实验一 语音信号 MFCC 特征提取

王瑞哲 PB19071509

>>> 实验原理

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

>>> 实验目标

掌握整个 MFCC 特征提取过程，并对提供的音频提取 MFCC 特征

掌握利用python编程语言的基本使用方法，并利用其进行音频信号的特征分析

>>> 实验过程

0. 运行环境准备与音频信号导入

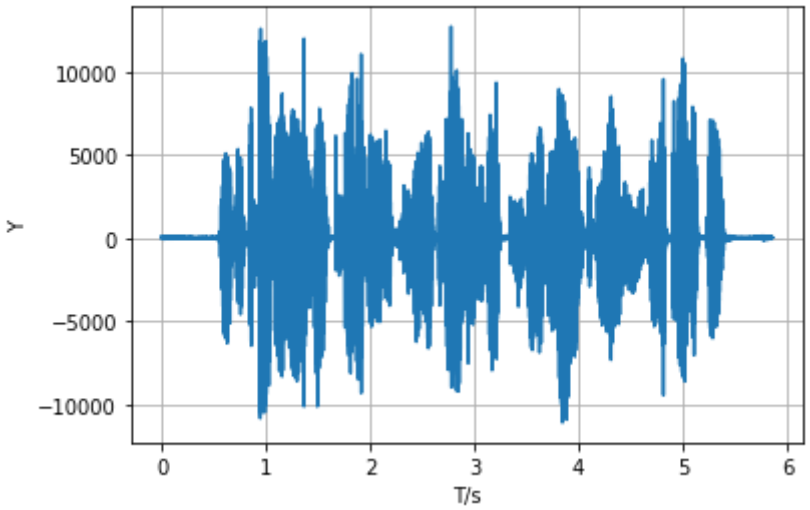
本次实验采用python编程语言，采用JupyterNotebook编辑运行代码

首先导入numpy、matplotlib等必要库，以及对音频分析所需要的库scipy（为了实现wav信号的导入以及DCT变换）：

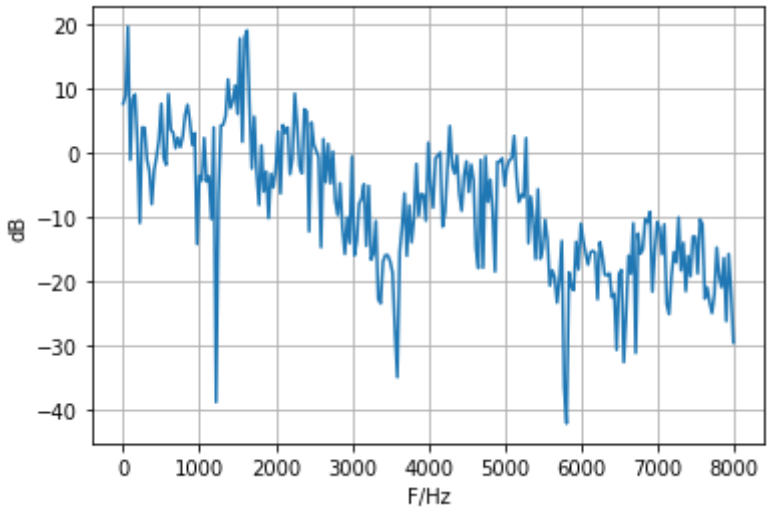
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.io import wavfile
from scipy.fftpack import dct

from utils import tdraw,fdraw,gdraw
```

利用wavfile导入音频信号1.wav，并观察其时域信号特征：

```
sample_rate, s = wavfile.read('./1.wav')
tdraw(s, sample_rate)
```



对信号做傅里叶变换，可得到其傅里叶频谱图：



1. 预加重（Pre-Emphasis）

通过这一处理来放大高频，使用一阶滤波器$y(t) = x(t) - \alpha x(t-1),\quad 0.95 < \alpha < 0.99$，并选择$\alpha = 0.97$

