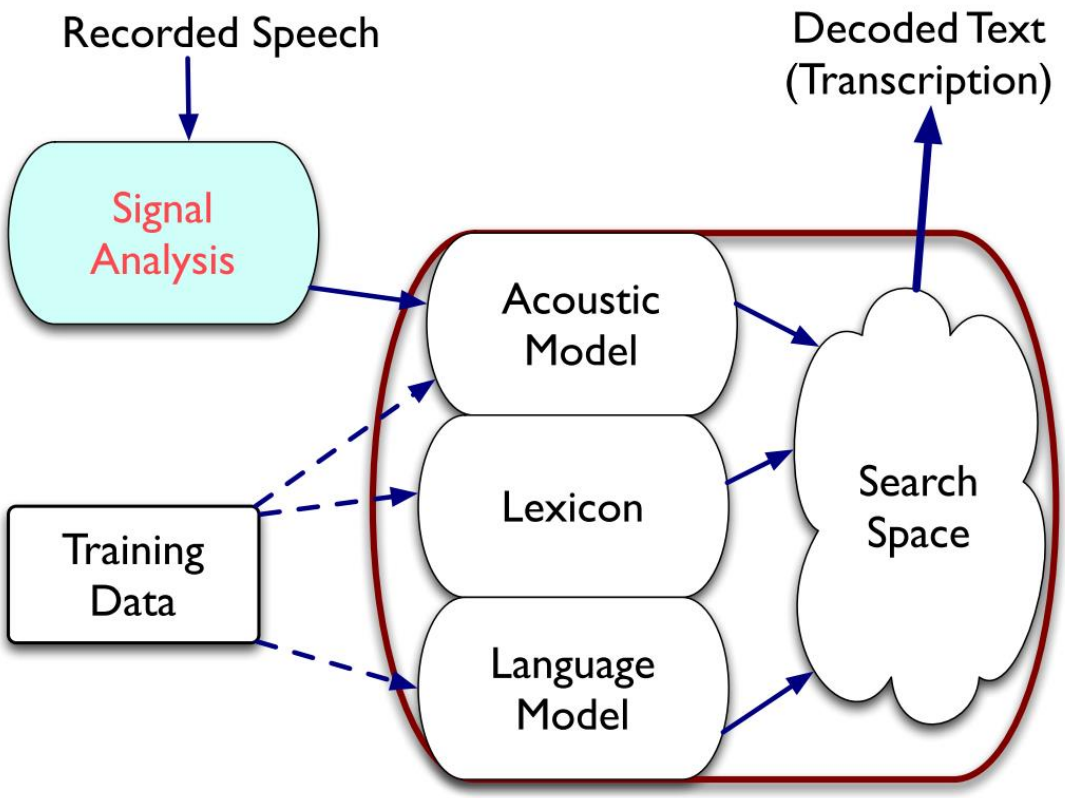
2020-4-5



KALDI 入门指南

从零学习 KALDI 基础知识

马洁锋

USTC-SPRAT

No. 443, Huangshan Road, Hefei, Anhui, 230027, China

Kaldi 入门指南

第一章　开篇简介

本书是用于记录个人在 kaldi 这一开源软件上的学习路径，主要的内容集中在 GMM-HMM 这一识别系统的原理及其在 kaldi 中的实现，另外附上一小节关于 kaldi 源码开发的基础知识，希望能给读者的 kaldi 学习之旅减轻一些负担。由于本书仅仅是个人的一些思考，缺漏之处在所难免，敬请各位指正。

本书的结构基于统计语音识别的基础公式展开，每一章对应其中的一部分内容，公式如下：

若X代表观测向量序列，W代表词序列，则产生X的最可能词序列W^*为：

$$W^* = \underset{w}{arg\,max}\, P(W|X)$$

对上式应用贝叶斯公式：

$$\begin{aligned} W^* &= \underset{w}{arg\,max}\, P(W|X) \\ &= \underset{w}{arg\,max} \frac{P(X|W)P(W)}{P(X)} \\ &= \underset{w}{arg\,max}\, P(X|W)P(W) \end{aligned}$$

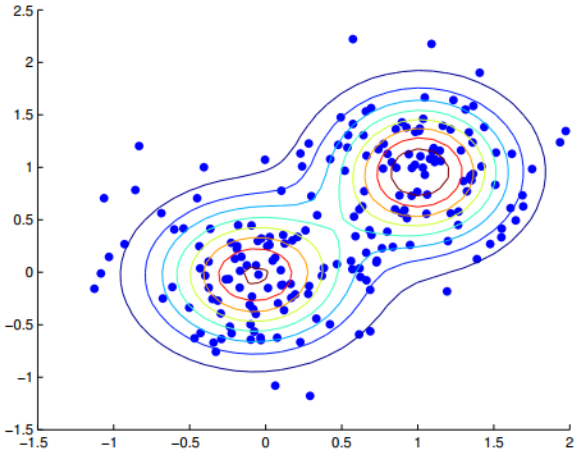
其中$P(X|W)$就代表着声学模型，$P(W)$代表着语言模型。

之后的章节与上述公式的对应关系【待补充，毕设结束之后补上】

第二章　单音素训练基本原理

一、　GMM 模型

(一)　基本概念



首先，我们考虑一个模式识别中的问题：如何描述某一类模式的观测向量的分布情况，也即第一章提及的声学模型$P(x|w)$。在传统方法中，应用最多的便是高斯模型。由于观测向量的维度通常都大于一，且需要多个高斯分布来拟合一类模式，所以一般情况下都采用高斯混合模型，也即 GMM：

$$\mathrm{p}(\mathbf{x}) = \sum_{\mathrm{m}=1}^{\mathrm{M}} \mathrm{P}(\mathrm{m})\mathrm{p}(\mathbf{x}|\mathrm{m}) = \sum_{\mathrm{m}=1}^{\mathrm{M}} \mathrm{P}(\mathrm{m})\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{\mathrm{m}}, \sigma_{\mathrm{m}}^2\mathbf{I})$$

上式表示一共有 M 个高斯模型构成的混合高斯模型，且假定 GMM 各维度不相关，GMM 中的每个 gauss-model 退化为 diagonal-gauss-model，此时协方差矩阵可以写作$\sigma_m^2\boldsymbol{I}$，其中$\boldsymbol{I}$为单位矩阵。

(二)　EM 参数估计

下面考虑一个 GMM

我们将对应到其的所有特征向量放到一起，组成序列$(o_1, o_2, \ldots, o_N)$，N 表示对应到这个 pdf-id 的特征向量共有 N 个，利用 EM 算法更新参数：

当前已知参数$\theta_m = (m = 1, 2, \ldots M) = (\mu_m, \sigma_m^2)$（即 GMM 的每个分量的均值和方差）以及每个分量当前所占的权重$P(m)$，M 表示此 GMM 共有 M 个分量，m 表示第 m 个分量。

（1）E-Step：根据当前模型参数（θ_m）计算出后验：$p(m|o_n)$

那么如何计算出这个后验呢，很简单，首先根据当前参数θ_m计算出每个分量得到任意观察特征的概率$p(o_n|m)$，然后根据贝叶斯公式就可以得到后验：

$$p(m|o_n)=\frac{p(o_n|m)P(m)}{p(o_n)}=\frac{p(o_n|m)P(m)}{\sum_{m'=1}^{M}p(o_n|m')P(m')}$$

（2）M-Step：计算新一轮迭代的模型参数

$$P(m)=\frac{\sum_{n=1}^{N}p(m|o_n)}{M}\qquad\mu_m=\frac{\sum_{n=1}^{N}p(m|o_n)o_n}{\sum_{n=1}^{N}p(m|o_n)}$$

$$\sigma_m^2=\frac{\sum_{n=1}^{N}p(m|o_n)\|o_n-\mu_m\|^2}{\sum_{n=1}^{N}p(m|o_n)}=\frac{\sum_{n=1}^{N}p(m|o_n){o_n}^2}{\sum_{n=1}^{N}p(m|o_n)}-{\mu_m}^2$$

