CHAPTER 10

Speech to Text and Vice Versa

In this chapter, you will learn about the importance of speech-to-text and text-to-speech conversion. You will also learn about the functions and components needed to do this type of conversion.

Specifically, I will cover the following:

- Why you would want to convert speech to text
- Speech as data
- Speech features that map speech to a matrix
- Spectrograms, which map speech to an image
- Building a classifier for speech recognition through mel-frequency cepstral coefficient (MFCC) features
- Building a classifier for speech recognition through spectrograms
- Open source approaches for speech recognition
- Popular cognitive service providers
- The future of speech of text

N. K. Manaswi, *Deep Learning with Applications Using Python*,
https://doi.org/10.1007/978-1-4842-3516-4_10

Speech-to-Text Conversion

Speech-to-text conversion, in layman's terms, means that an app recognizes the words spoken by a person and converts the voice to written text. There are lots of reasons you would want to use Speech-to-Text conversion.

- Blind or physically challenged people can control different devices using only voice.
- You can keep records of meetings and other events by converting the spoken conversation to text transcripts.
- You can convert the audio in video and audio files to get subtitles of the words being spoken.
- You can translate words into another language by speaking into a device in one language and converting the text to speech in another language.

Speech as Data

The first step of making any automated speech recognition system is to get the *features*. In other words, you identify the components of the audio wave that are useful for recognizing the linguistic content and delete all the other useless features that are just background noises.

Each person's speech is filtered by the shape of their vocal tract and also by the tongue and teeth. What sound is coming out depends on this shape. To identify the phoneme being produced accurately, you need to determine this shape accurately. You could say that the shape of the vocal tract manifests itself to form an envelope of the short-time power spectrum. It's the job of MFCCs to represent this envelope accurately.

Speech can also be represented as data by converting it to a spectrogram (Figure 10-1).

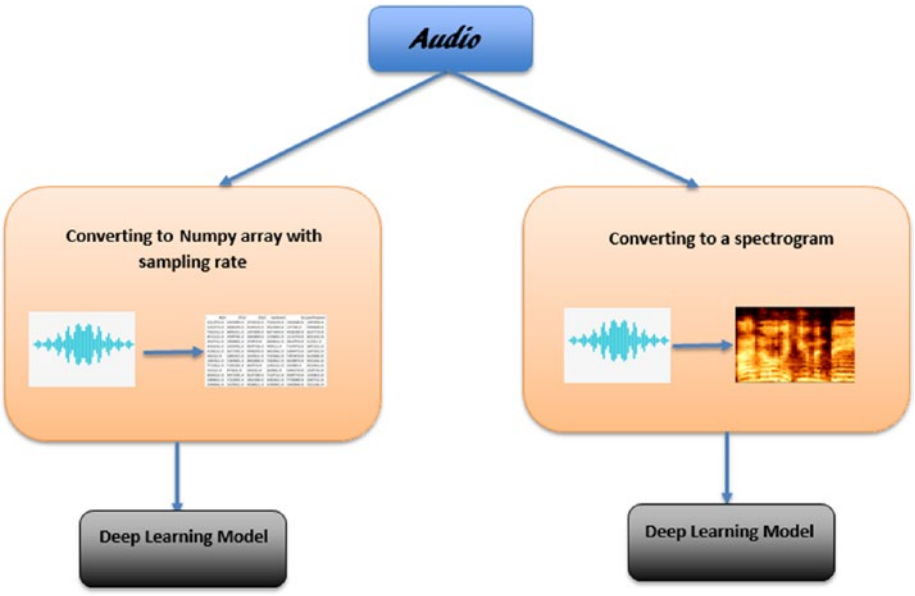

Figure 10-1. *Speech as data*

Speech Features: Mapping Speech to a Matrix

MFCCs are widely used in automated speech and speaker recognition. The mel scale relates the perceived frequency, or *pitch*, of a pure tone to its actual measured frequency.

