Project 2：LVCSR系统搭建

xxxx 杨炫锟

1. 系统搭建和模型训练

1.1. 数据处理及特征提取

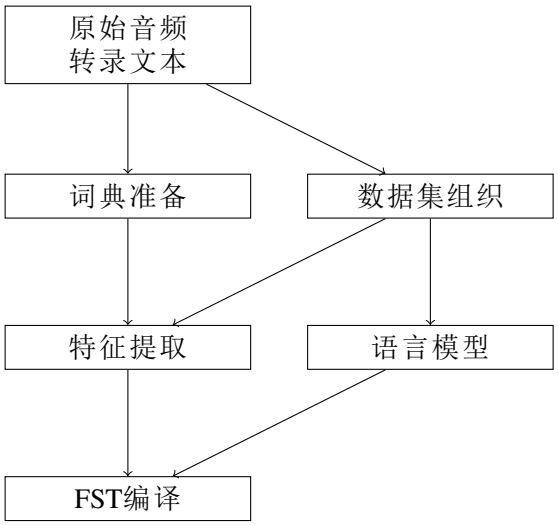

图 1: 数据处理与特征提取流程

该recipe的数据处理与特征提取流程遵循Kaldi 工具包的标准实践，针对中文普通话语音识别任务进行了优化，整体流程如图1所示。

1.1.1. 数据准备流程

该recipe在这一部分主要进行以下三个步骤：

1. 数据下载与解压：从`www.openslr.org/resources/33`下载并解压音频、转录文本（`data/data_aishell`）和资源（如词典，`data/resource_aishell`）
2. 词典准备：`aishell_prepare_dict.sh`用于生成发音词典，存储在`data/local/dict`
3. 数据整理：`aishell_data_prep.sh`将音频和转录文本整理为 Kaldi 格式，生成`data/train`、`data/dev`、`data/test`

助教给出的环境中已完成以上三个步骤，在使用时我直接将打包好的环境复制过来。

1.1.2. 语言与词典处理

这一部分为语音识别系统准备语言模型和词典相关资源，生成语言目录和解码所需的语言模型图，为后续声学模型训练和解码奠定基础。以下是我对具体步骤的解读：

1. 音素集准备：

   基于预准备的发音词典，生成 Kaldi 格式的语言目录（`data/lang`），定义音素集和语言模型的拓扑结构，提供音素和词汇的标准化表示，连接声学模型（音素建模）和语言模型（词序列建模），为后续解码图构建提供基础。

   主要由以下脚本完成：

   ```
   utils/prepare_lang.sh --position-dependent-
       phones false data/local/dict "<
       SPOKEN_NOISE>" data/local/lang data/lang
   ```

   `--position-dependent-phones false`指定了使用非上下文相关音素（即音素不依赖左右音素），简化建模，适合初始阶段。

2. 语言模型训练：

   基于 AISHELL 数据集的文本数据，训练一个未剪枝的 3-gram 语言模型，预测词序列的概率，用于解码时约束输出。

   主要由以下脚本完成：

   ```
   local/aishell_train_lms.sh
   ```

   未剪枝模型保留更多信息，适合实验性研究，但可能占用更多内存。

3. 图编译：

将语言模型与词典整合，生成测试语言目录（data/lang_test），并检查语言模型图（LG）的组合性，确保词典中的词与语言模型的词汇一致，为解码图准备。

主要由以下脚本完成：

```
utils/format_lm.sh data/lang data/local/lm/3
    gram-mincount/lm_unpruned.gz data/local/
    dict/lexicon.txt data/lang_test
```

1.1.3. 声学特征提取

这一部分从音频数据中提取声学特征（梅尔频率倒谱系数和音高），并进行归一化和数据校验，为声学模型训练提供输入，是连接原始音频与模型训练的桥梁。以下是我对具体步骤的解读：

1. MFCC + pitch特征提取：

   为有效表征语音信号，该recipe选用了梅尔频率倒谱系数（MFCC）和音高（pitch）这两个重要特征。这一部分从 train、dev、test 数据集的音频中提取MFCC和pitch特征，捕捉音频的频谱和声调信息，生成 Kaldi 格式的特征文件，提供声学特征，作为声学模型的输入。