```
alpha = 0.97
s = np.append(s[0], s[1:] - alpha * s[:-1])
```

这一步用于平衡频谱，避免在傅立叶变换操作期间出现数值问题，同时还可改善信号噪声比。

2. 成帧（Framing）

为避免对整个信号进行傅立叶变换时随时间丢失信号的频率轮廓，假设信号的频率在很短的时间内是固定的，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。 选择帧长为 25ms，帧移为 10ms（重叠 15ms）

```
frame_size, frame_stride = 0.025, 0.01
frame_length, frame_step = int(round(frame_size * sample_rate)),
int(round(frame_stride * sample_rate))
signal_length = len(s)
num_frames = int(np.ceil(np.abs(signal_length - frame_length) / frame_step)) + 1

pad_signal_length = (num_frames - 1) * frame_step + frame_length
z = np.zeros((pad_signal_length - signal_length))
pad_signal = np.append(s, z)

indices = np.arange(0, frame_length).reshape(1, -1) + np.arange(0, num_frames *
frame_step, frame_step).reshape(-1, 1)
frames = pad_signal[indices]
```

3. 加窗（Window）

选择汉明hamming窗$w(n) = 0.54 - 0.46cos(\frac{2\pi n}{N-1})$

这一步可以直接利用numpy库的内置函数实现

```
hamming = np.hamming(frame_length)
frames *= hamming
```

4. 傅立叶变换和功率谱(Fourier-Transform and Power Spectrum)

选取窗长为$N = 512$，利用numpy库中所提供的FFT函数

```
N = 512
mag_frames = np.absolute(np.fft.rfft(frames, N))
pow_frames = ((1.0 / N) * (mag_frames ** 2))
```

5. 滤波器组（Filter Banks）与FBANK特征

经过上面的步骤之后，在能量谱上应用Mel滤波器组，就能提取到FBank特征。

Mel刻度是一个能模拟人耳接收声音规律的刻度，人耳在接收声音时呈现非线性状态，对高频的更不敏感，因此Mel刻度在低频区分辨度较高，在高频区分辨度较低，与频率之间的换算关系为：

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

$f = 700(10^{m/2595} - 1)$

Mel滤波器组就是一系列的三角形滤波器，通常有40个或80个，在中心频率点响应值为1，在两边的滤波器中心点衰减到0。具体公式可以写为：

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k < f(m) \\ 1 & k = f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \le f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

最后在能量谱上应用Mel滤波器组，其公式为：

$Y_t(m) = \sum_{k=1}^{N} H_m(k)|X_t(k)|^2$

其中，k表示FFT变换后的编号，m表示mel滤波器的编号。

代码部分，首先计算Mel刻度的高低频成分：

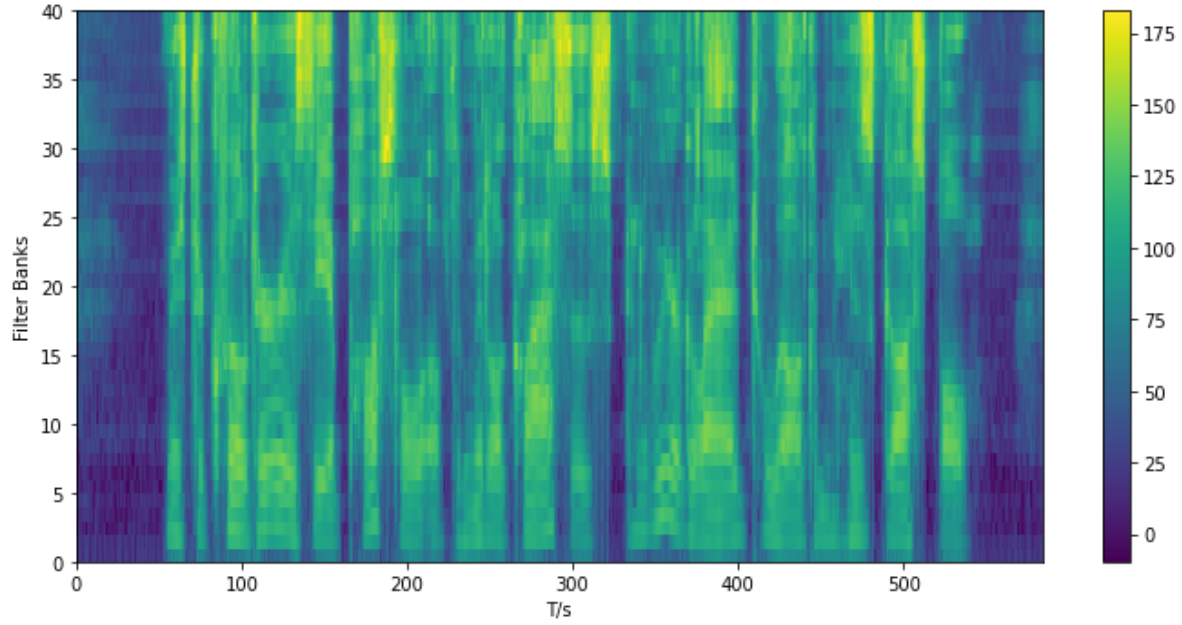
```
low_freq_mel = 0
high_freq_mel = 2595 * np.log10(1 + (sample_rate / 2) / 700)
print(low_freq_mel, high_freq_mel)
```

输入结果为0/2840.023046708319。然后计算Mel滤波器组：