二、 HMM 模型

(一) 基本概念

待完善，详细内容请参考 52NLP 相关博文

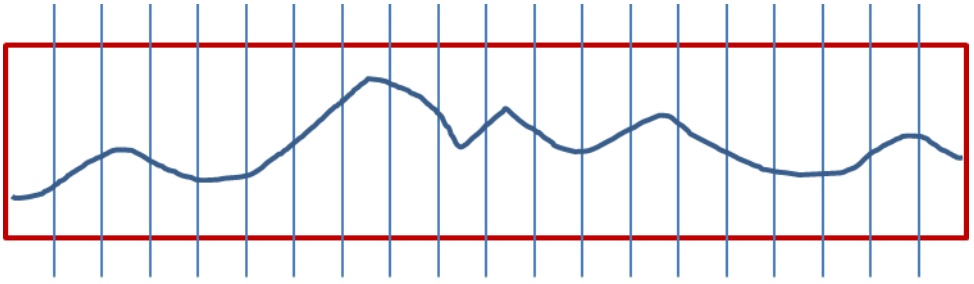
(二) HMM 三任务

待完善，详细内容请参考 52NLP 相关博文

三、 GMM-HMM 训练流程

这里以音频为例进行说明，文本行训练时与此类似，不同之处在于文本训练时一个文本只对应一个音素，一个音素常用 3/5 状态 HMM 建模。

语音 GMM-HMM 整体训练流程从下面这段音频说起。



假设我们有一段如上图所示的 2 秒的音频，其对应的文本为 speech。首先我们对其分帧，以帧长 100ms，帧移 100ms 为例进行切分，这样上面的音频就被我们分成 20 帧。(这里主要是为了方便说明，正常情况下帧长是 25ms，帧移是 10ms)

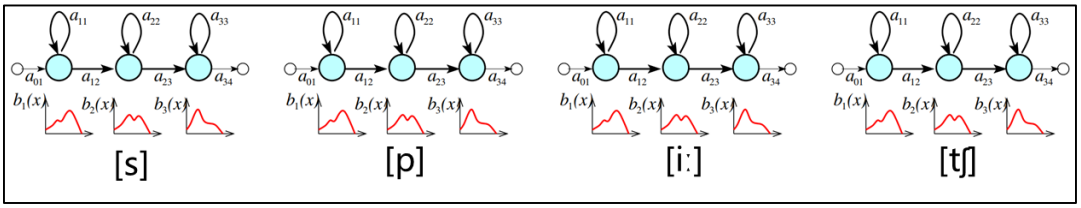
一、得到这段音频的标注，这里的标注可以是两种：

1、音素级别的标注。这是训练 HMM-GMM 模型所需要的标注，也就是说可以直接拿来训练的数据（每一帧对应哪一个音素的哪一个隐状态）。

2、字级别的标注。实际上我们更多时候拿到是的字级别的标注，因为音素级别的标注工作量很大。但我们可以通过发音词典，将字级别的标注转换成音素级别的标注。这种转换没办法消除多音字的影响，但实际上不会对结果产生太大影响。

比方说这里我们知道这段音频的文本是 speech，从 kaldi 的 lexicon 文件（需要自己准备，针对不同的任务有不一样的 lexicon）中找到对应的音素是[s] [p] [i:] [tʃ]。

二、对[s] [p] [i:] [tʃ]这 4 个声韵母分别使用 HMM-GMM 建模，使用 3 状态建模（OCR 常为 5 状态），如下图



为了方面说明，我们再给每个有发射概率的状态加上编号（每个音素还有起始状态、收束状态这两个没有发射概率的隐状态），从 1 开始一直到 12。

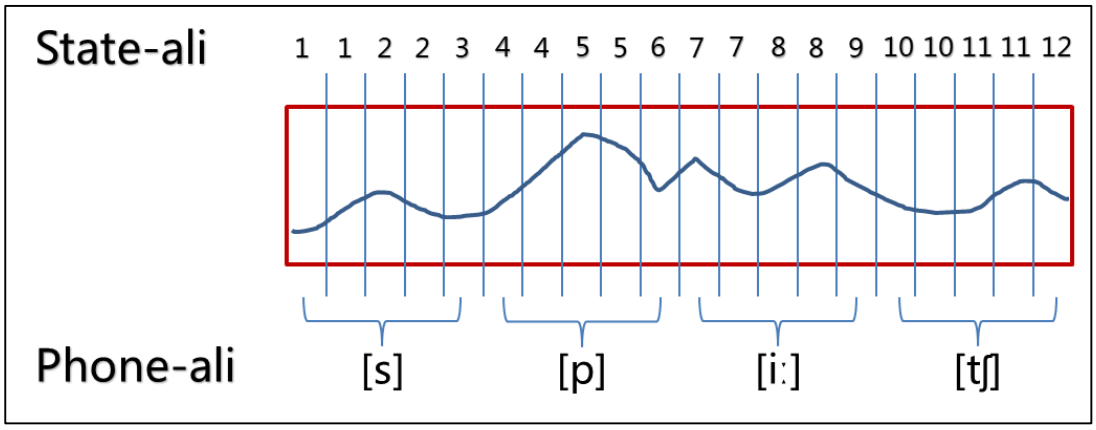
理解自环很重要，因为有了自环，它可以出现多次，因此我们可以对任意长的音频建模，这也是连续语音识别的基础。

三、对[s] [p] [i:] [tʃ]对应的 GMM-HMM 模型进行训练，得到模型参数

训练流程：

step1：初始化对齐。以上面的 20 秒音频为例，因为我们不知道每一帧对应哪个声韵母的某个状态，所以就均分。也就是说 1-5 帧对应[s]，6-10 帧对应[p]，11-15 帧对应[i:]，16-20 帧对应[tʃ]。同时[s]又有三个状态，那么就把 1-2 帧分给状态①，3-4 帧分给状态②，第 5 帧分给状态③。[p] [i:] [tʃ]亦如此。

初始化完成后，该段音频对应的 HMM 模型如下：



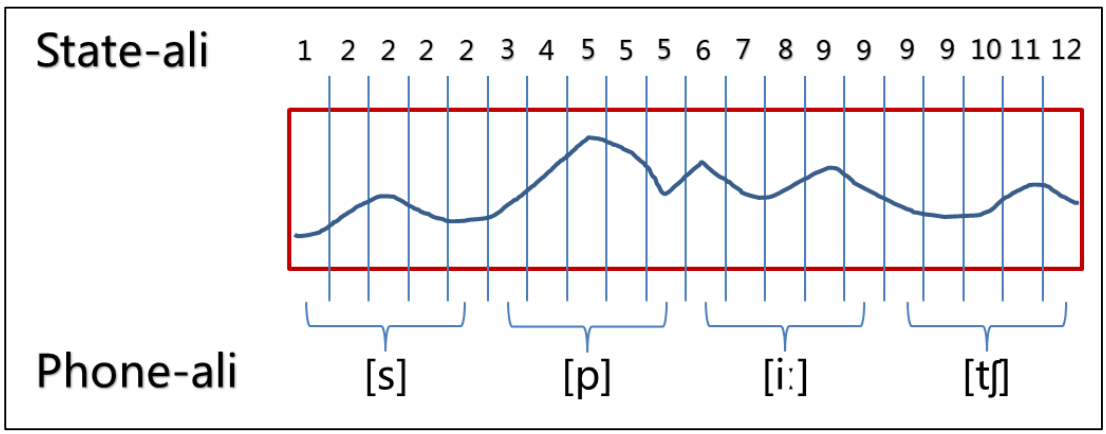
step2：更新模型参数。

1、转移概率：通过上图，我们可以得到①->①的转移次数，①->②的转移次数等等。然后除以总的转移次数，就可以得到每种转移的概率，这是一个统计的过程（MLE）。

2、发射概率：若用单高斯来解释，即均值和方差。以状态①的均值和方差为例，由上图我们可以知道第 1 帧和第 2 帧对应状态①。假设第 1 帧的 MFCC 特征是(4,3)，第 2 帧的 MFCC 特征的 MFCC 特征是(4,7)。那么状态①的均值就是(4,5)，方差是(0,8)

step3：重新对齐。根据 step2 得到的参数，重新对音频进行状态级别的对齐。这一步区别于 step1 的初始化，step1 的初始化我们是采用粗暴的均匀对齐，而这一步的对齐是根据 step2 的参数进行对齐的。这里的对齐方法有两种：a、硬对齐：采用维特比算法（kaldi 中为维特比训练）；b、软对齐：采用前后向算法。

经过重新对齐后，可能变成这样(第 2 帧已经不对应状态①了)：



step4：重复 step2 和 step3 多次，直到收敛。

第三章　数据说明

一、　字典

类别	文件	格式
字典文件 用于描述每个单字/音素的属性等，路径如下 /data/local/dict_nosp 由 prepare_dict.sh 创建	lexicon.txt	字典文件，格式为： Word phone1 … phoneN 此任务为 w1 UC0000 sp
	lexiconp.txt	带概率的字典文件，格式为： Word prob phone1 … phoneN 此任务为 w1 1.0 UC0000 sp
	nonsilence_phones.txt	非静音音素，每行代表一组相同发音、但可能声调不同的音素： a a1 a2 a3 此任务为 UCXXXX，因为文字识别中每个文本都不一样
	optional_silence.txt	包含一个单独的用作词典中的默认静音音素（sil）
	silence_phones.txt	静音音素 此任务为 sil\sp\spn 三种
	extra_questions.txt	用于决策树的构建，可以为空 此任务中为空，之后的状态绑定通过最大化熵来生成决策树
字典临时目录文件 用于描述每个单字/音素的属性等，路径如下 /data/local/lang_nosp	align_lexicon.txt	与 lexicon.txt 相同
	lex_ndisambig	存放有歧义的文本/发音
	lexiconp.txt	与上面相同
	lexiconp_disambig.txt	在 lexiconp.txt 中加入 disambig symbols（同音词后的消歧符，#1、#2 等）
	phone_map.txt	内容为音素 音素对 spn spn\UCXXXX UCXXXX
	phones	列出所有文本/音素，每行一个 此任务为 spn\UCXXXX