   主要由以下脚本完成：

   ```
   for x in train dev test; do
       steps/make_mfcc_pitch.sh --cmd "
           $train_cmd" --nj 10 --write-utt2num-
           frames true data/$x exp/make_mfcc/$x
            $mfccdir
   done
   ```

2. CMVN 统计：

   计算倒谱均值和方差归一化（CMVN）统计量，对特征进行归一化，减小说话者或环境差异，提高模型训练的稳定性及模型的泛化能力。计算方式如下：

   $$\mathbf{X}'_{t,d} = \frac{\mathbf{X}_{t,d} - \mu_d}{\sigma_d} \tag{1}$$

   其中，矩阵$\mathbf{X}$提表示取出的一个语音序列的 MFCC 特征，维度为 $T \times D$，T为帧数，D为每帧的 MFCC 维度，μ_d和σ_d分别对应第d维的均值和标准差。

   主要由以下脚本完成：

   ```
   for x in train dev test; do
       steps/compute_cmvn_stats.sh data/$x exp/
           make_mfcc/$x $mfccdir
   done
   ```

3. 数据校验：

   检查和修复数据目录，移除不匹配的发音段，确保特征、转录文本和其他元数据的格式和一致性，为后续声学模型训练提供可靠的输入。

   主要由以下脚本完成：

   ```
   for x in train dev test; do
       utils/fix_data_dir.sh data/$x
   done
   ```

语言与词典处理和声学特征提取这两部分分别准备了语音识别的“语言知识”（词典和语言模型）和“声学特征”（MFCC + pitch），为从音频到文本的转换提供了核心组件。

1.1.4. 遇到的问题

下面将介绍我在这一部分遇到的问题以及解决方案：

1. - 问题：

     我在进行特征提取时，遇到了这样的报错：

     ```
     cat: exp/make_mfcc/train/utt2num_frames
         .1: No such file or directory
     ```

     表明 steps/make_mfcc_pitch.sh 在处理 data/train 数据时，尝试读取文件 exp/make_mfcc/train/utt2num_frames.1，但该文件不存在。

   - 解决方案：

     通过观察发现utt2num_frames 文件由 steps/make_mfcc_pitch.sh 生成，记录每个话语的帧数，用于后

续特征处理。如果缺失，可能是脚本执行过程中中断或配置问题。这涉及到steps/make_mfcc_pitch.sh中的write-utt2num-frames参数，虽然这一参数在原始steps/make_mfcc_pitch.sh中已设置为true，我尝试在run.sh中重新传入--write-utt2num-frames true，解决了这一问题。

1.2. 模型训练

1.2.1. 用 Kaldi 训练 GMM-HMM 模型

本系统采用渐进式训练策略构建声学模型，从基础的单音素模型开始，逐步提升到高级的说话人自适应模型。整个训练流程如图 2 所示，包含 5 个主要阶段。

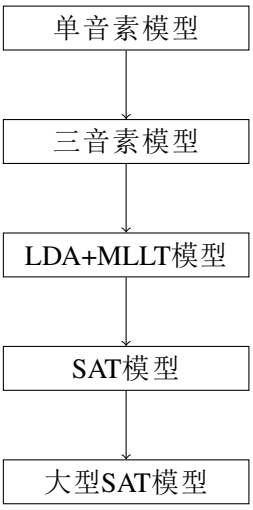

图 2: 渐进式 *GMM-HMM* 训练流程

1. 阶段 1：单音素模型训练

   在这一阶段，该recipe使用之前提取出的特征从零训练得到了一个单音素模型**mono**。

   在train_mono.sh中，在模型训练之前首先添加了一阶动态特征，在一定程度上提高模型对时间序列变化的捕捉能力。模型初始化阶段，脚本通过 gmm-init-mono 函数生成单音素的高斯混合模型。初始化时使用简单的等距对齐假设和上下文无关的拓扑结构，从而为后续训练提供基础模型。之后，将训练数据中的文本信息与语言模型结合，生成用于模型训练的有限状态转换图（FST），保证了训练数据的标注能够与声学模型输入保持一致。此外，训练过程分为多轮迭代（默认40），包括指定迭代步数进行对齐更新、统计参数累积和模型参数更新等。随着训练的深入，动态增加模型的高斯分量数量（totgauss），从而提升模型的表达能力。同时，通过不断调整对齐方式，模型能够更准确地对语音数据进行分割和建模。