You can convert an audio in frequency scale to the mel scale using the following formula:

$$M(f) = 1125 \ln(1 + f/700)$$

To convert it back to frequency, use the following formula:

$$M^{-1}(m) = 700(\exp(m/1125) - 1)$$

Here is the function to extract MFCC features in Python:

```
def mfcc(signal,samplerate=16000,winlen=0.025,winstep=0.01,
        numcep=13, nfilt=26,nfft=512,lowfreq=0,highfreq=None,
        preemph=0.97, ceplifter=22,appendEnergy=True)
```

These are the parameters used:

- `signal`: This is the signal for which you need to calculate the MFCC features. It should be an array of N*1 (read a WAV file).
- `samplerate`: This is the signal's sample rate at which you are working.
- `winlen`: This is the analysis window length in seconds. By default it is 0.025 second.
- `winstep`: This is the successive window step. By default it is 0.01 second.
- `numcep`: This is the number of ceptrum that the function should return. By default it is 13.
- `nfilt`: This is the number of filters in the filter bank. By default it is 26.
- `nfft`: This is the size of the fast Fourier transform (FFT). By default it is 512.
- `lowfreq`: This is the lowest band edge, in hertz. By default it is 0.
- `highfreq`: This is the highest band edge, in hertz. By default it is the sample rate divided by 2.

- `preemph`: This applies a preemphasis filter with `preemph` as the coefficient. 0 means no filter. By default it is 0.97.
- `ceplifter`: This applies a lifter to the final cepstral coefficients. 0 means no lifter. By default it is 22.
- `appendEnergy`: The zeroth cepstral coefficient is replaced with the log of the total frame energy, if it is set to true.

This function returns a Numpy array containing features. Each row contains one feature vector.

Spectrograms: Mapping Speech to an Image

A *spectrogram* is photographic or electronic representation of a spectrum. The idea is to convert an audio file into images and pass the images into deep learning models such as a CNN and LSTM for analysis and classification.

The spectrogram is computed as a sequence of FFTs of windowed data segments. A common format is a graph with two geometric dimensions; one axis represents time, and another axis represents frequency. A third dimension uses the color or size of point to indicate the amplitude of a particular frequency at a particular time. Spectrograms are usually created in one of two ways. They can be approximated as a filter bank that results from a series of band-pass filters. Or, in Python, there is a direct function that maps audio to a spectrogram.

Building a Classifier for Speech Recognition Through MFCC Features

To build a classifier for speech recognition, you need to have the python_speech_features Python package installed.

You can use the command `pip install python_speech_features` to install this package.

The `mfcc` function creates a feature matrix for an audio file. To build a classifier that recognizes the voices of different people, you need to collect speech data of them in WAV format. Then you convert all the audio files into a matrix using the `mfcc` function. The code to extract the features from the WAV file is shown here:

```
from python_speech_features import mfcc
from python_speech_features import delta
from python_speech_features import logfbank
import scipy.io.wavfile as wav

(samplerate,signal) = wav.read("audio.wav")
mfccfeatures = mfcc(signal,samplerate)
dmfccfeature = delta(mfccfeatures, 2)
fbankfeature = logfbank(signal,samplerate)

print(fbankfeature)
```

If you run the previous code, you will get output in the following form:

```
[[ 7.66608682  7.04137131  7.30715423 ...,  9.43362359  9.11932984
   9.93454603]
 [ 4.9474559   4.97057377  6.90352236 ...,  8.6771281   8.86454547
   9.7975147 ]
 [ 7.4795622   6.63821063  5.98854983 ...,  8.78622734  8.805521
   9.83712966]
 ...,
 [ 7.8886269   6.57456605  6.47895433 ...,  8.62870034  8.79965464
   9.67997298]
```

```
[ 5.73028657  4.87985847  6.64977329 ...,  8.64089442  8.62887745
  9.90470194]
[ 8.8449656   6.67098127  7.09752316 ...,  8.84914694  8.97807983
  9.45123015]]
```

Here, each row represents one feature vector.

Collect as many voice recordings of a person as you can and append the feature matrix of each audio file in this matrix.

This will act as your training data set.

Repeat the same steps with all the other classes.

Once the data set is prepared, you can fit this data into any deep learning model (that is used for classification) to classify the voices of different people.

Note To see the full code of a classifier using MFCC features, you can visit `www.navinmanaswi.com/SpeechRecognizer`.

