第5章

语音识别

本章主要介绍语音识别技术。首先讨论处理语音信号的方式，接着了解音频信号可视化的过程，学习处理语音信号的相关技术，最后介绍如何建立一个语音识别系统。

学完本章后，你将会知道：

- 处理语音信号。
- 可视化音频信号。
- 将音频信号从时域转换为频域。
- 生成音频信号。
- 提取语音特征。
- 构建语音识别系统——识别口语词汇。

17 语音处理 1

5.1 处理语音信号

语音识别是一门多学科交叉的技术。想要与机器流利地进行语音交流，就必须要让机器明白人们说话的含义，这是人们梦寐以求和不断探索的事情。语音识别被中国物联网校企联盟形象比作为“机器的听觉系统”。语音识别技术的目的是让机器识别和理解人们的语言，这个过程会把语音信号转换为相应的文本或命令，如图 5-1 所示。该技术近二十年来取得明显进步，已经从实验室走向了市场。人们预计在未来十年后，该技术会进入各个领域，例如通信、家电、工业、汽车电子、家庭服务、医疗、消费电子产品等。

图 5-1 语音识别

智能手机在日常生活中已经普及，在很多智能手机中都有一个十分方便的功能，那就是通过语音下达命令来让手机执行，例如对手机说“地图”，手机就会打开手机里面的地图 App，这就用到了语音识别技术，运用这项技术的基础就是手机能够感知声音。类似地，如果有一天想对计算机下达命令让计算机打开某个文件夹，还得使用语音识别技术。那么现在问题就来了——我们都知道人类是通过耳朵感知声音的，手机或者计算机是如何感知声音的呢？

其实这个过程就是通过对声波进行一系列处理，最终将其转换为音频文件（MP3 格式等），这样方便计算机存储和处理。首先话筒采集到声波通过传感器转换为电信号（如电压），就好比我们的耳朵里面的听觉感受器将声音传递给听觉神经。

但是不同于我们的大脑，计算机是无法识别连续信号的，所以我们只能将采样到的电信号变换为离散（不连续）信号，无论是在时间还是声波的幅度上都如此，通过时间采样使得声波在时间上离散，同理通过分层的思想让声波的幅度也离散，这样连续的声波就转换为离散的电信号，让计算机可以识别，并且编码保存起来。这一系列的处理包括多个步骤，例如采样、量化和编码等，如图 5-2 所示。

- 采样：在某些特定的时刻对模拟信号进行测量，对模拟信号在时间上进行量化。具体方法是每隔相等或不相等的一小段时间采样一次。
- 量化：分层就是对信号的强度加以划分，量化是对模拟信号在幅度上进行度量。具体方法是将整个强度分成许多小段。
- 编码：将量化后的整数值用二进制数来表示。

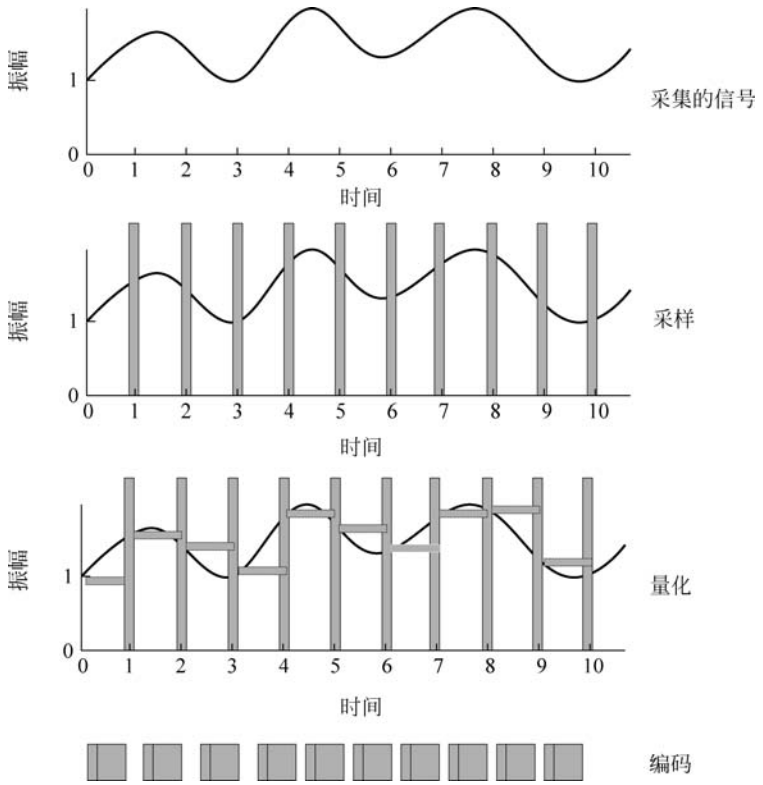


图 5-2　声音的采样、量化和编码

研究人员致力于语言的各个方面和应用，如理解口语单词、识别说话者是谁、识别情绪、识别口音等。语音识别技术在人机交互领域中是一个非常重要的环节。如果想制造能够与人类互动的认知机器人，就需要它们用自然语言与我们交谈。这就是近年来自动语音识别成为众多研究者关注的焦点的原因。

5.2 可视化音频信号

通过对声波的处理,计算机可以感知声音并且可以进行编码。就如同我们的大脑听到了某个声音并记了下来,接下来我们要做的就是识别出这是什么声音,是什么东西发出来的。那计算机到底是如何“理解”声音的呢?

