Speech is the primary and most basic form of human communication and is therefore a vital part of understanding human behavior. Speech Recognition in Artificial Intelligence is a technique deployed on computer programs to enable them in understanding human language in the form of spoken words. With our learnings of Natural Language Processing (NLP) in the last few chapters, we will now transition towards understanding speech-based cognitive recognition systems. In this chapter, we will concentrate on Speech Recognition.

A general Speech Recognition system is designed to perform three tasks:

1. The capture of speech (words, sentences, phrases) given by a human. This step only concentrates on the data acquisition part of a learning workflow.
2. Application of Natural Language Processing (NLP) on the acquired data to understand the content of speech.
3. Synthesis of the recognized words to help the machine speak a similar dialect.

This section concentrates on speech recognition and can be understood as an extension to the Natural Language Processing and Machine Learning pipeline we built in the previous chapter. Please remember that the speech signals that get processed by a computer program are captured with the help of a microphone and fed to the system.

**Processing Sound in Computer Systems**

Like images and videos, sound is also an analog signal that humans perceive through sensory organs. For machines, to consume this information, it needs to be stored as digital signals and analyzed through software. The conversion from analog to digital consists of the below two processes:

1. Sampling: It is a procedure used to convert a time-varying (changing with time) signal s(t) to a discrete progression of real numbers x(n). Sampling period (Ts) is a term that defines the interval between two successive discrete samples. Sampling Frequency (fs = 1/Ts) is the inverse of the sampling period. Common sampling frequencies are 8 KHz, 16 KHz and 44.1 KHz. A 1 Hz sampling rate means one sample per second and therefore high sampling rates mean better signal quality.
2. Quantization: This is the process of replacing every real number generated by sampling with an approximation to obtain a finite precision (defined within a range of bits). In majority of scenarios, 16 bits per sample are used for the representation of a single quantized sample. Therefore, raw audio samples generally have a signal range of -215 to 215 although, during analysis these values are standardized to the range (-1, 1) for simpler validation and model training. Sample resolution is always measured in bits per sample.

Constructing a Speech Recognition Mechanism

Speech Recognition, in terms of AI seldom called Automatic Speech Recognition (ASR) is the core of robotic AI processes that work by taking speech input from humans. Without ASR, robot human interaction is not possible. There are multiple challenges that are faced while building a speech recognizer, namely:

1. **Vocabulary Size:** The number of words or phrases that an algorithm has to process, greatly affects the complications of building a speech recognition system. A small sized vocabulary, for instance in a menu-based system contains about 200 words. A medium-sized vocabulary system contains about a thousand words and anything in the range of 10,000 words and above is considered a large vocabulary system.
2. **Characteristics of the Channel:** The medium through which speech signals are transmitted is also an important factor is building a model. For instance, a direct human voice input is recorded in full frequency and has a high bandwidth. On the other hand, the telephonic conversation has low bandwidth and a limited frequency range, making analysis difficult.
3. **Speech Style:** No two conversations between humans can sound the same. The reason being everyone’s tone is different and dialects change with regions. Therefore, in a single language like English, the style of speech differs greatly between people. Also, a formal styled speech is easier to analyze in comparison to casual speech.
4. **Presence of Noise:** External noise is present in almost all environments and it plays a significant role in deteriorating the quality of sound that reaches the analysis algorithm. Higher levels of noise (above 30 dB) are considered difficult to work with, while sounds in the range (30 dB to 10 dB) are moderate disturbances and anything below 10 dB does not affect analysis severely.
5. **Quality of Microphone:** The only way to take input for speech recognition algorithms is using a microphone. Therefore, the quality of this hardware is largely important in determining the difficulties to be faced during speech recognition.

Despite the existence of these difficulties, there are preprocessing methodologies that can be deployed within the analysis code to make it work smoothly. We will, in the coming sections of this chapter, learn more about building a speech recognition model and solving problems with it.

Diagram

Description automatically generated  
Processing of Speech (Audio) Signals

Let us now start the Signal Processing of Sound, with an input audio file. There are multiple sources from where test files can be downloaded. In addition, we will also go through a method that helps us record directly into the microphone and process that audio with Python.

