Project 1：语音端点检测

520030910155 邱一航

0. 说明

本文为 AI2651《智能语音识别》课程的第一次大作业报告。

1. 基于线性分类器和语音时域特征的简单语音端点检测算法

1.1. 数据预处理及特征提取

1.1.1. 音频预处理

对原始的输入音频，笔者尝试了如下的两种不同预处理方式。（记原始音频为 $x[n]$，预处理后的音频为 $\tilde{x}[n]$）

- **平滑**。将当前离散时间点附近的一段矩形窗内的所有信号取平均值作为当前时间点的信号。记矩形窗长度为奇数 $2L+1$，则该过程用如下公式表示：

$$\tilde{x}[n] = \frac{1}{2L+1} \sum_{k=n-L}^{n+L} x[k]$$

- **预加重**。查阅相关资料 [1] 后得知，该方法能够去除口唇辐射的影响，增强语音的高频部分及高频分辨率。预加重处理可用如下公式表示：（其中预加重系数 α 的一般取值范围是 $\alpha \in [0.9, 1]$）

$$\tilde{x}[n] = x[n] - \alpha x[n-1]$$

在附件中程序（`task1.py` 或 `task1.ipynb`）的 Hyperpatameter 定义部分，设置了两个变量决定是否使用上述两种预处理方式。

```
# For frame segmentation:
Smooth = False
#Pre-emphasis:
Emph = True
a = .9
```

其中，`Smooth` 和 `Emph` 为 `True` 时表示使用对应的方法进行预处理，a 是预加重系数。

1.1.2. 帧分割

笔者在程序中编写了 `Frame_Seg` 函数用于将原始音频切割为帧。

本次大作业中，笔者并未过多研究不同帧大小和帧偏移对语音端点检测效果的影响，仅做了几次实验对比检测效果，最终采用了效果较好的 0.032 秒和 0.008 秒的默认值。

1.1.3. 特征提取

对预处理后的数据，笔者进行了以下两种特征的提取。

- **短时能量**，即单个帧内所有采样点上信号的能量和。记 N 为帧的长度，用 s 表示当前帧，则短时能量可用如下公式表述：

$$E(s) = \sum_{k=0}^{N-1} (s[k])^2$$

- **过零率**，即每一帧中采样后信号跨越零点的次数。记 N 为帧的长度，用 s 表示当前帧，则过零率可用如下公式表示：

$$Z(s) = \frac{1}{2} \left[\sum_{k=1}^{N-1} \left| \mathtt{sgn}(s[k]) \right| - \left| \mathtt{sgn}(s[k-1]) \right| \right]$$

$$\text{其中 } \mathtt{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

1.2. 算法描述

下图（见下页中的图1）是一段音频短时能量和过零率的可视化。蓝色折线为短时能量，红色折线为过零率，黑色折线为语音标记，值为 1 表示是语音，为 0 表示不是语音。

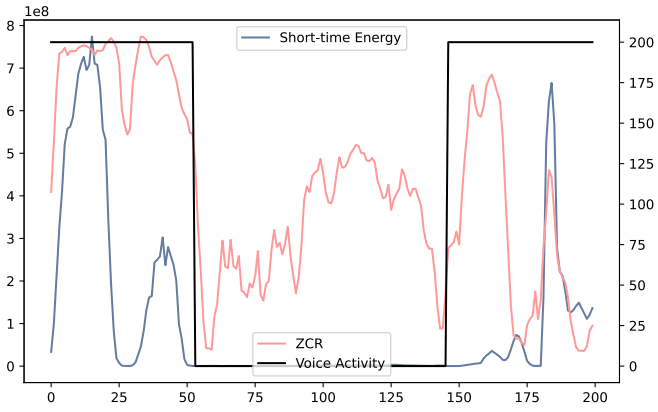

图 1: *Visualization of Short-time Energy and ZCR*

结合课程中已讲述的语音特征等知识，观察被标记为“语音”的音频段落与标记为“非语音”的段落的短时能量与过零率，可以发现：

- 语音部分可以近似看作“浊音”和“清音”的组合，且浊音与清音一般是互相紧邻出现的。
- 浊音部分短时能量较高，且浊音开始部分会迅速达到一个较高的峰值，随后逐渐下降。浊音的过零率不一定非常高，因此用短时能量判断更准确。
- 清音部分的短时能量较低，但过零率非常高。