我们都听过很多歌手唱歌,有些歌手的声音低沉沙哑,有些歌手的声音嘹亮高亢。即使他们唱同一首歌我们也能根据那些歌手的声音特色区分出是谁唱的。

那计算机是如何识别这些的呢?因此我们需要频谱——计算机分析音频的依据。频谱的纵坐标代表幅度,横坐标代表频率——相应频率的声音对应的振幅。频谱图反映了不同频率的声音占的能量多少,在频谱图上反映的就是频谱幅度的相对大小。例如一段乐曲中的高音强低音弱,那么在一定范围内频率高的区域频谱的振幅就大,反之在频率低的区域对应的频谱幅度大,如图 5-3 和图 5-4 所示。

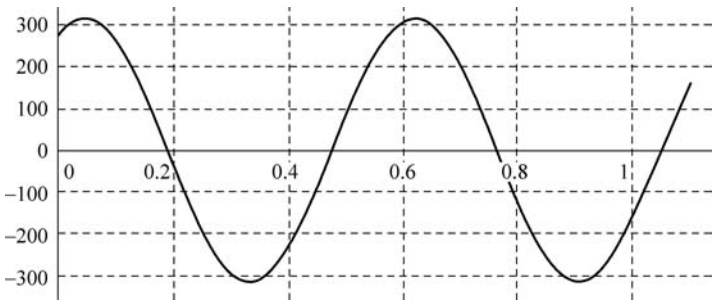


图 5-3 波形图

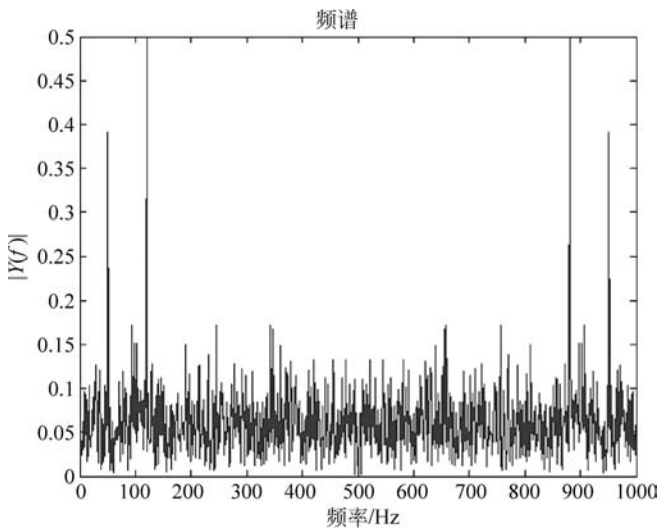


图 5-4 频谱图

我们在物理课上学到过声音三要素——音调、响度和音色。基于这些要素,我们就可以描述声音的特性。

- 音调：代表声音调子的高低。频率越高调子越高，反之，频率越低的调子就会越低。音调用频谱描述。
- 响度：常说的声音的大小，可以由波形的幅度来表示。
- 音色：声音的一种更加复杂的特征，就算是相同的音调和响度，不同的乐器演奏或者不同的人来演唱都会有不同的效果。原因就是不同的乐器和声带在振动发声的过程中，除了发出音调对应的频率 f 之外，还伴着一些高频的成分（频率为 $2f$，$3f$，$\cdots nf$），称为泛音。这些高频的成分对应的幅度各不相同，造成了特别的听觉感受。这也就解释了为什么有些人即使唱同一首歌会有不同的效果。音色图如图 5-5 所示。

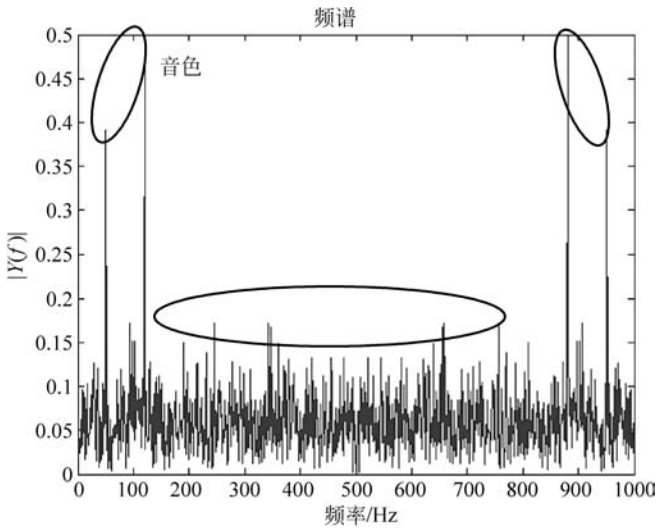


图 5-5　音色图

图 5-5 所示的音色图中第一个最高峰所处的频率就是音调，而在这个频率的整数倍的位置都有不同大小的峰值，它们之间的比例反映了声音音色的不同。通过这些特性，就能大概分出这是什么发出的声音了。

本章将可视化音频信号。我们会从一个文件中读取音频信号并使用它来生成频谱。这将帮助我们了解音频信号的结构。当使用麦克风录制音频文件时，音频信号会被采样并存储为数字化的形式。真实的音频信号情况是一个连续的波值，意味着不能按照实际音频原样存储它们，因此需要对某一频率的信号处理，首先采样，再转换为离散的数值。