```
nfilt = 40
mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)  # 所有的mel中心点，为了方便后面计算mel滤波器组，左右两边各补一个中心点
hz_points = 700 * (10 ** (mel_points / 2595) - 1)

nfilt = 40  # 通常设置成40个滤波器
mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)  # 所有的mel中心点，为了方便后面计算mel滤波器组，左右两边各补一个中心点
hz_points = 700 * (10 ** (mel_points / 2595) - 1)
fbank = np.zeros((nfilt, int(N / 2 + 1)))  # 各个mel滤波器在能量谱对应点的取值
bin = (hz_points / (sample_rate / 2)) * (N / 2)  # 各个mel滤波器中心点对应FFT的区域编码，找到有值的位置
for i in range(1, nfilt + 1):
    left = int(bin[i-1])
    center = int(bin[i])
    right = int(bin[i+1])
    for j in range(left, center):
        fbank[i-1, j+1] = (j + 1 - bin[i-1]) / (bin[i] - bin[i-1])
    for j in range(center, right):
        fbank[i-1, j+1] = (bin[i+1] - (j + 1)) / (bin[i+1] - bin[i])
```

然后计算并绘制FBANK特征图：

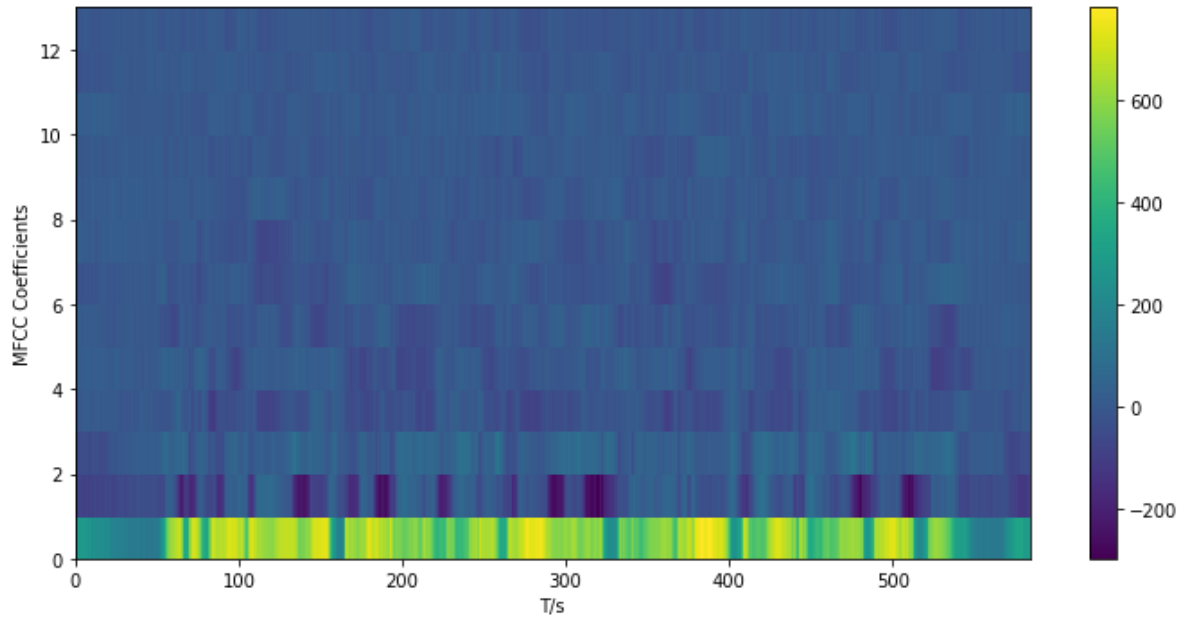
```
filter_banks = np.dot(pow_frames, fbank.T)
filter_banks = np.where(filter_banks == 0, np.finfo(float).eps, filter_banks)
filter_banks = 20 * np.log10(filter_banks)  # dB
print(filter_banks.shape)
```



6. 倒谱系数（Mel-frequency Cepstral Coefficients）（MFCCs）

事实证明，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。