因此，基于 **1.1** 中所提取的两种特征，笔者分别进行如下处理：

- 使用**短时能量**判断浊音。为了能取得更好的效果，笔者采用了**双门限法**判断，即当短时能量超过高阈值时，认为接下来的帧都属于浊音；当短时能量下降到低阈值时，认为浊音结束。这样处理可以更精确地判断浊音，同时又可以将浊音的尾部（短时能量相对较低的部分）正确地判断为浊音。
- 使用**过零率**判断清音。这里笔者尝试了两种不同方式：
  - 从所有已经获得的浊音两端出发，根据过零率判断当前帧是否为清音，认为过零率高于一定阈值即为清音。最终的所有语音帧即所有清音帧与浊音帧。
  - 对过零率也使用**双门限法**，单独查找清音。随后查找浊音与清音段落之间短时能量大于能量低阈值且过零率大于过零率低阈值的过渡段，标记为语音。

第一种方法（对短时能量使用双门限法判断浊音，再在浊音附近用过零率单门限判断清音）的伪代码如下。

Algorithm 1: 短时能量双门限、过零率单门限法

Function $VAD(S, \alpha, \beta, \gamma, \delta)$
  // S 是所有帧的集合。α 和 β 分别是短时能量的高、低阈值；γ 和 δ 分别是过零率的高、低阈值。
  $pred \leftarrow \{\text{非语音}\}_{s \in S}$;
  // $pred$ 标记每个帧是否为语音。
  while $s \in Ss$ **do**
    while $G(s) < \alpha$ **do**
      $s \leftarrow S$ 中的后一帧;
    end
    while $G(s) \geq \beta$ **do**
      $pred[s] \leftarrow$ 是语音;
      $s \leftarrow S$ 中的后一帧;
    end
  end
  while $s_0 \in S$ *s.t.* s_0 是语音 **do**
    $s \leftarrow s_0$;
    while s 的过零率 $\geq \gamma$ **do**
      $pred[s] \leftarrow$ 是语音;
      $s \leftarrow S$ 中的前一帧;
    end
    $s \leftarrow s_0$;
    while s 的过零率 $\geq \gamma$ **do**
      $pred[s] \leftarrow$ 是语音;
      $s \leftarrow S$ 中的后一帧;
    end
  end
  Return: $pred$
end

第二种方法（对短时能量和过零率均使用双门限法，并对过渡段特殊处理）的伪代码如下。

Algorithm 2: 能量双门限、过零率单门限法

Subroutine *2-Th*$(S, G, a, b, pred)$

```
// 双门限。
// S 是所有帧的集合，G 是短时能量或过零率的计算函数。a 是高阈值，b 是低阈值。pred 标记了每个帧是否为语音。
while s ∈ frames do
    while G(s) < a do
        s ← S 中的后一帧;
    end
    while G(s) ≥ b do
        pred[s] ← 是语音;
        s ← S 中的后一帧;
    end
end
```

end

Function *VAD*$(S, \alpha, \beta, \gamma, \delta)$

```
// S 是等待识别语音端点的音频经切割后得到的帧集合。α 和 β 分别是短时能量的高阈值和低阈值；γ 和 δ 分别是过零率的高阈值和低阈值。
pred ← {非语音}_{s∈S};
// pred 标记每个帧是否为语音。

2-Th(S, 短时能量, α, β, pred);
2-Th(S,ZCR, γ, δ, pred);
while s_0 ∈ S s.t. s_0 是语音 do
    s ← s_0;
    while s 的短时能量 ≥ β 且 s 的 ZCR≥ δ do
        pred[s] ← 是语音;
        s ← S 中的前一帧;
    end
    s ← s_0;
    while s 的短时能量 ≥ β 且 s 的 ZCR≥ δ do
        pred[s] ← 是语音;
        s ← S 中的后一帧;
    end
end
Return: pred
```

end

以上算法中的阈值均可视为“硬阈值”，即不会随音频文件变化而变化。然而，考虑到不同的音频文件中，短时能量的平均值未必一致，因此笔者还尝试将音频所有帧的短时能量**归一化处理**后，再进行上述操作。即对每个音频文件切割后的帧集合 S，做以下操作：