最常见的语音信号的采样频率为 44 100Hz，这意味着每秒的语音信号会被分解成 44 100 个小段，每个时间戳的离散数值都会被存储在一个输出的文件当中。每隔 1/44 100s 保存一次音频信号的值。在这种情况下，音频信号的采样频率是 44 100Hz。通过选择一个高采样频率，当人们听到它的时候，它会看起来像连续的音频信号。

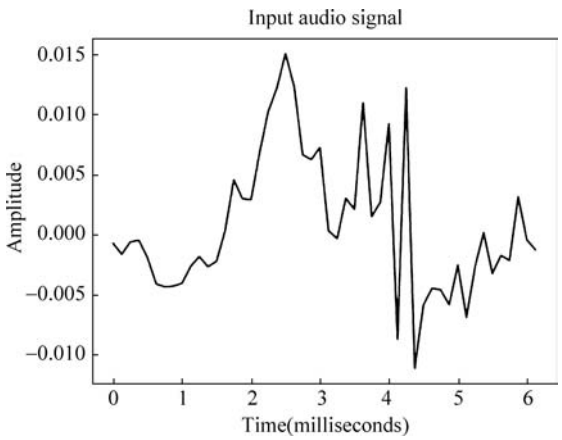
接下来通过一段程序将音频信号可视化。

程序 5.1　可视化音频信号

```
1:    importnumpy as np
2:    import matplotlib.pyplot as plt
3:    from scipy.io import wavfile
4:
```

```
5:   sampling_freq, signal = wavfile.read('random_sound.wav')
6:
7:   print('\nSignal shape:', signal.shape)
8:   print('Datatype:', signal.dtype)
9:   print('Signal duration:', round(signal.shape[0] / float(sampling_freq), 2),
10:  'seconds')
11:
12:  signal = signal / np.power(2, 15)
13:  signal = signal[:50]
14:
15:  time_axis = 1000 * np.arange(0, len(signal), 1) / float(sampling_freq)
16:
17:  plt.plot(time_axis, signal, color = 'black')
18:  plt.xlabel('Time (milliseconds)')
19:  plt.ylabel('Amplitude')
20:  plt.title('Input audio signal')
21:  plt.show()
```

输出：



分析：首先导入相关库，其中 SciPy 是一个常用软件包，主要用于科学、数学和工程领域，它可以处理很多复杂问题，包括积分、插值、优化、图像处理、常微分方程数值解的求解、信号处理等。接着用这个库中的 wavefile.read()方法读取输入的音频文件，它返回两个值——采样频率和音频信号。第 7～10 行是打印出信号的形状、数据类型和音频信号的持续时间。然后把信号进行标准化处理并从 NumPy 数组中提取前 50 个值用于绘图。最后以秒为单位绘制时间轴并将之前做标准化处理的音频信号绘制出来。

上面的截图显示了输入音频信号的前 50 个示例。

18 语音处理 2

5.3 将音频信号从时域转换为频域

为分析音频信号，需要知道音频信号的频率，并从中提取出有意义的信息。音频信号是由多种元素混合而成的，包括频率、相位和振幅的正弦波。如果仔细分析频率的分量，就能识别

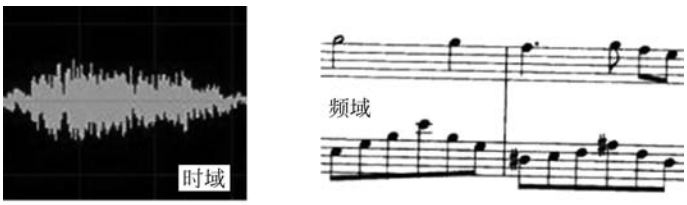
出许多特征。任何给定的音频信号特征都是其在频谱中的分布。

时域和频域是信号的基本性质。下面讲解什么是时域，什么是频域。

时域(time domain)是唯一实际存在的域，它是真实世界。简单来说，我们所接触的世界是随着时间在不断进行变化的，世界是在运动的过程，如图 5-6(a)所示。

频域(frequency domain)不是真实的，而是一个数学构造。时域是唯一实际存在的域，而频域是一个数学范畴，也是被称为“上帝视角”的遵循特定规则的数学范畴。频域中唯一存在的波形是正弦波，它是对频域的描述，也是频域中最重要的规则，并且频域中的任何波形都是可以用正弦波来合成的，如图 5-6(b)所示。

通过图片来直观解释：在时域里面，一段音乐是什么？音乐是一种振动，并且是随着时间变化而变化的，就像是钢琴上的琴弦，总是一会儿上一会儿下地摆动着。相比较，在频域里面，一段音乐是什么？是一个个音符，是乐谱。乐谱中的音符个数是固定有限的，但仅仅有限的音符却可以组合成无限多的乐曲。



(a) 时域　　(b) 频域

图 5-6　时域和频域

时域的基本变量是时间，它描述的是物理信号或数学函数与时间之间的关系。一个信号的时域波形表达出的信号是随着时间的变化而改变的。

频域的基本变量是频率，它描述的是一种坐标系，即信号在频率方面特性时用到的坐标系。坐标系上，横轴是频率，即频率是自变量；纵轴是该频率信号的幅度，就是频谱图，它描述了信号的频率结构及频率与该频率信号幅度之间的关系。

