信号建模与算法实践

MFCC 特征提取

Xu Weiye

2022-09-12

Content

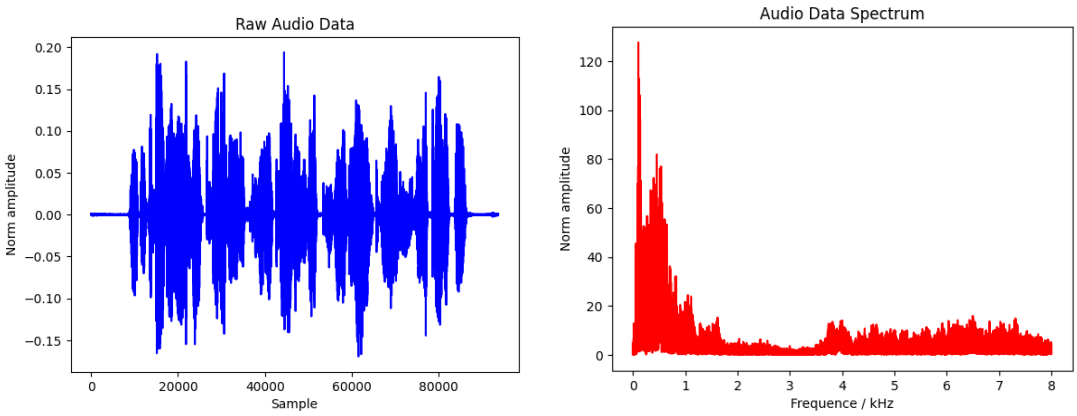
1 Introduction

语音处理在语音系统中都扮演着重要的角色，无论它是自动语音识别（ASR）还是说话者识别等等。长期以来，梅尔频率倒谱系数（MFCC）是非常受欢迎的特征。简而言之，信号通过预加重滤波器；然后将其切成（重叠的）帧，并将窗函数应用于每个帧；之后，我们在每个帧上进行傅立叶变换（或更具体地说是短时傅立叶变换），并计算功率谱；然后计算滤波器组。为了获得 MFCC，可将离散余弦变换（DCT）应用于滤波器组，以保留多个所得系数，而其余系数则被丢弃。

2 Process

2.1 Setup

对实验原始数据 1.wav 进行读取到程序中得到其采样频率是 16kHz 并且有大概 10s 左右的时，由于此音频是 16bit 量化的所以可以预先进行一次归一化，然后再进行之后的处理，之后再使用快速傅里叶变换 (FFT) 可以进一步进行可视化后得到以下结果



(a) 原始信号的归一化幅度图　　(b) 原始信号的频谱图

Figure 1: 将原始音频读入并且经过傅里叶变换

```
fs, data = wavfile.read('1.wav') # 音频数据导入
audioData = [(ele/2**16.) for ele in data]
audioDataFFT = fft(audioData)
```