二、 语言模型

类别	文件	格式
语言模型（不是 N-gram 模型），用于给出映射、拓扑等，路径如下 /data/lang_nosp 由 prepare_lang.sh 根据 /data/local/dict 创建 2.1 节字典中的 /data/local/lang_nosp 是这个脚本运行中的临时目录	phones.txt	音素到标号(0 开始的整数)的映射 格式为：<eps> 0\UC0000 4
	word.txt	词到标号(0 开始的整数)的映射，格式为<eps> 0\w1 3
	oov.txt	只包含一行内容<UNK>，它不是一个特定词，重要的是这个词要有一个发音(由 lexicon.txt 指定)，称为“垃圾音素”，这个音素将与各种噪音对齐。Kaldi 中 lexicon.txt 有<unk> spn，将<unk>对应到 spn
	oov.int	包括上述内容的 int 形式（从 words.txt 中提取），此处为 1。这里特别说明：
		<UNK> sil wxxxx 为 word(词) spn sil UCXXXX 为 phone(音素) 分别由 word.txt phone.txt 给出映射
	topo	初始的拓扑结构，实际上是一个<TopologyEntry>，为<HmmState>向量，注：
		typedef std::vector<HmmState> TopologyEntry; 其中<HmmState>包含了某一状态的两个 pdf_id、一组<int32, BaseFloat>向量，详见下文
	L.fst	音素词典的 fst（有限状态自动机）表示 作用是：将音素序列转化为词序列
	L_disambig.fst	为了消歧(disambiguation)而引入的模型，表述为 lexicon with disambiguation symbols，用于鉴别同音词(red、read)和同一词的不同形式(cat、cats)
音素文件 用于描述每个单字/音素的属性等，路径如下 /data/lang_nosp/phones	文件夹内包含关于音素集不同信息的文件，这些文件大多数都有三个不同的版本，分别是.txt .int 和 .csl 版本，所含的信息相同。.txt 中是音素或者单词(sil,sp)，.int 中是对应的标号(1,2)，.csl 中是冒号分隔的列表	
	align_lexicon	词-音素映射表，内容为 w1 w1 UC0000 sp
	context_indep	包含非真实音素，sil、sp 等，不需要决策树提取上下文
	silence	空格/静音/噪声音素列表，与上一文件相同
	nonsilence	非静音/噪声音素的列表，与上一文件互斥
	disambig	表示消歧符号，#1 #2 等
	optional_silence	可任意在词之间出现的单音素(sil)，须在 phones 中指定
	extra_questions	除了自动生成之外的问题，用作聚类，可以为空
	roots	用 shared 和 split 修饰（下文解释作用）的音素，
	sets	音素列表，相同的音素被放在同一行

关于 data/lang_nosp_0gram 文件夹：

prepare_lm.sh 会通过 data/lang_nosp 生成 data/lang_nosp_0gram，两者相比多出了 G.fst，相当于复制一遍并创建 G.fst 文件

这里说明一下，L.fst 是由 lexicon 等文件创建的，目的是将音素序列转换为文本序列，G.fst 是结合 language model 创建的，是转化为 fst 格式的语言模型。

Shared/split 修饰符解释：

kaldi 官网的说明：

- "shared" or "not-shared" says whether or not there should be separate roots for each of the **pdf-classes** (i.e. HMM-states, in the typical case), or if the roots should be shared. If it says "shared" there will be a single tree-root for all HMM states (e.g. all three states, in a normal topology) ; if "not-shared" there would be (e.g.) three tree-roots, one for each pdf-class.
- "split" or "not-split" says whether or not the decision tree splitting should actually be done for the roots in question (for silence, we typically don't split). If the line says "split" (the normal case) then we do the decision tree splitting. If it says "not-split" then no splitting is done and the roots are left un-split.

开头为 shared split 的部分，表示同一行元素共享一个树根，允许进行决策树分裂

三、 训练、测试数据

类别	文件	格式
数据索引及 ground truth 路径如下 /data-dnn/train	feats.scp	存储所有训练样本中提取出来特征的存放路径： 0392-f_8BF8 ../fea_train_ark/fea_train.ark:11 :11 表示从该文件的第 11 个字符位（注意是字符位而不是行数）开始存储样本特征
	cmvn.scp	包含倒谱均值和方差规整的数据，由 speaker_id 索引 0392-f ../fea_train_ark/cmvn_train.ark:7
	spk2utt	列出书写者/说话人对应的所有样本： 0392-f 0392-f_8BF8 0392-f_0033 … 一行对应一个书写者
	utt2spk	列出每个样本对应的书写者/说话人： 0392-f_8BF8 0392-f \ 0637-f_8618 0637-f
	text	每个样本的词级标注： 0392-f_8BF8 sil w4461 sil
数据特征文件 路径如下 /fea_train_ark	fea_train.ark（29G）	存放从所有样本中提取出来的特征矩阵： 392-f_8BF8 [1.165771 … … …] 矩阵中每行代表一个特征向量，向量维度即为特征维度，行数代表样本帧数（num of frames）
	cmvn_train.ark（1.7M）	存放所有说话人相关的特征（均值、方差）： 0392-f [15812.38 … 1304046.9 …] 矩阵共两行，分别代表均值、方差向量
	运用 copy-feats ark:xxx.ark ark,t:|head -50(|awk {'print NF'})命令可以查看对应 xxx.ark 文件的信息（行数），‘,t’选项表示以文本形式输出。 在 7000 类这一任务中： 不难发现样本的每一帧提取出来的特征为 50 维，书写者均值和方差为 51 维。 此处用 awk 命令显示的是字段数 NF，注意方差对应的行还有‘]’，故会显示为 52	

四、 中间数据

(一) .mdl 文件

针对 GMM-HMM 任务，.mdl 文件中包含了 TransitionModel 和 AmDiagGmm 的相关信息，可以使用$gmm-copy --binary=false 40.mdl - 命令查看。

