讨论。

到目前为止，所有的HMM都被存成文本格式，而且能像任何文本文件一样被检查。模型文件可能会变得更大，这就应该考虑空间和加载/存储时间。为了提升效率，HTK可以存储或加载二进制文件格式的MMF，只需要设置-B选项就可以了。

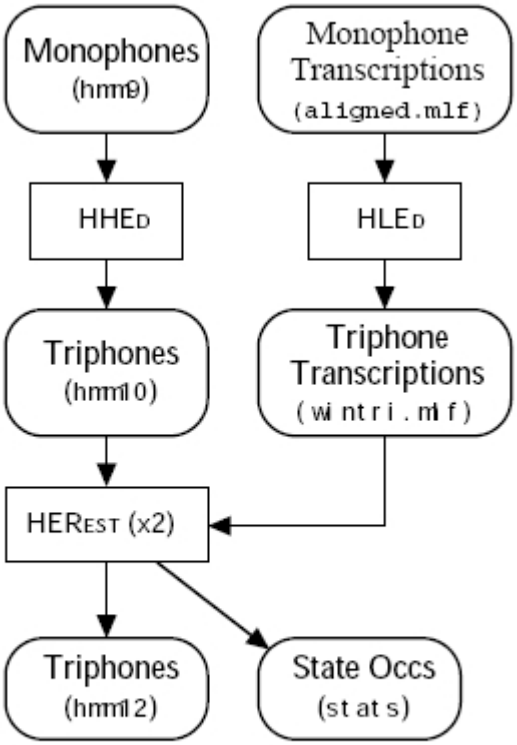


Fig. 3.13 Step 9

上下文相关模型克隆后，新的 triphone 将通过 HERest 重估。做法和先前的类似，唯一不同的是原来的 monophone 模型列表要换成 triphone 列表，并用 triphone 标注代替 monophone 标注。

HERest 的最后一步，应该使用-s 选项，生成一个状态样本统计的文件 stats。结合均值和方差，这些可以用于后面即将介绍的状态聚类过程中的概率计算。图 3.13 显示了 HMM 这一步的创建过程。重估需要进行两次，所以最终模型集合会存储在目录 hmm12 下。

```
HERest -B -C config -I wintri.mlf -t 250.0 150.0 1000.0 -s stats \
    -S train.scp -H hmm11/macros -H hmm11/hmmdefs -M hmm12 triphones1
```

3.3.2 第十步 —— 建立绑定状态的 Triphone（Tied-State Triphones）

前面一节得到了一组triphone HMM，其中所有的triphone共用相同的转移概率矩阵。重估这些模型时，最终输出的很多方差都仅是下限值，因为很多状态的训练数据不足，导致无法重估。模型训练过程的最后一步，是要绑定triphone的状态，用于共享数据从而实现参数鲁棒性重估。

前一步，TI命令用于明确地绑定一组转移概率矩阵的所有成员。然而，选择绑定哪些状态更加复杂微妙，因为语音数据的统计结果对应什么输出状态对识别器来说至关重要。

HHEd 提供了两种机制支持状态聚类，并将类中的状态进行绑定。第一种是数据驱动的，使用状态间相似度的度量。第二种使用决策树，且基于对每个triphone的左右上下文提问题。决策树试图寻找具有声学上最大差异的上下文，以此区分类别。

决策树状态的绑定一般通过执行HHEd来完成。即：

```
HHEd -B -H hmm12/macros -H hmm12/hmmdefs -M hmm13 \
        tree.hed triphones1 > log
```