Step 1: Reading a File for Audio Signals

File I/O in Python (scipy.io): SciPy has numerous methods of performing file operations in Python. The I/O method that includes methods *read(filename[, mmap])* and *write(filename, rate, data)* is used to read from a .wav file and write a NumPy array in the form of a .wav file. We will be using these methods to read from and write to sound (audio) file formats.

The first step in starting a speech recognition algorithm is to create a system that can read files that contain audio (.wav, .mp3, etc.) and understanding the information present in these files. Python has libraries that we can use to read from these files and interpret them for analysis. The purpose of this step is to visualize audio signals as structured data points.

* **Recording:** A recording is the file we give to the algorithm as its input. The algorithm then works on this input to analyze its contents and build a speech recognition model. This could be a saved file, or a live recording, Python allows for both.
* **Sampling:** All signals of a recording are stored in a digitized manner. These digital signatures are hard of a software to work upon since machines only understand numeric input. ***Sampling*** is the technique used to convert these digital signals into a discrete numeric form. Sampling is done at a certain frequency and it generates numeric signals. Choosing the frequency levels, depends on the human perception of sound. For instance, choosing a high frequency implies that the human perception of that audio signal is continuous.

|  |  |
| --- | --- |
| *# Source of the Audio File:* [*Sample Audio File Download*](https://www.opdsupport.com/downloads/miscellaneous/sample-audio-files)  *# Import the packages needed for this analysis*  import numpy as np  import matplotlib.pyplot as plt  from scipy.io import wavfile  *# We will now read the audio file and determine the audio signal and sampling frequency*  *# Give the path of the file*  freq\_sample, sig\_audio = wavfile.read("Welcome.wav")  *# Output the parameters: Signal Data Type, Sampling Frequency and Duration*  print('\nShape of Signal:', sig\_audio.shape)  print('Signal Datatype:', sig\_audio.dtype)  print('Signal duration:', round(sig\_audio.shape[0] / float(freq\_sample), 2), 'seconds')  >>> Shape of Signal: (645632,)  >>> Signal Datatype: int16  >>> Signal duration: 40.35 seconds  *# Normalize the signal values*  pow\_audio\_signal = sig\_audio / np.power(2, 15)  *# We shall now extract the first 100 values from the signal*  pow\_audio\_signal = pow\_audio\_signal [:100]  time\_axis = 1000 \* np.arange(0, len(pow\_audio\_signal), 1) / float(freq\_sample)  *# Visualize the signal*  plt.plot(time\_axis, pow\_audio\_signal, color='blue')  plt.xlabel('Time (milliseconds)')  plt.ylabel('Amplitude')  plt.title('Input audio signal')  plt.show()  >>>   |  | | --- | |  |   *# This is the representation of the sound amplitude of the input file against its duration of play. We have successfully extracted numerical data from audio (.wav) file.* |

Once reading from an audio file is complete, the next step of analysis is to learn to transform the audio signals observed in the file. Transformation of frequency is important to learn more about the audio present in the file. In the next section we will learn more about transforming frequencies of an audio input.

Step 2: Transforming Audio Frequencies

**Time-Based Representation:** The representation of the audio signal we did in the first section represents a time-domain audio signal. It shows the intensity (loudness or amplitude) of the sound wave with respect to time. Portions with amplitude = 0, represent silence. In terms of sound engineering, amplitude = 0 is the sound of static or moving air particles when no other sound is present in the environment. This analysis is not very fruitful since it only focuses on the loudness of the audio.

**Frequency-Domain Representation:** To better understand an audio signal, it is necessary to look at it through a frequency domain. This representation of an audio signal will give us details about the presence of different frequencies in the signal. Fourier Transform is a mathematical concept that can be used in the conversion of a continuous signal from its original time-domain state to a frequency-domain state. We will be using Fourier Transforms (FT) in Python to convert audio signals to a frequency-centric representation.

**Fourier Transforms in Python:** All audio signals are composed of a collection of many single-frequency sound waves that travel together and create a disturbance in the medium of movement, for instance, a room. Capturing sound is essentially the capturing of the amplitudes that these waves generate in space. Fourier Transforms is a mathematical concept that can decompose this signal and bring out the individual frequencies. This is vital for understanding all the frequencies that combined together to form the sound we hear. Fourier Transform (FT) gives all the frequencies present in the signal and also shows the magnitude of each frequency. In the code section below, we will take the same file we analyzed earlier and transform it to its frequency-domain. We will also represent individual frequencies along with its magnitude.