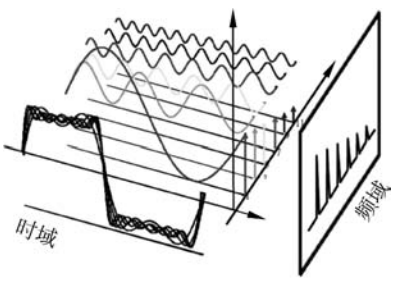


图 5-7　时域和频域坐标图

例如，眼前有一辆汽车，可以这样描述它。方面 1：颜色，长度，高度；方面 2：排量，品牌，价格。而对于一个信号来说，它也有很多方面的特性，例如时域特性，信号强度会随着时间的变化规律；频域特性；信号是被哪些单一频率的信号合成的。

为了把时域信号转换为频域信号，需要使用快速傅里叶变换(FFT)这样的数学工具，如图 5-7 所示。快速傅里叶变换实质是频域函数和时域函数的转换。如果对快速傅里叶变换不太了解的话，可以参考数学方面相关书籍，这里不再讲解。

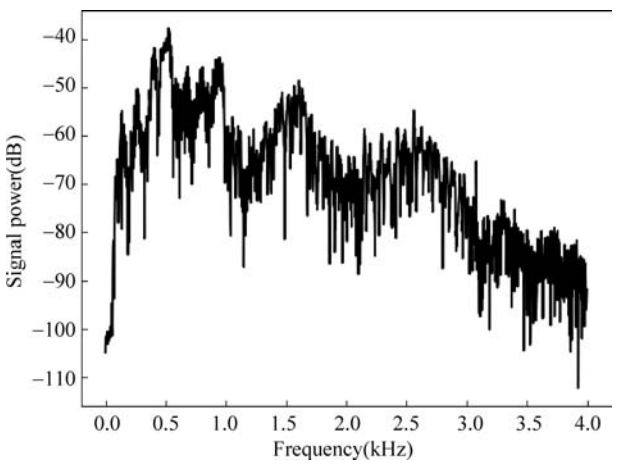
接下来看如何对音频信号进行转换，即从时域转换为频域。

程序 5.2　将音频信号从时域转换为频域

```
1:    import numpy as np
2:    import matplotlib.pyplot as plt
```

```
from scipy.io import wavfile

sampling_freq, signal = wavfile.read('spoken_word.wav')
signal = signal / np.power(2, 15)

len_signal = len(signal)
len_half = np.ceil((len_signal + 1) / 2.0).astype(np.int)

freq_signal = np.fft.fft(signal)

freq_signal = abs(freq_signal[0:len_half]) / len_signal
freq_signal **= 2

len_fts = len(freq_signal)
if len_signal % 2:
    freq_signal[1:len_fts] *= 2
else:
    freq_signal[1:len_fts-1] *= 2

signal_power = 10 * np.log10(freq_signal)

x_axis = np.arange(0, len_half, 1) * (sampling_freq / len_signal) / 1000.0

plt.figure()
plt.plot(x_axis, signal_power, color='black')
plt.xlabel('Frequency (kHz)')
plt.ylabel('Signal power (dB)')
plt.show()
```

输出：



分析：这段程序和程序 5.1 类似，先读取输入音频文件，并对其做标准化处理。其次提取出信号的长度和半长，进行快速傅里叶变换，将其变为频域信号。再对频域信号做归一

化处理，取平方。这里需要考虑两种情况，即频域信号为奇数和偶数。然后计算信号的功率值，它是根据得到的频域信号取对数计算所得。最后构建 x 轴，在本例中用千赫(kHz)表示频率。

上面的截图显示了信号在频谱上的强度。

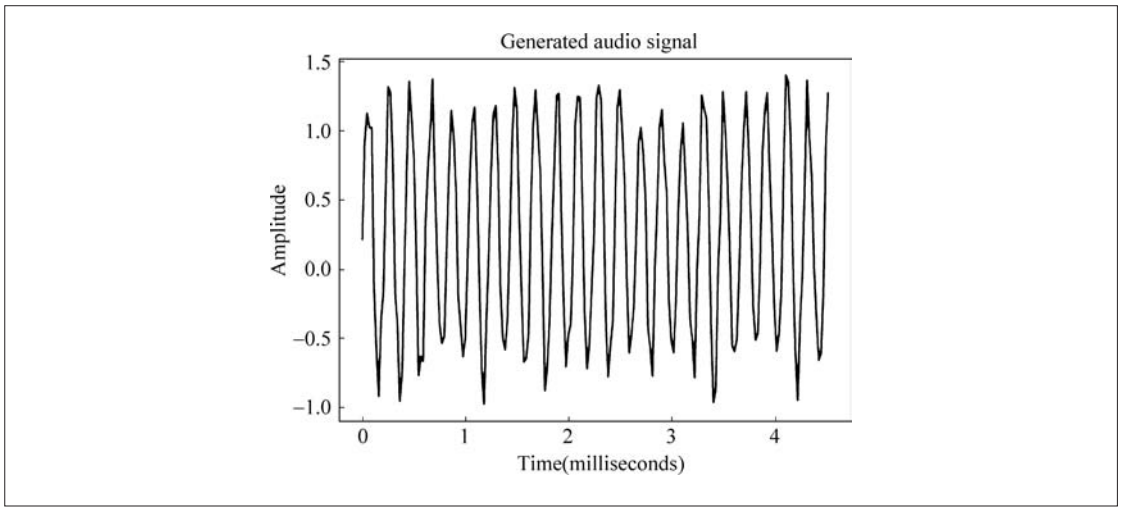
5.4 生成音频信号

知道了音频信号是如何工作的，再来看如何生成这样一个音频信号。可以使用 NumPy 包来生成不同的音频信号。由于音频信号是正弦波的混合，可以使用它来生成带有一些预定义参数的音频信号。