注意，输出是存储到一个log文件中的，这是很重要。因为通常需要对一些门限作调整。

编辑脚本tree.hed包含了关于检查可能聚类的上下文的指令,这个脚本可能很长且很复杂。RM Demo中可以找到一个自动生成此文件的脚本*mkclscript*。此指南中使用的一个版本的tree.hed脚本包含在HTKTutorial目录下。注意这个脚本仅完成生成多个TB命令的功能（状态的决策树聚类）。问题（QS）也需要用户定义，RM demo（lib/quests.hed）中提供了一个问题列表的例子，可能对很多任务都适用（至少作为一个例子是有参考价值的）。用于聚类英文音素模型的全部脚本太长了，不能将文本内容全部列出，但是，下面一段给出了它的主要内容：

```
RO 100.0 stats
TR 0
QS "L_Class-Stop" {p-*,b-*,t-*,d-*,k-*,g-*}
QS "R_Class-Stop" {*+p,*+b,*+t,*+d,*+k,*+g}
QS "L_Nasal" {m-*,n-*,ng-*}
QS "R_Nasal" {*+m,*+n,*+ng}
QS "L_Glide" {y-*,w-*}
QS "R_Glide" {*+y,*+w}
....
QS "L_w" {w-*}
QS "R_w" {*+w}
QS "L_y" {y-*}
QS "R_y" {*+y}
QS "L_z" {z-*}
QS "R_z" {*+z}
TR 2
TB 350.0 "aa_s2" {(aa, *-aa, *-aa+*, aa+*).state[2]}
TB 350.0 "ae_s2" {(ae, *-ae, *-ae+*, ae+*).state[2]}
TB 350.0 "ah_s2" {(ah, *-ah, *-ah+*, ah+*).state[2]}
TB 350.0 "uh_s2" {(uh, *-uh, *-uh+*, uh+*).state[2]}
....
TB 350.0 "y_s4" {(y, *-y, *-y+*, y+*).state[4]}
TB 350.0 "z_s4" {(z, *-z, *-z+*, z+*).state[4]}
TB 350.0 "zh_s4" {(zh, *-zh, *-zh+*, zh+*).state[4]}
TR 1
AU "fulllist"
CO "tiedlist"
ST "trees"
```

首先，RO命令加载上一步最后生成的统计文件stats，并将判决门限（outlier threshold）设为100.0。判决门限设定了任何分类的最小样本数（occupancy），防止单个状态由于声学特性和其他所有状态差异太大而形成单个分类。TR命令设置trace等级为0用于准备加载问题。每个QS命令加载一个问题，每个问题由一组上下文定义。例如，第一个QS命令定义一个名为L_Class_Stop的问题，如果左边的上下文是阻塞音p，b，t，d，k或g则返回真。

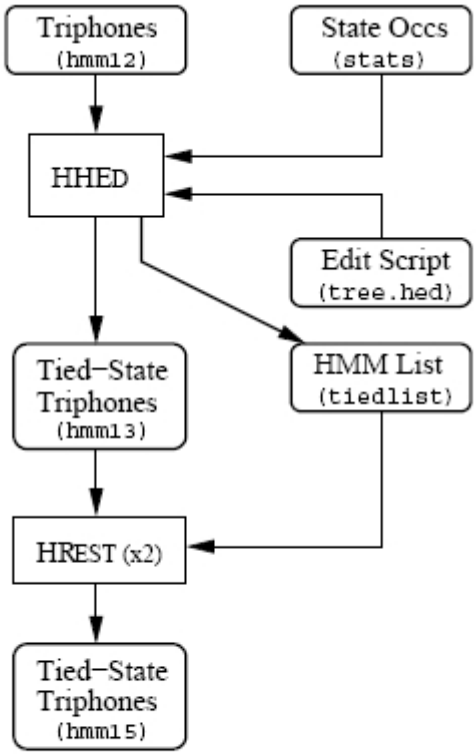


Fig. 3.14 Step 10

注意，在一个triphone系统中，必须包括一个phone的左、右上下文的相关问题集。问题集应该在有广度和一般性的分类上针对每个phone的细节上提出（比如辅音、元音、鼻音、双元音，等等）。理想情况是，使用QS命令加载的全套问题集应该包含每种可能的影响音素声学特性的上下文环境，并能包含任何语言学或语音学的相应分类。创建不必要的问题并不会造成危害，因为那些问题将视为同数据不相关的，并被忽略。

第二个命令TR使能中间处理报告有效。使得可以监控每个下面的TB命令。每个TB命令对状态组聚出一个分类。例如，第一个TB命令用于音素aa的所有上下文相关情况下，第一个输出状态（emitting state）（译注：即状态2，因为状态1是虚状态）。

