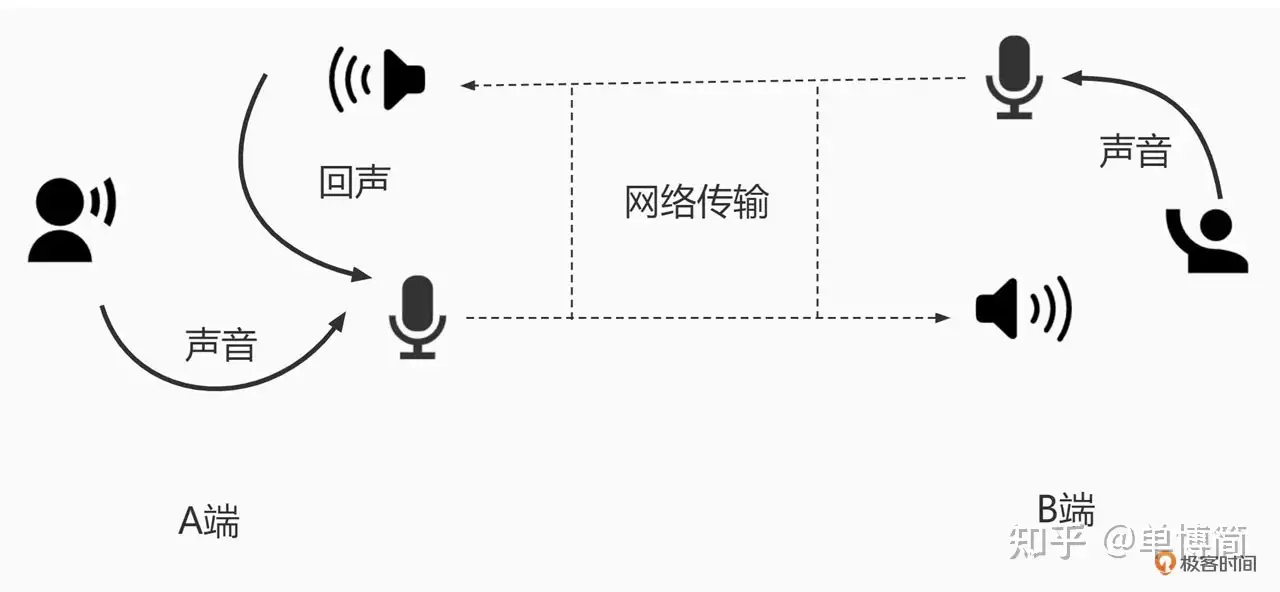
|  |  |  |
| --- | --- | --- |
|  | 不分声道 | 分声道 |
| 录音转文字 | 原始录音：符合模型的训练数据，在整体上更准确。（在杂音非常小的情况下，两者没有区别。） | 过滤对方杂音的数据：在杂音较大，且能与主说话人区分的情况下，更准确一点。 |
| 分清说话人 | 基于聚类：通过声音特征划分 说话人。在说话人语调不变化时基本准确。99。9%能区分，前后是不同的人在说话。但不能保证只识别成2两人，很小概率有3~4个人。 | 直接切分：左声道为客服，右声道为用户。  强先验的做法，简单有效。 |
| 说话交叉片段 | 混杂在一起分，分不清。但如果区别明显，则对模型影响不大。 | 粗暴划分为 声音大的人在说话。没有任何准确性保证。 |
| 不交叉片段 | 在‘单个字’级别上可能出错 | 有明显优势 |
|  |  |  |

电话录音由 双方的麦克风输入 后期合成，所以有两条音轨。



在己方说话，麦克风录音时，己方的扬声器可能在同时输出对方的声音。这个回声被麦克风捕捉到，就混入了杂音。

所有有很多 **回声消除和噪音抑制 算法。**这是非常复杂的问题。非专业公司研究这个没意义。https://www.zhihu.com/question/26686778/answer/2548156131

