哈爾濱工業大學

实验报告

实 验（一）

题　　目 语音信号的端点检测

专　　业 人工智能-视听觉处理

学　　号 1190201215

班　　级 1903602

学　　生 冯开来

指 导 教 师 郑铁然

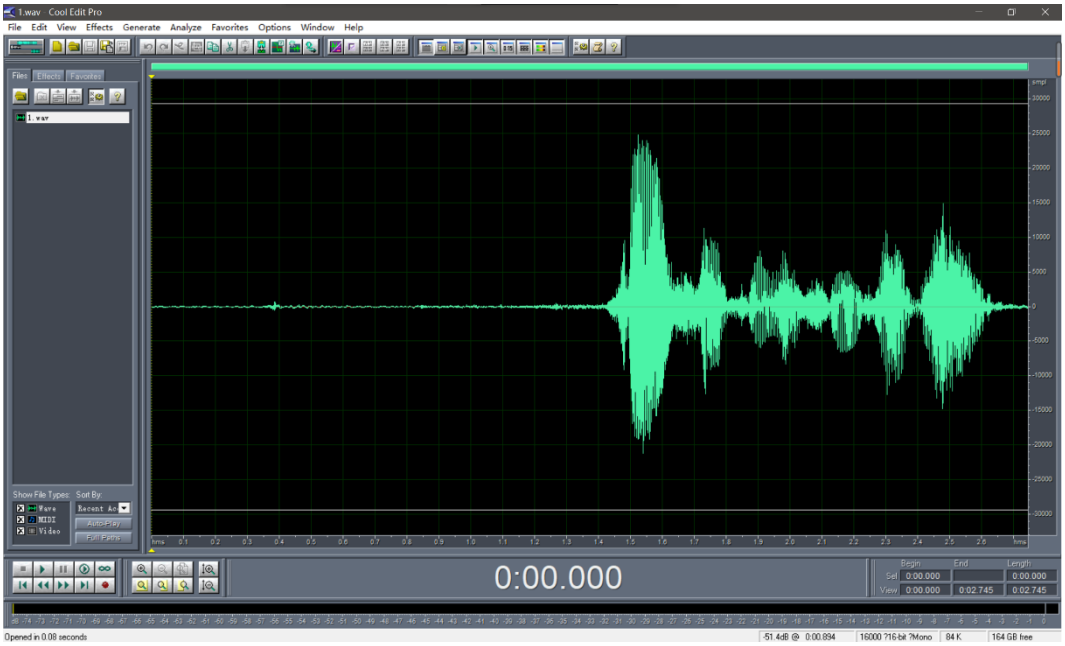
实 验 地 点 格物 207

实 验 日 期 12.4

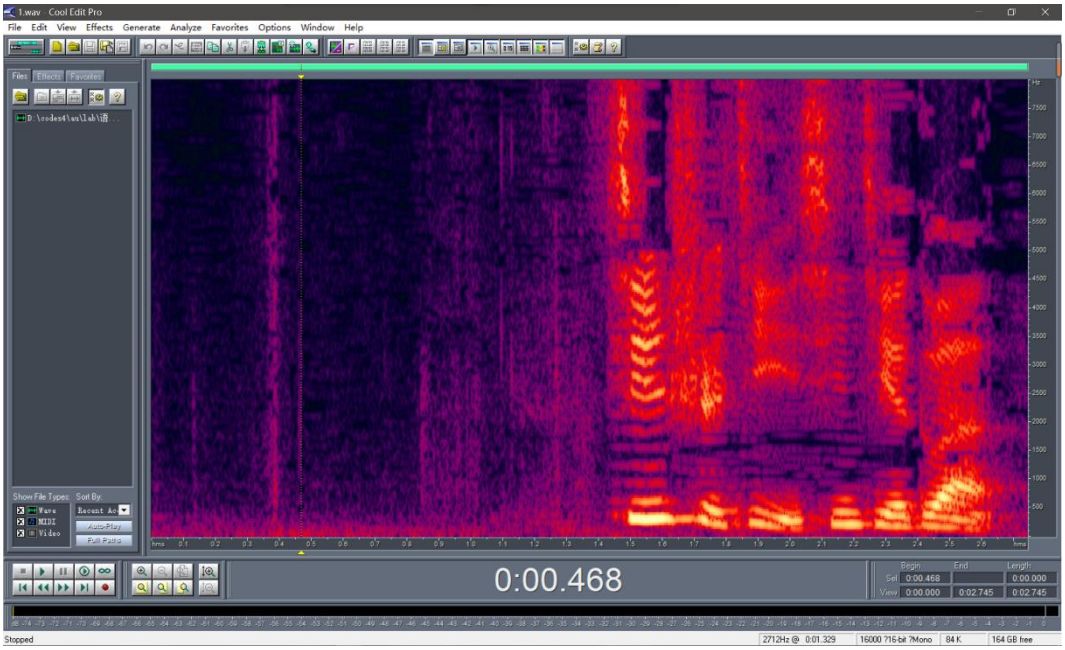
计算机科学与技术学院

一、 语音编辑和处理工具的使用

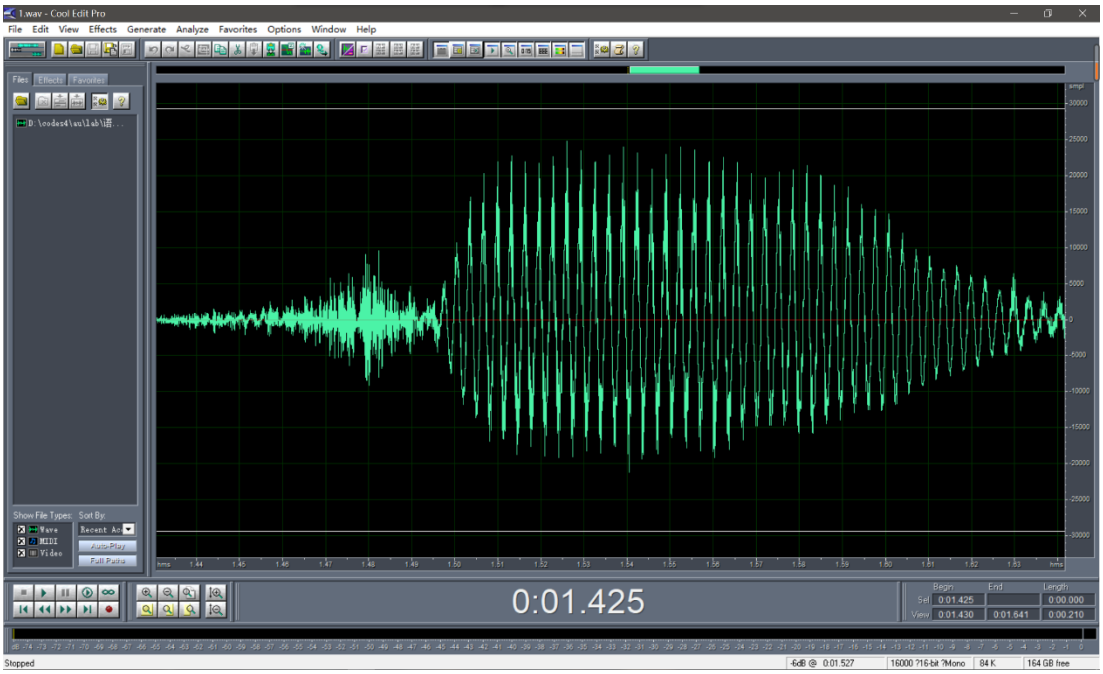
1.1 语音文件“1.wav”的时域波形截图



1.2 语音文件“1.wav”的语谱图截图



1.3 “1.wav”第一个音节的时域波形截图



1.4 语料的格式

采样频率　= 16000
量化比特数= 16-bit
声道个数　= 1

二、能量和过零率特征提取

2.1 给出特征提取算法，给出概要性介绍，标明所采用的开发工具

1) 开发工具

PyCharm Community Edition 2021.2.2；Windows 10；Python 3.9

2) 文件读取和分帧

第一步使用 wave 库读取音频文件，得到四个参数：声道数，量化位数，采样频率，采样点数（其实有 6 个参数，但是 wave 格式，后两个参数没有也不重要）。通过 readframes 得到字节流，因为是 16-bit 的采样值，所以我们将字节流转换成 short 类型，最后通过采样点数和采样频率我们可以得到每个样点的时间。

```
f = wave.open(file_path, "rb")
params = f.getparams()
# 声道数，量化位数，采样频率，采样点数
nchannels, sampwidth, framerate, nframes = params[:4]