Building a Classifier for Speech Recognition Through a Spectrogram

Using the spectrogram approach converts all the audio files to images (Figure 10-2), so all you have to do is convert all the sound files in the training data into images and feed those images to a deep learning model just like you do in a CNN.

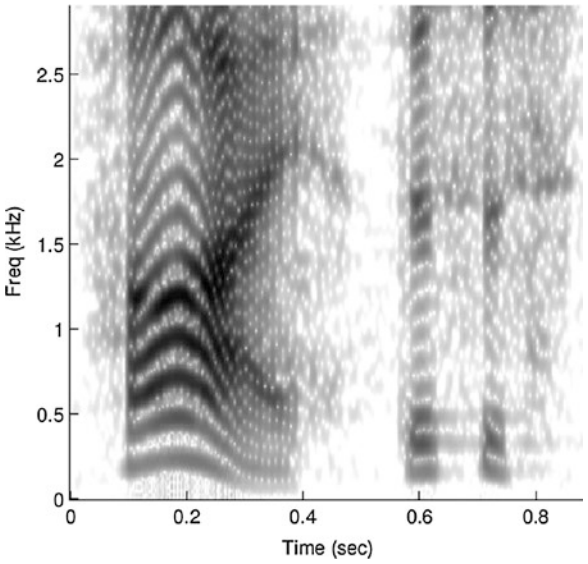

Figure 10-2. *Spectogram of speech sample*

Here is the Python code to convert an audio file to a spectrogram:

```
import matplotlib.pyplot as plt
from scipy import signal
from scipy.io import wavfile

sample_rate, samples = wavfile.read('monoAudioFile.wav')
frequencies, times, spectogram = signal.spectrogram(samples, sample_rate)

plt.imshow(spectogram)
plt.ylabel('Freq(kHz)')
plt.xlabel('Time (sec)')
plt.show()
```

Open Source Approaches

There are open source packages available for Python that perform speech-to-text and text-to-speech conversion.

The following are some open source speech-to-text conversion APIs:

- PocketSphinx
- Google Speech
- Google Cloud Speech
- Wit.ai
- Houndify
- IBM Speech to Text API
- Microsoft Bing Speech

Having used all of these, I can say that they work quite well; the American accent is especially clear.

If you are interested in evaluating the accuracy of the conversion, you need one metric: the word error rate (WER).

In the next section, I will discuss each API mentioned previously.

Examples Using Each API

Let's go through each API.

Using PocketSphinx

PocketSphinx is an open source API used for speech-to-text conversions. It is a lightweight speech recognition engine, specifically tuned for handheld and mobile devices, though it works equally well on the desktop. Simply use the command `pip install PocketSphinx` to install the package.

```
import speech_recognition as sr
from os import path
AUDIO_FILE = "MyAudioFile.wav"

r = sr.Recognizer()
with sr.AudioFile(AUDIO_FILE) as source:
 audio = r.record(source)

try:
   print("Sphinx thinks you said " + r.recognize_sphinx(audio))
except sr.UnknownValueError:
   print("Sphinx could not understand audio")
except sr.RequestError as e:
  print("Sphinx error; {0}".format(e))

================================================================
```

Using the Google Speech API

Google provides its own Speech API that can be implemented in Python code and can be used to create different applications.

```
# recognize speech using Google Speech Recognition
try:
    print("Google Speech Recognition thinks you said " +
    r.recognize_google(audio))
except sr.UnknownValueError:
    print("Google Speech Recognition could not understand audio")
except sr.RequestError as e:
    print("Could not request results from Google Speech
    Recognition service;{0}".format(e))
```

Using the Google Cloud Speech API

You can also use the Google Cloud Speech API for the conversion. Create an account on the Google Cloud and copy the credentials.

```
GOOGLE_CLOUD_SPEECH_CREDENTIALS = r"INSERT THE CONTENTS OF THE
GOOGLE CLOUD SPEECH JSON CREDENTIALS FILE HERE" try:
     print("Google Cloud Speech thinks you said " +
     r.recognize_google_cloud(audio, credentials_json=GOOGLE_
     CLOUD_SPEECH_CREDENTIALS))
except sr.UnknownValueError:
    print("Google Cloud Speech could not understand audio")
except sr.RequestError as e:
    print("Could not request results from Google Cloud Speech
    service; {0}".format(e))
```