每个TB命令以下面的方式工作。首先，最后一个参数定义的每组状态汇聚成一个的分类。QS命令加载的问题集中的每一个问题将一个分类分裂成两组，使训练数据对两组的log似然相对于一组来说有增加。而且将似然增加最大化的问题被作为决策树的第一个分枝。然后重复这个过程，直到在任何节点上用任何问题分类带来的log似然增加都小于命令第一个参数定义的裁剪门限（这里是350.0）。

注意RO和TB命令里给出的值会影响绑定程度，即聚类后系统的状态数目。命令中的值应该取决于可用的训练数据。作为聚类的最后一步，任何可能合并的分类如果由于合并造成的log似然值的降低在门限以下，那么这两个分类将被合并。完成后，每个分类i的状态都将绑定在同一个共享状态上，使用了一个宏，名为xxx_i，其中，xxx是TB命令的第2个参数给

定的。

到目前为止使用的triphone只包括那些对于训练数据必要的部分。AU命令将它的参数指定的列表扩展为包括识别器需要的所有triphone的新列表。这个列表可以通过以下方式生成，例如，使用HDMan将全部词典（不仅仅是训练词典），使用TC命令转化成triphones，并使用-n选项输出一个明确的triphone列表。

```
HDMan -b sp -n fulllist -g global.ded -l flog beep-tri beep
```

-b sp 选项指定sp 音素作为词边界使用，所以它被排除在triphone之外。AU命令的作用是使用决策树生成新的列表里先前没有出现的新的triphone。

所有状态绑定和综合新的模型一旦完成，一些模型可能完全共享3个状态和状态转移概率矩阵，因此完全相同。CO命令用于结合模型集合，通过发现所有同样的模型并将它们绑定到一起[5]，产生新的模型列表称为tiedlist。

使用决策树聚类的一个优势是它可以综合先前没有出现过的triphone。为完成这一点，决策树必须存储起来，这由ST命令完成。随后，如果需要新的先前没出现过的triphone，例如，新的词条的发音，扩展的模型集合能被HHEd重新加载，使用LT命令重新加载决策树，然后，通过使用AU命令建立一个新的扩展后的triphone列表。

HHEd完成后，绑定的效果可以被评估了，如果有必要，门限也将被调整。日志文件包含统计的结果，给定保留下来的总的物理状态数，以及结合之后的模型数。最终，模型使用HERest做最后两次重估。图3.14详细描述了HMM建立过程中的最后一步。训练好的模型将存储在hmm15/hmmdefs中。

3.4 识别器评估

识别器现在已经完成了，需要评估它的性能。识别网络和词典已经被构造，测试数据也录过了。因此，准备工作都已经做好，现在需要做的就是运行识别系统并使用HTK分析工具HResults评估结果。

3.4.1 第 11 步 —— 识别测试数据

假设test.scp中是编码过的测试文件列表，接着就将识别每个测试文件，脚本输出到一个名为recout.mlf的MLF文件中。命令如下：

```
HVite -H hmm15/macros -H hmm15/hmmdefs -S test.scp \
    -l '*' -i recout.mlf -w wdnet \
    -p 0.0 -s 5.0 dict tiedlist
```

选项-p 和 -s 分别用来设置 单词插入惩罚(*word insertion penalty*)和语言模型缩放因子(*grammar scale factor*)。

单词插入惩罚是一个固定值，当每个token从一个词的end节点转移到下个词的start节点时加上这个固定值。语法缩放因子表示当从一个单词结尾跳转到下一个单词的开始时，在语言模型概率加入到token之前，对其进行缩放的值。这些参数对识别器性能有重要影响，因此，针对测试数据的调整是很有必要的。

词典包含monophone标注，然而提供的HMM列表包含词内triphone。HVite加载词网络wdnet时将进行必要的转换。如果HMM列表既包含monophone又包含上下文相关音素，那HVite将很难处理。通过设置*FORCECXTEXP*为TRUE，*ALLOWXWRDEXP*为FALSE，即可以强制扩展词内网络。

---

[5] 注意如果转移概率矩阵没有被绑定，CO命令将失效，因为所有的模型都各不相同，它们有不同的转移概率矩阵。

假如MLF文件testref.mlf 包含每个测试文件[6]的单词级标注，那么可以运行测试识别器的实际性能了，运行格式如下：

HResults -I testref.mlf tiedlist recout.mlf

结果将按下面的格式输出

```
====================== HTK Results Analysis ==============
Date: Sun Oct 22 16:14:45 1995
Ref : testrefs.mlf
Rec : recout.mlf
---------------------- Overall Results ----------------
SENT: %Correct=98.50 [H=197, S=3, N=200]
WORD: %Corr=99.77, Acc=99.65 [H=853, D=1, S=1, I=1, N=855]
========================================================
```