```
num_ceps = 13    # 选取前13维
mfcc = dct(filter_banks, type=2, axis=1, norm='ortho')[:, 0:num_ceps]    # 切片操作 左闭右开
np.save("mfcc_wav1.npy", mfcc)    # 存储最终的特征文件
gdraw(mfcc.T, 'MFCC Coefficients')
```



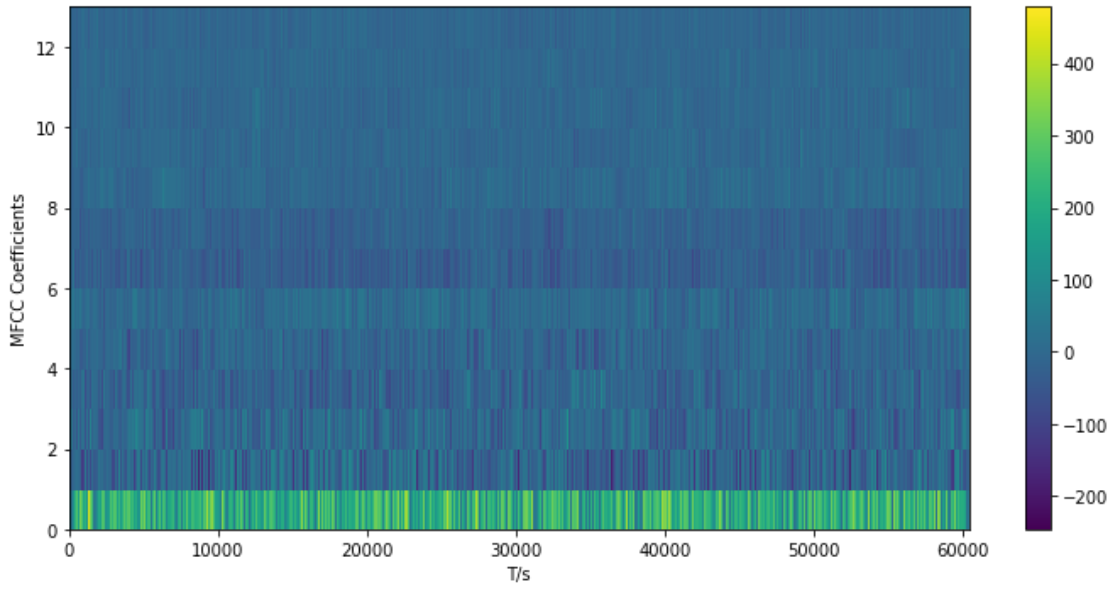
7. 运行耗时分析

选取时长更长的音频文件2.wav，重复上述过程，并利用python的time库分析运行时间