Using the Wit.ai API

The Wit.ai API enables you to make a speech-to-text converter. You need to create an account and then create a project. Copy your Wit.ai key and start coding.

```
#recognize speech using Wit.ai
WIT_AI_KEY = "INSERT WIT.AI API KEY HERE" # Wit.ai keys are
32-character uppercase alphanumeric strings
try:
    print("Wit.ai thinks you said " + r.recognize_wit(audio,
    key=WIT_AI_KEY))
except sr.UnknownValueError:
    print("Wit.ai could not understand audio")
except sr.RequestError as e:
    print("Could not request results from Wit.ai service; {0}".
    format(e))
```

Using the Houndify API

Similar to the previous APIs, you need to create an account at Houndify and get your client ID and key. This allows you to build an app that responds to sound.

```
# recognize speech using Houndify
HOUNDIFY_CLIENT_ID = "INSERT HOUNDIFY CLIENT ID HERE"
# Houndify client IDs are Base64-encoded strings
HOUNDIFY_CLIENT_KEY = "INSERT HOUNDIFY CLIENT KEY HERE"
# Houndify client keys are Base64-encoded strings
try:
    print("Houndify thinks you said " + r.recognize_
    houndify(audio, client_id=HOUNDIFY_CLIENT_ID, client_
    key=HOUNDIFY_CLIENT_KEY))
except sr.UnknownValueError:
    print("Houndify could not understand audio")
except sr.RequestError as e:
    print("Could not request results from Houndify service;
    {0}".format(e))
```

Using the IBM Speech to Text API

The IBM Speech to Text API enables you to add IBM's speech recognition capabilities to your applications. Log in to the IBM cloud and start your project to get an IBM username and password.

```
# IBM Speech to Text
# recognize speech using IBM Speech to Text
IBM_USERNAME = "INSERT IBM SPEECH TO TEXT USERNAME HERE" # IBM
Speech to Text usernames are strings of the form XXXXXXXX-XXXX-
XXXX-XXXX-XXXXXXXXXXXX
```

```
IBM_PASSWORD = "INSERT IBM SPEECH TO TEXT PASSWORD HERE" # IBM
Speech to Text passwords are mixed-case alphanumeric strings
try:
    print("IBM Speech to Text thinks you said " + r.recognize_
    ibm(audio, username=IBM_USERNAME, password=IBM_PASSWORD))
except sr.UnknownValueError:
    print("IBM Speech to Text could not understand audio")
except sr.RequestError as e:
    print("Could not request results from IBM Speech to Text
    service; {0}".format(e))
```

Using the Bing Voice Recognition API

This API recognizes audio coming from a microphone in real time. Create an account on Bing.com and get a Bing Voice Recognition API key.

```
# recognize speech using Microsoft Bing Voice Recognition
BING_KEY = "INSERT BING API KEY HERE" # Microsoft Bing Voice
Recognition API key is 32-character lowercase hexadecimal
strings
try:
    print("Microsoft Bing Voice Recognition thinks you said " +
    r.recognize_bing(audio, key=BING_KEY))
except sr.UnknownValueError:
    print("Microsoft Bing Voice Recognition could not
    understand audio")
except sr.RequestError as e:
    print("Could not request results from Microsoft Bing Voice
    Recognition service; {0}".format(e))
```

Once you have converted the speech into text, you cannot expect 100 percent accuracy. To measure the accuracy, you can use the WER.

Text-to-Speech Conversion

This section of the chapter focuses on converting written text to an audio file.

Using pyttsx

Using a Python package called pyttsx, you can convert text to audio.