有个c+写的音频工具库，Webrtc，speex。

还有开源的深度学习算法，<https://github.com/breizhn/DTLN-aec>。

己方扬声器≈对方麦克风，所以要在双声道内，相互消除回声。

那人的做法大概是这样。己方波形为 345，对方为010。那么己方变为 345-010=335（或者不减，因为这可能不是这么算，数字信号和模拟信号不同），对方变为000。己方连续几帧 比对方的帧 平均幅度大时，就判断进入己方说话。先分段，在相减。

现在的语音识别模型，基于真实场景数据训练。实际输入最好和训练的场景一致。简单的消除方法，无法保证没有切分某些成分。消除噪声能做到的，大模型大概率也能做到，况且funasr的模型本身就集成了消除噪声的模块。

阿里的formmer（我这个），是开源可定制‘热词’的离线模型。Large版本，900mb。

阿里的sensevoice(他那个)，是属于大模型结构，主要用于识别多种语言任务，情感分析之类。闭源，只提供了small版本，900mb。

准确率没有多大区别。后者在语音识别没有优势。想听懂某地方言，还是必须要训练。

这里写了一个。和他差不多，但什么价值。需要调几个参数。

import os

from pathlib import Path

from pydub import AudioSegment

os.chdir('e:/code/python/WangXiang')

def pr\_info(audio):

    print(f'时长（秒）：{len(audio) / 1000.0}')

    print(f'声道数（channels）：{audio.channels}')

    print(f'采样率（frame\_rate）：{audio.frame\_rate}')

    print(f'帧宽（sample\_width）：{audio.sample\_width} 字节')

# 分左右

audio = AudioSegment.from\_file(Path('data/record/audio/19.mp3'))

channels = audio.split\_to\_mono()

left\_channel = channels[0]

right\_channel = channels[1]

segment\_length = 200  # 毫秒

# 分切片，每个切片200ms

def segment\_audio(audio, segment\_length):

    segments = []

    for start\_ms in range(0, len(audio), segment\_length):

        segment = audio[start\_ms : start\_ms + segment\_length]

        segments.append(segment)

    return segments

left\_segments = segment\_audio(left\_channel, segment\_length)

right\_segments = segment\_audio(right\_channel, segment\_length)

n = len(left\_segments)

# 判断连续

flags = []

for idx in range(n):

    if left\_segments[idx].rms > right\_segments[idx].rms:

        flags.append(0)

    else:

        flags.append(1)

def modify\_segments(arr, min\_sec):

    min\_count = int(min\_sec / (segment\_length / 1000.0))

    print(f'最小连续段数：{min\_count}')

    i = 0

    while i < len(arr):

        count = 1

        while i + 1 < len(arr) and arr[i] == arr[i + 1]:

            count += 1

            i += 1

        if count < min\_count:

            for j in range(i - count + 1, i + 1):

                arr[j] = 1 - arr[i]

        i += 1

    return arr

# arr = [0, 0, 0,0,0,0,0,0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]

# 最短持续的秒数 ，1

modified\_flags = modify\_segments(flags, 1)

# 看结果

print(modified\_flags)

# 声音小的降为0

for idx in range(n):

    duration = len(left\_segments[idx])

    if flags[idx] == 1:

        left\_segments[idx] = AudioSegment.silent(duration=duration, frame\_rate=8000)

    else:

        right\_segments[idx] = AudioSegment.silent(duration=duration, frame\_rate=8000)

# 合并切片

combined\_left\_audio = AudioSegment.empty()

for seg in left\_segments:

    combined\_left\_audio += seg

combined\_right\_audio = AudioSegment.empty()

for seg in right\_segments:

    combined\_right\_audio += seg

# 合并左右

combined\_audio = AudioSegment.from\_mono\_audiosegments(combined\_left\_audio, combined\_right\_audio)

combined\_audio