此处以 40.mdl 为例说明文件格式

首先是一个 TransitionModel（转移模型），里面包含了 TopoLogy（拓扑信息），对应的 ForPhones 表示包含的音素。

这里包含了所有的音素 ID，音素 4~7360 表示正常文本，共享相同的拓扑结构，里面有五种发射状态，每个都有自环和下一个状态，还包括了最后一个不发射状态（state 5 没有 pdf，没有转移），音素 1、2、3 表示空格、静音等。

```
<TransitionModel>
<Topology>
<TopologyEntry>
<ForPhones>
4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
83 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 51
937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 9
2 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335
5 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698
8 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061
1 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424
4 2765 2766 2767 2768 2769 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2783 2784 2785 2786 2787
7 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138 3139 3140 3141 3142 3143 3144 3145 3146 3147 3148 3149 3150
0 3491 3492 3493 3494 3495 3496 3497 3498 3499 3500 3501 3502 3503 3504 3505 3506 3507 3508 3509 3510 3511 3512 3513
3 3854 3855 3856 3857 3858 3859 3860 3861 3862 3863 3864 3865 3866 3867 3868 3869 3870 3871 3872 3873 3874 3875 3876
6 4217 4218 4219 4220 4221 4222 4223 4224 4225 4226 4227 4228 4229 4230 4231 4232 4233 4234 4235 4236 4237 4238 4239
9 4580 4581 4582 4583 4584 4585 4586 4587 4588 4589 4590 4591 4592 4593 4594 4595 4596 4597 4598 4599 4600 4601 4602
2 4943 4944 4945 4946 4947 4948 4949 4950 4951 4952 4953 4954 4955 4956 4957 4958 4959 4960 4961 4962 4963 4964 4965
5 5306 5307 5308 5309 5310 5311 5312 5313 5314 5315 5316 5317 5318 5319 5320 5321 5322 5323 5324 5325 5326 5327 5328
8 5669 5670 5671 5672 5673 5674 5675 5676 5677 5678 5679 5680 5681 5682 5683 5684 5685 5686 5687 5688 5689 5690 5691
1 6032 6033 6034 6035 6036 6037 6038 6039 6040 6041 6042 6043 6044 6045 6046 6047 6048 6049 6050 6051 6052 6053 6054
4 6395 6396 6397 6398 6399 6400 6401 6402 6403 6404 6405 6406 6407 6408 6409 6410 6411 6412 6413 6414 6415 6416 6417
7 6758 6759 6760 6761 6762 6763 6764 6765 6766 6767 6768 6769 6770 6771 6772 6773 6774 6775 6776 6777 6778 6779 6780
0 7121 7122 7123 7124 7125 7126 7127 7128 7129 7130 7131 7132 7133 7134 7135 7136 7137 7138 7139 7140 7141 7142 7143
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.75 <Transition> 1 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.75 <Transition> 2 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.75 <Transition> 3 0.25 </State>
<State> 3 <PdfClass> 3 <Transition> 3 0.75 <Transition> 4 0.25 </State>
<State> 4 <PdfClass> 4 <Transition> 4 0.75 <Transition> 5 0.25 </State>
<State> 5 </State>
</TopologyEntry>
<TopologyEntry>
<ForPhones>
1 2 3
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.25 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 <Transition> 4 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 <Transition> 4 0.25 </State>
<State> 3 <PdfClass> 3 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 <Transition> 4 0.25 </State>
<State> 4 <PdfClass> 4 <Transition> 4 0.75 <Transition> 5 0.25 </State>
<State> 5 </State>
</TopologyEntry>
</Topology>
<Triples> 36800
1 0 0
1 1 1
1 2 2
1 3 3
1 4 4
2 0 5
```

```
7359 4 36794
7360 0 36795
7360 1 36796
7360 2 36797
7360 3 36798
7360 4 36799
</Triples>
<LogProbs>
 [ 0 -1.063394 -0.4545648 -4.60517 -4.60517 -0.7257204 -4.60517 -4.60517 -0.70
-1.481364 -0.2703681 -1.440111 -0.3506569 -1.218158 -0.308903 -1.325207 -0.240
327393 -0.7545279 -0.6353171 -0.6621695 -0.7251154 -0.4016964 -1.106192 -0.134
0.8472978 -0.4510175 -1.013296 -0.3058837 -1.333597 -0.1426375 -2.01792 -0.329
86438 -0.3027309 -1.342461 -0.1499539 -1.971468 -0.3005402 -1.348683 -0.446136
 -0.1831512 -1.787621 -0.3426442 -1.237498 -0.4588336 -0.9997278 -0.4569306 -1
474 -1.539034 -0.3322546 -1.263385 -0.3639548 -1.18719 -0.3099841 -1.322226 -0
-1.258557 -0.2605004 -1.472575 -0.1794232 -1.806379 -0.2399555 -1.544882 -0.26
9914 -0.2882443 -1.384609 -0.1833486 -1.78664 -0.3443576 -1.233317 -0.4504625
0.1711553 -1.849541 -0.4518103 -1.011907 -0.5775 -0.8239391 -0.4761449 -0.9706
3 -0.4791468 -0.9657739 -0.4901578 -0.9481161 -0.3934489 -1.123087 -0.3387204
```

<Triples>三元组第一列代表的是单音素的序号，第二列代表的是第一列音素的三个状态的其中一个状态，第三列代表的是 pdf，也就是概率分布函数的序号。

< LogProbs >是和 transition-id 对应起来的，描述了转移概率

```
5512 -0.3250197 -1.281981 -0.4717081 -0.977995 -0.3378435 -1.249343 -0.2622772 -1
2214 -0.4342441 -1.043426 -0.3764174 -1.159369 -0.3315601 -1.26515 -0.3975255 -1.
6412 -0.5153711 -0.9095111 -0.3263636 -1.278491 -0.4025353 -1.104498 -0.6198645 -
031009 -0.7921115 -0.3077336 -1.328445 -0.3338091 -1.259452 -0.4720258 -0.9774679
</LogProbs>
</TransitionModel>
<DIMENSION> 50 <NUMPDFS> 36800 <DiagGMM>
<GCONSTS>  [ -144.1552 -213.181 -130.6774 -888.9985 -90.49625 -58.31097 -228.8385
.103 -588.4317 -666.7811 -358.3106 -626.571 -1091.121 -751.8043 -899.475 -1043.58
9.0189 -935.5704 -919.9905 -662.601 -955.7927 -764.7581 -474.9196 -474.2731 -726.
<WEIGHTS>  [ 0.002043522 0.002704484 0.002578113 0.001902172 0.002135136 0.002388
02067026 0.001456 0.00251559 0.003452889 0.001335478 0.003164976 0.00270064 0.002
 0.002407198 0.001696089 0.002595083 0.001669192 0.002074874 0.001963483 0.002406
<MEANS_INVVARS>  [
  129.1592 -22.68469 9.538078 -19.28741 -3.642691 -5.301662 -0.003849551 -12.4951
  163.3862 -18.5247 21.11556 1.81296 25.22927 7.634031 11.31646 2.295114 8.719065
  131.867 -13.79254 18.79878 26.65744 20.30554 3.795197 -1.436297 4.144204 3.2454
  1160.41 14.39625 -14.4322 36.23379 30.5767 87.73178 -4.095012 -309.8751 168.920
  126.0064 -13.81143 8.252462 6.439684 18.97522 16.14955 -15.0484 -17.29918 7.003
```

可以看出一共有 36800=7360×5 个 PDF，也就是 GMM，每个 GM 有 50 维

```
jfma@sprat-198:/disk1/jfma/gmm_dnn_cnn_hmm/7000_类_no_lm_egsgmm-copy --binary=false exp/mono_5/40.mdl -|grep LogProbs -C 10|awk {'print NF'}
gmm-copy --binary=false exp/mono_5/40.mdl -
3
3
3
3
3
3
3
3
3
1
1
73627 ← Logprobs的个数为73627-2([ ])-1(0)=73624
1
1
5
453 ← GCONSTS一行的文本数，维度为453-3(文本+[ ])=450
453 ← WEIGHTS
2
50 ← MEANS_INVVARS的维度
50
```

PDF 和 state 的对应关系见下，这里 Transition-state 可以理解为每个 phone 的各个状态，Transition-id 可以理解为每条弧的标号，从这里可以看出 sil、spn、sp 的状态 0~3 有 4 个转移弧，状态 4 有 2 个转移弧，其余的 7357 个音素各有 5 个状态，每个状态有 2 个转移弧（自转、跳到下一状态），故总的弧数为

3×（4×4+1×2）+7357×（5×2）＝73624

这与上面提到的 LogProbs 总数一致。

具体内容见 TM 一节。

```
show-transitions ./graph_0gram_true/phones.txt 40.mdl
Transition-state 1: phone = sil hmm-state = 0 pdf = 0
 Transition-id = 1 p = 0.345282 [self-loop]
 Transition-id = 2 p = 0.634724 [0 -> 1]
 Transition-id = 3 p = 0.01 [0 -> 2]
 Transition-id = 4 p = 0.01 [0 -> 3]
Transition-state 2: phone = sil hmm-state = 1 pdf = 1
 Transition-id = 5 p = 0.483976 [self-loop]
 Transition-id = 6 p = 0.01 [1 -> 2]
 Transition-id = 7 p = 0.01 [1 -> 3]
 Transition-id = 8 p = 0.496032 [1 -> 4]
Transition-state 3: phone = sil hmm-state = 2 pdf = 2
 Transition-id = 9 p = 0.01 [2 -> 1]
 Transition-id = 10 p = 0.776637 [self-loop]
 Transition-id = 11 p = 0.0139951 [2 -> 3]
 Transition-id = 12 p = 0.199369 [2 -> 4]
Transition-state 4: phone = sil hmm-state = 3 pdf = 3
 Transition-id = 13 p = 0.0210316 [3 -> 1]
 Transition-id = 14 p = 0.01 [3 -> 2]
 Transition-id = 15 p = 0.920299 [self-loop]
 Transition-id = 16 p = 0.0486701 [3 -> 4]
Transition-state 5: phone = sil hmm-state = 4 pdf = 4
 Transition-id = 17 p = 0.489119 [self-loop]
 Transition-id = 18 p = 0.510881 [4 -> 5]
Transition-state 6: phone = sp hmm-state = 0 pdf = 5
 Transition-id = 19 p = 0.199917 [self-loop]
 Transition-id = 20 p = 0.0109952 [0 -> 1]
 Transition-id = 21 p = 0.773204 [0 -> 2]
 Transition-id = 22 p = 0.015883 [0 -> 3]
Transition-state 7: phone = sp hmm-state = 1 pdf = 6
```

```
show-transitions ./graph_0gram_true/phones.txt 40.mdl
 Transition-id = 73618 p = 0.376263 [1 -> 2]
Transition-state 36798: phone = UC8489 hmm-state = 2 pdf = 36797
 Transition-id = 73619 p = 0.533646 [self-loop]
 Transition-id = 73620 p = 0.466354 [2 -> 3]
Transition-state 36799: phone = UC8489 hmm-state = 3 pdf = 36798
 Transition-id = 73621 p = 0.555224 [self-loop]
 Transition-id = 73622 p = 0.444776 [3 -> 4]
Transition-state 36800: phone = UC8489 hmm-state = 4 pdf = 36799
 Transition-id = 73623 p = 0.695918 [self-loop]
 Transition-id = 73624 p = 0.304082 [4 -> 5]
jfma@sprat-198:/disk1/jfma/gmm_dnn_cnn_hmm/7000_类_no_lm_egs/exp/mono_5$
```