$$\mu(S) \triangleq \frac{1}{|S|}\sum_{s\in S} E(S)$$

$$\sigma(S) \triangleq \frac{1}{|S|}\sum_{s\in S} (E(S)-\mu)^2$$

$$\forall s \in S, \tilde{E}(s) = \frac{E(s)-\mu(S)}{\sigma(S)}$$

之后对 $\tilde{E}(s)$ 用上述两种算法处理出语音端点。此时设置的各种阈值可视为“可变阈值”，对归一化前的短时能量而言是一个与能量的均值、方差相关的变量。

因此实际上笔者共设计了四种不同的算法。下将每个音频的短时能量归一化后使用算法 1 的方法称为“算法 3”，短时能量归一化后使用算法 2 称为“算法 4”。这四种算法在不同的数据预处理下的效果请见 **1.3**。

1.3. 实验结果

在不同的数据预处理（是否平滑、是否预加重）下，**1.2** 中所提及的四种算法在开发集上的效果（AUC,ERR,ACC（准确率））如表1所示。

由于每种方法都有各自的参数需要调节，下表中的结果均为各自方法下的最好效果；各种情况下的最佳参数在表2中给出。

特别地，表中加粗的几种具有代表性的方法，其具体参数在附件中的程序文件中作为默认值存在。这几种代表性算法在测试集上的测试结果在 task1 文件夹中均有给出，其中 `test_label_task.txt` 保存的是预加重后使用算法 1 的结果。

（**注**：由于使用平滑后效果普遍**更差**（AUC 下降且 ERR 上升），因此并未对平滑且预加重原始数据的情况进行实验。）

方法	预处理	AUC	ERR	ACC
算法 1	**无**	**0.8906**	**0.1242**	**0.8690**
算法 2	无	0.8897	0.1297	0.8479
算法 3	**无**	**0.7525**	**0.3890**	**0.7283**
算法 4	无	0.7212	0.3803	0.7028
算法 1	平滑	0.8663	0.1858	0.8422
算法 2	平滑	0.8593	0.1901	0.8230
算法 3	平滑	0.7533	0.3906	0.7391
算法 4	平滑	0.7294	0.3952	0.7102
算法 1	**预加重**	**0.9012**	**0.1178**	**0.8782**
算法 2	预加重	0.8983	0.1216	0.8623
算法 3	预加重	0.7614	0.3778	0.7590
算法 4	预加重	0.7308	0.3695	0.7219

表 1: 基于语音时域特征的简单算法检测效果对比

注意表中加粗的仅为较有代表性且参数被保存的算法，并不一定是开发集上效果最佳的算法。

使用平滑预处理后，数据的过零率被改变了，因而不再能很好地通过过零率来判断清音的位置，导致几乎所有方法的 AUC 下降且 ERR 上升。

显然效果最佳的是使用预加重处理后的算法 1（对短时能量使用双门限判断浊音，再在浊音附近的语音片段上利用过零率查找清音语音片段），该方法也作为附件中程序中的默认方法。三种加粗的代表性算法对应的参数如下表。

方法	预处理	短时能量高阈值	短时能量低阈值	ZCR 阈值
算法 1	无	300000000	7500000	100
算法 3	无	1.0	0.8	0.9
算法 1	预加重	30000000	750000	1000

表 2: 三种代表性方法的最佳参数

2. 基于统计模型分类器和语音频域特征的语音端点检测算法

2.1. 数据预处理及特征提取

2.1.1. 音频预处理

由 **1.3** 实验可见，**预加重**这一预处理方式使所有方法的效果都得到了增强，因此这里我们依然采用预加重作为预处理方式，公式如下。笔者选取的预加重系数 $\alpha = 0.9$。

$$\tilde{x}[n] = x[n] - \alpha x[n-1]$$

2.1.2. 帧切割

由于后续将要使用频域特征，为了保证不引入频率噪声，我们在帧分割时将直接切割获得的帧乘上窗函数，使每一帧的边缘较为“柔和平滑”。

记原始音频为 $x[n]$，切割得到的某一帧为 $s[n]$，窗函数为 $w[n]$，帧长度为 L。则该过程即为

$$\tilde{s}[n] = s[n] \cdot w[n] \ (n = 0, 1, ..., L-1)$$

其中窗长（即 $w[n]$ 的长度）与帧长度 L 需要完全一致。

笔者在实验中选用了汉宁窗（Hanning Window），但在程序实现时也实现了使用其他窗进行边缘平滑的功能，详见附件中的程序 `task2.ipynb` 和 `task2.py`。

