Project 2：LVCSR 系统搭建

522030910109 刘翰文

1. 系统搭建和模型训练

1.1. 数据处理及特征提取

1.1.1. 数据集下载

首先，由于在 run.sh 中已经提供了语音识别训练模型所需的训练、测试和开法集的数据集路径。

```
data=/lustre/home/acct-stu/stu1718/aishell_data
. ./cmd.sh
```

所以可以按照 run.sh 中的指示，跳过注释中下载原 AIShell-1 训练集文本预料 data_aishell(训练集、测试集和开发集) 与 resource_aishell(发音词典、音素集等) 两个数据集，直接将所有需要的数据同数据集一起下载至 aishell 目录下。

```
# local/download_and_untar.sh $data $data_url
    data_aishell || exit 1;
# local/download_and_untar.sh $data $data_url
    resource_aishell || exit 1;
```

1.1.2. 准备词典和音素

在数据集下载完成后，按照 run.sh 进行下一步——准备模型训练过程中所需的发音词典 lexicon 和音素集 phoneme，并将数据存入 aishell/data/local/dict 目录下。

```
# Lexicon Preparation,
local/aishell_prepare_dict.sh $data/
    resource_aishell || exit 1;
```

进入 dict 目录，得到的词典是一个将字或词映射到其发音序列的文件，例如:

```
啊 aa a1
啊 aa a2
啊 aa a4
```

而音素集则包括了所有发音或不发音的语音识别系统中的基本单位，音素:

```
a1 a2 a3 a4 a5
aa
ai1 ai2 ai3 ai4 ai5
an1 an2 an3 an4 an5
```

这些词典在模型训练过程中为训练提供了基本的发音信息，同时不同的音素和词典也有助于模型针对不同的语音信号进行针对性的训练。

1.1.3. 数据集准备

接下来继续按照脚本命令执行，执行命令:

```
# Data Preparation,
local/aishell_data_prep.sh $data/data_aishell/wav
    $data/data_aishell/transcript || exit 1;
```

该命令在 aishell/data/local 目录下存入划分后的数据集，包括训练集、测试集和开发集，每个数据集下包括了对应数据集的音频序列、路径、标注和音频与说话人之间的关系文件，以便进行后续的训练、测试等。

1.1.4. 构建音素词典

继续运行 run.sh:

```
# Phone Sets, questions, L compilation
utils/prepare_lang.sh --position-dependent-phones
    false data/local/dict \
    "<SPOKEN_NOISE>" data/local/lang data/lang ||
    exit 1;
```

该命令将根据准备词典和音素步骤中在 aishell/data/local/dict 目录下得到的词典和音素，从词典中提取音素信息，在 aishell/data/lang 目录下生成 Kaldi 所需的 phones.txt 文件和其他音素相关的文件，创建词汇表 word.txt，创建有限状态转录机 FST 并消除在部分音素串之中的歧义，并将最后的 FST 存入 L_disambig.fst 文件中。

1.1.5. 创建语言模型

在完成了上述数据层面的准备后开始进行语言模型 lm 的创建，运行 run.sh 中的命令构建语言

模型，并将构建的模型存入 aishell/data/local/lm 目录下：

```
# LM training
local/aishell_train_lms.sh || exit 1;
```

该模型通过收集数据集中的音频标注，统计词频来捕捉语言的统计特性。

1.1.6. 转换语言模型

在创建得到了语言模型之后，脚本将语言模型转换为 FST 格式，生成 G.fst 文件。接着将语言模型和词典结合起来，生成用于语音识别解码的 LG.fst 文件并将其保存在 aishell/data/lang_test/tmp 目录下。

```
# G compilation, check LG composition
utils/format_lm.sh data/lang data/local/lm/3gram-mincount/lm_unpruned.gz \
    data/local/dict/lexicon.txt data/lang_test || exit 1;
```

1.1.7. MFCC 特征提取

最后，提取数据集中语音信号的梅尔倒谱系数 (MFCC) 特征。MFCC 特征反映了人耳对不同频率的声波有不同的听觉敏感度，比基于声道模型的 LPCC 相比具有更好的鲁邦性，更符合人耳的听觉特性，而且当信噪比降低时仍然具有较好的识别性能。其步骤如图 1 所示。按照 run.sh 中的脚本提取所有数据集的 MFCC 特征，将特征存入 aishell/mfcc 目录下，并将 mfcc 特征作为之后模型训练的主要输入。

```
# Now make MFCC plus pitch features.
# mfccdir should be some place with a largish disk where you
# want to store MFCC features.
mfccdir=mfcc
for x in train dev test; do
  steps/make_mfcc_pitch.sh --cmd "$train_cmd" --nj 10 data/$x exp/make_mfcc/$x $mfccdir || exit 1;
  steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x $mfccdir || exit 1;
  utils/fix_data_dir.sh data/$x || exit 1;
done
```