(二) tree 文件(EventMap)

```
jfma@sprat-198:/disk1/jfma/gmm_dnn_cnn_hmm/7000_类_no_lm_egs/exp/mono_5$ copy-tree --binary=false tree -|head -10
copy-tree --binary=false tree -
ContextDependency 1 0 ToPdf TE 0 7361 ( NULL TE -1 5 ( CE 0 CE 1 CE 2 CE 3 CE 4 )
TE -1 5 ( CE 5 CE 6 CE 7 CE 8 CE 9 )
TE -1 5 ( CE 10 CE 11 CE 12 CE 13 CE 14 )
TE -1 5 ( CE 15 CE 16 CE 17 CE 18 CE 19 )
TE -1 5 ( CE 20 CE 21 CE 22 CE 23 CE 24 )
TE -1 5 ( CE 25 CE 26 CE 27 CE 28 CE 29 )
TE -1 5 ( CE 30 CE 31 CE 32 CE 33 CE 34 )
TE -1 5 ( CE 35 CE 36 CE 37 CE 38 CE 39 )
TE -1 5 ( CE 40 CE 41 CE 42 CE 43 CE 44 )
TE -1 5 ( CE 45 CE 46 CE 47 CE 48 CE 49 )
jfma@sprat-198:/disk1/jfma/gmm_dnn_cnn_hmm/7000_类_no_lm_egs/exp/mono_5$ copy-tree --binary=false tree -|tail -10
copy-tree --binary=false tree -
LOG (copy-tree[5.5.76~1-535b]:main():copy-tree.cc:55) Copied tree
TE -1 5 ( CE 36760 CE 36761 CE 36762 CE 36763 CE 36764 )
TE -1 5 ( CE 36765 CE 36766 CE 36767 CE 36768 CE 36769 )
TE -1 5 ( CE 36770 CE 36771 CE 36772 CE 36773 CE 36774 )
TE -1 5 ( CE 36775 CE 36776 CE 36777 CE 36778 CE 36779 )
TE -1 5 ( CE 36780 CE 36781 CE 36782 CE 36783 CE 36784 )
TE -1 5 ( CE 36785 CE 36786 CE 36787 CE 36788 CE 36789 )
TE -1 5 ( CE 36790 CE 36791 CE 36792 CE 36793 CE 36794 )
TE -1 5 ( CE 36795 CE 36796 CE 36797 CE 36798 CE 36799 )
)
EndContextDependency jfma@sprat-198:/disk1/jfma/gmm_dnn_cnn_hmm/7000_类_no_lm_egs/exp/mono_5$
```

决策树文件开头一般会有这么几个词：ContextDependency X Y ToPdf

这里 X 表示上下文窗口的总长度为 X 个音素（X=3 表示包括上一个、这一个、下一个音素），Y 表示「这一个音素」的下标为 Y（下标从 0 起算）。如果一棵决策树是适用于上下文无关音素模型的，那么这两个数字就会是 1 和 0。

文件之后的内容，是递归保存了一棵决策树。树中的结点被称为 EventMap，有三种类型：Constant、Split、Table（还有一种空结点）。它们的递归定义如下：

```
EventMap := ConstantEventMap | SplitEventMap | TableEventMap | "NULL"

ConstantEventMap := "CE" <numeric pdf-id>

SplitEventMap := "SE" <key-to-split-on> "[" yes-value-list "]" "{" EventMap EventMap "}"

TableEventMap := "TE" <key-to-split-on> <table-size> "(" EventMapList ")"
```

Constant 结点就是叶子结点，它的表示方式就是 CE 加一个整数，这个整数表示 PDF（即高斯混合分布）的编号。

Split 结点表示有两个分支的结点，其格式为 SE <属性> [<值的列表>] { <是分支> <否分支> }。它针对一个属性提问，问题就是这个属性的值是否在一个列表中。属性可以是 0、1、2 这样的非负整数，表示针对上下文窗口中的第几个音素提问。比如在通常的三音素模型中，对属性 1 提问就是问「这一个音素」，对属性 0 提问就是问「上一个音素」。此时中括号中的整数就是音素的编号。属性还可以是 -1，表示问的是「当前状态是 HMM 中的第几个状态」，中括号中的整数就是若干状态的下标（从 0 起算）。大括号里面是两棵决策树，分别代表属性的值在或不在[中括号]内时，后续的决策树。

Table 结点表示有多个分支的结点，其格式为 TE <属性> <值的个数> (<每个值对应的决策树>)。属性值必须是从 0 开始的连续整数。如果某个值不可能取到，相应的决策树可以是 NULL。

Table 结点常用于属性是 -1 的情况，因为 HMM 状态的下标是从 0 开始的连续整数，而且每个状态往往有不同的 PDF。Split 结点则一般不用于属性是 -1 的情况。Split 结点中的「值的列表」，也就是问题涉及的音素集合，一般是对音素自动聚类获得的，而不是语言学家手动创建的。

(三) ali.*.gz 文件

```
jfma@sprat-198:/disk1/jfma/7000/exp/mono1007$ show-transitions ../../data/lang_nosp/phones.txt final.mdl |egrep -C10 "= 60711"
show-transitions ../../data/lang_nosp/phones.txt final.mdl
Transition-state 30341: phone = UC8618 hmm-state = 0 pdf = 30340
 Transition-id = 60705 p = 0.240409 [self-loop]
 Transition-id = 60706 p = 0.759591 [0 -> 1]
Transition-state 30342: phone = UC8618 hmm-state = 1 pdf = 30341
 Transition-id = 60707 p = 0.218421 [self-loop]
 Transition-id = 60708 p = 0.781579 [1 -> 2]
Transition-state 30343: phone = UC8618 hmm-state = 2 pdf = 30342
 Transition-id = 60709 p = 0.186301 [self-loop]
 Transition-id = 60710 p = 0.813699 [2 -> 3]
Transition-state 30344: phone = UC8618 hmm-state = 3 pdf = 30343
 Transition-id = 60711 p = 0.146552 [self-loop]
 Transition-id = 60712 p = 0.853448 [3 -> 4]
Transition-state 30345: phone = UC8618 hmm-state = 4 pdf = 30344
 Transition-id = 60713 p = 0.108108 [self-loop]
 Transition-id = 60714 p = 0.891892 [4 -> 5]
Transition-state 30346: phone = UC9097 hmm-state = 0 pdf = 30345
 Transition-id = 60715 p = 0.310023 [self-loop]
 Transition-id = 60716 p = 0.689977 [0 -> 1]
Transition-state 30347: phone = UC9097 hmm-state = 1 pdf = 30346
 Transition-id = 60717 p = 0.295238 [self-loop]
 Transition-id = 60718 p = 0.704762 [1 -> 2]
jfma@sprat-198:/disk1/jfma/7000/exp/mono1007$ copy-int-vector "ark:gunzip -c ali.1.gz |" ark,t:- |egrep "60711"
copy-int-vector 'ark:gunzip -c ali.1.gz |' ark,t:-
0637-f_8618 2 8 5 5 18 17 60706 60708 60707 60707 60710 60712 60711 60714 60713 21 30 36 35 2 1 8 5 5 18
0628-f_8618 2 8 5 5 5 18 60706 60705 60705 60705 60705 60708 60707 60707 60710 60709 60712 60711 60714 21 30 36 35 2 1 8 5 5 18
0776-f_8618 2 8 5 5 18 17 60706 60705 60705 60708 60707 60707 60710 60709 60712 60711 60714 60713 21 30 36 2 8 5 5 18
0507-f_8618 2 8 5 5 5 18 17 60706 60705 60705 60708 60707 60710 60712 60711 60714 60713 21 30 36 35 2 1 8 5 5 18
0699-f_8618 2 8 5 5 5 18 60706 60705 60705 60708 60707 60710 60709 60712 60711 60714 60713 60713 21 30 36 35 35 2 1 1 8 5 5 18
0728-f_8618 2 8 5 5 5 18 60706 60705 60705 60708 60707 60710 60709 60712 60711 60714 60713 60713 21 30 36 35 2 1 8 5 18
0566-f_8618 2 8 5 5 5 18 17 60706 60705 60705 60708 60707 60707 60710 60712 60711 60714 60713 21 30 36 35 35 2 1 8 5 5 5 18
LOG (copy-int-vector[5.5.76~1-535b]:main():copy-int-vector.cc:83) Copied 185111 vectors of int32.
jfma@sprat-198:/disk1/jfma/7000/exp/mono1007$
```