汉宁窗的时域和频域性质如图2所示。

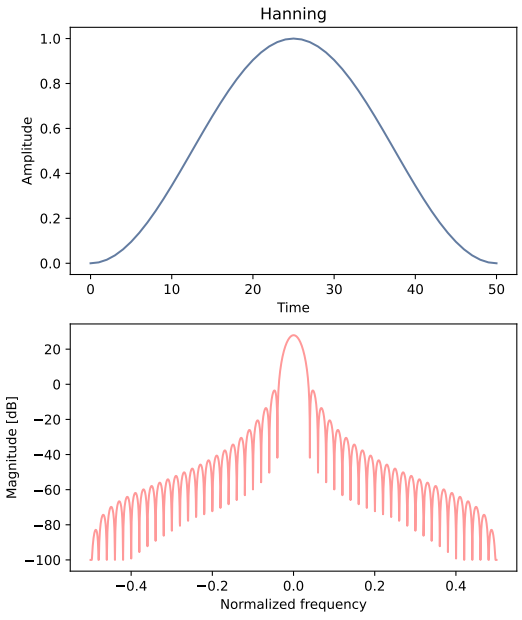

图 2: *The Visualization of Hanning Window on Time Domain and Frequency Domain*

2.1.3. 频域特征提取

笔者选用了 MFCC（Mel-Frequency Cepstral Coefficients，梅尔滤波器倒谱系数）和 FBank（Filter Bank）两种特征。实际上，前者是后者经过处理得到的，因此两者的本质实际上是一致的。具体的代码实现详见 `task2.ipynb` 或 `task2.py`。

Mel 滤波器是一系列（一般为 40 个）滤波器，滤波器通频上下限频率转化为梅尔后是等差数列。该滤波器组能很好地模拟人耳的听觉特性。

2.2. 算法描述

笔者基于 MFCC 和 FBank 两种频域特征，分别使用了神经网络 (DNN) 进行该部分实验。

笔者在 sklearn 和 pytorch 上分别尝试实现了神经网络，不同之处在于笔者直接调用了前者提供的 MLPClassifier，而在后者提供的平台上自己定义了神经网络及其训练。因此，实际上在该部分实验中，笔者一共实现了四个神经网络。它们的不同结构和训练方式如下。

2.2.1. 神经网络结构描述

四个网络均为 MLP，即多层感知器（最朴素的神经网络）。表格中给出了每一层的神经元个数。

（注："预设" 指使用了 sklearn 预设的 MLPClassifier；"自定义" 指基于 pytorch 提供的 nn.Module 模块进行自定义网络构建。）

基于特征	网络实现	结构
MFCC	预设	$12 \to 24 \to 12 \to 6 \to 4 \to 2$
FBank	预设	$40 \to 40 \to 20 \to 10 \to 4 \to 2$
MFCC	自定义	$12 \to 12 \to 4 \to 1$
FBank	自定义	$40 \to 10 \to 8 \to 4 \to 1$

表 3: 神经网络结构说明

2.2.2. 神经网络训练方式

对前两个网络，由于笔者使用了预设的 sklearn.neural_network.MLPClassifier 实现，训练时**只需要调用**.fit(features, labels) 函数即可，其中 features 是输入的特征（由整个训练集的特征组成），labels 是 features 对应的分类（此处即为 0/1，即语音或非语音）。

对后两个网络，笔者将网络最终的输出 pred（含义是语音的概率）与训练集上的标签 label 做二分类交叉熵 loss（BCE Loss），试图让两者尽可能接近（即非语音的输出接近 0，语音帧的输出接近 1）。但后续实验表明自定义网络在这种训练下效果不佳，具体分析详见 **2.3**。