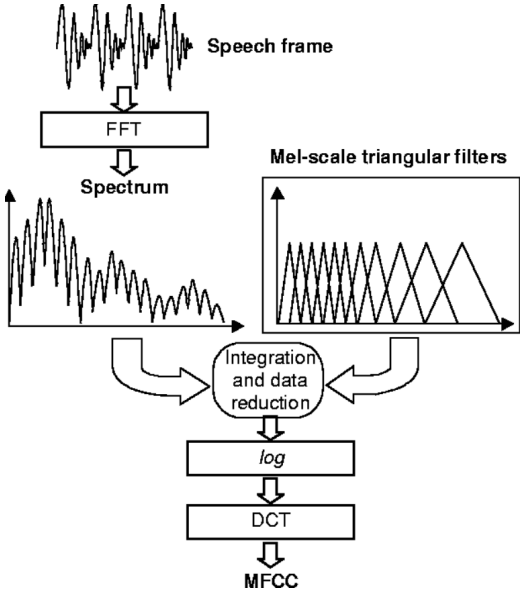

图 1: *MFCC* 特征提取步骤

1.2. 模型训练

1.2.1. 用 Kaldi 训练 GMM-HMM 模型

在官方的 recipe 中，脚本首先选择在语音数据上训练一个基于单音素 (monophone) 的声学模型。单音素模型是语音识别系统中最简单的模型类型之一，它将每个音素 (phoneme) 作为一个独立的模型进行训练，而不考虑音素在词中的具体位置或上下文带来的协同发音等影响。

```
# Train a monophone model on delta features.
steps/train_mono.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/mono || exit 1;
```

在训练过程中，首先初始化高斯混合模型，并将一系列的 MFCC 特征序列进行对齐，然后求出 HMM 的参数-转移概率。在单音素 GMM 训练中，每一个 HMM 状态有一个对应的 GMM 概率密度函数，所以有多少个 HMM 状态，就有多少个 GMM，也就有多少组 GMM 参数。之后找出一个 HMM 状态对应的所有观测，使用 EM 算法进行迭代优化 GMM 模型参数并同时不断进行对齐操作至收敛。最后，将得到的模型存入 aishell/exp/mono 目录下。

在得到训练完的模型之后，开始进行对模型的解码。

```
# Decode with the monophone model.
```

```
utils/mkgraph.sh data/lang_test exp/mono exp/mono/graph || exit 1;
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/mono/graph data/dev exp/mono/decode_dev
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/mono/graph data/test exp/mono/decode_test
```

该步骤首先运行 aishell/uils 目录下的 mkgraph.sh 脚本将声学模型 (H)、上下文决策树 (C)、词典 (L) 和语言模型 (G) 结合，生成一个包含了声学模型、词典和语言模型的信息，用于解码时搜索的解码图 HCLG.fst，并将其保存在 aishell/exp/mono/graph 目录下。HCLG.fst 是一个有限状态机，包含了声学模型、词典和语言模型的信息，用于解码时的搜索。

随后运行 aishell/steps 目录下的 decode.sh 脚本使用生成的解码图对开发集和测试集进行解码。解码过程中，模型根据输入音频的 MFCC 特征，通过搜索解码图 HCLG.fst 进行解码，产生词图，得到最终的字错误率 (CER) 和词错误率 (WER)，并将结果存入 aishell/exp/mono/decode_dev 和 decode_test 目录下。

最后，在单音素模型训练完成并经过解码后，使用 Viterbi 动态规划算法对训练数据进行对齐，将音频数据中的每一帧与模型中的音素状态对齐，以便后续的模型训练。

```
# Get alignments from monophone system.
steps/align_si.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/mono exp/mono_ali || exit 1;
```