以SENT开始的一行：指出有200条测试语料，其中197 (98.50%)条正确识别。下一行以WORD开始：给出单词级的统计结果，指出总共有855个词，其中853 (99.77%) 个词被正确识别，其中有1个删除错误（D），1个替换错误（S）和1个插入错误（I）。数值为99.65%的正确率（Acc）比准确率（Cor）要低，因为它考虑了插入错误，但是Cor忽略了。

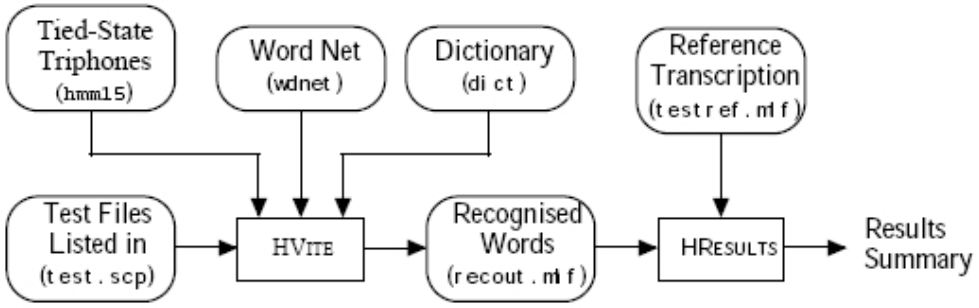


Fig. 3.15 Step 11

3.5 在线运行识别器

识别系统也能接收在线的输入。为实现这一点，只需要设置配置参数变量，将输入音频转换成正确的参数格式。特别地，需要在配置文件config里加上下面一段，存为新的配置文件config2。

```
# Waveform capture
SOURCERATE=625.0
SOURCEKIND=HAUDIO
SOURCEFORMAT=HTK
ENORMALISE=F
USESILDET=T
MEASURESIL=F
```

[6] 之前使用HLEd工具时可能在每个脚本的开始和结束都加入了静音模型，HResults提供了可选的选项-e用来忽略静音模型（或其他任何符号）。

```
    OUTSILWARN=T
```

以上这些，指出输入源是采样周期为62.5us的音频[译者注：采样频率为16KHz]。并开启静音检测器，并且在启动时，需要对背景的语音/静音噪声等级进行度量。最后一行用于确保当开启静音度量时，会打印一条通知信息。

当配置文件指明采用直接音频输入后，可以像前面一步提到的那样运行HVite，唯一区别在于，不需要在命令参数中指定文件。

```
HVite -H hmm15/macros -H hmm15/hmmdefs -C config2 \
    -w wdnet -p 0.0 -s 5.0 dict tiedlist
```

启动时，HVite将提示用户说出任意的句子（大约4秒），用来度量语音和背景静音的等级。接着，将开始循环识别，如果trace level设为1，每句话的识别过程都将被输出到终端上。典型的交互过程是这样的：

```
Read 1648 physical / 4131 logical HMMs
Read lattice with 26 nodes / 52 arcs
Created network with 123 nodes / 151 links

READY[1]>
Please speak sentence - measuring levels
Level measurement completed
DIAL FOUR SIX FOUR TWO FOUR OH
    == [303 frames] -95.5773 [Ac=-28630.2 LM=-329.8] (Act=21.8)

READY[2]>
DIAL ZERO EIGHT SIX TWO
    == [228 frames] -99.3758 [Ac=-22402.2 LM=-255.5] (Act=21.8)

READY[3]>
etc
```

加载期间，将打印出关于识别器的不同成分的信息。物理模型是系统中使用的不同的 HMM，逻辑模型包含所有模型名称。逻辑模型的数量比物理模型的数量要多，因为很多逻辑上不同的模型在前面的模型建立步骤中合并成同一个物理模型。网格信息是指识别器语法中的链接和结点数目。网络信息是指通过使用当前的HMM集合、词典和任何指定的上下文扩展规则扩展的网格，所建立的实际识别器网络。每句话说完后，都会给出总帧数，以及每帧的log似然值、总的声学得分、总的语言模型得分和活动模型的平均数。