程序 5.3 生成音频信号

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import write

output_file = 'generated_audio.wav'

duration = 4 # in seconds
sampling_freq = 44100 # in Hz
tone_freq = 784
min_val = -4 * np.pi
max_val = 4 * np.pi

t = np.linspace(min_val, max_val, duration * sampling_freq)
signal = np.sin(2 * np.pi * tone_freq * t)

noise = 0.5 * np.random.rand(duration * sampling_freq)
signal += noise

scaling_factor = np.power(2, 15) - 1
signal_normalized = signal / np.max(np.abs(signal))
signal_scaled = np.int16(signal_normalized * scaling_factor)

write(output_file, sampling_freq, signal_scaled)

signal = signal[:200]
time_axis = 1000 * np.arange(0, len(signal), 1) / float(sampling_freq)
plt.plot(time_axis, signal, color='black')
plt.xlabel('Time (milliseconds)')
plt.ylabel('Amplitude')
plt.title('Generated audio signal')
plt.show()
```

输出：



分析：首先定义一个输出音频文件名，这里为 generated_audio. wav。接着指定音频参数，时长定义为 4s、采样频率为 44 100Hz、音频频率为 784Hz、最小值为 -4π、最大值为 4π。使用这些定义好的参数生成音频信号，使用 NumPy 中的正弦函数生成音频信号。为了使音频信号看起来更像一些，使用随机数为音频信号添加一些噪声。接着需要对信号进行归一化和缩放处理。最后将生成的音频信号保存到输出的文件当中。

在本程序中，提取生成音频信号的前 200 个值用于绘图，如上输出所示。假如使用媒体播放器播放 generated_audio. wav 文件，会发现它是一个由 784Hz 的音频频率和噪声混合而成的音频信号。

5.5 提取语音特征

之前学习了如何把时域信号转换为频域信号，在语音识别系统中，频域特征被广泛应用，但是要知道真实世界的频域特征更加复杂。一旦把一个信号转换为频域，需要确保它可以以特征向量的形式供我们使用。这就涉及 Mel Frequency Cepstral Coefficients（MFCC）了。MFCC 是一种工具，用于从给定音频信号中提取频域特征。