值得一提的是，对 pytorch 上自定义的网络，如果 loss 函数仅考虑让网络的输出与训练集上的标签结果尽可能接近，很有可能会得到网络输出大部分为 1 或大部分为 0 的情况，因为该情况下 loss 函数也达到了一个极小值。

因此，笔者在后两个网络上也尝试了一些其他方式想要避免这种情况的发生，如引入**负采样**来让训练集中语音段落和非语音段落的出现概率更加接近（具体尝试详见 `task2_neg.ipynb`）。此外，笔者还试图在 loss 函数中加入如下的正则项来避免输出全为 0 或全为 1 的情况（该正则项在输出的平均值接近 0 或 1 时会很大，从而避免输出中 0 或 1 出现概率过高的情况）。

$$Reg(pred) = \texttt{tan}\left(\left(\texttt{mean}(pred) - \frac{1}{2}\right)\pi\right).$$

其中 $\texttt{mean}(\cdot)$ 为求平均值。则 loss 函数变为：

$$\begin{aligned} Loss(pred, label) = {} & BCELoss(pred, label) \\ & + \gamma Reg(pred) \end{aligned}$$

但后续实验证明这两种尝试的效果依然非常糟糕，远远不如 sklearn 预设的神经网络中的预设训练方式。因此，笔者最终只使用基于 sklearn 预设的两个神经网络在 test 集上生成语音端点识别结果。

2.2.3. 神经网络参数说明

神经网络训练采用梯度下降。对于基于 sklearn 预设实现的神经网络，需要设定最大训练轮数 max_iter。对于基于 pytorch 自定义建构

的神经网络，需要设置训练轮数。笔者在程序中的 Hyperparameter 定义部分设置了变量 Training_epoch 用以设置神经网络的训练轮数。

此外，训练时需要设置学习率。笔者在程序中的 Hyperparameter 定义部分设置了 Learning_rate 变量用以全局调节学习率。

此外，神经网络输入向量的形状实际上取决于 MFCC 和 FBank 的参数，即梅尔滤波器的数目和对 FBank 进行 DCT（离散余弦变换）生成 MFCC 时保留的频率个数。这两个参数会决定神经网络的输入向量的形状。

笔者在程序中设定的默认值也是前人使用 MFCC 和 FBank 参数的常用值，即 40 个梅尔滤波器，在 DCT 后的 FBank 中保留 13 个频率。

特别地，对使用 **2.2.2** 中正则项的网络，存在用于调节正则项在 loss 中占比大小的超参数 γ。

以上参数都在程序 `task2.ipynb` 和 `task2.py` 的 Hyperparameter 定义部分被定义，它们的默认值如下。

```
# For MFCC and FBank.
Num_filters = 40
Minmax_Ceps = (1,13)
# For Machine Learning Traning.
Training_epoch = 100
Learning_rate = 0.01
# Regularization on Loss.
gamma = 0.1
```

2.2.4. 训练完成后的端点预测

由于最后在 pytorch 上自定义的网络实验效果不佳，因此最终只使用基于 sklearn 预设网络来对无标签的 test 集进行端点预测。

基于 sklearn 的两个网络的最后一层输出均为 1×2 的向量，且由于输出时经过 sigmoid 和 softmax 函数，向量中的 2 个分量之和必定为 1，且都在 $[0,1]$ 范围内。因此其含义分别为非语音和是语音的概率。

如果将整个音频文件的帧集合 S 作为输出，则神经网络输出为 $2 \times |S|$ 的二维矩阵，每一行是对应帧非语音和是语音的概率。

因此，对无标签的音频文件的端点预测伪代码如下。

Algorithm 3: 用基于 sklearn 的神经网络基于 MFCC/FBank 对无标签音频进行语音端点预测

Function $VAD\_DNN(audio, mode)$

// $audio$ 是输入音频，$mode$ 为`"mfcc"` 或`"fbank"`，决定我们使用的是基于 `MFCC` 的 `DNN` 网络还是基于 `FBank` 的 `DNN` 网络。

$S \leftarrow Frame\_Seg(audio)$;

// 将 $audio$ 文件切割为帧，并加窗且预加重。函数中的其它参数如下：$window = Windowing, emph = Emph, emph\_alpha = alpha$。$S$ 是处理后帧的集合。

$MFCCs, FBanks \leftarrow mel(S)$;