# print(nchannels, sampwidth, framerate, nframes)
# 得到每个采样点的值
str_data = f.readframes(nframes)
f.close()
# 转成short类型
wave_data = np.frombuffer(str_data, dtype=np.short)
time = np.arange(0, nframes) / framerate
```

在这一步中我们可以对数据进行归一化处理。

```
# 归一化处理
# wave_data = wave_data * 1.0 / (max(abs(wave_data)))
# print(wave_data[:10])
```

第二步是对采样的数据进行分帧，通过上一个步骤我们得到一个一维数组，根据题目要求得到帧长是 256 个采样点，无帧叠，所以很简单的就可以将原来的一维数组 256 个采样点位一帧，分成若干帧，最后一帧不满 256 的我采取补零的操作。

```
signal_len = len(wave_data) # 信号总长度
fn = int(np.ceil((signal_len) / wlen))  # 帧数
# 可能存在帧数x采样点数量，计算得到实际长度大于本身信号长度，补零
zeros = np.zeros(fn * wlen - signal_len)
wave_data_zero = np.concatenate((wave_data, zeros))
```

在第二步中，值得一提的是，实际上分帧的时候 wlen 为帧长，inc 为帧移，overlap 为帧叠，信号总长度为 signal_len，满足 overlap = wlen – inc，所

以在分帧的过程中更为复杂，首先得到帧数 fn = (signal_len – overlap) / wlen 并向下取整，然后进行同样的操作进行分帧。