在单音素模型之后，run.sh 又构建了一个三音素模型，相比于先前的单音素模型，三音素模型将上下文的相关性引入，很大程度上怎加了语音识别的准确率和鲁棒性。

在三音素模型的构建中，首先完成了模型和参数初始化，并提取音频的 MFCC 特征以及其一二阶差分和对齐后的结果，然后进行模型的训练。训练过程中，首先对每个音素的每个状态都建立一个决策树，使用决策树对三音素组合进行聚类，然后于单音素的训练流程类似，在初始化之后使用 EM 算法对 GMM 模型参数不断优化至收敛。并在训练结束后进行解码和对齐。

```
# Train the first triphone pass model tri1 on delta + delta-delta features.
steps/train_deltas.sh --cmd "$train_cmd" \
 2500 20000 data/train data/lang exp/mono_ali exp/tri1 || exit 1;
# decode tri1
utils/mkgraph.sh data/lang_test exp/tri1 exp/tri1/graph || exit 1;
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/tri1/graph data/dev exp/tri1/decode_dev
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/tri1/graph data/test exp/tri1/decode_test
# align tri1
steps/align_si.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/tri1 exp/tri1_ali || exit 1;
```

随后，脚本又对训练一次后的脚本进行第二次训练，并同时进行解码、对齐和保存测试结果。其目的是通过多次训练提升模型的性能并提高模型的泛化能力。

在第三次训练中，脚本使用了 LDA+MLLT(线性判别分析 + 最大似然线性变换) 对 MFCC 特征进行降维和线性变换，用新的特征进行第三次训练、解码对齐和测试。

```
# Train the second triphone pass model tri3a on LDA+MLLT features.
steps/train_lda_mllt.sh --cmd "$train_cmd" \
 2500 20000 data/train data/lang exp/tri2_ali exp/tri3a || exit 1;
```

LDA 算法是一种降维算法，它基于降维之后类内距离最小化和类间距离最大化的准则，通过最大化目标函数 J:

$$
\begin{aligned}
J &= \frac{w^T S_b w}{w^T S_w w} \\
S_b &= \sum_{i=1}^{C} N_i (\mu_i - \mu)(\mu_i - \mu)^T \\
S_w &= \sum_{i=1}^{C} \sum_{x \in X_i} (x - \mu_i)(x - \mu_i)^T
\end{aligned}
\tag{1}
$$

将优化问题转化为特征值求解问题，其中 S_b, S_w 分别代表类间散度矩阵和类内散度矩阵。得到特征方程:

$$
S_w^{-1} S_b w = \lambda w \tag{2}
$$

求解方程，取出不同数目的特征向量组合实现对高维特征的降维。

由于对角化协方差矩阵会对似然度产生损失，所以使用了 MLLT 最大似然线性变换算法以获得更好的特征表示。通过引入 LDA 降维和 MLLT 最大似然线性变换，来获得更好的特征表示并从而提升模型的性能。

在第三次模型训练结束之后开始第四次模型训练，相较于前三次训练，第四次训练新引入了 feature-space MLLR(特征空间最大似然线性变换，fMLLR) 和说话人自适应训练 (SAT)。说话人自适应技术是利用特定说话人数据对说话人无关 (Speaker Independent,SI) 的码本进行改造，其目的是得到说话人自适应 (SPeaker Adapted, SA) 的码本来提升识别性能。也就是说，我们希望从不同口音、语速、语调等说话特征的人中将说话人和语义这两部分的信息从语音特征中分离，使模型专注于语义而忽略说话人不同对语音识别造成的干扰，从而提升模型的性能。为了实现 SAT，需要使用 fMLLR 算法，将不同说话人的特征映射到一个共享的特征空间中，而 fMLLR 的线性变化矩阵，可以通过对说话人的训练数据进行参数估计得到，其本质就是估计矩阵 A，使得 SI 码本可以用 SA 码本的线性变换表示：

$$x^* = Ax + b \tag{3}$$

假设有 K 个说话人的训练数据，每个说话人的 SA 码本均由 SI 码本线性变换得到，训练的目标是使得输出概率函数最大，即：

$$\begin{aligned}(A^*, \lambda^*) &= argmin f(x, \Sigma, A) \\ f(x, \Sigma, A) &= -\sum_{k=1}^{K} \sum_{t \in O(k)} log[P(o(t)|\lambda A_k]\end{aligned} \tag{4}$$

其中，λ 表示 SI 的参数，A_k 表示第 k 个人的变换矩阵，$A_k = [b_k, U_k]$，O_k 表示第 k 个人的训练数据集合，W^*, λ^* 表示参数的最佳估计。

```
# align tri3a with fMLLR
steps/align_fmllr.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/tri3a exp/tri3a_ali ||
    exit 1;
```

```
# Train the third triphone pass model tri4a on LDA
    +MLLT+SAT features.
# From now on, we start building a more serious
    system with Speaker
# Adaptive Training (SAT).
steps/train_sat.sh --cmd "$train_cmd" \
  2500 20000 data/train data/lang exp/tri3a_ali
    exp/tri4a || exit 1;
```

