An Introduction to Speech Processing

Dr. Rajesh Kumar Mundotiya

August 20, 2025

Outline

Where Do We See Speech Recognition?

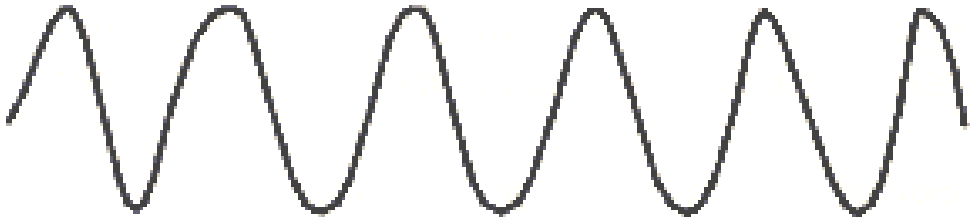
- It's the technology that powers the voice assistants we use every day.

Google Assistant

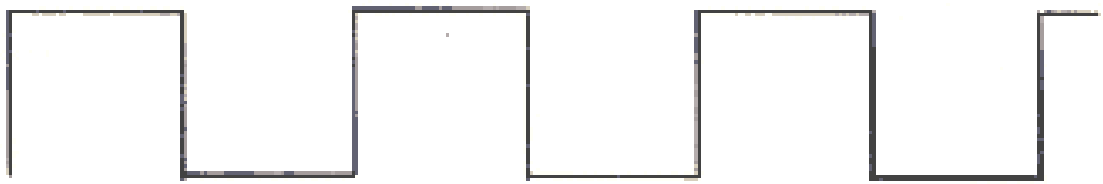
- **Question:**.Besides your phone, where else have you seen this?(e.g., car navigation, smart home devices, dictation software).

How a Computer "Hears"

Sound in the real world is **analog** (a continuous wave). Computers only understand **digital** information (discrete numbers).



Analog Signal



Digital Signal

Key Properties of Digital Audio

Sampling Rate (Hz) How many times per second we measure the wave.

- Like "frames-per-second" for audio.
- Common for speech: **16,000 Hz (16 kHz)**.

Bit Depth How much detail we use for each sample (amplitude/loudness resolution).

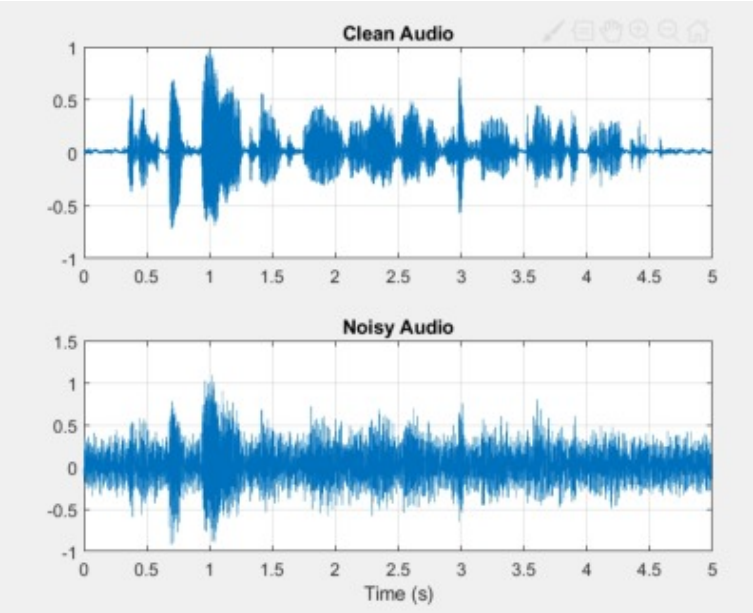
Channels The number of audio streams.

- **Mono** (1 channel) is standard for speech processing.
- **Stereo** (2 channels).

The Goal

"These two properties determine the quality and size of our audio file. For speech recognition, we need enough detail to capture human speech, but not so much that it becomes slow to process."

The Challenge: Raw Audio is Messy



- Problem 1: Background Noise. A clean recording vs. one with humming or chatter.
- Different Volumes. The same word spoken softly vs. loudly.
- Different Volumes. The same word spoken softly vs. loudly.

If we feed this raw, inconsistent audio directly into a model, it will get confused. The model might think a loud word is different from a soft word, or mistake noise for speech. This is why we must first clean it up.

What is Speech Preprocessing?

- It is the "Clean-Up" Phase
- **Definition**. A series of steps to clean, standardize, and transform raw audio into a consistent format for our model.