Diagram, engineering drawing

Description automatically generated  
Representation of Fourier Transform (Converting Time-Domain to Frequency-Domain)

* **NumPy (np.fft.fft):** This NumPy function allows us to compute a 1-D discrete Fourier Transform. The function uses Fast Fourier Transform (FFT) algorithm to convert a given sequence to a Discrete Fourier Transform (DFT). In the file we are processing, we have a sequence of amplitudes drawn from an audio file, that were originally sampled from a continuous signal. We will use this function to covert this time-domain to a discrete frequency-domain signal.

|  |  |
| --- | --- |
| *# Characterization of the signal from the input file*  *# We will be using Fourier Transforms to convert the signals to a frequency domain distribution*  import numpy as np  import matplotlib.pyplot as plt  from scipy.io import wavfile  freq\_sample, sig\_audio = wavfile.read("Welcome.wav")  print('\nShape of the Signal:', sig\_audio.shape)  print('Signal Datatype:', sig\_audio.dtype)  print('Signal duration:', round(sig\_audio.shape[0] / float(freq\_sample), 2), 'seconds')  >>> Shape of the Signal: (645632,)  >>> Signal Datatype: int16  >>> Signal duration: 40.35 seconds  sig\_audio = sig\_audio / np.power(2, 15)  *# Extracting the length and the half-length of the signal to input to the foruier transform*  sig\_length = len(sig\_audio)  half\_length = np.ceil((sig\_length + 1) / 2.0).astype(np.int)  *# We will now be using the Fourier Transform to form the frequency domain of the signal*  signal\_freq = np.fft.fft(sig\_audio)  *# Normalize the frequency domain and square it*  signal\_freq = abs(signal\_freq[0:half\_length]) / sig\_length  signal\_freq \*\*= 2  transform\_len = len(signal\_freq)  *# The Fourier transformed signal now needs to be adjusted for both even and odd cases*  if sig\_length % 2:  signal\_freq[1:transform\_len] \*= 2  else:  signal\_freq[1:transform\_len-1] \*= 2  *# Extract the signal's strength in decibels (dB)*  exp\_signal = 10 \* np.log10(signal\_freq)  x\_axis = np.arange(0, half\_length, 1) \* (freq\_sample / sig\_length) / 1000.0  plt.figure()  plt.plot(x\_axis, exp\_signal, color='green', linewidth=1)  plt.xlabel('Frequency Representation (kHz)')  plt.ylabel('Power of Signal (dB)')  plt.show()  >>>   |  | | --- | |  | |

We now see that audio signals can also be split into a frequency distribution that displays the power (amplitude) of each frequency encountered in the audio file. This is important in order to see the distribution of frequencies within the given sound input. In the coming section we will see how monotone signals and their individual amplitudes help in building signals, and how distribution of frequencies can help create filters for speech data. These filters help in creating distinct boundaries between frequency distributions.

Monotone Audio Signals and Their Importance

Before moving to the remainder of our signal preprocessing, we should understand the difference between stereo signals and monotone signals. All sound generated in any environment is always stereo sound. A mono audio signal is one that produces through one channel and is the easiest to analyze.

Sound, in terms of Physics, is a travelling vibration. In simple words, waves moving in an isolated medium (such as air) are the origin of sound. Sound waves emit and trasfer energy from particle to particle in air until it reaches a destination (for example, our ears). Two basic attributes that define sound are ***amplitude*** that focuses on the intensity/loudness of the sound wave, and ***frequency*** that measures a wave’s vibrations in given unit time.