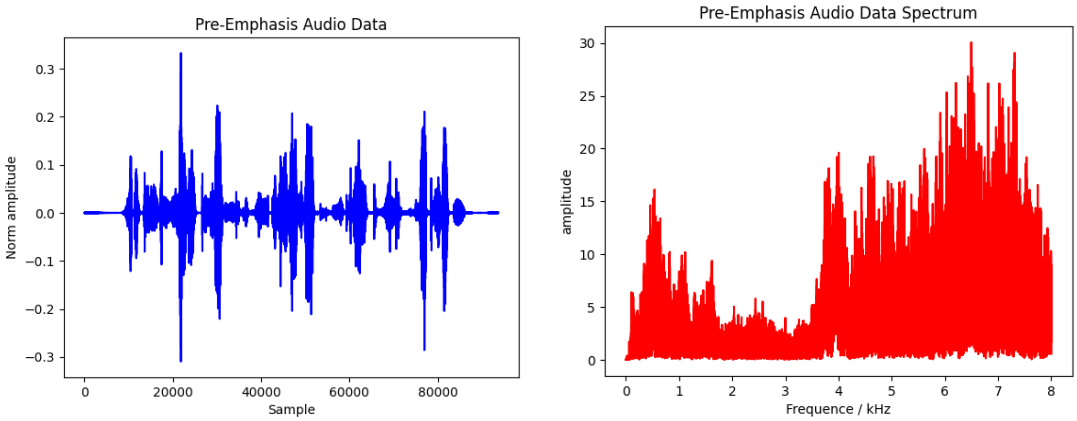
2.2 Pre-Emphasis

第一步是在信号上施加预加重滤波器，以放大高频。预加重滤波器在几种方面有用：

- 平衡频谱，因为高频通常比低频具有较小的幅度
- 避免在傅立叶变换操作期间出现数值问题
- 改善信号噪声比（SNR）预加重滤波器可以使用一阶滤波器应用于信号

预加重的可以使用一阶滤波器, 其中 α 一般设置成 0.97

$$y(t) = x(t) - \alpha x(t-1)$$



(a) 预加重信号的归一化幅度图 (b) 预加重信号的频谱图

Figure 2: 将原始音频经过预加重

在代码中我们需要注意的是音频中的第一个值不需要处理，这样得到的第一个份音频会正好整除之后的分帧长度

```
dataPreEmphasis = [audioData[ii]-alpha * audioData[ii-1] for ii in range(1,lengthAudio)]
dataPreEmphasis.append(audioData[0])
dataPreEmphasisFFT = fft(dataPreEmphasis)
```

2.3 Framing

经过预加重后，我们需要将信号分成短时帧。此步骤的基本原理是信号中的频率会随时间变化，因此在大多数情况下，对整个信号进行傅立叶变换是没有意义的，因为我们会随时间丢失信号的频率轮廓。为避免这种情况，我们可以安全地假设信号的频率在很短的时间内是固定的。因此，通过在此短时帧上进行傅立叶变换，我们可以通过串联相邻帧来获得信号频率轮廓的良好近似值。语音处理中的典型帧大小范围为 20 ms 至 40ms，连续帧之间有 50%(+/-10%) 重叠。实验中的设置是帧长为 25ms，帧移为 10ms（重叠 15ms）经过分帧后得到第一帧的幅度图为

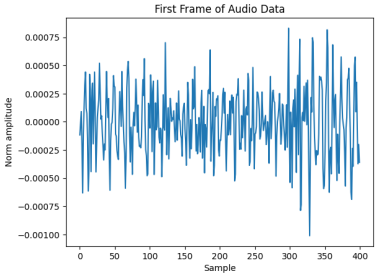


Figure 3: 经过分帧操作处理

在代码中我们通过对上一步得到的信号按照帧长和帧移进行分割

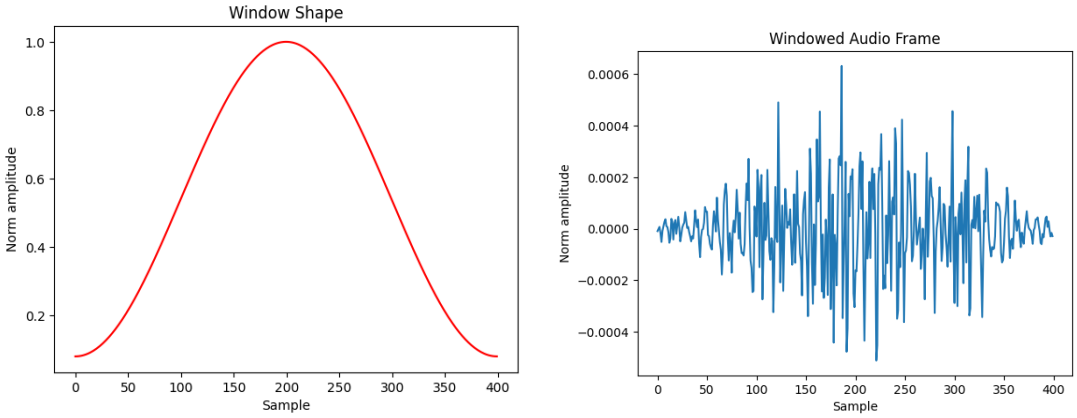
```
frameLen = int(fs*0.025) # 帧长为 25ms
gapLen = int(fs * 0.01)
audioList = [dataPreEmphasis[ii*gapLen:ii*gapLen + frameLen]
  for ii in range(1+int((lengthAudio-frameLen)/gapLen))]
```