What is Speech Preprocessing? (Part 2)

1. **Our Goals:**
2. Standardize the sampling rate.
3. Fix volume differences (Normalization).
4. Break audio into small, analyzable chunks (Framing).

The Goal

Think of this as preparing your ingredients before cooking. We need to make sure everything is clean, consistent, and ready for the main recipe—the recognition model.

Making Audio Consistent:Standardization in Action

- **Normalization :** Scaling the audio amplitude to a fixed range (e.g., -1 to 1). This ensures loudness doesn't affect performance.
- **Framing**. Dividing the audio into small, overlapping chunks (e.g., 25ms frames). This is because speech changes constantly, and we need to analyze these changes over time.

The Goal

Normalization makes sure we're analyzing what was said, not how loudly it was said. Framing lets us look at the speech signal moment by moment, which is essential for understanding words.

Key Properties of Digital Audio

We'll follow a standard set of steps to transform our audio.

1. Load Audio
2. Resample
3. Normalize
4. Frame & Window
5. Extract Features

Our Python Toolkit for Speech

We will use a few key Python libraries:

1. **librosa :**The industry standard for audio analysis.
2. **matplotlib :**For visualizing our data.
3. **numpy :** For numerical operations.

Installations

```
pip install librosa matplotlib numpy
```

The Goal

Now, let us move from theory to practice. These libraries will do most of the heavy lifting for us. Make sure you have them installed to follow along.

Get Your Noisy Audio File for the Demo

How to Create a Test File (Recommended)

1. **Find a Noise Source:** Play some background sound (like a YouTube "cafe ambience" video) on another device.
2. **Record Your Audio:** Using a voice memo app, record the following in one take:
   - First, 3 seconds of ONLY the background noise.
   - Then, without stopping, speak a short sentence.
3. **Export and Save:** Share or save the file as a `.wav` and name it `my_noisy_audio.wav` in your project folder.

Why the 3 seconds of pure noise?

The noise reduction algorithm needs a clean "fingerprint" of the background noise to know what sound it needs to subtract from your voice!

Loading and Visualizing Audio

We use a library like `librosa` to load an audio file and resample it to a consistent rate (e.g., 16 kHz).

```
import librosa
import matplotlib.pyplot as plt
# Load and resample audio
y, sr = librosa.load(
    'your_audio_file.wav',
    sr=16000
)
# Plot the waveform
plt.figure()
librosa.display.waveshow(y, sr=sr)
plt.title('Audio Waveform')
plt.show()

```

The Goal

Using ‘librosa.load‘ with ‘sr=16000‘ is key. It standardizes our audio on import. The plot shows the sound’s amplitude over time.

Step 4: Framing & Windowing

Speech changes quickly. We can't analyze the whole file at once.

- We slice the audio into short, overlapping chunks called **frames**.
- **Frame Size:** Length of a chunk (e.g., 25 ms).
- **Hop Length:** How far we slide the window for the next frame (e.g., 10 ms).

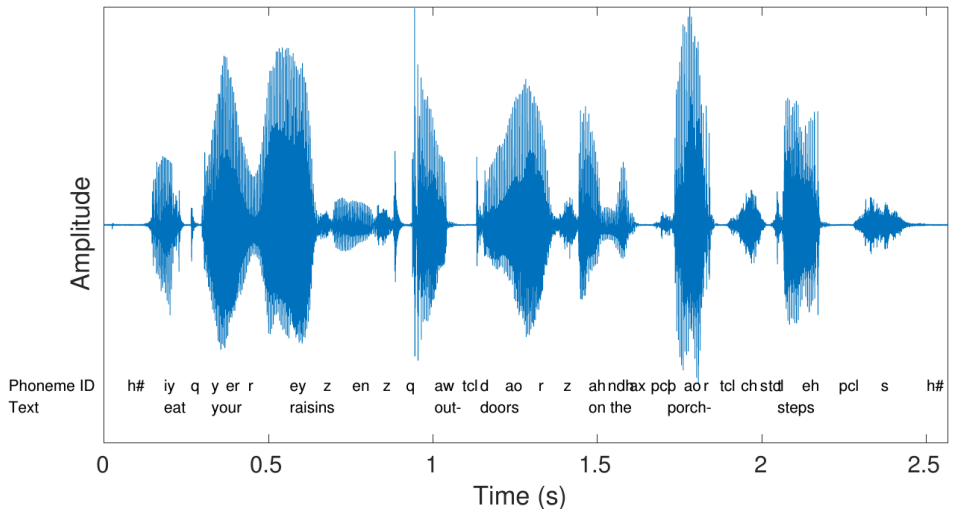


Figure: Overlapping frames extracted from a waveform.