MFCC：Mel 频率倒谱系数的缩写。基于人耳听觉特性提出 Mel 频率，它与赫兹（Hz）频率为非线性对应的关系。MFCC 是计算 MFCC 与赫兹（Hz）频率之间的非线性关系，得到的赫兹（Hz）频谱特征，使 MFCC 频率提高，精度下降。语音识别领域中，MFCC 已经被广泛应用，通常在应用中，我们只使用低频 MFCC，丢弃中高频 MFCC。

在 Mel 标度频率域中，提取出倒谱参数 MFCC。其中 Mel 频率描述的是人耳频率的非线性特性，它与频率的关系可用下式近似表示：

$$\mathrm{Mel}(f)=2595\times\lg(1+f/700)$$

式中 f 为频率，单位为 Hz。图 5-8 展示了 Mel 频率与线性频率的关系。

提取语音特征参数 MFCC 主要涉及以下流程，如图 5-9 所示。

（1）预加重。实质是将语音通过一个高通滤波器，目的是提升高频的部分，可以让信号频谱保持在低频到高频的整个频带中，变得平坦，从而能够用同样的信噪比求频谱。

（2）分帧。为方便分析语音，可以将语音采取分段。首先把 N（通常 N 的值为 256 或

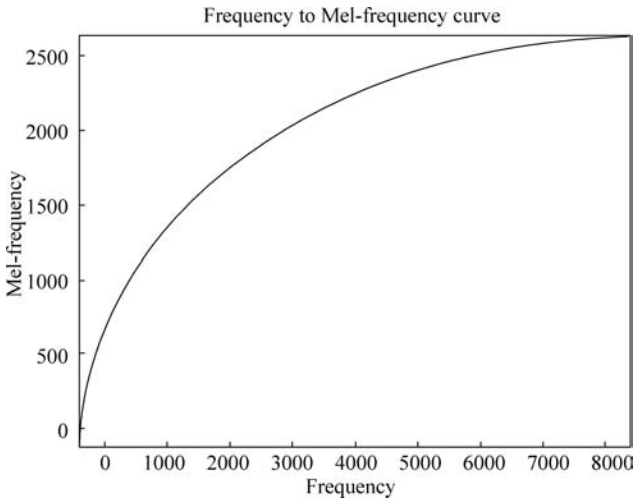


图 5-8　Mel 频率与线性频率关系图

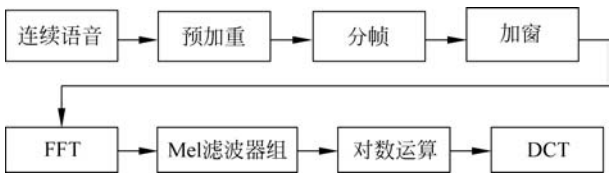


图 5-9　MFCC 参数提取基本流程

512，其中涵盖的时间为 20～30ms）个采样点集合成一个观测单位，称为“帧”。

（3）加窗。在长范围内，语音是不断发生变化的，如果没有固定的特性，就无法进行处理。因此，把每一帧代入窗函数（通常窗外的值设定为 0），目的是消除信号不连续性，因为各个帧两端可能会造成的不连续性。常用的窗函数有汉明窗（由于窗函数的频域特性，因此通常采用汉明窗）、方窗和汉宁窗等。

（4）FFT。由于信号在时域不断变化，因此很难看出信号的特性，所以把它转换为频域上的能量分布方便观察。不同语音的特性有着不同的能量分布。所以在以上汉明窗后，每帧还必须得到在频谱上的能量分布，可使用 FFT 变换得到。再对分帧加窗后的各帧信号用同样的方式以得到各帧的频谱。

（5）Mel 滤波器组。将频谱通过一组 Mel 尺度的三角形滤波器组。这么做的目的是平滑化频谱，消除谐波，使原先语音的共振峰更加突显。

（6）对数运算。

（7）DCT 离散余弦变换。经 DCT 得到 MFCC 系数。

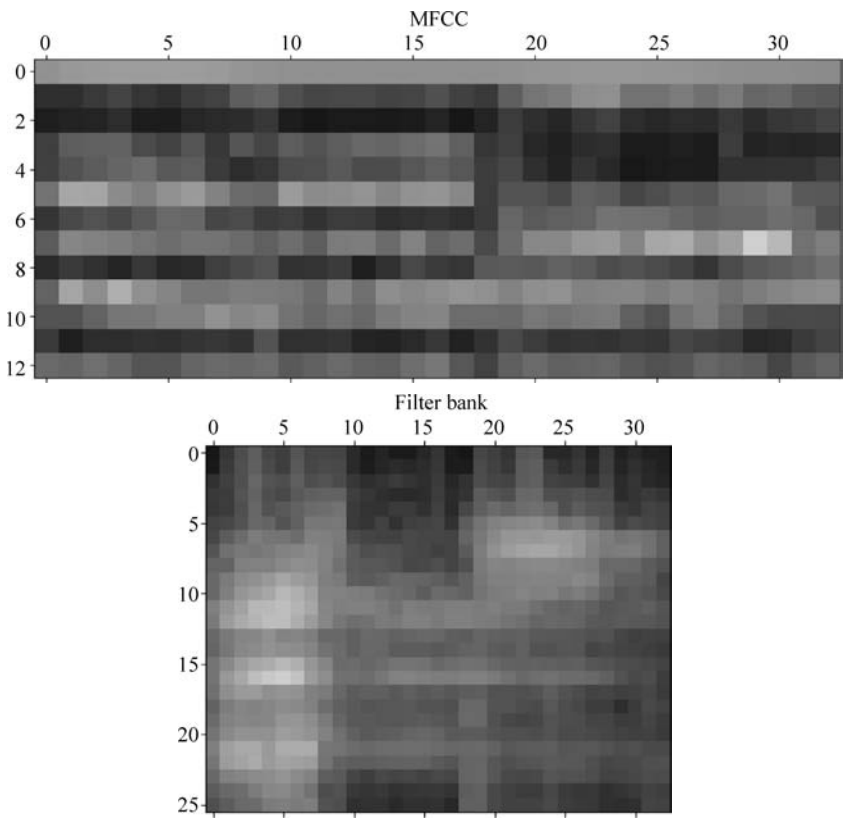
在本节中，将使用一个名为 python_speech_features 的 Python 库来提取 MFCC 特征，该库为语音识别技术提供常见的语音功能，包括 MFCC 和滤波器组。可以使用 pip 命令安装它。接下来介绍如何提取语音特征。

程序 5.4　提取语音特征