最后对于每一帧，我们可以调用海明窗。当然，本次实验中并没有调用。

```
    # window = np.hamming(wlen) # 调用海明窗
    # frames = frames * window # 信号加窗
    # for i in range(fn):
    #     frames[i] = frames[i] * window
```

3) 计算每一帧能量

对于每一帧来说，计算能量公式为

$$E_n = \sum_{m=-\infty}^{+\infty} [x(m)w(m)]^2$$

其实就是求每一帧里面，每个值的平方最后求和，然后得到了关于每一帧的能量的数组，保存至 i_en.txt 中：

```
# 计算每一帧能量
def compute_en(frames, i):
    fn, wlen = frames.shape
    en = np.zeros((fn, 1))
    for n in range(fn):
        en[n] = np.dot(frames[n], frames[n].T)
    np.savetxt("能量\\" + str(i+1) + "_en.txt", en, fmt="%.2f")
    return en
```

4) 计算每一帧过零率

对于每一帧来说，短时过零率的公式为：

$$Z_n = \sum_{m=-\infty}^{+\infty} |sgn[x(m)] - sgn[x(m-1)]|w(m)$$

$$sgn[x(n)] = \begin{cases} 1 & x(n) \geq 0 \\ -1 & x(n) < 0 \end{cases} \qquad w(n) = \begin{cases} \frac{1}{2N} & 0 \leq n \leq N-1 \\ 0 & n < 0,\ n > N-1 \end{cases}$$