Therefore, theortically, sound cannot be created without movement of waves. Additionally, since sound needs a medium to travel, its parameters like (strength and frequency) will always depend on external factors like noise and cannot be constructed. But, during speech recognition and analysis of sounds, it might be required that we have some signals of a predefined structure. Therefore, Python allows us to create audio signals and write them to an audio file (like .wav). In the section below, we will be practicing the idea of creating an audio signal with some predefined parameters.

|  |  |
| --- | --- |
| import numpy as np  import matplotlib.pyplot as plt  from scipy.io.wavfile import write  *# Specify the output file where this data needs to be stored*  output\_file = 'generated\_signal\_audio.wav'  *# Duration in seconds, Sampling Frequency in Hz*  sig\_duration = 8  sig\_frequency\_sampling = 74100  sig\_frequency\_tone = 802  sig\_min\_val = -5 \* np.pi  sig\_max\_val = 5 \* np.pi  *# Generating the audio signal*  temp\_signal = np.linspace(sig\_min\_val, sig\_max\_val, sig\_duration \* sig\_frequency\_sampling)  temp\_audio\_signal = np.sin(2 \* np.pi \* sig\_frequency\_tone \* temp\_signal)  *# The write() function creates a frequency-based sound signal and writes it to the created file*  write(output\_file, sig\_frequency\_sampling, temp\_audio\_signal)  sig\_audio = temp\_audio\_signal[:100]  def\_time\_axis = 1000 \* np.arange(0, len(sig\_audio), 1) / float(sig\_frequency\_sampling)  plt.plot(def\_time\_axis, sig\_audio, color='green')  plt.xlabel('Time (milliseconds)')  plt.ylabel('Sound Amplitude')  plt.title('Audio Signal Generation')  plt.show()  >>>   |  | | --- | |  | |

Generation of audio signals is an important task that helps analyses processes in tasks where test signals are required. The primary feature of importance here is that all parameters of the sound signal creation can be controlled and input as desired by the user. Let us now move forward with our analysis of speech.

Step 3: Extracting Features from Speech

Once speech is moved from a time-domain signal to a frequency domain signal, the next step is to convert this frequency domain data into a usable feature vector. We will read about MFCC for this task.

**Mel Frequency Cepstral Coefficients (MFCCs)**

MFCC is a technique designed to extract features from an audio signal. It uses the MEL scale to divide the audio signal’s frequency bands and then extracts coefficients from each individual frequency band, thus, creating a separation between frequencies. MFCC uses the Discrete Cosine Transform (DCT) to perform this operation. The MEL scale is established on the human perception of sound, i.e., how the human brain process audio signals and differentiates between the varied frequencies. Let us look at the formation of MEL scale below.

* **Human voice sound perception:** An adult human, has a fundamental hearing capacity that ranges from 85 Hz to 255 Hz, and this can further be distinguished between genders (85Hz to 180 Hz for Male and 165 Hz to 255 Hz for females). Above these fundamental frequencies there also are harmonics that the human ear processes. Harmonics are multiplications of the fundamental frequency. These are simple multipliers, for instance, a 100 Hz frequency’s second harmonic will be 200 Hz, third would be 300 Hz and so on.

The rough hearing range for humans is 20Hz to 20KHz and this sound perception is also non-linear. We can distinguish low frequency sounds better in comparison to high frequency sounds. For example, we can clearly state the difference between signals of 100Hz and 200Hz but cannot distinguish between 15000 Hz and 15100 Hz. To generate tones of varied frequencies we can use the program above or use this [tool.](http://www.szynalski.com/tone-generator/)

* **MEL Scale:** Stevens, Volkmann and Newmann proposed a pitch in 1937 that introduced the MEL scale to the world. It is a pitch scale (scale of audio signals with varied pitch levels) that is judged by humans on the basis of equality in their distances. It is basically a scale that is derived from human perception. For example, if you were exposed to two sound sources distant from each other, the brain will perceive a distance between these sources without actually seeing them. This scale is based on how we humans measure audio signal distances with the sense of hearing. Because our perception is non-linear, the distances on this scale increases with frequency.
* **MEL-spaced Filterbank:** To compute the power (strength) of every frequency band, the first step is to distinguish the different feature bands available (done by MFCC). Once these segregations are made, we use filter banks to create partitions in the frequencies and separate them. Filter banks can be created using any specified frequency for partitions. The spacing between filters within a filter bank grows exponentially as the frequency grows. In the code section, we will see how to separate frequency bands.