每个训练文件在 text 中有对应的音素列表（ground truth），ali 文件中存储的是该文件每一帧对应的 trans_id 序列而非 pdf_id 序列，所以 realignment 的时候需要进行维特比搜索的隐状态空间，就是由 sil UCXXXX sil 这三个音素对应的若干 trans_id 组成的隐状态空间。

五、 文件查看命令

文件	指令+说明
ark,scp	copy-feats ark:raw_mfcc.ark ark,t:-|head -10 ark 存的是二进制文件，后面加一个 ark,t:- 表示输出格式为文本 scp 可以直接用 cat 命令看。
fst	~/kaldi/tools/openfst-1.6.2/bin/fstprint L.fst | head -n 10 fstdraw [--isymbols=phones.txt --osymbols=words.txt] L.fst | dot -Tps | ps2pdf - L.pdf
mdl	~/kaldi/src/gmmbin/gmm-copy --binary=false 0.mdl - gmm-copy --binary=false 0.mdl (text_0.mdl) 将 0.mdl 内容转换成文本形式（并存入 text_0.mdl）
tree	~/kaldi/src/bin/copy-tree --binary=false tree -
ali.1.gz 对齐文件	copy-int-vector "ark:gunzip -c ali.1.gz|" ark,t:- | head -n 1 可以先解压 gunzip ali.1.gz 然后 ~/kaldi/src/bin/show-alignments ../../data/lang/phones.txt 40.mdl ark:ali.1 |head -n 2
raw	nnet3-info nnet3-copy --binary=false init.raw xx
ubm	gmm-global-copy --binary=false exp/diag_ubm/final.dubm xxx
vad	src/bin/copy-vector ark:vad_train.1.ark ark,t:- > vad_train1.txt copy-feats ark:raw_mfcc_train.1.ark ark,t:- > raw_mfcc_train1.txt nnet3-copy-egs 查看 egs.1.ark

第四章　脚本详解

一、　kaldi 命令行级 IO 机制

命令行 I/O 是指在 shell 上调用编译好的 Kaldi 工具的方法，理解 shell 管道这一概念是关键。

举个例子：

```
echo '[ 0 1 ]' | copy-matrix - -
echo '[ 0 1 ]' | copy-matrix --binary=false - -
```

第一句命令输出为二进制，第二句为文本格式，'--binary=false'选项表示转为文本形式

冒号后面的部分被解释为 wxfilename 或 rxfilename，这是 Kaldi 采用的一种扩展的文件名进行输入输出，意味着支持管道和标准输入/输出之类的东西

在 rspecifiers 中，"ark:-"意思是从标准输入读取数据作为存档，"scp:foo.scp"意思是脚本文件从 foo.scp 读取数据

在 rspecifiers 中，" ark,s,cs:-" 意思是当我们读取（此处指从标准输入）时，我们期望 key 被排序（,s），and we assert that they will be accessed in sorted order，（,cs）意味着我们知道程序将按排序顺序访问它们(如果这些条件不成立,程序将崩溃)。这允许 Kaldi 模拟随机访问而不会占用大量内存。

ark:-作为指令参数的一部分时，表示从输入流读数据或者将结果写入标准输出

(一)　非表格型 I/O

非表格型的 I/O 比较简单，一般是输入输出只包含一个或两个对象的文件或流（比如，声学模型文件，变换矩阵）。使用方式就是 shell 命令的使用方式。其中值得注意的有：

- Kaldi 采用一种扩展的文件名进行输入输出。读取文件的文件名称为 rxfilename ， 写入的文件名称为 wxfilename。

- 在 rxfilename/wxfilename 处采用 “-” 来表示标准输入输出。

- rxfilename/wxfilename 处可以使用管道命令，比如先用 gunzip 将压缩文件解压，再输入到 Kaldi 程序中，即可在文件输入路径处填入 “gunzip -c foo.gz|” 。

- rxfilename/wxfilename 后可以通过 “:” 来描述偏移量，如 “foo:1045” 表示从 foo 文件偏移 1045 个字节开始读取。

- 使用 --binary=true/false 来控制是否使用二进制输出。默认是 true。

- 有一组 copy* 命令，可以用来查看文件。如，copy-matrix –binary=false foo.mat -。

- log 信息和输出会混在一起显示，但是 log 信息是 stderr 上，所以不会传到管道中。可以通过添加 2>/dev/null 命令，将 log 信息重定向到 /dev/null。

(二) 表格型 I/O

对于一系列数据的集合，比如对应于每一句话的特征矩阵，或者每一条音频文件对应的路径，Kaldi 采用表格形式的数据表示。表格中，以没有空格的字符串为索引。Kaldi 中，称从表格文件中读取的一个字符串为 rspecifier，写入表格文件的一个字符串为 wspecifier。有两种表格文件格式：archive 和 script 。其读取方式和非表格型数据类似，也是使用 rxfilename/wxfilename 的格式，其格式为 [options]:path:[offset]。需要在 options 中注明是 archive 文件还是 script 文件。

例如：wav-to-duration scp:wav.scp ark,t:-

rspecifiers 和 wspecifier 是 Kaldi 中定义的两种术语，是以表格形式存在的文件的文件名，格式类似于 rxfilename/wxfilename，亦可以采用管道命令，对于 ark 文件，可以添加选项，如 ark,s,cs:-等。

两种存储格式:script-file 和 archive-file

script-file 是一种文件指针，里面包含的是具体文件所在路径，其格式为 key rspecifier|wspecifier

key 是字段名字，rspecifier/wspecifier 则是文件路径，rspecifier/wspecifier 可以有空格。例如 “utt1 gunzip -c /foo/bar/utt1.mat.gz” 。

archive-file 里面装的是实际的数据。其文件格式是 key1 object1 key2 object2 key3 object3 …

ark 文件可以连接在一起，依然是一个有效的 ark 文件。但是如果需要的文件时有序的话，连接的时候应该要注意。

ark 文件和 scp 文件有可能同时写，这时可以这么写：ark,scp:foo.ark,foo.scp。

二、 run_new.sh

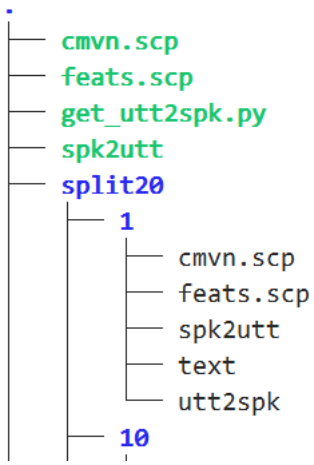
脚本名	详解
prepare_lang.sh	prepare_lang.sh --position-dependent-phones false \ --num-sil-states 5 --num-nonsil-states 5 \ data/local/dict_nosp "<unk>" data/local/lang_nosp data/lang_nosp
	作用： 构建字典 fst 模型（L.fst）、topo、phone、word 等
	IO：详细文件内容见 2.1、2.2 节 输入文件：data/local/dict_nosp <unk>：需出现在字典中，用于映射 OOV 单词到 data/lang_nosp/oov.txt 中间文件：data/local/lang_nosp 输出文件：data/lang_nosp
	参数： 1、--position-dependent-phones false 指不将 phone 更详细地拆分成开始、中间、结束、孤立等环节（分别用_B _I _E _S 表示） 2、--num-sil-states 5、--num-nonsil-states 5 指定所有音素（静音非静音）的 hmm 模型都有五状态
format_lm_sri.sh	LM=data/local/lm/lm.0gram.gz srilm_opts="-subset -prune-lowprobs -unk -tolower -order 1" format_lm_sri.sh --srilm-opts "$srilm_opts" data/lang_nosp $LM data/local/dict_nosp/lexicon.txt data/lang_nosp_0gram
	format_lm_sri.sh [options] <lang-dir> <arpa-LM> <out-dir> 将 ARPA 格式的语言模型转换成 FSTs
train_mono_no_delta.sh	dir=./exp/new_mono_5 train_data=./data-dnn/train lang=./data/lang_nosp train_mono_no_delta.sh --totgauss 1840000 --nj "$train_nj" \ --cmd "$train_cmd" $train_data $lang $dir
	作用： 单音素训练，得到 final.mdl，即 HMM 转移模型和 GMM 模型
	IO: 输入文件：$train_data 输入训练数据，$lang 输入语言模型 输出文件：$dir 输出模型数据（X.mdl）和一些参数文件（X_opts）
	参数： 1、--totgauss 1840000 指模型最终需要的总高斯数 2、--nj "$train_nj"指训练过程中的并行任务数（用以提高训练效率） 3、--cmd "$train_cmd"指定用何脚本进行 log 记录