执行 run.sh 脚本，首先使用 fMLLR 进行特征变换和对齐，然后使用 LDA、MLLT 进行与第三次训练相似的 MFCC 特征变换，最后进行 SAT 训练三音素模型和解码测试模型。

最后，进行第五次训练：

```
# Train tri5a, which is LDA+MLLT+SAT
# Building a larger SAT system. You can see the
    num-leaves is 3500 and tot-gauss is 100000
steps/train_sat.sh --cmd "$train_cmd" \
  3500 100000 data/train data/lang exp/tri4a_ali
    exp/tri5a || exit 1;
```

相较于第四次训练，本次训练大幅度的增加了参数量，决策树的叶节点增加到了 3500，高斯数量增加到了 100000 个，通过参数量的增加提高模型的表现。

1.2.2. 【可选 *A*】基于深度神经网络的模型

使用基于 DNN 的 DNN-HMM 语音识别模型，按照 run.sh 的脚本继续执行。

```
# nnet3
local/nnet3/run_tdnn.sh
```

nnet3 的训练借助了先前操作中获得的各种模型，音素词典等，对音频进行对齐、提取 MFCC 特征、LDA 降维等操作。此外，还通过训练进行音频的 I-Vector 特征提取，并于 MFCC 特征一起进行拼接送入模型训练以提高模型的泛化和性能。然后，通过一系列参数初始化 DNN 网络然后进行训练。最后，与其他模型类似进行解码和测试。

1.3. 实验结果

1.3.1. GMM-HMM 模型结果

表 1 展示了不同系统对应的评分结果相对路径，表 2 展示了不同系统的性能。

系统	路径
mono	aishell/exp/mono/decode_test/cer_10_0.0 aishell/exp/mono/decode_test/wer_10_0.0
tri1	aishell/exp/tri1/decode_test/cer_14_0.5 aishell/exp/tri1/decode_test/wer_15_0.5
tri2	aishell/exp/tri2/decode_test/cer_14_0.5 aishell/exp/tri2/decode_test/wer_14_0.5
tri3a	aishell/exp/tri3a/decode_test/cer_14_0.5 aishell/exp/tri3a/decode_test/wer_15_0.5
tri4a	aishell/exp/tri4a/decode_test/cer_13_0.5 aishell/exp/tri4a/decode_test/wer_15_0.5
tri5a	aishell/exp/tri5a/decode_test/cer_17_1.0 aishell/exp/tri5a/decode_test/wer_17_0.5

表 1: *GMM-HMM* 模型结果路径

系统	WER	CER
mono	58.39%	45.75%
tri1	43.79%	28.62%
tri2	43.70%	28.49%
tri3a	41.22%	25.88%
tri4a	**36.43**%	**20.67**%
tri5a	37.82%	22.12%

表 2: *GMM-HMM* 模型性能

通过对不同系统的性能分析可得，前四次训练模型性能不断上升，在使用 SAT 和 fMLLR 的第四次训练的模型达到了不管是在 WER 还是 CER 上的最佳性能，而在提高参数量后的第五次训练得到的模型性能反而有所下降，可能是参数过多带来的过拟合的影响。

1.3.2. 【可选 *A*】基于深度神经网络模型的结果

通过前述脚本的执行可以在 aishell/exp/nnet3/tdnn_sp/decode_test 目录下获得一系列模型的测试结果，其中最佳性能以及其与 GMM-HMM 的结果列入表格进行对比，结果如表 3 所示。对比 nnet3 和先前训练的 6 个 GMM-HMM 模型，发现 DNN-HMM 模型的性能达到了不管是在 WER 还是 CER 上的最佳，可见其对语音识别的优越性能。

系统	WER	CER
mono	58.39%	45.75%
tri1	43.79%	28.62%
tri2	43.70%	28.49%
tri3a	41.22%	25.88%
tri4a	36.43%	20.67%
tri5a	37.82%	22.12%
nnet3	**32.88**%	**17.40**%

表 3: *DNN-HMM* 与 *GMM-HMM* 模型性能结果对比

2. 【可选 B】优化系统

2.1. 方法

2.2. 性能对比

3. 最佳性能

CER=17.40%，WER=32.88%