```
import time

t0 = time.time()

# 导入音频
sample_rate, s = wavfile.read('./2.wav')
# 预加重
alpha = 0.97
s = np.append(s[0], s[1:] - alpha * s[:-1])
# 成帧
frame_size, frame_stride = 0.025, 0.01
frame_length, frame_step = int(round(frame_size * sample_rate)),
int(round(frame_stride * sample_rate))
signal_length = len(s)
num_frames = int(np.ceil(np.abs(signal_length - frame_length) / frame_step)) + 1

pad_signal_length = (num_frames - 1) * frame_step + frame_length
z = np.zeros((pad_signal_length - signal_length))
pad_signal = np.append(s, z)

indices = np.arange(0, frame_length).reshape(1, -1) + np.arange(0, num_frames *
frame_step, frame_step).reshape(-1, 1)
frames = pad_signal[indices]
# 加窗
hamming = np.hamming(frame_length)
frames *= hamming
# FFT计算
N = 512
mag_frames = np.absolute(np.fft.rfft(frames, N))
pow_frames = ((1.0 / N) * (mag_frames ** 2))
# FBANK特征计算
low_freq_mel = 0
high_freq_mel = 2595 * np.log10(1 + (sample_rate / 2) / 700)
nfilt = 40  # 通常设置成40个滤波器
mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)  # 所有的mel中心
点，为了方便后面计算mel滤波器组，左右两边各补一个中心点
hz_points = 700 * (10 ** (mel_points / 2595) - 1)
fbank = np.zeros((nfilt, int(N / 2 + 1)))  # 各个mel滤波器在能量谱对应点的取值
bin = (hz_points / (sample_rate / 2)) * (N / 2)  # 各个mel滤波器中心点对应FFT的区域编
码，找到有值的位置
for i in range(1, nfilt + 1):
    left = int(bin[i-1])
    center = int(bin[i])
    right = int(bin[i+1])
    for j in range(left, center):
        fbank[i-1, j+1] = (j + 1 - bin[i-1]) / (bin[i] - bin[i-1])
    for j in range(center, right):
        fbank[i-1, j+1] = (bin[i+1] - (j + 1)) / (bin[i+1] - bin[i])
filter_banks = np.dot(pow_frames, fbank.T)
filter_banks = np.where(filter_banks == 0, np.finfo(float).eps, filter_banks)
filter_banks = 20 * np.log10(filter_banks)  # dB
```

```
# MFCC特征计算
num_ceps = 13    # 选取前13维
mfcc = dct(filter_banks, type=2, axis=1, norm='ortho')[:, 0:num_ceps]    # 切片操作左闭右开
np.save("mfcc_wav2.npy", mfcc)    # 存储最终的特征文件
gdraw(mfcc.T, 'MFCC Coefficients')

t1 = time.time()
print("Processing \'2.wav\' cost {:.3f}s".format(t1-t0))
```

```
Processing '2.wav' cost 8.673s
```



>>> 实验总结与思考

通过本次实验，我基本了解到了MFCC特征提取在语音信号处理领域的基本应用和实现过程，同时熟悉了python语法的使用和基本库的调用方法。

对于FBank特征和MFCC特征的选取方面：FBank特征的提取更多的是希望符合声音信号的本质，拟合人耳接收的特性。而MFCC特征多的那一步则是受限于一些机器学习算法。很早之前MFCC特征和GMMs-HMMs方法结合是ASR的主流。而当一些深度学习方法出来之后，MFCC则不一定是最优选择，因为神经网络对高度相关的信息不敏感，而且DCT变换是线性的，会丢失语音信号中原本的一些非线性成分。

因此，在模型对高相关的信号不敏感时（比如神经网络），可以用FBank特征；在模型对高相关的信号敏感时（比如GMMs-HMMs），需要用MFCC特征。从目前的趋势来看，因为神经网络的逐步发展，FBank特征越来越流行。