2. 阶段 2：三音素模型训练

   在这一阶段，该 recipe 通过引入三音素特性提升声学模型的表达能力，训练了两个基础的三音素模型，并且在两个三音素模型中都使用了一阶与二阶动态特征，模型对时间序列建模能力更强。

   第一个三音素模型的初始对齐信息由单音素模型**mono**得到，训练得到三音素模型**tri1**，接着由**tri1**生成新的对齐信息，再由此训练第二个三音素模型**tri2**。上述对齐信息均是使用脚本steps/align_si.sh获得的，这是一种说话人无关的对齐方法。这两个三音素模型都是使用train_deltas.sh训练的，训练过程主要包括以下几个阶段：

   首先，脚本使用 acc-tree-stats 累积上下文相关的统计量，为决策树构建提供支持。随后，通过 cluster-phones 和 build-tree 对三音素的 HMM 状态进行聚类，生成决策树以优化状态建模，并利用 gmm-init-model 初始化三音素模型。接着，将单音素模型的对齐结果转换为适用于三音素模型的格式，确保训练数据标注与新模型匹配。

   同样的，在训练阶段，进行多轮迭代，包括指定迭代步数进行对齐更新、参数优化和高斯分量调整等。通过逐步增加高斯分量数量（totgauss），提升模型对语音特征的建模

能力，使模型具备上下文感知的能力。此外，该 recipe 通过决策树对状态进行聚类，有效解决数据稀疏问题。

3. 阶段 3：LDA+MLLT 变换

在这一阶段，该 recipe 训练了一个基于 LDA+MLLT （线性判别分析和最大似然线性变换）的三音素声学模型**tri3a**。下面是我对 LDA+MLLT 方法的一些理解：

- LDA是一种降维方法，旨在通过投影变换最大化类别间散度并最小化类别内散度，从而提高不同类别的判别性。其基本步骤如下：
  - 计算类内散度矩阵：

$$S_W = \sum_{i=1}^{C} \sum_{x \in X_i} (x - \mu_i)(x - \mu_i)^T \quad (2)$$

  - 计算类间散度矩阵：

$$S_B = \sum_{i=1}^{C} N_i (\mu_i - \mu)(\mu_i - \mu)^T \quad (3)$$

  - 优化目标：最大化广义瑞利商：

$$J(W) = \frac{\det(S_B)}{\det(S_W)} \quad (4)$$

  - 特征值分解：通过求解特征值问题得到投影矩阵，投影到低维空间。
- MLLT通过最大似然估计（MLE）优化高斯混合模型（GMM）的协方差结构。其主要步骤如下：
  - 协方差变换：

$$\Sigma' = W \Sigma W^T \quad (5)$$

  - 最大似然估计：优化变换矩阵 W。
  - 特征变换：

$$X' = XW \quad (6)$$

- LDA + MLLT：二者的组合使得特征变换在降维、分布优化和模型适配之间达到了良好的平衡。

该三音素模型的初始对齐信息由**tri2**得到，使用的脚本同样是`steps/align_si.sh`。该 recipe 使用`train_lda_mllt.sh`训练**tri3a**，训练过程包括以下几个主要阶段：

首先，利用对齐信息和 `acc-lda` 工具累积 LDA 统计量，并通过 `est-lda` 生成变换矩阵，实现特征的降维和分布优化。在初步训练模型的基础上，使用 MLLT 对特征进一步进行对角化变换以提高模型的鲁棒性。

在迭代过程中，每经历 10 次迭代，脚本执行对齐更新，并在指定的 MLLT 迭代中，累积和更新特征变换矩阵W。此外，脚本同样动态调整高斯分量数（`totgauss`），以逐步增加模型的复杂性和泛化能力。训练完成后，最终的模型和变换矩阵都会被保存。