Do a pip install pyttsx. If you are using python 3.6 then do pip3 install pyttsx3.

```
import pyttsx
engine = pyttsx.init()
engine.say("Your Message")
engine.runAndWait()
```

Using SAPI

You can also use SAPI to do text-to-speech conversion in Python.

```
from win32com.client import constants, Dispatch
Msg = "Hi this is a test"
speaker = Dispatch("SAPI.SpVoice")  #Create SAPI SpVoice Object
speaker.Speak(Msg)                  #Process TTS
del speaker
```

Using SpeechLib

You can take the input from a text file and convert it to audio using SpeechLib, as shown here:

```
from comtypes.client import CreateObject
engine = CreateObject("SAPI.SpVoice")
```

```
stream = CreateObject("SAPI.SpFileStream")
from comtypes.gen import SpeechLib
infile = "SHIVA.txt"
outfile = "SHIVA-audio.wav"
stream.Open(outfile, SpeechLib.SSFMCreateForWrite)
engine.AudioOutputStream = stream
f = open(infile, 'r')
theText = f.read()
f.close()
engine.speak(theText)
stream.Close()
```

Many times, you have to edit the audio so that you can remove a voice from the audio file. The next section shows you how.

Audio Cutting Code

Make a CSV file of audio that contains the comma-separated values of the details of the audio and perform the following using Python:

```
import wave
import sys
import os
import csv
origAudio = wave.open('Howard.wav', 'r') #change path
frameRate = origAudio.getframerate()
nChannels = origAudio.getnchannels()
sampWidth = origAudio.getsampwidth()
nFrames   = origAudio.getnframes()

filename =  'result1.csv' #change path
```

```
exampleFile = open(filename)
exampleReader = csv.reader(exampleFile)
exampleData = list(exampleReader)

count = 0

for data in exampleData:
 #for selections in data:
    print('Selections ', data[4], data[5])
    count += 1
    if data[4] == 'startTime' and data[5] == 'endTime':
        print('Start time')
    else:
        start = float(data[4])
        end = float(data[5])
        origAudio.setpos(start*frameRate)
        chunkData = origAudio.readframes(int((end-
        start)*frameRate))
        outputFilePath = 'C:/Users/Navin/outputFile{0}.wav'.
        format(count) # change path
        chunkAudio = wave.open(outputFilePath, 'w')
        chunkAudio.setnchannels(nChannels)
        chunkAudio.setsampwidth(sampWidth)
        chunkAudio.setframerate(frameRate)
        chunkAudio.writeframes(chunkData)
        chunkAudio.close()
```

Cognitive Service Providers

Let's look at some cognitive service providers that help with speech processing.

Microsoft Azure

Microsoft Azure provides the following:

- *Custom Speech Service*: This overcomes speech recognition barriers such as speaking style, vocabulary, and background noise.
- *Translator Speech API*: This enables real-time speech translation.
- *Speaker Identification API*: This can identify the speakers based on a speech sample of each speaker in the given audio data.
- *Bing Speech API*: This converts audio to text, understands intent, and converts text back to speech for natural responsiveness.

Amazon Cognitive Services

Amazon Cognitive Services provides Amazon Polly, a service that turns text into speech. Amazon Polly lets you create applications that talk, enabling you to build entirely new categories of speech-enabled products.

- 47 voices and 24 languages can be used, and an Indian English option is provided.
- Tones such as whispering, anger, and so on, can be added to particular parts of the speech using Amazon effects.

- You can instruct the system how to pronounce a particular phrase or word in a different way. For example, "W3C" is pronounced as World Wide Web Consortium, but you can change that to pronounce just the acronym. You can also provide the input text in SSML format.

IBM Watson Services

There are two services from IBM Watson.

- *Speech to text*: U.S. English, Spanish, and Japanese
- *Text to speech*: U.S. English, U.K. English, Spanish, French, Italian, and German

The Future of Speech Analytics

Speech recognition technology has been making a great progress. Every year, it is about 10 to 15 percent more accurate than the previous year. In the future, it will provide the most interactive interface for computers yet.

There are many applications that you will soon be witnessing in the marketplace, including interactive books, robotic control, and self-driving car interfaces. Speech data offers some exciting new possibilities because it is the future of the industry. Speech intelligence enables people to message, take or give orders, raise complaints and to do any work where they used to type manually. It offers a great customer experience and perhaps that is why all customer-facing departments and businesses tend to use speech applications very heavily. I can see a great future for speech application developers.