**Mathematics of MFCCs and Filter Banks**

MFCC and creation of filter banks is all motivated by the nature of audio signals and impacted by the way in which humans perceive sound. But this processing also requires a lot of mathematical computation that goes behind the scenes in its implementation. Python directly gives us methods to build filters and perform the MFCC functionality on sound but let us glance on the math behind these functions. Three discrete mathematical models that go into this processing are the Discrete Cosine Transform (DCT), which is used for decorrelation of filter bank coefficients, also termed as whitening of sound, and Gaussian Mixture Models - Hidden Markov Models (GMMs-HMMs) that are a standard for Automatic Speech Recognition (ASR) algorithms. Although, in the present day, when computation costs have gone down (thanks to Cloud Computing), deep learning speech systems that are less susceptible to noise, are used over these techniques. Also, a point to note is that DCT is a linear transformation algorithm, and it will therefore rule out a lot of useful signals, given sound is highly non-linear.

|  |  |  |
| --- | --- | --- |
| pip install python\_speech\_features  *# Import the necessary pacakges*  import numpy as np  import matplotlib.pyplot as plt  from scipy.io import wavfile  from python\_speech\_features import mfcc, logfbank  sampling\_freq, sig\_audio = wavfile.read("Welcome.wav")  *# We will now be taking the first 15000 samples from the signal for analysis*  sig\_audio = sig\_audio[:15000]  *# Using MFCC to extract features from the signal*  mfcc\_feat = mfcc(sig\_audio, sampling\_freq)  print('\nMFCC Parameters\nWindow Count =', mfcc\_feat.shape[0])  print('Individual Feature Length =', mfcc\_feat.shape[1])  >>> MFCC Parameters  >>> Window Count = 13  >>> Individual Feature Length = 93  mfcc\_feat = mfcc\_feat.T  plt.matshow(mfcc\_feat)  plt.title('MFCC Features')  >>>   |  | | --- | |  |   *# The first horizontal yellow lines below every segment are the fundamental frequency and at its strongest. Above the yellow line are the harmonics that share the same frequency distance between them*  *# Generating filter bank features*  fb\_feat = logfbank(sig\_audio, sampling\_freq)  print('\nFilter bank\nWindow Count =', fb\_feat.shape[0])  print('Individual Feature Length =', fb\_feat.shape[1])  >>> Filter bank  >>> Window Count = 93  >>> Individual Feature Length = 26  fb\_feat = fb\_feat.T  plt.matshow(fb\_feat)  plt.title('Features from Filter bank')  plt.show()  >>>   |  | | --- | |  | |

If we see the two distributions, it is evident that the low frequency and high frequency sound distributions are separated in the second image. The MFCC, along with application of Filter Banks is a good algorithm to separate the high and low frequency signals. This expedites the analysis process as we can trim sound signals into two or more separate segments and individually analyze them based on their frequencies.

With this, we now complete our analysis of sound waves. Working with speech is a specialized cognitive science application and there are numerous ways and algorithms that can be used. What we have covered in this chapter is the basics of sound engineering and understanding of audio signals. We have also seen how to transform audio files to numeric form that helps build data models. In the coming section of this chapter, we will use of the publicly available API(s) to understand speech and transcribe it to text. This is the final step of the speech recognition pipeline.

Step 5: Recognizing Spoken Words

Speech Recognition is the process of understanding human voice and transcribing it to text in the machine. There are several libraries available to process speech to text, namely, Bing Speech, Google Speech, Houndify, IBM Speech to Text, etc. We will be using the Google Speech library to convert Speech to Text.

**Google Speech API**

More about the Google Speech API can be read from the [Google Cloud Page](https://cloud.google.com/speech-to-text) and the [Speech Recognition](https://pypi.org/project/SpeechRecognition/) PyPi page. A few key features that the Google Speech API is capable of are adaptation of speech. This means that the API understands the domain of the speech. For instance, currencies, addresses, years are all prescribed into the speech to text conversion. There are domain-specific classes defined in the algorithm that recognize these occurrences in the input speech. The API works with both on-prem, pre-recorded files as well as live recordings on the microphone in the present working environment. We will analyze live speech through microphonic input in the next section.

**Working with Microphones:** The PyAudio open-source package allows us to directly record audio through an attached microphone and analyze it with Python in real-time. The installation of PyAudio will vary based on the operating system (the installation explanation is mentioned in the code section below).