2.4 Window

将信号切成帧后，我们对每个帧应用诸如汉明窗之类的窗函数。汉明窗具有以下形式

$$w[n] = 0.54 - 0.46cos(\frac{2\pi n}{N-1})$$

其中，$0 \leq n \leq N-1$, N 是窗长。



(a) 窗函数时域形式

(b) 加窗后单帧的图表

Figure 4: 将分帧信号加窗

在代码中可以使用数组相乘的 broadcast 特性，将多帧信号和一个窗函数直接相乘

```
w = [0.54 - 0.46 * np.cos(2*np.pi*ii/(frameLen-1)) for ii in range(frameLen)]
dataWindowed = [np.array(audioList[ii]) * np.array(w) for ii in range(len(audioList))]
```

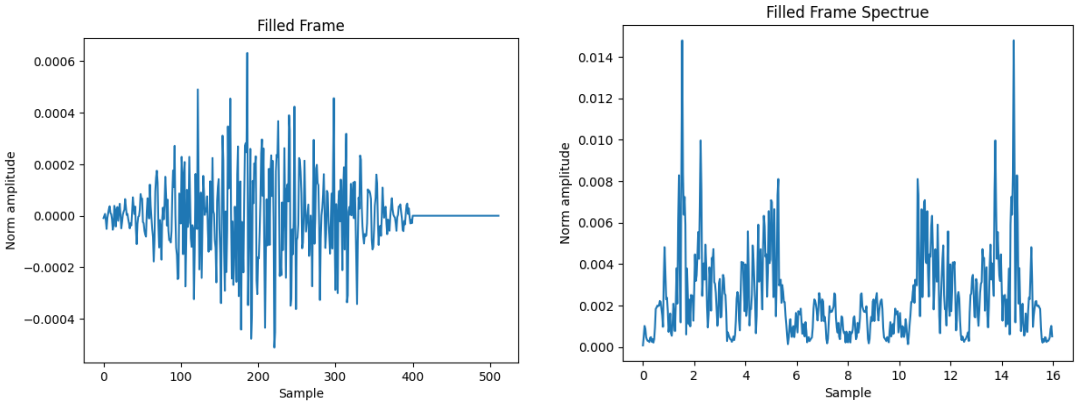
2.5 Fourier-Transform and Power Spectrum

对窗长 N 的每个帧进行补零填充，以形成一个扩展帧，该帧包含 512 个样本（16 kHz）。使用长度为 512 的 FFT 来计算信号的幅度谱。按照如下公式进行傅里叶变换

$$bin_k = \sum_{n=0}^{N-1} x_i(n) e^{-jnk\frac{2\pi}{N}} \quad k = 0, 1, ..., N-1$$

功率谱密度

$$P = \frac{FFT(x_i)^2}{N}$$



(a) 补零填充时域图 (b) 傅里叶变换后频谱图

Figure 5: 将分帧信号傅里叶变换

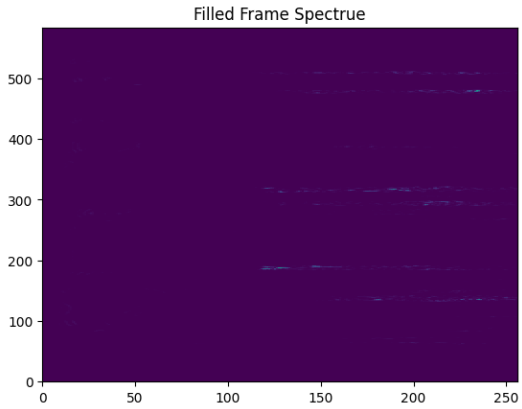


Figure 6: 功率频谱图

代码中可以使用内置函数进行补零，并且在代码中使用整个数列的。可以从中看出对称性

```
dataPad512 = np.pad(dataWindowed,((0,0),(0,512-np.shape(dataWindowed)[1])),'constant')
dataFFT512 = fft(dataPad512)
PowerFFT512 = np.power(abs(dataFFT512),2)/512
PowerFFT512 = PowerFFT512[:,:256]
```