Noise Cleaning (The "Cocktail Party" Slide)

How Do Machines Listen at a Noisy Party? Imagine you're at a loud party. Music is playing, people are laughing, dishes are clanking. Yet, somehow, you can tune all of that out and focus on what your friend is saying right in front of you. Your brain is an incredible noise canceling machine.
Our goal is to teach a computer to do the exact same thing. A raw audio recording is like hearing everything in that room at once, the speech we want and all the noise we don't."

How Do Machines Listen at a Noisy Party?

Speech changes quickly. We can't analyze the whole file at once.

- **The Problem:** Noise drowns out the important speech signal, confusing the computer.
- **The Solution:**Modern systems use AI that has learned what human speech sounds like versus what traffic, music, or a barking dog sounds like. It is like having a bouncer at the door of your ear who only lets speech sounds in and keeps the noise out.
- **The Payoff:** A clean, crisp audio signal where the speech is the star of the show. This makes the next step, actually understanding the words, much easier.

Let's Code: A Simple Noise Reduction

```
import noisereduce as nr
from scipy.io import wavfile

# 1. Load the noisy audio file and its sample rate
rate, data = wavfile.read("my_noisy_audio.wav")
# 2. Select a sample of pure noise
#    (e.g., the first 1 second). This part is crucial!
noise_clip = data[0:rate]
# 3. Perform noise reduction
reduced_noise_data = nr.reduce_noise(y=data,  sr=rate, y_noise
# 4. Save the clean audio to a new file
wavfile.write("my_clean_audio.wav", rate,
              reduced_noise_data)

print("Noise reduction complete!")
```

Feature A: The Spectrogram

- A spectrogram visualizes which frequencies are present over time.
- It's created by applying a Fourier Transform to each frame.
- **X-axis:** Time
- **Y-axis:** Frequency
- **Color:** Power/Amplitude

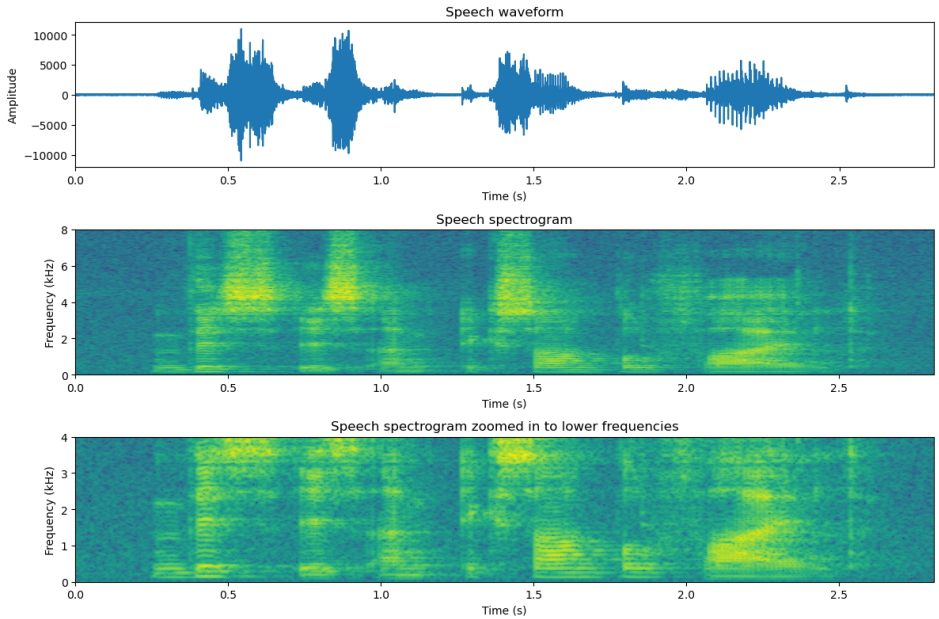


Figure: A spectrogram.

Feature B: Mel Spectrogram

- A spectrogram where the frequency axis is warped to the **Mel scale**.
- The Mel scale mimics human hearing, emphasizing frequencies most important for speech.

Feature Extraction: What the Model "Sees"

Mel-Frequency Cepstral Coefficients (MFCCs)

- The classic, powerful feature for speech recognition.
- A highly compact representation of the audio spectrum.
- Designed to mimic the human ear's perception of sound.