按照公式，带入算一下，其实在实际运算过程中，如果 x(m)和 x(m-1)异号，将过零率值加一，最后除以 N 即可。

```
# 计算每一帧过零率
def compute_zcr(frames, i):
    fn, wlen = frames.shape
    zcr = np.zeros((fn, 1))
    for m in range(fn):
        for n in range(wlen-1):
            # zcr[m] += np.abs(sgn(frames[m, n+1]) - sgn(frames[m, n]))
            if (frames[m, n] < 0 and frames[m, n+1] >=0) or (frames[m, n] >= 0 and frames[m, n+1] < 0):
                zcr[m] += 2
            else:
                zcr[m] += 0
        zcr[m] /= (2*wlen)
    np.savetxt("过零率\\" + str(i + 1) + "_zero.txt", zcr, fmt="%.8f")
    return zcr
```

5) 代码

```
# 读取文件
def getWaveData(file_path):
    f = wave.open(file_path, "rb")
    params = f.getparams()
    # 声道数，量化位数，采样频率，采样点数
    nchannels, sampwidth, framerate, nframes = params[:4]
    # print(nchannels, sampwidth, framerate, nframes)
    # 得到每个采样点的值
    str_data = f.readframes(nframes)
    f.close()
    # 转成 short 类型
    wave_data = np.frombuffer(str_data, dtype=np.short)
    time = np.arange(0, nframes) / framerate
    # print(nframes)
    # 归一化处理
    # wave_data = wave_data * 1.0 / (max(abs(wave_data)))
    # print(wave_data[:10])
    # 通过采样点数和取样频率计算每个取样的时间
    return wave_data, time, nchannels, sampwidth, framerate

# 分帧
def findSegment(wave_data, wlen=256, inc=256):
    # 无帧叠，帧长等于帧移，默认值为 256
    signal_len = len(wave_data) # 信号总长度
    fn = int(np.ceil((signal_len) / wlen))  # 帧数
    # 可能存在帧数 x 采样点数量，计算得到实际长度大于本身信号长度，补零
    zeros = np.zeros(fn * wlen - signal_len)
    wave_data_zero = np.concatenate((wave_data, zeros))
    # 给每个采样点加上标签，没用
    # indices = np.tile(np.arange(0, wlen), (fn, 1)) +
np.tile(np.arange(0, fn*inc, inc), (wlen, 1)).T
    # print(indices[171:172])
    # indices = np.array(indices, dtype=np.int32)
    # frames = wave_data_zero[indices]
    # 信号分帧，fn * wlen
    frames = np.array(wave_data_zero).reshape(fn, wlen)
    # print(frames)
    # window = np.hamming(wlen) # 调用海明窗
    # frames = frames * window # 信号加窗
    # for i in range(fn):
    #     frames[i] = frames[i] * window
    return frames

# 计算每一帧能量
def compute_en(frames, i):
    fn, wlen = frames.shape
    en = np.zeros((fn, 1))
    for n in range(fn):
        en[n] = np.dot(frames[n], frames[n].T)
    np.savetxt("能量\\" + str(i+1) + "_en.txt", en, fmt="%.2f")
    return en
```

```
# 计算每一帧过零率
def compute_zcr(frames, i):
    fn, wlen = frames.shape
    zcr = np.zeros((fn, 1))
    for m in range(fn):
        for n in range(wlen-1):
            # zcr[m] += np.abs(sgn(frames[m, n+1]) - sgn(frames[m,
n]))
            if (frames[m, n] < 0 and frames[m, n+1] >=0) or
(frames[m, n] >= 0 and frames[m, n+1] < 0):
                zcr[m] += 2
            else:
                zcr[m] += 0
        zcr[m] /= (2*wlen)
    np.savetxt("过零率\\" + str(i + 1) + "_zero.txt", zcr,
fmt="%.8f")
    return zcr
```