2.6 Filter Bank

计算滤波器组是将三角滤波器在 Mel 刻度上应用于功率谱以提取频带，通常设置成 40 个滤波器。梅尔音阶的目的是模仿低频的人耳对声音的感知，方法是在较低频率下更具判别力，而在较高频率下则具有较少判别力,Hertz(f) 和 Mel(m) 之间的转换关系为：

$$m = 2595log_{10}(1 + \frac{f}{700})$$

$$f = 700 * (10^{\frac{m}{2595}} - 1)$$

得到的频率组为 在书写代码的时候会有一些遇到一个问题就是梅尔频率是在梅尔频率域内均

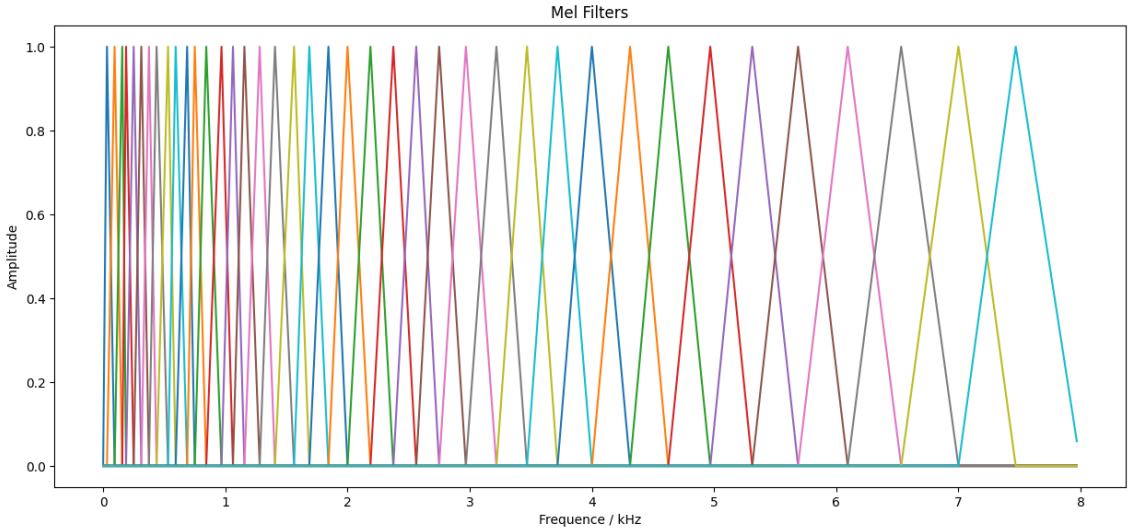


Figure 7: 梅尔滤波器组

匀采样，但是到实际频域的会是非线性采样如图7可以看出在实际中是低频密集，高频稀疏，如果直接按照在实际频域中均匀采用，会导致没法把滤波器的主要部分很好的取出来，得到如下结果 所以