注意，如果要识别一个新的名字，需要有两点改变：

1. 加入新的名字到语法（grammar）中。
2. 新名字的发音需要加入到词典（dictionary）中去。

如果需要新的不存在的triphone，那么可以通过加载已存在的triphone集合到HHEd中，再使用LT命令加载决策树，然后使用AU命令生成新的完整的triphone集合。

3.6 HMM 模型自适应

前面的章节已经描述了建立一个简单的语音拨号系统所需要的步骤。为了简化这一过程，只是使用了单用户的训练数据训练出说话人相关的模型。因此，识别器对任何其他用户的识别率将会很低。为克服这个局限性，应该建立一套说话人相关模型，但是这样可能需要

很大数量的不同说话人的训练数据。一种可行的方法是自适应现有的说话人相关模型，使用少量训练或自适应数据让模型自适应到新的说话人的特征上。通常，自适应技术应用于训练好的说话人无关模型集合，使其对于独特的说话人有更好的性能。

HTK支持有监督和无监督的自适应。所谓有监督，就是知道数据的正确脚本，而无监督就是使用假设的脚本。HTK中，有监督的自适应通过HEAdapt使用最大似然线性回归（MLLR）算法或最大化后验概率（MAP）算法重估一连串的转换或转换模型集合，使得当前模型集合和自适应数据之间的失配最小化，且都是离线完成的。无监督的自适应由HVite完成（见13.6.2小节），仅使用MLLR。

下面的章节描述了使用HEAdapt工具进行离线的有监督的自适应（使用MLLR）。

3.6.1 第 12 步 —— 准备自适应数据

在识别器建立的一般过程中，自适应的第一个步骤是数据准备。需要使用新用户的语音数据自适应模型和测试自适应系统。数据的获取类似于原始测试的获取。首先，使用HSGen生成自适应数据和测试数据的提示列表。例如，输入：

```
HSGen -l -n 20 wdnet dict > promptsAdapt
HSGen -l -n 20 wdnet dict > promptsTest
```

将会建立自适应数据和测试数据两个提示文件。需要的自适应数据的数量通常是根据经验获得的，但是需要30秒时间的输入后，可以提高整体表现。 既然这样，大概20句话就足够了。可以用HSLab录取相关的语音。

假设脚本文件codeAdapt.scp和codeTest.scp 分别列出了用于自适应和测试的数据的源文件和输出文件，那么可以使用HCopy命令得到这两组数据的编码后的结果。HCopy命令在下面给出：

```
HCopy -C config -S codeAdapt.scp
HCopy -C config -S codeTest.scp
```

数据准备的最后一步，包括生成用于自适应模型的自适应数据的上下文相关音素标注，以及生成用于评估性能的测试数据的单词级标注。测试数据的脚本可以使用prompts2mlf获得。为了使多发音问题最小化，可以通过HVite对自适应数据完成*forced alignment*得到自适应数据的音素级标注。假设在adaptWords.mlf中列出了单词级的标注，那么可以通过下面的命令将音素标注放置到adaptPhones.mlf中。

```
HVite -l '*' -o SWT -b silence -C config -a -H hmm15/macros \
    -H hmm15/hmmdefs -i adaptPhones.mlf -m -t 250.0 \
    -I adaptWords.mlf -y lab -S adapt.scp dict tiedlist
```

3.6.2 第 13 步 —— 完成转换

HEAdapt根据可用的自适应数据的数量，提供两种形式的MLLR自适应。如果仅有少量数据可用，那么对每个模型的每个输出状态使用全局变换。如果有更多可用的自适应数据，那么可以对特定Gaussians集合生成更多特定的变换。为了确定当前自适应数据可以实现的转换的数量，HEAdapt使用一种回归类树来聚类使用不同转换方法的输出分布状态。HTK工具HHEd可用于建立回归类树，并将之作为HMM集合的一部分存储。例如：

```
HHEd -B -H hmm15/macros -H hmm15/hmmdefs -M hmm16 regtree.hed tiedlist
```