三、 端点检测算法

3.1 给出端点检测算法，给出概要性介绍，标明所采用的开发工具

1) 开发工具

PyCharm Community Edition 2021.2.2；Windows 10；Python 3.9

2) 端点检测

本次实验使用的是双门限法进行端点检测，对语音进行“浊音/清音/无音”的判断，在汉语中，浊音处于音节的末尾容易通过短时能量区别，但在音节的前端，清音和环境噪声很难区分。浊音能量高于清音，清音的过零率高于无音。

在实际操作中，我让**能量的高门限为（所有帧平均能量/4），能量的低门限为（高门限能量/4），过零率门限为（所有帧过零率平均值-前五帧平均值/5）**至于为什么这样选择，是因为实践试出来的。

```
enHighThr = np.average(en) / 4 # 高能量阈值
enLowThr = enHighThr / 4 # 低能量阈值
zcrThr = np.average(zcr) - 0.2 * np.average(zcr[:5]) # 过零率阈值

new = []
labels = np.zeros(fn) # 值为status
status = 0  # 0:无声段, 1:清音段, 2:语音段
```

然后对每一帧能量进行筛选，先找到能量高于高门限的部分，标签记为 2，然后分别向前和向后遍历，找到门限高于低门限的部分，也标为 2。最后就是区别出清音和无音部分，继续向前和向后遍历，过零率高于门限的为清音部分，标为 1，其他均为 0。

```
for i in range(fn):
    if en[i] >= enHighThr:
        k = j = i
        # 语音部分
        while (en[j] >= enLowThr and j < fn):
            labels[j] = 2
            j += 1
        while (en[k] >= enLowThr and k >= 0):
            labels[k] = 2
            k -= 1
        # 清音
        while (j < fn):
            if zcr[j] >= zcrThr:
                labels[j] = 1
            j += 1
        while (zcr[k] >= zcrThr and k >= 0):
            k -= 1
            labels[k] = 1
```

最后将标签非 0 的部分连接在一起，得到新的语音段，即端点检测的结果。

3) 写入文件

本次实验的写入文件花了我很多时间，主要是因为在得到新的帧数组后设置同样的参数以 wave 格式写入文件发现声音变得奇怪，只有将采样频率调成两倍才正常，或者将单声道调成双声道声音在正常。

仔细分析发现这都和占的字节有关。根据查阅资料发现好像写入的时候是按照字节流写入，如果我将端点检测后的数据简单转换成 int 类型那就是占 16 位的整数，而单声道，16000 采样频率的 wave 文件应该是占 8 位的整数，既然整体所占的位数多了一倍，那确实会出现音频频率不正常的情况，所以在转成 int 类型的时候注意要转成 int16。

```python
def outputWaveData(frames, i, nchannels, sampwidth, framerate):
    # wave_data = []
    frames = frames.flatten().astype('int16')
    # frames = np.frombuffer(frames, dtype=np.short)
    # for n in range(len(frames)):
    #     if (frames[n] > 0):
    #         wave_data.append(frames[n])
    wave_data = np.array(frames)
    # wave_data.dtype = 'int32'
    print(wave_data, len(wave_data))
    f = wave.open("新语料\\" + str(i + 1) + ".wav", "wb")
    # f.setnchannels(nchannels)
    # f.setsampwidth(sampwidth)
    # f.setframerate(framerate)
    f.setparams((nchannels, sampwidth, framerate, len(wave_data), 'NONE', 'HIT-1190201215'))
    f.writeframes(wave_data)
    f.close()
```

4) 代码

```python
# 端点检测
def endPointCheck(frames, en, zcr):

    fn, wlen = frames.shape

    enHighThr = np.average(en) / 4 # 高能量阈值
    enLowThr = enHighThr / 4 # 低能量阈值
    zcrThr = 1 * np.average(zcr[:5]) # 过零率阈值

    new = []
    labels = np.zeros(fn) # 值为status
    status = 0  # 0:无声段, 1:清音段, 2:语音段

    for i in range(fn):
        if en[i] >= enHighThr:
            k = j = i
            # 语音部分
            while (en[j] >= enLowThr and j < fn):
                labels[j] = 2
                j += 1
            while (en[k] >= enLowThr and k >= 0):
                labels[k] = 2
                k -= 1
            # 清音
            while (j < fn):
```