```
1:    import numpy as np
2:    import matplotlib.pyplot as plt
```

```
from scipy.io import wavfile
from python_speech_features import mfcc, logfbank

sampling_freq, signal = wavfile.read('random_sound.wav')
signal = signal[:10000]

features_mfcc = mfcc(signal, sampling_freq)

print('\nMFCC:\nNumber of windows = ', features_mfcc.shape[0])
print('Length of each feature = ', features_mfcc.shape[1])

features_mfcc = features_mfcc.T
plt.matshow(features_mfcc)
plt.title('MFCC')

features_fb = logfbank(signal, sampling_freq)

print('\nFilter bank:\nNumber of windows = ', features_fb.shape[0])
print('Length of each feature = ', features_fb.shape[1])

features_fb = features_fb.T
plt.matshow(features_fb)
plt.title('Filter bank')
plt.show()
```

输出：



分析：首先，需要从 python_speech_features 库中导入 mfcc()和 logfbank()这两个函数。读取输入音频文件，提取音频信号的前 10 000 个值用于分析。接着使用 python_speech_features 库中的 mfcc()方法提取出 MFCC 特征并打印其参数。再使用 logfbank()方法提取出滤波器组特征，打印其参数并在终端输出。

上面第一幅图显示了 MFCC 特征，第二幅图显示了滤波器组特征。

提示：

> python_speech_features 库为语音识别技术提供常见的语音功能，包括 MFCC 和滤波器组。如果想了解更多，请参考 http://python-speech-features.readthedocs.org/en/latest。

5.6 构建语音识别系统——识别口语词汇

19 语音处理 3

前面已经学习了分析语音信号的相关技术，现在介绍如何识别口语词汇。语音识别系统以音频信号为输入，识别正在说话的单词。在本节中，将使用隐马尔可夫模型(HMM)来完成这项任务。

首先了解什么是马尔可夫链(Markov chain)。马尔可夫链是因安德烈·马尔可夫(A. A. Markov，1856—1922)而得名。马尔可夫链指数学中具有马尔可夫性质的离散事件随机过程。一个 n 阶的模型是指每个状态的转移会依赖于之前的 n 个状态的过程，n 是影响转移状态的数目。最简单的马尔可夫过程是一阶过程，每一个状态的转移只依赖于其之前的那一个状态。用数学表达式表示就是：

$$P(X_{n+1}=x \mid X_0=x_0, X_1=x_1, \cdots, X_n=x_n)=P(X_{n+1}=x \mid X_n=x_n)$$

系统根据概率分布，在马尔可夫链的每一步都可以从一个状态变到另一个状态，也可以保持在当前状态。其中转移指的是状态的转变过程，转移概率是与不同的状态改变相关的概率。

HMM 与时序相关，HMM 在马尔可夫链的基础上增加了观测事件(observed events)，即把马尔可夫链原本可见的状态序列隐藏起来，通过一个可观测的显层来推断隐层的状态信息，如图 5-10 所示。其中，隐层映射到显层通过发射概率(emission probability)或观测概率(observation probability)来计算，隐层状态之间的转移通过转移概率(transition probability)获得。

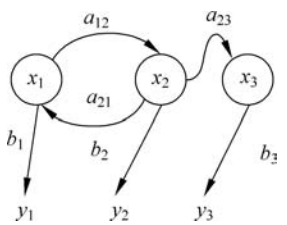


图 5-10 HMM 状态图

图 5-10 中，x 表示隐含状态，y 表示观察的输出，a 表示转换概率，b 表示输出概率。

举一个经典的例子：一个东京的朋友会每天根据天气({下雨，天晴})来决定当天进行哪项活动({公园散步，购物，清理房间})。每天能看到她发的推特，描述说："啊，我前天在公园散步，昨天去超市购物，今天清理房间了！"，那么可以推断东京这三天的天气。这个例子里，显状态是活动，隐状态是天气。

HMM 非常擅长分析时序数据。音频信号是一种时间序列信号(单位是时间)，是关于时序的一种数据表现形式。假设输出是由经过一系列隐藏状态的系统生成的。我们的目标是找出这些隐藏状态是什么，这样就能识别语音信号中的单词。将使用一个名为 hmmlearn 的软

件包来构建语音识别系统。

为训练语音识别系统,需要为每个单词建立一个音频文件集。在代码包中提供了一个名为 data 的文件夹方便使用,其中包含所有音频文件集。这个数据集中包含七个不同的单词。每个单词都有一个相关联的文件夹,其中每个文件夹有 15 个音频文件。在每个文件夹中都使用前 14 个音频文件进行训练,最后一个用于测试。注意,这实际上是一个非常小的数据集,可以使用更大的数据集来构建自己的语音识别系统。

接下来将继续为每一个单词构建一个 HMM。把所有这些模型都存储起来作为参考。当想要识别未知音频文件中的单词时,遍历所有这些模型并选择匹配度最高的那个单词。

下面介绍如何构建一个可以识别口语词汇的语音识别系统。

程序 5.5 构建语音识别系统——识别口语词汇

```
import os
import argparse
import warnings
import numpy as np
from scipy.io import wavfile
from hmmlearn import hmm
from python_speech_features import mfcc

def build_arg_parser():
    parser = argparse.ArgumentParser(description = 'Trains the HMM-based
        speech recognition system')
    parser.add_argument("--input-folder", dest = "input_folder",
                required = True, help = "Input folder containing the audio
        files  for training")
    return parser

class ModelHMM(object):
    def __init__(self, num_components = 4, num_iter = 1000):
        self.n_components = num_components
        self.n_iter = num_iter
        self.cov_type = 'diag'
        self.model_name = 'GaussianHMM'
        self.models = []
        self.model = hmm.GaussianHMM(n_components = self.n_components,
                covariance_type = self.cov_type, n_iter = self.n_iter)

    def train(self, training_data):
        np.seterr(all = 'ignore')
        cur_model = self.model.fit(training_data)
        self.models.append(cur_model)

    def compute_score(self, input_data):
        return self.model.score(input_data)