三、 train_mono_no_delta.sh

详细的分析见脚本内的注释，下面就整体流程进行说明，同时对脚本中的关键命令进行详细解释。

流程：

1) split_data.sh 将训练数据（其实是 scp 索引和 groudtruth）根据任务数进行分解



如上图所示，data-dnn/train 中的索引和 text 被均匀拆分为 JOB（20）部分

2) gmm-init-mono 利用少量数据快速构造第一个模型文件(.mdl)和决策树(tree)
3) compile-train-graphs 根据决策树，模型，L.fst，text 等文件，构造训练用 FST。fsts.JOB.gz 中使用 key-value 的方式保存每个句子和其对应的 fst 网络，通过 key 就能找到这个句子的 fst 网络。
4) align-equal-compiled 对每个样本进行均匀对齐
5) gmm-acc-stats-ali 对 align 后的数据进行按 trans_id 汇总统计，获得中间统计量
6) gmm-est 根据上面得到的统计量，更新每个 GMM 模型

接下来重复 7~9，直到达到指定 num_iters (迭代轮数)数目

7) gmm-align-compiled 仅在指定 iterations 执行 force-alignment 操作
8) gmm-acc-stats-ali 对 align 后的数据进行按 trans_id 汇总统计，获得中间统计量
9) gmm-est 根据上面得到的统计量，更新每个 GMM 模型

第五章　源码详解

一、　TransitionModel 类

TransitionModel 是 kaldi 定义在 C++中的一个类

变量	表示	标号	含义
trans_state	int32	从 1 开始	可以理解为某一音素（HMM）中的某一隐状态 标号为 i 的 trans_state 对应 tuples_[i-1]
trans_index	int32	从 0 开始	表示 trans_state 的下一个转移状态（HMM 内的 index），在五状态任务中，其值是 0,1,2,3,4
trans_id	int32	从 1 开始	表示 HMM 中的一个转移过程，对所有的 HMM 状态的弧进行从 1 开始编号 和 pair(trans_state, trans_index)之间存在一一对应关系

trans_state 对应的 tuple(四元组):

- 这个状态在哪个音素下面？ phone
- 这个状态是这个音素下面的第几个？（0,1,2?） hmm-state
- 转移到下一个状态的pdf是什么？ 自旋的pdf是什么？ pdf-id(forward-pdf-id及self-loop-pdf-id)

此处的 pdf-id = forward-pdf-id = self-loop-pdf-id

```
show-transitions ./graph_0gram_true/phones.txt 40.mdl
 Transition-id = 73618 p = 0.376263 [1 -> 2]
Transition-state 36798: phone = UC8489 hmm-state = 2 pdf = 36797
 Transition-id = 73619 p = 0.533646 [self-loop]
 Transition-id = 73620 p = 0.466354 [2 -> 3]
Transition-state 36799: phone = UC8489 hmm-state = 3 pdf = 36798
 Transition-id = 73621 p = 0.555224 [self-loop]
 Transition-id = 73622 p = 0.444776 [3 -> 4]
Transition-state 36800: phone = UC8489 hmm-state = 4 pdf = 36799
 Transition-id = 73623 p = 0.695918 [self-loop]
 Transition-id = 73624 p = 0.304082 [4 -> 5]
jfma@sprat-198:/disk1/jfma/gmm_dnn_cnn_hmm/7000_类_no_lm_egs/exp/mono_5$
```

TransitionModel 类中的函数（映射关系）:

```
(phone, HMM-state, forward-pdf-id, self-loop-pdf-id) -> transition-state
(transition-state, transition-index)                 -> transition-id

同时也存在着反向的映射关系，即:
                    transition-id -> transition-state
                    transition-id -> transition-index
                transition-state -> phone
                transition-state -> HMM-state
                transition-state -> forward-pdf-id
                transition-state -> self-loop-pdf-id
```

TransitionModel 中的属性和函数：

变量	含义
HmmTopology topo_	拓扑关系
vector<Tuple> tuples_	四元组（phone、HMM-state、forward-pdf-id = self-loop-pdf-id）
vector<BaseFloat> log_probs_	转移概率，即每个 trans_id 的概率取 ln 后的数值， 假设有 N 个 trans_id，log_probs 里面会有 N+1 个值，其中第一个值是 0，之后就是和 trans_id 对应
Vector<BaseFloat> non_self_loop_log_probs_	trans_state 非自环的概率
int32 num_pdfs_	pdf_id 的总个数
vector<int32> state2id_	``` int32 TransitionModel::PairToTransitionId(int32 trans_state, int32 trans_index) const { KALDI_ASSERT(static_cast<size_t>(trans_state) <= tuples_.size()); KALDI_ASSERT(trans_index < state2id_[trans_state+1] - state2id_[trans_state]); return state2id_[trans_state] + trans_index; } ``` 用于得到一个 pair 对应的 trans_id
vector<int32> id2state_	``` int32 TransitionModel::TransitionIdToPhone(int32 trans_id) const { KALDI_ASSERT(trans_id != 0 && static_cast<size_t>(trans_id) < id2state_.size()); int32 trans_state = id2state_[trans_id]; return tuples_[trans_state-1].phone; } ``` 根据一个 trans_id 得到其对应的 trans_state
vector<int32> id2pdf_id_	``` inline int32 TransitionModel::TransitionIdToPdf(int32 trans_id) const { KALDI_ASSERT(static_cast<size_t>(trans_id) < id2pdf_id_.size() && "Likely graph/model mismatch (graph built from wrong model?)"); return id2pdf_id_[trans_id]; } ``` 给定一个 trans_id 得到其对应的 pdf_id，对应 pdf_id 一样的 hmm states 共享相同的发射概率，但是转移概率还是不同的

二、 AmDiagGmm 类

AmDiagGmm 也是 kaldi 定义在 C++中的一个类，和 TransitionModel 一起被写到.mdl 文件中。

下面介绍 AM 相关的概念，这些都是 C++中的类。

DiagGmm

保存一个 GMM 的参数，包括分量权值 weights_、均值、方差、每一分量高斯分布里的常量部分取 log 后的数值 gconsts_（gconst=2log(2pi^(D/2)×|*|^(1/2))，|*|的值就是 VARIANCE 下面 39 个方差的乘积，随着方差的更新而不断更新）。注意均值和方差为了方便计算，保存的并不是原原本本的方差、均值，而是方差每一元素求倒数后的 inv_vars_、均值乘以 inv_vars_后的 means_invvars_。

.mdl 中 DiagGmm 信息如下：

<DiagGMM>

<GCONSTS> [G_1 G_2 ⋯ G_N]

<WEIGHTS> [W_1 W_2 ⋯ W_N]

<MEANS_INVVARS> [

第一个单高斯分量的 means_invvars_1

……

means_invvars_N] // 每个分量的值占一行

<INV_VARS> [

第一个单高斯分量的 inv_vars_1

……

inv_vars_N] // 每个分量的值占一行

</DiagGMM>

有多少个不同的 pdf-id（即.mdl 文件中的 NUMPDFS 值）就会有多少个这样的 DiagGmm。

DiagGmmNormal

对应于 DiagGmm 的标准形式的 GMM，保存一个 GMM 原原本本的参数：分量权重 weights_，均值 means_，方差 vars_

AmDiagGmm（简记为 AM）

保存所有的 GMM，可以简单的理解成一个向量，每个元素都是一个 DiagGmm。

AccumDiagGmm

对应于 DiagGmm，保存参数更新时（就是 EM 算法更新 GMM）所需的累积量，包括：

名称	含义
Num_comp_	混合分量个数 M，即单高斯的个数
Dim_	特征维数
Occupancy_	M 个元素，每一个元素是 $\sum_{n=1}^{N} p(m\|o_n)$，表示 N 个样本分别由 M 个分量产生的概率和（responsibilities for N samples）
Mean_accumulator_	M×D 维，每一行是 $\sum_{n=1}^{N} p(m\|o_n)o_n$，计算均值的中间变量
Variance_accumulator_	M×D 维，每一行是 $\sum_{n=1}^{N} p(m\|o_n)o_n^2$，计算方差的中间变量