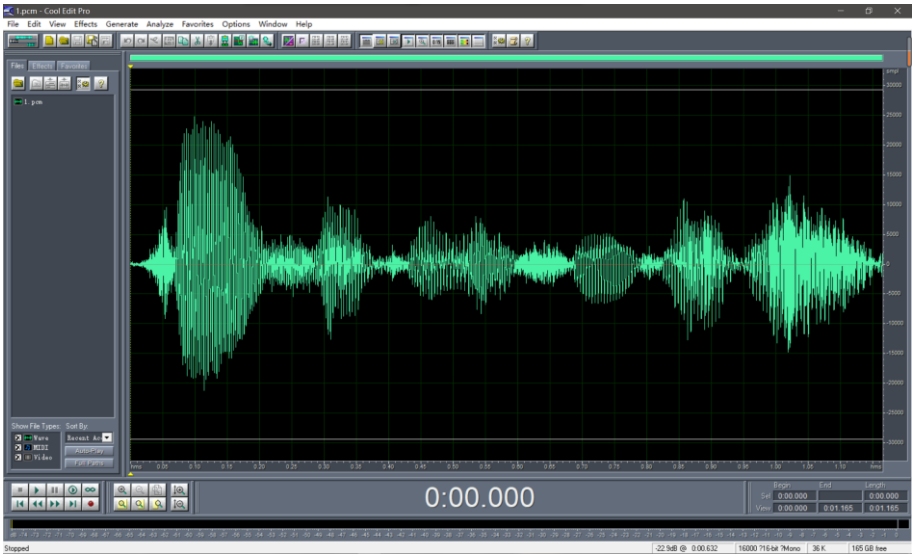
```
                if zcr[j] >= zcrThr:
                    labels[j] = 1
                j += 1
            while (zcr[k] >= zcrThr and k >= 0):
                k -= 1
                labels[k] = 1
    # print(labels)
    for i in range(fn):
        if labels[i] != 0:
            new.append(frames[i])
    new = np.array(new)

    return new

# 写入文件
def outputWaveData(frames, i, nchannels, sampwidth, framerate):
    # wave_data = []
    frames = frames.flatten().astype('int16')
    # frames = np.frombuffer(frames, dtype=np.short)
    # for n in range(len(frames)):
    #     if (frames[n] > 0):
    #         wave_data.append(frames[n])
    wave_data = np.array(frames)
    # wave_data.dtype = 'int32'
    print(wave_data, len(wave_data))
    f = wave.open("新语料\\" + str(i + 1) + ".wav", "wb")
    # f.setnchannels(nchannels)
    # f.setsampwidth(sampwidth)
    # f.setframerate(framerate)
    f.setparams((nchannels, sampwidth, framerate, len(wave_data),
'NONE', 'HIT-1190201215'))
    f.writeframes(wave_data)
    f.close()
```

四、 计算检测正确率

4.1 “1.wav”语料去除静音后的时域波形截图



4.2 正确率

正确检出文件的个数：10

正确率 = 100%

五、 总结

5.1 请总结本次实验的收获

1. 学会了使用 Cool Edit 软件，知道了怎么查看波形图和语谱图，同时会一点剪辑操作。
2. 学会了如何提取 wave 文件，wave 格式文件的头部参数分别有意义
3. 学会了如何对音频数据进行分帧，计算能量，过零率
4. 学会了如何利用能量和过零率进行端点检测，去除语音的静音和噪音部分
5. 学会了以 wave 格式写入文件，特别是转换数据类型时一定要注意。

5.2 请给出对本次实验内容的建议

希望不要用 Cool Edit 软件，改用 Adobe Audition，毕竟前者有点老，后者更新功能也更强大。