def build_models(input_folder):
    speech_models = []
```

```
37:
38:     for dirname in os.listdir(input_folder):
39:         subfolder = os.path.join(input_folder, dirname)
40:         if not os.path.isdir(subfolder):
41:             continue
42:
43:         label = subfolder[subfolder.rfind('/') + 1:]
44:         X = np.array([])
45:
46:         training_files = [x for x in os.listdir(subfolder) if
47:                           x.endswith('.wav')][:-1]
48:
49:         for filename in training_files:
50:             filepath = os.path.join(subfolder, filename)
51:             sampling_freq, signal = wavfile.read(filepath)
52:             with warnings.catch_warnings():
53:                 warnings.simplefilter('ignore')
54:                 features_mfcc = mfcc(signal, sampling_freq)
55:
56:             if len(X) == 0:
57:                 X = features_mfcc
58:             else:
59:                 X = np.append(X, features_mfcc, axis=0)
60:
61:         model = ModelHMM()
62:         model.train(X)
63:         speech_models.append((model, label))
64:         model = None
65:
66:     return speech_models
67:
68: def run_tests(test_files):
69:     for test_file in test_files:
70:         sampling_freq, signal = wavfile.read(test_file)
71:         with warnings.catch_warnings():
72:             warnings.simplefilter('ignore')
73:             features_mfcc = mfcc(signal, sampling_freq)
74:
75:         max_score = -float('inf')
76:         output_label = None
77:
78:         for item in speech_models:
79:             model, label = item
80:             score = model.compute_score(features_mfcc)
81:             if score > max_score:
82:                 max_score = score
83:                 predicted_label = label
84:
85:         start_index = test_file.find('/') + 1
86:         end_index = test_file.rfind('/') - 3
```

```
            original_label = test_file[start_index:end_index]
            print('\nOriginal: ', original_label)
            print('Predicted:', predicted_label)

    if __name__ == '__main__':
        args = build_arg_parser().parse_args()
        input_folder = args.input_folder

        speech_models = build_models(input_folder)

        test_files = []
        for root, dirs, files in os.walk(input_folder):
            for filename in (x for x in files if '15' in x):
                filepath = os.path.join(root, filename)
                test_files.append(filepath)

    run_tests(test_files)
```

输出：

```
Original:  data\apple\apple15
Predicted: data\apple

Original:  data\banana\banana15
Predicted: data\banana

Original:  data\kiwi\kiwi15
Predicted: data\kiwi

Original:  data\lime\lime15
Predicted: data\lime

Original:  data\orange\orange15
Predicted: data\orange

Original:  data\peach\peach15
Predicted: data\peach

Original:  data\pineapple\pineapple15
Predicted: data\pineapple
```

分析：首先导入相关依赖库，hmmlearn 实现了三种 HMM 类，按照观测状态是连续状态还是离散状态，可以分为两类。GaussianHMM 和 GMMHMM 是连续观测状态的模型，而 MultinomialHMM 是离散观测状态的模型。这里使用 GaussianHMM。

定义一个解析函数 build_arg_parser()，该函数用来指定包含训练语音识别系统所需音频文件的输入文件夹。接着定义一个 HMM 类。在这个类中指定了训练 HMM 所需的一些参数，例如协方差、HMM 类型和用来存储每个单词的模型。再定义一个 train()函数，该函数用来训练 HMM。然后定义一个计算输入数据得分的方法 compute_score()，在该方法中运行 HMM 对输入数据进行推理。

接着定义一个函数 build_models()，为训练数据集中的每个单词建立模型。在这个函数中，先初始化一个模型变量 speech_models 来存储单词模型，然后解析输入目录，获取子文件并从中提取出训练数据；然后初始化一个变量 x 以存储训练数据，再创建一个用于训练的文

件列表 training_files，将为每个文件夹保留一个文件用于测试。之后遍历训练文件并构建模型。提取当前文件路径，从当前文件中读取音频信号，提取出 MFCC 特征，并将其追加到变量 x 中。再创建一个 HMM，使用之前得到的训练数据训练该模型，然后将模型保存为当前单词。把模型置为空，继续下一次训练。最后返回单词模型。

训练函数完成之后，再定义一个测试函数 run_tests()来运行测试数据集上的测试。首先从输入文件中得到测试数据，提取出 MFCC 特征。然后定义变量来存储最大得分和输出标签。在所有 HMM 中运行当前 MFCC 特征，并选择得分最高的模型。最后打印出预测的输出。

在程序的 main()函数中，根据输入参数获取输入文件夹。然后为输入文件夹中的每个单词建立一个 HMM。在每个文件夹中留了一个文件用于测试。最后运行测试函数 run_tests()来查看模型的准确性。

如果要运行该代码，在终端中切换到该文件所在的目录下，执行以下命令：

```
python speech_recognizer.py --input-folder data
```

执行完之后，输出与书中显示一致。Original 显示的是测试使用的文件，正好是每个单词文件夹中对应的最后一个文件。Predicted 是预测输出。正如所看到的，语音识别系统正确地识别了所有的单词。

5.7 小结

本章学习了语音识别相关技术。首先讨论了如何处理语音信号及相关概念。接着介绍语音信号可视化，并通过快速傅里叶变换将其从时域变换为频域，还使用一些预定义的参数来生成语音信号。最后讨论了 MFCC 特征提取和 HMM，并用这些知识构建了一个可以识别口语单词的语音识别系统。

习题

1. 列举几个语音识别技术的应用领域。
2. 简单概述语音识别技术的原理。
3. 实现将音频信号从时域转换为频域。
4. 项目开发：基于 DeepSpeech2 开发英语语音识别系统。

（1）数据使用：LibriSpeech ASR corpus（http://www.openslr.org/12/）。包含 1000 小时 16kHz 的英文语音。

（2）语料库使用(包 pysoundfile)。

（3）特征工程(Mel-frequency cepstral coefficient (MFFCs))。提示：from python_speech_features import mfcc

（4）DS2 训练体系。该体系结构由多层递归连接、卷积滤波器和非线性以及应用于 RNN 的特定规范化实例组成。

（5）训练模型。

（6）测试和评估模型。

第 5 章
第 4 题
解析