// 计算音频每一帧的 `MFCC` 和 `Filter Bank`。函数中的其它参数如下：`num_filter=Num_filters, ceps=Minmax_Ceps`

if *mode is "MFCC"* **then**

  $out \leftarrow DNN\_mfcc(MFCCs)$;

else

  $out \leftarrow DNN\_fbank(FBanks)$;

end

// 其中 `DNN_mfcc` 是基于 `MFCC` 的深度神经网络，`DNN_fbank` 是基于 `FBank` 的深度神经网络

$out \leftarrow out[:, 1]$;

// 只保留是语音的概率，因为另一维度实际上是 `1` 与这一维度值的差，是冗余信息。

$out \leftarrow (out \geq 0.5)$;

// 若 `out[s]` 为 `1`，则 `s` 是语音；若 `out[s]` 为 `0`，则 `s` 不是语音

// 将标记每一帧是否为语音的数组转化为语音端点集合

$pred \leftarrow prediction\_to\_vad\_label(out)$;

Return: $pred$

end

2.3. 实验结果

基于 MFCC、FBank 两种特征，基于 sklearn 的预设和使用 pytorch 自定义构建的四个神经网络训练完毕后，在开发集上的检测效果如表4。

神经网络	特殊处理	AUC	ERR	ACC
MFCC, 预设	无	0.9433	0.1187	0.8947
FBank, 预设	**无**	**0.9778**	**0.0652**	**0.8991**
MFCC, 自定义	无	0.6951	0.3373	0.7165
FBank, 自定义	无	0.7941	0.4022	0.6704
MFCC, 自定义	负采样	0.5832	0.4347	0.8051
FBank, 自定义	负采样	0.6620	0.3989	0.8136
MFCC, 自定义	正则项	0.6230	0.3272	0.7123
FBank, 自定义	正则项	0.6941	0.4193	0.7931

表 4: 基于语音频域特征的神经网络检测效果对比

可以看出，基于 sklearn 预设实现的神经网络效果远远好于笔者使用 pytorch 定义的神经网络。导致这一结果的可能原因有：

- 笔者使用的 loss 函数虽然能提升预测是否为语音（输出 0/1）的准确性，但是从程序的中间输出来看，网络的输出中经常出现“大量 0”和“大量 1”的情况，虽然该结果的“预测准确率”确实升高了，但是 AUC 相对较少，实际上效果变差。因此，笔者设计的 loss 与实际目标并不完全相符。
- 训练轮数不足。由于自定义实现网络 debug 完成后，距离截止时限只有一天，为了能得出结果，笔者将训练轮数下调至 20。20 轮训练并不足以完成我们所定义的神经网络的训练（尤其是对引入负采样的神经网络）。因此，我们自定义的神经网络预测效果较差。

基于以上原因，最终笔者只使用基于 sklearn 预设的 MLP 构建的两个神经网络对测试集（test 集）中的无标签音频进行语音端点检测。结果分别为 `test_label_task.txt` 和 `test_label_task_mfcc.txt`，前者是基于 FBank 的神经网络得出的结果，后者是基于 MFCC 的神经网络得出的结果。

此外可以发现，无论是基于 sklearn 预设还是基于 pytorch 的自定义网络，基于 FBank 的网络预测效果均高于基于 MFCC 的网络。这是由于 MFCC 实际上是 FBank 经非线性变换并舍弃部分信息得到的，其信息含量不如 FBank，因此预测效果会稍微劣于基于 FBank 的神经网络。

值得一体的是，基于 sklearn 预设的两个网络在开发集上效果非常良好，它们极高的 AUC 和极低的 ERR 反而让我有些担忧存在过拟合的问题。但是神经网络是在训练集上训练得到的且在开发集上只测试了一次，且 ACC 并没有高得离谱，因此我认为这一惊人的结果应该不是过拟合的情况。

3. 鸣谢

完成本大作业的过程中，就 **2** 中神经网络的训练与 loss 正则项的设计问题，笔者与**王崇华**、**季弋琨**两位同学进行了讨论。在此特别感谢两位同学的帮助。

4. References

[1] 语音信号的预加重处理和加窗处理,CSDN, https://blog.csdn.net/ziyuzhao123/article/details/12004603