使用存储在目录hmm15下的模型建立一个回归类树。新的模型和回归类树信息一起被写入到目录hmm16下。HHEd编辑包含下面命令的脚本：

```
RN "models"
LS "stats"
RC 32 "rtree"
```

RN命令给HMM集合分配一个标识符。LS命令加载最后一次使用HERest生成hmm15中的模型产生的状态样本统计文件stats。RC命令使用这些统计数据，建立一个包含32个节点（根结点或叶子结点）的回归类树。

HEAdapt即可以实现静态的自适应，也可以实现增量自适应。静态自适应中所有的自适应数据作为单个块中处理；增量自适应需要在获得指定数目的语料后执行，且通过-i选项控制。在此指南内，默认使用静态自适应。

HEAdapt的典型使用包括两遍操作。第一遍，执行全局的自适应。第二遍，使用全局转换公式来变换模型集合，产生更好的帧/状态类别，并使用回归类树评估一套更精细的转换关系。估计转换之后，HEAdapt可以输出新的自适应后的模型集合或在转换模型文件（TMF）中输出转换本身。如果考虑存储，后者将非常有利，因为TMF一般比MMF小很多，而变换模型集合中使用TMF带来的计算量是可以忽略的。

下面的两个HEAdapt的应用，示范了一个静态的two-pass自适应方法，这里全局转换和回归类树转换分别存储在global.tmf 和rc.tmf文件中。标准的-J和-K选项分别用于加载和存储TMF。

```
HEAdapt -C config -g -S adapt.scp -I adaptPhones.mlf -H hmm16/macros \
        -H hmm16/hmmdefs -K global.tmf tiedlist
HEAdapt -C config -S adapt.scp -I adaptPhones.mlf -H hmm16/macros \
        -H hmm16/hmmdefs -J global.tmf -K rc.tmf tiedlist
```

3.6.3 第 14 步 —— 评估自适应系统

为了评估自适应系统的性能，要是用HVite识别先前录得的测试数据。假设testAdapt.scp包含一个所有编码过的测试数据的文件列表，可使用HVite识别先前录取的测试数据。假设testAdapt.scp包含所有编码过的测试文件列表，那么可以采用以前类似的方式调用HVite，不同之处在于加入了-J 参数，用于加载模型转换文件rc.tmf。

```
HVite -H hmm16/macros -H hmm16/hmmdefs -S testAdapt.scp -l '*' \
    -J rc.tmf -i recoutAdapt.mlf -w wdnet \
    -p 0.0 -s 5.0 dict tiedlist
```

自适应模型集合的结果通常可以通过HResults获取。

RM Demo 包含了一段说话人自适应的内容 (point 5.6)，使用自适应模型的识别的结果如下：

```
    ====================== HTK Results Analysis =======================
    Date: Wed Jan 06 21:09:23 1999
    Ref : usr/local/htk/RMHTK_V2.1/RMLib/wlabs/dms0_tst.mlf
    Rec : adapt/dms0_tst.mlf
    ------------------------ Overall Results --------------------------
    SENT: %Correct=66.33 [H=65, S=33, N=98]
    WORD: %Corr=94.25, Acc=93.10 [H=738, D=11, S=34, I=9, N=783]
    ===================================================================
```

可以通过使用未自适应的模型对测试数据进行识别后，对比评估自适应模型得到的性能改善。对RM Demo 任务来说，使用为自适应的模型得到下面的结果：

```
    ====================== HTK Results Analysis =======================
    Date: Mon Dec 14 10:59:28 1998
    Ref : usr/local/htk/RMHTK_V2.1/RMLib/wlabs/dms0_tst.mlf
    Rec : unadapt/dms0_tst.mlf
    ------------------------ Overall Results --------------------------
```

SENT: %Correct=46.00 [H=46, S=54, N=100]
WORD: %Corr=89.04, Acc=86.43 [H=715, D=26, S=62, I=21, N=803]
==========

3.7 小结

这一章描述了建造一个状态绑定的音素级的连续语音识别系统。在这个过程中，我们接触到HTK中的大部分工具：录音，数据预处理，HMM定义，训练工具，自适应工具，网络，解码和评估。本书接下来的部分将针对每个话题进行详细讨论。