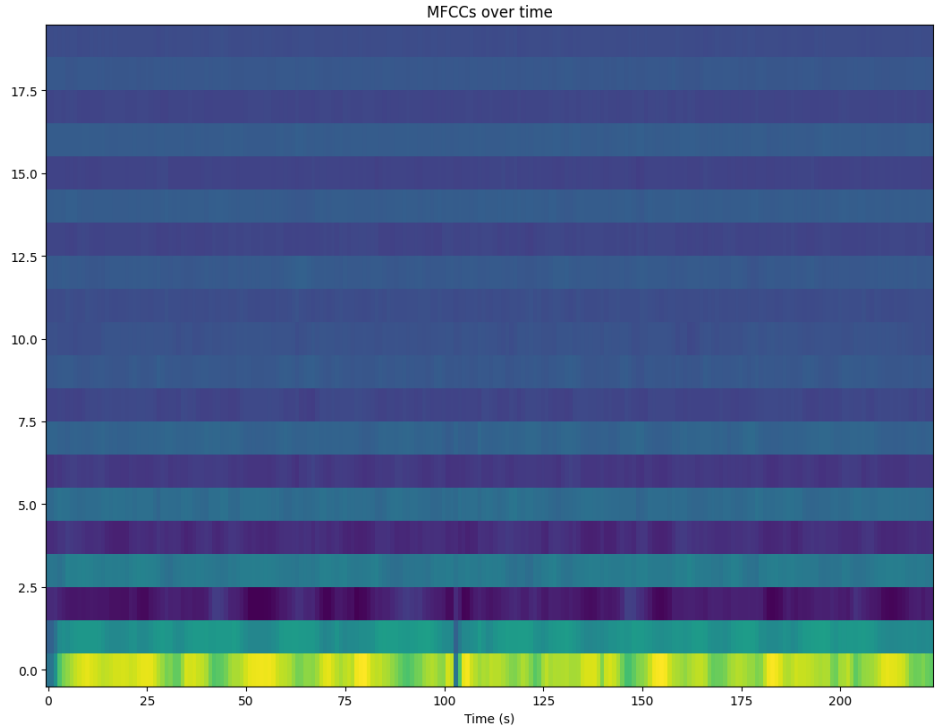


Figure: A visualization of MFCC features.

Putting It All Together: How Recognition Works

- Audio $\rightarrow$ Features (MFCCs) $\rightarrow$ Acoustic Model $\rightarrow$ Language Model $\rightarrow$ Text
- **Acoustic Model:** Matches features to sounds (phonemes).
- **Language Model:** Arranges sounds into probable words and sentences.

"The acoustic model is like the ear, figuring out the individual sounds. The language model is like the brain, using context to decide if you said 'recognize speech' or 'wreck a nice beach'.

Grading the Robot (Part 1: The Errors)

Think of it like grading a dictation test. You have the perfect sentence—the **Reference**—and what the computer wrote—the **Hypothesis**.

Example: Spotting the Errors

Reference: "turn the light on"

- **Substitution (S):** "turn a light on" (1 error)
- **Deletion (D):** "turn light on" (1 error)
- **Insertion (I):** "turn on the light on" (1 error)

Grading with WER (Part 2: The Formula)

The Word Error Rate (WER) Formula

$$\text{WER} = \frac{S + D + I}{N}$$

where **N** is the total number of words in the **Reference**.

Using Our Example

Reference: "turn the light on" $(N = 4)$
Errors: $S = 1, D = 1, I = 1$

$$\text{WER} = \frac{1 + 1 + 1}{4} = \frac{3}{4} = 75\%$$

Examples

The Bottom Line: Just like a golf score, a lower WER is always better!

Recap of Our Journey.

- We learned that raw audio is inconsistent and needs to be cleaned.
- itemize**Preprocessing** (resampling, normalization) standardizes our audio.
- **Feature Extraction (MFCCs)**creates a meaningful representation for models
- We used librosa for manual processing and the SpeechRecognition library for a quick and easy transcription.

Today, you've seen the entire fundamental pipeline of speech recognition, from a physical sound wave all the way to transcribed text."

Tasks

- **Practice:** Try transcribing your own voice using microphone input..
- **Preprocessing** (resampling, normalization) standardizes our audio.
- **Explore:** Look into modern, powerful models like OpenAI's Whisper or Wav2Vec2, accessible via libraries like Hugging Face transformers.
- Build: The ultimate next step is to train your own models using frameworks like TensorFlow or PyTorch.

Today, you've seen the entire fundamental pipeline of speech recognition, from a physical sound wave all the way to transcribed text."