该 recipe 的设计通过 LDA+MLLT 方法显著增强了特征的区分性，同时有效利用上下文信息，使得模型对语音信号中的细节建模更加精确。这一训练过程为进一步提升语音识别性能奠定了基础。

4. 阶段 4：说话人自适应训练（SAT）：

在这一阶段，该 recipe 基于**tri3a**得到对齐信息，然后使用说话人自适应训练的方法训练得到一个更强大的三音素模型**tri4a**。下面是具体的对齐信息获取与训练的过程：

- 基于 fMLLR 的对齐：
  训练一个说话人自适应的模型，在获取对齐信息的时候需要降低说话人之间的特征差异，使每个说话人的特征更加集中于语音类别相关的信息，这就引入了新的对齐方法 fMLLR，对应的脚本为`steps/align_fmllr.sh`。该方法假设说话人自适应码本（SA码本）可以通过说话人无关码本（SI）的线性变化得到：

$$SA = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \quad (7)$$

其中，$\mathbf{x}$为SI码本的特征向量，$\mathbf{A}$为$N \times N$的线性变化矩阵，N为特征维度，$\mathbf{b}$是偏移量。

不同说话人或句子的$\mathbf{A}$矩阵不同，以适应其个体差异，通过最大似然的方式估计$\mathbf{A}$矩阵。

该recipe利用`steps/align_fmllr.sh`对**tri3a**生成的对齐结果进行 fMLLR 特征变换, 根据说话人估计特征变换矩阵，并将特征对齐到新的语音空间。

- 训练 SAT 模型：

  SAT（Speaker Adaptive Training）模型训练使用的是`steps/train_sat.sh`，通过结合 fMLLR 特征变换和迭代优化，使得语音识别系统能够适应不同说话人的特征分布。以下是其主要过程：

  首先，在训练开始时，系统通过 CMVN、特征拼接和 fMLLR 转换，生成高质量的初始特征。接下来，系统通过决策树统计（`acc-tree-stats`）计算语音类别的相关参数，并使用聚类方法（`cluster-phones`）对音素进行分类，生成问题集和树文件。在此基础上，利用 GMM 模型初始化步骤，建立状态到特征的映射。

  在迭代训练阶段，系统根据训练数据不断优化 GMM 参数。在特定迭代步，重新对齐特征以更新 fMLLR 矩阵 $\mathbf{A}$，确保模型能够适应新的特征分布。每次迭代中，系统通过统计量累积和参数更新，不断提高模型对训练数据的拟合度。同时，在训练前若干轮次中，逐步增加高斯分量，以扩展模型的表达能力。

  最后，系统利用对齐模型（alignment model）对最终的 fMLLR 特征进行优化，确保生成的模型具备高质量的对齐能力和泛化性。同时，最终的线性变换矩阵$\mathbf{A}$也会被保存下来。整个训练流程通过 fMLLR 和 SAT 方法的结合，使得模型能够更有效地应对说话人差异，从而提升语音识别性能。

5. 阶段 5：大型 SAT 模型

   阶段2 ～ 4 训练得到的三音素模型设置的决策树叶子节点数量（`num-leaves`）均为 2500，高斯分量总数量（`totgauss`）均为 20000。在这一阶段，该 recipe 将决策树叶子节点数量增多至 3500，高斯分量总数量增加至 100000，并同样使用 fMLLR 的方法从**tri4a**获取对齐信息，使用`steps/train_sat.sh`进行训练，得到一个体量更大，划分更加细致的三音素模型**tri5a**。

1.2.2. 【可选*A*】基于深度神经网络的模型

1.2.3. 遇到的问题

我在训练 GMM-HMM 模型的时候遇到了下面这几个问题：

1. - 问题：

     脚本在执行`compile-train-graphs`时，多个并行任务（JOB=1, 2, 4, 6, 8）被系统杀掉（Killed）。

     错误信息：

     ```
     run.pl: 5 / 10 failed, log is in exp/mono/log/compile_graphs.*.log
     ```