* **Microphone Class:** The instance of .microphone() class can be used with Speech Recognizer to directly record audio within the working directory. To check if microphones are available in the system, use the list\_microphone\_names static method. To use any of the available listed microphones use the device\_index method (Implementation shown in the code below)
* **Capturing Microphone Input:** The listen() function is used to capture input given to the microphone. All the sound signals that the selected microphone receives, is stored in the variable that calls the listen() function. This method continues recording until a silent ( 0 amplitude) signal is detected.
* **Ambient Noise Reduction:** Any functional environment is prone to have ambient noise that will hamper the recording. Therefore, the adjust\_for\_ambient\_noise() method within the Recognizer class helps automatically cancel out ambient noise from the recording.

**Recognition of Sound:** The speech recognition workflow below explains the part after processing of signals where the API performs tasks like Semantic and Syntactic corrections, understands the domain of sound, the spoken language and finally creates the output by converting speech to text. Below we will also see the implementation of Google’s Speech Recognition API using the Microphone class.

Diagram

Description automatically generated

|  |  |
| --- | --- |
| *# Install the SpeechRecognition and pipwin classes to work with the Recognizer() class*  pip install SpeechRecognition  pip install pipwin  *# Below are a few links that can give details about the PyAudio class we will be using to read direct microphone input into the Jupyter Notebook*  *# https://anaconda.org/anaconda/pyaudio*  *# https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyaudio*  *# To install PyAudio, Run in the Anaconda Terminal CMD: conda install -c anaconda pyaudio*  *# Pre-requisite for running PyAudio installation - Microsoft Visual C++ 14.0 or greater will be required. Get it with "Microsoft C++ Build Tools" : https://visualstudio.microsoft.com/visual-cpp-build-tools/*  pip install pyaudio  import speech\_recognition as speech\_recog  *# Creating a recording object to store input*  rec = speech\_recog.Recognizer()  *# Importing the microphone class to check availabiity of microphones*  mic\_test = speech\_recog.Microphone()  *# List the available microphones*  speech\_recog.Microphone.list\_microphone\_names()  >>>   |  | | --- | | ['Microsoft Sound Mapper - Input',  'Microphone (Realtek(R) Audio)',  'Microsoft Sound Mapper - Output',  'Speakers (Realtek(R) Audio)',  'Stereo Mix (Realtek HD Audio Stereo input)',  'Headphones (Realtek HD Audio 2nd output)',  'Speakers 1 (Realtek HD Audio output with HAP)',  'Speakers 2 (Realtek HD Audio output with HAP)',  'PC Speaker (Realtek HD Audio output with HAP)',  'Mic in at front panel (black) (Mic in at front panel (black))',  'Microphone Array (Realtek HD Audio Mic input)'] |   *# We will now directly use the microphone module to capture voice input*  *# Specifying the second microphone to be used for a duration of 3 seconds*  *# The algorithm will also adjust given input and clear it of any ambient noise*  with speech\_recog.Microphone(device\_index=1) as source:  rec.adjust\_for\_ambient\_noise(source, duration=3)  print("Reach the Microphone and say something!")  audio = rec.listen(source)  >>> Reach the Microphone and say something!  *# Use the recognize function to transcribe spoken words to text*  try:  print("I think you said: \n" + rec.recognize\_google(audio))  except Exception as e:  print(e)  >>> I think you said:  >>> hello the weather is cold  *# With this, we conclude our speech recognition project* |

Coming Up Next

Speech recognition is an AI concept that allows a machine to listen to human voice and transcribe text out of it. Although complex in nature, the use cases revolving around Speech Recognition are plenty. From helping differently abled users with access to computing to an automated response machine, Automatic Speech Recognition (ASR) algorithms are being used by many industries today. This chapter gave a brief introduction to the engineering of sound analysis and showed some basic manipulation techniques to work with audio. Though not detailed, it will help in creating an overall picture of how speech analysis works in the world of AI. In the coming chapters we will focus more on Cognitive Sciences in AI and solve some problems in this domain.

1. Cognitive Science and Artificial Intelligence
2. Computer Vision in Python
3. The AI Market