它们其实就是一些统计量的和，有了这几个值我们就可以根据之前所说的 EM 算法来更新参数了。

例如在单音素训练中：

（1）gmm-acc-stats-ali 操作如下：

对于每一帧的特征和其对齐（transition-id）：

a、对于转移模型（TM），累积 tid 出现的次数，即某个 pdf-id 对应的总的特征向量个数 N；

b、对于 AM，由 tid 得到 pdf-id，也就是找到对应该 pdf-id 的 DiagGmm 对象，更新与该 DiagGmm 对象相关的 AccumDiagGmm 的参数，也就是计算得到三个 GMM 参数更新公式的分子部分(包括每一混合分量的后验（occupancy_）、每一分量的后验乘以当前帧的特征(mean_accumulator_，M×D 维)、每一分量的后验乘以当前帧的特征每一维的平方(variance_accumulator_，M×D 维))

处理完所有数据后，将 TM 和 AM 的累积量写到一个文件中：x.JOB.acc 中

（2）gmm-sum-accs

gmm-acc-stats-ali 生成的累计量分散在 JOB 个文件中，该程序将分散的对应同一 trans-id、pdf-id 的累计量合并在一起。

（3）gmm-est 主要分两部分，一部分更新 TransitionModel，一部分更新 GMM

a、更新转移模型：根据 gmm-acc-stats-ali 统计的 tid 出现的次数，做一个除法就可以更新转移概率矩阵 A。

b、更新 GMM：gmm-acc-stats-ali 已经得到了三个 GMM 参数更新公式的分子部分，方差累积量只需要减去更新后的均值的平方即可得到正确的方差更新公式。分母部分也已几乎得到，做一个简单的除法就可以更新 GMM 的分量概率、均值、方差。调用 MleAmDiagGmmUpdate()更新 GMM 的参数，然后在该函数里调用 MleDiaGmmUpdate()更新每一个 GMM 的参数，然后在后一个函数里更新每个 GMM 的每一个分量的参数。

AccumAmDiagGmm

保存所有的 AccumDiagGmm，可以把它看做由 AccumDiagGmm 组成的一个向量。

TransitionModel　简记为 TM

保存 HMM 拓扑，log 转移概率，transition-id、transition-state、triplets(phone, hmm-state, forward-pdf)等之间的映射。具体见 TransitionModel

我们在文件.mdl 中存储的信息即为 TM 和 AM，其余的信息都可以在具体要用到的时候根据这两者计算出来。

三、 训练流程

(一) 脚本

train_mono.sh

1、Usage: steps/train_mono.sh [options] <data-dir> <lang-dir> <exp-dir>

2、e.g.: steps/train_mono.sh data/train.1k data/lang exp/mono

【注：本任务中使用的 train_mono_no_delta.sh 与 train_mono.sh 十分类似，不同之处在于下面这条语句：

train_mono.sh: `feats="ark,s,cs:apply-cmvn $cmvn_opts --utt2spk=ark:$sdata/JOB/utt2spk scp:$sdata/JOB/cmvn.scp scp:$sdata/JOB/feats.scp ark:- | add-deltas ark:- ark:- |"`

~_no_delta.sh： `feats="copy-feats scp:$sdata/JOB/feats.scp ark:- |"`

_no_delta 脚本中没有将特征维度进行拓展（`add-deltas ark:- ark:-`将维度拓展成原来的三倍），而且没有对 feats.scp 做倒谱方差均值归一化（`apply-cmvn`）。】

1、初始化单音素模型。调用 gmm-init-mono，生成 0.mdl、tree。

2、编译训练时的图。调用 compile-train-graph 生成 text 中每句抄本对应的 fst，存放在 fsts.JOB.gz 中。

3、第一次对齐数据。调用 align-equal-stats-ali 生成对齐状态序列，通过管道传递给 gmm-acc-stats-ali，得到更新参数时用到的统计量。

4、第一次更新模型参数。调用 gmm-est 更新模型参数。

5、进入训练模型的主循环：在指定的对齐轮数，使用 gmm-align-compiled 对齐特征数据，得到新的对齐状态序列；每一轮都调用 gmm-acc-stats-ali 计算更新模型参数所用到的统计量，然后调用 gmm-est 更新模型参数，并且在每一轮中增加 GMM 的分量个数。

(二) 程序

gmm-init-mono

作用：初始化单音素 GMM。

1、Usage: gmm-init-mono <topology-in> <dim> <model-out> <tree-out>

2、e.g.: gmm-init-mono topo 39 mono.mdl mono.tree

计算所有特征数据每一维特征的全局均值、方差

读取 topo 文件，创建共享音素列表（根据$lang/phones/sets.int)，根据共享音素列表创建 ctx_dep（相当于 tree)

每一组共享音素的一个状态对应一个 pdf。对每一个状态，创建只有一个分量的 GMM，该 GMM 的均值初始化为全局均值、方差初始化为全局方差。(实际上，此时表示 GMM 的类是 DiagGmm，该对象根据多维高斯分布的公式和对角协方差矩阵的特殊性，为了方便计算，直接保存的参数并不是均值、方差，而是方差的逆（实际就是方差矩阵每个元素求倒数)、均值×方差的逆，还提前计算并保存了公式中的常数部分（.mdl 文件 GMM 部分的<GCONSTS>）

根据 ctx_dep 和 topo 创建转移模型。将转移模型、GMM 声学模型写到 0.mdl

将 ctx_dep 写到 tree.

compile-train-graphs

1、Usage: compile-train-graphs [options] <tree-in> <model-in> <lexicon-fst-in> <transcriptions-rspecifier> <graphs-wspecifier>

2、e.g.: compile-train-graphs tree 1.mdl lex.fst ark:train.tra ark:graphs.fsts

该程序的输出是 ark 格式的 graphs.fsts(存为 exp/mono/fst.JOB.gz)，包含 train.tra 中的每个 utt-id 的 FST，FST 由无转移概率的 HCLG 组成。生成与音频特征对齐的 HMM 状态序列时要用到每句话的 FST。

align-equal-compiled

1、Usage: align-equal-compiled <graphs-rspecifier> <features-rspecifier> <alignments-wspecifier>

2、e.g.: align-equal-compiled 1.fsts scp:train.scp ark:equal.ali

对每一句话，根据这句话的特征和这句话的 fst，生成对应的对齐状态序列。

gmm-acc-stats-ali

作用：Accumulate stats for GMM training.

1、Usage: gmm-acc-stats-ali [options] <model-in> <feature-rspecifier> <alignments-rspecifier> <stats-out>

2、e.g.: gmm-acc-stats-ali 1.mdl scp:train.scp ark:1.ali 1.acc;

对于每一帧的特征和其对齐（transition-id)：

对于转移模型（TM），累积 tid 出现的次数；

对于 AM，由 tid 得到 pdf-id，也就是找到对应该 pdf-id 的 DiagGmm 对象，更新与该 DiagGmm 对象相关的 AccumDiagGmm 的参数，也就是计算得到三个 GMM 参数更新公式的分子部分（包括每一混合分量的后验（occupancy_中保存∑nj=1γ^jk∑j=1nγ^jk)、每一分量的后验乘以当前帧的特征(mean_accumulator_中保存∑nj=1γ^jkyj∑j=1nγ^jkyj，MxD 维)、每一分量的后验乘以 当前帧的特征每一维的平方(variance_accumulator_中保存∑nj=1γ^jky2j∑j=1nγ^jkyj2，MxD 维)）

处理完所有数据后，将 TM 和 AM 的累积量写到一个文件中：x.JOB.acc 中

gmm-sum-accs

gmm-acc-stats-ali 生成的累计量分散在 JOB 个文件中，该程序将分散的对应同一 trans-id、pdf-id 的累计量合并在一起。

gmm-est

作用：Do Maximum Likelihood re-estimation of GMM-based acoustic model.

1、Usage: gmm-est [options] <model-in> <stats-in> <model-out>

2、e.g.: gmm-est 1.mdl 1.acc 2.mdl

主要分两部分，一部分更新 TransitionModel，一部分更新 GMM。

更新转移模型：根据 gmm-acc-stats-ali 统计的 tid 出现的次数，做一个除法就可以更新转移概率矩阵 A。

更新 GMM：gmm-acc-stats-ali 已经得到了三个 GMM 参数更新公式的分子部分，方差累积量只需要减去更新后的均值的平方即可得到正确的方差更新公式。分母部分也已几乎得到，做一个简单的除法就可以更新 GMM 的分量概率、均值、方差。调用 MleAmDiagGmmUpdate()更新 GMM 的参数，然后在该函数里调用 MleDiaGmmUpdate()更新每一个 GMM 的参数，然后在后一个函数里更新每一个分量的参数。