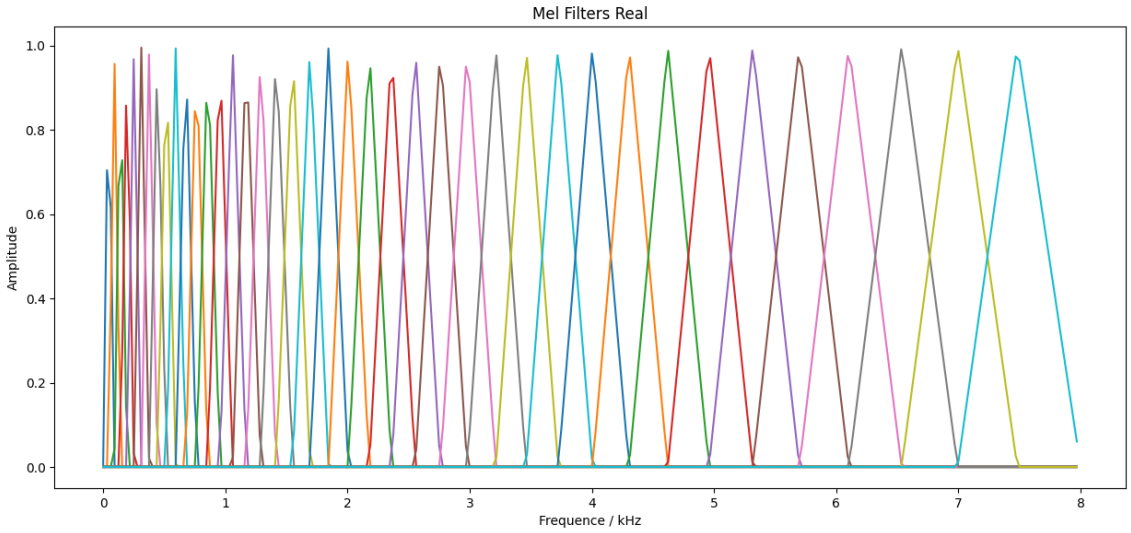


Figure 8: 原始梅尔滤波器组

在实际代码中我采用了取整的方式

```
def getMelFilterList(MelFilterN):
MMax = 2595 * np.log10(1+fs/1400)
MelStep = MMax/41
MelFrequenceList = [700*(10**(ii*MelStep/2595)-1) for ii in range(42)]
MelFilterList = list()
```

```
    fStep = fs/2/MelFilterN
    # 可以比较得到将 Mel-frequence 移动到最近的频率 会更好
    MelFrequenceList = np.rint(np.array(MelFrequenceList)/fStep)*fStep
    print(MelFrequenceList)
    for ii in range(40):
        filter = list()
        for fn in range(MelFilterN):
            if fStep * fn < MelFrequenceList[ii] or fStep * fn > MelFrequenceList[ii+2]:
                filter.append(0)
            elif fStep * fn < MelFrequenceList[ii+1]:
                filter.append((fStep * fn - MelFrequenceList[ii])/(MelFrequenceList[ii+1] -
                    MelFrequenceList[ii]))
            else:
                filter.append((- fStep * fn + MelFrequenceList[ii+2])/(MelFrequenceList[ii+2] -
                    MelFrequenceList[ii+1]))
        MelFilterList.append(filter)
    return MelFilterList
```

通过如上函数可以得到较好结果。最后和功率谱相乘再经过非线性变换：

$$fBank_m = H_m P$$

$$f_m = 10 * log_{10}(fBank_m)$$

可以得到效果如图

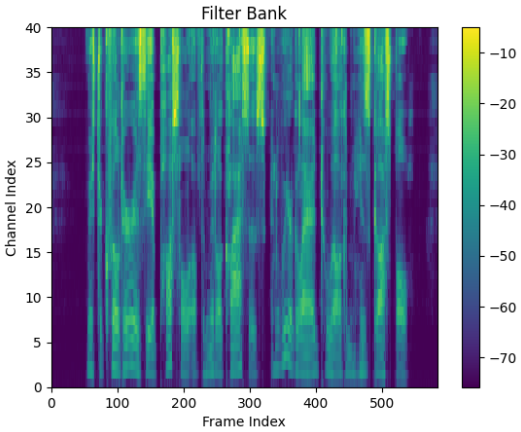


Figure 9: 滤波器组直接得到的结果

2.7 Mel-frequency Cepstral Coefficients

在上一步中计算出的滤波器组系数是高度相关的，这在某些机器学习算法中可能会出现问题。因此，我们可以应用离散余弦变换（DCT）对滤波器组系数进行解相关，并生成滤波器组的压缩表

示形式。这里选取前 13 维作为最终的倒谱系数。

$$C_i = \sum_{j=0}^{39} f_i \cos(\frac{i\pi(j-0.5)}{40}) \quad 0 \le i \le 12$$

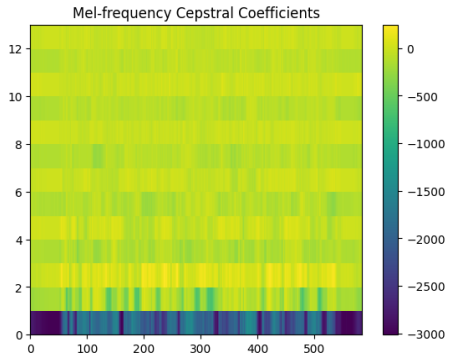


Figure 10: 倒谱系数

3 extantion

3.1 More experiment

对于 2.wav ，其处理方式和之前没有特别的差异这里就只分析其运行时间:

运行在 Intel Xeon Gold 6242R CPU @ 3.10GHz 并且需要绘制所效果图 (绘图会占用大部分时间)

Table 1: 运行时间分析

处理流程	运行时间 / s
Import Packages	0.1
Import Audio Data	10.7
Pre-Emphasis	25.2
Framing	0.7
Window	3.1
Fourier-Transform & Power Spectrum	111.9
Filter Bank	21.2
Mel-frequency Cepstral Coefficients	11.3
TOTAL	**184.2**

3.2 About Machine Learning

在机器学习里，如果是小数据量可以使用 fBank 特征即可，如果是大数据量，需要更多的特征所以会选择 MFCC 特征