   - 解决方案：

     检查log文件，发现存在退出码 137。退出码 137 通常表示进程被系统杀掉（SIGKILL，信号 9），最常见原因是内存不足（Out of Memory，OOM Killer）或资源限制。每个任务运行时间短（11-21 秒），未完成`compile-train-graphs`的主要工作，表明可能在初始化或加载数据时被中断。

     原本我申请的资源为 10 核和 32G 内存，可能存在内存不足的问题，于是我尝试

申请 10 核和 64G 内存，解决了这一问题。

2. • 问题：

   我在使用**mono**进行 decode 的时候出现了这样的报错：

   ```
   steps/decode.sh: Not scoring because
       local/score.sh does not exist or
       not executable.
   ```

   检查./exp/mono/目录下也没有score文件。

   • 解决方案：

   检查local/score.sh，发现该文件存在但没有执行权限，于是我为其添加了执行权限，解决了这一问题。

1.3. 实验结果

1.3.1. GMM-HMM 模型结果

完成所有训练、解码以及评分，我从/aishell/s5/exp/*/decode_test/best_wer和/aishell/s5/exp/*/decode_test/best_cer中读取出上述 6 个 GMM-HMM 模型在测试集上的性能表现，如表 1 所示。

Model	**mono**	**tri1**	**tri2**	**tri3a**	**tri4a**	**tri5a**
WER	45.85	28.64	28.50	26.82	23.56	**21.69**
CER	36.65	18.79	18.66	16.87	13.70	**12.13**

表 1: *GMM-HMM* 模型结果

从表 1 中可以看出，随着模型的递进，不论是 WER 还是 CER 均在下降，表明随着模型结构以及训练方法的改进，该语音识别系统也在逐步趋向稳健。

同时，不难发现，以上五个阶段之间的性能提升是比较明显的，尤其是单音素模型到三音素模型，**tri1**的 CER 接近**mono**的一半，表明通过上下文建模，三音素模型有效改善了单音素模型无法处理的协同发音问题。但是，同属于阶段 2 的模型**tri1**和**tri2**表现是非常接近的，这表明**tri2**的初始对齐信息是基于**tri1**获得的，但是两个模型在架构和参数层面都基本保持一致，单纯叠加模型获得的性能收益是很有限的。

此外，由**tri2**到**tri3a**存在性能提升，表明特征空间变换有效提升了模型鲁棒性，而相对温和的提升也表明特征优化存在收益递减点。由**tri3a**到**tri4a**是相对较大的，表明了 fMLLR 在减少说话人差异方面的高效性，也从侧面印证了说话人自适应码本（SA码本）可以通过说话人无关码本（SI）的线性变化得到这一假设的可行性。通过**tri4a**到**tri5a**，可以发现增加高斯分量带来的提升也是有限的，表明在 GMM-HMM 框架下模型容量有可能已接近饱和，这也启示我去不断探索后面的更先进的模型。

与此同时，我发现在 SAT 模型中，模型评分时会有不同的解码方式，一种是比较朴素的说话人无关解码（Speaker-Independent Decoding），即不使用任何说话人自适应技术，直接对输入特征进行解码，另一种结合了 fMLLR 及其他自适应技术，对特征进行了优化，使其更适合当前说话人。两种解码方式得到的评分也存在差异，具体如表 2 所示。

Model	**tri4a**		**tri5a**	
	SI-decoding	**SA-decoding**	**SI-decoding**	**SA-decoding**
WER	29.26	23.56	26.62	**21.69**
CER	19.29	13.70	16.74	**12.13**

表 2: *SAT* 模型不同解码方式对应性能

不难看出，结合自适应方法的解码方式（SA-decoding）较说话人无关的解码方式（SI-decoding）的性能表现是更优的，而 SI-decoding 方式得到的评分有可能较其余三音素模型还差。我认为这可能与 SAT 模型有关，SAT 模型使用的训练特征经过 fMLLR 等自适应方法处理，在解码的时候也需要与自适应方法相结合，并经过相应的特征变换处理，直接解码并评分自然会导致性能较差。

1.3.2. 【可选*A*】基于深度神经网络模型的结果

2. 【可选B】语料数量对模型性能的影响

3. 【可选C】优化系统

4. 最佳性能

CER=12.13%，WER=21.69%